

1 Executive Summary

We propose a project focused on research and development in the area of programming models for scalable parallel computing. Programming models are of paramount importance because they affect both the performance delivered by the computer and the productivity of the programmer seeking that performance. Every programming model presents tradeoffs among performance, portability, expressivity, and convenience, and so no one model fits all purposes. The coming generation of petascale computers exacerbates the problem, since vast numbers of nodes, exotic communication networks, and multicore chips will be the order of the day, requiring advanced implementation techniques for libraries and compilers while the scientific application community begs for simpler, more abstract, and convenient ways of conceptualizing their programs for these machines. Work carried out in this Center will advance the state of the art in the understanding, definition, implementation, and use of models expressed in libraries, languages, and annotations. Our team includes experts in the efficient implementation of widely used communication libraries, language definition experts (particularly for partitioned global address space languages), compiler specialists addressing both these languages and global view languages, and researchers that study programming patterns for high performance computing.

We will carry out research and development at a number of different levels and across a number of different types of programming models. The following are cross-cutting themes in the work we propose.

Productivity. Compiler technology for global view programming models; annotations to hide complexity of distributed data structures in MPI; design patterns for parallel programming. We will explore the implementation of a general, asynchronous work queue model for automated load balancing and effective support for out-of-core computations, in multiple languages and libraries.

Scalability. Compilation for 100K processors and beyond; latency management through function shipping and multithreading. Scaling library approaches such as MPI and GA will require both infrastructure improvements and new programming approaches.

Interoperability. Interoperability of parallel programming models and their implementations. We will ensure that combinations of MPI, Global Arrays, OpenMP, and partitioned global address space languages, where appropriate, have well-defined semantics for interoperability and that our implementations have been tested to exhibit those semantics.

Portability. Implementation of various models that take advantage of the newest features of networks and processors (e.g., InfiniBand RDMA, SMT and multicore processors).

Infrastructure. Enhancements to the runtime system to enable multilevel parallelism, memory management scalable to petascale systems, lightweight and scalable synchronization (processor teams, memory, etc.), multithreading and function shipping.

Performance. High-performance and scalable implementation of various models on the latest and emerging commodity networks/interfaces/protocols and vendor-proprietary networks used in high-end architectures.

Multithreading. OpenMP enhancements to support scientific computing; interoperability of OpenMP and other threading models with MPI; support of multithreaded codes in CAF, UPC, GA, and Titanium, for architectures based on multicore nodes. Basic research in thread safety for and hierarchical parallelism in PGAS languages and GA library.

2 Introduction

A national challenge in high-performance computing is to develop programming models for effectively harnessing the power of “leadership-class” computer systems composed of tens to hundreds of thousands of processors. To address this issue, we propose a program of integrated research by a team of researchers who are experts in compilers, communication libraries, and high-performance cluster interconnects. Our approach is to focus on a relatively small collection of interoperable, complementary technologies that will provide the scientific application developer with a set of tools appropriate to the task.

The desire to simplify the task of developing software for scalable parallel systems has spurred a resurgence of research into language-based parallel programming models as part of DARPA’s High Productivity Computing Systems initiative begun in 2002. This effort includes support for parallelism models beyond the popular message-passing abstraction. For example, for programs with relatively regular data structures, known at compile-time, global view models as expressed in High Performance Fortran (HPF) or OpenMP provide significant productivity advantages because the mapping of parallel work to processors is largely hidden within the compiler and runtime system. With language-based models for parallel programming, a tension exists between abstraction and scalable performance. Applications with nested parallelism and complex dependence patterns may require explicit programmer control over the mapping of parallel work as offered by the message-passing model in MPI [106] and Partitioned Global Address Space (PGAS) languages such as Co-array Fortran (CAF) [125, 126], Unified Parallel C (UPC) [23], and Titanium [173].

PGAS language models have three key strengths. First, they offer higher productivity than library-based programming models by making it easier to express shared data structures and to communicate implicitly and conveniently by reading and writing them. One-sided global address space models relieve developers from the need to orchestrate two-sided communication, which frustrates MPI programmers. Second, with communication and synchronization as language primitives, programs written in PGAS languages are more amenable to compiler-based communication optimization than are MPI programs with library-based communication. Third, like MPI, PGAS languages enable programmers to hand-craft locality-aware parallelizations by providing them with control over placement of data, computation, and communication. One current shortcoming of CAF and UPC is that they fail to provide support for dynamic multithreading. Dynamic multithreading is desirable to exploit hardware-level threading, to support applications with highly dynamic parallelism, and to hide latency. Within this broad space of language concepts there are trade-offs between virtualization of the underlying hardware and control over the hardware resources.

A principal goal of this project is refinement of language-based parallel programming models so that they are expressive (natural for expressing a broad spectrum of algorithms, constructing efficient parallel data structures, and supporting both static and dynamic strategies for decomposing work), productive (programs are as easy to write as possible), high performance (across the spectrum of computing platforms ranging from commodity clusters with multicore processors to leadership-class systems), scalable to leadership-class computer systems with tens to hundreds of thousands of processors, portable, and interoperable with other programming models, as well as program development tools. These programming models must meet the needs of complex irregular applications and support the evolution of existing applications.

Although parallel languages are emerging as practical models for parallel programming, library-based models are the dominant technology in use today. MPI is the de facto standard for programming scalable scientific applications, with its explicit management of communication. The Global Array model [114, 115, 119] enhances the MPI model by supporting a global view of distributed data structures. This capability makes high-productivity programming abstractions available to any

MPI application written in standard programming languages such as Fortran, C, C++, or Python without relying on compiler technology. Furthermore, the Disk Resident Arrays library [112, 52] provides a high-level programming abstraction for moving data from disk to in-core globally addressable data structures thus effectively integrating secondary storage with the global address space model of GA [113, 119].

OpenMP is a widely deployed global address space model that is increasingly used together with MPI to directly exploit hardware multithreading and fine-grained program parallelism. Its benefits include ease of use and features for dynamic multithreading. Yet it is difficult to achieve scalable performance under OpenMP and there is tension between the need to increase the expressivity of this model and the desire to retain its productivity advantages. Judicious extensions might enable OpenMP to be a more effective vehicle for exploiting chip parallelism as well as large platforms.

Complex applications often involve multiple parts, each of which might benefit from a different model. In addition, a given application may go through phases of performance optimization in which different parallelism paradigms are used. Thus, it becomes critical that implementations of the models interoperate and that a mixture of paradigms be supported in any given language. To support mixed paradigms, we will identify concepts in one model that may be used in another. For example, the addition of multithreading to MPI, GA, or PGAS languages may offer better support for machines constructed from multicore and SMP architectures. The programming elegance of a global view model is partly supported in local view languages through collective communication and collective data structure construction operations. Interoperability requires either a common communication substrate or well-defined interfaces for coexistence of runtime systems. The greatest challenge is pinning down the precise semantics of interoperability as models with static parallelism interact with more dynamic ones, or models with explicit communication interact with those that allow direct access to remote memory. MPI's scalability is particularly important for libraries built on it, and its interoperability with other models underlies the interoperability of such libraries.

2.1 Background

This proposal is for a continuation of the Center for Programming Models for Scalable Parallel Computing (PMODELS), described at [135]. The Center undertook work in three levels of programming models: those expressed by libraries (MPI, Global Arrays, and GP-SHMEM [132]), languages, both OpenMP and the Partitioned Global Address Space (PGAS) languages (UPC, Co-array Fortran, and Titanium), and more exotic languages based on multithreading. It included work on low-level support for language and library implementation, both InfiniBand and a common interface for advanced network protocols, as well as an I/O component. Major accomplishments included new levels of robustness and performance to MPI, GA, ARMCI, and the PGAS languages. This follow-on project will stress productivity, scalability, portability, modularity and interoperability.

This proposal focuses on the areas in which most progress was made and where the most benefit can be realized by continuing the work. In particular, the GP-SHMEM and parallel I/O components have been removed because they were not well integrated with the other work. The multithreading language components have been removed because this topic is now being explored by the better-supported language research efforts of DARPA's HPCS program.

Our goal remains, however, to increase the productivity of application developers by providing them with a set of *programming models*, instantiations of those models, and implementation techniques that allow them to choose an appropriate programming approach to their individual problems. These choices will allow them to obtain scalable performance on today's (and tomorrow's) largest parallel computers with an expressive and convenient combination of languages and libraries.

2.2 Overview of Proposed Work

As in the first generation of the Center, we address programming models expressed in both libraries and languages and include implementation technologies that will be essential for the widespread adoption of advanced programming models.

Library-based models are represented by MPI, the nearly universal standard instantiation of the classic message-passing model, now represented by multiple vendor and open-source implementations, and Global Arrays (GA), a higher-level abstraction of globally addressable shared arrays efficiently implemented in most computing environments and now relied on by several application communities. MPI now means MPI-2, with significant extensions to the message-passing model in the areas of remote-memory access (one-sided communication), parallel I/O, and dynamic process management. We have gained experience with both implementations and use of the one-sided operations and multithreading specifications in particular, and can usefully consider extensions to the MPI standard in these areas. The development of GA will aim to improve programmer productivity while providing additional opportunities for performance optimizations at run time and functionality extensions to help applications scale to petascale architectures.

Language-based models are represented by work on the partitioned global address space (PGAS) languages of UPC, Co-array Fortran, and Titanium, together with OpenMP, the most widely used approach to expressing shared-memory parallelism. Over the past five years, UPC has become well known and has been put into production in a number of applications. Co-array Fortran has gone from an internal Cray experiment to being considered for incorporation in the next Fortran standard. Titanium, in some ways the most “modern” of the three, incorporates an object-oriented approach to the programming model. For each of these programming models, we propose multiple improvements to these languages (directives in the case of OpenMP), compilers, and run-time systems to deliver greater expressivity, performance, and scalability.

To be relevant to the next generation of parallel computers, this work must deliver performance on a wide range of machines, from commodity clusters to petaflop supercomputers. The tension between portability and performance motivates research in compilation techniques and communication methodologies. Substantial compilation research is an integral part of the UPC, Co-array Fortran, Titanium, and OpenMP language development efforts, but we also undertake compilation research in the area of “global view” languages in general, in which parallelism is more implicit and independent of the number of processes. This work will attack the analysis and code generation problems that must be solved for efficient implementation of emerging languages (IBM’s X10, Cray’s Chapel, and Sun’s Fortress) being designed by vendors as part of DARPA’s High Productivity Computing Systems (HPCS) language effort.

Communication technology underlies both language and library implementations on scalable machines. We will continue to enhance ARMCI [121, 111] for petascale machines. ARMCI underlies our GA and Co-array Fortran implementations. We include work on applying the sophisticated InfiniBand interface, which is becoming common on clusters, particularly to MPI implementations.

One important development in computer architecture is the expanding use of multicore processors in response to the difficulty in continuing to increase processor clock speeds. Programmers will need better tools for developing correct and high-performance code for these processors. One obvious model for such processors is a multithreaded programming model; however, with the exception of certain parts of OpenMP, such models remain relatively low-level and fragile, as race conditions are all too common. One aspect of the proposed work is the improvement, for both correctness and performance, of programming models for multicore processors. In addition, in the near term, the programming model for the multicore processor will be combined with a scalable parallel programming model; ensuring the interoperation between the intraprocessor and interprocessor

programming models is another focus of our efforts.

To encourage adoption of new programming models, we will enable *incremental porting* by ensuring that the models may be used together. Achieving this requires interoperable communication subsystems. Building on our experience in exploring common runtime systems for MPI, GasNET, ARMCI, and other libraries, we will define and implement an interoperable communication system that will support incremental porting of applications.

We will also consider “programming patterns,” high-level abstractions found in multiple applications. Understanding such patterns can guide the requirements and specification of programming models.

2.3 Why Us?

The team assembled for this project consists of leading researchers in each of its multiple aspects. Bill Gropp and Ewing Lusk were leading members of the MPI and MPI-2 forums and developers of the widely used MPICH-2 implementation of MPI-2. Katherine Yelick is a co-designer of the UPC and Titanium languages and has led the development of open-source compilers for both languages along with the GASNet communication layer. John Mellor-Crummey is a leading researcher in compilers for data-parallel languages; he is also leading development of an open-source compiler for Co-array Fortran. D. K. Panda is a renowned InfiniBand expert and developer of the MVAPICH and MVAPICH2 MPI implementations. P. Sadayappan is an expert in compiler optimizations and systems support for scientific computing. Jarek Nieplocha is the developer of Global Arrays and the ARMCI library supporting multiple high-level models and was a member of the MPI-2 Forum. Barbara Chapman is a leading member of the OpenMP community and holds a seat on the OpenMP ARB. Marc Snir was a member of both MPI Forums and is now a leader in the study of high-level programming patterns.

3 Recent Accomplishments

In this section, we outline our recent activities and accomplishments relevant to the work proposed here. Much of this work was carried out as part of the first-generation Center for Programming Models for Scalable Parallel Computing.

3.1 PGAS Programming Languages

3.1.1 CAF

P MODELS research at Rice University has focused principally on the design, implementation, and evaluation of technology for compiling Co-array Fortran programs into efficient code for scalable parallel systems. A product of this research is `cafc`—an open-source, multiplatform compiler for Co-array Fortran [48, 164]. The `cafc` compiler translates CAF into SPMD Fortran 90 programs that use either library or hardware support for one-sided communication. Code generated by `cafc` is well suited for today’s commodity clusters. For clusters that lack hardware support for a shared address space, `cafc` generates code that uses library-based implementations of one-sided communication, namely PNNL’s ARMCI [121, 111] or U.C. Berkeley’s GASNet [13]. For shared-address space platforms such as the SGI Altix, `cafc` can generate code that uses loads and stores directly to access remote co-array data [41]. Experiments on commodity clusters (Alpha+Quadrics, Itanium2+Myrinet, Itanium2+Quadrics) and SGI Altix show that for carefully written CAF programs, code generated by `cafc` can deliver performance comparable to that of hand-optimized Fortran+MPI programs—the performance “gold standard” for parallel systems [41, 49, 42, 43]. A study comparing the performance of CAF, UPC, and MPI implementations of the NAS parallel benchmarks [6] MG, CG, SP, and BT on several modern architectures examined the challenges in

delivering top performance with PGAS languages [43]. These experiments showed that today CAF and UPC programs deliver top performance on clusters only when written to use bulk communication; better compiler optimization of communication and synchronization is needed for more natural implementations of these codes in PGAS languages to achieve highest efficiency. A wavefront application in CAF showed that achieving top performance requires overlapping communication with computation by explicitly managing multiple communication buffers [42]. We have begun to investigate language and compiler support that can deliver the performance of multiple buffers without the intricate programming.

3.1.2 UPC

A major accomplishment of the LBNL team, supported in part by PMODELS, was the development of a highly portable source-to-source translator and runtime for UPC. More than 1500 downloads of the compiler or runtime system have been made over the past 10 months. The translator is based on the Open64 open source infrastructure and translates UPC into C code with calls to the runtime layer. This Berkeley compiler is used as the standard UPC for the SGI Altix machine and is being ported to the Cray XT3 in collaboration with Cray. The compiler runs on shared- and distributed-memory machines and has been tested on most operating systems (11 total) and processor architectures (9 total). It can be used on shared-memory machines on top of POSIX threads, on distributed-memory machines using GASNet (described in Section 3.4.2), and on hybrids using a mixture of shared memory and message passing. The compiler supports mixed language programming with MPI and can therefore be used to produce UPC libraries callable from MPI or the reverse.

We have developed applications and benchmarks to evaluate the performance and productivity advantages of the global address space model. These applications were chosen primarily because they are challenging to write in other parallel programming models due to irregularity in parallelism, data structures, and communication patterns. The first was a mesh generation algorithm (Delauney triangulation), which uses a divide-and-conquer approach with user level caching [66]. The second was a hyperbolic solver based on the data structures used in the Chombo AMR package developed by Colella's group at LBNL. The most recent was an LU factorization code, designed at a high level for sparse matrices, although currently configured only for the dense case. This code has been run on a number of machines (SGI Altix, Opteron/InfiniBand cluster, Itanium/Quadrics cluster, and Cray X1), showing performance comparable to the HPL benchmark and up to 60% faster than ScaLAPACK. On a 1024 processor run on an Itanium/Quadrics machine (Thunder at LLNL), the code achieved 4.4 TFlops. The group has also developed smaller benchmarks, including a triangular solver for sparse matrices, a 3D FFT (NAS FT), a conjugate gradient algorithm (NAS CG), and a multigrid solver (NAS MG). The FFT algorithm features non-blocking communication spread throughout the computation phases to effectively hide communication time. The UPC code outperforms the standard NAS Fortran/MPI implementation by approximately 2x on several machines (Quadrics Elan3 & 4, InfiniBand, and Myrinet) with RDMA support and sustains 0.5 TFlops on the Thunder machine [10].

3.1.3 Titanium

Titanium is a dialect of JavaTM 1.4 with extensions for high-performance scientific computing. It was developed at Berkeley by Professors Aiken, Graham, Hilfinger, and Yelick in collaboration with LBNL Scientist Phillip Colella and a team of graduate students and postdocs. The past PMODELS effort has focused on compiler and runtime issues, with other funding supporting work on applications and language design. The major product of the Titanium effort is a source-to-source compiler (Titanium to C) that runs on most parallel and serial platforms, combined with a portable

runtime system based on GASNet. The language runs on top of shared memory using threads, distributed memory using GASNet, and clusters of shared-memory machines using a hybrid runtime system. Performance is comparable to C with MPI [47] and has shown significant productivity advantages for applications in adaptive mesh refinement (10x code reduction) [168] and heart simulation [56].

We have developed several analyses that are used in both optimization and error analysis for parallel languages: *concurrency analysis*, which determines whether two statements in the program could run in parallel; *synchronization analysis*, which lines up matching synchronization constructs; and *conflict analysis*, which finds pairs of accesses (at least one being a write) to a variable that may occur simultaneously [73, 91]. These analyses have been used for data race detection and memory model enforcement, which allows overlap and reordering of memory operations in the local memory or across the network, while ensuring that no reordering are visible at the user level [72]. The group also developed optimizations specifically targeting codes with irregular remote access patterns, such as sparse matrices and particle-mesh methods. Although the programs are written in a simple, shared memory style, the runtime system does performance-model-based communication optimizations; the performance on sparse matrix-vector multiply matches or exceeds that of the Aztec library [147].

3.2 Library-based Models

3.2.1 Global Arrays

In addition to the traditional application areas for Global Arrays such as chemistry, in the course of the project GA have been adopted by several additional DOE applications, including bioinformatics and astrophysics. Our effort has focused on enhancing capabilities and performance of the GA toolkit on high-end systems.

Latency tolerance mechanisms. Several complementary techniques have been developed for latency tolerance, including one-sided nonblocking communication, zero-copy protocols, and mirroring, and have improved performance of scientific applications [130, 158, 119, 84]. Up to 99% overlap of communication with computation have been demonstrated based on advanced network protocols (IBM-SP (LAPI), Myrinet (GM), Quadrics (Elan4), and InfiniBand (VAPI)) [119]. We also introduced a technique called mirroring based on caching latency-sensitive data in shared memory of an SMP cluster [130], which has been adopted in the latest release of NWChem to improve performance on commodity clusters.

Advanced optimizations. GA's communication operations use the fastest communication protocols available on modern high-performance clusters (i.e., shared memory within SMP nodes and RDMA between nodes) by exploiting data locality information. Locality and cluster information APIs have also been implemented to enable communication optimizations at the application level. GA communication operations have been optimized to enable zero-copy on networks such as Quadrics, InfiniBand and Myrinet. These mechanisms have been shown effective across a wide range of platforms and were essential in development of a novel parallel matrix multiplication algorithm called SRUMMA [84, 83, 87, 86].

Multilevel parallelism using processor groups/teams. GA has been extended to use processor groups (called teams in CAF) [117]. Using arrays defined on disjoint processor groups, the GA-based implementation of the NAS Conjugate Gradient (CG) benchmark [176] showed higher performance relative to MPI than previously reported for any other programming model including UPC, CAF, and ZPL. This required adding to ARMCI the ability to allocate and manage global/shared memory by processor teams [88]. A hybrid multilevel parallelization approach based

on group-aware GA also resulted in an order of magnitude improvement in scalability [81] for a time-consuming Hessian calculation in NWChem [76, 60].

Synchronization control. Frequent barrier synchronization in data-parallel programs degrades performance [61]. To reduce the overhead of barriers in GA programs, we implemented a mask operation [119] to enable applications to avoid potentially redundant barrier synchronizations embedded in GA data-parallel operations.

3.2.2 High Performance and Scalable MPI over InfiniBand

PMODELS research at the Ohio State University focused on the emerging InfiniBand networking technology. Specifically, we have focused on the following: taking advantage of InfiniBand's RDMA-based mechanism [95] for both eager and Rendezvous protocols of MPI, exploiting the Shared Receive Queue (SRQ) mechanism to design scalable buffer management [150], achieving better overlap of computation and communication with RDMA read-based optimized Rendezvous protocols [151], fast and efficient collective operations using both RDMA and InfiniBand hardware multicast [93, 101, 100, 149, 77, 152], multirail support to provide maximum performance to bandwidth hungry applications [94], scalable connection management [175], and RDMA based optimized support for one-sided MPI operations [70, 69, 65, 166].

Solutions from this research have been incorporated into two open source distributions: MVAPICH and MVAPICH2 [108]. These implementations are based on MPICH, MPICH2 and MVICH stacks. Two versions of this MPI are available: MVAPICH with MPI-1 semantics and MVAPICH2 implementation with MPI-2 semantics. This open source software was first demonstrated at SC '02 and has been steadily gaining acceptance in the HPC and InfiniBand community. As of March 2006, more than 325 organizations (national labs, universities, and industry) worldwide have downloaded this software from OSU's Web site. In addition, many vendors (both InfiniBand and server) and integrators have been incorporating MVAPICH/MVAPICH2 into their software stacks and distributing it. Several supercomputers using MVAPICH have obtained high ranks in the latest November 2005 TOP500 list ranking, most notably the #5-ranked Sandia ThunderBird cluster with 4,000 nodes (8,000 processors). The latest versions of MVAPICH and MVAPICH2 are also available with the OpenIB/Gen2 stack in an integrated manner. The OpenIB/Gen2 version of MVAPICH was used by a large number of vendors on the SC'05 exhibition floor using the OpenIB-SCINet.

3.3 Directive-based Programming Models

OpenMP uses directives embedded in comments to enable the expression of parallelism.

OpenMP language. The University of Houston (UH) has focused on research to increase the scalability and range of applicability of OpenMP. We have evaluated its behavior on multithreading [90] and DSM platforms [27], investigating thread placement, overheads and scalability issues. Our results indicate that a traditional SPMD workload will not provide the best performance where there is a need to avoid resource contention. We have proposed extensions to define, shape and exploit subteams [29] of threads, to permit control over thread placement and to provide finer synchronization. These features help target higher thread counts, tolerate long-latency I/O and take the hierarchical nature of emerging platforms into account. Some of them are being discussed by the OpenMP ARB, which is working on OpenMP 3.0. To influence the standard, Chapman founded "cOMPunity" [44], which joined the ARB. Chapman is a member of the management committee that drives the ARB's efforts, and the team participates in its subcommittees. We have also collaborated to provide a public-domain Fortran/C/C++ OpenMP validation suite [107]. It contains a unique crosscheck framework in a user-configurable environment.

Compiling for the shared-memory model. We have built an optimizing, portable OpenMP compiler [129, 89] (C++/C/Fortran and OpenMP 2.5) based on Open64 as a reference implementation and as a testbed for language research. Features from major branches of Open64 [133, 39] were integrated and enhanced. `OpenUH` includes support for OpenMP code creation and is available for download. We have also developed compiler and run-time technology to improve the scalability and applicability of OpenMP. Optimizations proposed include translating OpenMP into SPMD style [96], wide-area array privatization [97], and asynchronous execution [169, 170]. Experiments showed the benefits of these optimizations [27, 28]. We also translated OpenMP into Global Arrays [63, 98, 64] for clusters. Advantages of this approach over Software DSM are the efficiency and precision of data transfers and the ability of the compiler to optimize the generated code.

Hybrid programming model. We extended Sphinx [145] by MPI+OpenMP overheads measurements [136]. We developed an analytical performance measurement framework comprising a *system profile* (using Sphinx and Perfsuite [134]) that characterizes system influence on program performance and an *application signature* to characterize application behavior [1]. Interactions between our system and external tools support hybrid programming [50, 153, 171, 134].

3.4 Runtime Systems

3.4.1 ARMCI and Collective Communication for Programming Models

PMODELS research on the ARMCI portable remote-memory copy interface [121, 111] focussed on enhancements to better support the Global Arrays and GP-SHMEM [132] libraries, Co-array Fortran, and Co-array Python [139]. Research and development efforts at PNNL and OSU included latency hiding mechanisms, fundamental technologies for efficient one-sided communication, memory synchronization, and high-performance collective communication. These technologies have been incorporated into ARMCI and are available for download in release 1.2.

Latency-hiding mechanisms. The new *aggregated* communication protocol in ARMCI combines multiple communication calls issued by the user or compiler. Aggregated nonblocking communication, in addition to being effective for latency hiding, reduces the number of network packets and increases the bandwidth by transmitting larger chunks of aggregated data [123, 121]. We demonstrated the advantage of this method [122] in the context of a sparse matrix-vector multiplication for one of the matrices from the Harwell-Boeing collection. Another significant addition to ARMCI was portable one-sided *nonblocking* communication for contiguous and noncontiguous data [123]. The effectiveness of these mechanisms for overlapping communication with computations was validated in scientific benchmarks such as the NAS parallel benchmarks [159], or the SRUMMA parallel matrix multiplication algorithm [84].

Fundamental technologies for one-sided communication. The efficacy of latency hiding mechanisms such as *aggregated* and *non-blocking* communication depends on their ability to deliver performance very close to that of the network hardware. We implemented several new and portable protocols and methods for implementing one-sided communication efficiently across different networks [123, 124, 109, 116]. Example innovations include the Host-Based NIC-Assisted (HBNA) method for optimizing noncontiguous communication for InfiniBand Verbs [163] and the hybrid host and NIC-based method [120] that exploits the programmable network adapter of the Quadrics QsNetII network [8], and scalable distributed memory synchronization [21].

High-performance collectives. Our research on high-performance collective communication included developing multimethod topology-aware implementations (Best Paper, IPDPS 2003) [162, 62] and exploiting concurrency at all levels in system and network hardware [160]. These algorithms showed a significant improvement over the traditional methods used for collective communication.

For example, our concurrent algorithm for the allgather operation showed an improvement of 89% over the traditional method on a Quadrics Elan-4 cluster.

Memory synchronization and management. To support the Co-array Fortran compiler, we added remote notification mechanisms to signal completion of nonblocking one-sided operations. To support multilevel parallelism in programming models, we incorporated the concept of process groups into ARMCI and added the necessary system-wide memory management by processor groups/teams [88]. We also implemented a Fortran 95 wrapper library to ARMCI [110] that supports the Symmetric data objects introduced by Cray, without requiring specialized hardware or compiler support. Symmetric data objects greatly simplify parallel programming by allowing programmers to reference remote instance of a data structure by specifying address of the local counterpart.

3.4.2 GASNet

The GASNet communication layer is used by runtime systems for the Berkeley and Intrepid UPC compilers, the Titanium compiler, and the Rice CAF compiler. GASNet uses a two-level implementation model for portability: A small core based on active messages is required for each platform. The full interface can be implemented in terms of the core (such an implementation is provided), or parts can be optimized for a given platform. There are optimized implementations for Myrinet GM, Quadrics Elan 3 & 4, InfiniBand VAPI, IBM LAPI, Cray X1, SGI Altix, Dolphin (done by the University of Florida [146]), and Cray/SGI SHMEM. In addition, for further portability there is an implementation on top of MPI and another on UDP for Ethernet.

3.4.3 MPI and Common Communication Systems

During the first round of the PMODELS work, we concentrated on high-performance communication systems that could support a variety of programming models, including MPI, the PGAS languages, and others. As part of this work, the ANL team evaluated systems used by PMODELS, such as ARMCI, GasNET, and the MPICH2 ADI3, and systems used by the broader community, such as IBM's LAPI. We considered both the functional requirements of the client programming models and performance issues. We found that no existing system addresses all of the issues faced by the current programming models, and designed and prototyped a Common Communication Subsystem [14]. Also as part of evaluating designs for communication subsystems, the ANL and OSU groups collaborated to find opportunities for exploiting RDMA and remote operation capabilities [18, 16] and at the efficient implementation of MPI's one-sided (also called remote memory access) operations [70, 92]. These results are described in more detail in section 3.2.2.

These and other investigations indicated that there were many opportunities for improving the performance of the current communication subsystems used by many MPI implementations. We are developing a new implementation of the communication subsystem for MPICH2, called Nemesis, which seeks to minimize the overhead in message passing. On a shared-memory platform, we can perform an MPI send-receive in roughly 400 instructions and less than 400 ns, the fastest time reported for an MPI implementation [15].

In addition to the work supported by PMODELS, we have continued to conduct research into improving the efficiency of MPI. This includes better implementations of the MPI collective operations [157, 154], which outperform those of several vendors. We have shown how to make use of the MPI semantics to significantly improve the performance of one-sided operations in MPI [156, 155, 58]; our implementation is up to six times faster than that of the vendor's MPI. In both the collective and one-sided cases, the enhanced performance comes from picking and developing better algorithms than are in general use. The ANL team has also worked on efficient implementations of

MPI one-sided operations as a joint effort with OSU for Infiniband [71] and as a joint effort with IBM for IBM’s BG/L system [4].

To enhance productivity, we have developed three tools that exploit the MPI profiling interface. One provides runtime checking for the use of MPI collectives [51], exploiting ideas from [59] to efficiently detect mismatched type signatures. FPMPI2 [57] (www.mcs.anl.gov/fpmi2) provides concise summary data; its novel feature is the use of message length bins to separate latency and overhead-dominated operations from bandwidth dominated ones. Jumpshot [172, 26] provides detailed performance visualization, using a novel scalable (in time) logfile representation to make it feasible to view logfiles of arbitrary size (tested up to 100 GB).

3.5 Compiling Global View Data-parallel Languages

To generate high-performance code for a data-parallel language, a compiler must exploit parallelism effectively, balance load, minimize communication frequency and volume, hide communication latency, and generate an efficient node program. To meet this challenge, the dHPF project has developed a spectrum of program analysis and code generation techniques that enable it to generate code that matches the performance of expertly-written MPI codes.

Set-based analysis framework. dHPF uses a formal analysis framework to reason about data-parallel programs by manipulating symbolic sets of integer tuples that represent data indices, loop iterations, and processors [2]. To support analysis of these sets and mappings between them, dHPF uses techniques based on integer programming (known as polyhedral methods) for analyzing and enumerating symbolic sets of integer tuples described by Presburger arithmetic formulas¹.

Advanced data distributions. Compilers for data-parallel languages use data distributions to guide the partitioning of computation among the processors in a parallel system. We developed a new class of skewed-cyclic block distributions known as *generalized multipartitionings* for mapping a d -dimensional data volume onto an arbitrary number of processors and compiler support for using these distributions (Best Paper, IPDPS 2002) [45, 46]. Multipartitionings enable better parallelization of line-sweep computations than HPF’s standard BLOCK or CYCLIC(κ) partitionings [165].

Flexible computation partitionings. Compilers for data-parallel languages generally use the *owner-computes* rule [140] to partition computation among the processors. The dHPF compiler uses a more general strategy that enables computation to be partially replicated to reduce communication. For NAS BT, partially replicating computation reduces message volume by 66% and boosts overall performance by 38% [32, 33].

Communication optimizations. Fast parallel programs require very efficient communication. A complex loop nest may contain multiple statements with data dependences that constrain communication. To optimize communication for such code fragments, dHPF employs a powerful set of integrated optimizations. It *vectorizes* communication out of as many loop levels as possible. It *coalesces* overlapping communications corresponding to the same array to eliminate redundancy [34, 32, 33]. To reduce message frequency, dHPF *aggregates* communication events for nonoverlapping data [35, 104]. To tolerate communication latency and process asynchrony, dHPF generates code that uses split-phase nonblocking communication primitives [35].

Efficient node programs. dHPF uses a code generation strategy based on Fourier-Motzkin elimination [142, 137, 75] to enumerate points in communication, iteration and processor sets. It also employs contextual knowledge to simplify control flow [103]. In addition, dHPF optimizes memory hierarchy utilization through array padding to reduce conflict misses and carefully manages communication buffers to reduce the cache footprint of off-processor data.

¹The dHPF compiler uses Pugh et. al.’s Omega library [74] for this purpose.

Evaluation of these optimizations underscores their promise [32]. In 2005, we used Rice’s dHPF compiler in conjunction with Looptool [138] to generate explicitly parallel code for IMPACT-3D, a plasma simulation code written in HPF for the Earth Simulator. On 1,024 processors of the Lemieux cluster at the Pittsburgh Supercomputer Center, the generated code achieved over 17% of peak [31], which is competitive with the performance of optimized MPI codes on clusters [128].

3.6 Programming Patterns

Work on new programming models is useful only to the extent it focuses on simplifying frequently performed tasks. A successful methodology for cataloging such tasks is provided by Design Pattern Languages [55]. These present, in a stylized form, high-quality solutions to frequently occurring problems in object-oriented programming, organized in a hierarchical and compositional manner. This approach has been used to document programming practices in many programming areas, including parallel programming [102], but with only limited participation of the HPC code developers.

We have recently started an effort aimed at bringing this community together and at providing a useful catalog of prevalent tasks in programming for HPC [144]. Our first workshop took place in 2005, and follow-up workshops are expected to occur annually. This effort has led to the identification of important classes of HPC programming activities that need to be cataloged and has created a first catalog of such tasks. In particular, we have classified and formalized tasks related to load balancing, regular and irregular communication patterns, dynamic adaptation of the solution domain (multigrid, adaptive-mesh refinement, spectral methods, etc.) and so forth.

4 Proposed Work

In this section we describe our research plans. Some of the work is a continuation of work begun in the first generation of the Center; some represents new or enlarged efforts with the same overall goals.

4.1 PGAS Programming Languages

PGAS languages (CAF, UPC, and Titanium) and their open source compilers need refinement to enhance productivity, scalability, performance, and interoperability. The CAF research will be done primarily at Rice University; the Titanium effort will be at U.C. Berkeley; and the UPC work will be done as a joint Berkeley/LBNL team effort (with funding predominantly from other sources). The specifics of the work breakdown by institution are found in the Appendix. In some cases, we will investigate a common problem in all three languages but explore different solutions tailored to the different design philosophies and user communities of the languages.

4.1.1 Language Refinements

Function shipping. PGAS languages offer some form of remote *read/write* or *put/get* for access to remote variables. Remotely manipulating complex data structures is inefficient on clusters. We propose extending all three languages with the ability to spawn asynchronous remote computation on the node where a data structure resides so that the data can be manipulated locally. Several open questions remain: should these functions execute atomically? Can they call other remote operations internally? How are they ranked with respect to remote work?

Hierarchical parallelism. To better support hyperthreaded and multicore processors, we may wish to relax the current SPMD model used in PGAS languages. We will consider several models, pursuing different options in the three languages. In CAF we propose to add the ability to spawn local asynchronous computations with Cilk-like concurrency [54]; in Titanium we will leverage an

existing unordered `foreach` construct to add parallel loops within an SPMD thread; and in UPC we will add hierarchy to the currently linear set of thread identifiers.

Collective communication. Built-in constructs for collectives not only are a programming convenience for programming; but are essential for performance portability. They also enable more efficient implementations through a combination of compiler and runtime support. CAF lacks support for collective communication; we plan to add such constructs and extend the Rice `cafc` compiler to support them. We will also work with the UPC Consortium on their collective model, which has a sophisticated set of synchronization flags in the interface to specify whether barriers are implied as the operation is called or returns. For UPC, we believe that better compiler support could be used to simplify programming while retaining runtime flexibility. Titanium has a set of scalar collectives; we will expand this set to include bulk versions as well. In all cases, we will engineer compiler and runtime systems to select the most efficient implementation of collective calls for the target communication fabric and use.

Structured teams. Currently, none of the PGAS languages provides a mechanism for a group of processes to define a *team* analogous to creating an MPI subcommunicator. CAF's present notion of teams is too unstructured to support collective operations on processor subsets. In Titanium, various analyses conflict with teams, and in UPC, collective memory allocation is problematic within a team. Teams are critical; they were needed in our UPC implementation of a Delaunay mesh triangulation code and in LU factorization. We propose to develop constructs for subdividing the global set of threads into a hierarchy of structured teams, along with support for team-based collective communication.

Process topologies. We propose augmenting CAF with processor *cospaces*, which are analogous to MPI virtual process topologies, along with intrinsics for specifying neighbor relationships. Using cospace intrinsics to specify communication partners instead of arbitrary functions of processor indices will make communication patterns more transparent to the compiler, aiding analysis and optimization of synchronization.

Global-view data structures. UPC and CAF support distributed arrays built from a single global-view declaration. However, these abstractions are at a much lower level than multi-dimensional array layouts in HPF. In CAF, we will explore the implications of extending the language with support for higher-level HPF-like array abstractions. Titanium has very general layouts, but distributed data structures are built in two phases: local pieces are allocated and initialized, and then references to them are exchanged to give all threads a view into all others. This mechanism can handle arbitrary tilings (e.g., arbitrary 3D tilings and adaptive meshes), but it is overly general for simple computations. Using data parallel languages as inspiration, we propose to augment Titanium with support for building collective data structures; this will simplify writing applications that use uniform meshes and large dense matrices. In UPC we will work with the community to allow dynamic blocking factors in response to user demand.

Mixed global/local view control. Writing single-threaded programs with data-parallel constructs is convenient but not expressive enough for all applications. A fundamental parallel language question, which the HPCS languages are also addressing, is how to combine the convenience of data parallelism with the expressiveness of a more general explicitly parallel model. Adding support for specifying global-view computation will simplify programming in CAF, but a key challenge will be supporting interoperability between such global-view programming and CAF's current local-view model. When mixing global and local view programming in Titanium, compile-time checking must ensure that all threads are ready to execute a global view computation even though sometimes they independently execute local code. We will explore language extensions to make this practical.

4.1.2 Compiler and Runtime Technology

Refinements to PGAS compiler and runtime technology are needed not only to support the new features described above but also to enable programs written in a natural, productive style achieve high performance across a broad range of platforms.

Parallel program analyses. The Titanium compiler has a set of analyses designed for parallel languages, as described in Section 3.1.3. Currently, alias analysis handles a two-level memory hierarchy with local and remote, but not a hierarchical model. We propose to extend our analyses to handle multiple levels of memory. The Titanium analyses rely on the ability to line up barrier synchronization points at compile-time [3], but in UPC and CAF this is not possible in general. We will develop analyses appropriate for other synchronization models, perhaps by combining static information with dynamic checking.

Communication optimization. We propose to extend the PGAS compilers with support for automatically vectorizing communication, packing data, aggregating messages, and converting blocking to non-blocking communication as guided by application, communication library, and platform parameters. We propose to explore opportunities for transforming *gets* into *puts* by understanding the global structure of synchronization, as is done in an inspector-executor optimization phase in Titanium [147]. Several of these communication optimizations have been demonstrated in experimental versions of UPC [68, 37] and Titanium [38], but need to be generalized for use in full applications. In addition, we have found that excessive aggregation and packing can reduce overlap opportunities, so the optimizations must be guided by performance models [67, 147, 10]

Synchronization strength reduction. Barriers are simpler to use than point-to-point synchronization primitives; however, this simplicity comes at a price: programs using barriers are as much as 30% slower than counterparts using point-to-point synchronization [48]. We propose to explore algorithms for safely softening barriers into faster point-to-point synchronization.

Distributed multithreading. Function shipping and multithreading are aspects of distributed multithreading; adding this support introduces challenges at all levels: refining the memory consistency model, designing a new runtime system to efficiently support both local and remote invocations, and determining at compile time whether remote operations can be combined or whether a particular remote operation can be handled as an interrupt rather than a thread. We propose to develop compiler and runtime support for efficient distributed multithreading.

Interoperability. New compiler and run-time support is needed for CAF and Titanium programs and libraries to interoperate with other programming models. UPC already has a notion of interoperability built under PMODELS. The ability to call other library-based models, such as MPI, is relatively straightforward as long as the runtime layers are compatible. Titanium and UPC support this model, which has been used to call FFTW [53], Chombo [40], and other packages. Calling from another model into a PGAS language requires more support. In UPC this was provided by creating initialization routines that are called explicitly in any application that will make a call to UPC. We will test our UPC support in numerical libraries being developed under independent funding and will add similar support to CAF and Titanium.

4.2 Library-based models

Here we describe enhancements to MPI and GA implementations that are particularly relevant to the programming models they represent.

4.2.1 MPI

We will explore two ways to improve the MPI programming model. The first is to improve the performance of MPI implementations by addressing both communication overhead and scalability. Reducing message overhead allows applications to use smaller messages and thus finer-grained decompositions, thereby increasing both performance and scalability. For massively parallel systems, there are parts of the MPI specification (such as the graph topology routines) whose interface is not scalable, and others, such as communicator construction, whose implementations require careful design to avoid such non-scalable representations as enumerating all processes that are members of a communicator on all processes. We will develop scalable representations for these data structures (see also Section 4.2.2).

The second way to improve the MPI programming model is to re-examine some of the design choices and look at alternatives that better represent the current abilities of parallel computer hardware. For example, the MPI one-sided specification has no fetch-and-increment or compare-and-swap operation, and implementing these scalably within MPI using only the one-sided operations is difficult. The lack of these operations makes it difficult to use MPI both in certain applications and, perhaps more important, as the basic portability layer for other programming models. The one-sided model of MPI was also designed to make minimal requirements on cache coherence in the underlying hardware; it is time to consider other consistency models that may simplify the MPI model and provide additional opportunities for achieving the best possible performance.

We will address the following:

- Efficient implementation of thread-safe MPI (in particular, minimize the use of locks and context switches and provide smooth interface to multithreaded programming models)
- Better use of topology information in process layout, communication, and implementation of collective communications within MPI (this addresses scalability issues that are becoming apparent on highly scalable systems such as IBM BG/L)
- Enhanced one-sided operations, for example, possible extensions to MPI, including one-sided read-modify-write operations
- Low-overhead implementation of MPI operations. By paying close attention to the implementation, we can significantly reduce the overhead of MPI (we are currently at 339 ns for a zero-byte send-receive on a 2.4 GHz dual processor Opteron using shared memory). Reducing the overhead of MPI calls can improve the scalability of applications that use MPI and are dominated by short messages.

In addition, interoperation with other programming models is needed; this is discussed in Section 4.4.6.

Related to the work on MPI is work on providing better support to MPI programmers in handling distributed data structures. MPI provides the tools for programming with arbitrary data structures distributed over arbitrary sets of processes but provides no aids to help the programmer with this task. Joint work with the effort on Programming Patterns (Section 4.6) and annotations (Section 4.3.3) will develop tools to aid in the management of distributed data structures, including unstructured meshes, graphs, and distributed arrays.

4.2.2 Global Arrays

Work in Global Arrays will address two issues: improving scalability and single-node performance, and enhancing programmer productivity. This work will be primarily pursued at PNNL and OSU.

In the first phase of the PMODELS project, prototype support for multilevel parallelism has been developed for GA [118] and was shown to improve scalability in benchmarks and applications

[82, 127, 176]. Multi-level and hierarchical parallelism offers additional opportunities for scientific applications to exploit the forthcoming petascale systems. We will extend this capability to the full set of functionality in GA and harden it for production use. We will develop additional functionality to help improved scaling at the application level, for example nonblocking or split-phase team synchronization operations that work in the context of processor groups. The GA toolkit was originally developed to support system configurations with up to 2000 processors. Internal enhancements in the implementation of GA will be necessary to support the forthcoming petascale systems. For example, we will exploit a new global memory allocation scheme proposed for ARMCI (see Section 4.4.4) to reduce the size of the internal data structure that represents a global array (O(P) memory consumption systemwide).

To improve single-node performance we will offer additional locality and data layout optimizations for distributed arrays in GA, for example by exploiting tiling or topology-aware mapping that has been recently prototyped [36]. In addition, we will develop support for mixed mode-parallelism for multicore processors based on threads and OpenMP models.

To enhance programmer productivity and improve application performance, we will develop a taskpool model that provides a high-level mechanism for load balancing for GA and GA/DRA applications. Such a model facilitates locality-conscious load balancing because the location of global data accessed by each task is explicitly known. Further, since the granularity of the task is controllable by the user, it is feasible to balance the communication and scheduling overheads with the degree of load balancing achieved. We plan to implement both static and dynamic approaches to load balancing. The taskpool model is particularly attractive for multicore systems: distinction between intranode parallelism (within a multi-core processor chip) and internode parallelism (across different chips) can be kept transparent to the programmer, but the actual mapping of tasks to processors can fully exploit intranode locality. Optimizations of the taskpool model for multicore processors will be pursued. An initial proof-of-concept exploration of the taskpool approach carried out using the PaTOH hypergraph partitioner [25, 24] for communication minimization [79, 78] has produced promising results.

One of the features most frequently requested by application developers is support for more complex and dynamic data structures in Global Arrays. We will develop interfaces allowing users to specify more complex data structures. A prototype implementation has been developed over ARMCI, to support a class of block sparse matrices used in ab initio quantum chemistry [80] in the the Tensor Contraction Engine (TCE) domain-specific compile/runtime system [5, 7].

Another proposed enhancement is the development of XGA (eXtended Global Arrays) [11], a transparent access interface to in-memory data and on-disk array data. The current GA/DRA model has an explicit collective-call interface to move multidimensional blocks of data between GA and DRA [112, 52]. It would be much more convenient for programmers if a single interface were available and movement of data between disk and memory were handled automatically by the system. We will develop both compile-time optimizations and run-time caching and aggregation optimizations to attain good performance.

In addition, we will provide better support for compilers that use GA as the compilation target, such as the OpenMP compiler from U. Houston, the global-address-space MATLAB system GAMMA [131], and the Tensor Contraction Engine. Part of this effort will involve exposing access to the global array data structures through ARMCI that will enable use of more advanced communication interfaces in ARMCI, such as the aggregate handle nonblocking communication. This capability will be useful for optimizing performance of the OpenMP compiler for clusters. We will also work with DARPA HPCS vendors (IBM, Cray) on interoperability of GA with proposed languages such as X10 and Chapel. One of the goals is to provide GA distributed array capabilities as enhancements to these languages.

4.3 Directive-based Programming Models

Here we describe our proposed work in three areas where a programming model is primarily expressed in directives (comments) embedded in serial code.

4.3.1 OpenMP Language

Work is needed if OpenMP is to retain its productivity benefits, be highly suitable for exploiting multithreading SMPs, provide scalability on hierarchically parallel platforms, and interoperate smoothly with other high-end programming models. Issues that we will work toward include:

- To enable the expression of higher levels of parallelism and to enable it to help overcome thread resource contention, OpenMP requires more flexible means of assigning work to threads. In particular, the worksharing construct requires additional power. Multilevel parallelism needs to be expressed without the overheads of the dynamic nested parallelism.
- OpenMP must be adapted to better fit the hierarchical parallelism of new platforms. It will need to be able to express mappings of threads to parts of a machine, either directly or indirectly, especially for a cluster implementation. An abstract machine description is one possibility; another is to impose structure on the threads in a given team, possibly with additional information that helps the compiler/run time determine appropriate thread bindings. Data distributions for clusters might come in the form of library routines or via the addition of a new data attribute.
- Synchronization options in OpenMP are limited and often lead to many barriers at run time. More efficient and scalable methods for expressing synchronization, such as atomic blocks, need to be explored in the context of OpenMP. The concept of clocks might be used to help the expression of a variety of concurrent workloads. Synchronization between individual threads and between subteams of threads should be facilitated.
- The ability of other programming models to interact with OpenMP needs to be properly explored, from the suitable placement of processes and threads across a platform via the compatibility of language constructs to the thread safety of their implementation.
- Language features for object-oriented programming need to be more thoroughly evaluated, as does the problem of robustness. OpenMP does not directly report on errors at run time, instead relying on the programmer to insert tests. This approach needs to be reconsidered.

4.3.2 OpenMP Compiler and Tools Technology

We have created a portable reference compiler for OpenMP based upon Open64. We will continue to improve our compiler and run-time system technology throughout this project and will develop new techniques to support our language research. We will exploit the project's runtime substrate for our cluster implementation and will work with partners to implement hybrid programming models. In particular, we will address the following issues:

- OpenMP compiler technology requires further work to improve the optimization process, in particular to better deal with the problems posed by loop nests and false sharing. This requires extending traditional compiler analyses to directly take into account the architecture and the programming model.
- More compile-time work is needed to enable OpenMP to be successfully implemented on clusters. Particular attention needs to be paid to the translation of sequential regions, optimization of communication, and handling of synchronization, which may be much improved via the proposed new language features.

- The compiler needs to be provided with a cost model that describes multithreading platforms. This model may be used for a variety of improvements including setting application and target machine specific execution defaults. To support dynamic modifications, the compiler’s own runtime environment needs to be extended to support adaptive execution modes.

4.3.3 Annotations

Program annotations allow the programmer to add semantic information or code generation hints to a source code. Annotations are similar to directives in extending the base language, but are different in usually being applied through source-to-source transformations. For example, a simple annotation may mark a loop as using aligned data (suitable for extra-wide load and store instructions or extra floating point units, as in IBM’s BG/L); a more complex annotation might describe a distributed array [141]. The advantage of using source-to-source transformations is that extending and customizing the language through annotations can be done independent of the base compiler. By taking advantage of this *separation of concerns*, it is possible to more rapidly develop and deploy extensions. We have found this approach particularly valuable for the IBM BG/L, where annotations have allowed us to significantly improve the performance of some application kernels.

A disadvantage with a simple approach based on blind transformation of the source is that the user can easily err in applying the annotation (for example, forgetting to use an annotation on a variable for which the annotation is providing global-view semantics). In addition, for complex transformations (such as the application of loop restructuring), a common “metalanguage” is needed to provide interoperability with transformation tools. Thus, annotations fit between the full-featured support possible when such features are completely integrated within the language (as in our work with CAF and Titanium) and the manual code development faced by programmers using library-based models. Annotations do offer a faster way to experiment with new distributed data structures and code generation techniques. As part of this work, we will develop an advanced annotation system to address single-node (including multicore) performance through targeted code transformations (building on existing tools, such as loop restructuring tools), productivity of MPI programmers through the addition of global-view and communicator-view annotations, and robustness of annotated software through the use of compilation tools to enable stronger checks of annotation correctness and completeness.

4.4 Runtime Communication Substrates

The proposed activities will include enhancements to the runtime support for MPI, Global Arrays, and PGAS languages. In particular, our research will be directed toward the following goals:

- Enabling scalability to very large processor configurations (100K+), in particular the emerging petascale architectures
- Achieving highest-possible performance on hardware of interest to DOE
- Providing interoperability between the programming models represented in the project
- Enabling portability across a broad range of architectures and networks

We will work on achieving these goals within ARMCI, GASNet, and the MPI Communication Subsystem. Of particular importance to our project are one-sided and collective communication operations as they are required for implementation of different programming models in the project.

4.4.1 Support for Emerging Systems

We will continue development and optimization of high-performance implementations of our communication interfaces on new networks, such as the next-generation Quadrics, Myrinet, and Infini-

band, and on major terascale and petascale architectures that emerge during the project, starting with IBM BlueGene, Cray XT3, and IBM LAPI. The Berkeley, PNNL and Argonne groups will collaborate closely on a optimized runtime support for BlueGene systems from IBM that satisfy the needs of the various runtime clients. Our IBM LAPI effort for the Power architecture line is a part of an ongoing collaboration with the LAPI team at IBM [143], through which we influenced the design of the new RDMA API in LAPI to reflect runtime requirements of PMODELS project. Multicore architectures are emerging as building block of choice for high-end systems, with the expectation that the number of cores per chip will double every two years. We plan to explore new designs that can dedicate one or more communication threads per core to achieve faster communication progress. These communication threads can be utilized by several capabilities in the runtime system including collectives, datatype processing engine (including zero-copy schemes), and one-sided synchronization implementations.

We will address scalability limitations in the design of MPI point-to-point two- and one-sided (MPI and ARMCI) operations to support the emerging systems with very large processor counts (100K+). For example, with InfiniBand, Shared Receive Queue (SRQ) mechanism can be used [150] to reduce buffer memory consumption for reliable protocols. This mechanism can also be used together with connection-less and on-demand connection management protocols [175] to achieve further scalability since these schemes do not require posting of buffers for every queue pair.

4.4.2 One-Sided Communication

The PMODELS team has been instrumental in showing the benefits of one-sided communication for exposing the highest possible performance of underlying network and translating this advantage into performance advantages at the application level [83, 85, 10, 47]. Such performance depends on having highly tuned implementations, which we will continue to develop for the strategic network architectures of interest to DOE.

High-performance implementations of noncontiguous data transfers are critical to the performance of CAF, GA, and Titanium and will remain an important element of our effort. We will introduce more flexible interfaces in ARMCI, allowing users to specify aggregation of contiguous and noncontiguous messages to improve performance for more complex data transfers occurring for example for block-sparse data structures.

Our current work on MPI one-sided communication for Infiniband [70, 69, 65] exploits RDMA and atomic operations in InfiniBand. We will also investigate several scalable algorithms to alleviate the performance and scalability bottleneck in one-sided operations with active synchronization.

4.4.3 Collective Operations

Collective communication is featured in each of our programming models. As the number of processors in high end systems grows, and full bisection bandwidth becomes prohibitively expensive, we expect to need more clever collective algorithms to deliver scalability and efficiency.

In preliminary work on MPI [93, 101, 100, 152, 149], we have shown the benefits of InfiniBand features such as Hardware-Multicast and have improved collective protocols using RDMA. We plan to consider several new and challenging design options, such as design of collectives over point-to-point UD-transport, to deal with resource scalability constraints, and the dynamic connection model for MPI where only certain processes communicate via reliable connection oriented channels of IBA(RC-transport). In addition, we will consider the dynamic usage of RDMA Write and RDMA Read and *application-bypass* capability [105, 174, 19, 20, 16, 17, 18] to design high performance collective communication support with asynchronous progress.

Information about the system topology and task mapping is critical for optimizing performance of collective operations [161]. InfiniBand defines several management classes that are responsible

for managing various features of the network; we will exploit them in our design. Information about the topology and task mapping will be used in designing efficient collective communication algorithms by avoiding hot-spots and exploiting network proximity and concurrency (for example, by exploiting multiple paths in the network [160], including creation of hardware multicast groups on Infiniband and using these for other collective operations).

We will also optimize collective communication in GASNet, which includes a form of non-blocking collective to allow global operations to be performed without global synchronization. Given the enormous design space for implementing this functionality and the differing system characteristics, we plan to use performance models and, if necessary, search-based self-tuning mechanisms to automatically select implementations at installation time.

4.4.4 Memory Management

Memory management is one of the most subtle features of these runtime systems, and different programming models have different demands. At one extreme, Titanium does not distinguish private and local space and gives control of memory management to a standard conservative garbage collector [12].

We will develop a new memory allocator in ARMCI for globally addressable memory to address the scalability shortcomings of the current interface. For example, to allocate memory for a global array $O(P^2)$ pointers are needed systemwide, clearly not scalable. The new scheme will be based on pointer virtualization and will use only $O(P)$ pointers systemwide.

Our initial GASNet port for the Cray XT3, done in collaboration with Cray, will support a mode of GASNet (partitioned private/shared memory) that is suitable for UPC, but not for Titanium. We will extend that implementation with support for the GASNet Firehose algorithm, which supports dynamic registration and deregistration of memory and caches information about remote registration tables to save on overhead [9].

4.4.5 Functionality Enhancements

Extensions of our PGAS languages and GA for remote function invocation in turn require extensions to the runtime systems. We will explore different levels of generality and implementation techniques and identify design trade-offs.

Our ARMCI effort will be directed toward the Global Procedure Calls (GPC) capabilities recently prototyped in ARMCI. GPCs allow an application to specify a handler function that can modify globally addressable data without specifying the process that would execute it and without relying on polling to achieve progress. GPCs will be used to support work forwarding as well as other advanced capabilities such as I/O and sparse and dynamic data structure processing.

Hints will be added to allow compilers and libraries accessing ARMCI provide context information that can be used internally to optimize the usage of resources (internal message buffers) or select appropriate low-level data transfer protocol (e.g., host-based or NIC-based packing for noncontiguous data). We have realized the importance of these optimizations when optimizing communication in SRUMMA matrix multiplication [84, 83, 87, 86].

GASNet is already suited to a limited form of remote function invocation, because of its use of an active message [167] model at the core. That experience has revealed a number of nontrivial semantic issues, such as potential deadlocks if handler functions, which run without preemption, can acquire locks or initiate communication. This is addressed by limiting the functionality of handlers, for example, by creating a separate virtual network for handler response messages and using only *handler-safe* locks.

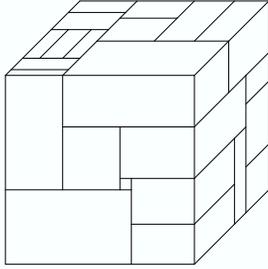


Figure 1: A 3D generalized tiling.

```

distribute a(block) onto P
distribute b(block) onto P
DO l = 1, levels
.....
! multiple butterflies
DO k = 1, n, 2m
.....
! a single butterfly instance
DO i = k, k+m-1
  b(i+m) = a(i) - w * a(i+m)
  b(i) = a(i) + w * a(i+m)

```

Figure 2: Global-view FFT code fragment.

4.4.6 Interoperability

We will develop *interoperable* communication subsystems, rather than a single common runtime communication system. Among the major issues in interoperability [14] are the initialization and finalization of the communication subsystems (such as memory registration for RDMA operations) and the definition of a common notification interface that can pass the communication event to the appropriate subsystem for further processing. Another important issue is the efficient management of the system resources such as ports on the network interface or pinned memory segments. Each communication subsystem needs to discover what processes belong to the parallel job; this requires a common or at least interoperable process management interface. For optional features, a common introspection interface will be developed to allow communication subsystems to determine what functionality they will need to provide and what they can leverage on the underlying system.

4.5 Compiling Global View Parallel Language Models

Global view models based on data-parallelism (e.g., Parallel Matlab) are a promising approach to improving programmer productivity. (A global-view language model is one in which computation is conceptually single threaded, data is globally accessible and communication is implicit.) However, mapping such models efficiently onto distributed-memory machines is difficult. It is an underconstrained problem that benefits from user guidance, including specifications for data layout (alignment and distribution) and data/computation affinity. HPF is an example of a global-view language designed over a decade ago based on this strategy. Today, emerging HPCS languages (Cray's Chapel [22], IBM's X10 [30], and Sun's Fortress [148]) embrace HPF's support for locality-aware programming by using data distributions to map global view programs onto parallel systems.

Achieving high performance with global view languages requires partitioning work effectively, colocating data and computation, minimizing communication, and synthesizing efficient node programs. As part of DARPA's HPCS program, Cray, IBM, and Sun are exploring compiler and runtime technology for mapping global view languages to custom tightly coupled architectures. We propose to build on a decade of research on locality-aware compilation for HPF to compile more general global-view programs into efficient code for commodity distributed-memory platforms. Rice University will lead this research in collaboration with PNNL. Below, we outline three of the key challenges we will address.

Analysis and code generation for user-defined tiling distributions. A principal (and fair) criticism leveled at HPF was that the set of data distributions it supported was too limited. Chapel and Fortress propose to expand the set of data distributions available to user-defined hyper-rectangular partitionings of Cartesian domains such as the one shown in Figure 1. With such tilings, a tile may have neighboring tiles of various sizes and alignments. We propose to extend the set-based analysis framework developed for the Rice dHPF compiler to handle analysis and

code generation for such user-defined distributions. Our goal is to preserve the property of being able to statically generate symbolically parameterized code that enumerates the elements to be communicated between a pair of tiles. Managing such tilings efficiently will require tight integration between the compiler and runtime system.

Efficient computation partitioning of complex global view programs. Generalizing computation partitioning methods to complex loop nests with nonrectangular iteration spaces is a major challenge. Consider the iteration space of the FFT code shown in Figure 2. Partitioning the computation of the i loop involves restricting its bounds to the data elements mapped to a particular processor node; this is a well-understood application of the owner computes rule. Partitioning the k loop involves having each processor execute only the iterations whose i loops touch local data; its bounds must be inferred indirectly. Since the data is partitioned, some iterations in a processor’s $[i,k]$ iteration space will access non-local data. For this code, data dependences permit the region of a processor’s $[i,k]$ iteration space that accesses non-local data to be split off from other iterations to enable communication to be overlapped with computation. Realizing this overlap for FFT’s symbolically parameterized, strided, nonrectangular iteration space and mapping it onto p processors where $p \neq 2^k$ is fiendishly difficult. From this example, one begins to get a sense for when a formal framework for analysis and transformation is useful. We propose to explore the challenges in generating efficient partitioned loop nests for global-view programs with complex iteration spaces.

Compiling tightly coupled global view programs. A tightly coupled loop is one that both produces and consumes data values for an overlapping set of array elements; such loops are the hardest to parallelize by hand. For distribution-based global view languages to succeed, it must be possible to write such loops in these languages and compile them into efficient parallel code for distributed-memory systems. To understand the challenge, consider compiling a global-view implementation of Gaussian elimination with partial row pivoting for a distributed memory system. For highest parallel performance, the serial work of computing a row pivot should be overlapped with the parallel work of performing an elimination step on a submatrix. Achieving such a parallelization requires understanding the data dependences in the code, understanding the serialized nature of the pivot computation, and then radically restructuring the code by applying iteration space splitting in conjunction with a software pipelining transformation to compute the next pivot as early as possible so that it can be overlapped with parallel elimination work. Such code generation is well beyond the capabilities of today’s parallel compilers. We propose to explore the full range of code generation challenges for such tightly coupled global view loop nests.

4.6 Programming Patterns

In the past year, we began regular workshops on Patterns for High Performance Computing, established a centralized portal for this effort, and started expanding the catalog of collected patterns.

We expect to expand this activity by continuing with yearly workshops, engaging a broader community of DOE practitioners in the effort and expanding the catalog of documented patterns. A significant part of this effort will focus on patterns that document the activities of performance oriented programmers: “performance” is the middle name of high performance computing, and a significant amount of programming time in HPC is spent on performance tuning. We do not believe that compiler technology can replace expert performance tuners; rather, we expect better support for tuning in the new programming paradigms (e.g., via annotations).

Consider a simple example: one must often merge multiple parallel tasks into one sequential task, so that the number of tasks equal or be a small multiple of the number of processors. This “task coarsening” pattern is often easily supported in OpenMP (a parallel loop is replaced by a

sequential loop) but may require significant code rewriting if MPI or PGAS languages are used; the ability of different models to support such a pattern will not be apparent from examples of code but will be apparent if the pattern is documented and illustrated in the different proposed programming models.

We will produce an increasing catalog of parallel programming patterns, illustrated using the various programming models studied under this project. The final outcome will be a pattern language for high-performance computing: that is, a structured catalog of prevalent programming patterns for HPC, illustrated using some of the programming models of this project. In addition, we will produce a “synoptic table” illustrating for the most important patterns how they are handled in each of the proposed models. This table will be a measure of the expressiveness of the programming models—their ability to support well the most common activities of HPC programmers.

We will collaborate with the annotation effort described in Section 4.3.3 to ensure that key patterns are captured by the proposed annotations.

5 External Interactions

5.1 Co-Array Fortran Features in the Fortran Standard

The Rice PMODELS team has been engaged in a dialog with the Fortran J3 standardization committee, which is considering the addition of co-array language constructs into the Fortran 2008 standard. Rice will continue to work with the committee to ensure that the set of language constructs adopted as part of the standard are appropriate. Having co-array features as part of the Fortran standard will reassure applications scientists that developing codes using Co-array Fortran will not be a wasted investment and that they can count upon its long-term availability. Rice’s open-source `cafc` compiler will serve as a reference implementation that will provide a model for vendors adding support for co-array features to commercial Fortran compilers.

5.2 Work with the OpenMP Architecture Review Board

OpenMP is maintained by the OpenMP Architecture Review Board (ARB), an organization to which most hardware vendors, several software companies, and major users belong. The project team at the University of Houston has been actively involved in the work of the ARB since the first year of PMODELS and has participated in language and tools committees. OpenMP is under active development and is expected to evolve in the near future to meet the needs of new architectures and applications. The Houston team will continue to work on the ARB subcommittees, where it will focus on efforts to improve the expressivity and scalability of the language for scientific computing. In particular, the team will champion features to ensure that the language is better able to address the needs of multithreading architectures, to provide higher levels of expressivity, and to enable the appropriate expression of locality. Error-handling features, better support for object-oriented applications and support for tools are also of major concern. The new features must not compromise the high level of productivity that is one of the major benefits of this API. The Houston team will provide reference implementations of proposed language features within their OpenUH compiler infrastructure in order to demonstrate their usefulness.

5.3 The UPC Consortium

The PMODELS groups from LBNL and ANL have worked as part of the UPC Consortium (a group from academia, industry and government labs) on the UPC language specification, specifically the definition of a parallel I/O interface for UPC and an initial prototype implementation. LBNL has also been active in the specification of UPC collectives and the UPC memory model and is working with the University of Florida to interface their performance tool work to the Berkeley compiler.

5.4 DARPA High Productivity Computing Systems

Several of the participants in this group also work with the DARPA High Productivity Computing Systems Project. We are evaluating the DARPA-funded, vendor-designed HPCS languages (X10, Chapel, Fortress), developing compilation and communication techniques likely to be of use in implementing such languages, and developing the PGAS languages described here as potential transition languages to one or more future generation HPCS languages.

5.5 Applications

The UPC group plans to work closely with Andrew Johnson at AHPARC, who has developed a large CFD application in UPC for the Cray X1E, and would like to port it to clusters. The UPC group is also working on a sparse direct solver in UPC, with the long term objective of using it in the TOPS project and in accelerator modeling, and on the use of one-sided communication to optimize AMR codes from Phillip Colella's groups at LBNL. The Titanium project has an AMR benchmark developed jointly with Colella's group and a heart simulation developed in collaboration with Charles Peskin's group at NYU. In addition, many applications groups use PMODELS software, in particular MPI, without the direct involvement of the PMODELS team.

Within the DOE community, the Global Array library has been deployed in multiple application areas such as chemistry (the NWchem, Molpro, COLUMBUS, Molcas, GAMESS-UK projects), structural biology (in the project Data Intensive Computing for Complex Biological Systems), astrophysics (at ORNL), atmospheric chemistry of aerosols (PEGASUS project), bioinformatics (ScalaBLAST code). We will explore new collaborations with application scientists regarding deployment of Global Arrays in their codes. Recently a new effort was started to parallelize a visual analytics code InSPIRE (R&D-100 award winner in 1996) using Global Arrays, and this effort will also use a prototype of global procedure calls in ARMCI.

5.6 Interactions with Computer Vendors

OSU has been closely interacting with primary InfiniBand vendors (Mellanox, Cisco/Topspin, Voltaire, Silverstorm and PathScale), major server vendors (Intel, AMD, Sun, and IBM), major systems integrators (Linux Network, Appro and Microway), and ISVs (such as Fluent). OSU and LBNL have also worked with Etnus to have Totalview support for MVAPICH and Berkeley UPC, respectively. OSU is also closely working with the OpenIB consortium and the latest releases of MVAPICH and MVAPICH2 are currently available as an integrated part of the OpenIB SVN repository. LBNL is working with Cray on XT3 compiler support and with IBM on the evaluation and use of their RDMA support in GASNet. PNNL has been working with Cray and IBM on enhancements of their low-level communication layers as required for efficient support of one sided communications. Argonne collaborates with IBM, Cray, Intel, and Microsoft on MPI. All of them use MPICH2 as the core of their vendor-supplied MPI implementations. UH collaborates with Intel, Sun, IBM, SGI and other vendors on the OpenMP ARB with respect to OpenMP language, compiler and runtime issues.

6 Milestones

The detailed milestones for the work proposed here are described in the Appendix on a per-institution basis. Here we give a representative summary.

Year 1: Establishing the software base. Thread-safe MPI and ARMCI for interoperability. GASNet and ARMCI for BlueGene. Release of open source Titanium and `cafc` compilers. Prototype GA taskpools and XGA. New MPI release with Infiniband-specific collective communication. Thread subteams in OpenMP. Patterns workshop featuring MPI. New Web site.

Year 2: Focusing on global view of data. MPI extensions for remote memory access. User-defined global data structures in Global Arrays and Titanium. Compilation for global view with multi-level data structures. Locality awareness in OpenMP. Patterns workshop on single-node performance with annotations.

Year 3: Ensuring performance on leadership class machines. Topology awareness in MPI specification and Infiniband implementation. Multicore support in Global Arrays and Titanium. `cafc` compiler for leadership-class machines. Static and dynamic load balancing for GA taskpools, data aggregation for XGA. Enhanced synchronizations in OpenMP. Patterns workshop featuring a PGAS language.

Year 4: Adding Further Optimizations and Interoperability. MPI and GA for multicore systems. Optimization of ARMCI for leadership class machines. Self-tuning infrastructure for GASNet, hence UPC and Titanium. Compiler support for interoperability. GA taskpools for DAGs. OpenMP interoperability with MPI, GA, and PGAS languages. Workshop on patterns in PGAS languages for hierarchical machines.

Year 5: Rounding off. Second-generation annotations for single node performance. Port of ARMCI to DARPA HPCS systems. Release of full Titanium compiler for 10K or more processors. Interoperability between global view models (like HPCS languages) and PGAS languages. GA taskpools and XGA release. MPI on ultrascale Infiniband clusters. OpenMP on large platforms. Patterns book.

7 Project Management

The participants have worked well together during the first generation of the Center, and there has been considerable exchange of technology. We will coordinate with one another through yearly meetings and monthly conference calls, augmented by more intense interactions (visits, email conversations) on particular topics. The Center will maintain a Web site that will publicize Center publications, open source software related to the Center, and other working documents.

References

- [1] L. Adhianto and B. Chapman. Performance modeling and analysis of hybrid MPI and OpenMP applications. Technical report, University of Houston - Department of Computer Science, 2006.
- [2] V. Adve and J. Mellor-Crummey. Using Integer Sets for Data-Parallel Program Analysis and Optimization. In *Proceedings of the SIGPLAN '98 Conference on Programming Language Design and Implementation*, Montreal, Canada, June 1998.
- [3] A. Aiken and D. Gay. Barrier inference. In *Principles of Programming Languages, San Diego, California*, January 1998.
- [4] G. Almási, C. Archer, J. G. C. nos, J. A. Gunnels, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, K. Pinnow, J. Ratterman, B. SteinmacherBurow, W. Gropp, and B. Toonen. Design and implementation of message-passing services for the Blue Gene/L supercomputer. *IBM Journal of Research and Development*, 49(2/3):393–406, March/May 2005. Available at <http://www.research.ibm.com/journal/rd49-23.html>.
- [5] A. Auer, G. Baumgartner, D. E. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Krishnamoorthy, S. Krishnan, C.-C. Lam, M. Nooijen, R. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov. Automatic code generation for many-body electronic structure methods: The Tensor Contraction Engine. *Mol. Phys.*, 104(2):211–218, 20 January 2006.
- [6] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0. Technical Report NAS-95-020, NASA Ames Research Center, Dec. 1995.
- [7] G. Baumgartner, A. Auer, D. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Hirata, S. Krishnamoorthy, S. Krishnan, C. Lam, Q. Lu, M. Nooijen, R. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov. Synthesis of high-performance parallel programs for a class of Ab Initio quantum chemistry models. *Proceedings of the IEEE*, 93(2):276–292, 2005.
- [8] J. Beecroft, D. Addison, D. Hewson, M. McLaren, D. Roweth, F. Petrini, and J. Nieplocha. Qsnetii: Defining high-performance network design. *IEEE Micro*, 25(4), 2005.
- [9] C. Bell and D. Bonachea. A new DMA registration strategy for pinning-based high performance networks. In *Workshop Communication Architecture for Clusters (CAC03) of IPDPS'03, Nice, France*, 2002.
- [10] C. Bell, D. Bonachea, R. Nishtala, and K. Yelick. Optimizing bandwidth limited problems using one-sided communication and overlap. In *20th International Parallel and Distributed Processing Symposium (IPDPS)*, Apr. 2006. Also available as Lawrence Berkeley National Lab Tech Report LBNL-59207.
- [11] D. Bernholdt, J. Nieplocha, and P. Sadayappan. Raising the level of programming abstraction in scalable programming models. In *Proceedings of the HPCA Workshop on Productivity and Performance in High-End Computing*. IEEE Computer Society, 2004.
- [12] H. Boehm and M. Weiser. Garbage collection in an uncooperative environment. *Software Practice and Experience*, pages 807–820, Sept. 1988.

- [13] D. Bonachea. GASNet specification, v1.1. Technical Report UCB/CSD-02-1207, University of California—Berkeley, 2002. Updated v1.6 specification available at <http://gasnet.cs.berkeley.edu/dist/docs/gasnet.pdf>.
- [14] D. Buntinas and W. Gropp. Designing a common communication subsystem. In B. D. Martino, D. Kranzluüller, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, number LNCS 3666 in Lecture Notes in Computer Science, pages 156–166. Springer Verlag, Sept. 2005. 12th European PVM/MPI User’s Group Meeting, Sorrento, Italy.
- [15] D. Buntinas, G. Mercier, and W. Gropp. The design and evaluation of Nemesis, a scalable low-latency message-passing communication subsystem. Technical Report ANL/MCS-TM-292, Argonne National Laboratory, 2005.
- [16] D. Buntinas and D. K. Panda. Implementing NIC-level support for atomic operations in Myrinet. In *SAN-1 Workshop, held in conjunction with HPCA-8*, 2002.
- [17] D. Buntinas and D. K. Panda. NIC-based reduction in Myrinet clusters: Is it beneficial? In *SAN-02 Workshop (in conjunction with HPCA)*, Feb 2003.
- [18] D. Buntinas, D. K. Panda, and R. Brightwell. Application-bypass broadcast in MPICH over GM. In *Proceedings of Cluster Computing and Grid ’03*, May 2003.
- [19] D. Buntinas, D. K. Panda, and P. Sadayappan. Fast NIC-level barrier over Myrinet/GM. In *Proceedings of Int’l Parallel and Distributed Processing Symposium (IPDPS)*, 2001.
- [20] D. Buntinas, D. K. Panda, and P. Sadayappan. Performance evaluation of NIC-level barrier over Myrinet/GM. In *Proceedings of Int’l Workshop on Communication Architecture for Clusters (CAC)*, 2001.
- [21] D. Buntinas, A. Saify, D. Panda, and J. Nieplocha. Optimizing synchronization operations for remote memory communication systems. In *Parallel and Distributed Processing Symposium*, page 8 pp., 2003. TY - CONF.
- [22] D. Callahan, B. L. Chamberlain, and H. P. Zima. The Cascade high productivity language. In *Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS’04)*, Santa Fe, NM, Apr. 2004.
- [23] W. W. Carlson, J. M. Draper, D. E. Culler, K. Yelick, and K. W. E. Brooks. Introduction to UPC and language specification. Technical Report CCS-TR-99-157, IDA Center for Computing Sciences, May 1999.
- [24] U. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. *IEEE TPDS*, 10(7):673–693, 1999.
- [25] U. V. Çatalyürek and C. Aykanat. *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*. Bilkent University, Department of Computer Engineering, 1999.
- [26] A. Chan, D. Ashton, E. Lusk, and W. Gropp. Jumpshot-4 users guide. <http://www-unix.mcs.anl.gov/perfvis/software/viewers/jumpshot-4/usersguide.html>, June 2003.
- [27] B. Chapman, F. Bregier, A. Patil, and A. Prabhakar. Achieving high performance under OpenMP on ccNUMA and software distributed share memory systems. *Concurrency and Computation Practice and Experience*, 14:1–17, 2002.

- [28] B. Chapman, A. Patil, and A. Prabhakar. Performance oriented programming for NUMA architectures. In *OpenMP Shared Memory Parallel Programming: International Workshop on OpenMP Applications and Tools, WOMPAT 2001*, pages 137–154. Springer-Verlag Heidelberg, 2001.
- [29] B. M. Chapman, L. Huang, G. Jost, H. Jin, and B. R. de Supinski. Support for flexibility and user control of worksharing in OpenMP. Technical Report NAS-05-015, National Aeronautics and Space Administration, October 2005.
- [30] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications*, pages 519–538, New York, NY, USA, 2005. ACM Press.
- [31] D. Chavarría-Miranda, G. Jin, and J. Mellor-Crummey. COTS clusters vs. the Earth Simulator: An application study using IMPACT-3D. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, Denver, CO, Apr. 2005.
- [32] D. Chavarría-Miranda and J. Mellor-Crummey. An evaluation of data-parallel compiler support for line-sweep applications. In *Eleventh International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Charlottesville, VA, Sept. 2002. ACM.
- [33] D. Chavarría-Miranda and J. Mellor-Crummey. An evaluation of data-parallel compiler support for line-sweep applications. *The Journal of Instruction-Level Parallelism*, 5, February 2003. (<http://www.jilp.org/vol5>).
- [34] D. Chavarría-Miranda and J. Mellor-Crummey. Effective communication coalescing for data parallel applications. In *Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2005)*, Chicago, Illinois, June 2005.
- [35] D. Chavarría-Miranda, J. Mellor-Crummey, and T. Tsarang. Data-parallel compiler support for multipartitioning. Technical Report CS-TR01-374, Dept. of Computer Science, Rice University, Mar. 2001.
- [36] D. Chavarría-Miranda, J. Nieplocha, and V. Tipparaju. Topolgy-aware Tile Mapping for Clusters of SMPs. In *Proceedings of the ACM International Conference on Computing Frontiers*, Ischia, Italy, May 2006.
- [37] W. Chen, C. Iancu, and K. Yelick. Communication optimizations for fine-grained upc applications. In *14th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2005.
- [38] W. Chen, A. Krishnamurthy, and K. Yelick. Polynomial-time algorithms for enforcing sequential consistency in spmd programs with arrays. In *16th International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, 2003.
- [39] Y. Chen, J. Li, S. Wang, and D. Wang. ORC-OpenMP: An OpenMP compiler based on ORC. In *International Conference on Computational Science*, pages 414–423, 2004.
- [40] Chombo project web site. seesar.lbl.gov/anag/chombo/.

- [41] C. Coarfa, Y. Dotsenko, J. Eckhardt, and J. Mellor-Crummey. Co-Array Fortran Performance and Potential: An NPB Experimental Study. In *Proc. of the 16th Intl. Workshop on Languages and Compilers for Parallel Computing*, number 2958 in LNCS. Springer-Verlag, October 2-4, 2003.
- [42] C. Coarfa, Y. Dotsenko, and J. Mellor-Crummey. Experiences with Sweep3D implementations in Co-Array Fortran. In *Proceedings of the Los Alamos Computer Science Institute Fifth Annual Symposium*, Santa Fe, NM, Oct. 2004. Distributed on CD-ROM.
- [43] C. Coarfa, Y. Dotsenko, J. Mellor-Crummey, F. Cantonnet, T. El-Ghazawi, A. Mohanti, Y. Yao, and D. Chavarría-Miranda. An evaluation of Global Address Space Languages: Co-Array Fortran and Unified Parallel C. In *Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2005)*, Chicago, Illinois, June 2005.
- [44] cOMPunity - the community of OpenMP users. <http://www.compunity.org/>.
- [45] A. Darté, D. Chavarría-Miranda, R. Fowler, and J. Mellor-Crummey. Generalized multi-partitioning for multi-dimensional arrays. In *Proceedings of the International Parallel and Distributed Processing Symposium*, Fort Lauderdale, FL, Apr. 2002.
- [46] A. Darté, J. Mellor-Crummey, R. Fowler, and D. Chavarría-Miranda. Generalized multi-partitioning of multi-dimensional arrays for parallelizing line-sweep applications. *Journal of Parallel and Distributed Computing*, 63(9):887–911, Sept. 2003.
- [47] K. Datta, D. Bonachea, and K. Yelick. Titanium performance and potential: an npb experimental study. In *Languages and Compilers for Parallel Computing (LCPC)*, 2005.
- [48] Y. Dotsenko, C. Coarfa, and J. Mellor-Crummey. A Multiplatform Co-Array Fortran Compiler. In *Proceedings of the 13th Intl. Conference of Parallel Architectures and Compilation Techniques*, Antibes Juan-les-Pins, France, September 29 - October 3 2004.
- [49] Y. Dotsenko, C. Coarfa, J. Mellor-Crummey, and D. Chavarría-Miranda. Experiences with Co-Array Fortran on Hardware Shared Memory Platforms. In *Proceedings of the 17th International Workshop on Languages and Compilers for Parallel Computing*, September 2004.
- [50] Eclipse software framework. <http://www.eclipse.org>, 2002.
- [51] C. Falzone, A. Chan, E. Lusk, and W. Gropp. Collective error detection for MPI collective operations. In B. D. Martino, D. Kranzluüller, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, number LNCS 3666 in Lecture Notes in Computer Science, pages 138–147. Springer Verlag, Sept. 2005. 12th European PVM/MPI User’s Group Meeting, Sorrento, Italy.
- [52] I. Foster and J. Nieplocha. *Disk Resident Arrays: An Array-Oriented I/O Library for Out-of-Core Computations, Services for Distributed System Integration*, pages 488–498. High-Performance Mass Storage and Parallel I/O. IEEE and Wiley Press, 2002.
- [53] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. special issue on "Program Generation, Optimization, and Platform Adaptation".

- [54] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. In *PLDI '98: Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation*, pages 212–223, New York, NY, USA, 1998. ACM Press.
- [55] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, Boston, MA, 1995.
- [56] E. Givelberg and K. Yelick. Distributed immersed boundary simulation in titanium. *SIAM Journal on Scientific Computing*, 2006. To appear.
- [57] W. Gropp, D. Gunter, and V. Taylor. FPMPI: A fine-tuning performance profiling library for MPI, Nov. 2001. Poster presented at SC2001.
- [58] W. Gropp and R. Thakur. An evaluation of implementation options for MPI one-sided communication. In B. D. Martino, D. Kranzluüller, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, number LNCS 3666 in Lecture Notes in Computer Science, pages 415–424. Springer Verlag, Sept. 2005. 12th European PVM/MPI User’s Group Meeting, Sorrento, Italy.
- [59] W. D. Gropp. Runtime checking of datatype signatures in MPI. In J. Dongarra, P. Kacsuk, and N. Podhorszki, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, number 1908 in Springer Lecture Notes in Computer Science, pages 160–167, Sept. 2000. 7th European PVM/MPI Users’ Group Meeting.
- [60] M. F. Guest, E. Apra, D. E. Bernholdt, H. A. Fruchtl, R. J. Harrison, R. A. Kendall, R. A. Kutteh, X. Long, J. B. Nicholas, J. A. Nichols, H. L. Taylor, A. T. Wong, G. I. Fann, R. J. Littlefield, and J. Nieplocha. High-performance computing in chemistry: NWChem. *Future Generation Computer Systems*, 12(4):273–289, 1996. Article.
- [61] M. Gupta and E. Schonberg. Static analysis to reduce synchronization costs of data-parallel programs. In *ACM Symposium on Principles of Programming Languages (POPL)*, 1996.
- [62] R. Gupta, V. Tipparaju, J. Nieplocha, and D. Panda. Efficient barrier using remote memory operations on via-based clusters. In *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, pages 83–90, 2002. TY - CONF.
- [63] L. Huang, B. Chapman, and R. Kendall. OpenMP for clusters. In *the Fifth European Workshop on OpenMP, EWOMP'03*, Aachen, Germany, 2003.
- [64] L. Huang, B. Chapman, and Z. Liu. Towards a more efficient implementation of OpenMP for clusters via translation to Global Arrays. *Parallel Computing*, 31(10-12), 2005.
- [65] W. Huang, G. Santhanaraman, H.-W. Jin, and D. K. Panda. Scheduling of MPI-2 one sided operations over InfiniBand. In *Proceedings of Workshop on Communication Architecture for Clusters (CAC 2005); In Conjunction with the International Parallel and Distributed Processing Symposium*, 2005.
- [66] P. Husbands and K. Yelick. Parallel triangulation in Unified Parallel C (UPC). In *SIAM Meeting on Parallel Processing for Scientific Computing*, Feb. 2004. Presentation only.
- [67] C. Iancu, P. Husbands, and W. Chen. Message strip mining heuristics for high speed networks. In *VECPAR*, 2004.

- [68] C. Iancu, P. Husbands, and P. Hargrove. Hunting the overlap. In *14th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2005.
- [69] W. Jiang, J. Liu, H. Jin, D. K. Panda, W. Gropp, and R. Thakur. High performance MPI-2 one-sided communication over InfiniBand. In *IEEE/ACM International Symposium on Cluster Computing and the Grid*, Chicago, IL, April 2004.
- [70] W. Jiang, J. Liu, H.-W. Jin, D. K. Panda, D. Buntinas, R. Thakur, and W. Gropp. Efficient implementation of MPI-2 passive one-sided communication on InfiniBand clusters. In *Proceedings of EuroPVM/MPI '04*, Budapest, Hungary, September 2004.
- [71] W. Jiang, J. Liu, H.-W. Jin, D. K. Panda, D. Buntinas, R. Thakur, and W. Gropp. Efficient implementation of MPI-2 passive one-sided communication on InfiniBand clusters. In D. Kranzlmüller, P. Kacsuk, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, number LNCS3241 in Lecture Notes in Computer Science, pages 68–76. Springer Verlag, 2004. 11th European PVM/MPI User’s Group Meeting, Budapest, Hungary.
- [72] A. Kamil, J. Su, and K. Yelick. Making sequential consistency practical in titanium. In *Supercomputing, SC05*, 2005.
- [73] A. Kamil and K. Yelick. Concurrency analysis for parallel programs with textually aligned barriers. In *18th International Workshop on Languages and Compilers for Parallel Computing*, Hawthorne, New York, Oct. 2005.
- [74] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. The Omega Library Interface Guide. Technical report, Dept. of Computer Science, Univ. of Maryland, College Park, Apr. 1996.
- [75] W. Kelly, W. Pugh, and E. Rosser. Code generation for multiple mappings. In *Frontiers '95: The 5th Symposium on the Frontiers of Massively Parallel Computation*, McLean, VA, Feb. 1995.
- [76] R. A. Kendall, E. Apra, D. E. Bernholdt, E. J. Bylaska, M. Dupuis, G. I. Fann, R. J. Harrison, J. L. Ju, J. A. Nichols, J. Nieplocha, T. P. Straatsma, T. L. Windus, and A. T. Wong. High performance computational chemistry: An overview of NWChem a distributed parallel application. *Computer Physics Communications*, 128(1-2):260–283, 2000. Article.
- [77] S. P. Kini, J. Liu, J. Wu, P. Wyckoff, and D. K. Panda. Fast and scalable barrier using RDMA and multicast mechanisms for InfiniBand-based clusters. In *EuroPVM/MPI*, Oct. 2003.
- [78] S. Krishnamoorthy, U. Catalyurek, J. Nieplocha, A. Rountev, and P. Sadayappan. An extensible global address space framework with decoupled task and data abstractions. In *Proc. IPDPS Workshop on Next Generation Software*, 2006.
- [79] S. Krishnamoorthy, U. Catalyurek, J. Nieplocha, and P. Sadayappan. An approach to locality-conscious load balancing and transparent memory hierarchy management with a global-address-space parallel programming model. In *Proc. IPDPS Workshop on Performance Optimization of High Level Languages and Libraries*, 2006.

- [80] S. Krishnamoorthy, J. Nieplocha, and P. Sadayappan. Data and computation abstractions for dynamic and irregular computations. In *Proc. Intl. Conference on High Performance Computing*, 2005.
- [81] M. Krishnan, Y. Alexeev, T. L. Windus, and J. Nieplocha. Multilevel parallelism in computational chemistry using common component architecture and global arrays. In *Proceedings of SuperComputing*. ACM and IEEE, 2005.
- [82] M. Krishnan, Y. Alexeev, T. L. Windus, and J. Nieplocha. Multilevel parallelism in computational chemistry using common component architecture and global arrays. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 23, Washington, DC, USA, 2005. IEEE Computer Society.
- [83] M. Krishnan and J. Nieplocha. Optimizing parallel multiplication operation for rectangular and transposed matrices. In *10th IEEE International Conference on Parallel and Distributed Systems (ICPADS'04)*., 2004.
- [84] M. Krishnan and J. Nieplocha. SRUMMA: a matrix multiplication algorithm suitable for clusters and scalable shared memory systems. In *Parallel and Distributed Processing Symposium*, pages 70–79, 2004. TY - CONF.
- [85] M. Krishnan and J. Nieplocha. SRUMMA: a matrix multiplication algorithm suitable for clusters and scalable shared memory systems. In *Parallel and Distributed Processing Symposium*, pages 70–79, 2004. TY - CONF.
- [86] M. Krishnan and J. Nieplocha. Optimizing performance on linux clusters using advanced communication protocols: Achieving over 10 teraflops on an 8.6 teraflops linpack-rated Linux cluster. In *Proceedings of the 6th International Conference on Linux Clusters: The HPC Revolution*, 2005.
- [87] M. Krishnan and J. Nieplocha. Memory Efficient Parallel Matrix Multiplication Operation for Irregular Problems. In *Proceedings of the ACM International Conference on Computing Frontiers*, Ischia, Italy, May 2006.
- [88] M. Krishnan, V. Tipparaju, B. Palmer, and J. Nieplocha. Processor-group aware runtime support for shared-and global address space models. In *Proceedings of International Conference on Parallel Processing, ICPP 2004 Workshops*, pages 506–513, August 2004.
- [89] C. Liao, O. Hernandez, B. Chapman, W. Chen, and W. Zheng. OpenUH: An optimizing, portable OpenMP compiler. In *12th Workshop on Compilers for Parallel Computers*, 2006.
- [90] C. Liao, Z. Liu, L. Huang, and B. Chapman. Evaluating OpenMP on chip multithreading platforms. In *First international workshop on OpenMP*, Eugene, Oregon USA, June 2005.
- [91] B. Liblit, A. Aiken, and K. Yelick. Type systems for distributed data sharing. In *International Static Analysis Symposium*, San Diego, California, June 2003.
- [92] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and implementation of MPICH2 over InfiniBand with RDMA support. In *Proceedings of Int'l Parallel and Distributed Processing Symposium*, April 2004.

- [93] J. Liu, A. Mamidala, and D. K. Panda. High performance MPI-level broadcast on InfiniBand with hardware multicast support. In *International Parallel and Distributed Processing Symposium*, 2004.
- [94] J. Liu, A. Vishnu, and D. K. Panda. Building multirail InfiniBand clusters: MPI-level design and performance evaluation. In *SuperComputing Conference*, 2004.
- [95] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High performance RDMA-based MPI implementation over InfiniBand. In *17th Annual ACM International Conference on Supercomputing*, June 2003.
- [96] Z. Liu, B. Chapman, Y. Wen, L. Huang, and O. Hernandez. Analyses for the translation of OpenMP codes into SPMD style with array privatization. In *OpenMP Shared Memory Parallel Programming: International Workshop on OpenMP Applications and Tools, WOMPAT 2003, June 26-27, 2003. Proceedings*, volume 2716 of *Lecture Notes in Computer Science*, pages 26–41. Springer-Verlag Heidelberg, June 2003.
- [97] Z. Liu, B. Chapman, T.-H. Weng, and O. Hernandez. Improving the performance of OpenMP by array privatization. In *OpenMP Shared Memory Parallel Programming: International Workshop on OpenMP Applications and Tools (WOMPAT '03)*, pages 244–259. Springer-Verlag Heidelberg, 2003.
- [98] Z. Liu, L. Huang, B. Chapman, and T.-H. Weng. Efficient implementation of OpenMP for clusters with implicit data distribution. In *OpenMP Shared Memory Parallel Programming: International Workshop on OpenMP Applications and Tools (WOMPAT 2004)*, pages 121–136. Springer-Verlag Heidelberg, 2004.
- [99] Looptool. <http://lacs.i.rice.edu/software/looptool>.
- [100] A. Mamidala, H. W. Jin, and D. K. Panda. Efficient hardware multicast group management for multiple MPI communicators over InfiniBand. In *EuroPVM/MPI*, 2005.
- [101] A. Mamidala, J. Liu, and D. K. Panda. Efficient barrier and allreduce on IBA clusters using hardware multicast and adaptive algorithms. In *IEEE Cluster Computing*, 2004.
- [102] T. Mattson, B. Sanders, and B. Massingill. *Patterns for Parallel Programming*. Addison-Wesley, Boston, MA, 2004.
- [103] J. Mellor-Crummey and V. Adve. Simplifying control flow in compiler-generated parallel code. *International Journal of Parallel Programming*, 26(5), 1998.
- [104] J. Mellor-Crummey, V. Adve, B. Broom, D. C. a Miranda, R. Fowler, G. Jin, K. Kennedy, and Q. Yi. Advanced optimization strategies in the Rice dHPF compiler. *Concurrency: Practice and Experience*, 14(8-9):741–767, 2002.
- [105] A. Moody, J. Fernandez, F. Petrini, and D. Panda. Scalable NIC-based reduction on large-scale clusters. In *SC '03*, November 2003.
- [106] MPI Forum. MPI documents page. <http://www.mpi-forum.org/docs/docs.html>.
- [107] M. S. Müller, C. Niethammer, B. Chapman, Y. Wen, and Z. Liu. Validating OpenMP 2.5 for Fortran and C/C++. In *Sixth European Workshop on OpenMP*, KTH Royal Institute of Technology, Stockholm, Sweden, October 2004.

- [108] Network-Based Computing Laboratory. MVAPICH: MPI for InfiniBand on VAPI Layer. <http://nowlab.cse.ohio-state.edu/projects/mpi-iba>.
- [109] J. Nieplocha, E. Apra, J. Ju, and V. Tipparaju. One-sided communication on clusters with Myrinet. *Cluster Computing*, 6(2):115–124, 2003. TY - JOUR.
- [110] J. Nieplocha, D. Baxter, V. Tipparaju, C. Rasmunssen, and R. W. Numrich. Symmetric data objects and remote memory access communication for Fortran-95 applications. In *Lecture Notes in Computer Science*, volume 3648, pages 720 – 729, August 2005.
- [111] J. Nieplocha and B. Carpenter. *ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-Time Systems*, volume 1586 of *Lecture Notes in Computer Science*, pages 533–546. Springer-Verlag, 1999.
- [112] J. Nieplocha and I. Foster. Disk resident arrays: an array-oriented i/o library for out-of-core computations. In *Frontiers of Massively Parallel Computing*, pages 196–204, 1996. TY - CONF.
- [113] J. Nieplocha, J. Harrison, R., and I. Foster. Explicit management of memory hierarchy. *Advances in High Performance Computing*, NATO ASI 3/30:185–200, 1996.
- [114] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global arrays: A portable shared memory programming model for distributed memory computers. In *Supercomputing*, pages 340–349. IEEE CS Press, 1994.
- [115] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global arrays: A nonuniform memory access programming model for high-performance computers. *Journal of Supercomputing*, 10(2):169–189, 1996. Article.
- [116] J. Nieplocha, J. L. Ju, and T. P. Straatsma. A multiprotocol communication support for the global address space programming model on the ibm sp. In *Euro-Par 2000 Parallel Processing, Proceedings*, volume 1900 of *Lecture Notes in Computer Science*, pages 718–728. 2000. Article.
- [117] J. Nieplocha, M. Krishnan, B. Palmer, V. Tipparaju, and Y. Zhang. Exploiting processor groups to extend scalability of the GA shared memory programming model. In *ACM Computing Frontiers*, Italy, 2005.
- [118] J. Nieplocha, M. Krishnan, B. Palmer, V. Tipparaju, and Y. Zhang. Exploiting processor groups to extend scalability of the GA shared memory programming model. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 262–272, New York, NY, USA, 2005. ACM Press.
- [119] J. Nieplocha, B. Palmer, V. Tipparaju, M. K. ishnan, H. Trease, and E. Apra. Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit. *International Journal of High Performance Computinga and Applications*, 20(2), 2006.
- [120] J. Nieplocha, V. Tipparaju, and Krishnan. Optimizing strided remote memory access operations on the Quadrics QsNet-II network interconnect. In *HPC-Asia 2005*, Beijing, China, 2005.

- [121] J. Nieplocha, V. Tipparaju, M. Krishnan, and D. Panda. High Performance Remote Memory Access Communications: The ARMCI Approach. *International Journal of High Performance Computing and Applications*, 20(2), 2006.
- [122] J. Nieplocha, V. Tipparaju, M. Krishnan, G. Santhanaraman, and D. Panda. Optimizing mechanisms for latency tolerance in remote memory access communication on clusters. In *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, pages 138–147, 2003. TY - CONF.
- [123] J. Nieplocha, V. Tipparaju, M. Krishnan, G. Santhanaraman, and D. Panda. Optimization and performance evaluation of mechanisms for latency tolerance in remote memory access communication on clusters. *Int. J. High Performance Computing and Networking*, 2(2/3/4):198–209, 2004.
- [124] J. Nieplocha, V. Tipparaju, A. Saify, and D. K. Panda. Protocols and strategies for optimizing performance of remote memory operations on clusters. In *Communication Architecture for Clusters (CAC'02) Workshop, held in conjunction with IPDPS '02*, pages 164 – 173, 2002.
- [125] R. W. Numrich and J. K. Reid. Co-Array Fortran for parallel programming. Technical Report RAL-TR-1998-060, Rutheford Appleton Laboratory, August 1998.
- [126] R. W. Numrich and J. K. Reid. Co-Array Fortran for parallel programming. *ACM Fortran Forum*, 17(2):1–31, August 1998.
- [127] C. Oehmen and J. Nieplocha. ScalaBLAST: A scalable implementation of BLAST for high performance data-intensive bioinformatics analysis. *IEEE Trans. Parallel Dist. Sys. Special issue on high-performance computational biology*, 2006. To appear.
- [128] L. Oliker, A. Canning, J. Carter, and J. Shalf. Scientific computations on modern parallel vector systems. In *Proceedings of SC2004*, Pittsburgh, PA, Nov. 2004. IEEE Computer Society Press.
- [129] The OpenUH compiler project. <http://www.cs.uh.edu/~openuh>, 2005.
- [130] B. Palmer, J. Nieplocha, and E. Apra. Shared memory mirroring for reducing communication overhead on commodity networks. In *International Conference on Cluster Computing*, pages 420–428, 2003. TY - CONF.
- [131] R. Panuganti, M. M. Baskaran, D. Hudak, A. Krishnamurthy, J. Nieplocha, A. Rountev, and P. Sadayappan. GAMMA: Global Arrays meets MATLAB. Technical Report OSU-CISRC-1/06-TR15, Ohio State University, Jan. 2006.
- [132] K. Parzyszek, J. Nieplocha, and R. A. Kendall. Generalized portable shmем library for high performance computing. In M. Guizani and X. Shen, editors, *IASTED Parallel and Distributed Computing and Systems*, pages 401–406, Las Vegas, Nevada, 2000. IASTED.
- [133] Pathscale EKOPATH compiler suite for AMD64 and EM64T. <http://www.pathscale.com/ekopath.html>, 2006.
- [134] Perfsuite. <http://perfsuite.ncsa.uiuc.edu/>.
- [135] Pmodels project web site. <http://www.pmodels.org>.

- [136] A. Prabhakar, V. Getov, and B. Chapman. Performance comparisons of basic OpenMP constructs. *Lecture Notes in Computer Science*, 2327:413–424, 2002.
- [137] W. Pugh. A practical algorithm for exact array dependence analysis. *Communications of the ACM*, 35(8):102–114, Aug. 1992.
- [138] A. Qasem, G. Jin, and J. Mellor-Crummey. Improving performance with integrated program transformations. Technical Report TR03-419, Rice University, Department of Computer Science, Oct. 2003.
- [139] C. E. Rasmussen, M. J. Sottile, J. Nieplocha, R. W. Numrich, and E. Jones. Co-array python: A parallel extension to the python language. In *Proceedings Euro-Par*, pages 632–637, 2004.
- [140] A. Rogers and K. Pingali. Process decomposition through locality of reference. In *Proceedings of the SIGPLAN '89 Conference on Programming Language Design and Implementation*, Portland, OR, June 1989.
- [141] M. Rosing, J. Nieplocha, and S. Yabusaki. Toward efficient compilation of user-defined extensible Fortran directives. In *HIPS*, pages 61–69. IEEE Computer Society, 2004.
- [142] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, Chichester, Great Britain, 1986.
- [143] G. Shah, J. Nieplocha, J. Mirza, C. Kim, R. Harrison, R. Govindaraju, K. Gildea, P. DiNicola, and C. Bender. Performance and experience with lapi-a new high-performance communication library for the ibm rs/6000 sp. In *International Parallel Processing Symposium IPPS/SPDP*, pages 260–266, 1998. TY - CONF.
- [144] M. Snir and T. Mattson. Patterns for high performance computing. Presentation at Dagstuhl Seminar on Architectures and Algorithms for Petascale Computing, February 2006.
- [145] SPHINX. <http://www.llnl.gov/casc/sphinx/sphinx.html>.
- [146] H. Su, B. Gordon, S. Oral, and A. George. Sci networking for shared-memory computing in upc: Blueprints of the gasnet sci conduit. In *IEEE Workshop on High Speed Local Networks (HSLN)*, 2004.
- [147] J. Su and K. Yelick. Automatic support for irregular computations in a high-level language. In *19th International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.
- [148] Sun Microsystems. The Fortress language specification, version 0.866. <http://research.sun.com/projects/plrg/fortress0866.pdf>.
- [149] S. Sur, U. Bondhugula, A. Mamidala, H.-W. Jin, and D. K. Panda. High performance RDMA based all-to-all broadcast for InfiniBand clusters. In *International Conference on High Performance Computing*, 2005.
- [150] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda. Shared receive queue based scalable MPI design for InfiniBand clusters. In *International Parallel and Distributed Processing Symposium*, April 2006.
- [151] S. Sur, H.-W. Jin, L. Chai, and D. K. Panda. RDMA read based rendezvous protocol for MPI over InfiniBand: Design alternatives and benefits. In *Symposium on Principles and Practice of Parallel Programming*, March 2006.

- [152] S. Sur, H.-W. Jin, and D. K. Panda. Efficient and scalable all-to-all exchange for InfiniBand-based clusters. In *International Conference on Parallel Processing*, August 2004.
- [153] TAU - tuning and analysis utilities. <http://www.cs.uoregon.edu/research/tau/home.php>, 2006.
- [154] R. Thakur and W. Gropp. Improving the performance of collective operations in MPICH. In J. Dongarra, D. Laforenza, and S. Orlando, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, number LNCS2840 in Lecture Notes in Computer Science, pages 257–267. Springer Verlag, 2003. 10th European PVM/MPI User’s Group Meeting, Venice, Italy.
- [155] R. Thakur, W. Gropp, and B. Toonen. Minimizing synchronization overhead in the implementation of MPI one-sided communication. In D. Kranzlmüller, P. Kacsuk, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, number LNCS3241 in Lecture Notes in Computer Science, pages 57–67. Springer Verlag, 2004. 11th European PVM/MPI User’s Group Meeting, Budapest, Hungary.
- [156] R. Thakur, W. Gropp, and B. Toonen. Optimizing the synchronization operations in MPI one-sided communication. *High Performance Computing Applications*, 19(2):119–128, 2005.
- [157] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computer Applications*, 19(1):49–66, 2005.
- [158] V. Tipparaju, M. Krishnan, J. Nieplocha, G. Santhanaraman, and D. Panda. Exploiting non-blocking remote memory access communication in scientific benchmarks. In *High Performance Computing - HiPC*, volume 2913 of *Lecture Notes in Computer Science*, pages 248–258. Springer, 2003. Article.
- [159] V. Tipparaju, M. Krishnan, J. Nieplocha, G. Santhanaraman, and D. K. Panda. Exploiting non-blocking remote memory access communication in scientific benchmarks. In *International Conference on High Performance Computing (HiPC’03)*, Hyderabad, India, December 2003.
- [160] V. Tipparaju and J. Nieplocha. Optimizing all-to-all collective communication by exploiting concurrency in modern networks. In *ACM/IEEE SC 2005 Conference (SC’05)*, Seattle, Washington, November 2005.
- [161] V. Tipparaju, J. Nieplocha, and D. Panda. Fast collective operations using shared and remote memory access protocols on clusters. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, page 10 pp., 2003. TY - CONF.
- [162] V. Tipparaju, J. Nieplocha, and D. K. Panda. Fast collective operations using shared and remote memory access protocols on clusters. In *Int’l Parallel and Distributed Processing Symposium (IPDPS ’03)*, Apr. 2003.
- [163] V. Tipparaju, G. Santhanaraman, J. Nieplocha, and D. K. Panda. Host-assisted zero-copy remote memory access communication on infiniband. In *18th International Parallel and Distributed Processing Symposium Conference (IPDPS’04)*, Santa Fe, New Mexico, April 2004.
- [164] R. University. `caf`: An open-source compiler for Co-array Fortran. <http://www.hipersoft.rice.edu/caf>.

- [165] R. F. Van der Wijngaart. Efficient implementation of a 3-dimensional ADI method on the iPSC/860. In *Proceedings of Supercomputing 1993*, pages 102–111. IEEE Computer Society Press, 1993.
- [166] A. Vishnu, G. Santhanaraman, W. Huang, H.-W. Jin, and D. K. Panda. Supporting MPI-2 one sided communication on multi-rail InfiniBand clusters: Design challenges and performance benefits. In *International Conference on High Performance Computing*, 2005.
- [167] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 256–266, Gold Coast, Australia, May 1992.
- [168] T. Wen and P. Colella. Adaptive mesh refinement in titanium. In *19th International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.
- [169] T.-H. Weng and B. Chapman. Implementing OpenMP using dataflow execution model for data locality and efficient parallel execution. In *Proceedings of the 7th workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS-7)*. IEEE Press, 2002.
- [170] T.-H. Weng and B. Chapman. Toward optimization of OpenMP codes for synchronization and data reuse. In *The 2nd Workshop on Hardware/Software Support for High Performane Scientific and Engineering Computing (SHPSEC-03), in conjunction with the 12th International Conference on Parallel Architectures and Compilation Techniques (PACT-03)*, 2003.
- [171] F. Wolf and B. Mohr. Automatic performance analysis of hybrid MPI/OpenMP applications. *J. Syst. Archit.*, 49(10-1):421–439, 2003.
- [172] C. E. Wu, A. Bolmarcich, M. Snir, D. Wootton, F. Parpia, A. Chan, E. L. Lusk, and W. Gropp. From trace generation to visualization: A performance framework for distributed parallel systems. In *Proceedings of SC2000*, 2000.
- [173] K. A. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. N. Hilfinger, S. L. Graham, D. Gay, P. Colella, and A. Aiken. Titanium: A high-performance Java dialect. *Concurrency: Practice and Experience*, 10(11–13), September–November 1998.
- [174] W. Yu, D. Buntinas, and D. K. Panda. High performance and reliable NIC-based multicast over Myrinet/GM-2. In *Proceedings of the International Conference on Parallel Processing '03*, October 2003.
- [175] W. Yu, Q. Gao, and D. K. Panda. Adaptive connection management for scalable MPI over InfiniBand. In *International Parallel and Distributed Processing Symposium*, April 2006.
- [176] Y. Zhang, V. Tipparaju, J. Nieplocha, and S. Hariri. Parallelization of the NAS conjugate gradient benchmark using the global arrays shared memory programming model. In *19th IEEE International Parallel and Distributed Processing Symposium, 2005*, April 2005.