

Center for Programming Models for Scalable Parallel Computing

Annual Report

January, 2008

1. Introduction

Programming Models have emerged as the critical topic for making progress in the applications of large-scale computing to scientific problems. They stand at the intersection of the roads to productivity of programmers and performance of the machines they use. The Center for Programming Models for Scalable Parallel Computing (Pmodels) comprises researchers at national laboratories and universities focusing on a fundamental problem facing high-performance computing today: How will the application programs of the future be written so as to exploit the exiting directions in hardware architecture for new science? For details on the project's personnel, goals, and approach, see <http://pmodels.mcs.anl.gov>.

2. Recent Developments

Several developments in the overall environment for this project have taken place since the project started.

- New leadership-class computers have been installed at Argonne, Oak Ridge and NERSC that bring DOE's computing power into the petaflop range. These machines all support the message-passing model specified by the MPI standard but also have multiple cores per memory system, allowing more complex, multithreaded programming models.
- INCITE awards of substantial blocks of time on those machines have enabled new science for those who can surmount the new scalability challenges with new applications or modifications to existing ones.
- ERCAP awards to approximately 300 projects with 3000 users have been allocated on the NERSC machines. These are typically more modest blocks of time than INCITE. In addition to doing science with their computations, several projects will be scaling their codes in an effort to be considered for future INCITE scale projects.
- A 5832-core SiCortex machine has been acquired at Argonne to support computer science research and scalability development (as opposed to production applications). The machine is available to all Pmodels participants. With six cores per node (more than the IBM and Cray leadership-class machines) and a Linux operating system, it supports both the OpenMP and explicit pthread approaches to multithreading
- A new book on OpenMP has been published by one of the participants in this project. The OpenMP 3.0 version of the standard has recently appeared, with new features for HPC.

- HPC vendors have announced plans for chips with large numbers of cores, generating concern about how application programming methods will need to evolve in order to exploit these cores.
- The MPI Forum has been reconstituted and has begun meeting to modernize the existing MPI-2 standard and consider extensions and modifications appropriate to the latest developments in MPC hardware and software.

A crosscutting theme in all these developments is the coming use of multithreaded and hybrid (message-passing or another model plus threads) programming approaches. In the Pmodels project, many of the accomplishments of the past year are related to this development. A summary follows; see the next section for details.

- The interaction of multithreaded programs with the MPI library is being extensively explored at Argonne, and refinements are being added to the open-source MPICH implementation in order to conduct the experiments necessary for research; provide a free, thread-safe MPI implementation to the thousands of MPICH users; and furnish high-performance, thread-safe code for vendors to use in their own MPI implementations.
- The ARMCI library, which underlies the widely used Global Arrays model as well as some language efforts, has been adapted for multithreaded use at Pacific Northwest National Laboratory and Ohio State University.
- Co-Array Fortran (CAF), as a Partitioned Global Address Space (PGAS) language has the potential for efficient execution on multicore processors; however, refinements to the language are needed to make it more expressive. A variety of improvements have been made to the Rice CAF compiler to deal with multithreading and Rice is working with the J3 Fortran standards committee to refine the language.
- Unified Parallel C (UPC), another PGAS language, has a second implementation for shared memory based on processes rather than threads. The process-based implementation uses OS-supported shared memory and adds to interoperability with MPI applications that use multiple processes per shared memory node. In addition the Berkeley UPC group implemented a cooperative threading library to provide latency-hiding support and demonstrated its use in a UPC implementation of the High Performance Linpack benchmark.
- At the University of Houston, OpenMP as a shared-memory mode expressed in a well-developed and widely implemented language is being moved forward along two fronts: (1) The Pmodels PI there is on the OpenMP ARB, which defines the standard itself; and (2) the Houston OpenMP research compiler, a branch of the Open64 compiler toolkit, has been augmented to provide deeper insight into the research issues connected to the use of OpenMP for HPC.
- Titanium is another PGAS language particularly well adapted to use on multicore chips since it runs atop a hybrid shared and distributed memory runtime layer. Influenced by collaborations with the HPCS language efforts, the Titanium team at Berkeley has developed global view optimizations for Titanium, which will allow programs to be expressed more simply while still providing high

- performance. They have also developed a tool to automatically detect certain performance scaling bugs and program analysis to detect data races.
- At Ohio State University, we are extending the scalability of both MPI and Global Arrays over Infiniband clusters of multicore systems. A further contribution is the task-parallel model for Global Array applications, as well as the XGA extension to the GA library.

Details of these accomplishments and related work are presented in the next section, which is organized by institution. References as associated with each section separately.

3. Individual Contributions

In this section we cover recent accomplishments from each of the participating institutions.

3.1 Argonne National Laboratory – William Gropp, Ewing Lusk, Rajeev Thakur

Work at Argonne has focused on the interaction of MPI with threads and on self-consistent MPI performance requirements.

3.1.1 Test Suite for Evaluating Performance of MPI Implementations That Support MPI_THREAD_MULTIPLE

MPI implementations that support the highest level of thread safety for user programs, MPI_THREAD_MULTIPLE, are becoming increasingly common. As a result, users are able to write multithreaded MPI programs that make MPI calls concurrently from multiple threads. Thread safety does not come for free, however, because the implementation must protect certain data structures or parts of the code with mutexes or critical sections. Developing a thread-safe MPI implementation is a fairly complex task, and the implementers must make several design choices, both for correctness and for performance [Gropp06]. To simplify the task, implementations often focus on correctness first and performance later (if at all). As a result, even though an MPI implementation may support multithreading, its performance may be far from optimized. Users, therefore, need a way to determine how efficiently an implementation can support multiple threads. Similarly, as implementers experiment with a potential performance optimization, they need a way to measure the outcome. (We ourselves face this situation in MPICH2.) To meet these needs, we have developed a number of performance tests that are motivated by typical application scenarios. These tests cover the overhead of providing the MPI_THREAD_MULTIPLE level of thread safety for user programs, the amount of concurrency in different threads making MPI calls, the ability to overlap communication with computation, and other features. We obtained performance results with this test suite on several platforms (Linux cluster, Sun and IBM SMPs) and MPI implementations (MPICH2, Open MPI, IBM, and Sun). This work was published at Euro PVM/MPI 2007 and was selected as an outstanding paper [Thakur07].

The Test Suite. Users of threads in MPI often have the following expectations of the performance of threads, both those making MPI calls and those performing computation concurrently with threads that are making MPI calls.

- The cost of thread safety, compared with lower levels of thread support, such as `MPI_THREAD_FUNNELED`, is relatively low.
- Multiple threads making MPI calls, such as `MPI_Send` or `MPI_Bcast`, can make progress simultaneously.
- A blocking MPI routine in one thread does not consume excessive CPU resources while waiting.

Our tests are designed to test these expectations; in terms of the above categories, they are as follows:

Cost of thread safety. One simple test to measure `MPI_THREAD_MULTIPLE` overhead.

- **Concurrent progress** Tests to measure concurrent bandwidth by multiple threads of a process to multiple threads of another process, as compared with multiple processes to multiple processes. Both point-to-point and collective operations are included.
- **Computation overlap** Tests to measure the overlap of communication with computation and the ability of the application to use a thread to provide a nonblocking version of a communication operation for which there is no corresponding MPI call, such as nonblocking collectives or I/O operations that involve several steps.

The entire test suite can be downloaded from <http://www.mcs.anl.gov/~thakur/thread-tests>.

We ran the tests on a GigE connected Linux cluster using MPICH2 and Open MPI and on Sun and IBM SMPs using Sun and IBM MPI implementations. The performance results indicate that the difference between the multithreaded and multiprocess performance is relatively small for both MPICH2 and Open MPI on the Linux cluster, with MPICH2 performing better than Open MPI (less overhead). However, the multithreaded performance of IBM and Sun MPIs on a single SMP box is quite poor (see Figures 1 and 2). Detailed performance results can be found in [Thakur2007].

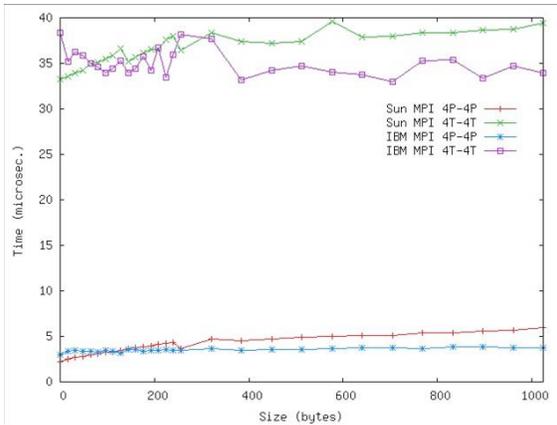


Figure 1. Concurrent Latency Test on Sun and IBM SMPs. (Multiple threads communicating with multiple threads, or multiple processes communicating with multiple processes.)

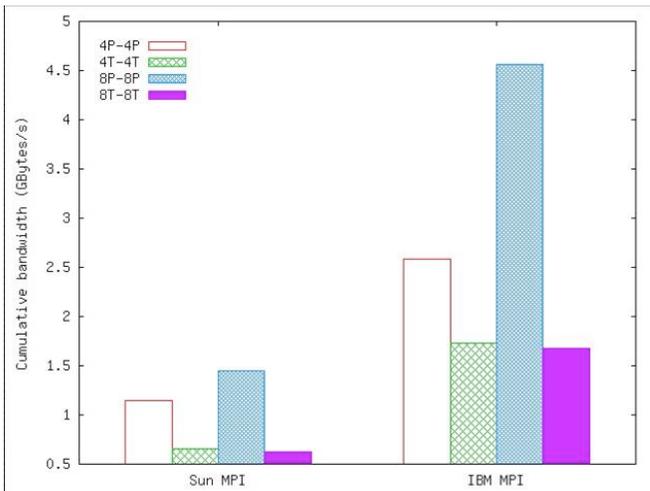


Figure 2. Concurrent Bandwidth Test on Sun and IBM SMPs.

3.1.2 Self-Consistent MPI Performance Requirements

For good reasons, MPI comes without a performance model and, apart from some “advice to implementers”, without any requirements or recommendations as to what a good implementation should satisfy regarding performance. The main reasons are that the implementability of the MPI standard should not be restricted to systems with specific interconnect capabilities and that implementers should be given maximum freedom in how to realize the various MPI constructs. The widespread use of MPI over an extremely wide range of systems, as well as the many existing and quite different implementations of the standard, show that this was a wise decision.

However, users often complain about the poor performance of some of the MPI functions in MPI implementations and of the difficulty of writing code whose performance is portable. Solving this problem requires defining performance standards that MPI

implementations are encouraged to follow. In collaboration with Jesper Larsson Traff from NEC Research Labs in Germany, we have defined some basic, intrinsic performance rules for MPI implementations. This work was published at Euro PVM/MPI 2007 and was selected as an outstanding paper [Traff07].

The general principle behind the performance rules is that the *library internal* implementation of any arbitrary MPI function in a given MPI library should not perform any worse than an *external (user)* implementation of the same functionality in terms of (a set of) other MPI functions. For example:

- Subdividing messages into multiple messages should not reduce the communication time.
- Replacing an MPI function with a similar function that provides additional semantic guarantees should not reduce the communication time.
- Replacing a specific (collective) MPI operation with a more general operation by which the same functionality can be expressed should not reduce communication time.
- Replacing a (collective) operation by a sequence of other operations implementing the same functionality should not reduce communication time.
- A virtual process topology should not make communication between all pairs of neighboring processes slower than communication between the same processes in any other communicator.

In [Traff07, Traff08], we have defined these rules more formally. These rules can be automatically verified by benchmarks and performance evaluation tools, thereby giving both users and implementers insight into the behavior of an MPI implementation and indicating areas needing improvement. We also provided examples where some of these rules are being violated. For example, in Figure 3, a user with a 1500-byte message will achieve better performance on this system by sending two 750-byte messages. This example shows one of the implementation features that competes with performance portability---in this case, the use of limited message buffers.

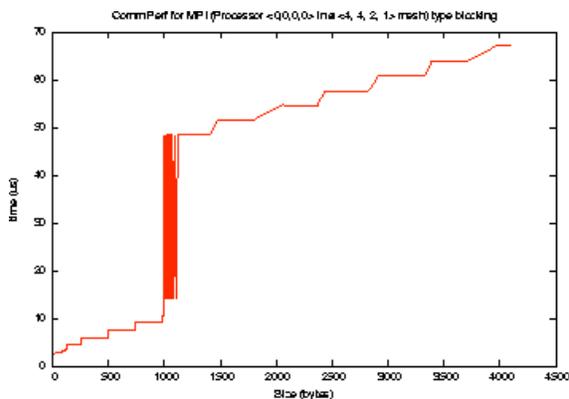


Figure 3. Measured performance of short messages on IBM BG/L. Note the large jump around 1024 bytes; this is the transition from eager to rendezvous protocol in the MPI implementation.

3.1.3 Scalable Tools Communication Infrastructure

We have been working on the Scalable Tools Communication Infrastructure (STCI) initiative. The purpose of the STCI initiative is to develop a common, scalable, high-performance communication infrastructure to support development tools on emerging peta- and exa-scale high performance computing systems. Members of the initiative include tool developers, middleware developers and system software developers from industry, academia and national labs. Since our initial meeting in July, we have identified expected use cases and requirements, and have designed the core architecture. We are currently defining APIs for the various components and beginning implementation.

References

- [Gropp06] William Gropp and Rajeev Thakur, "Thread Safety in an MPI Implementation: Requirements and Analysis," *Parallel Computing*, (33)9:595-604, September 2007.
- [Thakur07] Rajeev Thakur and William Gropp, "Test Suite for Evaluating Performance of MPI Implementations That Support MPI_THREAD_MULTIPLE," in *Proc. Of the 14th European PVM/MPI Users' Group Meeting (Euro PVM/MPI 2007)*, September 2007. (selected as outstanding paper)
- [Traff07] Jesper Larsson Traff, William Gropp, and Rajeev Thakur, "Self-Consistent MPI Performance Requirements," in *Proc. Of the 14th European PVM/MPI Users' Group Meeting (Euro PVM/MPI 2007)*, September 2007. (selected as outstanding paper)
- [Traff08] Jesper Larsson Traff, William Gropp, and Rajeev Thakur, "Self-Consistent MPI Performance Requirements," in preparation for submission to *ACM Transactions on Programming Languages and Systems*, 2008.
- G. Bosilca, D. Buntinas, R. Graham, G. Vallee, G. Watson, "Scalable Tools Communication Infrastructure." Submitted to the 6th annual Symposium on OSCAR and HPC Cluster Systems (OSCAR '08), Feb. 2008.
- "Languages for High-Productivity Computing: The DARPA HPCS Language Project", by E. Lusk and K. Yelick, *Parallel Processing Letters* vol. 17, No. 1 (March 2007) pp. 89-102.

3.2 University of California, Berkeley – Katherine Yelick

3.2.1 Support for Dynamic Multithreading in PGAS Languages

Much of high-end scientific computation is organized into a bulk-synchronous model having distinct phases of communication and computation. This has advantages in code

simplicity, but it not a good fit to some algorithms, and even something a regular at dense matrix factorization can be unnecessarily constrained by the bulk-synchronout model. The Berkeley team developed a data-driven implementation of LU factorization built on the UPC language, which has one-sided communication via its global address space, locality control through the partitioning of the address space, and a static parallelism model with barrier synchronization which lends itself well to a bulk-synchronous style. They explored extensions of the basic UPC execution model to better support problems such as matrix factorization with interesting dependence patterns, and evaluated them on LU factorization.

The long term goal of our project is to develop highly optimized matrix factorization routines for both dense and sparse matrices. In addition, the UPC community is exploring possible extensions to UPC to improve productivity and performance.

Two of the most common parallel LU factorization codes for distributed memory machines are from the ScaLAPACK library and the High Performance Linpack (HPL) benchmark used in determining the Top 500 list. Both of these codes are written for portability and scalability using the two-sided message passing model in MPI, and are written to keep the processors somewhat synchronized in order to manage the matching of sends and receives and the associated buffer space for messages. The UPC code is designed with latency hiding as primary goal, and we explore the programmability and performance benefits of UPC's one-sided communication model, coupled with a dynamic parallelism model. The experience with this algorithm highlights some of the subtle pitfalls of dynamic threading and the need for application-level control for thread management, which is relevant to the HPCS languages (X10, Forress, and Chapel) as well as existing libraries like Charm++ or (if augmented with locality control) Cilk. Scheduling decisions must be made to balance the needs of parallel progress, memory utilization, and cache performance.

Several challenges arise in using a highly parallel dataflow view of the algorithm as we do. First, because we want to run on hundreds or even thousands of processors and across clusters, locality is critical. We use UPC's global address space to statically distribute blocks of the input matrix and build scheduling queues for the tasks associated with each block; both the matrix blocks and queues are remotely accessed through the global address space. Second, the multithreading support that is needed to expose available parallelism can have a significant runtime cost; we explore several different strategies for implementing fast user-level threads. Third, while the algorithm is highly dynamic, control over task scheduling is critical and non-obvious. For highest performance we use an application-specific scheduling policy to ensure proper prioritization. Fourth, as with any attempt to expose all available parallelism to the runtime layer, memory resources can easily be strained, and deadlock may result in a constrained memory environment, because tasks that have been allocated may not be able to run until other unallocated tasks complete. We use a novel dependence-constrained task allocation mechanism to avoid deadlock. Finally, we incorporate some of the best-practice optimizations from prior work, including recursive algorithms to increase granularity and the combining certain tasks to improve local task size and thereby boost serial performance. We found each of these optimizations necessary to high performance.

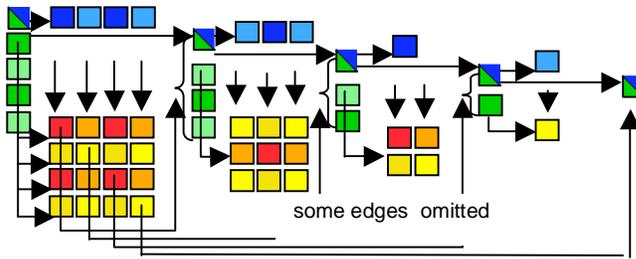


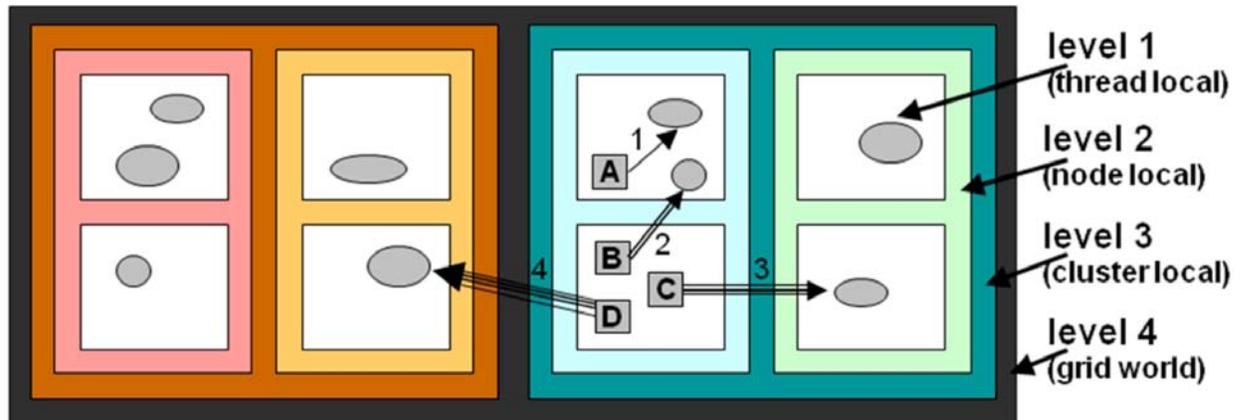
Figure 1. Dependencies in LU factorization:

3.2.2 Automated Performance Analysis for PGAS Languages

One of the most attractive features of PGAS languages is the ability to access remote memory implicitly through shared memory reads and writes. But this benefit does not come without a cost. It is very difficult to spot communication by looking at the program text, since remote reads and writes the same as local reads and writes at the execution point—one needs to examine the types to see potential communication. This makes manual communication performance debugging difficult. The Titanium group developed tool called ti-trend-prof that can do automatic performance debugging using only program traces from small processor configurations and small input sizes in Titanium. ti-trend-prof presents trends to the programmer to help spot possible communication performance bugs even for processor configurations and input sizes that have not been run, in particular scalability problems when communication is not scaling as expected. We used titrend-prof on two of the largest Titanium applications, adaptive mesh refinement and heart simulation, and found bugs that would have taken days to discover in under an hour. In one case the performance bug (a missing type annotation of “local”) was inadvertently introduced in other routine software maintenance and did indeed take days to discover prior to the development of this tool.

3.2.3 Hierarchical Pointer Analysis for Titanium

Parallel machines are often built with hierarchical memory systems, with local caches or explicitly managed local stores associated with each process. For example, partitioned global address space (PGAS) languages may run on shared memory, distributed memory machines or hybrids, with the language runtime providing the illusion of shared memory through the use of wide pointers (that store both a processor node number and an address), distributed arrays, and implicit communication to access such data. Hierarchies also exist within processors in the form of caches and local stores. For example, the Cell game processor has a local store associated with each of the SPE processors, which can be accessed by other SPEs through memory move (DMA) operations. Additional levels of partitioning are also possible, such as partitioning memory in a computational grid into clusters, each of which is partitioned into nodes, as show below.



In Titanium, threads are arranged in the following three-level hierarchy:

- *Level 1:* an individual thread
- *Level 2:* threads within the same physical address space, such as a UNIX process
- *Level 3:* all threads

The *distance* between two threads is the lowest level of the hierarchy at which both threads are located. Thus, the distance between a thread and itself is 1, between two different threads in the same physical address space is 2, and between threads in different physical address spaces is 3.

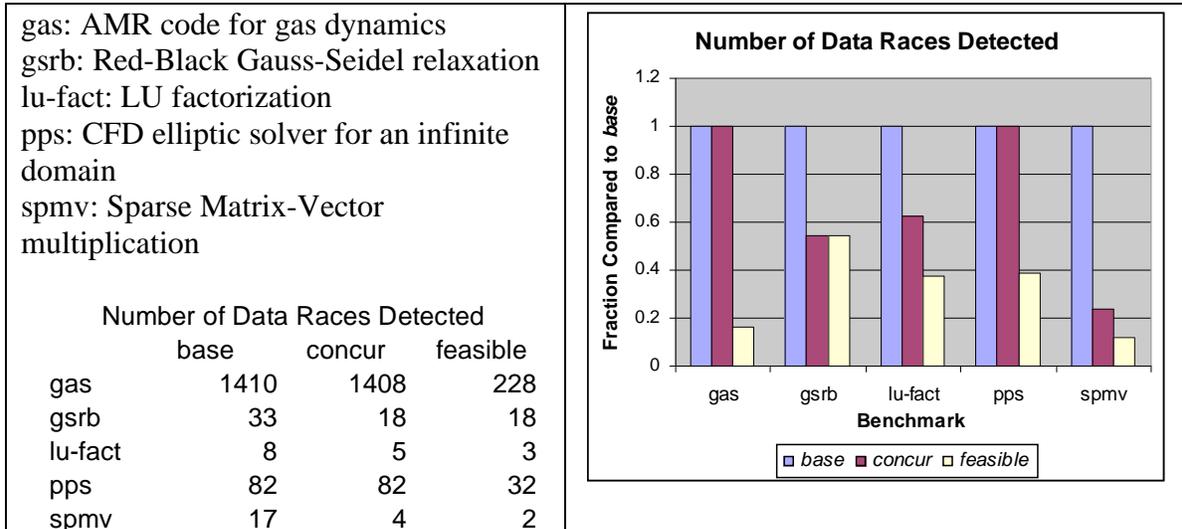
We produced a pointer analysis for a simple parallel language with a generic machine hierarchy, called *Ti*. The analysis determines not only which memory locations can be referenced by each variable and memory location, but also on which threads the memory can be located. Dynamic memory locations are represented using *abstract locations*, which consist of an allocation site and a level in the machine hierarchy. The abstract location (l, n) can only refer to memory allocated at the allocation site labeled l and on a thread whose distance is at most n .

For every variable and abstract location in a program, the analysis computes which abstract locations they can refer to. The analysis has special inference rules for each construct in *Ti*: variables, memory allocations, dereferences, type conversions, broadcasts, variable assignments, and dereferencing assignments. These rules are based on the operational semantics for each construct, which we also defined. We proved that the inference rules are sound and provided an algorithm that uses these rules to perform the pointer analysis.

We implemented three versions of the hierarchical pointer analysis in Titanium. They are a single-level analysis that does not distinguish between the three levels in the memory hierarchy, a two-level analysis that combines levels 2 and 3, and a three-level analysis that separates all the levels in the Titanium memory hierarchy. The three-level analysis only takes 10% longer than the single-level analysis, despite the fact that it is much more accurate.

One of the main challenges in shared memory parallel programming is avoiding data races, which occur when two threads access the same variable simultaneously and at least

one of them performs a write operation. Data races usually lead to nondeterministic program behavior and are often programming errors. A great deal of effort has gone into the automatic discovery of data races using either static analysis or runtime tools. The static analysis techniques are necessarily conservative. Below are the data for our concurrency analysis, combined with a type-based alias analysis on several scientific Titanium benchmarks.



The data shows that importance of our most sophisticated concurrency analysis based on feasible execution paths. This analysis is probably still not accurate enough to use in a user tool as the reported races are not real program bugs, and so would be viewed as false positives. In future work we plan to incorporate an array alias analysis to further reduce conflict edges.

References

- [1] Alfredo Buttari, Jack Dongarra, Parry Husbands, Jakub Kurzak and Katherine Yelick, "Multithreading for synchronization tolerance in matrix factorization," The proceedings of the SciDAC 2007 Conference, Boston, Massachusetts, July 24-28, 2007. Published in the Journal of Physics: Conference Series. Volume 78, 2007, June, 2007.
- [2] Jimmy Su and Katherine Yelick, "Automatic Performance Debugging in Partitioned Global Address Space Programs" 20th International Workshop on Languages and Compilers for Parallel Computing (LCPC), Urbana, Illinois, October 2007. To appear in Springer Lecture Notes in Computer Science.
- [3] Parry Husbands and Katherine Yelick, "Multithreading and One-Sided Communication in Parallel LU Factorization." Proceedings of Supercomputing (SC07), Reno, NV, November, 2007.
- [4] Tong Wen, Jimmy Su, Phillip Colella, Katherine Yelick and Noel Keen, "An Adaptive Mesh Refinement Benchmark for Modern Parallel Programming Languages." Proceedings of Supercomputing (SC07), Reno, NV, November 2007.

- [5] Amir Kamil and Katherine Yelick, "Hierarchical Pointer Analysis for Distributed Programs," Static Analysis Symposium (SAS), Kongens Lyngby, Denmark, August 22-24, 2007.
- [6] Katherine Yelick, Paul Hilfinger, Susan Graham, Dan Bonachea, Jimmy Su, Amir Kamil, Kaushik Datta, Phillip Colella, and Tong Wen, "Parallel Languages and Compilers: Perspective from the Titanium Experience." *Journal of High Performance Computing Applications*, August 2007, vol. 21, pp. 266-290.
- [7] K. Yelick, D. Bonachea, W.-Y. Chen, P. Colella, K. Datta, J. Duell, S. Graham, P. Hargrove, P. Hilfinger, P. Husbands, C. Iancu, A. Kamil, R. Nishtala, J. Su, M. Welcome, T. Wen, "Productivity and Performance Using Partitioned Global Address Space Languages," *Proceedings of Parallel Symbolic Computation (PASCO)*, London, Ontario, July 27-28, 2007.
- [8] Ewing Lusk and Katherine Yelick, "Languages for High-Productivity Computing: The DARPA HPCS Language Project," *Parallel Processing Letters*, Vol. 17, No. 1 (2007) 89-102.
- [9] Wei Chen, Dan Bonachea, Costin Iancu, and Katherine Yelick, "Automatic Nonblocking Communication for Partitioned Global Address Space Programs," *Proceedings of the International Conference on Supercomputing (ICS)*, Seattle, Washington, June 16-17, 2007.
- [10] Shivali Agarwal, Rajkishore Barik, Dan Bonachea, Vivek Sarkar, Rudrapatna Shyamasundar, Katherine Yelick, "Deadlock-Free Scheduling of X10 Computations with Bounded Resources," *Symposium on Parallel Algorithms and Architecture (SPAA)*, San Diego California, June 9-11, 2007.

3.3 University of Houston – Barbara Chapman

3.3.1 Overview of OpenMP Research at UH

In this project the UH team directed by Dr. Barbara Chapman is working with partners and hardware vendors to increase the scalability and applicability of OpenMP for multi-core platforms and distributed memory systems by focusing on language extensions, compiler optimizations, as well as runtime library support. We are also collaborating with our partners to explore OpenMP's interoperability with other programming models and ensure that it can be deployed with each of the message passing libraries and PGAS models. We are working on providing support for users to determine suitable approaches for exploiting hybrid models for their applications. Our work is implemented within the robust OpenUH compiler infrastructure which serves as an indispensable test bed for ensuring the success of our research.

In this year, we worked in introducing new OpenMP language features to address the scalability and to facilitate the development of hybrid MPI and OpenMP programs. We enhanced the OpenUH compiler infrastructure by improving its portability, and we are

currently working on implementing OpenMP 3.0 features. We also continued our initial PModels work on improving the OpenMP translation strategy for execution on clusters, particularly for exploring how to handle sequential parts translation efficiently. We explored the existing cost model within our compiler and implemented a novel OpenMP cost model by considering OpenMP and multi-core architecture characteristics. We also refined our performance modeling for MPI and OpenMP programs taking both static and dynamic information into account.

We took part in the OpenMP 3.0 specification weekly discussion meetings and the face-to-face meeting in IWOMP07 Beijing. We have contributed our results to the OpenMP Architecture Review Board (ARB) and have collaborated in language committees of the ARB to discuss future OpenMP features for multi-core programming. We have successfully organized the HPCC07 conference at Houston and presented our work in a variety of conferences and workshops including an invited talk in Parco 07 for “*OpenMP in the Multi-Core Era*”. The SiCortex workshop held at Argonne recently gave us a good opportunity to explore the latest multi-core architectures and collaborate with our partners to explore hybrid programming models. Also the book “*Using OpenMP*” [3], authored by Chapman *et al.*, was published by the MIT press this year and described how best to write parallel programs with OpenMP based on our experiences.

3.3.2 OpenMP for Multi-Core Architectures

OpenMP continues to gain more acceptance as a widely used parallel programming model in the shared memory multi-core and many-core systems. Large-scale computing platforms are built on top of multi-core processors. As a result, the new levels of architectural parallelism and new types of resource contention introduced require further exploration into the existing programming models, compiler, and runtime tools. We need to adapt OpenMP to support the creation of efficient programs, including MPI+OpenMP hybrid program development, that utilize the power of these systems. For example, the locality of operations that execute in concurrent threads becomes critical in multi-core systems, because threads must compete for shared resources and memory per thread is limited. While OpenMP provides features for assigning work to user-level threads, there currently is no way to specify the subgrouping of these threads, the mapping of threads to the hardware, or data placement. Language features are therefore needed to enable a more flexible assignment of work to threads and permit a careful mapping of threads to the target platform. To enable better control of locality, and to facilitate the creation of efficient hybrid MPI+OpenMP programs, we have defined and implemented the concept of subteams [4] in OpenMP. The subteam extension is inspired by MPI's group and topology concepts. These ideas can be used to parallelize multi-dimensional loop nests for large thread counts, to describe a variety of execution scenarios including pipelining, as well as to flexibly assign work across a system with non-uniform resource sharing. We have successfully implemented and tested the subteam concept [6] in the OpenUH compiler.

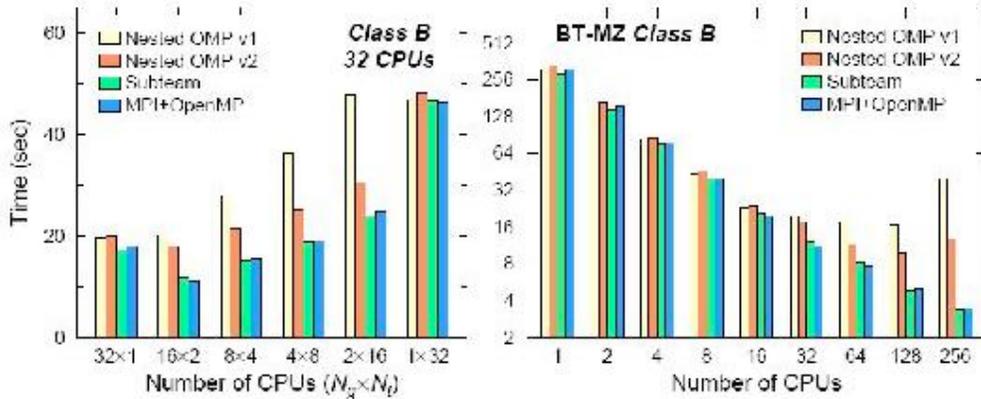


Figure 1: Comparison of subteams with equivalent versions of BT multizone benchmark.

Our experiments show that the subteam concept is easy to use and can greatly enhance the scalability of code. Together with colleagues at NASA Ames Research Center, we have evaluated the performance of four versions of the NAS BT Multi-zone benchmarks using OpenMP nested parallelism (2 versions), OpenMP with the subteam implementation, and hybrid MPI+OpenMP [8] on an SGI Altix system with 512 Itanium 2 processors. Figure 1 presents the results. Our experiments demonstrated that the subteam and hybrid versions are close in performance since both enable a similar data layout and reuse the data efficiently. Moreover, the code for the subteam version turned out to be much simpler than the other versions.

3.3.4 Enhancing the OpenUH Compiler

We have further enhanced the portability and robustness of the OpenUH compiler framework [12, 11] by providing more backends to support a variety of architectures. OpenUH is a robust OpenMP 2.5 compiler based upon the open-source Open64 compiler framework, and it provides state-of-the-art optimizations for Fortran and C/C++. OpenUH is currently used in a number of research projects around the world. We are currently implementing the new features in OpenMP 3.0 and plan to release it to the community in the next year to provide researchers a solid test bed to experiment with the OpenMP 3.0 features. We are also working on the alternative OpenMP compiler translation approach to transform an OpenMP code to a collection of sequential tasks and a task graph that indicates their execution constraints [2]. This approach forms the basis of OpenMP 3.0 implementation, and can potentially reduce synchronization costs and improve data locality.

3.3.5 Cost Model for OpenMP on Multi-cores

We have extended the cost model in OpenUH to effectively estimate the execution cost of OpenMP programs during compile time on multi-core architectures. An OpenMP cost model is an analytical model that estimates the cost of executing (regions of) OpenMP programs, typically in terms of clock cycles. Such a model could guide compiler

transformations, enhance adaptive runtime support, and assist performance analysis. However, existing cost models for OpenMP make over-simplifying assumptions and ignore many software and hardware details. We have designed and implemented a novel compile-time cost model [10, 9] for OpenMP that explicitly considers thread scheduling policies, synchronization overheads, cache configuration, and much more. The resulting model is able to provide sufficiently accurate cost estimates to support the optimization process with reasonable overheads. We are exploring extensions to the model to take into account multi-core resource contention, architecture topologies, and energy consumption.

References

- [1] Laksono Adhianto. *A new Framework for Analyzing, Modeling and Optimizing MPI and/or OpenMP Applications*. PhD thesis, Department of Computer Science, University of Houston, 2007.
- [2] Barbara Chapman and Lei Huang. Enhancing openmp and its implementation for programming multicore systems. In *Parallel Computing 2007 (ParCo 2007)*, September 2007.
- [3] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP*. The MIT Press, Cambridge, Massachusetts and London, England, 2007.
- [4] Barbara M. Chapman, Lei Huang, Haoqiang Jin, Gabriele Jost, and Bronis R. de Supinski. Extending OpenMP worksharing directives for multi-threading. In *Euromicro 2006*, pages 645-654, 2006.
- [5] Deepak Eachempati, Lei Huang, and Barbara M. Chapman. Strategies and implementation for translating openmp code for clusters. In *HPCC*, pages 420-431, 2007.
- [6] Lei Huang, Barbara Chapman, and Chunhua Liao. An implementation and evaluation of thread subteam for OpenMP extensions. In *Workshop on Programming Models for Ubiquitous Parallelism (PMUP 06)*, Seattle, WA, September 2006.
- [7] Lei Huang, Girija Sethuraman, and Barbara Chapman. Parallel data flow analysis for openmp programs. In *Proceedings of IWOMP 2007*, June, 2007.
- [8] Haoqiang Jin, Barbara Chapman, and Lei Huang. Performance evaluation of a multi-zone application in different openmp approaches. In *Proceedings of IWOMP 2007*, June, 2007.
- [9] Chunhua Liao. *A Compile-time OpenMP Cost Model*. Phdthesis, Department of Computer Science, University of Houston, 2007.

- [10] Chunhua Liao and Barbara Chapman. Invited paper: A compile-time cost model for openmp. In *12th International Workshop on high-Level Parallel Programming Models and Supportive Environments (HIPS)*, March 2007.
- [11] Chunhua Liao, Oscar Hernandez, Barbara Chapman, Wenguang Chen, and Weimin Zheng. Openuh: an optimizing, portable openmp compiler: Research articles. *Concurr. Comput.: Pract. Exper.*, 19(18):2317-2332, 2007.
- [12] The OpenUH compiler project. <http://www.cs.uh.edu/~openuh>, 2005.
- [13] Girija Sethuraman. Parallel data flow analysis for openmp programs. Master's thesis, Department of computer Science, University of Houston, Spring, 2007.

3.4 Ohio State University -- D.K. Panda, P. Saddayappan

Here we describe the on-going research accomplished by the OSU team under the Pmodels2 project. The team has worked on various angles: designing high performance MPI implementations on modern networking technologies [Mellanox InfiniBand (including the new ConnectX architecture)], QLogic InfiniPath, IBM 12X InfiniBand and the emerging iWARP/10GigE), studying MPI scalability issues for multi-thousand node clusters, designing scalable collective communication libraries for emerging multi-core architectures, and designing MPI-level solutions to avoid congestion avoidance in multi-core clusters.

Working with PNNL, we are implementing enhancements to the GA/DRA suite, aimed at enhancing programmer productivity. The main foci are: 1) implementation and evaluation of a "work-sharing" taskpool model, and, 2) transparent interface for in-memory and on-disk arrays (XGA).

During the first year of the project, progress has been made along the following directions:

3.4.1 MPI Implementations on Modern Networking Technologies

InfiniBand is emerging as an open-standard interconnect for designing next generation high performance clusters. For the last several years, OSU has been engaged in designing high performance MPI (MVAPICH with MPI-1 semantics and MVAPICH2 with MPI-2 semantics) [9] for InfiniBand-based clusters. As the InfiniBand networking standard matures, the next-generation of hardware are being released. Two of the prominent new network-interface adapters are the ConnectX architecture by Mellanox technologies and 12X adapter from IBM. The ConnectX architecture is geared towards improving communication performance and scalability on multi-core platforms. Our analysis in [10] reveals that latency on multi-core platforms can be improved by an order of magnitude, even when all cores are communicating simultaneously. IBM's 12X adapter is geared towards improving aggregate bandwidth and providing fail-over mechanisms by utilizing multiple send/receive engines. In [11], we have introduced new

designs in the MPI library which can exploit the multiple send/receive engines in the IBM 12X adapter and provide optimal performance to end MPI applications. We have also designed and developed an optimized implementation of MPI for QLogic's InfiniPath adapter. Recently, 10GigE networks are becoming popular for cluster computing through the iWARP standard. In [8], we conduct a detailed performance evaluation of MPI on top of state-of-the art iWARP networking stack.

All these designs and solutions have been integrated into the open-source MVAPICH and MVAPICH2 software stacks and have been distributed to the community. These software packages are currently being used by more than 580 organizations worldwide. These packages are also available with the Open Fabrics Enterprise Distribution (OFED) stack.

3.4.2 Study of MPI Scalability Issues

The ever increasing demand for more computational power by scientific applications has lead to the increase in scale of compute clusters. Clusters with thousands of processing cores are already deployed and even larger clusters with tens-of-thousands of cores are in the planning stages. The MPI library utilized by the scientific applications should accordingly scale both in terms of performance delivered and resources consumed. The OSU team has been pursuing this direction of research. We have explored the viability of using the InfiniBand Unreliable Datagram as a scalable transport for MPI. Our designs and experimental results presented in [2] show that the performance and resource consumption of the MPI library could be significantly improved at large scale (with 8K-16K processors). In addition, our research in using a message-coalescing approach geared towards reduction of memory consumption in reliable connection oriented models [4] reveals that significant savings in resource consumption can be obtained while not sacrificing end application performance. New designs including UD-based support [3] and hybrid UD-RC-based support [1] are being investigated to scale MVAPICH library to multi-thousand node clusters.

3.4.3 Scalable collective communication over large scale multi-core InfiniBand clusters

As large scale InfiniBand multi-core clusters are being increasingly deployed, several challenges emerge pertaining to scalability and performance of collective operations. Collective Communications exhibit varying communication patterns and behaviors and accordingly, intelligent design decisions are required which guarantee high performance and scalable resource usage. We have studied the implications of using different transports of InfiniBand such as Reliable Connection (RC) and Unreliable Datagram (UD). The focus of this study has been to demonstrate the semantic advantages offered by these transport protocols and their performance to memory trade-offs. The study demonstrates the significance of using Connection-Less transport over Connection-Oriented transport for better performance and improved resource utilization. Further, the utility of RDMA semantics for collective operations is also studied [7]. The team has on-going research effort for improving performance of collective operations over multi-core

clusters. Especially for certain collective operations, techniques such as message aggregation are proposed to cut-down the number of network operations and improve network utilization [6].

3.4.4 Congestion avoidance with InfiniBand For clusters, fat tree has become the most popular interconnection topology, due to its multi-pathing capabilities. However, even with fat tree, hot-spots may occur in the network depending upon the route configuration between end nodes and communication patterns in the application. To make matters worse, the deterministic routing nature of InfiniBand limits the application from effective use of multiple paths transparently and avoid the hot-spots in the network. To alleviate this situation, we have designed an MPI functionality which provides hot-spot avoidance for different communication patterns, without a prior knowledge of the pattern[12]. We have leveraged LMC (LID Mask Count) mechanism of InfiniBand to create multiple paths in the network, and studied its efficiency in creation of contention free routes. Our evaluation with NAS Parallel Benchmarks and collective communication primitives shows significant improvement compared to the current state-of-the-art designs.

3.4.5 Prototype development of taskpool model for independent tasks The basic execution model of Global Arrays (GA) in its current distribution is MPI-like, i.e. there are P GA processes (typically equal to the number of physical processors for execution) that are "long-lived" and exist through the parallel program's execution. Like an MPI process, each GA process also has persistent "local" state in its copy of all local variables. While it improves upon MPI in providing a convenient global view of large arrays, the above GA model does not provide any better support for load balancing than MPI - the programmer must do it explicitly. In order to provide system-supported load balancing, the taskpools extension to the GA model is being developed.

A taskpool is a set of tasks, where each task's inputs and outputs are specified as portions of global arrays. A task in the taskpool can perform arbitrary local computation, but using the GetfromGlobal+ComputeLocal+PuttoGlobal model. Thus, its input operands are all portions of global arrays, and its outputs specify portions of global arrays, but within the body of a task's code, no references to global arrays are permitted. In the first prototype of taskpools, all tasks in the pool are independent. The taskpool model was created primarily to provide system support for load balancing. Since the tasks explicitly specify the portions of global arrays that they need to copy in or update, the system can perform locality-aware load balancing of tasks among the processors. Affinity of processors to memory - with multi-core and SMP nodes - can be exploited in this load balancing process, without needing to impose a two-level (e.g. MPI+OpenMP) programming model.

A prototype implementation of the taskpool model for independent tasks has been implemented over the GA/ ARMCI interface. A simple global lock-based dynamic load balancing scheme has been implemented. Development of a hierarchical load-balancing scheme for SMP and multi-core systems is underway.

3.4.6 Development of prototype for XGA (eXtended Global Arrays) for transparent access to global arrays in memory and out-of-core: Intelligent and automated management of data movement is a fundamental and unifying theme for the Extended Global Array interface we are developing. The goal is to have a single interface for managing data and high level representation of the mathematical algorithms operating on multidimensional arrays while the details on the underlying data movement between secondary storage, distributed memory, shared memory, and local memory are handled by the XGA implementation. A key runtime functionality needed for implementation of XGA is asynchronous one-sided access to portions of a disk-resident array.

The pre-existing DRA API only allows collective movement of data between disks and memory. This has been enhanced by an asynchronous implementation for efficient non-collective I/O (GPC-IO) as part of this framework. As a generalization of the Remote Procedure Call (RPC) that was used as a foundation for the Sun NFS system, we developed a global procedure call (GPC) to invoke procedures on a remote node to handle non-collective I/O. We considered alternative approaches that could be employed in implementing this functionality. The approaches are evaluated using a representative computation from quantum chemistry. The results [5] demonstrate that GPC-IO achieves better absolute execution times, strong-scaling, and weak-scaling than the alternatives considered.

References

- [1] M. Koop, T. Jones, and D.K. Panda. MVAPICH-Aptus: Scalable High-Performance Multi-Transport MPI over InfiniBand. In *Int'l Parallel and Distributed Processing Symposium (IPDPS)*, 2008.
- [2] M. Koop, S. Sur, Q. Gao, and D.K. Panda. High Performance MPI Design using Unreliable Datagram for Ultra-Scale InfiniBand Clusters. In *International Conference on Supercomputing (ICS07)*, 2007.
- [3] M. Koop, S. Sur, and D.K. Panda. Zero-Copy Protocol for MPI using InfiniBand Unreliable Datagram. In *IEEE International Conference on Cluster Computing (Cluster'07)*, 2007.
- [4] Matthew J. Koop, Terry Jones, and Dhabaleswar K. Panda. Reducing Connection Memory Requirements of MPI for InfiniBand Clusters: A message Coalescing Approach. In *International Symposium on Cluster Computing and the Grid*, pages 495-504, 2007.
- [5] Sriram Krishnamoorthy, Juan Piernas Canovas, Vinod Tipparaju and Jarek Nieplocha, and P. Sadayappan. Non-collective parallel i/o for global address space programming models. In *Proceedings of Cluster 2007*, Sept. 2007.
- [6] Amith R. mamidala, Debraj De. Abhinav Vishnu, Sundeep Narravula, and Dhabaleswar K. Panda. Scalabel Collective Communication for Next-Generation

Multicore Clusters with InfiniBand. In *technical Report No. OSU-CISRC-6/07-TR49*, 2007.

[7] Amith R. Mamidala, Sundeep Narravula, Abhinav Vishnu, Gopalakrishnan Santhanaraman, and Dhableswar K. Panda. On Using Connection-Oriented vs. Connection-Less Transport for Performance and Scalability of Collective and One-Sided Operations: Trade-offs and Impact. In *Symposium on Principles and Practices of Parallel programming*, pages 46-54, 2007.

[8] Sundeep Narravula, Amith Mamidala, Abhinav Vishnu, Gopal Santhanaraman, and Dhableswar K. Panda. High Performance MPI over iWARP: Early Experiences. In *International Conference on Parallel Processing*, 2007.

[9] Network-Based Computing Laboratory. MVAPICH/MVAPICH2: MPI-1/MPI-2 for InfiniBand and iWARP. <http://mvapich.cse.ohio-state.edu>.

[10] S. Sur. M. J. Koop, L. Chai, and D.K. Panda. Performance Analysis and Evaluation of Mellanox ConnectX InfiniBand Architecture with Multi-Core Platforms. In *International Workshop on High-Level Parallel programming Models and Supportive Environments, IPDPS*, 2007.

[12] Abhinav Vishnu, Matthew J. Koop, Adam Moody, Amith R. Mamidala, Sundeep Narravula, and Dhableswar K. Panda. Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective. In *International Symposium on Cluster Computing and the Grid*. Pages 479-486, 2007.

3.5 Pacific Northwest National Laboratory

3.5.1 Ports to New Platforms. In preparation for the emerging petascale systems ARMCI runtime and Global Arrays toolkit have been ported to run on several new platforms. This includes the IBM Blue Gene/L and the Cray XT3/XT4. We have been collaborating with IBM and Cray to develop and optimize these ports. In addition, we developed a port of ARMCI to the OpenIB layer that replaces vendor-specific flavors of the Infiniband verbs. In an [SC'06] paper, IBM has described its port of ARMCI and Global Arrays for the IBM Blue Gene/L system. In addition, the IBM BG/L port, the version 4.0.8 of Global Arrays includes a port to the Cray XT3/XT4 systems. These ports enable DoE codes such as the NWChem computational chemistry package and ScalaBLAST bioinformatics application and several other scientific codes that use ARMCI and the Global Arrays toolkit to run on these systems at a large scale. As described in the recent paper in *Journal of Chemical Theory and Computation* (vol. 3 no. 2, 2007), performance of Molecular Dynamics Simulations to Sample Free-Energy States has shown significant improvement with Global Arrays toolkit and potential of this method to scale to several thousands of processors.

3.5.2 Global Procedure Calls (GPC). We have been developing a mechanism enabling users to ship computations to be executed remotely on global address space data. This

capability extends ARMCI data movement operations that support global address programming models. GPC can be thought of as an extension of the traditional Remote Procedure Calls (RPC) to parallel processing environments with global address space. Unlike most implementations of RPC and Active Messages, GPC provides truly unilateral progress model without relying on implicit or explicit polling. Its implementation relies on the shared memory and threads to achieve high performance. Memory bandwidth is the most constrained system resource on the current multicore processors. To eliminate memory copies that waste memory bandwidth our implementation uses shared memory. In addition to saving bandwidth in the GPC implementation, shared memory is the fastest available communication protocol on multicore and multiprocessor SMP nodes. The globally addressable memory allocated by ARMCI from the operating system is shared memory. It allows execution of GPC within multicore or SMP multiprocessor nodes without any intermediate memory copies. We have prototyped a nonblocking execution model of GPC callbacks to support I/O and increase responsiveness. In addition, we have also been considering requirements of emerging DARPA languages like the IBM X10 and Cray Chapel in development of GPC.

3.5.3 Thread Safe ARMCI The growing importance of multicore systems led to renewed interests in multithreaded rather than multiprocess execution models. Multithreaded execution model has been used in the implementation of OpenMP as well as it is default for new programming models such as X10. During last year, we have redesigned the internal implementation protocols in ARMCI to make ARMCI thread safe. In particular, this required modifications to the buffer management layers used in implementation of atomic and one-sided communication operations with noncontiguous interfaces that involve intermediate packing of the data. Our objective has been to maximize communication concurrency and minimize memory consumption by allowing a thread to process and complete network communication calls even when initiated by another thread. In addition, the underlying implementation of blocking communication calls in ARMCI has been revised to prevent any single thread locking while waiting for network communication calls which would delay other threads ability to initiate network communication or in some applications would even lead to deadlocks. Initial evaluation of the thread-safe ARMCI using application benchmarks such as the SPLASH LU indicate the multiprocess and multithreaded implementations have the same performance.

3.5.4 Advanced Global Data Structures The Global Arrays library supports the dense multidimensional arrays with global view. Although this abstraction can be used to implement other data structures including sparse arrays, certain application areas require more advanced data structures like linked lists or hash maps with global view and independent, one-sided access. The global hash map can be asynchronously accessed (insert/update); however it is physically distributed among processes. An example of such an application is text processing engine in visual analytics area. The hash map distribution among processes is based on the hash value of the input. A global hash map is created collectively by all processes to store the unique terms (i.e. strings) and generate a global term IDs for each term inserted into the hash map. GPCs have been used to implement scalable distributed hash maps. They have been employed in the first ever

scalable implementation of the visual analytics text processing engine of the INSPIRE application.

3.5.5 One-Sided Access Model to Disk Resident Arrays As a part of the PModels project, we are developing an integrated programming model that supports out-of-core computations. The goal is to extend the one-sided access to globally addressable data to the secondary storage systems. The goals of the eXtensible Global Arrays (XGA) is combine the Global Arrays and Disk Resident Arrays into a single programming abstraction and relieve the programmer from the burden of explicit orchestration of data movements between in-core and out-of core storage. To advance toward this goal, we have been developing a noncollective, one-sided access to data in the secondary storage (Disk Resident Arrays). This capability follows the get/put style of communication for the distributed memory data. The main challenge is due to the fact that Disk Resident Arrays are implemented on top of collection of local disks in the cluster computing nodes and not just parallel filesystems. As described in the Cluster'07 paper, we addressed this requirement by developing remote read/write operations using GPC.

References

- [1] Manojkumar Krishnan, S. Bohn, W. Cowley, Vernon L. Crow and Jarek Nieplocha}, "Scalable Visual Analytics of Massive Textual Datasets", in *IPDPS*, 2007, pp. 1-10.
- [2] Vinod Tipparaju, Andriy Kot, Jarek Nieplocha, Monika ten Bruggencate, and Nikos Chrisochoides, "Evaluation of Remote Memory Access Communication on the Cray XT3", *IPDPS*, 2007, pp. 1-7.
- [3] Aniruddha G. Shet, P. Sadayappan, David E. Bernholdt, Jarek Nieplocha and Vinod Tipparaju, "A Performance Instrumentation Framework to Characterize Computation-Communication Overlap in Message-Passing Systems", 2006.
- [4] Sriram Krishnamoorthy, Juan Piernas Canovas, Vinod Tipparaju, Jarek Nieplocha and P. Sadayappan, "Non-Collective Parallel I/O for Global Address Space Programming Models", in *CLUSTER*, 2007.
- [5] Michael Blocksome, Charles Archer, Todd Inglett, Patrick McCarthy, Michael Mundy, Joe Ratterman, A. Sidelnik, Brian Smith, George Almási, José G. Castaños, Derek Lieber, José E. Moreira, Sriram Krishnamoorthy, Vinod Tipparaju, Jarek Nieplocha, "Blue Gene system software - Design and implementation of a one-sided communication interface for the IBM eServer Blue Gene", in *SC*, 2006, pg. 120.
- [6] Chris Oehmen and Jarek Nieplocha, "ScalaBLAST: A Scalable Implementation of BLAST for High-Performance Data-Intensive Bioinformatics Analysis", in *IEEE Trans. Parallel Distrib. Syst.*, 17 (8) pp. 740-749, 2006.

[7] Jarek Nieplocha, Bruce Palmer, Vinod Tipparaju, Manojkumar Krishnan, Harold Trease and Edoardo Aprà, “Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit”, in *Int. J. High Perform. Comput. Appl.*, 20, (2), pp. 1094-3420, 2006.

3.6 Rice University – John Mellor-Crummey

Rice researchers continue to explore the interaction of threads and Co-Array Fortran on leadership-class machines.

3.6.1 Overview. The highest priority goal of research and development efforts of the PModels team at Rice University has been to refine a version of Rice’s cafc Co-array Fortran compiler for use on the DOE leadership-class machines. In early June 2007, the Rice team completed an initial port of the cafc compiler and runtime system to ORNL’s leadership-class Cray XT platform and successfully executed CAF implementations of the NAS MG, CG, and LU parallel benchmarks. Currently, the cafc runtime system is using the GASNet 1.8.0 runtime layer. A limitation of this early release of GASNet for the Cray XT platform is that it limits the size of shared memory segments mapped on each node to 110MB, which limits our ability to test codes that manipulate large-scale data such as CAF implementations of HPC Challenge codes developed in the fall of 2006. Ongoing implementation work at Rice is aimed at improving the CAF language coverage along with the scalability and robustness of the run-time system. Ongoing research has focused on support to improve the expressiveness and performance of CAF. We have developed prototype language and run-time support for function shipping, structured teams, multiversion variables for producer-consumer communication, and process topologies. We are beginning to explore using software transactional memory as a mechanism to implement atomic actions for synchronization on distributed-memory multicore platforms.

3.6.2 Introduction As part of the Center for Programming Models for Scalable Parallel Computing, Rice University is collaborating with project partners in the design, development and deployment of language, compiler, and runtime support for parallel programming models to support application development for the “leadership-class” computer systems at DOE national laboratories. A principal goal of work at Rice University is refinement of the Co-array Fortran programming language, compiler and runtime so that it is 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 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. A secondary goal of this effort is to develop new compiler technology that will support higher-level “global view” parallel programming models. Over the term of the award, the work at Rice University will focus on the following themes:

- *Refinement of language-based parallel programming models for emerging platforms.* Rice will collaborate with members of the project team to refine existing designs for language-based global address space parallel programming models. This effort will focus on scaling such programming models to systems of 10,000 or more processors and effectively exploiting hardware threading on multi-core processors. Rice will lead the design of refinements to Co-array Fortran (CAF).
- *Refinement of language-based programming models for higher performance and better expressiveness.* Rice will collaborate with members of the project team in refining language-based global address space programming models so that they are natural for expressing a broad spectrum of algorithms and parallelization styles, as well as exploring language constructs that simplify achieving high performance by minimizing exposed communication latency. Another important goal of this work is to refine CAF so that it can be compiled into efficient code for a wide range of parallel platforms.
- *Investigation of compiler technology for global view parallel programs.* Rice will lead the investigation of techniques for compiling global view languages into efficient programs for scalable distributed-memory parallel systems. This investigation will focus on the design and evaluation of compiler technology for analyzing and optimizing programs by manipulating symbolically-parameterized descriptions of data, computation and communication. The aim of this work is to support parallel programming models with a higher level of abstraction.
- *Interoperability of parallel programming models.* Rice will work with members of the project team to refine run-time systems and application programming interfaces as necessary to make Co-array Fortran interoperable with other language and library-based models for parallel programming.
- *Integration with programming environments and tools.* Rice will work with members of the project team to develop strategies that enable language-based parallel programming models to interoperate with programming environments along with tools for debugging and performance analysis.

Rice will design, develop, and deploy open-source software that embodies the results of this research and development.

7.1.3 Technical Accomplishments

Rice University's FY2007 accomplishments as part of the Center for Programming Models for Scalable Parallel Computing include:

- *CAF implementation.* We ported Rice's `caf.c` compiler and run-time system to Jaguar, the leadership-class Cray XT platform at Oak Ridge National Laboratory.
- *Graduate education.* Two members of the Co-array Fortran project team, Yuri Dotsenko and Cristian Coarfa, completed their Ph.D. dissertations, which explored the design and implementation of compiler and runtime support for Co-array Fortran[1, 3]. This research was begun under support from the DOE under

the PModels project and completed at the end of January 2007 with support from this project.

- *CAF language, compiler, and runtime system research.* As Dotsenko finished up his dissertation, we finished prototyping and evaluation of compiler and run-time support for function shipping, structured teams, multiversion variables for producer-consumer communication, and process topologies. The results of this effort are detailed in his dissertation [3].
- *Synchronization support for multithreaded runtime systems.* We began exploration of strategies to make software transactional memory efficient enough for providing atomicity within a multi-threaded runtime on leadership-class machines.
- *Exploring CAF expressiveness and performance.* We designed and developed preliminary CAF versions of HPC Challenge benchmarks: RandomAccess, FFT, and streams. A student project recently yielded a draft of a blocked version of LU decomposition in CAF.
- *Work on compiler technology for global view languages.* We began exploring extensions to analysis and code generation techniques using polyhedral methods to partition computation efficiently for complex iteration spaces.

The prototype Co-array Fortran compiler `cafc` developed in with support from the DOE Office of Science as part of the PModels project is available as open source software from <http://www.hipersoft.rice.edu/caf>. Research and development results from this cooperative agreement are being delivered as enhancements to this software.

3.6.4 CAF Implementation The highest priority goal of our development efforts has been to port Rice's `cafc` compiler to the DOE leadership-class machines. In early June 2007, we completed an initial port of the `cafc` compiler and runtime system to ORNL's leadership-class Cray XT platform and successfully executed CAF implementations the NAS MG and CG parallel benchmarks. Experiments with other codes indicate that additional debugging is needed.

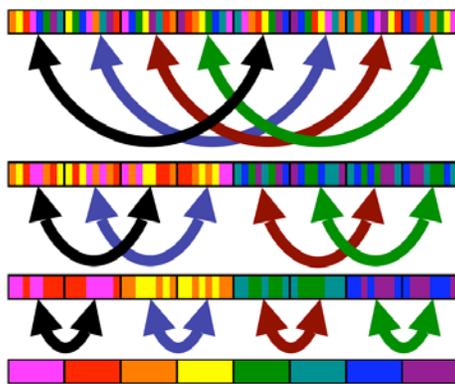
Porting to this platform involved modifying the `cafc` compiler runtime system to interoperate with PGI's Fortran 90 compiler. To support efficient communication, `cafc`'s runtime system must manage shared data (co-arrays) differently from other data managed by the Fortran 90 runtime system. For this reason, the `cafc` runtime delegates allocation of shared data to the GASNet or ARMCI communication libraries and then initializes Fortran 90 pointers so that this shared data can be manipulated efficiently from within Fortran.

Currently, the `cafc` runtime system is using the GASNet 1.8.0 runtime layer. On the Cray XT3/4 platform, this GASNet release relies on MPI for communication. A version of GASNet that uses Catamount's Portals communication layer directly is undergoing testing and debugging by our collaborators at UC Berkeley. That version will provide enhanced performance and we will migrate to it when our collaborators notify us that it is ready for external use.

3.6.5 Synchronization support for multithreaded runtime systems At present, a weakness of CAF is that it lacks adequate support for coordinating multithreaded computations on multicore processors. Lock-based synchronization is currently the most popular mechanism for synchronizing multithreaded computations. On distributed memory systems, acquiring locks on remote data and then manipulating remote data structures with PUT and GET primitives will lead to poor performance. In addition, lock-based problems often suffer from problems such as deadlock, priority inversion and convoying. We are interested in extending CAF with atomic actions to avoid such problems.

Software Transactional Memory (STM) is a promising technique for supporting atomic actions in multithreaded programs. STM provides a higher level synchronization abstraction than locks, the underlying synchronization is managed by a runtime system transparent to the programmer; and deadlocks, priority inversions and convoying cannot happen. If STM is to be useful in HPC applications, its overhead must be low.

Two major issues in software transactional memory implementation are conflict detection and validation. Conflict detection discovers the cases where two transactions perform two operations on an object at the same time and at least one of the operations is a write. Validation is a technique that detects harmful inconsistencies in the global memory state. To reduce the overhead of STM, we have developed a low-overhead validation strategy.



Communication pattern of HPC Challenge RandomAccess coded in CAF.

3.6.6 Plans

While the research and development of both the CAF language and compiler technology for PGAS languages has demonstrated that CAF can be used to achieve high performance on clusters, significant additional work is needed before CAF an attractive technology for computational scientists. Several issues need significant attention.

3.6.7 Implementation issues. When constructing an implementation of a blocked parallel version of LU factorization, three implementation issues were identified as impediments to user productivity. First, `caf.c` needs support for co-array variables as part of Fortran 90/95 modules. Second, `caf.c` needs enhanced support for inheriting implicit procedure interfaces for procedures that manipulate co-arrays. Third, `caf.c`

needs to support application of Fortran 90 intrinsic functions to co-array data. Our plan is to address these issues as soon as we have a functioning prototype of `caf_c` working on the ORNL's leadership-class machine based on the Cray XT architecture.

3.6.8 Enhanced support for manipulating remote data. Currently, CAF supports an MPI-like model for SPMD programming. For CAF to support a broader range of application styles, it needs better support for manipulation of remote data and more flexible data-oriented synchronization. Enhancing the expressiveness of synchronization is a focus of our CAF language refinement effort as part of this project.

3.6.9 Memory model. As we continue our exploration of advanced features in the CAF runtime (including support for atomic actions based on software transactional memory), an important task will be for us to define a memory model that will precisely describe the language semantics in the presence of function shipping and multithreading. The memory model must be natural for application programmers to understand and use, yet also not unnecessarily hobble performance.

3.6.10 Function shipping. We need to explore function shipping and multithreading on the leadership-class machines. Support for threading recently became available as the operating system on this machine was upgraded to Compute Node Linux. We need to precisely define what semantic guarantees about threading that the CAF language should provide to an application developer with function shipping.

References

- [1] C. Coarfa. Portable High Performance and Scalability of Global Address Space Languages. Ph.D. thesis, Rice University, Department of Computer Science, January 2007.
- [2] C. Coarfa, J. Mellor-Crummey, N. Froyd, and Y. Dotsenko. Scalability analysis of SPMD codes using expectations. In Proceedings of the International Conference on Supercomputing, Seattle, WA, June 2007.
- [3] Y. Dotsenko. Expressiveness, Programmability and Portable High Performance of Global Address Space Languages. Ph.D. thesis, Rice University, Department of Computer Science, January 2007.

4. Crosscutting Plans

In addition to the plans described by the individual institutions, two project-wide initiatives are in the planning state.

- The idea of a common communication subsystem API definition is being explored with a number of those attending the MPI Forum meetings. This is relevant to MPI and GA implementations and also to PGAS language implementations.
- The HPCS language project sponsored by DARPA was terminated this year. We hope to revive the cooperative approach obvious at the last meeting under the auspices of Pmodels. The HPCS languages represent the “third wave” of

programming models and languages, where the first two waves are libraries (MPI and Global Arrays) and PGAS languages (UPC, CAF, and Titanium).