The GrADS Project: Software Support for High-Level Grid Application Development*

Francine Berman, Andrew Chien, Keith Cooper, Jack Dongarra, Ian Foster,
Dennis Gannon, Lennart Johnsson, Ken Kennedy, Carl Kesselman,
Dan Reed, Linda Torczon, and Rich Wolski

February, 15, 2000

Abstract

Advances in networking technologies will soon make it possible to use the global information infrastructure in a qualitatively different way—as a *computational* resource as well as an information resource. This idea for an integrated computation and information resource called the *Computational Power Grid* has been described by the recent book entitled *The Grid: Blueprint for a New Computing Infrastructure* [18]. The Grid will connect the nation's computers, databases, instruments, and people in a seamless web, supporting emerging computation-rich application concepts such as remote computing, distributed supercomputing, tele-immersion, smart instruments, and data mining.

To realize this vision, significant scientific and technical obstacles must be overcome. Principal among these is *usability.* Because the Grid will be inherently more complex than existing computer systems, programs that execute on the Grid will reflect some of this complexity. Hence, making Grid resources useful and accessible to scientists and engineers will require new software tools that embody major advances in both the theory and practice of building Grid applications.

The goal of the *Grid Application Development Software (GrADS) Project* is to simplify distributed heterogeneous computing in the same way that the World Wide Web simplified information sharing over the Internet. The GrADS project is exploring the scientific and technical problems that must be solved to make Grid application development and performance tuning for real applications an everyday practice. This requires research in four key areas, each validated in a prototype infrastructure that will make programming on grids a routine task:

- Grid software architectures that facilitate information flow and resource negotiation among applications, libraries, compilers, linkers, and runtime systems;
- base software technologies, such as scheduling, resource discovery, and communication, to support development and execution of performance-efficient Grid applications;
- languages, compilers, environments, and tools to support creation of applications for the Grid and solution of problems on the Grid; and
- mathematical and data structure libraries for Grid applications, including numerical methods for control of accuracy and latency tolerance.

In addition, the GrADS project is developing MicroGrid testbeds to systematically explore, demonstrate and characterize the effectiveness of the technologies developed, while using the evolving national Grid testbeds for full-scale experimentation and demonstration.

The GrADS Project will carry out technology transfer via two principal mechanisms. First, the principal investigators will work directly with application developers to understand the mechanisms that would be useful for bringing up their applications on the Grid. Second, the project will collaborate with industrial partners to encourage the adoption and standardization of system software technologies that arise from the research.

To be successful, GrADS will need foster research and technology transfer programs that will contribute to revolutionary new ways of utilizing the global information infrastructure as a platform for computation, changing the way scientists and engineers solve their everyday problems.

*This project is being supported by the National Science Foundation Next Generation Software Program.

1 Introduction

A recently published volume entitled "The Grid: Blueprint for a New Computing Infrastructure" [18] has established a compelling vision of a national computational and information resource that will change the way the nation's scientists and engineers attack complex problems [40, 18]. This *Computational Power Grid* will use advanced networks such as those being established by the Next Generation Internet program to couple the nation's computers, databases, instruments, and scientists in a seamless web of computing and distributed intelligence. The emerging use of distributed resources as a computational platform is of national importance because it enables a new generation of multi-disciplinary, high-performance applications. Typical of these new applications are hybrid computations that integrate supercomputer simulations with distributed data sources, all controlled from the scientist's or engineer's desktop. For example, in the aircraft industry, the Grid will allow engineers and subcontractors to collaborate on optimizing an entire design by using globally distributed simulations linked with geographically distributed databases of manufacturing and airline specifications.

Many of the physical resources needed to support such advanced Grid applications are available today. However, experience with prototype systems (*e.g.,* [30, 13]) shows that the development of such applications is incredibly difficult, because of (1) the complex, dynamic nature of the Grid environment, (2) the complexity of the applications themselves, and (3) the almost complete absence of software tools. *Significant research advances are needed before Grid environments can be used routinely for science and engineering.* The situation is similar to that of information sharing on the Internet prior to the development of the World Wide Web: the core capabilities were there, but they were not easy to use. It is this combination of the national importance of the problem and the need for multidisciplinary research advances that has led us to initiate, with support from the National Science Foundation, a project of broad scope and significant duration in this area:

> The goal of the *Grid Application Development Software* (GrADS) project is to conduct fundamental research leading to the development, in prototype form, of technologies needed to make the Grid usable on a daily basis by scientists and engineers.

1.1 Opportunities

The existence of a successful Grid and associated application development infrastructure will spur the creation of radical new classes of applications and approaches to problem solving. For example:

The Grid will enable the construction of new desktop tools that use *on-demand access* to remote hardware capabilities and specialized software resources. Dynamic acquisition of remote hardware resources can increase the instantaneous performance accessible at the desktop by several orders of magnitude. The integration of advanced or proprietary algorithmic techniques encoded in remote software can produce similar improvements. Finally, on-demand access to advanced computing will catalyze creation of new problem solving approaches and tools, including desktop engineering design systems, chemistry solvers, and numerical modeling packages.

The Grid will make possible *distributed supercomputing* applications that harness the nation's fastest supercomputers, applying extraordinary and diverse computing resources to large problems that cannot be solved on any single system. These computations may bring together large multi-disciplinary problem solving teams, such as those responsible for complex aircraft design problems. In times of crisis, they may be used to plan responses to a major environmental disaster, earthquake, or terrorist attack.

The Grid will permit the development of *information sharing* applications that couple computers with globally distributed information resources, such as databases, sensors, and scientific instruments. For example, an astrophysicist may integrate information located in multiple astronomical databases to compute correlations between different properties of stellar regions; a computer-enhanced electron microscope may search image databases to identify images with similar characteristics; and a collaborative engineering system may connect multiple users to large, distributed simulations.

1.2 Challenges

We referred above to three major obstacles that must be overcome before Grid applications such as those just listed can be widely developed and used:

Platform Complexity The Grid is a highly heterogeneous environment, encompassing different computer architectures, networking technologies, communication protocols, and operating systems. Performance characteristics vary greatly, with latencies orders of magnitude greater than in conventional computing systems and bisection bandwidths commensurately smaller. The large scale of Grid systems leads to lowered reliability and extensive sharing of resources; hence, knowledge of system structure and state is incomplete, and dynamic and unpredictable behavior becomes the norm. These characteristics have major implications for applications that require performance guarantees. For example, *adaptivity* is likely to be a key requirement, so that applications can function appropriately as resource utilization and availability change, computers and networks fail, old components are retired, new systems are added, and both software and hardware on existing systems are updated and modified.

Application Complexity The diversity of resources included in the Grid will inevitably produce increased application complexity, due to tighter integration of techniques that are common today, such as visualization and simulation, and the addition of new technologies, such as tele-immersion. This complexity will lead to applications constructed using components obtained from—or located at—remote locations. Moreover, these grid applications frequently will have demanding performance requirements, including high computational rates, high data access rates, and stringent real-time requirements. These characteristics have major implications for the technologies used to construct applications. Approaches to *program composition* that can manage complexity while promoting performance and numerical accuracy will be important.

Lack of Appropriate Tools and Techniques Experience shows that existing application development techniques are inadequate for complex, dynamic Grid applications and environments. Applications developed with conventional distributed or parallel computing technologies often suffer from inordinate complexity, poor performance, and extreme fragility in response to changing system configurations or behavior. New approaches to application development that enhance our ability to encapsulate dynamic behavior in reusable components are needed. These are likely to include new problem solving methods, programming models, programming tools, and approaches to reusability and performance optimization.

Collectively, these challenges can be summarized as a single requirement: *We need application development technologies that make it easy to construct and execute applications with reliably high performance in the constantly-changing computing environment of the Grid.* Today's distributed and high-performance computing technologies do *not* satisfy this requirement. Distributed computing technologies will provide critical building blocks and frameworks for Grid application development; however, distributed computing is not concerned with, and does not address, the stringent performance requirements, large resource needs, and multidisciplinary nature of Grid applications. High-performance computing is a useful source of performance-oriented technologies, but does not address the dynamism of the Grid environment.

1.3 The GrADS Approach

Effective application development for Grid environments requires a new approach to the design, implementation, execution, and optimization of applications. A new approach is needed because the traditional development cycle of separate code, compile, link, and execute stages assumes that the properties of underlying resources are static and relatively simple. In the Grid, this assumption is not valid.

To address the complex problems presented by the dynamic nature of the Grid execution problem, are developing a new software architecture that postpones many decisions currently made by the developer, compiler, and linker until run time. The GrADSoft architecture, which follows the vision laid out for the Distributed Computing Support (DisCoS) Environment [12], will break down traditional barriers separating application, library, compiler, linker, and runtime system to allow adaptation to dynamic changes in target configuration and problem structure. This will be accomplished by embedding substantive portions of the functionality of problem-solving environment, compiler, linker, and performance monitor in an enhanced scheduler and runtime system. This approach, described in Section 2 and depicted in Figure 1, will permit a computation to be continuously adapted to changes in the target configuration without intervention by the programmer. It depends on the development of a notion of "performance contracts" that can be used as yardsticks to measure the extent to which application performance is meeting specifications.

This new approach will require an unprecedented degree of integration and interaction among different

system components. Advances in application, algorithms, libraries, compilers, schedulers, and runtime systems must therefore be pursued simultaneously and in concert. These advances must be coupled with investigations of both the theoretical limits to the performance that can be achieved in dynamic environments, and the practical utility and usability of new approaches. This tight coupling among diverse disciplines, along with the long-range nature of the research needed to achieve success, have motivated us to initiate a national-scale project to achieve the stated goals. This project has undertaken a multi-year research program to:

- explore a new, integrated approach to the design and construction of Grid application development software (GrADS), that enables coordinated responses to adaptivity and composition at the runtime, compiler, and library/application levels;
- pursue fundamental research in (1) the base software technologies that support application development tools in a Grid, (2) languages and compilers, and (3) algorithms and libraries, designed to enable and exploit the above integrated approach;
- build prototype systems that allow practical evaluation of this new approach and associated research advances, particularly in collaboration with the NSF PACIs, which will serve as a major motivator for, and consumer of, this research; and
- work closely with industry to transfer key technologies to the commercial sector.

These activities are discussed in more detail in subsequent sections of the proposal.

We anticipate that the successful completion of this research program will lead to revolutionary new ways of utilizing the global information infrastructure as a platform for computation, changing the way scientists and engineers approach problem solving.

2 GrADSoft Architecture

To achieve reliably high performance in a constantly changing environment, the GrADS project will employ a software architecture designed to support adaptation and performance monitoring. The GrADSoft architecture, depicted in Figure 1, replaces the discrete steps of application creation, compilation, execution, and post-mortem analysis with a continuous process of adapting applications to both a changing Grid and a specific problem instance.

2.1 Overview

At the heart of the GrADSoft architecture is an enhanced execution environment, depicted on the right side of Figure 1. This execution system continually adapts the application to changes in the Grid resources, with the goal of maintaining overall performance at the highest possible level. Two key concepts are critical to the working of this system. First, an application must be encapsulated as a *configurable object program*, which can be optimized rapidly for execution on a specific collection of Grid resources. Second, it relies upon a system of *performance contracts* that detail the expected performance of modules as a function of available resources. These performance contracts include the criteria needed to guide the run-time environment in configuring the object program to available resources and in deciding when to interrupt execution and reconfigure to achieve better performance. In short, the run-time system will dynamically adapt the application to a changing Grid environment via a closed execution loop.

The key to this dynamic adaptation is deep information sharing among all components of the GrADSoft infrastructure, including PSEs, libraries, compilers, and tools. The *performance contract* is the primary interface for this sharing, by making the resource needs and performance characteristics of individual modules and collections of modules available. The GrADSoft compilers analyze systems of modules, produce reconfigurable object programs, and annotate them with resource and performance information. To run a reconfigurable object program, the scheduler invokes the dynamic optimizer and passes it information about the actual runtime configuration. The dynamic optimizer uses the program's annotations, the results of analysis by the GrADS compiler, and knowledge of the runtime configuration to tailor the code to the specific execution environment.

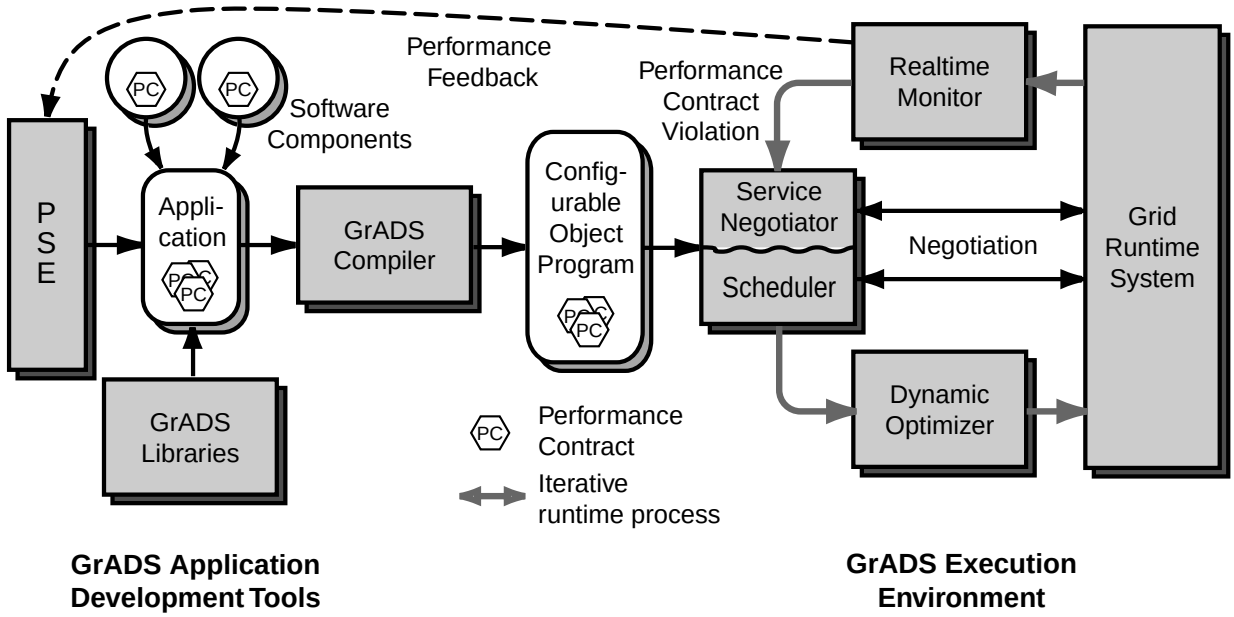

Figure 1: GrADS Program Preparation and Execution Architecture

In brief, the performance contracts in the GrADS system will serve as the basis for collaboration and information sharing among PSEs, static and runtime compilers, service negotiators, and performance monitors. These contracts create a closed loop system of adaptive control and optimization. This control system will work to ensure that performance goals are met by detecting performance anomalies during execution and initiating reorganization and reoptimization of computations to remove them.

2.2 GrADS Application Development Tools

The left side of Figure 1 depicts the tools used to construct configurable object programs. We expect that most application developers will use high-level problem solving environments (PSEs) to assemble Grid applications from a toolkit of domain-specific components. Another path allows developers to build the specialized components that form these PSE toolkits (*e.g.,* a library for solving PDEs on computational grids) or to create new modules for their specific problem domain.

In either scenario, modules are constructed using derivatives of standard languages with Grid-specific extensions (*e.g.,* data or task distribution primitives). Modules are bound together into larger components, libraries, and applications with a coordination language.This process creates malleable modules, annotated with information about their resource requirements and predicted performance for a wide variety of resource configurations.

In addition to supporting the closed-loop execution environment, the realtime monitor provides feedback about system performance to the user, through realtime displays, and to the program development tools, through stored annotations. In response, the user can steer program behavior, advise the service negotiator about preferred execution sites or resources, or recommend reoptimization of specific components. The GrADS compilers and PSEs use stored performance data to tailor recompilations.

The goal is to provide tools that will free the user from many of the low-level concerns that programming for the Grid entails today, permitting greater focus on the high-level design and tuning of programs for a heterogeneous distributed computing environment.

2.3 GrADS Execution Environment

When a configurable object program is delivered to the execution environment, the GrADSoft infrastructure must first determine available resources and secure them for the application. Using annotations from the

performance contracts and the results of static analysis by the various compilers, service negotiators fulfill contract terms by brokering the allocation and scheduling of module components on Grid resources. Next, the GrADSoft infrastructure will dynamically select a precompiled instance of the module and invoke the dynamic optimizer to tailor the reconfigurable object program for better performance with the available resources. During program execution, the realtime monitor tracks program behavior and validates observed behavior against performance contract guarantees.

Should a performance contract be violated, the runtime system initiates contract renegotiations, spurring several possible actions. It can invoke the dynamic optimizer with more information (from performance monitoring) to improve program behavior in the current execution context. It can negotiate a new execution context where the existing executable is more likely to satisfy the old contract. It can do both, creating a new context and a newly tailored executable. Dynamic forecasts of resource performance and Grid capacity are used to reduce renegotiation overhead.

The goal of this closed loop system is to ensure that execution of the application proceeds reliably, meeting the specifications of its performance contracts, in the constantly changing Grid environment.

3 Motivating Examples

Before discussing specifics of the research, we will describe three example applications to illustrate the capabilities that we are developing. These examples are representative of three major classes of Grid application: coupled systems, real-time data analysis, and large-scale distributed simulation. Each already exists in a grid-enabled form, albeit in an inflexible, hand-coded form. *GrADS research will allow each of these applications to provide reliable high performance in arbitrary rather than pre-selected environments.*

3.1 Coupled Systems

The Linear Systems Analyzer (LSA) is a problem solving environment that has been used for such applications as the interactive steering of an industrial mold filling simulation [19]. Using a control panel, a user can construct a coupled system comprising a parallel simulation, an interactive visualization client, and one or more "steering" modules that dynamically select and tune parallel linear algebra solvers as the application executes. A Globus-based prototype supports the placement of individual components on different computers, but requires manual configuration.

GrADS technologies will allow LSA to operate in a much wider range of Grid environments. The current system requires manual resource selection and configuration for each coupled system defined by a user. A GrADS-based implementation will use compiler-generated performance specifications for both components and the coupling framework to guide the location, selection, and configuration of computers and networks. Specifications will allow this selection and configuration process to optimize variously for absolute performance, cost, and reliability. These same specifications can be used in conjunction with GrADS-supplied tunable Grid libraries to select and configure appropriate linear algebra solvers on the basis of computational and numerical performance characteristics. Once the program is running, application performance requirements (expressed in terms of a contract) can be monitored automatically, and application-level code invoked if computer, network, or algorithm performance degrades below acceptable limits.

3.2 Real-Time Data Analysis

This application uses distributed resources for interactive and collaborative analysis of data from a real-time source. The specific application that we consider is an image processing pipeline that takes data from either a beamline at the Advanced Photon Source (a high-brilliance synchrontron X-ray source) or an electron microscope. It uses multiple processors to preprocess this data and then to reconstruct the image. Finally, it passes the resulting image data to multiple remote users for collaborative viewing, analysis, and perhaps remote steering of the scientific instrument [43]. The application was implemented using Globus. As with LSA, it relies on dedicated resources and extensive manual configuration to appropriately balance requirements for both accuracy and real-time response.

GrADS technologies will allow development of a next-generation system that supports true online reconstruction and remote steering of data acquisition. We envision this system operating as follows. The user will provide the code for individual pipeline elements and a set of performance requirements, expressed in terms of high-level concepts such as reconstruction fidelity, real-time response, reliability, and cost. GrADS

compilers, libraries, and service negotiators will then work together to map these requirements into resource requirements, to establish appropriate performance contracts with individual resource elements, and to configure both individual components and the pipeline as a whole. During execution, online monitoring of experimental parameters and resource properties may lead to renegotiation of requirements and contracts. Similarly, the user might request higher reconstruction fidelity or the forecasting subsystem might suggest that the quality of a particular resource may soon be degraded. Renegotiation may lead to a migration to new resources, changes in reconstruction parameters, a reconfiguration of the pipeline, or a switch from best-effort to guaranteed network service, with the actual strategy employed depending on both resource availability and the high-level policies established by the user.

3.3 Large-Scale Numerical Simulation

This example uses multiple networked computers (typically supercomputers) to perform very large-scale numerical simulations. The specific application that we consider is Cactus, a modular code constructed at NCSA, Washington University, and the Max Planck Institute in Potsdam. Cactus solves strongly gravitating astrophysical systems, such as neutron stars and black holes. The problems addressed by Cactus are so large, and the computational complexity so high, that the coupling of multiple computers on the Grid is both desirable and feasible. A Globus-based version of Cactus has been constructed. Early experiments have used it to connect three T3Es, two in Germany and one in San Diego [4]. These experiments established feasibility; performance and automated configuration remain for the future.

GrADS technologies will help Cactus developers automate tasks such as determining required computational resources, locating those resources in a network, decomposing the problem domain across computers, selecting algorithms, and organizing communications. GrADS compiler technologies can be used to determine computational costs associated with Cactus components, while communication operations are encapsulated in grid-aware GrADS libraries. A service negotiator can be used to encapsulate the logic used to adapt computation structure when adaptive mesh refinement occurs: for example, by acquiring new resources, adjusting computation parameters, or even recompiling the code for a different host. Ultimately, our hope is that a Cactus user should be able to negotiate with the GrADS runtime system to determine an appropriate combination of solution accuracy, run time, and cost, and then have all other tasks required to perform the computation be handled automatically by the GrADS-based Cactus implementation.

We have been using these three example codes to guide our design and development efforts. The experience gained with preliminary implementations will influence the design of our first generation of tools—the runtime systems, the languages and environments, the algorithms and libraries, and the testbeds. We will reimplement the examples using the first generation tools. This will provide us with valuable insight into issues of usability, suitability, and effectiveness. Finally, we will distribute our implementations of these codes as worked examples to help other application developers understand the tools and techniques used to construct them. The next four sections describe these four major research thrusts: runtime support, languages and environments, algorithms and libraries, and testbeds.

4 Runtime Support for Grid Applications

The GrADS research program focuses on integrated application development rather than on the underlying network and computational infrastructure from which the Grid is constructed. (For infrastructure, we rely upon existing and proposed networks, such as vBNS and Internet-II.) However, while underlying networks define basic communication protocols, the goals of the GrADS research program require that these basic protocols be augmented with a variety of higher-level application-oriented services. These services both define the runtime environment in which applications run and serve as the target for application generation tools such as compilers, linkers and libraries.

The runtime support services required for GrADS application development can be partitioned into two general classes. A first set are concerned with such concerns as resource management, resource location, resource scheduling, and security (*i.e.,* the basic capabilities required to execute applications in Grid environments). For these we will rely on the services provided by Globus system [17], with various enhancements motivated by the specific requirements of the GrADS framework.

A second set of services, to be developed as part of our research, is concerned more specifically with the information infrastructure and event notification capabilities required to support the development of

grid-aware applications. A *grid-aware application* is one that at runtime can identify Grid characteristics and then dynamically reconfigure resource requirements and/or application structures to maintain desired application-level performance. As noted above and emphasized in our application examples, we view this concept as being critical to the successful use of Grid environments.

Experience with the development of prototype grid-aware applications (*e.g.,* [5, 39, 41]) leads us to believe that robust application performance can only be achieved if *all* system components are allowed to advertise their performance requirements and expectations to other elements of the system, and then react if those requirements are not being met. Thus applications and GrADS components must be able to push performance specifications into the runtime system as well as pull dynamic performance information from the run-time system so that scheduling and resource control mechanisms can make effective decisions based on application and user requirements.

It is this focus on information flow and performance infrastructure that motivates the second set of services referred to above. These services will comprise three principal components:

- A *comprehensive grid information service* will provide uniform access to information about the structure and state of grid resources. The Globus Metacomputing Directory Service (MDS) [16] and the Network Weather Service (NWS) [45] provide proof-of-principle prototypes for certain elements of this infrastructure, but its scope needs to be expanded dramatically.

- A general *performance contract framework* will support the negotiation of requirements and expectations. Again, experience with existing technologies such as Globus resource brokers [11] and AppLeS schedulers [5] provide some confidence that such a framework can be established.

- A *monitoring and event notification architecture* will support the notification of relevant system and application components when resource and/or application component characteristics exceed specified bounds. The Autopilot distributed performance monitoring and adaptive control system [37] embodies several of the ideas on which we will build our distributed event notification and monitoring systems.

These infrastructure elements will provide the foundation on which can be developed the other GrADS components, such as grid-aware compilers, grid-aware libraries, and resource negotiators.

4.1 Research Directions

We are conducting the following key research activities to address the infrastructure requirements of GrADS components:

Performance Contracts The creation of modular system components that are efficient in dynamic Grid environments will require new interface specification and assertion mechanisms. These mechanisms must be able to match the resource needs of modules with deliverable performance and capabilities of resources. We have proposed the *performance contract* as a vehicle for sharing complex, multi-dimensional performance information between application and system components, enabling a dynamic negotiation process to occur between resource providers and consumers. A performance contract may involve performance goals, cost limits, scheduling constraints, and other requirements. These elements may be specified by the application or derived by other GrADS components. We are developing a calculus of performance-contract negotiation and study the use of dynamic contract negotiation to create interoperable applications, libraries, compilers, and runtime systems. In addition, a *service negotiator* is being developed to orchestrate interactions among upper-level components and the underlying system, as dictated by their performance contracts.

Runtime Support for Adaptive Strategies The dynamic performance characteristics of Grid environments requires new strategies for achieving application performance. GrADS components and tools must be able to support both the development and execution of adaptive "Grid-aware" applications. Such applications will require performance information that depends on their function and structure, as well as the time-frame in which they will execute. New strategies must be developed so that the compiler, scheduler, runtime system, and other GrADS components cooperate to extract pertinent information from Grid applications non-intrusively, and can feed it back to the application level as necessary.

Development of a Performance Economy The development of performance contracts and a service negotiation framework will result in a system in which all components can participate in a virtual "performance economy." The development of frameworks to support the negotiation of performance contracts for GrADS applications will be critical to the efficient functioning of the GrADS system. This work will investigate the development of performance economies and the way in which they can be structured to optimize the behavior of GrADS applications.

5 Programming Languages and Environments

The goal of the languages and environments research is to discover methodologies and frameworks that make it easy to build applications for the dynamic environment of the Grid. The challenges that must be overcome include (1) development of abstract programming models for Grid software development that are both powerful and flexible, (2) design of an implementation strategy that maps those programming models to the Grid computational environment in a fashion that permits adaptation to changing configurations and resources, and (3) design and development of performance monitoring and control systems that can recognize performance problems as they arise and reconfigure applications to improve efficiency.

Our efforts are focusing on two different programming-model development strategies, each appropriate to a different class of software developers. First, we expect the majority of practicing scientists and engineers to prefer building programs from pre-written, domain-specific components through advanced *problem-solving environments* (PSEs) (*e.g.,* like LSA [19], Khoros [28], SciRun [9] and NetSolve [9]). These PSEs are designed to shield the user from the complexity of Grid computing via easy-to-use interfaces for specifying component interactions. Moreover, the graphical nature of many of these PSE interfaces [21, 31, 9] simplifies module composition and integration.

Second, we will provide support for developers building the specialized software components that form the toolkits exploited by one or more PSE systems (*e.g.,* a module or library to solve PDEs or linear algebra problems on a scalable metacomputing platform [20, 9]). These components will typically be written using derivatives of standard languages — Fortran or Java — but with extensions for designing Grid programs.

In either case, the role of the development tools, compiler, and runtime system is to prepare the program for high-performance execution on whatever computing configuration is available when the program is initiated. This is complicated by the dynamic nature of the Grid's execution environment. To achieve high performance, the compilation process must be redesigned so that compilation is an evolutionary process, with some parts occurring immediately before execution or during execution, in response to changing resource availability.

For example, in the GrADS design shown in Figure 1, the *dynamic optimizer* is invoked after the available resources, such as the size, shape, location, and nature of the target configuration, are known. If performance anomalies are detected at runtime, the scheduler and dynamic optimizer can be reinvoked to produce a different runtime configuration. When invoked in the midst of execution, the dynamic optimizer may have knowledge that was not available any earlier, such as the size and location of data structures. This strategy for addressing dynamic runtime behaviors is essential if the system is to satisfy the specified application performance contracts. The compiler is aided in this task by the annotations provided by the library designer (see the next section on libraries). These annotations, when combined with parameterized libraries, compiler-directed performance instrumentation, and the runtime system performance notification described in the previous section, give the compiler more information to exploit in optimization.

5.1 Research Directions

To achieve this vision for grid program development and execution, a complex set of research challenges will need to be addressed.

Compilation and Execution Architecture Designing a programming system architecture that supports the dynamic compilation strategy depicted in Figure 1 is a major goal of our languages and environments effort. Previous research on runtime compilation has focused on single computer optimization problems [3, 34], but work related to dynamic optimization for high-performance distributed applications is only now emerging [38, 8, 15]. The system of performance contracts at the heart of this proposal will make it possible for real-time monitoring systems to cooperate with runtime compilers, ensuring that performance goals are

met by detecting performance anomalies during execution and initiating reorganization and optimization of computations to remove them [1].

Real-time Monitoring and Adaptive Control The transient, rarely repeatable behavior of the Grid means that the standard model of post-mortem performance optimization must be replaced by a real-time model that optimizes application and runtime behavior during program execution. This model of closed loop control will require design of new performance specification interfaces and assertion mechanisms, measurement tools, and decision procedures for adaptive configuration of application modules and runtime system components [35, 36], as well as compiler interfaces for just-in-time compilation and performance prediction. Via runtime validation of these performance contracts, the real-time measurement and control system will activate dynamic recompilation and runtime resource configuration,

Efficient PSE Implementation A typical strategy for PSE implementation would provide a collection of pre-packaged library components, along with a (possibly) graphical scripting language for module composition. For example, these may be scientific library modules that have been encapsulated in an object framework that may be based on DCOM, Enterprise Java Beans, or the proposed CORBA 3.0 components [32]. We begin by recompiling and relinking them with the GrADS libraries. To achieve high performance in this implementation, we will generate a control program and optimize the entire resulting problem solution—control program plus components—as a single module. To accomplish this, the program execution environment must not only optimize the pairwise communication between computations [8, 27, 23], it must also provide a global strategy for introducing additional concurrency to hide latency or smooth scheduling jitter. This task is greatly simplified if many of the individual components are parameterized in the manner of the library components described in the next section. The individual computational components must also support automatic checkpoint, cloning, and restart at other computational sites, as the resources assigned to them become overloaded or simply cease to be available. In all cases, the ability to dynamically decompose, recompile and reconfigure an application is critical.

Language Strategies for Reusable Distributed Programs To present the component developer with a simple but powerful model for Grid programming, including an abstract view of data structure distribution [24] and load balancing, we are exploring expression of data distribution through user-programmable access routines. This approach could make it possible for the user to implement data distribution explicitly while relying on language, library, and compiler mechanisms to hide the implementation details. Thus, the scientist or engineer could develop libraries of distributed data structures and use them as if they were language intrinsics, amenable to all of the compiler's extensive optimizations.
Enabling the construction of applications that execute on the Grid poses challenges that have not been encountered in traditional program development environments. Consequently, we believe it is critical to explore a radically new approach to program preparation and execution. The proposed GrADS system, illustrated in Figure 1, integrates language processing, library design, and middleware for PSEs to form a comprehensive framework for designing and implementing Grid applications.

5.2 Research Strategy

Over the lifetime of the project we will need to address and solve a number of critical problems that are linked to one another. To guide us through this process, we have begun a detailed analysis of the applications, described in Section 3 that will drive much of our thinking. Our principal goal is to (a) determine how applications of this character could be made reconfigurable and (b) provide programming systems that would make them easier to develop for the Grid. This experiment will be followed in later years by implementations of the same applications using the prototype GrADS software development framework. This analysis will drive out thinking in several key areas, described in the following paragraphs.

Reconfigurable Object Programs and Dynamic Optimization A central goal of the research is to define a standard for reconfigurable object programs and construct a dynamic optimizer that will tailor them to a specific target configuration. The reconfigurable object program should contain all the information necessary to tailor it in the dynamic optimizer. To improve dynamic optimizer performance, the reconfigurable object program may include custom analyzers and transformers generated by the static compiler. A second goal is to generate information sharing and data gathering interfaces that will assist the performance

monitoring system in detecting performance problems that must be addressed through reconfiguration. This work is being done in close collaboration with the library effort which will use the same object program and data gathering framework for the encoding of the parameterized algorithm libraries.

Static Compilation Framework We will construct an interprocedural framework for analyzing whole programs and generating reconfigurable object programs. The heart of the system will be a language-independent framework for managing the compilation of Java and Fortran programs. This framework will support both intelligent integration of library calls and global performance optimization. Our intent is to simplify the task of adding additional languages to the framework. Our own efforts will be directed toward the support of Java and Fortran. The output of the static compiler will be a reconfigurable object program, which will be further optimized dynamically when the target configuration is known.

Performance Optimization Toolkit We will develop a toolkit for static and dynamic performance tuning that is integrated with the static compilation framework, dynamic optimization and just-in-time compilation systems, object libraries, and PSEs. Based on deep compile-time information, runtime validation of performance contracts, and adaptive control techniques, the performance toolkit will enable users to develop codes that meet performance design goals even when resource availability changes dynamically. The early research is concentrating on the definition of performance contract interfaces and validation mechanisms. We expect dynamic, adaptive performance measurement and control to follow in later years.

Problem-Solving Environments Our goal is to enable the construction of application-specific problem solving environments that allow users reliable, transparent access to vast computational resources and remote information sources. These PSEs should present a simple, but high level programming model that allows applications to be built from reconfigurable library and data-structure components. The PSE also provides a platform where the user can negotiate performance contracts for program execution based on requirements derived from the users view of application. It is up to the PSE system to translate these application-level specifications into the lower-level requirements that can be used by the compiler and runtime to meet the users goals.

6 Algorithms and Libraries

The primary goals of our effort in algorithms and libraries are (1) to develop a new generation of algorithms and software libraries needed for the effective and reliable use of (wide area) dynamic, distributed and parallel environments, and (2) to validate the resulting libraries and algorithms on important scientific and engineering applications. To consistently obtain high performance in Grid environments will require advances in both algorithms and their supporting software. These same advances may help with other advanced computing environments, including distributed systems, heterogeneous systems, clusters of workstations, distributed shared memory machines, and the proposed petaflops architectures, will require advances in algorithms and supporting software. Some of the challenges in this arena have already been encountered. For example, to make effective use of current high-end machines, the software must manage both communication and the memory hierarchy. This problem has been attacked with a combination of compile–time and run–time techniques. On the Grid, the increased scale of computation, depth of memory hierarchies, range of latencies, and increased variability in the run–time environment will make these problems much harder.

To address these issues, we must rethink the way that we build libraries. The issues to consider include software architecture, programming languages and environments, compile versus run–time functionality, data structure support, and fundamental algorithm design. Among the challenges that we must deal with are:

- Library software must support performance optimization and algorithm choice at run time.
- The architecture of software libraries must facilitate efficient interfaces to a number of languages, as well as effective support for the relevant data structures in those languages.
- New algorithmic techniques will be a prerequisite for latency tolerant applications.
- New scalable algorithms that expose massive numbers of concurrent threads will be needed to keep parallel resources busy and to hide memory latencies.

6.1 Research Directions

These considerations lead naturally to a number of important areas where research in algorithm design and library architecture is needed.

Grid-Aware Libraries To enable use of the Grid as a seamless computing environment, we will develop parameterizable algorithms and software annotated with performance contracts. These annotations will help the dynamic optimizer tune performance for a broad range of architectures. This tuning will in many cases be accomplished by having the dynamic optimizer and run-time system provide input parameters to the library routines that will enable them to make a resource-efficient algorithm selection. We will also develop new algorithms that use adaptive strategies by interacting with other GrADS components. For example, libraries will incorporate performance contracts for dynamic negotiation of resources, as well as runtime support for adaptive strategies to allow the compiler, scheduler, and runtime system to influence the course of execution. We will use the "Grid information service" and the "Grid event service" to obtain the information needed to guide adaptation in the library routines.

Latency-Tolerant Algorithms Remote latency is one of the major obstacles in achieving high efficiency on today's high-performance computers. The growing gap between the speed of the microprocessors and memory coupled with deep memory hierarchy implies that the memory subsystem has become a large performance factor in modern computer systems such as the DOE ASCI computers [6, 42]. In the unpredictable and dynamic world of the Grid, this problem will be even worse. Research into latency tolerant algorithms that explore a wider portion of the latency/bandwidth/memory space is needed. Furthermore, tools are needed for measuring and managing latency. We plan to design and construct libraries that are parameterized to allow their performance to be optimized over a range of current and future memory hierarchies, including the ones expected in computational Grids.

Compiler-Ready Libraries In the past, library development has typically focused on one architecture at a time with the result that much of the work must be repeated to migrate the code to a new architecture and its memory hierarchy. We are exploring the design and development of parameterized libraries that permit performance tuning across a broad spectrum of memory hierarchies. Typically, developers of portable libraries rely on HPC Compilers to analyze and parallelize their programs. These compilers are large and complex; they do not always discover parallelism when it is available. Automatic parallelization may be adequate for general scientific programming, but experts writing libraries find it frustrating, because they must often perform tedious optimizations by hand—optimizations that could be handled by the compiler if it had a bit of extra information from the programmer. Programmers may find it both easier and more effective to annotate their code with information that will help the compiler generate the desired code with the desired behavior. We have begun to identify opportunities where information about algorithms contained in library functions can help the compiler and the run–time environment[26] and work with the GrADS compiler group to develop a new system of annotation to provide information about semantics and performance that aids in compilation. At the library interface level, this would include memory–hierarchy tuning parameters and semantic information, such as dependency information to make it possible to block LU with pivoting, and floating–point semantic information (for example, to indicate that it is acceptable to reorder certain floating point operations or to handle exceptions in particular ways)[44]. The goal is to make it possible to "build in" to the compiler knowledge about these libraries far beyond what can be derived by a compile-time analysis of the source.

Sparse Linear Algebra Many modern applications rely on algorithms and implementations for sparse linear algebra, which can benefit from new direct and iterative methods, packaged in object libraries that make them easy to use. There has been a great deal of recent progress in understanding how to exploit memory hierarchies for efficiency [14, 2] as well as special problem structures (domain decomposition, geometric and algebraic multigrid). In addition to generic libraries, this work should be closely tied to particular Grid applications to drive development.

Fast Hierarchical Algorithms and DSP Fast hierarchical algorithms and structured DSP algorithms [25] are also undergoing rapid development. There is a large class of fast, hierarchical algorithms, of which the Fast Multipole Method is perhaps the best known example, that vary slightly from application to application

and are complicated to implement. Grid-aware implementations of these techniques need to be developed and incorporated into the PSE library framework. In addition, DSP applications will require library routines to support DSP applications, particularly those that use techniques from FFTs. These routines, updated to exploit recent algorithmic improvements, will also be needed in adaptable form for use on the Grid.

7 Testbeds

Programming grid applications is extremely challenging as the dimensions of heterogeneity, wide-area distribution, dynamicity, adaptation, and resource failure are all integral parts of the execution environment. Developing robust advances in technology which enable Grid resources to be useful for scientists and engineers require a range of testbeds which support exploration and experimentation, supporting both the science of understanding how to build robust adaptive techniques and the engineering of how to build large scale grid systems.

The GrADS effort is pursuing a two-level approach to testbeds – *MicroGrid* studies and *MacroGrid* Studies. MicroGrid studies involve a controllable environment, built via simulation on a single computer (or a small number) of computers to model the complexities of grid environments. MicroGrid studies can be controlled experiments of the effectiveness and dynamic range of grid software. These issues can then be systematically studied across a range of network, adaptation, and resource environments. MicroGrid studies will generally be pursued with partial-scale applications. MacroGrid studies will generally involve specialized resources, wide-area networks, and full-scale applications. Because of limited available resources and reasonable economic concerns about the effective utilization of such resources, the quantity of MacroGrid experimentation is necessarily limited. We believe both types of experimentation are essential, and the greatest benefits will accrue from a coordinated approach of experimentation and validation. We describe the specific advantages and limitations of both types of studies and the tools we are building below.

7.1 MicroGrid Studies

The MicroGrid toolkit will emulate a variety of grid configurations and dynamic behaviors, supporting controlled studies and allowing large numbers of computer scientists and application scientists to experiment systematically with a variety of grid configurations and dynamic resource environments. Such resource-controlled studies enable characterization of Grid software's capability to *adapt* to the conditions of the computing environment. Controlled studies are an essential complement to full-scale deployment of dynamic systems, allowing exploration of system stability, performance stability, and robustness versus a variety of resource management regimens. MicroGrid studies allow exploration of a larger space than typically occurs, providing an opportunity for a grounded understanding of system overload or other unexpected dynamic behavior. In addition, because MicroGrid studies can be more easily instrumented, such studies provide a vehicle for improving our understanding of how to build robust Grid applications and systems. To ensure functional compatibility and realistic grid behavior the MicroGrid environment is likely to run a subset (or base configuration) of the Globus software.

Leveraging Chien's efforts to build High Performance Virtual Machines (HPVM's) [10, 29, 22, 33], we plan to build a MicroGrid toolkit which enable control and systematic exploration of the following parameters of a grid resource space: number of computing nodes, computational power of nodes, network bandwidth, network latency, network loss, job initiation delay, process control, scheduler dynamics, and resource failure. While these tools cannot hope to support the full diversity of heterogeneity which will exist in the actual Grid, they will allow experimenters to emulate "virtual grids" which capture the dynamicity of resources which is one fundamental challenge of robust Grid software. Specifically, these tools will be used to provide a controllable environment, understand potential system dynamics, perform robustness studies, and otherwise explore the performance space of adaptive software. As appropriate, development of canonical configurations to stress software, as well as techniques of randomization will be explored to exercise grid software. Note that the MicroGrid toolkit should not be confused with GrADSoft (which includes Globus), the Grid software through which application programs will manage the complexities of a grid resource environment. In the MicroGrid toolkit a scientist only controls the behavior of an *emulated environment*.

Thus, the MicroGrid studies enable the science of building robust application and tools software for the Grid by supporting large numbers of controlled experiments. The MicroGrid environment will be widely disseminated to the community to enhance research in building Grid software. Within the GrADSoft project,

Chien's group at UCSD is assembling a 128-processor cluster coupled with several multigigabit networks and the UIUC Department of Computer Science is assembling a 512 processor cluster. A fraction of these resources will be made available for large MicroGrid experiments.

7.2 MacroGrid Studies

Experiments on the *MacroGrid*—the large-scale, wide-area grid environments on which numerous grid applications will run—are required both to validate MicroGrid results and to explore configurations that cannot be simulated effectively in a MicroGrid environment.

GrADS PIs are already working with the NSF Partnerships for Advanced Computational Infrastructure (the Alliance and NPACI), NASA, and DOE to build large-scale prototype grids: for example, the Globus Ubiquitous Supercomputing Testbed Organization (GUSTO) spans some 40 institutions worldwide and incorporates major supercomputers, large storage systems, and specialized resources such as scientific instruments and high-end virtual reality systems. We plan to perform large "at scale" experiments in these environments and, in addition, to explore the use of virtual private network (VPN) technology to construct persistent testbeds in which semi-controlled experiments are possible. Research networks such as CAIRN and components of NGI will provide another source of MacroGrid testbeds.

8 Summary

The GrADS project has established an effort to pioneer technologies that will be needed for ordinary scientific users to develop applications for the Grid. These technologies will include a new program preparation framework and an execution environment that employs continuous monitoring to ensure that reasonable progress is being made toward completion of a computation.

The GrADS research and development activities are organized as three parallel, interdependent thrusts: basic research, testbed development, and application evaluation. The basic research thrust will begin by defining performance contracts, exploring adaptivity (both experimentally and theoretically), and creating initial prototypes of the GrADS development and execution environment. With knowledge gleaned from this work, the research thrust will then explore an integrated approach to Grid software development that emphasizes compile-time and runtime information sharing among algorithms, compilers, tools, and libraries.

The testbed development thrust will create a GrADS software toolkit (GrADSoft) that will provide a basis for experimental verification of our ideas and for technology transfer. Exploiting the core infrastructure provided by Globus and software components from our AppLeS, NWS, Autopilot, NetSolve, D95, and scalar compiler systems, the GrADSoft prototype will bring together an increasingly sophisticated set of languages, libraries, compilers, schedulers, service negotiators, and performance tools.

Finally, in concert with our PACI, ASCI, and other partners, the evaluation thrust will use the emerging GrADSoft prototype to develop and assess at least one compelling, Grid-enabled application. This evaluation will couple the basic research and testbed efforts and provide a blueprint for a powerful technology transfer mechanism for the GrADS project and possible extensions thereof.

In brief, the research, testbed, and application evaluation thrusts will create a tight cycle of exploration, development, and experimental validation that focuses research on problems that are both important and practical. The milestones below emphasize the interdependence of the three thrusts, leading to a GrADSoft prototype that will catalyze new Grid applications.

Acknowledgments

The authors would like to thank Frederica Darema, our NSF program manager for her constant support for our effort and her intellectual contributions to the ideas underlying the GrADS project. In addition, we would like to thank the other members of the GrADS project, including ??, who have contributed to the development of the ideas in this paper.

References

[1] V. Adve, J.-C. Wang, J. Mellor-Crummey, D. Reed, M. Anderson, and K. Kennedy. An Integrated Compilation and Performance Analysis Environment for Data–Parallel Programs. In *Proceedings of Supercomputing '95*, San Diego, California, December 1995.

[2] P. Amestoy, M. Daydé, I. Duff, and P. Morére. Linear Algebra Calculations on a Virtual Shared Memory Computer. *International Journal of High Speed Computing*, 7(1):21–43, March 1995.

[3] J. Auslander, M. Philipose, C. Chambers, S. Eggers, and B. Bershad. Fast, Effective Dynamic Compilation. In *Proceedings of the 1996 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 149–159, Philadelphia, Pennsylvania, May 1996.

[4] W. Benger, I. Foster, J. Novotny, E. Seidel, J. Shalf, W. Smith, and P. Walker. Numerical Relativity in a Distributed Environment. (To appear in *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, March 1999.).

[5] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application Level Scheduling on Distributed Heterogeneous Networks. In *Proceedings of Supercomputing '96*, Pittsburgh, Pennsylvania, November 1996.

[6] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, 1997.

[7] L. S. Blackford, A. Cleary, J. Dongarra, A. Petitet, R. Whaley, J. Demmel, I. Dhillon, H. Ren, K. Stanley, and S. Hammarling. Practical Experience in the Numerical Dangers of Heterogeneous Computing. *ACM Transactions on Mathematical Software*, 23(2):133–147, June 1997.

[8] F. Breg, S. Diwan, J. Villacis, J. Balasubramanian, E. Akman, and D. Gannon. Java RMI Performance and Object Model Interoperability: Experiments with Java/HPC++ Distributed Components. In *Concurrency: Practice and Experience, Special Issue from the Fourth Java for Scientific Computing Workshop*. John Wiley and Sons, 1998.

[9] H. Casanova, J. Dongarra, C. Johnson, and M. Miller. Application–Specific Tools. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 159–180. Morgan Kaufmann, 1998.

[10] A. Chien, S. Pakin, M. Lauria, M. Buchanan, K. Hane, L. Giannini, and J. Prusakova. High Performance Virtual Machines (HPVM): Clusters with Supercomputing APIs and Performance. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, Minnesota, March 1997. (Also available electronically via `http://www-csag.cs.uiuc.edu/papers/hpvm-siam97.ps`.).

[11] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proceedings of the Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[12] F. Darema. A Distributed Computing Support (DisCoS) Environment. (Available electronically via `http://www.ccic.gov/cicrd/pca-wg/hecc.html`.).

[13] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I–WAY: Wide–Area Visual Supercomputing. *The International Journal of Supercomputer Applications and High Performance Computing*, 10(2):123–130, Summer/Fall 1996.

[14] J. Demmel, J. Gilbert, and X. Li. An Asynchronous Parallel Supernodal Algorithm for Sparse Gaussian Elimination. (To appear in *SIAM Journal on Matrix Analysis and Applications*.).

[15] E. Eide, K. Frei, B. Ford, J. Lepreau, and G. Lindstrom. Flick: A Flexible, Optimizing IDL Compiler. In *Proceedings of the 1997 ACM SIGPLAN Conference on Programming Language Design and Implementation*, Las Vegas, Nevada, June 1997.

[16] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High–Performance Distributed Computations. In *Proceedings of the Sixth IEEE Symposium on High–Performance Distributed Computing*, pages 365–375, August 1997.

[17] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[18] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[19] D. Gannon, R. Bramley, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Breg, S. Diwan, and M. Govindaraju. Developing Component Architectures for Distributed Scientific Problem Solving. *IEEE Computational Science and Engineering*, 5(2), April/June 1998.

[20] D. Gannon and A. Grimshaw. Object–Based Approaches. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 205–236. Morgan Kaufmann, 1998.

[21] G. Geist, J. Kohl, and P. Papadopoulos. CUMULVS: Providing Fault Tolerance, Visualization, and Steering of Parallel Applications. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):224–235, Fall 1997.

[22] L. Giannini and A. Chien. A Software Architecture for Global Address Space Communication on Clusters: Put/Get on Fast Messages. In *Proceedings of the Seventh IEEE Symposium on High–Performance Distributed Computing*, 1998. (Also available electronically via `http://www-csag.cs.uiuc.edu/papers/hpdc7-giannini.ps`.).

[23] A. Gokhale and D. C. Schmidt. Principles for Optimizing CORBA Internet Inter–ORB Protocol Performance. In *Proceedings of the 31st Hawaii International Conference on System Sciences*, pages 376–385, Kohala Coast, Hawaii, January 1998.

[24] High Performance Fortran Forum. High Performance Fortran Language Specification. *Scientific Programming*, 2(1–2):1–170, Spring 1993.

[25] Y. Hu and S. L. Johnsson. A Data Parallel Implementation of Hierarchical N–body Methods. *International Journal of Supercomputer Applications*, 10(1):3–40, 1996. (Also available as TR-26-94, Harvard University, Division of Applied Sciences, September 1994.).

[26] S. L. Johnsson and K. K. Mathur. High Performance, Scalable Scientific Software Libraries. In *Portability and Performance for Parallel Processing*, pages 159–208. John Wiley & Sons, Ltd., Chichester, Uuited Kingdom, 1994. (Also available as TR-19-93, Harvard University, 1993.).

[27] K. Keahey and D. Gannon. PARDIS: A Parallel Approach to CORBA. In *Proceedings of the Sixth IEEE International Symposium on High–Performance Distributed Computing*, August 1997.

[28] Khoral Software. *Khoros Pro, Version 2.2*, 1998.

[29] M. Lauria, S. Pakin, and A. Chien. Efficient Layering for High Speed Communication: Fast Messages 2.x. In *Proceedings of the Seventh IEEE Symposium on High–Performance Distributed Computing*, 1998. (Also available electronically via `http://www-csag.cs.uiuc.edu/papers/hpdc7-lauria.ps`.).

[30] P. Lyster, L. Bergman, P. Li, D. Stanfill, B. Crippe, R. Blom, C. Pardo, and D. Okaya. CASA Gigabit Supercomputing Network: CALCRUST Three–Dimensional Real–Time Multi–Dataset Rendering. In *Proceedings of Supercomputing '92*, Minneapolis, Minnesota, November 1992. (Poster session.).

[31] Numerical Algorithms Group. *IRIS Explorer*, 1997. (Available electronically via `http://www.nag.co.uk/Welcome_IEC.html`.).

[32] R. Orfali and D. Harkey. *Client/Server Programming with Java and CORBA (second edition)*. John Wiley and Sons, 1998.

[33] S. Pakin, V. Karamcheti, and A. Chien. Fast Messages: Efficient, Portable Communication for Workstation Clusters and MPPs. *IEEE Concurrency*, 5(2):60–73, April–June 1997. (Also available electronically via `http://www-csag.cs.uiuc.edu/papers/fm-pdt.ps`.).

[34] R. Ponnusamy, J. Saltz, and A. Choudhary. Runtime Compilation Techniques for Data Partitioning and Communication Schedule Reuse. In *Proceedings of Supercomputing '93*, pages 361–370, Portland, Oregon, November 1993.

[35] D. Reed, C. Elford, T. Madhyastha, E. Smirni, and S. Lamm. The Next Frontier: Interactive and Closed Loop Performance Steering. In *Proceedings of the 1996 International Conference on Parallel Processing Workshop*, pages 20–31, August 1996.

[36] D. Reed and R. Ribler. Performance Analysis and Visualization. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 367–394. Morgan Kaufmann, 1998.

[37] R. Ribler and D. Reed. The Autopilot Performance–Directed Adaptive Control System. In *Proceedings of 11th ACM International Conference on Supercomputing — Workshop on Performance Data Mining: Automated Diagnosis, Adaption and Optimization*, Vienna, Austria, July 1997.

[38] S. Singhal, B. Nguyen, R. Redpath, M. Fraenkel, and J. Nguyen. Building High–Performance Applications and Servers in Java. In *Proceedings of the 12th Annual Conference on Object–Oriented Programming Systems, Languages and Applications*, Atlanta, Georgia, October 1997.

[39] N. Spring and R. Wolski. Application Level Scheduling of Gene Sequence Comparison on Metacomputers. In *Proceedings of the 12th ACM International Conference on Supercomputing*, pages 141–148, Melbourne, Australia, July 1998.

[40] R. Stevens, P. Woodward, T. DeFanti, and C. Catlett. From the I–WAY to the National Technology Grid. *Communications of the ACM*, 40(11):50–60, November 1997.

[41] A. Su, F. Berman, and R. Wolski. Using AppLeS to Schedule a Distributed Visualization Tool on the Computational Grid. In *Proceedings of the 1998 Clusters and Computational Grids Workshop*, 1998.

[42] Thinking Machines Corporation. *CMSSL for CM Fortran, Version 3.1*, 1993.

[43] G. von Laszewski, I. Foster, J. Insley, J. Bresnahan, C. Kesselman, M. Su, M. Thiebaux, M. Rivers, I. McNulty, B. Tieman, and S. Wang. Real–Time Analysis, Visualization, and Steering of Microtomography Experiments at Photon Sources. (To appear in *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, March 1999.).

[44] R. Whaley and J. Dongarra. Automatically Tuned Linear Algebra Software (ATLAS). In *Proceedings of Supercomputing '98*, Orlando, Florida, November 1998.

[45] R. Wolski. Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service. In *Proceedings of the Sixth IEEE Symposium on High–Performance Distributed Computing*, August 1997.