| | |
|---|---|
| **Technical Topic Area:** | Systems Software and Services |
| **Proposal Title:** | **Common Runtime Support for High Performance Data Parallel Languages** |
| **Joint Proposal by:** | Syracuse University, Cooperating Systems, Harvard University Indiana University, Rice University, University of Maryland University of Rochester, and University of Texas at Austin |
| **Technical Contact:** | Geoffrey C. Fox<br>NPAC at Syracuse University<br>111 College Place, Syracuse , NY 13244-4100<br>voice: (315) 443-2163 fax: (315) 443-1973 |
| **Administrative Contact:** | Matthew Clark, Office of Sponsored Programs<br>Syracuse University, 113 Bowne Hall, Syracuse, NY 13244-1200<br>voice: (315) 443-9356 fax: (315) 443-9361 |
| **Type of Business:** | Other Educational |
| **Proposed Duration:** | Three years |
| **Proposed Cost:** | $3,346,222 |

**Key Project Personnel:**

| | |
|---|---|
| Geoffrey C. Fox | Syracuse University |
| Alok Choudhary | Syracuse University |
| Tom Haupt | Syracuse University |
| Sanjay Ranka | Syracuse University |
| Marina C. Chen | Cooperating Systems |
| Thomas Cheatham | Harvard University |
| Amr Fahmy | Harvard University |
| Dennis Gannon | Indiana University |
| Ken Kennedy | Rice University |
| Charles Koelbel | Rice University |
| Joel Saltz | University of Maryland |
| Alan Sussman | University of Maryland |
| Wei Li | University of Rochester |
| Jim Browne | University of Texas, Austin |

Parallel Computing is a critical technology whose widespread adoption will only be possible when improved software environments are available. One essential component is high level languages, and several are currently under development for exploiting the data and/or task parallelism available in applications. We propose the development of a basic public domain infrastructure to provide runtime support for high level parallel languages. This would support several projects developing different compilers for a given language such as C++, Fortran but also give a unified support for compilers of different languages.

This proposal will build on several existing Arpa funded compiler development activities. We will design and develop public domain software systems that will provide runtime support for both our and other compiler projects. Our proposal has been planned in three major meetings over a one year time span. It will enhance current Arpa technology and make it more easily usable by other commercial and academic organizations.

Fifteen institutions representing nearly all the HPCC Arpa compiler related projects, were involved in the meetings. We examined and compared the base runtime software needed for HPCC compilers and identified three software areas where communal activities would be particularly beneficial. These three areas were the support of data parallelism, the support of task or control parallelism, and finally the common debugging and performance visualization infrastructure. We also identified other communal activities including common benchmarking procedures, software engineering techniques as well as the need for an ongoing set of meetings. This agenda was described in a group report issued in July 1993. In this proposal, a subset of this Parallel Compiler Runtime Consortium involving eight institutions intends to address the first type of communal software infrastructure – namely the runtime support of data parallel compilers. We will also continue to hold the group meetings and study the other software areas identified above. We considered these as needing further study but a later proposal for their development could be submitted.

This proposal would combine the expertise of eight institutions for producing a common runtime software base which exploits the experience and expertise of the groups. Syracuse University will coordinate this effort. A single software system will support six existing and one new compiler whose design and implementation will be modified to use the new infrastructure. We expect that the stringent requirements of supporting seven different compilers will ensure that the runtime system will indeed be generally useful to the HPCC academic and commercial community. Each of the seven compilers will benefit from this proposal as they will share a high functionality runtime system which has built in the experience and capabilities of the many separate systems - each of lower functionality than the new system. We will pay special attention to software engineering issues for this new runtime system. Cooperating Systems and University of Texas, Austin would be responsible for this part of the project. This task would involve engineering the runtime system for reuse and extensibility as well as testing and validation.

This proposal combines HPCC software development and its dissemination to a broad community; two activities identified as critical in the future HPCC initiative. Our experience in combining several different systems to produce a communal software base should be very valuable as future projects of this type will be surely needed as HPCC systems mature.

The need for high level language support for parallel computers is universally accepted.

This is illustrated by the intense interest in High Performance Fortran (HPF) – data parallel Fortran stemming directly from Arpa funded technology. Although HPF is now adopted by most HPCC vendors, much research remains for parallel compilers – this includes extensions of data parallelism to other languages, support for a wider range of applications than those included in HPF and study of compilers for machines with new and different hardware capabilities. Thus we expect an increase in the need for data parallel compiler work. This proposal will provided the runtime support which will encourage this necessary activity. Our proposal only includes two languages, Fortran and C++. However, we expect that our runtime support system will be applicable to other languages including ADA. Note that data parallelism is a property of the problem and not the language and so one would expect that program execution could be supported by a common runtime system. We refer to such data parallel languages as HPL - High Performance Languages - in the following.

One does need multiple languages because even if parallelism belongs to the problem, different languages will express different forms of parallelism better. Further each language represents a different trade-off between performance, reliability, user familiarity and other metrics. This fact is corroborated by the findings of the recent multiagency workshop on HPCC and grand challenge applications at Pittsburgh. A typical example of software development involved using C++ as a high-level language to achieve modularity, Fortran as a high-performance "assembly language" for coding computationally intensive fragments, and AVS for visualization. Thus integrated support of different languages appears an essential pragmatic feature of high performance computing environment.

This proposal is for three years, starting Sept 1, 1994. The following is a summary of important research issues and innovations that would result from designing such a unified framework:

- *Portable and Scalable Multi-platform Runtime Support*

  Runtime support must efficiently support the address translations and data movements that occur when mapping a shared address space program onto a multiple processor architecture. Compilers and runtime support for HPLs can be built to use multiple independent processors and an interface to a message passing system (such as PVM, Express, MPI, or proprietary vendor libraries). Alternately, compilers and runtime support can use hardware address translation and data migration mechanisms (such as those found on the Kendall Square KSR-1).

  We expect that all HPL compilers will make use of at least some optimizations for reducing data movement costs. In implementations built on message passing systems, this will include techniques such as message blocking, collective communication, message coalescing, message aggregation and use of asynchronous messages. Prototype runtime support for these optimizations exists for the structured, adaptive, block structured and tree structured problem classes. We will develop an integrated runtime support system to carry out address translation and communication optimizations; this runtime support will be built on top of a message passing interface.

  Machines with hardware address translation and data migration mechanisms also require substantial optimization. We will develop runtime support for the mappings between logical program addresses and locations in the machine's distributed memory required by HPL data structure decompositions. The complexity of these mappings will require a careful balancing between hardware capabilities and software exploitation of those capabilities.

- *Integrated Multilanguage Support*

  We will develop common code and data descriptors, and routines that operate on them, for supporting data parallelism in HPLs. This will allow different programming languages to share data structures that are distributed across the memory hierarchy of a scalable parallel system.

  We will design a common compiler data movement interface specification that will provide a set of communication standards for compiled code. Unlike the user level message passing interface standard, the compiler interface can be more extensive in its capabilities, ranging from very low level primitives that exploit special hardware properties to very high level primitives directly coupled to the common array and data structure formats. The interface standard will make it possible to write compilers that achieve a much greater efficiency on a wider variety of machines than we can with current user level message passing mechanisms. In addition, a common runtime interface will allow a compiler to be easily adapted to a new machine, and still allow customization in the library implementation to improve performance.

- *Software Engineering*

  We will develop a methodology for the engineering aspect of the described runtime support to allow ease of use, modification, specialization, and extension. The kind of extension we consider includes support for new distributed data structures, new language features, new runtime system mechanisms and algorithms, and new message passing or distributed shared memory interfaces.

  System level test and validation of the Common Runtime System requires development of a methodology and framework for composing, executing and measuring systems built from programs in multiple parallel languages. The methodology and framework established for test and validation will be valuable to application system developers as an initial basis for development of systems incorporating components built in different parallel programming languages. The methodology and framework will include integration of specifications for measurement and debugging programs composed from components in multiple languages.

- *Interoperable Compilers*

  We would have seven compilers sharing the same runtime system which would be available for use and experimentation in multilanguage systems.

The main research results of this work will be as follows:

- **Portable and Scalable Multi-platform Runtime Support:** We will design a multi-platform runtime system that will reduce communication costs by collective communication, message blocking, coalescing, aggregation and latency hiding on message passing parallel machines. We will build basic versions of this library on widely-available message passing standards such as MPI. We will also investigate optimized versions of the same interface for particular architectures of interest.

  A separate runtime system will reduce data movement costs on machines with hardware support for shared address spaces by properly utilizing those features. Techniques on these machines will include data prefetching, poststoring, address translation optimizations, and cache management.

- **Methodology for Integrated Multilanguage Support:**

  We will develop common code and data descriptors, and routines that operate on them, for supporting data parallelism in HPLs. This will allow different programming languages to share data structures that are distributed across the memory hierarchy of a scalable parallel system.

  We will design a common compiler data movement interface specification that will provide a set of communication standards for compiled code. The interface standard will make it possible to write compilers that achieve a much greater efficiency on a wider variety of machines than we can with current user level message passing mechanisms. In addition, a common runtime interface will allow a compiler to be easily adapted to a new machine, and still allow customization in the library implementation to improve performance.

- **Software Engineering Framework:**

  Deriving an integrated model of common runtime system interaction will require building an engineering framework capable of assimilating existing runtime research. By pressing object-oriented techniques into service for modularization and interface design, the proposed research will advance the state of the art of the software engineering of parallel runtime systems.

  Construction of systems by integration of component programs written in several different languages will require a support environment. The requirements to be addressed include mapping across interfaces, measurement of execution behavior and merging of outputs. An initial support environment will be evolved from the capabilities implemented to support test and validation of the common runtime system and systems of programs built upon it. This initial development environment will allow application developers early access to all of the capabilities to be provided by the common runtime system.

  O-O interface standardization and class inheritance will make possible degrees of component-wise reuse and extension in the final product which would be impossible to achieve otherwise. By demonstrating the stages necessary to abstract and unify various research efforts into a common system, the project will set the standard for other groupware infrastructure efforts in compilers, operating systems, and programming tools.

Since the common runtime system would be a continuing project, new releases have to be tested and validated. We would develop system level test and validation system for these requirements.

- **Interoperable Compilers:** We will develop seven separate compilers each sharing the same runtime system. This will allow a set of important multilanguage and multisystem experiments. Our experience in integrating different technology projects will be valuable for future such activities.

The following are the detailed deliverables:

- **Common Code and Data Descriptors:**

  The design of a "Common Code and Data Descriptor" definition language.

- **Address Translation Mechanisms:**

  - A standard interface for address translation, based on the common distributed array descriptors developed.
  - Runtime support software for table-based translation schemes, demonstrated on (at least) one distributed memory architecture.
  - Runtime support software for hybrid regular/irregular data distributions, demonstrated for at least one distributed memory architecture and one distributed shared memory architecture.
  - Runtime support software that optimizes use of the hardware address translation mechanisms, demonstrated for (at least) one distributed shared memory architecture.

- **Common Computational Functions:**

  Library of common computational functions that uses the common code and data descriptors on one distributed memory architecture and one distributed shared memory architecture.

- **Common Data Movement Routines:**

  - Structured collective communications routines utilizing data descriptors for at least one distributed memory architecture and one distributed shared memory architecture.
  - Unstructured collective communications routines utilizing data descriptors for at least one distributed memory architecture and one distributed shared memory architecture.

- **Common Data Movement Deriving and Optimizing Routines:**

  - Runtime support software for communication optimizations targeted at block structured problems and regular problems, demonstrated on (at least) one distributed memory architecture and one distributed shared memory architecture.
  - Software to implement caching methods that recognize when multiple copies of the same off-processor data item are being requested, demonstrated on (at least) one distributed memory and one distributed shared memory architecture.

  - Software to optimize communication characterized by runtime determined dependency patterns, including development of a small set of primitives capable of covering different communication graph characteristics (e.g. tree, grid, directed acyclic graph). The software will be implemented for (at least) one distributed memory and one distributed shared memory architecture.
  - Interface specifications for runtime support required to partition irregular data structures from concurrent irregular problems and data structures described by runtime data dependencies, for both distributed memory and distributed shared memory architectures.

- **Common Compiler Data Movement Interface Specifications:** Definition of a communication template description language and and its supporting software.

- **Software Engineering**

  *Engineering the Runtime System for Reuse and Extensibility:* The proposed work will produce overhauled object-oriented versions of the runtime system components from each participating team. Each new runtime component will include documentation of internal assumptions about its calling environment as well as pre- and post-assertions about the state of shared system resources. Each component will provide some form of profiling information about itself; many will also be able to drive standard tools for performance analysis and debugging. Through publication of the standard runtime module interface, accompanied by this level of supporting self-documentation, the user will be able to pick and choose from among these deliverable components to construct new, customized runtime systems.

  - *Test and Validation Environment* The common runtime system will be a continuing project. It will be necessary to regularly validate new releases of the common runtime system. There will be a requirement for system level test and validation throughout the life of the runtime system. This requirement can be meet by the System Level Test and Validation System developed herein. The Test and Validation System is a significant deliverable.

    The second deliverable derived from test and validation is an initial development environment for multiple language application systems. This initial development environment, because it will give users early convenient access to an important result of the common runtime system, is a significant deliverable.

- **Retargeting Current Compilers and Developing New Compiler based on the Runtime Support:** Indiana, Syracuse, Rochester, Cooperating Systems, Maryland, Rice would retarget their current compilers to use the above runtime support. Harvard's would use this runtime support for the development of new compiler. These compilers sharing the same runtime system will be available for use and experimentation in multilanguage systems and would be significant deliverables.

- **Compiler Infrastructure:** Syracuse will deliver a base extended HPF compiler infrastructure which extends capabilities of its current prototype Fortran 90D compiler.

There are no proprietary claims.

Parallel Computing is generally recognized as a successful critical technology which underlies the High Performance Computing and Communication Initiative. Parallel computing has been shown to work and will be essential in the many national and grand challenges. However the widespread adoption of this technology is limited by the available application software. We need more attractive powerful software development environments to support the many users and applications of parallelism. This proposal supports this goal by developing a public domain software runtime environment which can be used by essentially all high level data parallel language compilers. These languages represent the scalable portable approach which will be essential for widespread use of parallel computers. Analysis of different problems has isolated the important data and functional parallel aspects of applications. Here we propose a project to support the first(data) form of parallelism which is now relatively well understood but still requires major research and experimentation to find those language features and compiler optimizations which are needed and effective. Our proposed runtime software infrastructure will support several projects developing different compilers for a given language such as C++, Fortran and ADA and so make rapid prototyping of new compilers much easier. Further our software would also give unified support for compilers of different languages and so enable experimentation with environments that simultaneously support several different languages.

One does need multiple languages because although parallelism is a feature of applications and not solely the language, different languages will express different forms of parallelism better. Thus there is currently no universally "best" language; each excels in different aspects of performance, expressivity, reliability, user familiarity and other metrics. This fact is corroborated by the findings of the recent multiagency workshop on HPCC and grand challenge applications at Pittsburgh [3]. A typical example of grand challenge software development involved using C++ as a high-level language to achieve modularity, Fortran as a high-performance "assembly language" for coding computationally intensive fragments, and AVS for visualization. In some cases, including Multidisciplinary Analysis and Design, AVS is also used to coordinate different modules that make a complete metaproblem. Thus integrated support of different languages appears an essential pragmatic feature of high performance computing environments. The growing interest in heterogeneous metacomputers linking different machines of different architectures will also emphasize the need for integrated support of several languages with a common runtime system.

The need for and utility of high level language support for parallel computers is universally accepted. This is illustrated by the intense interest in High Performance Fortran (HPF) – data parallel Fortran stemming directly from ARPA funded technology. Although HPF is now adopted by most HPCC vendors, much research remains for parallel compilers – this includes extensions of data parallelism to other languages, support for a wider range of applications than those included in HPF and study of compilers for machines with new and different hardware capabilities. Thus we expect an increase in the need for data parallel research and development. This proposal will provided the runtime support which will encourage this necessary activity. Our proposal only includes two languages, Fortran and C++, but we expect that our runtime support system will be applicable to other languages including ADA. Note that data parallelism is a property of the problem and not the language and so one would expect that program execution could be supported by a common runtime system. We refer to such data parallel languages as HPL - High Performance Languages - in the following.

This proposal combines the expertise of the eight institutions to produce a common runtime software base which exploits the experience and expertise of the groups. In the following section IIIE.1, we describe the history and status of the Parallel Compiler Runtime Consortium whose workshops have shown the need and technical backing for the software system proposed here. In section IIIE.2, we describe a simple architecture for the overall runtime system to show how the various pieces of work fit together. We have divided the total project into eight tasks each with a lead institution and associated team which is a subset of the proposing organizations. This project management structure is detailed in the statement of work in section III.H. Syracuse University is responsible for the overall monitoring and coordination of these eight tasks whose technical description is given in eight subsections of section IIIE.3. The eight tasks fall into three types. Six tasks of the first type develop individual components of the whole runtime system. Two other tasks are of a different type and correspond firstly to use of the new software system in seven different compilers and secondly to software engineering which will be critical in a project of this kind.

This one software system will support six existing and one new compiler whose design and implementation will be modified to use the new infrastructure. We expect that the stringent requirements of supporting seven different compilers will ensure that the runtime system will indeed be generally useful to the HPCC academic and commercial community. We will pay special attention to software engineering issues for this new runtime system. Cooperating Systems and Texas would be responsible for this part of the project. This task would involve engineering the runtime system for reuse and extensibility as well as testing and validation.

This proposal combines HPCC software development and its dissemination to a broad community; two activities identified as critical in the future HPCC initiative. Our experience in combining several different systems to produce a communal software base should be very valuable as future projects of this type will be surely needed as HPCC systems mature.

# 1 Parallel Compiler Runtime Consortium

With the encouragement of ARPA, especially Gil Weigand, we organized a series of informal(get togethers at ARPA PI meetings) and formal meetings to discuss the possibility of defining a common runtime system to support the different compilers. A total of 15 institutions including industry and academia were involved. In addition to those represented in this proposal other groups in the meetings included ARPA itself, Argonne National Laboratory, Berkeley, California Institute of Technology, University of Illinois-Urbana, University of Massachusetts, University of Oregon, Stanford, Software Options and Verdix. Our goals were:

- Provide a machine-independent layer for portability across machines
- Enable shared efforts among groups of developers
- Encourage better software design through definition of interfaces between pieces of software

- Provide validation of results by facilitating comparisons between different approaches
- Allow inter-operability between different runtime systems
- Enable comparative study of multiple programming paradigms and multiple machine architectures
- Provide a framework for identifying commonality in runtime systems built for ostensibly different environments

These discussions led to the formation of the Parallel Compiler Runtime Consortium(PCRC) and at a meeting at Syracuse in June 1993, we summarized our work in a group report (Common Runtime Support for High-Performance Parallel Languages) which is available electronically and is being modified for publication. We also presented and discussed this material at a public workshop held during Supercomputing 93. We examined and compared the base runtime software needed for HPCC compilers and identified several areas where communal activities would be particularly beneficial. These were:

- The support of data parallelism
- The support of task or control parallelism
- The common debugging and performance visualization infrastructure.
- Common benchmarking and software engineering procedures
- Support of software to integrate different language systems in the same runtime environment

In this proposal, a subset of this Parallel Compiler Runtime Consortium involving 8 institutions intends to address the first type of communal software infrastructure – namely the runtime support of data parallel compilers – as well as the software engineering and benchmarking issues. We will also continue to hold yearly group meetings and study the other software areas identified above. These are discussed in the group report and as they become better understood and unified, other subgroups of the full consortium can be expected to prepare proposals to address these other issues. As this consortium evolves, we will discuss it s future with ARPA. In particular we have had interest from other organizations – including those not currently funded by ARPA. It would probably be beneficial to expand the membership of PCRC and as part of this proposal, we are prepared to organize this larger activity. This proposal provides funding for both these meeting and the coordination of the work of the various groups. This coordination will be the responsibility of Syracuse University.

# 2 Runtime Support for Data Parallelism

Figure 1 provides a high level description of the different layers of runtime support required for supporting Data parallelism. Runtime support must efficiently support the address translations and data movements that occur when one embeds a globally indexed program onto a multiple processor architecture. We expect that all HPL compilers will make use of at least some optimizations for reducing data movement costs such as message blocking, collective data movement routines, message coalescing, aggregation and latency hiding. Thus, a common runtime support must define

- Distribution of different data structures across processors
- Association of a data structure with an index set

and provide

- Procedures for determining how data structures are to be partitioned between processors
- Methods for accessing distributed data using global indices
- Procedures that remap data sets
- Optimized Data Movement Procedures
  - Structured Data Movement Routines
  - General purpose gather/scatter Communication

This support needs to be built on basic data movement operations which are provided by a given architecture.

# 3 Proposed Work

The proposed work will produce overhauled object-oriented versions of the runtime system components from each participating team. These components together would provide a portable, scalable, multilanguage, multi-platform runtime support for data parallelism. A high level description of the important tasks is given in Figure 2. The figure shows the dependencies between the different tasks. In the following subsections, we describe the important issues and the corresponding work required for each of the tasks.

## 3.1 Common Code and Data Descriptors

Because of the scale and complexity of grand challenge applications that will be ported to high performance machines, HPC programs written in Fortran 90, C++, Ada, and other relevant languages will need to work together. This implies that an HPF program may need to call routines and libraries written in HPC++ and vice versa. The challenge for the compiler implementer is to make certain that array and structure data types have a run-time representation that is consistent between languages and, if possible, across machines. In the case of HPF this is a non-trivial problem. Array alignment and distribution information passed from the programmer to the compiler by HPF annotations must be encoded into the runtime structure of the array so that a software module written in another language can understand

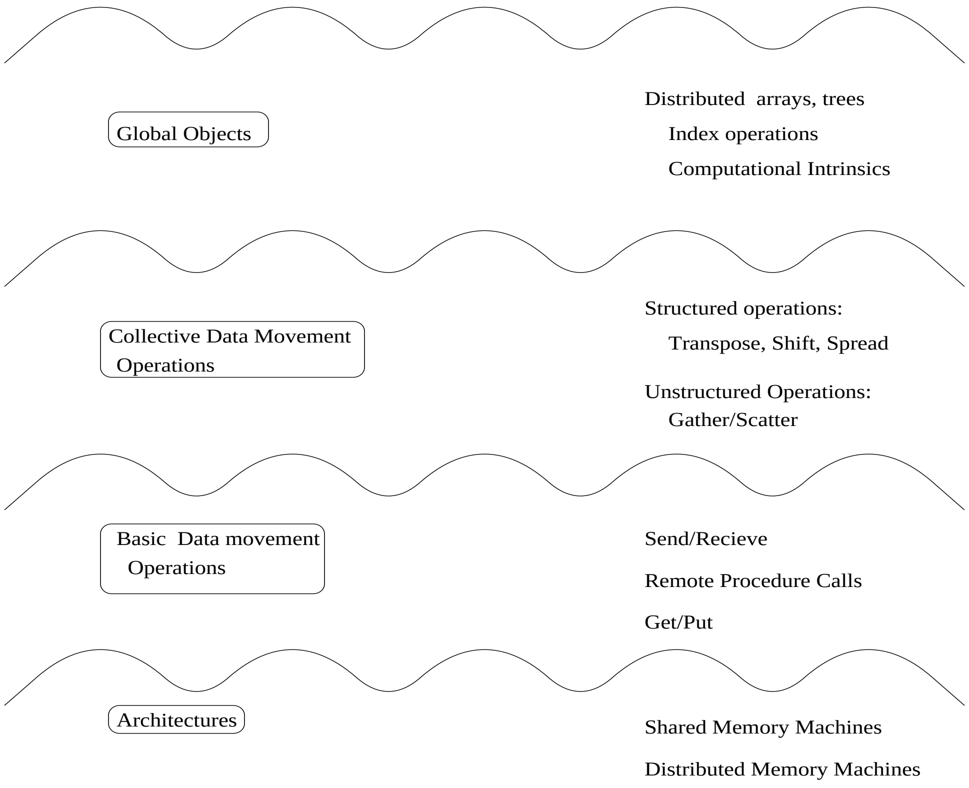


Figure 1: Layers of Runtime Support

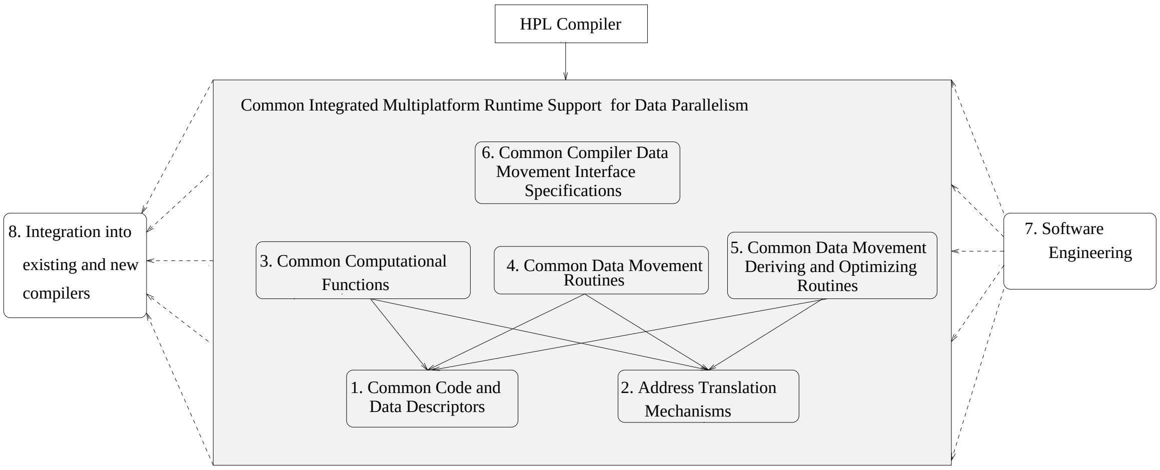


Figure 2: Overview of the Common Runtime System (CRTS)

the data placement. Consequently, data access protocols as well as access functions should be shared.

An interface standard for the layout descriptors for distributed data structures will have to be flexible enough to describe the way data is partitioned and accessed on all contemporary MPP systems. The most challenging problem to be solved is how to build the data access descriptors and functions in a "platform neutral" manner. Unfortunately, because of the diversity of the architectures and the fact that commercial and research systems are already well along the development cycle, we will not be able to dictate the subroutine interface to each runtime system.

A platform-neutral solution is one that supplies only access policy mechanisms and not specific bit-by-bit data layout rules and operations. Thus, we will provide a common language for the description of the layout and access methods. From this common description we can define access templates which can be used in the design of portable compiler and library systems. Consequently, it will be possible to define an implementation for each machine that best exploits the hardware of that machines but still provides one coherent view of the data to the different language implementations.

Our technical approach to this design will be to organize a series of meetings of the groups involved in the design of HPF[1], HPC++[24, 25, 26, 27] and parallel ADA systems. This will include hardware and software vendors as well as academic research efforts. This committee will draft the layout specification document which will be used in defining the common compiler data movement interface specification.

## 3.2 Address Translation Mechanisms

Given the descriptors for distributed arrays, the runtime system must still have standardized methods of using them to produce address translations. In essence, this defines a semantics for the syntax of the descriptors. An important subproject will therefore be to develop a standard interface for address translation that will be closely tied to the distributed array descriptors. This interface will include:

- Formulas and procedures for finding the processor storing a particular global index (i.e. an array reference made relative to the full array).
- Formulas and procedures for translating global indexes into local indices (i.e. an offset relative to the subsection of an array stored on a particular processor).
- Formulas and procedures for translating a local index to a global index.
- Procedures for querying attributes of a global or local index, such as whether the corresponding array element is replicated on other processors.

This will form a common basis for compilers to represent array references, thus allowing higher-level operations to be defined. It is important to note that compilers may optimize the formulas in the actual implementation, and thus may not appear to use them directly.

However, these address translation mechanisms will ensure that there is agreement on data layout and addressing conventions, thus avoiding integration problems that may exist today.

Additional runtime support is required for irregular data distributions. For distributed memory machines, we will work on table based address translation schemes, such as the paged translation tables used in the Parti/Dybbuk routines for irregular problems [12]. Also, we will investigate techniques for handling address translation for hybrid regular/irregular distributions. For example, we will look at the issues that arise in translating addresses for a multi-dimensional array that is irregularly distributed in one dimension and block distributed in another dimension.

For machines that provide hardware support for distributed shared memory address translation, we will look at the usefulness of techniques for addressing irregularly distributed data that have already been developed for distributed memory machines [21]. However, such techniques should improve the performance of the hardware address translation mechanisms, by improving locality (i.e. making more data accesses local, rather than requiring data located elsewhere). For example, on a KSR machine, address translation optimization should be performed with the KSR subpage and page structure in mind, and data migration (communication) should take advantage of the KSR hardware support. We also plan to investigate the hardware address translation properties of the Cray T3D machine, for such operations as remote (off processor) scatters and gathers of distributed array data.

## 3.3 Common Computational Functions

Several basic computational operations are very valuable in parallel algorithm design for massively parallel machines. HPF adds several classes of parallel operations to those in Fortran 90, either as intrinsics or as standard library functions. Some of the important computational intrinsics which are part of HPF are

- *Simple Reduction Functions*: ALL, ANY, COUNT, MAXVAL, MINVAL, PRODUCT, SUM.
- *Combining Scatter Functions*: ALL_SCATTER, ANY_SCATTER, COUNT_SCATTER, MAXVAL_SCATTER, MINVAL_SCATTER, PRODUCT_SCATTER, SUM_SCATTER
- *Prefix and Suffix Functions*: ALL_PREFIX, ANY_PREFIX, COUNT_PREFIX, MAXVAL_PREFIX, MINVAL_PREFIX, PRODUCT_PREFIX, SUM_PREFIX, ALL_SUFFIX, ANY_SUFFIX, COUNT_SUFFIX, MAXVAL_SUFFIX, MINVAL_SUFFIX, PRODUCT_SUFFIX, SUM_SUFFIX
- *Array Sorting Functions*: GRADE_UP, GRADE_DOWN
- *Array Manipulation Functions*: CSHIFT, EOSHIFT, TRANSPOSE.
- *Array Location Functions*: MAXLOC, MINLOC
- *Array Construction Functions*: SPREAD, MERGE, PACK, UNPACK

- *Vector and Matrix Multiplication Functions*: DOT_PRODUCT, MATMUL.
- *Bit Manipulation functions*: IAND, IOR, POPCNT, POPPAR, LEADZ
- *Elemental Intrinsics functions*: SIN, COS, TAN

We have developed a library of routines for a subset of HPF intrinsic functions for different data distributions [4]. We would develop a library of common computational functions that uses the common code and data descriptors described in the earlier subsections

## 3.4 Common Data Movement Routines

Efficient parallelization of data parallel constructs on distributed memory machines requires the use of collective communication to access non-local data. This communication could be structured (like shift, broadcast, all-to-all communication) or unstructured.

We have developed routines which remap data between regularly and irregularly distributed arrays [17]; converting arrays from one data layout to another [10]; carry out structured collective communication patterns such as shift, transpose, spread, scans, etc. [8]; scheduling and carrying out unstructured collective communication [19, 20]; carry out the communication needed to fill in overlap regions or ghost cells [2, 8]; carry out the communication associated with moving regular subsections within a particular distributed array or between two conforming or non-conforming distributed arrays [2]; carry out the dynamic data partitioning and communication patterns associated with many tree structured problems [6, 7].

We will develop a common library of collective structured and unstructured data movement routines. Experimentation and experience gained from development of these routines would be useful in the development of the Common Compiler Data Movement Interface (to be described later).

The common runtime system needs to support explicit synchronization for loop level parallelism, which is unique to shared memory parallelizing compilers. For parallel loops with data dependences between iterations, explicit synchronization must be inserted by shared memory parallelizing compilers. This is not the case for message passing compilers, since synchronization in most cases is implicit in message passing.

## 3.5 Common Data Movement deriving and optimizing routines

We will develop a common set of routines that optimize the data movement required by various data distribution schemes at runtime. These routines will help provide efficient implementations of the data movement primitives for different communication patterns required by the routines in the common data movement library. Such routines would use runtime techniques to optimize communication for regular, irregular, block structured, tree structured and other types of problems. Since the common runtime interface is targeted at multiple machines, the runtime techniques may include both machine independent and dependent optimizations, and provide efficient implementations for both message passing and distributed shared memory machines. The routines for regular and block structured problems would be based on existing work at Syracuse and Yale for the Fortran 90D compiler [8, 15, 9] and Maryland for the multiblock Parti library [2, 22]. The routines for problems with completely irregular data distributions would be derived from the Parti/Dybbuk routines [5, 11, 12, 17]. Routines for tree structured communication distributions would be drawn from work carried out at Yale [6, 7].

We will also use the results of ongoing research to develop runtime support that implements optimizations that target communication patterns characterized by runtime dependency patterns. In runtime dependent communication patterns, communication cannot occur concurrently as dependency conditions constrain the order in which communication can be carried out. Development of efficient communication optimizations for runtime dependent communication patterns is an active area of current research. A joint project [16] by Yale and IBM tackling the problem of deriving such runtime dependent parallelization and communication optimizations by new compile-time analysis techniques and a compiler-generated runtime "scheduler generator". Optimizations targeted at tree structured problems have been carried out at Yale [6, 7]. Also, a joint Maryland, Rutgers, and IBM project has investigated scheduling of runtime dependent communication patterns in the context of sparse factorization methods [23].

We will develop a common interface for the partitioning (and repartitioning) of irregularly distributed data structures. The partitioners will use a (distributed) representation of information that is commonly used to generate irregular data structure partitions. In general, data structures for irregularly distributed data are defined in the form of a graph. The information contained in such graphs that can be used for partitioning can be limited to one or more of the following: 1) the connectivity of the graph representing dynamic dependence information, 2) the weights of the nodes of the graph representing the computational load, and 3) a static graph representing the physical geometry of the computation. Our interface will also be defined in a way that makes possible encoding of runtime data dependency information. An interface will be developed both for the partitioning algorithms (to define both the input and output data for a partitioner), and for the support routines for moving data associated with partitioned graphs. Also, the data structures required for a common interface will be defined. This definition will draw on work associated with the generic **GeoCoL** graph (which allows the use of **G**eometry, **C**onnectivity and **L**oad information) for the Parti routines [17], the IDG (iteration dependency graph) representation for the scheduler generator [16], and on the results of ongoing work by the Yale, Maryland, Syracuse and Rice groups on data structures that encode runtime dependency information [6, 7, 21].

Tree structures are useful in supporting many applications such as N-body simulations. Runtime support for tree operations can be divided into two interacting subsystems, *link traversal* and *per-node computation*. A tree traversal begins as a single-link traversal to the logical root, following which control alternate between single-link traversal and per-node evaluation until the traverse is complete. A third subsystem, the *mailbox module*, supports the sending of raw or typed data between any two tree nodes, using some application-supplied global addressing methods. These primitive operations can be used for developing more complex application-specific protocols (e.g. PUT and GET [7]) which may involve multiple synchronization and communication phases. A fourth subsystem, distribution module, provides key services to the other three, resolving logical link traversals and global mailbox addresses to either local pointers or remote processor/pointer pairs. Data distribution, adaptive load balancing and optimization of physical communication are the responsibility of the distribution module.

More concretely, the work to be done consists of:

- Runtime support for communication optimizations targeted at block structured problems and regular problems in which the number of processors and array dimensions are not bound until runtime. Carry this out for one distributed memory architecture (e.g. Intel Paragon, TMC CM-5, IBM SP-1) and one distributed shared memory architecture (e.g. KSR-1, Cray T3D).

- Optimizations for concurrent irregular communication patterns, including implementation of software caching methods that recognize when multiple copies of the same off-processor data item are being requested. Carry this out for one distributed memory and one distributed shared memory architecture.

- Optimizations for communication characterized by runtime determined dependency patterns. Draw information from ongoing research projects in this area, evaluate and compare different approaches to scheduling runtime dependent information. Develop a small set of primitives capable of covering different communication graph characteristics (e.g. tree, grid shaped, directed acyclic graph). Implement for one distributed memory and one distributed shared memory architecture.

- Standardization of common interface for runtime support employed in partitioning irregular data structures used in concurrent irregular problems. Implementation using standard interface in distributed memory and distributed shared memory architecture.

- Standardization of common interface for runtime support employed in partitioning data structures described by runtime data dependencies. Implementation using standard interface in distributed memory and distributed shared memory architecture.

## 3.6 Common Compiler Data Movement Interface Specifications

Compilers need efficient runtime support to move data between address spaces in distributed memory MPP systems, or between memory hierarchies in machines with a shared name space. In many cases these are regular data moves involving blocks of data that are permuted in well defined patterns. In other cases, these are random or indirect data access operations where support for message coalescing, data caching and prefetching are all essential.

From the applications programmers who is working in pure SPMD, non-HPF environments, MPI (the Message Passing Interface specification) has been proposed as a standard subroutine library for message communication between address contexts in a parallel systems. The MPI library is designed for end users, i.e. application and library designers. The purpose of MPI is to make it possible for users to write portable software. However, it is important to note that even though user code must be portable, compiler object code is tailored to an architecture. Just as with a machine instruction set design, where CISC (complex) interfaces to the hardware are now known to be inferior to the RISC (simple) interfaces that are used by the compiler, it is necessary to rethink message passing in terms of what a compiler is capable of exploiting most efficiently.

Another way of expressing this is that a library programmer is motivated to write one code that will work on a large number of machines. On the other hand each compiler is tuned to generate code for a specific machine. Our goal is not to write a single family of compilers that can be easily targeted to any machine. To do this we need a communication interface that makes it easy to express the communication potential of each machine as a set of simple instruction templates. In this way once compiler interface can be quickly adapted to each system.

This project will deliver a CMPI (Compiler Message Passing Interface) which will be called only by compilers. Because our objective is to build efficient reliable compilers in a short amount of time, we need a simple language to describe the primitives of data movement rather than a package that is efficient on some machines and slow on others.. Consequently, the interface language must allow the compiler to exploit special hardware properties and it must be linked to the common array and data structure formats. CMPI will differ from MPI in that it will have less protection against improper use by users and it will be more like a language to describe the primitives of the communication architecture. The compiler will them be programmed with the templates for communication that are most appropriate for each architecture. The interface standard will make it possible to write compilers that achieve a much greater efficiency on a wider variety of machines than we can with current user level message passing mechanisms.

There are a number of ways in which one might design the communication interface language. One approach to a common data movement interface is to provide a global name space. Moving data is then nothing more than reading and writing values as is done on a shared memory architecture. However, for massively parallel machines this is insufficient. One needs support for data movement as collective operations among a set of execution threads. An additional requirement is the ability to read and write blocks of data of arbitrary shape and size. On some machines it is necessary to build these operations as split-phase communications (where a data read or write may be initiated and other computation can be allowed to progress until the read or write is complete. The alternative, RISC approach to the design is to examine exactly what the compilers that currently exist are capable of exploiting. This will serve to focus the definition and allow an initial specification to be designed in the shortest possible time. Our plan will be to follow this more direct path. If at the end of the first year, we feel there is need to examine alternative strategies we will use the initial design as a base upon which to build.

We will set a series of meeting to take place in year one of this project. The attendees will be restricted to those directly involved in building HPF or HPC++ systems. The purpose will be to decide exactly what sort of a common interface is necessary and how we can decide on a standard. Care must be taken because many efforts are well along and many have a local solution to these problems . Our goal is to see how we can combine the experience and wisdom these early efforts to build a common framework that can be exploited as an industry resource and not a proprietary technology.

## 3.7 Software Engineering Issues

### 3.7.1 Engineering the Runtime System for Reuse and Extensibility

Our integrative approach to common runtime system design will require a significant amount of internal standardization, incremental analysis of project requirements and tool environments, and redesign of existing research prototypes. This process of cross-team integration will advance the state of the art in the software engineering of runtime system design by solving three primary engineering challenges.

**Modular encapsulation of commonality and difference.**

For the first time on any significant scale, the proposed work would derive a conceptual model of parallel runtime systems as networks of plug-compatible, cooperating modular services. The first stage in the design of a common runtime basis will identify the basic tenets of this model in light of existing runtime system research.

Any process of component-wise abstraction and integration starts with analysis of modes of commonality and difference. If the end product of integration can be thought of as a "mesh" of cooperating modular runtime system services, then there are two sources for discerning commonality and difference, namely, the "nodes" and "edges" of this mesh.

The first step is characterization of existing work in light of the mesh edges, that is, the common styles of interaction between runtime system components (common code and data descriptors, address translation mechanisms, and routines for moving data between modules). We can then begin to draw out the commonalities and differences between mesh nodes; that is, to define what it means for runtime modules to be cooperative or interchangeable by virtue of their design.

**Taxonomy of functionality and requirements.**

This process of initial analysis will produce a requirements document for the design of a given runtime system module. This document will include a characterization of the assumptions which each module makes about the runtime environment at the point of invocation, any requisite pre- and post-assertions, use of shared resources, and cooperative or competitive relationships with other runtime components.

Coupled with the project-global overview of the functionality of available runtime modules, we can proceed to derive a taxonomy of runtime services. This will suggest an integrative strategy for the final product, and lay the groundwork for the final design of the common module interface.

**Cross-team integration through incremental redesign.**

If the first challenge is the definition of the building blocks of the common runtime system, and the second the mapping of the functionality and requirements of existing runtime services, then the final task is to engineer the concrete interfaces and execution environment which will pull together all contributions into a releasable whole.

Most current runtime design efforts take place in either C or C++; therefore, it makes sense to use the object-oriented properties of C++ to define the edges and nodes of the mesh of runtime services as class abstractions. This strategy will use specialization and multiple inheritance to concretize our desire for "interchangeability" and "cooperation" among runtime modules. Once the requisite requirements analysis has been performed, retrofitting conformant O-O interfaces to runtime research in progress will be a straightforward exercise for each participant group.

Adherence to object-oriented engineering concepts and the production of requirement documents for the construction and integration of new runtime modules will combine to promote commercial reuse and extensibility of the common runtime system.

By making specific runtime services available to users through a standard interface, in an industry-standard programming environment (that of C++), the proposed work will lower the barrier to entry of new researchers and product developers into the parallel runtime software market.

The core technologies for these component runtime designs have been maturing in the literature for years. But by finally overcoming the significant software engineering challenges, this work will transform the common runtime effort from a bundle of academic curiosities into a viable research tool for the larger parallel programming community.

### 3.7.2 Test and Validation of the Common Runtime System and Multiple Language Application Systems

A major goal of the Common Runtime System is to facilitate composition of application systems from components written in different parallel languages. Validation that it is consistent across all of its referents can be accomplished only through thorough system level testing. System level testing requires development and careful measurement and analysis of total application systems involving components written in different languages. There is no established methodology and there are no tools focusing on development of programs composed from components written in multiple languages. Yet even test and validation will require development and analysis of significant applications. Additionally these applications will have to be run many times in many different environments and many different parameterizations for validation. The capabilities developed for system level test and validation can be evolved into an initial application development environment which can be used by application system developers while more elaborate development support environments are designed and developed.

#### System Level Test and Validation

System level test and evaluation of the common runtime system will be accomplished by building application systems on top of it which include multiple languages, multiple execution environments and which include use of debugging and performance evaluation on this software. It is only with a comprehensive approach to exercise of the common runtime system which includes use of all facilities that its consistency and effectiveness be evaluated. Development of such a comprehensive test and validation system from "scratch" would be a major enterprise which should consume 25%-30% of the total effort of the project if a usable system is to result. Fortunately there already exists a project which provides a framework for test and evaluation of the common runtime system which can greatly lower the cost and effort of test and evaluation.

The CODE 2.0 [38] parallel programming system is intended to support multiple languages, multiple execution environments and including use of debugging and performance evaluation. In particular CODE 2.0 can invoke components specified in such languages as CC++ and Fortran 90. CODE 2.0 incorporates a highly dynamic model of parallel computation which will enable ready development of a wide spectrum of tests which can lead to a through evaluation. The class hierarchy structuring of the code generation phase of the development environment is specifically designed to enable translation to multiple execution environments. The debugging facilities integrate shared memory and distributed memory programming styles [39]. Programs prepared in CODE can ready exercise almost all of the features of the the common runtime system from a single source in a consistent and uniform manner.

This test and evaluation approach will focus on integration and coordination of all of the runtime system's facilities across languages, execution environments and functionality. It will be assumed that use of common runtime systems' functions by individual compilers have been thoroughly unit tested by the responsible developers.

#### Development Environment for Multiple Language Systems

Extension of the System Level Test and Validation support system to be a primitive development environment will be straightforward. The major additional effort will be development of documentation and an on-line help system.

## 3.8 Retargeting Current Compilers and Developing New Compiler based on the Runtime Support

We would integrate the above runtime support into our existing compiler work at Cooperating Systems, Harvard, Indiana, Maryland, Rice, Rochester and Syracuse. The common support of separate compilers has three important immediate benefits. Firstly each compiler will get better functionality as it is usis the pooled capability of many, previously separate, compilers. Secondly the constraint that one runtime system will support seven different compilers, makes it likely that most other compiler projects can make use of it. Finally these separate compilers can be used in integrated multilanguage and mulyi system experiments.

The following is brief description of the compiler efforts at different organizations.

#### Cooperating Systems

Cooperating System's compiler, runtime, and application toolkit, **CRAFT**, currently consists of sample Fortran 90 subset compilers for Thinking Machines' Connection Machine series, plus the CLASSIC graphical interface. Using CLASSIC, X windows users can examine the Fortran compiler's specification, rebuild the compiler, run the compiler over sample Fortran 90 codes, and examine the resultant intermediate and object code. The user can graphically interrogate the generated compilers, phase by phase, about the results of parsing, program transformation, and code generation.

The CRAFT Fortran 90 compilers use a single semantic representation (SemRep) as an intermediate notation between the front end and code generation. The front end produces SemRep, the transformative phase applies black-box pattern-matching transformations over SemRep, and the back end generates code for a hierarchy of processors by partitioning and lowering transformed SemRep code.

The runtime system targeted by the initial release of the CRAFT compilers is Thinking Machine's own CMRT runtime system for the CM/2 and CM/5. Under the proposed effort, CSC would benefit by helping to construct and patching into a standardized runtime interface, mitigating our current dependency on vendor-specific system code.

### Harvard

Harvard is developing a kernel based compiler system that provides a general purpose compiler that has been "unbundled" into a suite of tools. The overall goals that have guided the development of the system are:

- Extensibility: The compiler is suitable for a large class of languages and, in particular, it is expected to be used to implement extensions to existing programming languages.
- Soundness: An important goal is that the several components of the compiler are well founded and that one be able to develop proofs, or at least convincing arguments, of their correctness.
- Efficiency: The generality of the compiler is not gained by compromising the efficiency of the code ultimately produced. Indeed, one of the code generator tools available in the system has produced extremely efficient native code.
- Simplicity: While we do not believe that application programmers will often extend the components of a compiler, we want such extensions to be realistically considered by programmers who understand the structure of the compiler. For the above reason, the compiler design has strived for simplicity in the representations provided and in the various tools.

The compilation is divided into several stages to provide the following ways of executing the program:

- Interpretation which lets the the user interact with the execution.
- Conversion into an intermediate language from which it is quite straightforward to generate code in languages like C and Fortran.
- Code Generation for C, variants of C like C* or MPL, Fortran, and so on.

### Indiana

Indiana University is managing the development of the data parallel component (pC++) of the HPC++ consortium. Though we already have a prototype system that runs on all the target platforms mentioned in this proposal, the major task of integrating pC++ with HPF remains to be done. The common runtime system proposed in this document is critical to achieving that goal. In addition new compiler optimization will be added to the pC++ code generator to take advantage of the optimized data movement and collective communication routines defined in this proposal.

A major component of HPC++ is a task parallel subsystem defined by the caltech CC++ project. The Indiana team will be responsible for integrating the common data parallel runtime system defined here with the dynamic thread model used by CC++.

### Maryland

The Maryland group is developing methods that make it possible to produce portable compilers targeted at scalable architectures that generate efficient code for sparse, irregular, adaptive and block structured computations.

The development of sophisticated runtime support forms the foundation of our research effort. We have developed the CHAOS library, a runtime support library that couples partitioners to compilers, partitions loop iterations, remaps data and index arrays, and generates optimized communication schedules. We have implemented address translation methods that support irregular mappings by using highly optimized page-based translation tables. We have also developed a partitioner library, (PIECES library) containing a wide range of efficient partitioners that are based on data access patterns of a problem, computational load information and problem geometry.

There are two ongoing Fortran compiler efforts to target irregular and block structured problems. Using the CHAOS/PARTI libraries and the Syracuse Fortran 90D compiler we have developed a prototype distributed memory compiler able to generate efficient code for templates extracted from adaptive, irregular and block structured problems. By making use of the Rice D System, we have developed loop slicing methods capable of dealing with loops with unstructured data access patterns, including distributed array accesses with multiple levels of indirection.

### Rice

Rice University has developed a prototype compiler for Fortran 77D targeting the Intel iPSC/860 system and the Thinking Machines CM-5. We are considering other targets as well. To date our efforts have focused on aggressive compile-time optimizations, primarily of the communication on distributed-memory systems. These have been extremely fruitful, even producing code that is better than the CM Fortran compiler for certain problems. However, the compiler is not robust at this time. In addition, there are fundamental limits to compiler analysis in the worst case as well as practical limits.

As part of the parallel compiler runtime consortium, we will extend our techniques in two ways. For programs for which compiler analysis is feasible, we will make optimal use of the common runtime support. This will not require fundamental changes to our compilation strategy, but will allow us to produce portable compiled code (to the extent that the common runtime support is widely implemented). For programs that are not amenable to compiler analysis, we will add compiler transformations to call the runtime support. The compiler will then effectively optimize those complex programs by relying on the runtime implementation. This will directly benefit our work by providing more robust and efficient support for unanalyzable programs.

We will also continue our collaborations with Syracuse University and the University of Maryland to investigate irregular computations. This class of programs relies on excellent runtime support such as the common runtime support described in this proposal.

### Rochester

As one of the groups that has been building parallelizing compilers for *distributed shared* memory machines in the parallel compiler runtime consortium for data parallelism, the Rochester group would contribute to the design of the common runtime system from the distributed shared memory perspective. They will also retarget ROC (the Rochester Optimizing Compiler for distributed shared memory machines) to the common runtime system.

ROC is a parallelizing/optimizing compiler for distributed shared memory multiprocessors such as the KSR1, BBN Butterfly and T3D. It is a continuation of Pnuma [34, 35], a parallelizing Fortran compiler for NUMA (Non-Uniform Memory Access) parallel machines. The compiler takes programs written in Fortran extended with data distribution information as in HPF, and currently generates parallel code for the KSR1 and the BBN Butterfly. Data distribution information is used to drive the loop transformation strategy and code parallelization. Data accesses in the resulting code are local wherever possible, and non-local accesses are performed using block transfers if possible. The system can also produce uniprocessor code optimized for cache locality.

A common runtime system will assist the transfer of the compiler technologies developed in ROC more easily to other machine platforms. The loop transformation framework and locality enhancing algorithms implemented in ROC will be made available to the high performance computing community.

### Syracuse

Syracuse University has developed a prototype source-to-source compiler for Fortran 90D/High Performance Fortran for distributed memory systems. It takes as input a program written in Fortran 90D/HPF and outputs a SPMD (Single Program Multiple Data) program with appropriate data and computation partitioning and communication calls. Along with University of Maryland, we have extended this compiler for supporting unstructured problems.

We have also developed libraries for communication scheduling and graph partitioning which are embedded in this compiler.

Task 9 of this proposal funds the continuation of the base compiler infrastructure development. The current infrastructure will be extended so it can support the common runtime system proposed here, heterogenous machine targets, parallel I/O and irregular adaptive problems.

Adherence to object-oriented engineering concepts and the production of requirement documents for the construction and integration of new runtime modules will combine to promote commercial reuse and extensibility of the common runtime system.

By making specific runtime services available to users through a standard interface, in an industry-standard programming environment (that of C++), the proposed work will lower the barrier to entry of new researchers and product developers into the parallel runtime software market.

The core technologies for these component runtime designs have been maturing in the literature for years. But by finally overcoming the significant software engineering challenges, this work will transform the common runtime effort from a bundle of academic curiosities into a viable research tool for the larger parallel programming community.

Several of the institutions involved have had a tremendous success in transferring the technology developed by projects supported by ARPA and other federal agencies.

Rice and Syracuse have transferred the HPF technology to industry through HPF Forum. Syracuse has a technology alliance with Portland Group Incorporated to transfer technology for a HPF compiler. Rice has a technology alliance with Convex.

The work at Maryland on runtime libraries and compilers for distributed memory parallel machines has been transferred to both industry and government laboratories. The Maryland group has distributed the CHAOS/PARTI libraries to government agencies, including several NASA sites and Los Alamos National Lab. In the commercial world, the Maryland group has worked with compiler groups at Thinking Machines, the Portland Group, and Applied Parallel Research.

Indiana as one of the leaders of HPC++ consortium is involved with a number of technology transfer project that include users in NSF Grand Challenge Groups (the Computational Cosmology Consortium, The Maryland Land Dynamics Group and the University of Colorado Turbulence GC project), National laboratories including Argonne National Labs, Los Alamos and Lawrence Livermore, and commercial vendors including Rogue Wave Software, a supplier of scientific C++ class libraries. Scientific Computing Associates, who are building the Fortran Linda system.

To the best of our knowledge, this effort is the first one which would integrate several runtime systems. Hence, there is essentially little related work. However, significant amount of runtime support has been used by several compiler projects for private memory multiprocessors. These include the Fortran D compiler projects at Rice and Syracuse [43, 41], the Vienna Fortran compiler project [52] at the University of Vienna. the DINO project at Colorado [49], Kathy Yelick's work [42] at Berkeley, Linda programming environment at Yale [40], Paradigm project at University of Illinois [44], the CODE project at UT, Austin, the Kali project [46], Marina Chen's work at Yale [47] and the PARTI project [48, 51, 50]. There are also a number of efforts to develop task-parallel runtime systems for a variety of high-level programming languages, such as CC++ [53], Jade [54], Natasha [55], and Fortran M [56]. In addition, several groups are developing parallelizing compilers that recognize implicit task parallelism in sequential programs [57, 58, 59, 61, 60, 62].

Naturally much of the related research is represented by the base compilers of the collaborating institutions - these are described in Sect III.E 3.8.

| Tasks | Indiana | Maryland | Rice | Syracuse | Harvard | Cooperating Systems | Rochester | Texas |
|---|---|---|---|---|---|---|---|---|
| 1 | L | $\surd$ | $\surd$ | $\surd$ | | $\surd$ | $\surd$ | |
| 2 | $\surd$ | L | $\surd$ | $\surd$ | | $\surd$ | $\surd$ | |
| 3 | | | | L | | | | |
| 4 | $\surd$ | $\surd$ | $\surd$ | L | | $\surd$ | $\surd$ | |
| 5 | | L | | $\surd$ | | $\surd$ | $\surd$ | |
| 6 | L | $\surd$ | $\surd$ | $\surd$ | | $\surd$ | $\surd$ | |
| 7 | | | | | | L | | L |
| 8 | E | E | E | L,E | N | E | E | |
| 9 | | | | L | | | | |

Table 1: Distribution of work: The description of each of these tasks can be found in the technical rationale (task n is described in Sect III.E 3.n)

$\surd$-Member; L - Leader
E- Existing Compilers; N - New Compilers

Table 1 gives a description of the involvement of different groups in different tasks. Each task is related to the corresponding section on technical rationale The leader would coordinate the activities within the group.

Specific work that will be done in the course of this project is as follows:

- **Task 1: Common Code and Data Descriptors for Arrays in HPC Languages:**

  An initial meeting of the people designing HPF and HPC++ systems to come to terms with what already exists in commercial and prototype systems. The goal is to construct a vocabulary and language to characterize the approaches that have been taken so far. The result will be a summary report which can be used in new code/compiler design.

  A Second Meeting will try to establish if a consensus has developed around the standard language for data descriptors. If so, we will build a standard model and design multi-target libraries that support it.

- **Task 2: Address Translation Mechanisms:**

  We will Develop a standard interface for address translation that will be closely tied to the distributed array descriptors, on one distributed memory architecture (e.g. Intel Paragon, TMC CM-5, IBM SP-1). We will also provide runtime support for hybrid regular/irregular data distributions, on one distributed memory architecture and one distributed shared memory architecture (e.g. KSR-1, Cray T3D).

- **Task 3: Common Computational Functions:**

We would develop a library of common computational functions that uses the common code and data descriptors. These functions will be optimized for one distributed memory architecture and one distributed shared memory architecture (e.g. KSR-1, Cray T3D).

- **Task 4: Common Data Movement Routines:**

  We will develop a common library of collective structured and unstructured data movement routines utilizing the distributed array descriptors.

  - Develop structured collective communications routines for one distributed memory architecture (e.g. Paragon, CM-5, SP-1) and one distributed shared memory architecture (e.g. KSR-1, T3D).
  - Develop unstructured collective communications routines for one distributed memory architecture (e.g. Paragon, CM-5, SP-1) and one distributed shared memory architecture (e.g. KSR-1,

  Experimentation and experience gained from development of these routines would be used in the development of the Common Compiler Data Movement Interface.

- **Task 5: Common Data Movement deriving and optimizing routines**

  - Provide runtime support for communication optimizations targeted at block structured problems and regular problems in which the number of processors and array dimensions are not bound until runtime, on one distributed memory architecture (e.g. Paragon, CM-5, SP-1) and one distributed shared memory architecture (e.g. KSR-1, T3D).
  - Design optimizations for concurrent irregular communication patterns, including implementation of software caching methods that recognize when multiple copies of the same off-processor data item are being requested, and implement on one distributed memory and one distributed shared memory architecture.
  - Design optimizations for communication characterized by runtime determined dependency patterns. Drawing on information from ongoing research projects in this area, evaluate and compare different approaches to scheduling runtime dependent information. Develop a small set of primitives capable of covering different communication graph characteristics (e.g. tree, grid shaped, directed acyclic graph). Implement for one distributed memory and one distributed shared memory architecture.
  - Standardize the common interface for runtime support employed in partitioning irregular data structures used in concurrent irregular problems. Implement using standard interface in distributed memory and distributed shared memory architecture.
  - Standardize common interface for runtime support employed in partitioning data structures described by runtime data dependencies. Implement using standard interface in distributed memory and distributed shared memory architecture.

- **Task 6: Common Compiler Data Movement Interface Specifications**

  We will establish a CMPI steering committee that will guide the process that will result in the design of an interface. Meetings, in concert with the common code and data descriptors effort, will examine the alternatives and propose a preliminary draft of a common date movement interface specification.

  An initial library will be designed that reflects the decision of the evaluation committee. We will attempt to integrate the result into the current academic HPF and HPC++ compiler projects.

  A final distribution of a set of code generation templates will be completed that demonstrate how one compiler family can adapt to a variety of potential targets.

- **Task 7: Software Engineering**
  - Engineering the Runtime System for Reuse and Extensibility:
    - Understand the commonality and differences of existing approaches for each of the above runtime sub-systems.
    - Sort out the assertions each module must satisfy for its invocation. Structure runtime system modules in terms of functionality and assertions in an object-oriented fashion.
    - Design a coherent set of object-oriented runtime system module interfaces.
    - Retrofitting the modules with the new interfaces.
  - Test and Validation The workplan for test and evaluation has four tasks
    - The extensions which must be made to CODE 2.0 to efficiently support generation of parameterized tests and recording and analysis of test results,
    - development of the test suite and the test data analysis programs,
    - execution and reporting of the test results and
    - extension of the test system to serve as an initial multi- language component development environment.

- **Task 8: Retargeting Current Compilers and Developing New Compiler based on the Runtime Support:**
  - Harvard would implement the several components of the unbundled compiler and provide appropriate documentation of this implementation.
  - Indiana would retarget the current pC++ data parallel component of the HPC++ project to work with the common runtime system and the C++ libraries for adaptive structures designed by CSC.
  - Syracuse would retarget their current compiler to use the common runtime system for supporting adaptive problems and non-uniform computational environments.
  - CSC would retro-fit CSC's F90/HPF compiler with the array subset of the common runtime system. It would also develop new C++ library classes for adaptive trees structures using the common runtime system.

  - Rochester will also retarget ROC (the Rochester Optimizing Compiler for distributed shared memory machines) to the common runtime system.

- **Task 9: Base Compiler at Syracuse** The Syracuse group will continue work on the base HPF (Fortran 90D) compiler developed with ARPA funding from 1991-94. This base compiler infrastructure is designed so that it can be used to experiment with new capabilities in areas such as parallel I/O, heterogenous computing, advanced optimizations and irregular adaptive problems. In this fashion it can tast new language and compiler concepts which are possible enhancemants for the future HPF language and its implemantations. This task directly supports the Syracuse work in tasks 1-8.

**Geoffrey Fox**

Geoffrey Fox's expertise encompasses both software and applications, and his research program is built around the linkage of these areas. This is illustrated by the strong Application focus of the Fortran 90D/HPF project. Fox established and directs the New York State founded program ACTION, which is focused on transfer of HPCC technology to the state's industry. ACTION has excellent access to industry in both application and technology areas. Fox has written several articles on the interdisciplinary field of computational science and established Masters and Ph.D. programs in this area at both Caltech and Syracuse.

Fox earned his Ph.D. in theoretical physics from Cambridge University in 1967. His research experience includes work at the Institute for Advanced Study at Princeton; Lawrence Berkeley Laboratory; Cavendish Laboratory at Cambridge; Brookhaven National Laboratory; and Argonne National Laboratory. He has served as Dean for Educational Computing and Associate Provost for Computing at Caltech. Fox is currently professor of Computer Science and Physics at Syracuse University and Director of the Northeast Parallel Architectures Center.

**Sanjay Ranka**

Sanjay Ranka's main research areas include design and analysis of parallel algorithms, models of parallel computation, compilers and software environments for parallel machines, and neural networks. He has published over 90 journal and conference papers in the above areas. He played a leading role in the development of Fortran 90D/HPF compiler and its runtime support. He serves as a subject area editor (Algorithms and Scientific Computing) for Journal of Parallel and Distributed Computing.

Ranka is an Associate Professor in the School of Computer Science at Syracuse University, where he has been on the faculty since 1988. He received his B. Tech in Computer Science and Engineering from Indian Institute of Technology, Kanpur in 1985 and Ph.D. in Computer Science from the University of Minnesota, Minneapolis in 1988.

**Jim Browne**

J.C. Browne is Professor of Computer Science and Physics and holds the Regents Chair # 2 in Computer Sciences at The University of Texas at Austin. Browne's qualifications for executing the system level test and validation portion of the CRTS project are: the decade of work on multi- language, multi-execution environment parallel programming leading to the current CODE 2.0 programming environment which is an advanced outpost from which to start system level quality assurance for the CRTS and twenty years of development work on commercial software with demonstrated high standards for product quality. Browne has published approximately 100 papers in computational physics and 150 papers in Computer Science. He has given approximately 200 invited lectures to most major Computer Science and Physics Universities in the United States.

**Marina Chen**

Marina C. Chen received her Ph.D. degree in Computer Science from California Institute of Technology in 1983, and her B.S. degree in Electrical Engineering from National Taiwan University in 1978. She is the President of Cooperating Systems Corporation specializing in high performance computing software since July 1993. Before that, she has been an Associate Professor in the Systems and Programming Languages Group at the Department of Computer

Science, Yale University, New Haven, CT, where she began as an Assistant Professor in 1983. Her research interests include parallel processing, high performance computing software, compiler construction, and formal methods for software development.

**Tom Cheatham**

Thomas E. Cheatham, Jr. received the B. S. and M. S. degree in mathematics from Purdue University, Lafayette, IN, in 1951 and 1963, respectively.

Since 1969 he has been Gordon McKay Professor of Computer Science and Director of the Center for Research in Computing Technology at Harvard University, Cambridge, MA. As of 1981 he has been Chairman of the Board of Software Options, Inc., Cambridge, MA. His current research interests include symbolic evaluation of programs, mechanical theorem proving for program verification, the design of wide spectrum programming languages, and the construction of systems for program development and maintenance.

Professor Cheatham is a Fellow of the American Academy of Arts and Sciences and a member of Sigma Xi and the Association for Computing Machinery.

**Alok N. Choudhary**

Alok Choudhary's main research interests are in parallel and distributed processing, software development environments for parallel computers including compilers and runtime support, parallel computer architectures, and parallel I/O. He has published over 60 journal and conference papers in the above areas. Choudhary played a leading role in the development of Fortran 90D/HPF compiler and runtime support for distributed memory machines.

Alok Choudhary received his Ph.D. from University of Illinois, Urbana-Champaign, in Electrical and Computer Engineering, in 1989, M.S. from University of Massachusetts, Amherst, in 1986. He joined the department of Electrical and Computer Engineering at Syracuse University in 1989, where he is currently an associate professor. He received the NSF Young Investigator award in 1993 for research in compiler and runtime optimizations for distributed memory machines.

**Amr Fahmy**

Amr Fahmy was born May 25th 1958 in Alexandria Egypt, he received his B.Sc. in Mathematics from The University of Alexandria in 1980, MA. and Ph.D. in Computer Science from Duke University in 1984 and 1989 respectively. He is currently a Research Associate at Aiken Computation Laboratory at Harvard University conducting research on optimizing compilers for parallel programming languages. His research interests lie in the areas of compilers, parallel programming and machine learning.

**Dennis Gannon**

Dennis Gannon received his Ph.D. in Mathematics in 1976 from the University of California, Davis and his Ph.D. in Computer Science from the University of Illinois in 1980. He served on the faculty of computer science at Purdue University from 1980 to 1985 and from 1985 to date at Indiana University where he is an Associate Professor of Computer Science and Research Director for the Center for Innovative Computer Applications. He has spent the last 14 years working on the problems associated with software and algorithms for massively parallel systems. He is currently director of the ARPA HPC++/pC++ project involving Indiana University, the University of Oregon and the Oregon Graduate Institute.

**Tomasz Haupt**

Tom Haupt's research is concentrated on developing parallel programming methodology. Currently, he is the project leader of the Fortran 90D group, the project to develop a new language for parallel and distributed computer systems, to develop HPF/Fortran 90D compiler, the compiler run-time support, and a benchmark suite. He is also involved in developing applications under AVS for distributed, heterogeneous systems. Haupt is a member of the High Performance Fortran Forum, MPI committee, as well as the International Parallel Benchmark Working Group, PARKBENCH.

Haupt got his Ph.D. in Physics from Institute of Nuclear Physics in Krakow, Poland, in 1985. From 1985 to 1990 he did research in experimental high energy physics and computation physics at INP in Krakow, and NIKHEF-H, Nijmegen, the Netherlands. From 1990 he has been at Syracuse University, where experience working with Geoffrey Fox resulted in changing his profile from an experimental and computational physicist to a computer scientist.

**Kenneth Kennedy**

Ken Kennedy conducts research on optimizing compilers, scientific programming environments, parallel computation, and automatic detection of parallelism. His current research focus is the development of compiler and programming environment technology that will make it possible to support machine-independent parallel programming. To this end, he is involved in three major research projects: (1) the development of a prototype parallel programming environment, called the D system, that includes program development, debugging, and performance tuning tools; (2) the development of a prototype compiler that will translate shared-memory Fortran programs, augmented with data distribution specifications, to efficient programs on distributed memory multicomputers; and (3) a project to develop techniques for automatically managing the memory hierarchy and improve the performance for the new generation of superscalar microprocessors.

Kennedy received his Ph.D. in Computer Science from New York University in 1971. He currently occupies a Noah Harding Professorship in the Department of Computer Science at Rice University and serves as Scientific Director of the Center for Research on Parallel Computation (CRPC). He was elected to the National Academy of Engineering in 1990.

**Charles Koelbel**

Chuck Koelbel's research interests center on mapping algorithms and programs onto distributed memory multiprocessors. His doctoral dissertation presented a compiler which translated Kali, a shared-memory imperative language, for execution on the Intel iPSC/2. A unique feature of this work was a unified treatment of regular computations, for which the communications can be generated at compile time, and unstructured computations, for which run-time support is necessary. Both types of support are necessary for a truly general system. Since coming to Rice he has worked closely with Ken Kennedy and others to apply his methods to the Fortran D compiler project.

He is also studying new techniques for implementing adaptive and dynamically varying data structures on distributed memory machines. These data structures are needed in many areas, including sparse matrix calculations which have applications in finite element methods and linear programming. In addition to his research responsibilities, Dr. Koelbel is serving as executive director of the High Performance Fortran Forum, a group of academic, industrial, and government organizations defining extensions to the Fortran language for use on scalable parallel machines.

**Wei Li**

Wei Li received his Ph.D in computer science from Cornell University in 1993. He has been an assistant professor in computer science at University of Rochester from 1993. His research interests include compiling for parallel architectures with memory hierarchies, programming languages and computer architectures. He worked on compiling for NUMA machines, and developed a loop transformation theory and optimization algorithms for improving data locality. Currently, he is working on ROC (Rochester Optimizing Compiler), a parallelizing/optimizing compiler for distributed shared memory machines such as the KSR1, T3D and BBN Butterfly.

**Joel Saltz** Joel Saltz is Associate Professor of Computer Science at the University of Maryland, College Park and is the director of the High Performance Systems Software Laboratory. Saltz leads a research group at the University of Maryland College Park whose goal is to develop methods that make it possible to produce portable compilers that generate efficient multiprocessor code for irregular scientific problems, i.e. problems that are unstructured, sparse, adaptive or block structured. The target languages are Fortran and C++.

The Maryland group works closely with applications scientists and engineers whose problem areas include computational fluid dynamics, computational chemistry, computational biology, structural mechanics and electrical power grid calculations. The irregular problem optimization techniques developed by this group have already been widely adopted in research compilers; a number of vendors are currently developing new compilers which will make use of these methods.

**Alan Sussman**

Alan Sussman is a Research Associate in the Department of Computer Science at the University of Maryland, College Park. He has been a member of the High Performance Systems Software Laboratory, directed by Joel Saltz, since December 1992. Prior to that, he was a staff scientist at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center. He received his Ph.D. in Computer Science from Carnegie Mellon University in 1991. He has worked on run-time libraries and compilers for distributed memory parallel machines, with the main goal of providing software tools for applications scientists and engineers to effectively utilize such machines. His other research interests include parallel computer architectures and parallel supercomputer applications.

| Tasks | Year I | Year II | Year III |
|---|---|---|---|
| 1. Common Code and Data Descriptors | 1 | 0.5 | 0.5 |
| 2. Address Translation Mechanisms | 2 | 2 | 2 |
| 3. Common Computational Functions | 3 | 2 | 2 |
| 4. Common Data Movement Routines | 2 | 3 | 2 |
| 5. Common Data Movement deriving and optimizing routines | 4 | 4 | 4 |
| 6. Common Compiler Data Movement Interface Specification | 2 | 1 | 1 |
| 7. Software Engineering Issues | 3 | 3 | 3 |
| 8. Retargeting Current Compilers and Developing New Compilers | 1 | 3 | 3 |
| 9. Base Compiler at Syracuse | 4 | 4 | 4 |
| Total | 22 | 22.5 | 22.5 |

Table 2: Relative units of work required (Year I is Sept 1, 94 to Aug 31, 95 etc.)

## Schedule

The group has broken up each project into work units (each around $50K per year). The budget for each organization was based on the break up of table 2 and the task responsibilities described in Sect III.H. The tasks are interrelated with a dependency shown in fig.3 which is at crude resolution the time line for each task – year I is at the top and year III at the bottom of fig.3.

Because of the central nature to much of the run time system of the common code and data descriptors, Task 1 must be completed in the first year of the project. Initial candidate proposals for the common model will be collected in the first month. A working group of the compiler implementors will need to meet (by video conference) weekly until a draft plan is adopted. Initial implementations on each of the base platforms will be designed and implemented by the end of year 1. In the following years, the working group will meet as needed to discuss extensions to structures other than arrays and to provide additional functionality required by other aspects of the runtime system.

Task 2, 3, 4, and 5 would be based on prototype runtime support developed for the compiler projects at Harvard, Indiana, Maryland, Rice, Rochester, Syracuse. These runtime support prototypes exist in several disjoint libraries; these libraries would be expanded, merged, and designed so that they share the same data access descriptors.

Task 6 will organize an on-going effort to define and document communication library primitives for use at the compiler level. A working group will be established to define these functions to make sure they are well documented and they can be used in the same environment without interfering with each other.

The software engineering aspect (Task 7) of the common runtime system effort will be an on-going process while we perform the integration and reorganization based on an existing repertoire of runtime modules. Development of new modules should follow the design principles and system organization. Cooperating Systems and University of Texas are leading this effort.

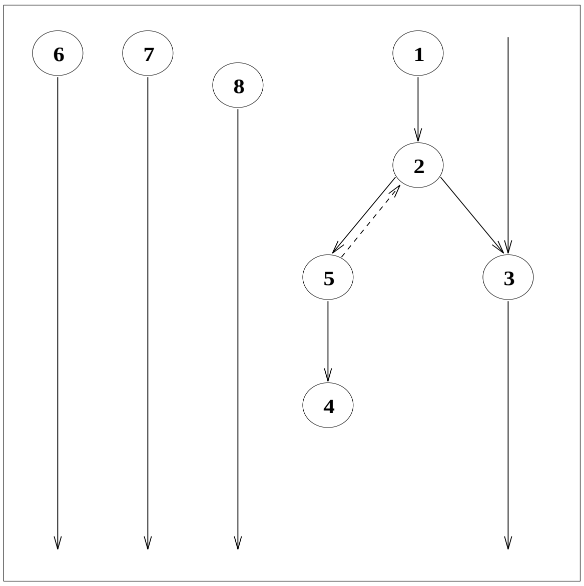


Figure 3: Time dependencies between different tasks 1-8

Solid Arrow refers to a direct dependency
Dashed Arrow refers to feedback

All of the above would be incorporated at different stages into the compiler efforts at Cooperating Systems, Harvard, Indiana, Maryland, Rice, Rochester, and Syracuse (Task 8). Figure 3 gives the relationship between different tasks (1 through 8).

Table 2 gives an assessment of the relative amount of work required for the different tasks during the different phases of the project.

Task 9 involves developing the base compiler infrastructure at Syracuse University. This continues current ARPA funded work which developed the first experimental HPF compiler. This task includes modifying compiler to accomodate new capabilities which are candidates for inclusion in HPF updates. We have identified the areas of irregular problems, parallel I/O and heterogeneous computing as having particular importance. The schedule of this work is defined implicitly so that the compiler has the functionality to test these new ideas.

# References

[1] High Performance Fortran Forum. High performance fortran language specification version 1.0. *Draft, Also available as technical report CRPC-TR92225 from the Center for Research on Parallel Computation, Rice University.*, Jan. 1993.

[2] Gagan Agrawal, Alan Sussman, and Joel Saltz. Compiler and runtime support for structured and block structured applications. Technical Report CS-TR-3080 and UMIACS-TR-93-45, University of Maryland, Department of Computer Science and UMIACS, April 1993. To appear in Supercomputing '93.

[3] Proceedings, Workshop and Conference on Grand Challenges Applications and Software Technology, May 4-7 1993, Pittsburgh, Pennsylvania.

[4] Ishfaq Ahmad, Alok Choudhary, Geoffrey Fox, Kanchana Parasuram, Ravi Ponnusamy, Sanjay Ranka, and Rajeev Thakur. Implementation and Scalability of Fortran 90D Intrinsic Functions on Distributed Memory Machines. Technical Report SCCS-256, Northeast Parallel Architectures Center at Syracuse University, March 1992.

[5] Harry Berryman, Joel Saltz, and Jeffrey Scroggs. Execution time support for adaptive scientific algorithms on distributed memory machines. *Concurrency: Practice and Experience*, 3(3):pp. 159–178, June 1991.

[6] Sandeep Bhatt, Marina Chen, James Cowie, Cheng-Yee Lin, and Pangfeng Liu. Object-Oriented Support for Adaptive Methods on Parallel Machines. *OONSKI'93 Object Oriented Numerics Conference*, Sunriver, Oregon, April 25-27, 1993.

[7] S. Bhatt, M. Chen, C.Y. Lin, and P. Liu. Abstractions for Parallel N-body Simulations. *Proceedings of Scalable High Performance Computing Conference (SHPCC '92)*, Williamsburg, Virginia, April 1992.

[8] Zeki Bozkus, Alok Choudhary, Geoffrey Fox, Tomasz Haupt, Sanjay Ranka, and Jhy-Chun Wang. Message Passing Environment Requirements for the Fortran 90D Compiler. Technical Report SCCS-FEB-93, Northeast Parallel Architectures Center at Syracuse University, February 1993.

[9] M. Chen and J. Cowie. Prototyping Fortran-90 Compilers for Massively Parallel Machines. *ACM Symposium on Programming Language Design and Implementation (PLDI)*, June 1992, San Fransisco, California.

[10] M. Chen and J. Wu. Optimizing Fortran-90 Programs for Data Motion on Massively Parallel Systems. Technical Report TR-882, Yale University, 1991.

[11] R. Das, R. Ponnuwamy, J. Saltz, and D. Mavriplis. Distributed memory compiler methods for irregular problems - data copy reuse and runtime partitioning. J. Saltz and P. Mehrotra, editors, *Languages, Compilers and Runtime Environments for Distributed Memory Machines*, pp. 185–220. Elsevier, 1992.

[12] Raja Das, Mustafa Uysal, Joel Saltz, and Yuan-Shin Hwang. Communication Optimizations for Irregular Scientific Computations on Distributed Memory Architectures. Submitted to *Journal of Parallel and Distributed Computing*, June 1993.

[13] J. Dongarra, R. van de Geijn, and D. Walker. A Look at Scalable Dense Linear Algebra Libraries. *Proceedings of Scalable High Performance Computing Conference (SHPCC-92)*, Williamsburg, Virginia, April 1992.

[14] Max Lemke and Daniel Quinlan. P++, A Parallel C++ Array Class Library for Architecture-Independent Development of Structured Grid Applications. *ACM SIGPLAN Notices*, 28(1):pp. 21–23, September 1992.

[15] Jingke Li and Marina Chen. Compiling communication-Efficient Programs for Massively Parallel Machines. *IEEE Transaction on Parallel and Distributed Systems*, pp. 361-376, July 1991.

[16] Lu and Marina Chen. Parallelizing loops with indirect array references or pointers. *the 4th Workshop on Programming Languages and Compilers for Parallel Computing*, Santa Clara, California, August, 1991. In *Programming Languages and Compilers for Parallel Computing*, Research Monographs in Parallel and Distributed Computing series, MIT Press.

[17] Ravi Ponnusamy, Joel Saltz, and Alok Choudhary. Runtime-compilation techniques for data partitioning and communication schedule reuse. Technical Report CS-TR-3055 and UMIACS-TR-93-32, University of Maryland, Department of Computer Science and UMIACS, April 1993. To appear in Supercomputing '93.

[18] S. Ranka and J.C. Wang. Static and runtime scheduling of unstructured communication. *International Journal of Computing Systems in Engineering*, 1993. To appear.

[19] S. Ranka, J.C. Wang and G. Fox. Static and runtime algorithms for all-to-many personalized communications on permutation networks. *Proceedings of the 1992 International Conference on Parallel and Distributed Systems*, pp. 211–218, HsinChu, Taiwan, December 1992.

[20] S. Ranka, J.C. Wang, and M. Kumar. Personalized Communication Avoiding Node Contention on Distributed Memory Systems. *Proceedings of the 1993 International Conference on Parallel Processing*, St. Charles, IL, August 1993. To appear.

[21] Joel H. Saltz, Ravi Mirchandaney, and Kay Crowley. Run-time parallelization and scheduling of loops. *IEEE Transactions on Computers*, 40(5):pp. 603–612, May 1991.

[22] A. Sussman, J. Saltz, R. Das, S. Gupta, D. Mavriplis, R. Ponnusamy, and K. Crowley. PARTI primitives for unstructured and block structured problems. *Computing Systems in Engineering*, 3(1-4):pp. 73–86, 1992. Papers presented at *the Symposium on High-Performance Computing for Flight Vehicles*, December 1992.

[23] S. Venugopal, V.K. Naik, and J. Saltz. Performance of distributed sparse Cholesky factorization with pre-scheduling. *Proceedings Supercomputing '92*, pp. 52–61. IEEE Computer Society Press, November 1992.

[24] J. K. Lee and D. Gannon, *Object Oriented Parallel Programming: Experiments and Results* Proceedings of Supercomputing 91 (Albuquerque, Nov.), IEEE Computer Society and ACM SIGARCH, 1991, pp. 273–282.

[25] D. Gannon, *Libraries and Tools for Object Parallel Programming*, Proc. CNRS-NSF Workshop on Environments and Tools For Parallel Scientific Computing, Saint Hilaire du Touvet, France, Elsevier Science Publisher, Advances in Parallel Computing, Vol. 6, pp. 231-246, 1993.

[26] F. Bodin, P. Beckman, D. Gannon, S. Yang, S. Kesavan, A. Malony, B. Mohr, *Implementing a Parallel C++ Runtime System for Scalable Parallel Systems*, Proceedings of the 1993 Supercomputing Conference, Nov. 1993.

[27] D. Gannon, F. Bodin, P. Beckman, S. Yang, *Distributed pC++: Basic Ideas for an Object Parallel Language*, Proeedings of the First Annual Object-Oriented Numerics Conference, Sunriver, Oregon, Apr. 1993, pp. 1-24. Also to appear in March 1994, Special Issue of Scientific Programming.

[28] A. Malony, B. Mohr, F. Bodin, P. Beckman, D. Gannon, S. Yang, *Performance Analysis of pC++: A Portable Data-Parallel Programming System for Scalable Parallel Computers*, Proceedings of the 1994 IEEE Internatio nal Parallel Processing Symposium, Mexico, April, 1994.

[29] U. Banerjee. Unimodular transformations of double loops. In *Proceedings of the Workshop on Advances in Languages and Compilers for Parallel Processing*, pages 192–219, August 1990.

[30] H. Berryman, J. Saltz, and J Scroggs. Execution time support for adaptive scientific algorithms on distributed memory machines. *Concurrency: Practice and Experience*, 3(3):159–178, June 1991.

[31] S. Bhatt, M. Chen, J. Cowie, C. Lin, and P. Liu. Object-oriented support for adaptive methods on parallel machines. In *OONSKI'93 Object Oriented Numerics Conference*, April 1993.

[32] Z. Bozkus, A. Choudhary, G. Fox, T. Haupt, S. Ranka, and J. Wang. Message passing environment requirements for the Fortran 90D compiler. Technical Report Technical Report SCCS-FEB-93, NPAC at Syracuse University, February 1993.

[33] S. Hiranandani, K. Kennedy, and C. Tseng. Compiler optimizations for FORTRAN-D on MIMD distributed-memory machines. Technical Report TR91-156, Rice University, April 1991.

[34] W. Li. *Compiling for NUMA Parallel Machines.* PhD thesis, Dept. of Computer Science, Cornell University, August 1993.

[35] W. Li and K. Pingali. Access Normalization: Loop restructuring for NUMA compilers. In *Proc. 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1992.

[36] A. Rogers and K. Pingali. Process decomposition through locality of reference. In *Proc. of the 1989 SIGPLAN Conference on Programming Language Design and Implementation*, 1989.

[37] M. Wolf and M. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, October 1991.

[38] P. Newton and J Browne "The CODE 2.0 Graphical Parallel Programming Environment" Proceedings of ACM Intl. Conf. on Supercomputing (Washington, D.C., July 1992), pp. 167-177.

[39] S. Hyder and J Browne " A Unified Approach to Parallel Debugging" Proceedings of the 1993 Intl. Conf. on Parallel Processing ( Chicago, IL, August 1993)

[40] S. Ahuja, N. Carriero, and D. Gelernter. Linda and friends. *IEEE Computer*, pages 26–34, August 1986.

[41] Zeki Bozkus, Alok Choudhary, Geoffrey Fox, Tomasz Haupt, Sanjay Ranka, and Min-You Wu. Compiling Fortran 90D/HPF for distributed memory MIMD computers. To appear in the Journal of Parallel and Distributed Computing, March 1993.

[42] Soumen Chakrabarti and Katherine Yelick. Implementing an irregular application on a distributed memory multiprocessor. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP)*, May 1993. ACM SIGPLAN Notices, Vol. 28, No. 7.

[43] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu. Fortran D language specification. Department of Computer Science Rice COMP TR90-141, Rice University, December 1990.

[44] M. Gupta and P. Banerjee. Demonstration of automatic data partitioning techniques for parallelizing compilers on multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):179–193, March 1992.

[45] S. Hiranandani, K. Kennedy, and C. Tseng. Compiler support for machine-independent parallel programming in Fortran D. In *Compilers and Runtime Software for Scalable Multiprocessors, J. Saltz and P. Mehrotra Editors*, Amsterdam, The Netherlands, To appear 1991. Elsevier.

[46] C. Koelbel, P. Mehrotra, and J. Van Rosendale. Supporting shared data structures on distributed memory architectures. In *2nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 177–186. ACM, March 1990.

[47] L. C. Lu and M.C. Chen. Parallelizing loops with indirect array references or pointers. In *Proceedings of the Fourth Workshop on Languages and Compilers for Parallel Computing*, Santa Clara, CA, August 1991.

[48] R. Mirchandaney, J. H. Saltz, R. M. Smith, D. M. Nicol, and Kay Crowley. Principles of runtime support for parallel processors. In *Proceedings of the 1988 ACM International Conference on Supercomputing*, pages 140–152, July 1988.

[49] Matthew Rosing, Robert B. Schnabel, and Robert P. Weaver. The DINO parallel programming language. *Journal of Parallel and Distributed Computing*, 13(1):30–42, September 1991.

[50] J. Saltz, H. Berryman, and J. Wu. Runtime compilation for multiprocessors. *Concurrency: Practice and Experience*, 3(6):573–592, 1991.

[51] J. Wu, J. Saltz, S. Hiranandani, and H. Berryman. Runtime compilation methods for multicomputers. In *Proceedings of the 1991 International Conference on Parallel Processing*, volume 2, pages 26–30, 1991.

[52] H. Zima, P. Brezany, B. Chapman, P. Mehrotra, and A. Schwald. Vienna Fortran – a language specification. Report ACPC-TR92-4, Austrian Center for Parallel Computation, University of Vienna, Vienna, Austria, 1992.

[53] K. M. Chandy and C. Kesselman. Compositional C++: Compositional Parallel Programming. California Instititue of Technology, 1992.

[54] M. C. Rinard, D. J. Scales, and M. S. Lam. Jade: A High-Level Machine-Independent Language for Parallel Programming. *Computer*, 26(6):28-38, June 1993.

[55] L. A. Crowl and T. J. LeBlanc. Control Abstraction in Parallel Programming Languages. In *Proceedings of the International Conference on Computer Languages*, pages 44-53, Oakland, CA, April 1992.

[56] I. T. Foster and K. M. Chandy. Fortran M: A Language for Modular Parallel Programming. Preprint MCS-P327-0992, Argonne National Laboratory, June 1992.

[57] S. P. Amarasinghe and M. S. Lam. Communication Optimization and Code Generation for Distributed Memory Machines. In *Proceedings of the SIGPLAN '93 Conference on Programming Language Design and Implementation*, Albuquerque, NM, 21-25 June 1993.

[58] J. M. Anderson and M. S. Lam. Global Optimizations for Parallelism and Locality on Scalable Parallel Machines. In *Proceedings of the SIGPLAN '93 Conference on Programming Language Design and Implementation*, Albuquerque, NM, 21-25 June 1993.

[59] C. D. Polychronopoulos and others. Parafrase-2: A Multilingual Compiler for Optimizing, Partitioning, and Scheduling Ordinary Programs. In *Proceedings of the 1989 International Conference on Parallel Processing*, St. Charles, IL, August 1989.

[60] W. Li and K. Pingali. Access Normalization: Loop Restructuring for NUMA Compilers. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 285-295, Boston, MA, 12-15 October 1992.

[61] R. Eigenmann, J. Hoeflinger, G. Jaxon, Z. Li, and D. Padua. Restructuring Fortran Programs for Cedar. *Proceedings of the 1991 International Conference on Parallel Processing*, I, Architecture:57-66, August 1991.

[62] D. Padua and R. Eigenmann. Polaris: A New Generation Parallelizing Compiler for MPPs. Technical Report 1306, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1993.