Common Runtime Support for High Performance Languages

Interim Final Report

November 30, 1997

Prepared By

Geoffrey C. Fox
Northeast Parallel Architectures Center
at Syracuse University
111 College Place
Syracuse, NY 13244-4100

Prepared For

ESC/ENS
5 Eglin Street, Building 1704
Hanscom AFB, MA 01731-2116
Attn.: Mr. Harry Koch

Sponsored By

Advanced Research Projects Office/CSTO
Common Runtime Support for High Performance Parallel Languages
ARPA Order No. B526
Issued by ESC/ENS under contract
F19628-94-C-0057

Contents

1 Introduction

Aims

Widespread adoption of parallel computing depends on the availability of improved software environments. An essential component of these environments will be high-level languages. Several languages for exploiting data-parallelism (or task-parallelism) have been developed, or are under development. The stated goal of this project has been to provide a public domain infrastructure for runtime support of these high-level languages. The targetted languages include parallel versions of Fortran and C++, but our intention has been to provide uniform runtime support for many source languages[33].

The active participants in the project include

Syracuse University, coordinating the project and responsible for development of a major contribution to the runtime (emphasisizing regular communication patterns and collective computation operations) and demonstration of an experimental HPF translator exploiting the runtime.

University of Maryland, responsible for runtime support, especially for irregular array mappings and communication patterns, and authors of the METACHAOS package for interfacing between the various runtime components developed in the project.

Indiana University, contributing the Tulip runtime of pC++, the data-parallel component of the HPC++ consortium.

University of Rochester, investigating the utility of the new runtime systems for distributed shared-memory architectures.

University of Texas, Austin, developing the DAGH system for adaptive grid problems.

Cooperating Systems, providing help in software quality assurance and documentation.

University of Florida, providing advanced algorithms for collective computations operations such as the combining scatter operation.

Rice University, studying compiler requirements.

The original proposal identified eight major areas of development: common code and data descriptors, address translation mechanisms, common computational functions, common data movement routines, data movement derivation and optimization, compiler data movement interface (interpretted as a compiler-level "MPI" interface), software engineering issues, and retargetting compilers and development of new compilers using the new runtime software.

Achievements

Most of the original tasks have been addressed in the course of the project. In a some cases developments in the field or within the participating institutions over the past three years have shifted the emphasis of tasks. These issues will be discussed in more detail in section 2. Within the terms of the original proposal, major achievements of the project include

- The development at Syracuse of a new *kernel runtime*, built on MPI, with interfaces for several different languages and compilers.
- The development at Maryland of the METACHAOS system, which enables interoperability between the runtime contributions of the various partners.
- The development at Syracuse of an infra-structure for construction of compilers based on the new runtime. In particular a new subset HPF translator has been constructed in the framework.

Due to changes in focus of the collaborating research groups, the original statement about developing seven compilers sharing one runtime was modified. A revised statement of work was submitted earlier this year. An unforseeable influence on the development of the project was the emergence of Java. In May, 1996 the participants of the project issued a draft white paper on the implications of Java for HPCC [34]. Subsequently Syracuse organized a series of workshops [39, 40, 43] on the theme of Java for Computational Science and Engineering. Many of the subcontractors reports in this document reflect the shift towards Java as a primary focus of investigation. At least three of the partners are now engaged in development of parallel or optimizing Java compilers.

This document

Section 2 reviews the progess of the project in relation to the goals in the original proposal and the revised statement of work submitted in July this year. Sections 3 to 10 summarize progress within the individual institutions. The appendix lists software deliverables and publications relating to the project.

2 Project Overview

The next eight subsections review progress and achievements within the task areas identified in the original proposal.

2.1 Common Data Descriptors

An early output of the project was the definition of a common Distributed Array Descriptor (DAD) for languages and compilers dealing with HPF-like distributed arrays[18]. This DAD is a simple, flat table of parameters that describe the mapping from global index space to the local array space in each SPMD node program. The DAD structure can be made visible to (for example) C and Fortran SPMD programs.

The tabular DAD of [18] had a less direct influence on the development of our libraries and compilers than originally anticipated. The adoption of object-oriented languages and design principles favoured treatment of the array descriptor as an opaque object, accessed through suitable methods and inquiries. The tabular version of the DAD survives as a useful concrete model of the more abstract implementations adopted in practice, but the NPAC runtime kernel (for example) implements its array descriptor as a true C++ object. Fortran or C programs access its fields through suitable inquiry functions.

The descriptor implemented in the NPAC kernel handles a generalization of the HPF 1.0 distributed array model. It can describe arrays of any rank, any data distribution format supported by HPF 1.0 (plus distribution formats of some earlier data parallel languages such as `C*`), and any of the alignment options of HPF. The descriptor also handles Fortran 90-style any regular section of an array. Within certain limits, new distribution formats can be added by extending the class hierarchy that describes the layout of individual array dimensions. For example, support for HPF 2.0-like ghost regions has been added in this manner, and before the end of the extended project we expect to add support for at least the block-irregular distribution format of HPF 2.0.

2.2 Address translation mechanisms

Both the CHAOS system of Maryland and the PCRC kernel of Syracuse provide address translation mechanisms for converting between global subscripts and local, SPMD offsets and process coordinates. In CHAOS the emphasis is on irregular data distributions whereas the Syracuse software is mainly concerned with HPF-compliant, regular data distributions.

2.3 Common Computational Functions

The body of software developed over the course of the project includes support for all Fortran 90 array transformational functions (including matrix multiplications and transposes, shift and spread operations, and all F90 scalar and array-valued reduction functions) plus a subset of the HPF standard library. These functions operate uniformly on *any* array representable by the DAD described above [19].

The University of Florida subcontractors have developed an MPI version of their algorithms for combining scatter. This version is link-compatible with the PCRC kernel developed at Syracuse. Syracuse has provided interface code so that these routines are now callable using the generic DAD array descriptors employed by the kernel library [65, 66, 3, 6].

Work continues on implementing the full HPF specification for parallel prefix and sorting operations (these are the only significant functions from the standard HPF library not implemented in the NPAC kernel library).

2.4 Common Data Movement Routines

Syracuse group has implemented an extensive set of collective data movement routines in their runtime [19, 74]. The communication operations supported include HPF/F90 array intrinsic operations such as `CSHIFT`, the function `pcrc_write_halo`, which updates ghost areas of a distributed array, the function `remap`, which is equivalent to a Fortran 90 array assignment between a conforming pair of sections of two arbitrarily distributed HPF arrays, and various gather- and scatter- type operations allowing irregular patterns of data access (from regularly distributed arrays). The CHAOS library provides analogous functions. Once again the two libraries are complementary. The Maryland contribution emphasizes irregular data distributions, whereas the functions of the Syracuse library are optimized to operate uniformly on multi-dimensional data arrays with arbitrary HPF-supported distributions.

2.5 Deriving and optimizing data movement

Both the Maryland and Syracuse libraries are based on runtime-computed, reusable communication schedules. To a greater or lesser extent the runtime systems sort and coalesce messages to minimize and aggregrate communication on the network. In the compiler domain, Xiaoming Li has proposed a scheme based on *shift homomorphism* for analysis of HPF forall statements [51]. This scheme classifies the communication needs of these statements. These ideas have been integrated into the Syracuse HPF translation system, as reported in [74].

2.6 Compiler Message Passing Interface

Widespread acceptance of the MPI standard by vendors, who are providing optimized implementations for their hardware, lead us to the conclude that the case for a high-performance compiler-oriented message-passing or data-movement interface is weaker than originally supposed. Project runtime libraries have been successfully developed on, or ported to, standard MPI. Reference [32] is a survey of the subset of MPI calls used in the various runtime components developed over the project. It suggests that there is only limited consensus amongst runtime developers on what is the most important subset of MPI. Viewed from another perspective, it suggests that the MPI standard itself is an adequate basis for runtime and compiler development.

2.7 Software Engineering

Object Oriented principles

Most of the software delivered by the project follows object-oriented design and implementation principles. For example the NPAC runtime kernel and compiler development infrastructure have been implemented entirely in C++ using object-oriented techniques. The work of Indiana on HPC++ and CORBA, and the emergence of Java as a major focus in the project further illustrate the commitment to object-oriented software design.

Test and Validation

We continue the emphasis on quality of deliverable software. GNU systems and conventions are employed to manage the development process. The runtime software delivered by NPAC includes comprehensive suites of test programs that can be executed automatically when the system is configured and built on a new platform.

We have developed Fortran syntax test suite for testing of Fortran 90 and High Performance Fortran (HPF) compilers. This has been expanded considerably to include exhaustive syntax testing and some semantics testing [20, 73]. We find that our test suite can be used to “debug” commercial HPF compilers. As a result of this extensive testing, our new HPF front end is now stable and has been released to the public. Beyond the front end, we have developed an HPF compiler test program generator: a multi-level HPF code generator for validating HPF compiler transformations. This has been used to evaluate leading commercial compiler limitations (DEC, IBM, PGI).

A suite of more complete HPF applications, suitable for testing compilers, has been made available [64].

2.8 Compilers

We have developed a Common Interoperable Intermediate Representation (CI2R) data format for PCRC compilation systems. Currently this is used by the NPAC HPF and Java language translation tools.

A new, extensively tested HPF front-end (full HPF 1.0) was placed in the public domain this year. This transforms source language code into the CI2R abstract syntax tree symbolic form. Syracuse have developed HPF compiler transformations for communication detection and data distributions. The front-end, transformation system and runtime have been integrated and demonstrated in the Syracuse HPF compilation system [74]. As HPF has evolved to version 2.0, we have extended our HPF front-end to cover HPF 2.0.

The Syracuse common runtime also supports an older subset HPF system [52], has a user-level C++ interface, and is being used as the basis of a new *HPJava* translator.

Rochester [23, 26] and Indiana [9, 10, 13, 12] are also developing Java compilers. Rice is working on Java compiler optimizations [15].

2.9 Java and SPMD programming

A significant new task in the revised statement of work submitted in July was the proposal to provide an infrastructure supporting high-level SPMD programming practice. This will be a valuable by-product of the project. Beyond its role as a target for parallel compilers, our runtime may also be directly used as an infrastructure to support SPMD programming. In particular, application programmers will not have to use `send()` and `receive()` in their code. Instead, they use high level facilities provided by the runtime. We intend to use Java as a vehicle for demonstrating this SPMD programming model. More specifically, user will be able to write Java SPMD programs that call PCRC runtime functions as native methods.

The new Java binding of the NPAC kernel runtime is a first step in this direction. We are also investigating a minimal set of extensions to the Java language to support data-parallel programming using distributed arrays plus explicit calls to PCRC communication functions. A source-to-source preprocessor will convert the extended Java dialect to ordinary Java, adding run-time calls as required for, eg, address translation. This can be seen as a new compiler targetting the run-time—an *HPJava*—although the work of this compiler is much simpler than for HPF, because the programmer will explicitly specify the communication requirements.

For early results of the work on Java see [16, 17, 38, 37, 41].

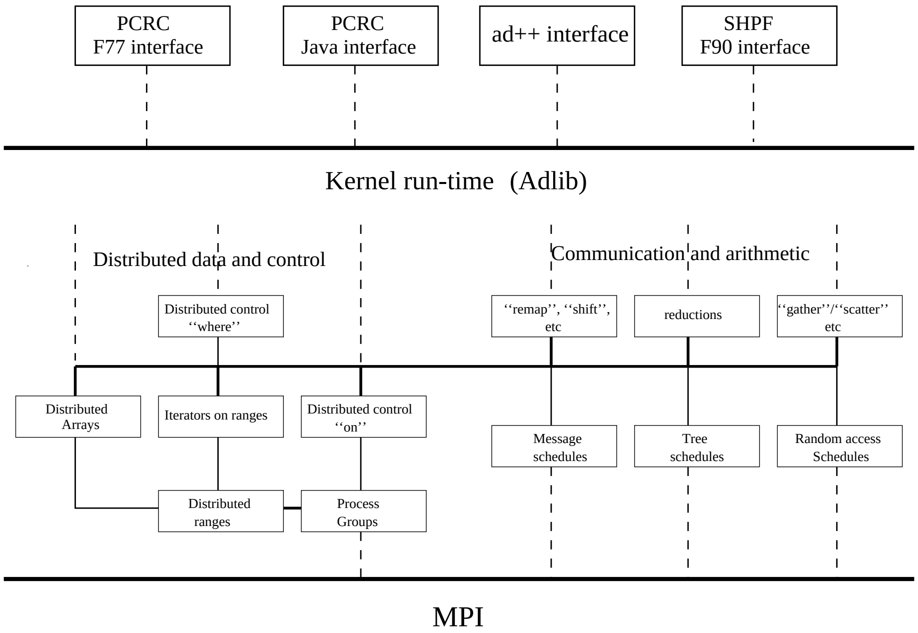

Figure 1: NPAC PCRC runtime architecture

3 Syracuse University Report

NPAC Kernel Runtime

The kernel of NPAC library [19] is a C++ class library. It is primarily descended from the run-time library of an earlier research implementation of HPF[52] with influences from the Fortran 90D [22] run-time and the CHAOS/PARTI libraries. The kernel is currently implemented on top of MPI. The library design is solidly object-oriented, but efficiency is maintained as a primary goal. Inlining is used extensively, and dynamic memory allocation, unnecessary copying, true procedure calls, virtual functions and other forms of indirection are generally avoided unless they have clear organizational or efficiency advantages.

The overall architecture of the library is illustrated in figure 1. At the top level there are several compiler-specific interfaces to a common run-time kernel. The four interfaces shown in the figure are illustrative. They include two different Fortran interfaces (used by different HPF compilers), a user-level C++ interface called ad++[1], and a proposed Java interface. The development of several top-level interfaces has pro-

[1]ad++ is currently implemented as a set of header files defining distributed arrays as type-secure container class templates, function templates for collective array operations, and macros for distributed control constructs.

duced a robust kernel interface, on which we anticipate other language- and compiler-specific interfaces can be constructed relatively straightforwardly.

The largest part of the kernel is concerned with global communication and arithmetic operations on distributed arrays. These are represented on the right-hand side of figure 1. The communication operations supported include HPF/F90 array intrinsic operations such as `CSHIFT`, the function `pcrc_write_halo`, which updates ghost areas of a distributed array, the function `remap`, which is equivalent to a Fortran 90 array assignment between a conforming pair of sections of two arbitrarily distributed HPF arrays, and various gather- and scatter- type operations allowing irregular patterns of data access. Arithmetic operations supported include all F95 array reduction and matrix arithmetic operations, and HPF combining scatter. A complete set of HPF standard library functions is under development.

Nearly all these operations (including many of the arithmetic operations) are based on reusable schedules, in the PARTI/CHAOS mold. As well as supporting the inspector-executor compilation strategy, this organization is natural in an object-oriented setting—a communication pattern becomes an object. As an illustration, consider the reduction operations. All reductions from a distributed array to a global result are described by an abstract base class using virtual functions for local block reductions. Specific instances such as `SUM` or `PRODUCT` are created by deriving concrete classes that instantiate the arithmetic virtual functions. This is a cleaner and more type-secure (hence, potentially, more efficiently compilable) alternative to passing function pointers to a generic reduction function.

All the data movement schedules are dependent on the infra-structure on the left-hand side of the figure 1. This provides the distributed array descriptor, and basic support for traversing distributed data ("distributed control"). Important substructures in the array descriptor are the *range* object, which describes the distribution of an array global index over a process dimension, and the *group* object, which describes the embedding of an array in the active processor set.

At the time of writing the kernel is fully functional and quite mature, two of the four interfaces illustrated are complete, and others are in progress.

Run-time based HPF compiler

The NPAC compiler [74] is implemented as a translator from HPF to Fortran 77. It focusses on exploitation of explicit forall parallelism in the source HPF program.

The HPF compiler is embedded in a compiler construction framework illustrated in the figure 2. Currently it handles various HPF language features, including data mapping directives, expressions and assignment, program execution control, I/O, and procedures.

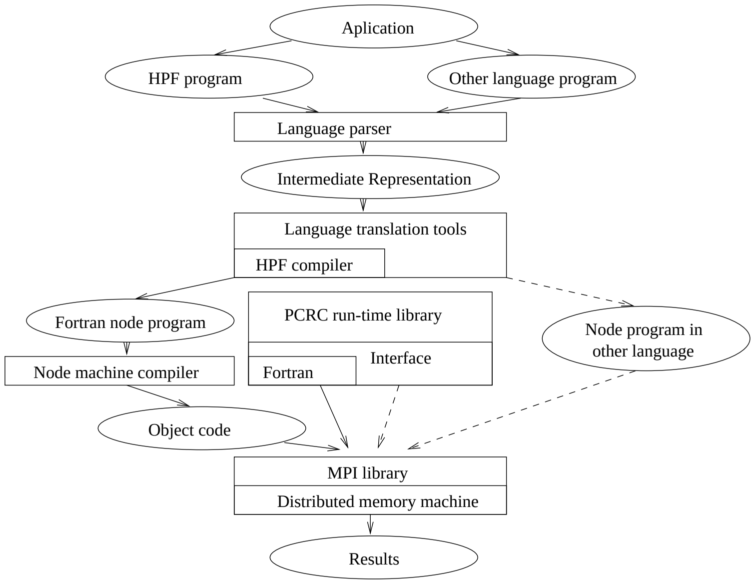

Figure 2: Compilation system overview

The compiler construction framework, called the *frontEnd* system [50], is an extensible software package for constructing programming language processing tools. The system analyzes the input source language and converts it to a uniform intermediate representation (IR) which can later be manipulated by a set of functions provided in a C++ class library.

The package is divided into the following components:

- **Language parsers** The language parsers are used to convert program source code to IR. The IR is designed to be suitable for multiple programming languages. We already have an HPF language parser (including a semantics checking module to check all constraints in the language definition) and a Java language parser. A C++ parser is being developed.

- **Package tools** These tools are used for displaying the intermediate representation. Two of them are available now. One tool unparses the IR back to the language source code for which the parse tree was generated. The other dumps the IR in plain text or HTML format for debugging purpose.

- **C++ class library** A C++ class library has been extended from the Sage++ class library. In the library, the nodes of the syntax tree are mapped to different C++ classes, with associated member functions provided to perform standard transformations, such as inserting or deleting statements from the tree.

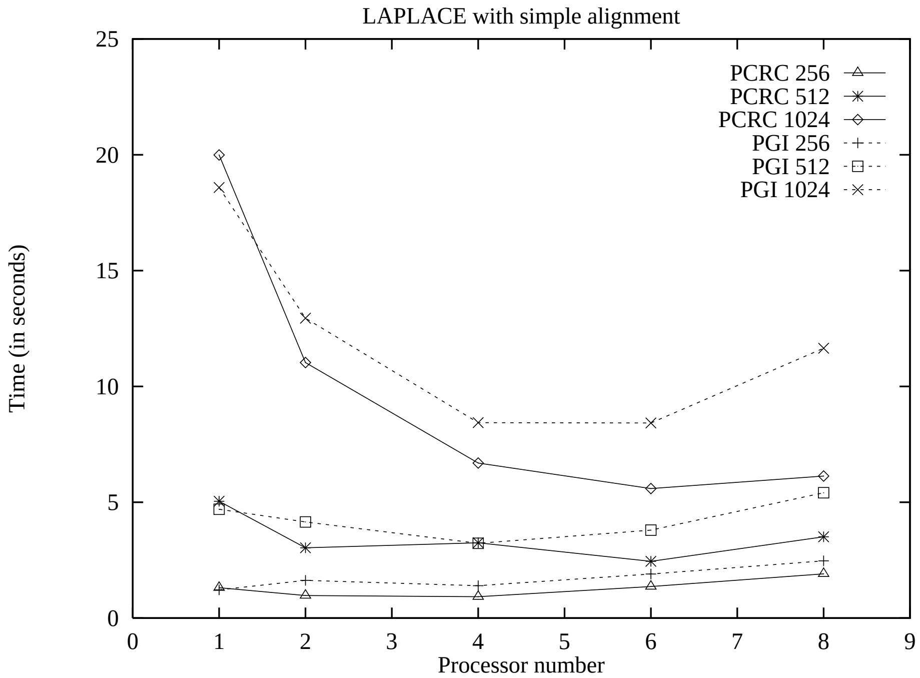

Figure 3: Laplace update.

- **Test suite** An extensive test suite is designed for different language parsers. For example, the package includes a "first order" exhaustive syntax test, in which each BNF syntax rule is used at least once. It also includes all example programs from "Migrating to Fortran 90" and the "HPF handbook" as test cases for the HPF parser.

The package is autoconfigured and tested on different platforms, including: IBM AIX, SUN OS/Solaris, DEC OSF, HP HP-UX, SGI IRIX and PC Linux.

Preliminary Benchmark Results

Figures 3 and 4 display select results of a benchmark comparision between the new NPAC compiler the PGI HPF compiler, version 2.0. The programs were run on the IBM SP2 at NPAC. The Laplace benchmark performs Jacobi relaxation on 256×256, 512×512 and 1024×1024 arrays, distributed blockwise over various numbers of processor. Both compilers achieve about the same performance on a single node, but generally our compiler exhibits better speedup on multiple processors, presumably due to more effective handling of communication. The synthetic benchmark involves no communication—it is a forall assignment involving large arrays. It suggests that (unlike the PGI compiler) we deal with address translation efficiently, even for cyclic

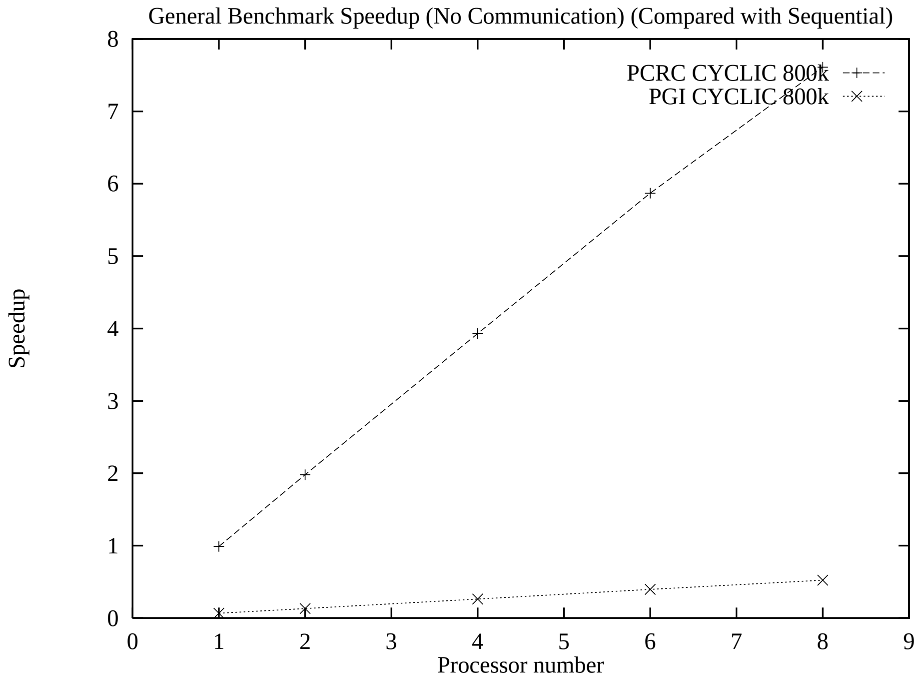

Figure 4: Synthetic benchmark involving cyclic distribution format.

distribution format. (Speedup is relative to an equivalent sequential program compiled with the IBM Fortran compiler.)

4 University of Maryland Report

Parallel Program Interoperability

One major part of the Maryland effort has focused on developing techniques for transferring data between distributed data structures owned by different data parallel runtime libraries. These techniques have been implemented in the Maryland Meta-Chaos library, which currently is capable of transferring data between (distributed) arrays that have been created using High Performance Fortran, the Maryland CHAOS and Multiblock PARTI libraries, and the the Indiana pC++ runtime library, Tulip. For example, the library allows an irregularly distributed array created via the CHAOS library to be copied into an HPF distributed array (with an arbitrary HPF distribution) that has been created using the HPF compiler and runtime library. The only requirement on the libraries, so that Meta-Chaos can access their data, is that each library provide several inquiry functions, similar to those that were specified previously in the PCRC DDD specification. Meta-Chaos can be used both for moving data between multiple data parallel libraries used within a single program and for moving data between libraries used in separate programs.

A paper on the design and implementation of the library has been published in the proceeding of the proceedings of the 1997 International Parallel Processing Symposium, and is also available on the web as a Maryland technical report at `http://www.cs.umd.edu/projects/hpsl/tool/metaChaos-paper.html`. We have used Meta-Chaos to implement the communication for a complete distributed application for the Maryland Grand Challenge project in land cover dynamics, which consists of a complex set of interacting parallel (and sequential) modules that use different data parallel libraries. The interacting modules include a parallel image database engine for retrieving satellite imagery, an image classification algorithm (parallelized using the Maryland Multiblock Parti library) and a user interface sequential program. The user interface program generates the image database queries and provides the facilities to build the distributed application from the various modules, and provides the information necessary for any of the modules to interact directly with the user, and display the results of the applications.

Additional work related to parallel program interoperability was directed at controlling data transfers between programs, in addition to actually moving the data using Meta-Chaos. In papers published in the 1996 International Conference on Supercomputing and the 1996 Parallel Object-Oriented Methods and Applications Workshop (POOMA96), we proposed an approach that achieves direct application to application data transfer. The approach is library-based and is independent of the programming language used to develop the communicating applications. Programs written to use this approach are required to adhere to a certain discipline with respect to the data structures involved in the interaction, but they do not need to know either the identity or the number of programs they interact with.

The approach is built around the notion of mappings between data structures in different data-parallel programs. Mappings are established at runtime. Every mapping has a consistency specification which mandates the logical frequency with which the mapped structures are to be made mutually consistent. Mappings, or the identity of the processors involved, do not have to be known at compile-time or even link-time. A priori knowledge of the mapping specification at run-time allows for overlapping execution of the interacting programs by buffering the data. Efficient data movement is achieved by pre-computing an optimized plan (schedule) for data movement. The prototype implementation uses the Meta-Chaos library to perform the data movement, so is able to couple data-parallel programs written in several different languages (including High Performance Fortran, C and pC++) and using different communication libraries (including the Maryland Multiblock Parti and Chaos libraries).

We developed a prototype implementation based on this approach. The implementation ran on a cluster of four-processor Digital Alpha Server 4/2100 symmetric multiprocessors. The results indicate that data-parallel programs can be coupled together in a flexible fashion with acceptable overhead.

Mobile Programs

Another main area of research at Maryland has been the investigation of mobility for Java programs. Mobile programs can move an active thread of control from one machine to another during execution. This flexibility has many potential advantages in a distributed environment. For example, a program that searches distributed data repositories can improve its performance by migrating to the repositories and performing the search on-site instead of fetching all the data to its current location. Similarly, an Internet video-conferencing application can minimize overall response time by positioning its server based on the location of its users. The primary advantage of mobility in these scenarios is that it can be used as a tool to adapt to variations in the operating environment.

For different applications, different resource constraints are likely to govern the decision to migrate, e.g. network latency, network bandwidth, memory availability, server availability. In recent work at Maryland, we have investigated network-aware mobile programs, i.e. programs that position themselves based on their knowledge of network characteristics. Whether the potential performance benefits of network-aware mobility are realized in practice depend on answers to three questions. First, how should programs be structured to utilize mobility to adapt to variations in network characteristics? In particular, what policies are suitable for making mobility decisions? Second, is the variation in network characteristics such that adapting to them proves profitable? Finally, can adequate network information be provided to mobile applications at an acceptable cost? A paper on this topic has been published in the proceedings of the USENIX 1997 Annual Technical Conference. The paper addresses all these mobility issues, and experiments indicate that network-aware programs can successfully use mobility to adapt to spatial and temporal variations in network latency over the Internet.

We have investigated these mobility issues in the context of Sumatra, an extension of the Java programming environment that provides a flexible substrate for adaptive mobile programs. A paper on Sumatra was published in a Springer-Verlag Lecture Notes in Computer Science book entitled "Mobile Object Systems". The paper is a first step in demonstrating that distributed programs can use mobility as a tool to adapt to variations in their operating environment. The main contribution of the work is that it shows the feasibility and profitability of the approach. We have established the feasibility of the approach by providing a programming interface and efficient system support for thread and object migration in Sumatra. Experiments with a distributed network latency monitor, called Komodo, and a mobile Internet chat server, called Adaptalk, both developed as part of this project, indicate that network-aware programs can successfully use mobility to adapt to spatial and temporal variations in network latency over the Internet.

Another paper discussing additional work on adapting to network bandwidth variations for wide-area data access has been submitted to the 1997 International Conference on Distributed Computing Systems and will soon also be available as a University of Maryland technical report.

See [35, 61, 60, 36, 63, 62, 1].

5 Indiana University Report

The work at Indiana University can be divided into three areas of development:

1. Support for HPC++ runtime in heterogeneous environments,
2. Development of Java compilation tools to support parallelism in Java and interoperability between Java and HPC++
3. Development of tools based on the Object Management Group's Common Object Request Broker Architecture (CORBA) specification.

In the following paragraphs we will describe the work and results in each of these areas.

HPC++ and Runtime Support for Heterogeneous Systems

High Performance C++ consists of a suite of tools for programming parallel application using object oriented methods. Based on C++, HPC++ has three basic components.

1. Parallel STL—A parallel implementation of the C++ Standard Template Library (STL). This library provides the standard high level data structures (vectors, lists, sets, maps, etc) that are used in many C++ applications.
2. HPC++Lib—A library containing
   - a Java style thread class.
   - basic synchronization primitive classes
   - template based collective operations for synchronization such as barriers, reductions, broadcasts, etc.

- a global pointer template library to access remote data structures and objects.

3. An method that programmers can use to allocate program loops to allow a preprocessor to generate versions of the code that run in parallel.

The standard runtime system for HPC++ is called Tulip and it is designed for homogeneous parallel processing systems with support for shared memory. However, our goal for the PCRC project has been to make HPC++ both portable and interoperable with other systems.

For that reason we have ported HPC++ to a new runtime system based on a "meta-object protocol" which allows us to easily host computations on a variety of lower level protocols. In this model each object can be associated with one or more communication "meta-objects". Each communication meta-object is associated with one low level protocol.

To make HPC++ interoperable within the Globus metacomputing environment, we have implemented a Nexus meta-object. To operate in the domain of CORBA, we are now working on an IIOP meta-object (see section on CORBA below).

Using our Nexus implementation we have been about to link parallel HPC++ program running on two different architectures together as one single parallel program. In addition, we have been able to "wrap" Fortran+MPI programs with HPC++ so that they can be accessed by HPC++ parallel programs running in different locations (see [42, 48]).

Parallel Java and Interoperability with Remote Objects

We have developed a Java preprocessor that will allow Java programs that are written with loops and certain types of recursion to be transformed into multi-threaded code that executes in parallel on shared memory multiprocessors. As with HPC++, this is accomplished with the aid of user annotation which are comments placed in front of the target loops and function calls. The preprocessor translates these into code that uses the Java threads interface to execute different loop iterations and function calls concurrently. The details have been reported in [9, 10, 13, 12]. In many cases, the Java source for a program is not available. Consequently, the required transformations must be applied at the level of Java byte code. Because this code is not annotated, a deeper analysis of the program structure must be made. The byte code restructurer is described in [9, 13]. We have also made an analysis of the performance of native level 1 BLAS from Java Programs [11].

While this support for parallelism is important for many compute intensive Java applications, a much larger problem is achieving high performance on Java remote method calls. More specifically, in distributed and heterogeneous application, Java program must be able to make calls on objects that reside elsewhere on the network. Java provides an elegant mechanism to call remote Java object from Java applications. This is known as Java Remote Method Invocation (RMI). However, for many application it is necessary for Java applets to call methods on non-java remote objects. Such objects may be HPC++ objects, CORBA objects, or encapsulated HPF and Fortran/MPI programs. There is already a standard for Java-CORBA calls, but it is not compatible with RMI. However, in the last month, SUN and OMG have agreed to make these compatible.

In our work we have approached this problem as follows. RMI provides a very clear model for how objects can be serialized and transmitted over the network and how remote methods can be invoked. By using our existing Java compiler technology, we have re-implemented the Java RMI so that it runs over Nexus. The details of this effort have been described in [14]. The exact protocol used to implement the RMI semantics was chosen to be compatible with the HPC++ over Nexus protocol. Consequently, we are now able to register a HPC++ object in the Java/Nexus registry and Java programs can access and execute remote method calls on that C++ object. This mechanism is used in the Linear System Analyzer which is a problem solving environment for solving large linear systems of equations. The design is based on a distributed software component architecture [42]. Much remains to be done to support a reasonable subset of Java serialization.

Parallel and High Performance CORBA

As the OMG's CORBA grows in popularity as a tool for interoperability, we have focused many of our efforts on pushing the performance of CORBA in high performance, parallel application. We have developed a parallel CORBA ORB called PARDIS which allows SPMD object implementations to communicate with other similar application using parallel communication channels [48]. In particular, PARDIS extends the concept of a CORBA sequence to a distributed sequence. This system has been used to link parallel Fortran/MPI applications with HPC++ SPMD computations.

Work Plan for Sept 1997 - June 1998

At the level of object-to-object communications and software component technology, we are working on IIOP and Nexus links between HPC++, Java, CORBA and object encapsulated HPF and Fortran with MPI. We expect Sun and OMG to deliver a standard binding for the CORBA Interface Definition Language (IDL) that supports RMI over IIOP and a subset of Java serialization.

We have already developed a IDL-to-C++ preprocessor technology and we have a suite of Java preprocessor tools. By Dec. 30, 1997 we expect to have a basic framework for object serialization between HPC++ and Java based IDL specifications.

The first quarter of 1998 will focus on two areas. First, we will extend IDL to support parallel streams in Java along the lines of our work on PARDIS. Second, we will attempt to extend the Java Beans and Java Studio Component model to allow us to build components that are built from HPC++, CORBA or encapsulated HPF or Fortran with MPI.

In the second quarter of 1998, we will deliver an example of a distributed component framework based on these technologies and demonstrate it with an extension of the Linear System Analyzer and one other distributed component application.

6 University of Texas, Austin Report

Goal

The goal for this project was to "productize" the Hierarchical Dynamic Distributed Array/Directed Acyclic Grid Hierarchy (HDDA/DAGH) infrastructure for parallel implementations of adaptive mesh computations and prepare the HDDA/DAGH infrastructure for integration with the pC++ runtime system.

Definition and Description of HDDA/DAGH

The structure of the HDDA/DAGH runtime system is given in the schematic of of figure 5.

The HDDA implements a hierarchical dynamic distributed array. "Hierarchical" means that each element of an instance of an array can itself be an array. "Dynamic" means that the number of levels in the hierarchy of arrays can be varied during execution of the program. "Distributed" means that the array may be partitioned across multiple address spaces transparently to C or Fortran programs which operate on these arrays.

DAGH is a layer of programming abstractions built upon the HDDA implementing a hierarchy of grids. DAGH is a PDI which implements instantiation of operations on a dynamic hierarchy of nested grids. DAGH supports implementation of a considerable span of AMR and multi-grid solution methods including vertex centered, cell-centered and edge -centered grids.

The conceptual framework for design and implementation of this PDI follows.

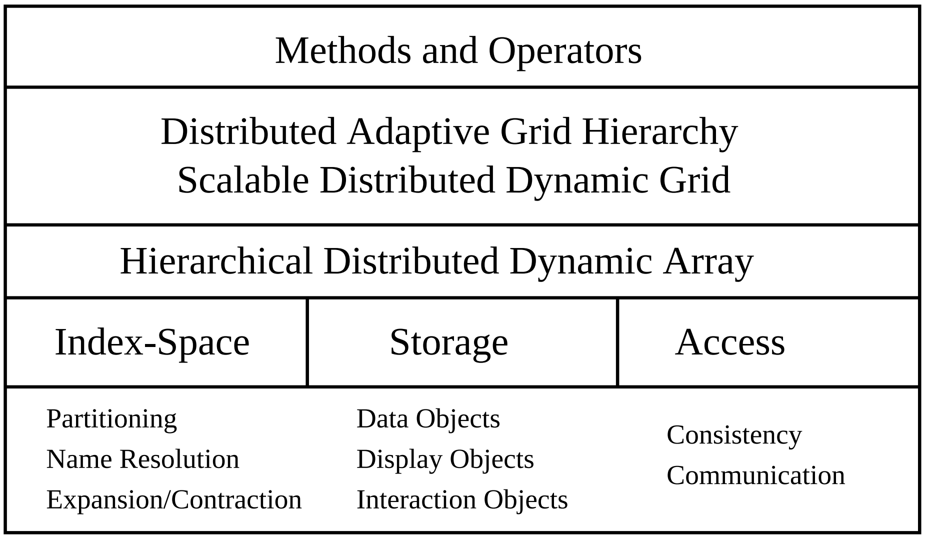

Figure 5:

Hierarchical Index Spaces

The hierarchical, extendible index space component of the HDDA is derived directly from the application domain using space-filling mappings which are computationally efficient, recursive mappings from N-dimensional space to 1- dimensional space. Figure 6 illustrates a 2-dimensional Peano-Hilbert curve. The solution space is first partitioned into segments. The space filling curve then passes through the midpoints

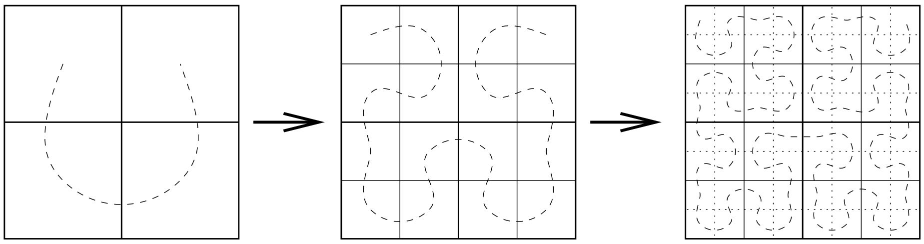

Figure 6:

of these segments. Space filling mappings encode application domain locality and maintain this locality though expansion and contraction. The self-similar or recursive nature of these mappings can be exploited to represent a hierarchical structure and to maintain locality across different levels of the hierarchy.

Space-filling mappings allow information about the original multi-dimensional space to be encoded into each space-filling index. Given an index, it is possible to obtain its position in the original multi-dimensional space, the shape of the region in the multi- dimensional space associated with the index, and the space-filling indices that are adjacent to it. The index-space is used as the basis for application domain partitioning, as a global name-space for name resolution, and for communication scheduling.

Mapping to Address Spaces

The mapping from the multi-dimensional index space to the one-dimensional physical address space is accomplished by mapping the positions in the index space to the order in which they occur in a traversal of the space filling curve. This mapping can be accomplished with simple bit-interleaving operations to construct a unique ordered key. This mapping produces a unique key set which defines a global address space. Coalescing segments of the linear key space into a single key, blocks of arbitrary granularity can be created.

Storage and Access–Extendible hashing for distributed dynamic storage and access

Data storage is implemented using extendible hashing techniques to provide a dynamically extendible, globally indexed storage (see the figure below). The keys for the Extendible Hash Table are contractions of the unique keys defined as described preceding.

Similarly, when data associated with a DAGH block entry is ready to ship, the associated bucket is shipped as is. The overall HDDA/DAGH distributed dynamic storage scheme is shown in figure 7.

Entries into the HDDA correspond to DAGH blocks. Expansion and contraction are local operations involving at most two buckets. Locality of data is preserved without copying. The HDDA data storage provides a means for efficient communication between DAGH blocks. To communicate data to another DAGH blocks, the data is copied to appropriate locations in the HDDA. This information is then asynchronously shipped to the appropriate processor.

Similarly, data needed from remote DAGH blocks is received on-the-fly and inserted into the appropriate location in the HDDA. Storage associated with the HDDA is maintained in ready-to-ship buckets. This alleviates overheads associated with packing and unpacking. An incoming bucket is directly inserted into its location in the HDDA.

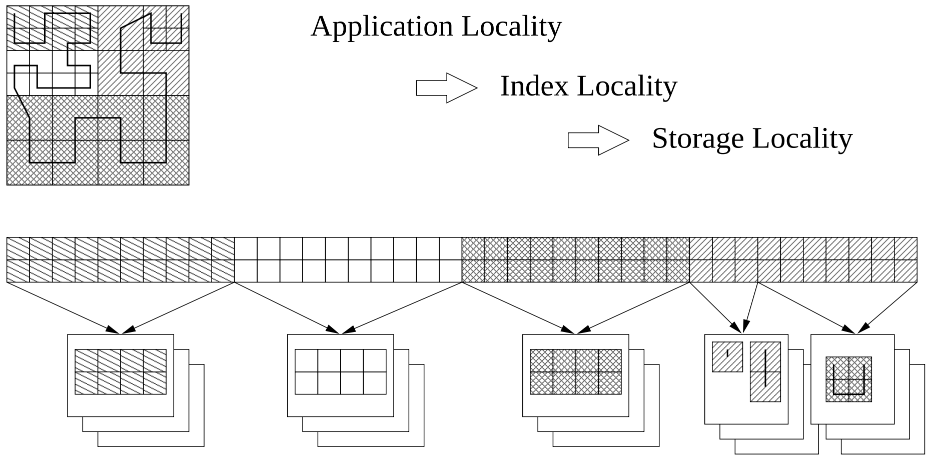

Figure 7:

Distributed Dynamic Storage - Partitioning and Communication

An instance of a DAGH is mapped to an instance of the HDDA. The granularity of the storage blocks is system dependent and attempts to balance the computation-communication ratio for each block. Each block in the list is assigned a cost corresponding to its computational load. In case of an AMR scheme, computational load is determined by the number of grid elements contained in the block and the level of the block in the AMR grid hierarchy. The former defines the cost of an update operation on the block while the latter defines the frequency of updates relative to the base grid of the hierarchy. Note that in the representation described above, space-filling mappings are applied to grid blocks instead of individual grid elements. The shape of a grid block and its location within the original grid is uniquely encoded into its space-filling index, thereby allowing the block to be completely described by a single index. Partitioning a DAGH across processing elements using this representation consists of appropriately partitioning the DAGH key list so as to balance the total cost at each processor. Since space-filling curve mappings preserve spatial locality, the resulting distribution is comparable to traditional block distributions in terms of communication overheads.

The individual objects referenced by each key, usually arrays, are stored in the standard layouts used by C and/or FORTRAN so that computational routines written in conventional sequential languages can be applied to these stored objects.

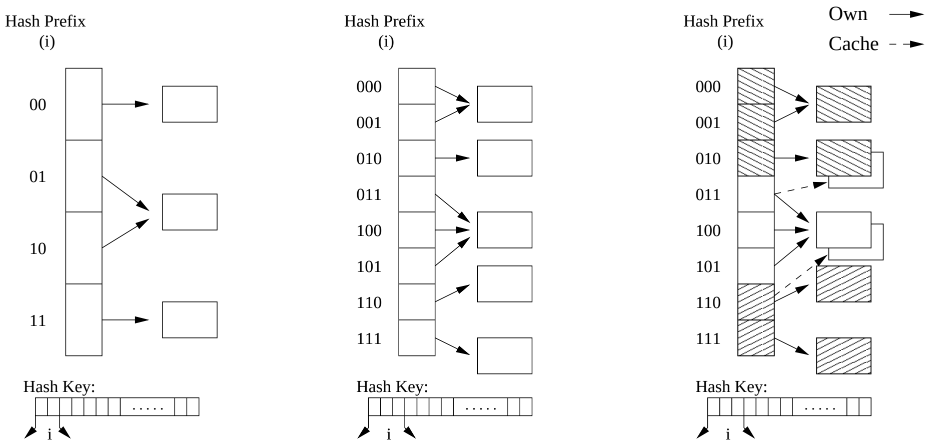

Figure 8: Dynamic hashing by splitting and coalescing buckets

Technical Tasks

The technical tasks to enable integration of HDDA/DAGH with other runtime libraries included:

1. Enhancement of the functionality of the DAGH layer to support a broader spectrum of computation abstractions.
2. Through testing and bug fixing for both the HDDA and the DAGH layers.
3. Development of documentation including a User Reference Manual, a Developers Reference Manual and a Tutorial on use of the DAGH class library.
4. Development of an installation procedure which will enable ready downloading and customization of the HDDA/DAGH library to most common execution environments.

When these tasks are fully completed HDDA/DAGH can be smoothly integrated into any C++ class library on either a source or runtime basis.

Accomplishments

1. A release of HDDA/DAGH with all of the additional capabilities including face-centered and vertex centered grids was released in May 1997.
2. A release which corrected the known bugs in the May release was completed in August and has been (and still is) undergoing extensive testing.

3. Documentation is still underway. A tutorial has been completed and can be downloaded from http://www.cs.utexas.edu/users/dagh. The Users Reference will be completed in about two months. The Developers Reference is still pending.

4. The installation procedure has been completed and is in place on the web page cited above.

Continuing Work

As can be seen from the previous section we are far from having completed the tasks of fully productizing HDDA/DAGH. However, funding for continuation of this project has been made available from the NCSA NSF Alliance and work continues. We currently estimate full complete in February of 1998.

For project-related publications, see [46, 45, 47, 44, 57, 53, 56, 54, 55, 8, 7].

7 University of Rochester Report

Rochester's goal in this project has been to access the utility of the new runtime system for distributed shared-memory architectures. We made a lot of progress towards our goal of building a parallelizing compiler and runtime system for distributed shared-memory machines. We have been studying the design of the new Compiler Runtime Systems in the context of data transformations and optimizations for distributed shared-memory machines. Our work can be divided into three categories: locality optimizations, scheduling for network of workstations, parallel and distributed data mining. We are also building a Java compiler.

Locality optimizations

We have developed compile-time optimizations that reduce false sharing. Many parallel computers provide shared memory with coherent caches. False sharing is a serious problem for many applications that run on such multiprocessors. This problem has been studied both in the context of generating parallel code from sequential programs and of programs with user-level parallelism. We consider explicitly parallel programs with user level parallelism that conform to the SPMD (Single Program Multiple Data) model. Several ways to eliminate false sharing in programs with task parallelism have been proposed. Most of the solutions to date require programmer intervention. We propose methods that can be automatically applied to effectively eliminate false sharing in some classes of applications. We use two techniques: array remapping (static and dynamic) and software caching. We implement those techniques in a source-to-source optimizer for C programs. We test the effectiveness of the transformations on a set of six applications and conclude that a significant improvement of the running time can be achieved. For all but the smallest data sizes, an improvement of at least a factor of two is observed. For some applications the execution times are reduced as much as ten times.

This result has been published in the 10th Annual International Conference on High Performance Computers, June 5-7 1996, Ottawa, Canada [25].

In another paper on data reuse for distributed shared-memory machines, we have compared experimentally the inherent data reuses in the standard benchmarks, the data locality behavior from program execution and the reuse patterns captured by static compiler models. The inherent data reuses provide the machine-independent upper bound on data reuses. We define 3 levels of reuse: *program reuse*, *loop nest reuse*, and *innermost loop reuse*. We show that there is more reuse to be exploited if we consider optimizations across loop nests. We observe that static models only capture reuses within the innermost loops or a loop nest. Most of the static models provide the similar information about the innermost loop reuses. However, we need more global compile-time reuse analysis and algorithms to exploit global locality.

This result has been published in the 10th Annual International Conference on High Performance Computers, June 5-7 1996, Ottawa, Canada [24].

Networks of workstations

Our recent work showed that a compile and run-time modeling and decision process which, combined with a run-time load balancing library, would enable us to choose the best strategy. Since there are a large number of loop scheduling techniques which have been proposed in the literature, this analysis helps a parallelizing compiler in customizing the dynamic load balancing strategy for a given application.

A paper on this result has been accepted for publication in the 5th IEEE International Symposium on High-Performance Distributed Computing, Syracuse, New York, August 1996 [67].

Data mining

Data mining is an emerging research area, whose goal is to extract significant patterns or interesting rules from large databases. High-level inference from large volumes of routine business data can provide valuable information to businesses, such as customer buying patterns, shelving criterion in supermarkets and stock trends. Many algorithms have been proposed for data mining of association rules. However, research so far has mainly focused on sequential algorithms. In this work we present parallel algorithms for data mining of association rules, and study the degree of parallelism, synchronization, and data locality issues on the SGI Power Challenge shared-memory multi-processor. We further present a set of optimizations for the sequential and parallel algorithms. Experiments show that a significant improvement of performance is achieved using our proposed optimizations. We also achieved good speed-up for the parallel algorithm, but we observe a need for parallel I/O techniques for further performance gains.

A paper on this result has been accepted for publication in Supercomputing'96, Pittsburgh, Pennsylvania, Nov. 1996 [68].

In another piece of work, we look at the problem of compiler analysis for database programs. Most parallel databases exploit two types of parallelism: *intra-query* parallelism and *inter-transaction* concurrency. Between these two cases lies another type of parallelism: *inter-query* parallelism within a transaction or application. Exploiting inter-query parallelism requires either compiler support to automatically parallelize the existing *embedded* query programs, or programming support to write explicitly parallel query programs. We have developed compiler analysis to automatically detect parallelism in the *embedded query programs*. We have proposed compiler dependence test algorithms for detecting both *internal* and *external* dependences.

A paper on this result has been accepted for publication in the Eighth IEEE Symposium on Parallel and Distributed Processing, New Orleans, Louisiana, October, 1996 [59].

See also [69].

Java compiler

In this work, we present a Java compiler architecture which uses a unique combination of front- and back-ends to deliver great flexibility. Our compiler is designed to use the same optimization passes no matter which pair of front- and back-end is used.

More details can be found in [23].

As a companion to the Java compiler work, we have built a system for network based visualization of profile information generated by Java applets/bytecode. The system, called **NetProf**, is composed of several components each of which is interesting in their own right. The components are a bytecode to Java source code translator, a profiler that includes a static pass to insert profiler code, a dynamic runtime library that records relevant events and finally a visualization mechanism which highlights the profile information in an easy to use manner. All of this can be done over the Internet using a client-server approach and is independent of the underlying architecture/machine and human intervention.

More details can be found in [58].

Other Rochester publications see [27, 31, 30, 29, 28, 49, 26, 71, 72, 70].

8 Florida University Report

Florida became involved in the project after Sanjay Ranka, one of the original co-PIs, moved there from Syracuse.

We evaluated the High-Performance Fortran (HPF) language as a candidate for implementing scientific, engineering and computer science software on parallel machines [64]. We reviewed major HPF language features and discusses general algorithmic issues common to broad classes of SPMD applications. We developed concrete examples on how they may be programmed in HPF, or have provided code fragments demonstrating the pertinent language features. These fragments were representative of applications in magnetic spin models, computational fluid dynamics, financial simulations, image processing, particle dynamics, and the solution of matrix equations. We found that High Performance Fortran language is suitable for these applications. The codes are part of the NPAC HPF Applications suite, and the further expanding and benchmarking them is an ongoing process.

Parallelization of applications requires each processor to access all the nonlocal data required for its local computation. For a number of applications these accesses can be described more formally by using Random Access Reads (RAR) and Random Access Writes (RAW). This formalism covers the basic runtime support required for the parallelization of the assignment statement, forall statement/construct, redistribution statement and several computational intrinsics in HPF for arbitrary distributions (regular as well as irregular). The theoretical work was developed a few years ago and was recently revised and accepted in JPDC [65, 66].

We extended the theoretical work and developed practical algorithms for combining gather/scatter functions. These algorithms involved several novel ideas for local and global combining of accesses at runtime [3]. We performed extensive benchmarking on parallel machines and demonstrated that the algorithms developed were practical. They were robust in the presence of hotspots for data access. The software developed was integrated as part of the PCRC runtime library and required generalizations for arbitrary distributions, arbitrary number of dimensions and arbitrary number of processors.

Vector reduction and prefix operations can be used effectively in many applications with unstructured data accesses such as *Pack/Unpack*, *Array Reduction/Prefix Functions*, and *Array Combining Scatter Functions*, which are defined in Fortran 90 and in High Performance Fortran. They can also be used in applications with structured accesses such as HPF array reduction and array prefix/suffix functions. We have developed a scalable and work optimal algorithms for this operation [5]. This algorithm is shown to be practical and shows a factor of 3-5 improvement for large number of processors.

PACK/UNPACK are Fortran 90/HPF array construction functions which derive new arrays from existing arrays. We developed new algorithms for performing these operations on coarse-grained parallel machines [4]. Our algorithms are relatively architecture independent and can be applied to arrays of arbitrary dimensions with arbitrary distribution along every dimension.

Integer sorting is a subclass of the sorting problem where the elements have integer values and the largest element is polynomially bounded in the number of elements to be sorted. It is useful for applications in which the size of the maximum value of element to be sorted is bounded. We developed a new algorithm, *distributed radix-sort*, for integer sorting [2]. The structure of our algorithm is similar to radix sort except that it requires less number of communication phases. Our experimental results on the CM-5 show that the algorithm is better than previous algorithms designed for sorting arbitrary radix integers. These algorithm can be utilized to support Sort routines (`Grade_up` and `Grade_down`) in the HPF runtime library.

9 Cooperating Systems

Cooperating Systems has contributed primarily in a support role, providing software test, evaluation, and documentation support to other consortium members (particularly NPAC). Additionally, CSC has played a leading role in two primary PCRC contract tasks: derivation of a Common Code and Data Descriptor for Arrays in HPC Languages [18] and compiling the Common Compiler and Data Movement Interface Specification [32]. They also presented a paper at the PCRC-sponsored workshop on Java for Computational Science and Engineering [21].

10 Rice University

Rice have contributed through the runtime for Fortran 77D [22]. They have also been working on optimization of Java programs [15] and have advised on associated runtime requirements.

A Deliverable Software

There are three software packages currently available at the NPAC site:

`http://www.npac.syr.edu/projects/pcrc`

They are

- The NPAC PCRC runtime kernel
- An HPF 2.0 front-end ,which also incorporates a Java front-end
- A Fortran-to-Java translator.

Software developed by other members of PCRC are maintained at respective sites, including

- Chaos library for irregular computation from University of Maryland,
- Sage++/pC++ for high performance C++ from Indiana University,
- Meta-chaos for interoperability of different runtime systems from University of Maryland,
- and DAGH for adaptive mesh refinement from University of Texas at Austin.

B Project-related Publications

References

[1] Anurag Acharya, M.Ranganathan, and Joel Saltz. Sumatra: A language for resource-aware mobile programs. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[2] K. Alsabti and S. Ranka. Integer sorting algorithms for coarse-grained parallel machines. To appear in HiPC 1997.

[3] S. Bae and S. Ranka. Array combining scatter functions on coarse-grained, distributed-memory parallel machines. Submitted to IPPS '98.

[4] S. Bae and S. Ranka. Vector reduction and prefix on coarse-grained, distributed-memory parallel machines. Submitted to IPPS '98.

[5] S. Bae and S. Ranka. PACK/UNPACK on coarse grained machines. *Journal of Parallel And Distributed Computing (Special Issue on Compilation Techniques for Distributed Memory Systems)*, pages 204–216, November 1996.

[6] Seungjo Bae and Sanjay Ranka. nplc_tsc: Implementation of HPF array combining scatter functions by using two-stage algorithm, 1997. URL: http://www.npac.syr.edu/projects/pcrc/July97/doc.html.

[7] Dwip Banerjee and J.C. Browne. Complete parallelization of computation: Integration of data partitioning and functional parallelism for dynamic data structures. In *10th International Parallel Processing Symposium*, pages 354–360, Honolulu, Hawaii, April 1996.

[8] Dwip Banerjee and J.C. Browne. Optimal parallelism through integration of data and control parallelism: A case study in complete parallelization. In *Workshop on Languages and Compilers for Parallel Computing*, 1997.

[9] A.J.C. Bik, F. Breg, and D.B. Gannon. Compiler support for Java parallelization. *Compilers for Physics*, 1998. To appear.

[10] A.J.C. Bik and D.B. Gannon. Exploiting implicit parallelism in Java. *Concurrency: Practice and Experience*, 9(6):579–619, 1997.

[11] A.J.C. Bik and D.B. Gannon. A note on native level 1 BLAS in Java. In *2nd Workshop on Java for High Performance Computing*, 1997. To appear in *Concurrency: Practice and Experience*.

[12] A.J.C. Bik, J.E. Villacis, and D.B. Gannon. Experiences with loop parallelization in JAVAR. In *10th International Workshop on Languages and Compilers for Parallel Computing*, 1997. To appear in Lecture Notes in Computer Science.

[13] A.J.C. Bik, J.E. Villacis, and D.B. Gannon. JAVAR: a prototype Java restructuring compiler. In *2nd Workshop on Java for High Performance Computing*, 1997. To appear in *Concurrency: Practice and Experience.*

[14] F. Breg. Compiler support for an RMI implementation using NexusJava. Technical report, Department of Computer Science, Indiana University, Bloomington, IN, 1997.

[15] Zoran Budimlic and Ken Kennedy. Optimizing Java—theory and practice. *Concurrency: Practice and Experience*, 9(6):445, 1997.

[16] Bryan Carpenter, Yuh-Jye Chang, Geoffrey Fox, Donald Leskiw, and Xiaoming Li. Experiments with HPJava. *Concurrency: Practice and Experience*, 9(6):633, 1997.

[17] Bryan Carpenter, Yuh-Jye Chang, Geoffrey Fox, and Xiaoming Li. Java as a language for scientific parallel programming. In *10th International Workshop on Languages and Compilers for Parallel Computing*, 1997. To appear in Lecture Notes in Computer Science.

[18] Bryan Carpenter, James Cowie, Donald Leskiw, and Xiaoming Li. The distributed array descriptor for a PCRC HPF compiler, version 2.0. Technical Report SCCS-770d, Northeast Parallel Architectures Center, Syracuse University, October 1996. URL: http://www.npac.syr.edu/projects/pcrc/July97/doc.html.

[19] Bryan Carpenter, Guansong Zhang, and Yuhong Wen. *NPAC PCRC Runtime Kernel Definition*, 1997. In preparation. For current draft, see http://www.npac.syr.edu/projects/pcrc/July97/doc.html.

[20] D.B. Carpenter. PCRC Fortran 90 and HPF syntax test suite. Technical report, Northeast Parallel Architectures Center, Syracuse University, February 1996. URL: http://www.npac.syr.edu/projects/pcrc/July97/doc.html.

[21] Marina Chen and James Cowie. Java's role in distributed collaboration. *Concurrency: Practice and Experience*, 9(6):509, 1997.

[22] Alok Choudhary, Geoffrey C. Fox, Tomasz Haupt, Xiaoming Li, Chao-Wei Ou, Sanjay Ranka, and Ken Kennedy. Fortran 77D and Fortran 90D: Scalable and

portable software modules for simd and mimd distributed memory parallel computers. Technical report, Northeast Parallel Architectures Center, Syracuse University, August 1995.

[23] M. Cierniak and W. Li. Briki: a flexible Java compiler. Technical Report TR 621, Computer Science Department, University of Rochester, May 1996.

[24] M. Cierniak and W. Li. Evaluation of data reuse, locality and compiler models. In *Proceedings of the 10th Annual International Conference on High Performance Computers*, Ottawa, Canada, June 1996.

[25] M. Cierniak and W. Li. A practical approach to the compile-time elimination of false sharing for explicitly parallel programs. In *Proceedings of the 10th Annual International Conference on High Performance Computers*, Ottawa, Canada, June 1996.

[26] M. Cierniak and W. Li. Briki: an optimizing Java compiler. In *Proceedings of the IEEE CompCon '97*, San Jose, California, February 1997.

[27] Michal Cierniak and Wei Li. Just-in-time optimizations for high-performance Java programs. To appear in Concurrency: Practice and Experience.

[28] Michal Cierniak and Wei Li. Interprocedural array remapping. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT '97)*, San Francisco, CA, November 1997. An earlier version available as URCS Technical Report 642, November 1996.

[29] Michal Cierniak and Wei Li. Optimizing Java bytecodes. *Concurrency: Practice and Experience*, 9(6):427–444, 1997.

[30] Michal Cierniak, Wei Li, and Mohammed Javeed Zaki. Compile-time scheduling algorithms for heterogeneous network of workstations. To appear The Computer Journal, 1997.

[31] Michal Cierniak and Suresh Srinivas. A portable browser for performance programming. To appear in Concurrency: Practice and Experience.

[32] Parallel Compiler Runtime Consortium. Common compiler and data movement interface specification. URL: http://www.cooperate.com/PCRCdev/T6.html.

[33] Parallel Compiler Runtime Consortium. Common runtime support for high-performance parallel languages. In *Supercomputing '93*. IEEE Computer Society Press, 1993.

[34] Parallel Compiler Runtime Consortium. HPCC and Java—a report by the Parallel Compiler Runtime Consortium. URL: http://www.npac.syr.edu/users/gcf/hpjava3.html, May 1996.

[35] Guy Edjlali, Alan Sussman, and Joel Saltz. Interoperability of data parallel runtime libraries with Meta–Chaos. Technical Report CS-TR-3633 and UMIACS-TR-96-30, University of Maryland, Department of Computer Science and UMIACS, 1996.

[36] Guy Edjlali, Alan Sussman, and Joel Saltz. Interoperability of data parallel runtime libraries. In *Proceedings of the Eleventh International Parallel Processing Symposium*. IEEE Computer Society Press, April 1997.

[37] Geoffrey Fox and Wojtek Furmanski. Computing on the Web—new approaches to parallel processing—petaop and exaop performance in the year 2007. Technical Report SCCS-784, Northeast Parallel Architectures Center, Syracuse University, January 1997.

[38] Geoffrey Fox, Xiaoming Li, and Zheng Qiang. A prototype of Fortran-to-Java converter. In *ACM Workshop on Java for Science and Engineering Computation*, June 1997. To appear in Concurrency: Practice and Experience, special edition.

[39] Geoffrey C. Fox, editor. *Java for Computational Science and Engineering—Simulation and Modelling*, volume 9(6) of *Concurrency: Practice and Experience*, June 1997.

[40] Geoffrey C. Fox, editor. *Java for Computational Science and Engineering—Simulation and Modelling II*, volume 9(11) of *Concurrency: Practice and Experience*, November 1997.

[41] Geoffrey C. Fox, Xiaoming Li, Yuhong Wen, and Guansong Zhang. Studies of integration and optimization of interpreted and compiled languages. Technical Report SCCS-780, Northeast Parallel Architectures Center, Syracuse University, February 1997.

[42] D. Gannon, R. Bramley, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Breg, S. Diwan, and M. Govindaraju. Component architectures for distributed scientific problem solving. Submitted to IEEE CS&E.

[43] ACM 1998 workshop on Java for high-performance network computing. To be held at Palo Alto, California, February 28 and March 1, 1998. URL: http://www.cs.ucsb.edu/conferences/java98.

[44] Ajita John and J.C. Browne. Compilation of constraint programs with non-cyclic and cyclic dependencies to procedural parallel programs. To appear in Journal of Parallel and Distributed Systems.

[45] Ajita John and J.C. Browne. Compilation of constraint systems to procedural parallel programs. In *Workshop on Languages and Compilers for Parallel Computers*, Sunnyvale, California, August 1996. To appear in Lecture Notes in Computer Science.

[46] Ajita John and J.C. Browne. Compilation to parallel programs from constraints. In *Third International Conference on High Performance Computing*, pages 73–79, Delhi, India, December 1996.

[47] Ajita John and J.C. Browne. Extraction of parallelism from constraint specifications. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 1996)*, volume 3, pages 1501–1512, San Jose, California, August 1996.

[48] Kate Keahey and Dennis Gannon. PARDIS: A parallel approach to CORBA. In *6th IEEE International Symposium on High Performance Distributed Computation*, August 1997.

[49] Leonidas I. Kontothanassis, Galen C. Hunt, Robert J. Stets, Nikolaos Hardavellas, Michal Cierniak, Srinivasan Parthasarathy, Jr. Wagner Meira, Sandhya Dwarkadas, and Michael L. Scott. VM-based shared memory on low-latency, remote-memory-access networks. In *Proceedings of the 24th Annual ACM/IEEE International Symposium on Computer Architecture*, Denver, Colorado, June 1997. Also available as URCS Technical Report 643, November 1996.

[50] Xiaoming Li. HPFfe: a front-ed for HPF. Technical Report SCCS-771, Northeast Parallel Architectures Center, Syracuse University, October 1996. URL: http://www.npac.syr.edu/projects/pcrc/July97/doc.html.

[51] Xiaoming Li. Runtime oriented HPF compilation. Technical Report SCCS-779, Northeast Parallel Architectures Center, Syracuse University, January 1997.

[52] John Merlin, Bryan Carpenter, and Tony Hey. shpf: a subset High Performance Fortran compilation system. *Fortran Journal*, pages 2–6, March 1996.

[53] Manish Parashar and J.C. Browne. Systems engineering for high performance computing software: The HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh. In *Structured Adaptive Mesh Refinement Grid Methods*, IMA Volumes in Mathematics and its Applications. Springer-Verlag.

[54] Manish Parashar and J.C. Browne. Distributed dynamic data structures for parallel adaptive mesh refinement. In *International Conference for High Performance Computing*, pages 22–27, New Delhi, India, December 1995.

[55] Manish Parashar and J.C. Browne. Object-oriented programming abstractions for parallel adaptive mesh refinement. In *Parallel Object-Oriented Methods and Applications Workshop (POOMA)*, Santa Fe, New Mexico, February 1996.

[56] Manish Parashar and J.C. Browne. On partitioning dynamic adaptive grid hierarchies. In *29th Annual Hawaii International Conference on Systems Sciences*, pages 604–613, Maui, Hawaii, January 1996.

[57] Manish Parashar and J.C. Browne. A common data management infrastructure for adaptive algorithms for PDE solutions. In *Supercomputing '97*, 1997.

[58] S. Parthasarathy, M. Cierniak, and W. Li. NetProf: Network-based high-level profiling of Java bytecode. Technical Report TR 622, Computer Science Department, University of Rochester, May 1996.

[59] S. Parthasarathy, W. Li, M. Cierniak, and M. J. Zaki. Compile-time inter-query dependence analysis. In *Proceedings of the Eighth IEEE Symposium on Parallel and Distributed Processing*, New Orleans, Louisiana, October 1996.

[60] M. Ranganathan, A. Acharya, G. Edjlali, A. Sussman, and J. Saltz. Flexible and efficient coupling of data parallel programs. In *Proceedings of Parallel Object-Oriented Methods and Applications (POOMA96)*, February 1996.

[61] M. Ranganathan, A. Acharya, G. Edjlali, A. Sussman, and J. Saltz. Runtime coupling of data-parallel programs. In *Proceedings of the 1996 International Conference on Supercomputing*, pages 229–236. ACM Press, May 1996.

[62] M. Ranganathan, Anurag Acharya, and Joel Saltz. Adapting to bandwidth variations in wide-area data access. Technical Report In preparation, Department of Computer Science and UMIACS, University of Maryland, October 1997. A shorter version has been submitted to ICDCS'98. Submission is available at http://www.cs.umd.edu/ ranga/icdcs98/submitted.

[63] M. Ranganathan, Anurag Acharya, Shamik Sharma, and Joel Saltz. Network aware mobile programs. In *Proceedings of the USENIX 1997 Annual Technical Conference*. Usenix Association, January 1997.

[64] Sanjay Ranka, Hon W Yau, Kenneth A Hawick, and Geoffrey C Fox. High-Performance Fortran for SPMD programming: An applications overview, 1996. URL: http://www.npac.syr.edu/hpfa/Papers/HPFforSPMD/.

[65] Ravi V. Shankar and Sanjay Ranka. Random data accesses on a coarse-grained parallel machine I. one-to-one mappings. *Journal of Parallel and Distributed Computing*, pages 15–23, July 1997.

[66] Ravi V. Shankar and Sanjay Ranka. Random data accesses on a coarse-grained parallel machine II. one-to-many and many-to-one mappings. *Journal of Parallel and Distributed Computing*, pages 24–34, July 1997.

[67] M. J. Zaki, W. Li, and S. Parthasarathy. Customized dynamic load balancing for a network of workstations. In *Proceedings of the 5th IEEE International Symposium on High-Performance Distributed Computing*, Syracuse, New York, August 1996.

[68] M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel data mining for association rules on shared-memory multi-processors. In *Proceedings of Supercomputing '96*, Pittsburgh, Pennsylvania, November 1996.

[69] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Wei Li, , and Mitsunori Ogihara. Evaluation of sampling for data mining of association rules. In *7th International Workshop on Research Issues in Data Engineering (RIDE'97)*, Birmingham, UK, April 1997. (An older version also available as Technical Report 617, Computer Science Dept., U. Rochester, May 1996).

[70] Mohammed Javeed Zaki, Srinivasan Parthasarathy, and Wei Li. A localized algorithm for parallel association mining. In *9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Newport, Rhode Island, August 1997.

[71] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New parallel algorithms for fast discovery of association rules. To appear in Data Mining and Knowledge Discovery: An International Journal, special issue on Scalable High-Performance Computing for KDD, late 1997.

[72] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In *3rd International Conference on Knowledge Discovery and Data Mining (KDD)*, Newport, California, August 1997.

[73] Guansong Zhang. Summary of HPF front-end test. URL: http://www.npac.syr.edu/projects/pcrc/July97/doc.html.

[74] Guansong Zhang, Bryan Carpenter, Geoffrey Fox, Xiaoming Li, Xinying Li, and Yuhong Wen. PCRC-based HPF compilation. In *10th International Workshop*

on Languages and Compilers for Parallel Computing, 1997. To appear in Lecture Notes in Computer Science.