NAS2-11530

# OVERCOMING COMPUTATIONAL LIMITS BY LARGE-SCALE COOPERATION

*Peter J. Denning*
*George B. Adams III*
*Robert L. Brown*
*Barry M. Leiner*
*Michael R. Raugh*

October 30, 1985

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS TR 85.15



RIACS

Research Institute for Advanced Computer Science

# OVERCOMING COMPUTATIONAL LIMITS BY LARGE-SCALE COOPERATION

*Peter J. Denning*
*George B. Adams III*
*Robert L. Brown*
*Barry M. Leiner*
*Michael R. Raugh*

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS TR 85.15
October 30, 1985

This report describes an acceleration of the RIACS core research program that could be completed by 1989 with additional funding.

We seek to increase vastly the ability of scientists and engineers to make effective use of computers. We propose four interrelated projects founded on the premises that any scheme to achieve vast increases in power must be based on cooperation among many entities (humans and computers) working in parallel and that it is necessary to fold as much knowledge of a discipline as possible into the computers supporting each scientist's working environment. Our projects are entitled: *(1) Decomposition of problems into parallel structures, (2) Programmability of specific parallel machines, (3) Distributed program composition system (DPCS),* and *(4) Distributed system development environment (DSDE).* Examples drawn from the aerospace regime at NASA Ames Research Center, where RIACS is located, will sharpen the knowledge, methodologies, and tools developed in these projects and will provide a demonstration of our results.

We act on the principle that computer scientists and other-discipline scientists must work together as teams to build the envisaged computer systems. Our research program addresses (a) ultra high speed computing machines and their algorithms, (b) exploring radical changes in the processes by which communities develop computing to support their research, and (c) supporting large-scale cooperation among dispersed groups of scientists.

---

Work reported herein was supported in part by Contract NAS2-11530 from the National Aeronautics and Space Administration (NASA) to the Universities Space Research Association (USRA).

# OVERCOMING COMPUTATIONAL LIMITS BY LARGE-SCALE COOPERATION

Peter J. Denning
George B. Adams III
Robert L. Brown
Barry M. Leiner
Michael R. Raugh

Research Institute for Advanced Computer Science

October 30, 1985

**Goal:** *Vast increases in the ability of scientists and engineers, singly and in groups, to make effective use of computers in their work.*

## 1. Introduction

Our research focuses on the effective usability of computers. It is based on two premises about the nature of projects and systems that meet the goal.

**Premise 1.** *Any scheme to achieve vast increases in power must be founded on cooperation among many entities working in parallel.*

Software that controls a single processor cannot accomplish as much in a given time as software that controls many computers operating in parallel. Similarly, one person cannot accomplish as much in a given time as a group cooperating toward a common goal. These ideas – parallelism in the operation of computers

and in the cooperation of people -- are duals:

1. MASSIVE PARALLELISM: One person commands a network of many cooperating computers.
2. MASSIVE COOPERATION: One computer network serves many cooperating people.

Parallelism is not simply a convenience to accelerate actions; it is a necessity to overcome fundamental limits in "single entity" computing [Denn85a, Denn85b].

The need to deal with many cooperating computers arises from two factors. The first is the limit on the speed of a single-processor computer, which appears to be on the order of 1-10 GFLOPS. Current software technology, which is strongly tied to sequential processing, is similarly limited. To overcome these limits, we need parallel computer architectures, and we need new programming environments and new programming methods that explicitly take account of computations involving many (hundreds or thousands of) computers. The second factor compelling multi-computer architectures is the heterogeneity of the problems themselves. Many solutions comprise different parts, each best handled by a particular type of machine -- e.g., symbolic, numeric, and graphical processors.

The need to deal with networks that support cooperation among many researchers arises from the limit on one person's abilities. Many can accomplish, in concert, what one cannot alone. Recognizing this, U.S. Government agencies (e.g., DARPA) and major industries (e.g., MCC) are now using a variety of methods to stimulate consortia and other large-scale efforts in which a community goal is achieved by cooperation among many participants. We are entering a new era of scientific projects conducted on a national or international scale. To harness the potential of "massive parallelism" among many scientists, we need communications and distributed systems capable of supporting human cooperation on extensive scales. These systems must deal with persons sharing community goals and interacting loosely. Centralized project management schemes are inherently limited for the scales of cooperation we envision.

The level of abstraction at which a discipline expert deals with a computer must be close to the one used with colleagues:

> **Premise 2.** *We must seek to fold as much discipline knowledge as possible into the computers operating in the scientist's working environment.*

This premise may seem at odds with the principles of "general purpose programming." It is not. We intend to approach the goal of reusing software and methodologies in a new way -- through new software tools that aid in the construction of new high-level, discipline-oriented interface systems.

It is obvious that multidisciplinary teams must be formed to design and build the systems implied by Premises 1 and 2. This fact, often recognized and yet often ignored, is so important that we give it special emphasis:

> **Team Principle.** *Computer scientists and other-discipline scientists must work together as teams to build the envisaged computer systems.*

The most important reason for multidisciplinary teams is the strong possibility that unforeseen new methods will be discovered when old points of view meet in new ways.

A major benefit of the team approach is to bridge the gulf that now exists between computer science and other sciences. On the one hand, most computer scientists do not appreciate the types of computing required for real problems in science and engineering. On the other hand, most scientists and engineers do not appreciate the elegant methods and high-level programming tools developed by computer scientists.

Teams can explore the higher ground, from which mechanisms such as UNIX™ or Fortran libraries will appear as low-level details. Teams can arrive jointly at efficient mechanisms for expressing or invoking computations at the level of abstraction of the discipline. Teams can combine the best of both worlds.

The systems based on these premises and the team principle will be so different in their approach to specifying computations that familiar terms are misleading when thinking about them. For example, "programming" connotes the process of designing and implementing algorithms in a particular language. The systems we envisage will be capable of constructing programs from pictures and other high level specifications given by the scientist user, but the user will not be programming in the conventional sense. Similarly, "language" connotes strings of symbols of given syntax. The systems we envisage will employ pictures, speech, and other means in addition to conventional languages to communicate with their users. Likewise, "productivity" connotes the number of correct lines of code per unit time created by a programmer. The systems we envisage will aid (groups of) scientists to achieve their research goals much faster than today, yet conventional concepts of productivity cannot be used to measure this.

## 2. Overview

The long-term goal of the RIACS core research program is to automate the entire process of scientific investigation, from problem formulation to results dissemination. The implications of this goal, and the progress expected for the currently-funded program during 1986, are summarized in Appendix A. This report addresses a significantly accelerated research program for a major subgoal – large-scale cooperation among humans and computers. Additional funding would enable completion of the tasks described below by 1989.

Working with NASA, we will assemble teams of NASA scientists in selected domains and identify the significant computational problems in their disciplines (the kernel problems). RIACS will contribute computer science expertise to these teams. NASA will contribute other-discipline expertise. (The relation between RIACS and NASA is explained in Section 9.) Domains of critical importance to NASA are computational fluid dynamics, computational chemistry, computational structural mechanics, flight simulation, and control systems (e.g., as required for the Space Station). Working with these teams, RIACS scientists will undertake four projects (shown in italics below), seeking answers to the two general questions of this research:

A. How do we program parallel machines?
   1. *Decomposition of problems into parallel structures.*
   2. *Programmability of specific parallel machines.*

B. How may networks facilitate cooperation on a large scale?
   3. *Distributed program composition system (DPCS).*
   4. *Distributed system development environment (DSDE).*

Project 1 explores a top-down approach to programmability, Project 2 a bottom-up approach. The former seeks to identify suitable architectures for a given problem; the latter seeks to identify suitable problems for a given architecture. By including both approaches, we expect faster convergence on answers to the programmability question. The team principle is also important in these projects. Discipline experts will define kernel problems at high levels of abstraction; computer scientists will define applicable methods and mappings to the architectural level. The teams can be expected to make new discoveries in parallel algorithms.

Projects 3 and 4 explore the duality noted in Premise 1: Project 3 considers how to permit a wide range of resources to be used within a single computation; Project 4 considers how to permit a wide range of people and institutions to cooperate in the construction of distributed systems for their community's research goals.

## 3. Decomposition of Problems into Parallel Structures

Parallel computers are the only architectures capable of achieving sustained computing power in the multi-GFLOPS range. While innovations in design concepts and engineering have made possible an abundance of new machine types, the potential impact of any one of these types on scientific computation remains almost wholly unknown. Few analytical and programming tools exist to smooth the path of research.

This project will explore a "top-down" approach to parallel architectures in scientific computing. The goal is to study the nature of scientific problems in selected disciplines and determine the most suitable parallel architectures and algorithms for solving them. This project will be complemented by a "bottom-up" study that endeavors to learn how to program specific machines likely to be used for scientific computation; that project is described in the next section of this document.

We will form research teams consisting of computer scientists (supported by this project) and NASA-Ames scientists (not supported by this project). The teams will study kernel problems from selected disciplines. A "kernel" is the nexus on which a solution depends; the speed of the entire solution is limited by the speed of the kernel. We will study kernel problems in depth to determine their intrinsic properties, partitionability, and communication needs among computational modules. We will seek optimal parallel algorithms for their solution. We will study the numerical properties, stability, and convergence of these algorithms. We will use the results to determine desirable and undesirable properties of parallel computers for these problems. We will compare kernel problems looking for common properties.

This study will have a strong mathematical flavor. We are aiming to understand the requirements of parallel algorithms at a fundamental level; many of the results will be about parallel algorithms and abstract parallel machines. But we also believe that some of the properties of parallel algorithms can be determined only by experiment; thus part of our effort will be directed toward implementations of our results.

We expect to draw kernel problems from the areas in which NASA-Ames has the greatest expertise -- computational fluid dynamics and computational chemistry. In computational fluid dynamics, for example, there are many challenging kernel problems: turbulence simulation and modeling, separated and vortex flows, Reynolds-averaging for computation of pressure and drag, subsonic through hypersonic flows, aerothermodynamics, and shock analysis. In computational chemistry, there are challenging kernel problems in self-consistent potential fields and molecular dynamics. None of these problems has been seriously studied for solution on highly parallel machines.

We will choose kernel problems whose solutions will give insights far beyond their disciplines. For example, a large class of problems in the fluid dynamics

research combine wind tunnel experiments with numerical simulations as part of basic research in complex vortical flows and turbulent boundary layers. These problems employ implicit methods and thus involve the solution of large, sparse linear systems. Sparse linear systems are of great importance in almost all areas of numerical modeling of large systems [Heat84]. Other areas in which fast parallel algorithms for sparse linear systems would have a significant impact are: (a) discrete approximations of continuous fields arising from partial differential equations involving more than 5,000 degrees of freedom; (b) large linear programming systems in econometrics; (c) static physical structures models in mechanical engineering; and (d) accurate location of geographical positions from satellite data, as in the so-called North American data problem [Kola78].

### 3.1. Task Specification

The general goals of problem decomposition for parallel computation will be met by three subtasks:

1. Pan-Reif algorithm for large linear systems.
2. Finite element fluid flow algorithms.
3. Other fundamental kernel problems.

These subtasks are described below.

#### 3.1.1. Pan-Reif Algorithm for Large Linear Systems

This subtask focuses on experimenting with the new algorithm of Victor Pan and John Reif for large, sparse linear systems. This algorithm presupposes a machine capable of massive parallelism [Pan85]. In an initial study of this algorithm, Tony Chan of Yale University has concluded that it has great potential for exploiting massive parallelism to solve problems in the mechanics of viscous interactions and drag. We will continue this line of experiment. We will look for other applications of the results, such as accurate location of geographic features from satellite data.

#### 3.1.2. Finite Element Fluid Flow Algorithms

This subtask is motivated by the problem of transonic flow around an F-16 fuselage and wings, which is under study at NASA Ames Research Center. The traditional approach for solving such problems has been finite difference equations. Thomas Hughes of Stanford has recently discovered a method of using finite element calculations to compute shocks without computing results that violate the second law of thermodynamics [Hugh85]. Because finite element calculations are easy to map on to parallel machines, this method may open the door for a new family of algorithms capable of handling multi-dimensional

transonic and high-Mach-number flows. Finite element algorithms are more attractive than finite difference algorithms in many applications because the computational space is often bound to a geometric description that may be conveniently expressed in a finite element form.

### 3.1.3. Other Fundamental Kernel Problems

Our researchers will select other kernel problems for study after significant progress has been made with the two above. The selection criteria will be that parallel computation should have a significant effect and that the solution methods should be useful in other areas.

## 4. Programmability of Specific Parallel Machines

A computer system is "programmable" to the extent that it is easy to instruct and use. Parallel machines create demands on programmers that are not supported well by current programming environments. Several barriers must be overcome to make parallel machines easy to program and use:

1. Understanding which problem domains are best matched to given architectures.
2. Developing mappings from the problem spaces in which parallel algorithms are designed to the machine's hardware.
3. Developing higher-level languages that allow the tasks of many processors and their intercommunications to be expressed easily.

The desire for high performance places additional demands on programmers, who wish to lay out programs that are free of communications or processing bottlenecks at local points in the machine. (The ratio of communication time to processing time is important in these machines.)

We propose a project within the class of parallel machines for which Cooperating Sequential Processes (CSP) is the natural program model [Hoar78]. The Intel iPSC hypercube and Sequent Balance multiprocessor are commercial examples. This family of machines is of great interest in the computing community today and is likely to be used heavily in high-performance applications of the future.

In September 1984, we conducted a study of Question (1) on the MIT Static Data Flow Machine [Adam85a, Adam85b]. Although that machine is not a member of the CSP class, the methodology of the study itself can be used to evaluate the utility of CSP machines in specific disciplines.

RIACS has already begun evaluations of two CSP class machines, which are now part of our facility. One is the Sequent Balance 8000 computer, with 12

processors and 16 MBytes of main memory. The other is an Intel iPSC hypercube with 32 processing nodes containing 512 KBytes main memory each. In January 1986, Tony Chan of Yale University will begin a sabbatical at RIACS and will lead a study of programmability of the hypercube for problems in computational fluid dynamics and computational chemistry [Chan85]. Meanwhile, RIACS scientist Harry Partridge is working with NASA scientists Charles Bauschlicher and Richard Jaffee; all are chemists developing hypercube implementations of self-consistent field calculations and molecular dynamics on the hypercube [Part85].

### 4.1. Task Specification

The general goals of studying programmability of CSP class machines will be met by four subtasks:

1. Performance monitoring tools.
2. Virtual machine variants of the hypercube.
3. Efficient problem-to-machine mappings.
4. High level programming environment.

These subtasks are described below.

#### 4.1.1. Performance Monitoring Tools

We will develop tools to study and manipulate the execution behavior of application programs by a 32-node iPSC hypercube computer. Except for a small reduction in available memory space, these tools will be invisible to the applications program. They will give the capability of measuring (by nodes) instruction counts in the applications programs, time spent in computation, time spent originating communication, time spent receiving communication, time spent forwarding communications (to other nodes in the hypercube), time spent waiting for network access, time spent idle, and distribution of communications distances through hypercube. The measurement tools will be designed to be extensible to any CSP-class machine, for example, the Sequent machine.

Experience with parallel machine simulators has shown that event traces and performance data can be extremely valuable for debugging parallel algorithms. Our tools will be designed with this in mind.

#### 4.1.2. Virtual Machine Variants of the Hypercube

The performance monitoring tools will be augmented with virtual machine tools to allow simulations of variations of the basic architecture. For example, we might be interested in knowing whether certain additional or substitute communication paths, outside the formal hypercube structure, would

significantly improve the performance of the machine in some applications. We might like to determine the effect of faster busses or faster processors on the computation and location of bottlenecks.

The virtual machine tools will simulate properties of the variant that are not present on the real iPSC. The performance monitoring tools will correct the data so that it corresponds to what would be observed if the variant were real.

### 4.1.3. Efficient Problem-to-Machine Mappings

One's first impulse in developing an algorithm for an $N$-node hypercube is to partition the physical problem into $N$ parts, one for each node. One quickly discovers, however, that the amounts of computation may vary considerably from one part to another; the exact nature of the variation depends on the data. (In image processing, for example, the amount of computation in each region of the image may depend on the number of distinguishable features in that region.) To avoid the "bottleneck phenomenon" – in which the entire computation is forced to run at the speed of the slowest part – programmers have a strong incentive to construct load-balancing mappings from the problem space to the machine. The search for good mappings will divert energy from the task of finding good algorithms in much the same way as solving the overlay problem diverted programming energy before virtual memory.

This subtask aims at studying the way real computations can be mapped for balanced load on the machine. On the basis of these studies, we will make recommendations on how these mappings can be automated. The performance monitoring tools will be especially useful in formulating these recommendations.

### 4.1.4. High Level Programming Environment

Subtasks 1 and 2 will provide information about the ability of CSP-class machines to support parallel computation in selected disciplines. Subtask 3 will provide information about the ways in which particular algorithms, mapped to the machine, will load the elements of the machine. This subtask aims to integrate this knowledge into a programming environment for CSP-class machines. A starting point for such an environment may be the Poker Programming Environment [Snyd84].

## 5. Distributed Program Composition System (DPCS)

Two factors make necessary the programming of computations whose parts may reside in different machines of a network. One is the proliferation of local networks connecting different types of machines acting as "servers." The other is the proliferation of wide-area networks containing nonmovable facilities that are available as resources to any user of the network. An important technique for dealing with multi-machine computations in a local network is the remote procedure call [Nels81]. An important technique for dealing with multi-machine computations in wide networks is manually opening a connection to the remote machines and issuing appropriate commands. The lack of techniques more advanced than these has been a serious barrier to distributed computing.

We will build a prototype system in which scientists can employ diverse mixes of computers on a network to solve a single problem. The system is based on the premise that resources on a network – programs, data, facilities, and services – can be treated as software parts that can be plugged together as needed to construct computations. Network resources will conform to standards that make them usable as parts. The software tools used by the scientist will be strongly influenced by the intuition of a "part":

1. Programming will be accomplished by positioning parts, represented by icons, on a graphics screen and connecting them together. The resulting diagram will be a data flow network representing a computation comprising various processes, similar to, but higher level than, the object data flow graphs in [Davi82] and the icon-based programming described in [Glin84].
2. Programming will be hierarchical: a picture representing a distributed computation can be collapsed into an icon, which can be used as a new part in future computations.
3. Selection of program components will be accomplished through an interactive database that stores parts descriptions and locates parts by keywords. (The database will eventually be distributed throughout the network.)
4. Parts can be bound to ("mounted on") specific machines in the network.
5. A part is activated by mounting it and supplying its input data. Data transmissions between parts will be realized via network protocols.

We call this a distributed program composition system (DPCS). The DPCS will be designed to work with any network, whether it be local-area or wide-area technology. This is feasible because protocols such as Ethernet and TCP/IP are widely used standards. The DPCS will be "invisible" in the senses that it (a) hides the details of accessing and using computational resources located on many machines and (b) guarantees that program correctness will not depend on any prior assumptions about network topology or distribution of programs and

databases among machines.

During summer 1985, RIACS summer student Jeff Yost, with supervision from RIACS scientists Bob Brown and Peter Denning, constructed a program called Graphical Shell (gsh) that is a prototype for the DPCS. Figure 1 is an image of the gsh screen. The user has specified a computation that splits the output of the UNIX "who" command into two streams, one of which is sorted. Figure 2 illustrates a data flow program for computing factorials. The gsh allows the construction of these pictures with the same effort as using Apple's MacDraw program. When the "run" button is selected, gsh actually executes these programs.

The scientist user will interact with the DPCS from a personal workstation. He will have access to software components in the form of libraries and subroutines that may be distributed among the various computer systems on the network – systems such as a hypercube, systolic processor, optical processor, or vector processor. Transparent naming conventions, message passing protocols, and remote procedural call strategies will allow the user to invoke a program on his workstation and have portions of the computations carried out concurrently or in sequence on various systems throughout the network. The DPCS will transform a workstation into a "distributed system control panel" because the user will feel as though he is in command of one homogeneous system.

### 5.1. Description of Tasks

The distributed program composition system will use the Graphical Shell (gsh) program as a starting point. We plan these subtasks:

1. Implement redesigned user interface.
2. Design and implement parts standards.
3. Design and implement datatype standards.
4. Design and implement location independent binding methods.
5. Develop prototype parts inventory database.

These tasks are described more fully below.

#### 5.1.1. Implement Redesigned User Interface

This subtask will make the gsh interface more usable with such improvements as (a) better icons for programs, files, splits, joins, input ports, and output ports; (b) better methods for connecting arrows (data flow paths) between parts; (c) a "describe" function that pops up a window containing important information about parts or data flow paths; (d) allowing parameters to be set and passed to parts; and (e) allowing any UNIX program to be associated with a part.

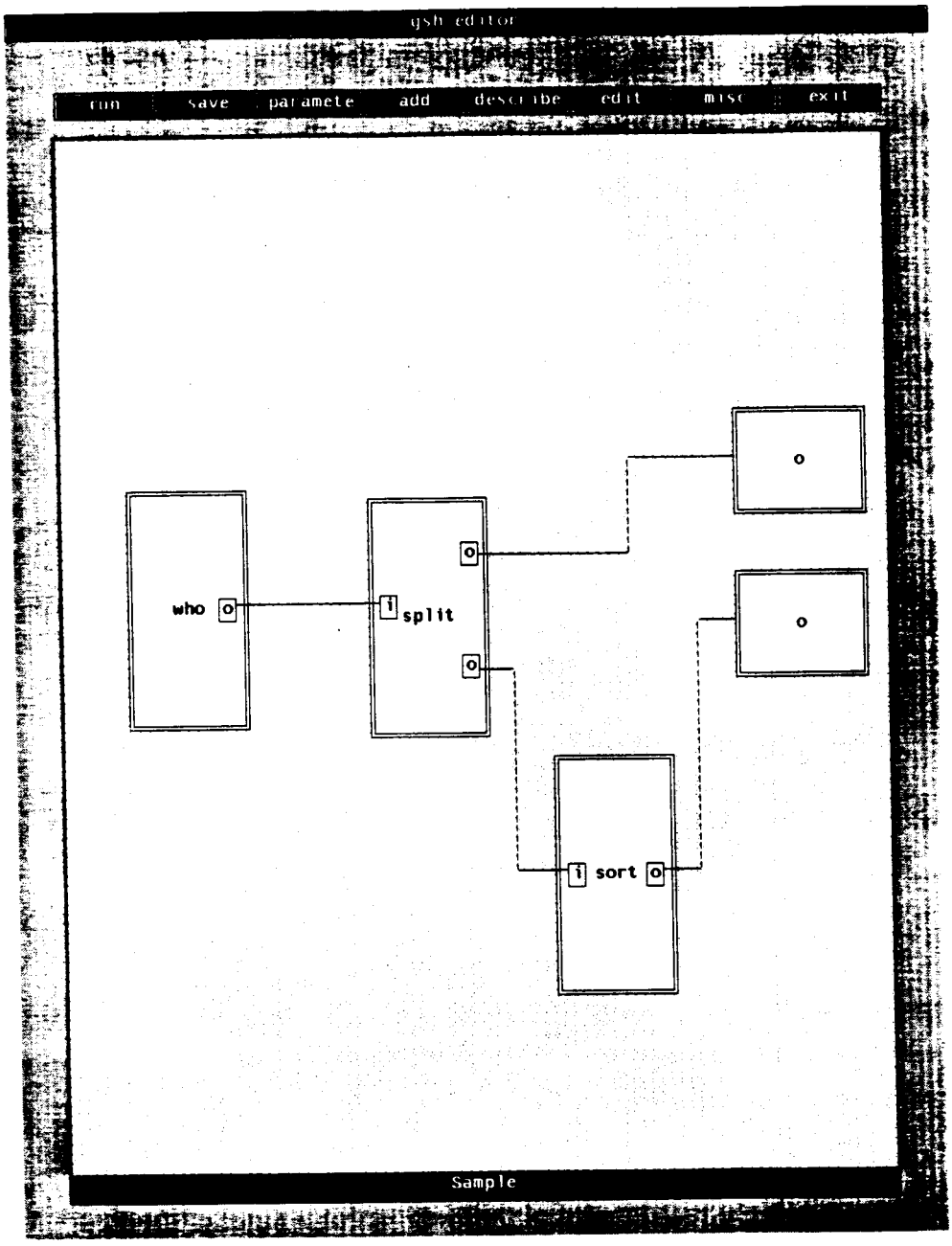


**Figure 1. Simple Graphical Shell Example**

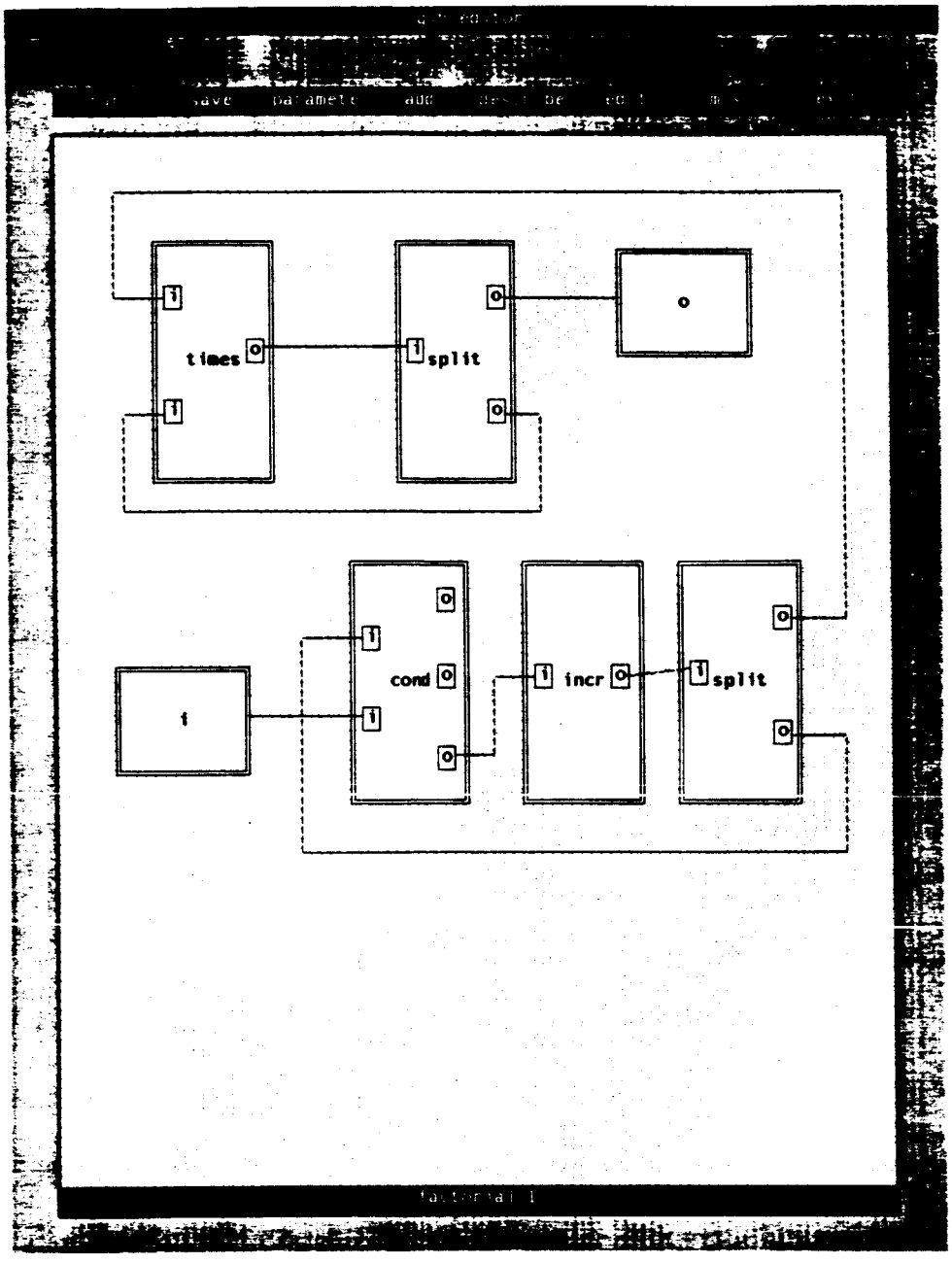


**Figure 2. Graphical Shell Factorial**

### 5.1.2. Design and Implement Parts Standards

This subtask will design the standards for parts throughout the network. A part will consist of an input checker, an algorithm, and an output checker. The input checker will verify that any connections opened to other parts will supply expected datatypes. The output checker will verify that other parts to which connections are established are expecting the datatype supplied. The algorithm

will process input data to output data according to a well-defined functional specification. This strategy allows existing software to be converted easily to the standard form simply by supplying an input checker preprocessor and an output checker postprocessor.

### 5.1.3. Design and Implement Datatype Standards

This subtask considers the problem of specifying datatypes for transmission between parts. The composition system should give warnings if data paths are specified that violate the input or output specifications of parts. The execution system should refuse to start a part whose input or output assumptions are violated. The method of defining datatypes and locating their specifications can be modeled after similar schemes in frame-based expert systems [Fike85] or Smalltalk-80 [Gold84].

### 5.1.4. Design and Implement Location Independent Binding Methods

This subtask makes possible the binding of parts to specific machines anywhere in the network. It must solve these problems: (a) assigning location-independent names to resources in the network; (b) allowing a DPCS user to bind an icon to a resource, after checking that the binding is legitimate; and, (c) designing a mapping from location-independent names to actual network addresses so that the binding can be remapped if the location of the resource changes [Giff81]. Location independent names are essential for reliability: it would be intolerable if programs around the network mysteriously ceased to function or began to malfunction if someone moved a file to a new location.

### 5.1.5. Develop Prototype Parts Inventory Database

This subtask takes up the design of a parts and datatypes database. Each resource usable in a distributed computation would be registered in the database. Specifications of datatypes would also reside there. Parts descriptions would contain standard components, such as a diagram of its icon, specification of the datatypes for each input and output, required machine types if any, and detailed specifications of its function. The database would allow users to locate parts by formulating queries about functions and datatypes. The database would be distributed because no one site can have full knowledge of the entire system's state.

## 6. Distributed System Development Environment (DSDE)

The distributed program composition system gives one scientist command over a large computation spanning many resources in a network. But to achieve cooperation among many scientists dispersed over wide areas, we need new software development aids. Considerable technology already exists for software development environments -- e.g., software configuration control, high level language development of software, and languages for distributed computations. This project seeks to design and demonstrate extensions to these tools for development of distributed computations and to develop prototypes of new tools where needed.

It is important to distinguish between distributed computation and distributed development. The problems of specifying and executing distributed computations are addressed by the previous project. The problems of coordinating many scientists and software developers who are themselves dispersed are addressed by this project.

A good example of a distributed system development problem is a communication network. A network consists of communication processors linked by various communications technologies such as radio and telephone. It also includes protocol software that runs in the user host computers. Several groups must cooperate in the design, development, and fielding of such a network. A network architecture group would specify the interactions among components and design the protocols for those interactions. A network hardware group would design and build the communication processors; they (or possibly a separate group) would design the software for those processors. Subcontractors would provide the communication links connecting the various processors. Finally, each user installation must designate someone to provide hardware and software support for the interface to the network. The ARPANET (the world's first packet switched network) was designed by a cooperative of such groups over a period of about five years. Subsequent DARPA-sponsored network projects [Lein85] have demonstrated the feasibility of distributed development when the developers had the right tools; in those cases, the primary tool for the distributed development was electronic mail over the ARPANET itself.

Other planned systems have similar characteristics. For example, the various Services are planning command and control systems that owe their survivability to distributed redundancy. Networks of sensors to achieve diversity for reliable detection of targets also involve distributed heterogeneous systems. Complex simulations of aerodynamics problems requires a number of programs running on different processors and developed by a number of programmers and scientists. The concept of "Telescience" envisages distributed control systems whereby groups of researchers at various institutions can cooperatively control space-borne experiments and analyze the data. All these systems are more ambitious than the ARPANET and must be put into operation in less time.

When operational, all these systems will include modern networking tools, including Multimedia Mail and Multimedia Real-Time Conferencing [Reyn85], to support collaboration among the many groups using them. But distributed system development also requires the same networking tools. We are planning a separate project with F. Ron Bailey, Manager of the Numerical Aerodynamic Simulator (NAS) Project Office at NASA Ames Research Center, to investigate networking tools to support collaborative research within the community served by the NAS facility. That effort will help provide the communications infrastructure to support the collaborative design of large systems and will be used as a starting point for that part of this project.

Distributed development, however, requires more than conventional networking tools. It requires a software environment for aiding the various groups who are participating in hardware and software development and for enforcing system-wide conventions. The tasks below address this question.

### 6.1. Description of Tasks

The distributed program composition system will serve as the basis for distributed development system. This means that the purpose of the DSDE will be to construct and verify large distributed systems that can be described with the DPCS. The DSDE must provide developers with a means to specify interfaces between modules, performance goals, and real-time requirements; this can be done with extensions to existing revision-control systems and with performance testing operators embedded in the DPCS. The DSDE must also integrate multimedia mail and teleconferencing to permit developers to carry out their work without face-to-face meetings. The DSDE should include simulation tools and facilities for connecting prototypes and testbed systems. This effort will be completed via four subtasks:

1. System Requirements Definition
2. System Specification
3. System Implementation
4. Utility Demonstration

#### 6.1.1. System Requirements Definition

This first subtask will develop the requirements document for the DSDE. That document will define precisely the environment seen by a user of the DSDE. It will assume that each resource used by or usable within the DSDE will be usable from within the DPCS. It will define which existing tools, such as software revision control systems, multimedia mail and teleconferencing, performance measurement, and simulation, will be part of the DSDE.

### 6.1.2. System Specification

This subtask will use the results of the first subtask to generate a specification of the components of the DSDE and their interfaces. It will specify the interface between the user's workstation and the rest of the network. It will define in detail the required data flows, control interactions, and subsystem specifications. We anticipate that the mechanisms of the DPCS will be adequate, but this hypothesis needs to be validated. We will then begin development of the DSDE meeting these specifications.

### 6.1.3. System Implementation

This subtask will implement and test the DSDE specified by the second subtask.

### 6.1.4. Utility Demonstration

To ensure that the DSDE is firmly rooted in reality as its development proceeds, we will use it to develop a distributed computation of interest to scientific researchers. The computation should have the attributes that it requires use of a number of dissimilar facilities, such as highly parallel processing for a portion of the computation and symbolic manipulation for others, and also that the software be large enough that it requires development by multiple programmers and scientists. The specific example will be determined by the time this subtask is started. The later section on the Demonstration System illustrates the type of application being considered.

The demonstration project will be used to design scenarios that will influence the requirements and specifications subtasks. It will then be used as a test for the prototype DSDE itself.

## 7. Demonstration System

Each of the four projects addresses an important aspect of a single theme: overcoming computational limits by large-scale cooperation. To ensure that the projects remain coordinated, we will select a real application in which parallel programming, distributed computation, and distributed development are important. We will use this application as a unifying force against which to validate concepts proposed in each of the projects. This system will also allow us to explore means of folding discipline knowledge into the system architecture where appropriate.

We are committed to interdisciplinary research and are ideally situated for a demonstration of this nature. NASA employs many of the world's leaders in the computational physics disciplines, especially fluid dynamics, chemistry, structural mechanics, and aerospace vehicle design. Although we will initially interact with NASA Ames scientists, it is probable that we will extend the scope of the demonstration to encompass other NASA centers and selected universities.

The remainder of this section will describe a demonstration problem that is particularly attractive to us: the numerical solution of transonic wing flows using an Euler/Navier-Stokes zonal approach [Hols85]. The relation between this demonstration project and each of the four research projects will be indicated.

Computing transonic air flows around the body of a high performance aircraft, such as an F-16 fighter plane, is computationally demanding. These aircraft maneuver at speeds near Mach 0.8; some portions of the plane experience subsonic flows, others hypersonic. The computation comprises several dissimilar components: a precise description of surface geometry of the aircraft, subdivision of the surrounding volume of airflow into appropriate regions for analysis, grid generation within each region, computation of the flow within the regions taking into account the passage of flow across regional boundaries, and graphical representations of the results. Eventually the results of these computations may be provided to an optimizer within a computer-aided design (CAD) system to generate improved aircraft geometry.

Figure 3 shows the components of a system for the transonic flow problem. They are sketched out in a form that might be generated by the distributed program composition system (DPCS). Each system component, represented by a double boxed icon, is a complex software subsystem. For example, a great deal of effort is normally expended to verify the surface geometry generated by a CAD system before it is entered into the data file shown as input to the figure. Grid generation, part art and part science, is complicated by the fact that the volume surrounding the body of the aircraft must be subdivided into zones to reduce the amount of data and computation required. The shape of the zones is strongly influenced by the geometry of the airframe. The boundary layer near the skin of the airframe comprises the viscous-flow zone, requiring the finest mesh and numerically most demanding computation (Navier-Stokes). The zone just beyond the boundary layer may be treated with a coarser grid and computationally less demanding set of equations (Euler), whereas the outermost zone may be dealt with simply by full-potential methods. Zone definitions must satisfy three requirements: first, the grid density and chosen computational technique must reflect the physics of the flow within the zone; second, the data and computation generated by all the zones must be manageable within the computer on which the flow is calculated; and, third, the flow across a boundary between zones must be consistent with the flow on either side of the boundary.

Once the grid description has been formed, the partial differential equations for the flow in each zone must be solved. The computation is usually massive.

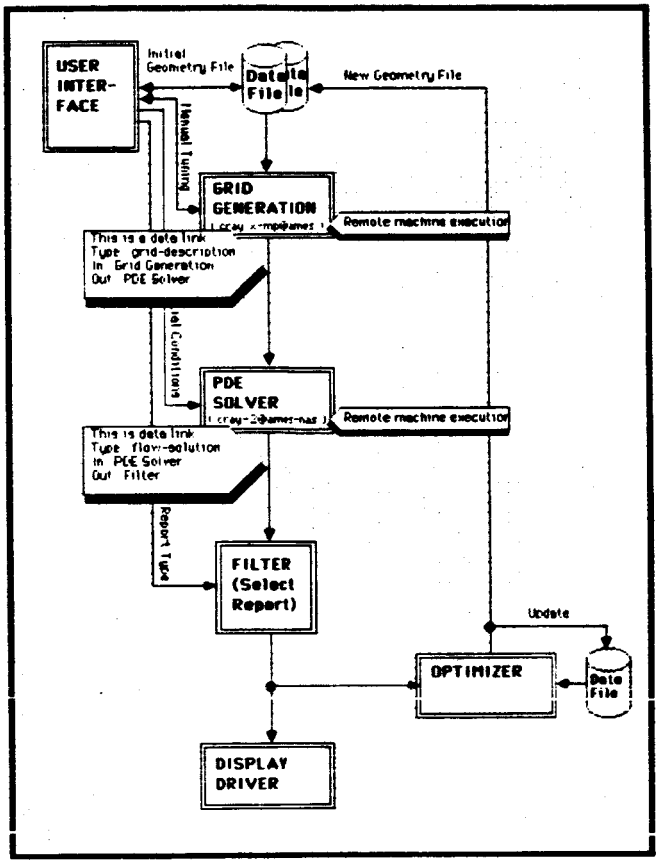


**Figure 3. Transonic Flow Problem**

The result is a large file containing a vector of answers for each grid point. The subset of this file corresponding to a particular type of report requested by the user can be selected for display by a filter; report-types include flow vectors, pressures, and densities. It may eventually be possible to pass a report to the optimizer program in the CAD system, which will produce an improved geometry.

A major complication in this problem is that each component of the system may need to be carried out on a separate machine -- either the machine has special capabilities or it is the only machine on which the software works. In the picture, some icons are labeled with an ARPANET-like address of the machine to which they are bound. For example, the grid generator is mounted on the Cray X-MP in the Ames NAS facility and the PDE solver is mounted on the

Cray-2 in the Ames NAS facility. The other icons are by default bound to the user's workstation.

Each of the projects in this proposal addresses an important aspect of the transonic flow problem:

1. *Decomposition of problems into parallel structures.* The benefits of greatest potential are, first, the possibility that innovative use of a massively parallel computer could lead to significantly faster flow calculation, and, second, a finite-elements method for calculating transonic flows may lead to simpler zoning procedures and mesh generation.
2. *Programmability of specific parallel machines.* The hypercube and other CSP-class machines will provide testbeds for new grid-generation and PDE-solving algorithms.
3. *Distributed program composition system (DPCS).* This system can permit the construction of the distributed computation with no more effort than to draw the picture discussed earlier. That picture shows a possible new-style user interface, including pop-up help boxes that describe components of the picture. Important operations, such as binding the software elements to their proper machines and communicating with them over the network, would be handled automatically by the DPCS during run time.
4. *Distributed system development environment (DSDE).* Transonic flow simulations, such as that described above, are current large-scale research topics. They involve teams of specialists such as aerodynamicists, design engineers, system specialists, programmers, and program managers. Typically, such an assorted team will be distributed geographically and use a multiplicity of computing systems and workstations. The DSDE project is designed to enable such teams to use networks conveniently to coordinate the development process.

A large, computationally demanding project based at Ames, such as that described above, could serve as a unified testbed for the tools and theories of the projects offered in this proposal. There are two advantages to implementing the testbed. First, the feedback to RIACS computer scientists from Ames personnel will lead to a better overall design. Second, the direct contact of the Ames scientific community with leading computing technologies will accelerate the transfer of those technologies from our laboratory into aerospace practice.

## 8. References


[Adam85a]
Adams, G. B., R. L. Brown, and P. J. Denning, "Report on an Evaluation Study of Data Flow Computation," RIACS TR 85.2, Research Institute for Advanced Computer Science (Apr. 1985).

[Adam85b]
Adams, G. B., R. L. Brown, and P. J. Denning, "On Evaluating Parallel Computer Systems," RIACS Technical Report 85.3 (Apr. 1985).

[Chan85]
Chan, T. F. and Y. Saad, "Multigrid Algorithms on the Hypercube Multiprocessor," Research Report YALEU/DCS/RR-368 (Feb 1985).

[Davi82]
Davis, Alan L. and Robert M. Keller, "Data Flow Program Graphs," *IEEE Computer Magazine* **15**(2)(Feb. 1982).

[Denn85a]
Denning, P. J., "Parallel Computation," *in The Science of Computing column* **73**(4), pp. 322-323 American Scientist, (July-Aug. 1985).

[Denn85b]
Denning, P. J., "The Evolution of Parallel Processing," *in The Science of Computing column* **73**(5), pp. 414-416. American Scientist, (Sept.-Oct. 1985).

[Fike85]
Fikes, R. and T Keller, "The Role of Frame-Based Representation in Reasoning," *Communications of the ACM* **28**(9), pp. 904-920 (Sept. 1985).

[Giff81]
Gifford, David K., "Information Storage in a Decentralized Computer System," CSL 81-8, Xerox PARC, Palo Alto (June 1981).

[Glin84]
Glinert, Ephraim P. and Steven L. Tanimoto, "Pict: An Interactive Graphical Programming Environment," *IEEE Computer Magazine* **17**(11)(Nov. 1984).

[Gold84]
Goldberg, A., *Smalltalk-80: The Interactive Programming Environment,* Addison-Wesley, Menlo Park, CA (1984).

[Heat84]
Heath, M. T., "Sparse Matrix Computations," *23rd IEEE Conference on Decision and Control*, , pp. 662-665. (Dec. 1984).

[Hoar78]
Hoare, C. A. R., "Communicating Sequential Processes," *Communication of the ACM* **21**(8), pp. 666-677 (Aug. 1978).

[Hols85]
Holst, T. L., K. L. Gundy, J. Flores, and et al, "Numerical Solution of Transonic Wing Flows Using an Euler/Navier-Stokes Zonal Approach," *AIAA 18th Fluid Dynamics and Plasmadynamics and Lasers Conference*, (July 1985).

[Hugh85]
Hughes, T. J. R., "Symmetric Forms of the Compressible Euler and Navier-Stokes Equations and the Second Law of Thermodynamics," *Computer Methods in Applied Mechanical Engineering*, (to appear).

[Kola78]
Kolata, G. B., "Geodesy: Dealing with an Enormous Computer Task," *Science* **200**(4340), pp. 421-422 (Apr. 28, 1978).

[Lein85]
Leiner, B. and et. al., "The DARPA Internet Protocol Suite," *IEEE Communications Magazine*, (Mar. 1985).

[Nels81]
Nelson, Bruce Jay, "Remote Procedure Call," CSL 81-9, Xerox PARC, Palo Alto (May 1981).

[Pan85]
Pan, V. and J. Reif, "Efficient Parallel Solution of Linear Systems," *17th Annual ACM Symposium on the Theory of Computing*, , pp. 143-152 (May 1985).

[Part85]
Partridge, H. and C. W. Baushclicher, Jr., "Algorithms and Architectures for Computational Chemistry Codes," in *Algorithms Versus Architectures and the Future of Scientific Computation*, ed. T. Tajima and F. Matsen,University of Texas Press, Austin, Texas(1985).

[Reyn85]
Reynolds, J. and et. al., "The DARPA Experimantal Multimedia Mail System," *Computer* **18**(10), pp. 82-91 (Oct. 1985).

[Snyd84]
Snyder, L., "Parallel Programming and The Poker Programming Environment," *Computer*, (July 1984).

APPENDIX A

# Goals of RIACS Core Research Program

Summarized from Annual Reports for 1983 and 1984
and from Cooperative Agreement Proposal (TR-85.16)

## Long Term Core Research Goals

The RIACS core research program has been referred to as Project "R" in RIACS documents, brochures, and reports. The goal of Project "R" is very high level support of the entire process of scientific investigation from problem formulation to results dissemination. Three levels of computer systems are required to support the process of scientific investigation:

1. SCIENTIST'S AIDE: An intelligent user interface system that deals with the user in language, concepts, and pictures of the discipline.

2. CONCURRENT PROCESSING SYSTEMS: Virtual machines and software parts libraries for various models of parallel computation.

3. HARDWARE AND NETWORKS: The actual machinery on which computations are performed.

Project "R" focuses on systems at Levels 1 and 2. Level 2 has received most of the attention so far. The capabilities envisioned in computer systems that meet these needs are:

1. High speed numeric calculation via parallel processing.

2. Rapid solution of partitionable problems using distributed processing.

3. Computing environment to permit rapid development of distributed computations requiring heterogeneous elements.

4. New algorithmic techniques and functional languages to exploit massive parallelism and distributed networks.

5. Classification of problems by best combinations of domains, computational models, and architectures.

6. User interfaces to allow scientists to develop tools and results in their own domains, interfaces tailored to those domains; allowing access to capabilities of many machines.

7. Single user environment integrating symbolic, numeric, and other types of processing systems.

8. Multimedia telecommunications integrated with the computing environment to permit large scale collaboration on scientific research.

**1986 Core Research Goals**

During 1986, we hope to achieve the following. (The names of responsible staff members are in parentheses.)

1. The Intel hypercube will be fully operational and serving as focal point for research and a seminar in new algorithms for CFD and CC. The goal is to discover how many kernel problems from CFD and CC can be solved with algorithms that run (almost) $N$ times faster when partitioned for $N$ processors. At least three problems will be evaluated. (Raugh, Chan)

2. Performance tools capable of measuring processor and network loads on the hypercube will be implemented and used to instrument hypercube programs. Measurements will support parallel algorithm research noted above and will assist in debugging parallel programs. (Adams)

3. A suite of programs that load parallel multiprocessors in various ways will be constructed and used to quantitatively compare parallel machines such as the Sequent multiprocessor and Intel hypercube. (Brown, Adams, Denning)

4. Concurrent C will be made available for experimentation and use in programming hypercube and other multiprocessors. (Brown)

5. Stage 1 of a distributed program composition system will become operational in the form of a graphical shell. It can be used to program computations that require multiple, heterogeneous UNIX machines, for example SUN workstations and the Cray-2 computer. (Brown, Denning)

6. Simulators for the Sparse Distributed Memory (SDM) will be completed and under test. The SDM is a memory of 1 million locations, each containing 1000 bytes; addresses of 1000 bits select groups of memory cells within Hamming distance of 451 bits. This memory organization is well suited for applications in robotics and autonomous systems. Simulators are intended for both the Intel hypercube and the Symbolics LISP machine. In cooperation with Mike Flynn at Stanford, a breadboard model of the memory will be constructed. (Kanerva, Raugh)

7. An image enhancement application will be implemented on the hypercube partly to study new algorithms for processing images and partly to study parallel algorithms that load the machine unequally at different points (e.g., more computation is required in regions of the image containing more features). (Adams)

8. A study of the ability of expert systems to aid CFD research will be conducted and a report issued. (TBD)

9. Prototypes for multimedia mail and conferencing systems tailored to the needs of the scientific community will be developed. A report on architectures that take into account the needs of this community in access control and privacy will be completed.

(Leiner)

10. Bounds on the performance of distributed algorithms will be explored. These bounds will be stated in terms of the communication needs of the algorithms and the capacities of the network. (Leiner, Denning)