The Dynamic Exascale Global Address Space (DEGAS) Project

Progress Report

March 2013

1. Introduction

Exascale computing will pose new challenges for software in the areas of concurrency, energy efficiency and resiliency. These challenges will be exacerbated by performance and programmability obstacles arising from deep memory hierarchies, heterogeneity, specialized processors, hardware adaptations, non-uniform data access latencies, variable network loads, and the challenges of fault tolerance. The challenge of supporting billion-way parallelism will be most disruptive within compute node, where parallelism is expected to increase by 2--3 orders of magnitude and may take different forms compared with today's systems; inter-node parallelism will increase by less than an order of magnitude and use an interconnect model that is functionally similar to existing platforms. Other challenges within the node include adapting to different types of data and task parallelism, hierarchical and possibly partitioned memory spaces, as well as performance and functional heterogeneity across cores.

The Dynamic, Exascale Global Address Space (DEGAS) project is a comprehensive effort to redesign programming models, and runtime systems for the radical machines design changes in the exascale era. It provides a tightly integrated view of the system across the intra-node and inter-node levels and thereby avoids unnecessary synchronization between levels. The approach focuses on a vertically integrated programming and execution environment that incorporates the latest algorithmic approaches and application structures to effectively service ultra-scale science and energy applications. The primary focus areas of DEGAS and the proposed integrated software stack are shown in Figure 1.

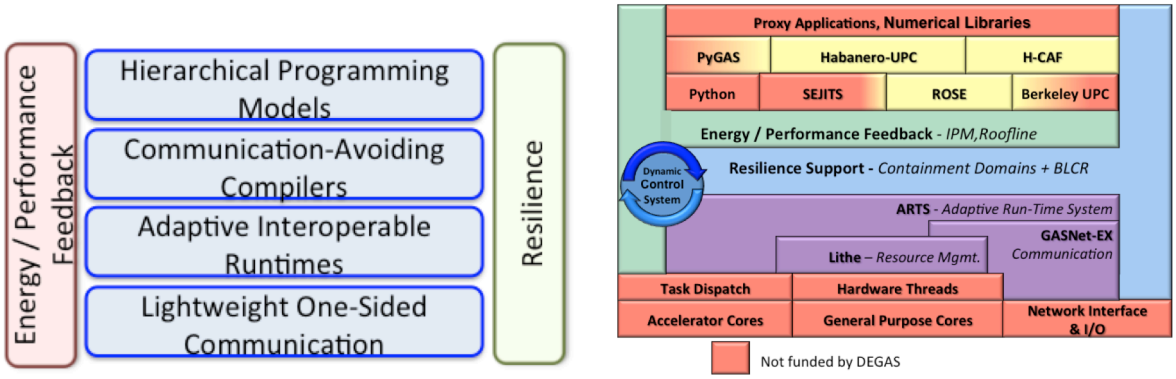

Figure 1: The primary research areas (left) and software stack (right) for DEGAS

DEGAS is multi-institutional center involving Lawrence Berkeley National Laboratory, Rice University, the University of Texas at Austin, the University of California at Berkeley, and Lawrence Livermore National Laboratory.

2. Recent Developments

Several developments in the overall DOE computing environment have taken place since the project started. New systems have been installed at Argonne (ALCF), and are in the process of installation and acceptance at Oak Ridge (OLCF) and Berkeley Lab (NERSC) that represent some of the challenges that will arise at the exascale. The OLCF Titan machine is a Cray XK6 with heterogeneous processing nodes containing an AMD X86 processor and an NVIDIA GPU; the ALCF Mira machine is an IBM BG/Q requiring fine-grained threading on a node and a unique form of SIMD parallelism; the NERSC Edison machine has more conventional Intel X86 processing nodes, but still has their own form of SIMD parallelism. Both the ALCF and NERSC systems involve new interconnection networks, and issues of congestion management at the node and global level and topology optimizations appear to be important across all three platforms.

These machines all support the message-passing model specified by the MPI standard but also have multiple cores per node, complex NUMA effects within nodes, and, at least the Titan and Mira systems cannot be programmed with solely with MPI concurrency. The DEGAS work is focused on future generations of exasacle applications and systems, which will look quite different than any of these existing architectures, but will nevertheless use these platforms and simulators to help validate the research ideas. Here are some of the highlights of the recent DEGAS work, with more detailed descriptions in each of the following sections.

- Habanero/UPC
- CA-Compilers
- Resilience

Details of two of these accomplishments are described below. In addition to these research activities, the project teams are planning software releases, which involved bug fixes, performance improvements, and extensive porting and testing across systems.

3. Hierarchical Partitioned Global Address Space (HPGAS) Programming Models

The challenges of extending PGAS to HPGAS can be divided into control (parallelism and synchronization) issues, and data (layout and communication) issues. We are looking at these in both a C/C++ context and in a FORTRAN context, with some exploratory work in Python.

3.1 C++ library for H-PGAS (LBNL)

To facilitate the transition from existing MPI+X applications to the PGAS model, and as a possible long-term implementation strategy for a C++ HPGAS language, we are

working on a C++ library approach for implementing the hierarchical PGAS programming model. We have designed the first version of the library API, which contains global address space pointers, dynamic global memory management, one-sided communication, synchronization primitives, and remote function calls.

In our current design, global address space pointers are represented by a <*node_id, local_address*> tuple and the referenced type information is statically included as a template parameter, for example, "*global_ptr_t<int> p*". Dynamic memory management functions are similar to the C "*malloc*" and "*free*" functions except that an extra argument about the desired memory location is added when allocating space. One-sided communication interface includes *Put* and *Get* operations as in UPC and GASNet. Synchronization primitives include *barriers* and *locks*. Remote functions calls mimic *active messages* but with the flexibility of calling arbitrary functions.

Most of the features are currently defined as C++ templates and we plan to provide a C interface in the future. In ongoing work, we are developing the programming constructs for hierarchical and heterogeneous systems.

3.2 Habanero with a Global Object Space (Rice)

As complement to the work on merging UPC and Habanero-C, the Rice team also recently developed the Habanero Asynchronous Partitioned Global Name Space (APGNS) programming model. This uses global naming of objects rather than global addressing. In this model, distributed tasks form the basic building blocks for parallel computations. The tasks communicate via single-assignment distributed data items. Each item has a unique global name, and a user-provided or system-generated distribution function specifies the home location for a given data item. The name of a data item can be viewed as a globally unique id (guid) that enables accesses to the same data item to be performed from any node in the system (by using the mapping provided in the distribution function).

The APGNS model can be implemented atop a wide range of communication runtimes that includes GASNet and MPI, and it also supports tight integration of intra-node task parallelism with inter-node communication. We have obtained preliminary results based on the use of MPI as the communication runtime for Habanero-APGNS. (MPI is not visible to users of the APGNS model, however.) Our implementation designates one core per node to serve as the "communication worker" whereas other cores serve as "computation workers". Early scalability results for benchmarks such as Smith-Waterman and Unbalanced Tree Search (UTS) show that the APGNS model can deliver significant improvements for both, relative to baseline MPI versions. These results are included in part of our IPDPS 2013 paper. We plan to extend these results in the future to use GASNet instead of MPI as the communication runtime.

3.3 PyGAS (LBNL)

PyGAS is a PGAS extension to Python. The goal of PyGAS is to support rapid development and evaluation of new parallel algorithms at large scale by providing an interactive and dynamic high-level programming system. The PGAS model is a powerful extension for parallel programming with Python in that the global address space makes it easy to build distributed data structures and express different types of parallelism. PyGAS is also a tool for the DEGAS team to research and experiment hierarchical PGAS programming features.

We implemented an initial PyGAS prototype and made it publicly available at https://github.com/mbdriscoll/pygas. Our PyGAS prototype implements PGAS features via a Python package extension and thus it requires no modification to the Python interpreter. The prototype implementation leverages one-sided communication, active messages and collective communication functions from the GASNet communication library. PyGAS uses the single program multiple data (SPMD) execution model and executes one Python interpreter per process. Data structures and programming constructs for data parallelism are being developed.

In addition, we have been studying different implementation approaches for dynamic PGAS languages in general and comparing their trade-offs. In particular, we analyzed the design space of:

- Representation of global shared objects
- Semantics of function calls for remote objects
- Efficient support for both data parallelism and task parallelism

4. Communication-Avoiding Compilers

The DEGAS project has three existing compilers, the CAF compiler based on Rose, the Habanero compiler also based on Rose, and the UPC compiler based on Open64.

4.1 Habanero-UPC Compiler (LBNL and Rice)

We have designed a compilation framework for Habanero-UPC programs, which extends our existing Habanero-C and Berkeley UPC software infrastructures.

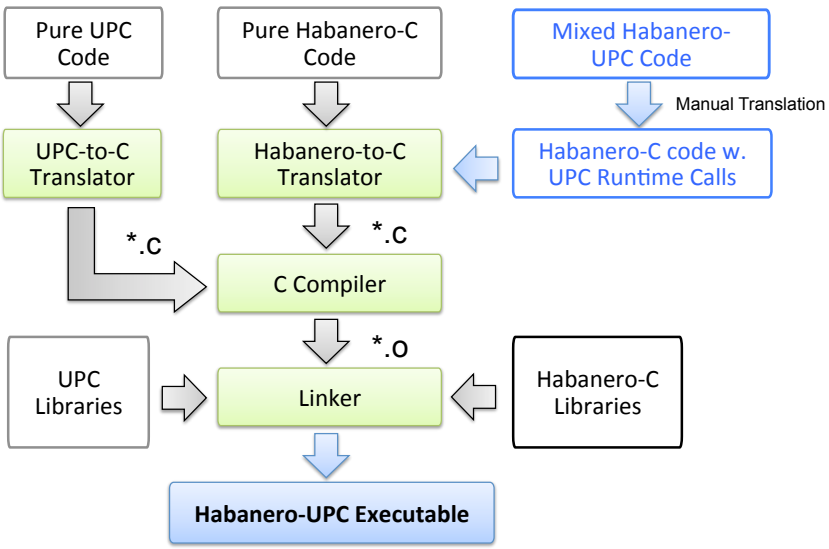

Figure 2: Compilation framework for Habanero-UPC programs

In this initial compilation framework design, UPC programs and Habanero-C programs are parsed separately by the existing Berkeley UPC compiler and Rice Habanero-C compiler, respectively. For mixed Habanero-UPC codes, we first manually translate the UPC usage in the program to the equivalent UPC runtime function calls, and then pass the resulting Habanero-C code with UPC runtime calls to the Habanero-C compiler to generate C code. The bulk of the translation from a Habanero-UPC program to a Habanero-C program with UPC runtime calls is straightforward because many UPC features are directly provided through a library interface in Berkeley UPC. This agile prototyping strategy will enable us to start application study for the new Habanero-UPC programming model in parallel to the development of more comprehensive compiler and runtime support.

In the next step, we plan to automate the translation from Habanero-UPC code to C code by two possible approaches: (1) add a source-to-source compiler front-end to parse Habanero-UPC programs and generate C code with runtime function calls; (2) leverage standard C++ compiler support for generic programming (templates) and operator overloading to do the translation.

4.2 Co-Array Fortran (CAF 2.0) Compiler (Rice)

Many scientific applications perform stencil computations on a multi-dimensional Cartesian domain decomposed among a set of processors and employ a communication pattern known as a "halo exchange," in which processors exchange data along the boundary of blocks that they own. To investigate issues related to this important application pattern, we adapted the CAF version of the CGPOP mini-application derived from LANL's parallel ocean program into Rice's CAF 2.0. The original CAF version of CGPOP relied on MPI to provide collective communication. The CAF 2.0 version uses CAF 2.0 primitives for collectives instead. Also, we have begun adapting Sandia's MiniGhost mini-application into CAF 2.0 and will shortly begin experimentation with this code as well.

Initial small-scale experiments with CGPOP on a Cray XE6 (Hopper) show that our CAF 2.0 version, implemented with a source-to-source translator based on ROSE, has communication performance comparable to both the CAF and MPI versions of the benchmark. However, overall the CAF 2.0 was slower than the Cray CAF version. An investigation with Rice's HPCToolkit performance tools showed that the node performance of CAF 2.0 generated code for computation-intensive loop nests was twice as slow because it uses Fortran 90 pointers to manipulate data for Coarrays. Experiments in which the two most costly loops in CGPOP were outlined to avoid use of pointers in the loops confirmed this as the root of the problem. With Fortran 2008 contiguous attributes for pointers, we expect comparable performance for CAF 2.0 generated code without outlining. Larger-scale experiments with CGPOP revealed intermittent delays using netcdf for parallel I/O from CAF 2.0 problems and intermittent failures with

GASNet initialization. We are investigating causes of this interference and initialization failures.

Ultimately, our aim with codes such as CGPOP and MiniGhost is to exploit one-sided communication in CAF to move away from bulk synchronous communication and make better use of communication networks on parallel systems by spreading communication traffic out over time.

To address the needs of applications that manipulate data structures with irregular access patterns, e.g., particles or graphs, we have begun exploring the design space of language, compiler, and runtime features to support these computations. HPF 2.0 provided language support for managing schedules and compiled global view irregular computations to leverage library primitives based on two-sided communication. With partitioned global address space languages, one-sided communication and runtime caching of remote data is an alternative to communication libraries based on two-sided primitives. In an SC12 paper, Checconi et al. describe the fastest implementation of the Graph 500 breadth-first search algorithm on Blue Gene systems which uses one-sided RDMA. By considering these different approaches to performing irregular computation, we aim to gain insight into the right balance between language, compiler, and runtime systems, along with a better understanding of the important idioms for mapping irregular computations efficiently into PGAS languages.

In the area of communication-avoiding compilers, we have begun to explore how to map these algorithms efficiently onto multi-level memory hierarchies. To motivate work on communication-avoiding algorithms, we have begun to experiment with several DOE applications and benchmarks to identify an appropriate kernel that we can use to help guide our work in this area.

Finally, we have been exploring the design and implementation of the phaser synchronization construct for distributed memory machines. Our design for a distributed phaser implementation uses a data structure based on distributed skip lists. When each member of a team of processors indicates that they want to participate using signal, wait, or signal/wait, we synthesize the distributed phaser data structure using an efficient algorithm based on scan operations. Currently, we are exploring how to accommodate dynamic additions and deletions to the phaser representation.

4.3 Generalizing Communication Avoiding Algorithms to Compilers (UCB)

Prior work showed a class of "2.5D" algorithms that are communication optimal in dense linear algebra. We are pursing the use of these ideas in other domains, for general loop nests containing array expressions. The specifics will appear in paper in the next few months, but we have developed a theory (in collaboration with others) that works for an arbitrary number of loops and for arrays containing linear index expressions.

Over the past few months, we have also applied these ideas outside the linear algebra domain to consider a direct method for calculating particle forces as in n-body

calculations. The starting point is an $O(n^2)$ computation for an all-pairs approach, although cut-off distances are also considered in the framework. The basic idea is that, going beyond domain decomposition of the particles, each particle is replicated over a subset of the processor and the updates to that particle done independently by the copies, which are combined together with a reduction operation at the end. This "replicate-and-reduce" strategy goes

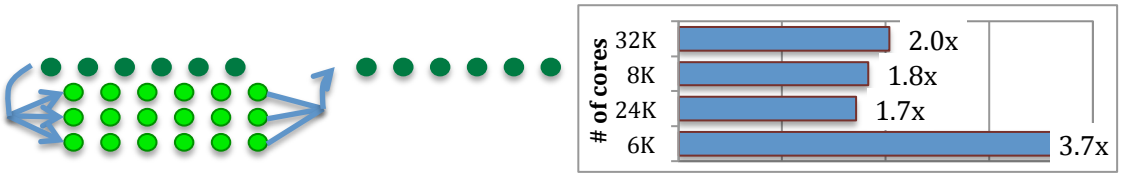

Figure 3: Replicate and reduce idea and summary of performance speedups on a Cray XE6 (6K and 24K cores) and BG/P (8K and 32K cores)

In addition, the LBNL team has looked at the problem of combining communication-avoiding algorithms and communication-overlap. On the surface, these are orthogonal ideas, as avoidance reduces the number and size of messages (sometimes trading the two off against each other) while overlap reduces the latency cost by overlapping with computation or other communication events. While communication avoidance is important in 3 of the 4 examples shown in Figure 4, there is an additional benefit in 2 of the codes from adding overlap. The Cholesky example is currently limited by dependences in the code, and is being rewritten to expose more parallelism over the next year.

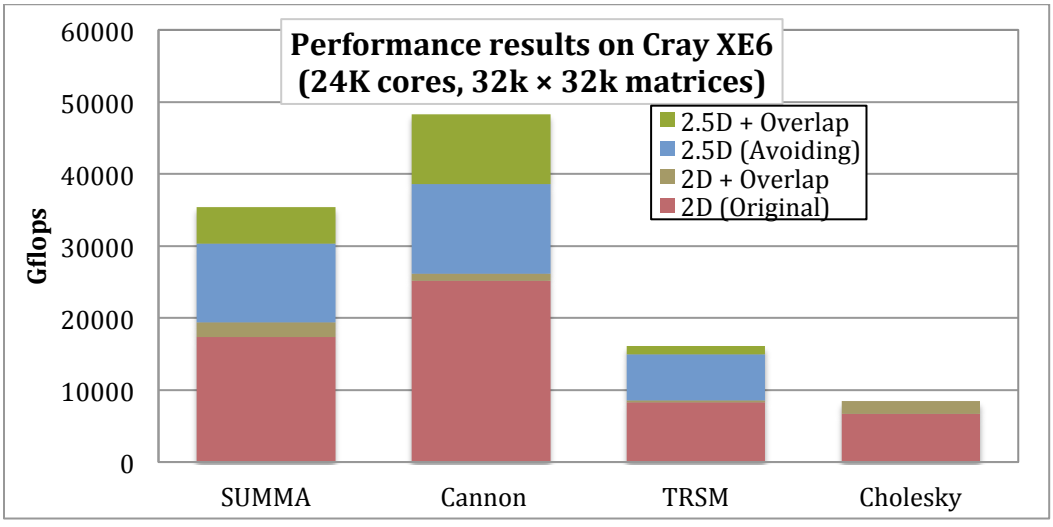

Figure 4: Combination of communication avoiding algorithms (2.5D) and communication overlap

5. Adaptive, Interoperable Runtimes

During the review period we have started the integration of the multiple software infrastructures from DEGAS participants, as well as researching the support and interfaces between modules in the DEGAS ARTS. We have worked in the following areas: 1) UPC-Habanero runtime integration; 2) resource management in ARTS; and 3) integration of communication and task scheduling. We also released the Berkeley UPC infrastructure at SC 2012 and plan another release around May 2013.

5.1 Habanero Runtime and Merge with UPC Runtime (Rice and LBNL)

The Rice team made the following progress on adaptive inter-operable runtimes:

1) We have implemented a node-level runtime system for Habanero-C with a "communication worker" model that can support tight integration of intra-node parallelism with inter-node communication. This implementation is described in our IPDPS 2013 paper, and is now part of the Habanero-C trunk.

2) We have also updated the compiler support for Habanero-C to the latest version of Rose, in preparation for integrated compiler support for Habanero-C and UPC.

3) Finally, to demonstrate a connection between Habanero-C and the recent Open Community Runtime (OCR) project, we have built a library for Habanero-C constructs (HClib) that can be implemented on OCR with support for constructs like "finish" available in HClib but not directly available in OCR i.e., we had to implement "finish" on OCR to enable this support.

The LBNL team has demonstrated the interoperability of the UPC and Habanero languages using simple benchmarks. In the current design, UPC acts as an SPMD driver, which launches per-node instances of the Habanero runtime. Inter-node communication is performed through the UPC runtime. Both UPC and Habanero can be used for intra-node programming. As an initial driver for UPC and Habanero evaluations, we have ported an MPI+OpenMP multigrid benchmark (also used by XTune) that represents some of the challenges associated with combustion codes like LMC. As ROSE compilation errors are currently impeding the Habanero evaluation aspect, we are also creating a UPC+OpenMP version that will allow us to evaluate the performance and productivity trade-offs of one-sided communication on the current inventory of DOE platforms.

5.2 Resource Management in ARTS (UCB and LBNL)

In the area of resource management we have extended Lithe and Juggle with dynamic capabilities. Lithe provides allocation of cores to applications, while Juggle provides scheduling and load balancing over a set of cores.

5.2.1 Lithe

For this initial period, our primary activity has been a substantial rewrite of the base Lithe functionality, which is due for release next month. The Lithe rewrite incorporates the following major changes:

1) Clean separation of Lithe's API from the underlying operating system functionality.
In the new version, any schedulers developed on top of Lithe should beportable to any operating system that implements the Lithe layer. The new version runs successfully on top of Linux/x86, Linux/x86_64,Akaros/x86, Akaros/RISC-V, and is currently being ported to Tessellation/x86 and Tessellation/RISC-V (Akaros and Tessellation are Berkeley research operating systems, while RISC-V is a Berkeley research ISA).

2) Restructuring to support future preemption work.
Although the new release is supports only co-operative sharing of hardware threads, the internal structure of Lithe has been altered in preparation for the addition of user-level preemption support. In particular, the tracking of which hardware threads are under control of each scheduler has been modified to make it easier to locate the appropriate hardware thread to reallocate in the case of preemption.

3) Updated OpenMP and TBB ports.
The GNU OpenMP (gomp) and Intel TBB runtime libraries have been ported to the new Lithe layer and a number of outstanding bugs were fixed in the port. With this update, OpenMP and TBB runtimes should be immediately portable to any operating system to which Lithe has been ported, with OpenMP and TBB able to cooperatively share hardware threads within the same application.

4) Extensive documentation and website.
A new website (lithe.eecs.berkeley.edu) has been established and now contains extensive documentation of the Lithe API to support development of new schedulers. The source code repository is now publicly available on github.

5.2.2 Juggle

We have extended the Juggle scheduling framework to include support for dynamic thread counts, i.e. Juggle can now adapt to an increased or decreased number of threads on a node. This improves our ability to support SPMD models in dynamic settings, for example, if checkpointed threads on a failed node are migrated to running nodes, Juggle can incorporate the new threads and balance the load accordingly, allowing the job to keep running.

5.3 Integration of Communication and Task Scheduling (LBNL)

We have performed work in the areas of integration of tasking and communication with respect to memory consistency models and message ordering.

Recently, Cray introduced interconnects, Gemini/Aries, supporting multiple ordering relaxation of RDMA memory operations. While the use of strictly-ordered operations provides the easiest path to correct execution, it can penalize the performance significantly for RDMA operations, leading to up to 4x slowdown compared with the best sustained bandwidth, see **Error! Reference source not found.**. On the other hand,

relaxed ordering of memory transfers improves the performance significantly, see **Error! Reference source not found.**, but is more difficult to use for correct execution.

During this research period, we implemented a mechanism that does support relaxed ordering of RDMA, while correctly meeting the memory consistency models supported by the runtime. When needed, we ensured the atomicity of remote updates, thus allowing to establish a global order of interconnect transactions and local memory transactions. This implementation is incorporated into GASNet communication library, which is used by Berkeley UPC and other global partitioned address space languages.

We also conducted a performance study of multiple sets of low-level APIs provided by Cray for runtime and compiler development, GNI and DMAPP.

Showing that both APIs can provide the same sustained performance with appropriate tuning, we focused our efforts on GNI because it provides a mechanism for both messaging and remote direct memory access. Additionally, GNI provides two mechanisms for communication messages, FMA and BTE. The first is optimized for small messages while the latter is optimized for large memory transfer (higher setup cost and less later involvement of the software layer). For optimal performance, switching between these two communication modes is dependant on runtime information such as message size and the concurrency in injecting messages. We studied heuristics to optimally switch between these modes.

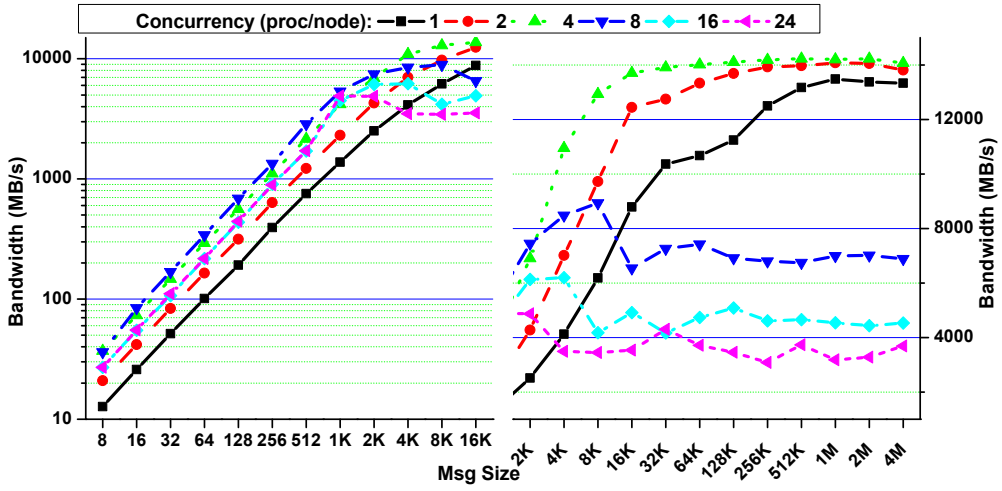

Figure 5 Impact of concurrency on the BTE Get operation bandwidth with strict ordering on Cray XE6. Having more than 4 injecting processes can degrade performance by up to 4x for large messages

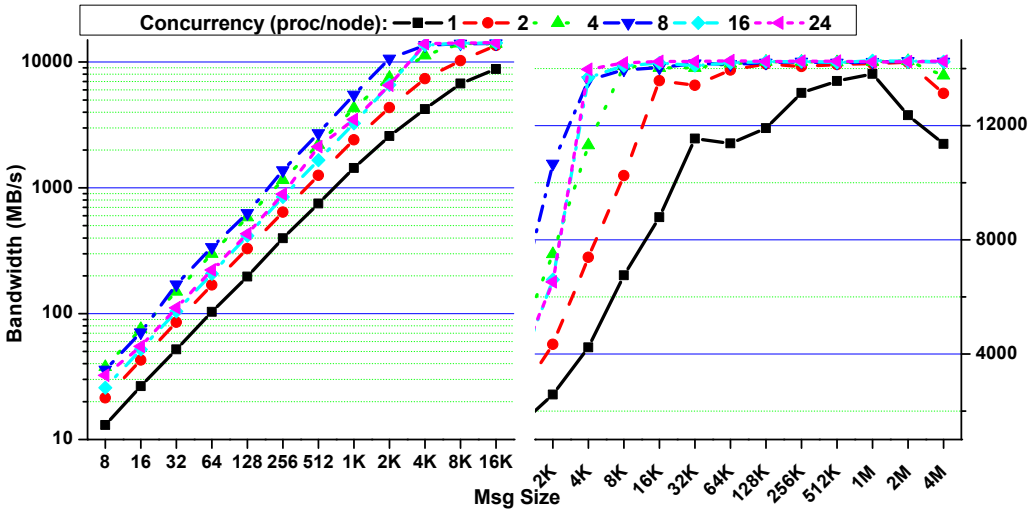

Figure 6 Impact of concurrency on the BTE Get operation bandwidth with relaxed ordering on Cray XE6. Increased concurrency always improves performance.

We have also started the interface design for the integration of THOR (Throughout Oriented Runtime) with the Berkeley UPC and Habanero runtimes. THOR is a message scheduling layer able to provide on the fly message coalescing, re-ordering and end-point flow control mechanisms. THOR can perform communication through "servers" which are tasks dedicated to performing communication operations. This strategy is also employed by HC-MPI (Habanero-MPI) which spawns communication dedicated workers. We are extending THOR to be a plug-and-play replacement.

6. Lightweight One-Sided Communication (LBNL)

The focus of the Lightweight One-Sided Communications area is to specify and develop GASNet-EX, the successor to GASNet. GASNet has been an on-going joint project of LBNL and UC Berkeley since 1999, under funding from the DOE Office of Science, and the DoD. GASNet's main features are a rich set of one-sided Put and Get interfaces for implementation of remote memory access (RMA) primitives in PGAS programming models, and Active Messages (AM) for efficient remote execution. GASNet has been widely adopted by implementers of PGAS languages and libraries, making it the *de facto* standard.

The GASNet-EX effort within the DEGAS project is a major revision of the GASNet specification and implementation. Within this project we are revising the specification to better support modern PGAS models and current and future system architectures – both areas of significant growth and change since GASNet began. These revisions will make GASNet-EX better suited for implementation of the DEGAS software stack than its predecessor. With the new specification will come a new implementation, rather than incremental changes, enabling some major new features such as support for the checkpoint/restart and migration capabilities of DEGAS's Resilience area.

The goals for year 1 of the project are

- A draft API specification for GASNet-EX
- Prototype implementation of forward-compatible features

While an important long-term (year 3) goal is

- Complete implementation for DOE supercomputers of 2015

Progress toward each of these three goals is described in the following paragraphs.

6.1 GASNet-EX Draft Specification

The revised GASNet-EX specification is not targeted at only the DEGAS software stack. For that reason, we conducted face-to-face meetings (2 to 4 hours each) with several PGAS implementers while at SuperComputing in November 2012. These included three current users of GASNet: the CAF and OpenSHMEM implementers from University of Houston, and Cray's Chapel team. Additionally we met with the IBM X10 team, who are not currently users of GASNet, but instead maintain their own Open Source runtime to support the X10 language. These meetings were extremely valuable to collect feedback on ideas that we had already been considering, and to collect new ideas. Those meetings resulted in several pages of notes, some of which have been converted to electronic form at https://sites.google.com/a/lbl.gov/gasnet-ex-collaboration/. Over time the remainder of the notes will be entered there, and the site will become a focal point for open discussion as the GASNet-EX spec develops.

Following SC12, we were contacted by the Cray Chapel team to follow-up with discussion of the GASNet-EX plans for resilience. The summary of that dialogue is available from the GASNet-EX collaboration website (URL above).

In addition to the face-to-face meetings with other PGAS implementers, we are in electronic contact with our user community. Two discussion threads on the Berkeley UPC user's and developer's mailing lists (which also serve as the GASNet support lists) have lead to promising ideas for improving the usability and expressiveness of GASNet-EX.

6.2 Prototyping Forward-Compatible Features

As a side-effect of the work (see 6.3) on Cray Aries and Gemini, a top-to-bottom survey of the current GASNet software architecture is underway. The goal is to develop a clearer understanding of which new features planned for GASNet-EX are candidates for prototype implementation in the GASNet code base. Specifically we are studying which features can be added to GASNet without breaking current semantics, and without excessive rewriting of the code. To put that another way: we are sorting out which additions can be incremental and which cannot.

Features already in the "pencil and paper" stage of prototyping include: exposing local completion of RMA operations, fenced RMA operations, callbacks for client progress, expanding the AM interfaces, and relaxation of the thread-safety rules for non-blocking

operations. These particular planned enhancements are all of significant potential benefit to the sort of asynchronous runtime that DEGAS is developing.

As we identify planned features that *cannot* be implemented as incremental changes to the current GASNet code, we are taking careful note of what architectural changes will be required. For example support for multiple clients (necessary to support multi-language or multi-model applications) and for multiple endpoints (to allow threads to have a "first class" status) will require changes to how peers are named and a corresponding move away from dense one-dimensional arrays to represent peers. Such a change is also highly desirable for scalability reasons.

6.3 Support for Current and Future Systems

As was mentioned in Recent Developments, Mira is now online at ALCF and Titan and Edison are each in some stage of availability at OLCF and NERSC, respectively. Given the lifespan of system at all three centers, we can reasonably expect that GASNet-EX will need to support all three.

Mira, the BlueGene/Q at ALCF, uses IBM's PAMI network API. IBM has positioned PAMI as their single network API for HPC, merging the LAPI (from the IBM SP line) and DCMF (from the BG/P) efforts. As such, IBM also offers PAMI on the P775 (aka IBM PERCS) system and on InfiniBand clusters. As of our October 2012 release GASNet fully supports PAMI and has been tested on both BG/Q and P775 systems.

The design of PAMI appears to have drawn on several ideas from the Berkeley AM2 work, which is also the origin of many of GASNet's key features. As a result, GASNet's PAMI support is currently the "thinnest" of the GASNet conduits (our term for the network-specific portion of GASNet). Some of the biggest additions to GASNet-EX will be the multi-client and multi-endpoint support, both of which map to corresponding features in PAMI. We anticipate IBM's PAMI-based systems will be the easiest to support in GASNet-EX.

Titan, the XK7 at OLCF, uses the same Gemini interconnect hardware and uGNI network API as Hopper, the XE6 at NERSC. GASNet's gemini-conduit for Cray's XE and XK series has been available since October 2011, and is used in production on Hopper. However, gemini-conduit has remained in "Beta" status since its original release because it was known to have some serious performance problems. The most significant performance defect was due to the reliance on "strict ordering" for correctness, while "relaxed ordering" is necessary to achieve acceptable performance when utilizing more than four cores. This problem has been overcome and a mechanism has been developed (with the crucial assistance of Cray) to allow use of relaxed ordering without sacrificing correctness. That effort was described in more detailed in Section 5, which includes micro-benchmark results showing the benefit.

In addition to the switch from strict to relaxed ordering, GASNet's gemini-conduit has been extensively rewritten over the past two months to improve performance and

scalability. At the time this report is being written the rewrite is nearly complete and approximately half of the code in gemini-conduit is new since the October 2012 release. As a result, the Spring 2013 release (expected late April or early May) of GASNet will no longer label support for the Cray XE and XK series as "Beta".

Two UPC benchmarks run at small scale on Hopper illustrate the magnitude of the improvement in performance for the rewritten gemini-conduit. The first is "cg.D.O3.256", the CG benchmark from GWU's UPC version of the NAS Parallel Benchmarks with the Class D problem size and 256 UPC threads. The problem was run on 16 nodes, using 16 (out of 24) cores on each. The times for three different UPC compilers are reported in Table 1 which reports the mean and standard deviation of 15 runs (three for each compiler from each of five batch jobs). By running the application as built by all three compilers in the same batch job we ensure that effects of job layout on Hopper will impact the results fairly.

	BUPC 2.16.0	BUPC Current	Cray CCE 8.1.3
cg.D.O3.256	20799 ± 946 Mop/s	25536 ± 805 Mop/s	21917 ± 1121 Mop/s

Table 1. CG benchmark on Hopper using Berkeley UPC and Cray CCE

The first two results are from the Berkeley UPC compiler, where "BUPC 2.16.0" is the October 2012 release as installed for production use on Hopper and "BUPC Current" is the Berkeley UPC compiler rebuilt with the current version of GASNet's gemini-conduit. The result is a 23% performance improvement. Additionally, the performance goes from trailing Cray's result by 5% to leading by 17%.

The second UPC benchmark illustrating the improved performance of the rewritten gemini-conduit is the "guppie" benchmark which is a variation of on the same theme as the well-known RandomAccess benchmark from the HPC Challenge suite. By its nature, this problem is latency bound, and therefore the results (expressed in "up/s") is very sensitive to how the job gets placed on Hopper's torus network (unfortunately, placement on a contiguous portion of the torus is rare). The problem size tested was 2^{30} updates with a table size of 2^{26}, using all 384 cores of 16 compute nodes. All runs were "paired" with both the 2.16.0 and "current" BUPC compiler's versions of the code run five times in each batch job. For the least contiguous job placements, both versions of the code faired equally poorly with results below 35Mup/s. The lack of statistically significant performance differentiation in these runs is not surprising because the network latency is the dominant factor and we've not changed the network traffic patterns. For the few jobs that received "favorable" job placements the new code showed measurably improved results, as the latencies we *can* address in software became more significant. Results of six representative[1] runs are shown in Table 2, in which the runs have been ordered by the performance of the code as compiled by BUPC 2.16.0. It is worth noting that we could not get this benchmark problem size to run reliably when compiled with Cray's compiler.

Job No.	BUPC 2.16.0	BUPC Current	Change

[1] From ten batch jobs we present the 2 slowest, the 2 fastest and the 2 median (middle-most).
[2] We have been working actively with Cray to get the documentation updated to match the implementation,

2876148	32.19 ± 6.04 Mup/s	36.13 ± 1.40 Mup/s	*
2876158	32.89 ± 2.32 Mup/s	33.43 ± 1.88 Mup/s	*
2876154	40.44 ± 0.23 Mup/s	41.07 ± 0.53 Mup/s	*
2876157	41.55 ± 0.48 Mup/s	42.84 ± 0.40 Mup/s	3.1%
2876149	47.07 ± 0.27 Mup/s	51.72 ± 0.62 Mup/s	9.9%
2876154	52.28 ± 0.86 Mup/s	73.45 ± 0.98 Mup/s	40%

Table 2. Guppie benchmark results on Hopper for two versions of BUPC. A '*' appears in the Change column when the difference is less than 2σ.

The newest machine on the floor of a DOE center is NERSC's Edison, a Cray XC30 (a.k.a. "Cascade) system. Relative to the XE and XK series both the CPU vendor and network hardware have changed. While the switch from AMD to Intel CPUs has little impact on GASNet or GASNet-EX, the change of network from Gemini (a 3D torus) to Aries (a dragonfly) is a significant change. Fortunately, Cray has kept the uGNI API, but unfortunately there were non-trivial changes in uGNI. These changes include dropping support for one feature and changing the semantics for others[2]. The dropped feature was one we relied heavily on in gemini-conduit, and so from the start it was clear that non-trivial work was required to support Aries. Since Gemini and Aries share the uGNI API, the decision was made to support them from a common code base, with appropriate conditional compilation (#ifdef) to deal with the differences. The development of a new aries-conduit was done over the past two months, together with the rewrite of gemini-conduit and their code is currently 98% common. We anticipate that the choice to support them from a common code base will prove to be the best option for maintainability.

At the time of this writing aries-conduit is complete, stable, and at least moderately tuned. It has been released to a limited set of partners for initial testing. We fully expect to include aries-conduit in our Spring 2013 release of GASNet and Berkeley UPC.

In addition to the PAMI and uGNI APIs from IBM and Cray, GASNet also has support for three InfiniBand APIs, of which we expect at least the most widely used one will carry forward into GASNet-EX. Of additional interest for the future is Portals4 from Sandia, UNM and Intel. Though Sandia is not a partner in DEGAS, we have a verbal commitment from Sandia to produce a portals4-conduit for GASNet in time for our SC13 release of GASNet, and hope to see an initial version ready for Beta in the Spring 2013 release. We expect to share with Sandia the support of portals4-conduit, and it is likely to be carried forward into GASNet-EX.

7. Resilience

For the first year of this project, the resilience team has two overall tasks. The first task is to define a useful and implementable fault and recovery model for PGAS languages. The

[2] We have been working actively with Cray to get the documentation updated to match the implementation, but it seems that not even Cray was fully aware of everything that they had changed.

second task is to develop the infrastructure required to support a working implementation of this model.

In the past few months we focused our efforts on two main tasks. The first task focuses on our first-year deliverable of a generic and complete PGAS resilience consistency model, which includes partial and uncoordinated preservation and recovery. The second focuses on ground work necessary for our long-term project deliverable of demonstrating a containment domains model and runtime for hierarchical PGAS.

There are several significant challenges to defining consistency in the context of a generic resilient PGAS application. Unlike models with explicit two-sided communication, any parallel task can potentially access any global memory address across the system at any time. As a result, uncoordinated recovery is problematic because: (1) tasks that are recovering by re-execution observe memory state that is inconsistent with their original execution; (2) tasks making forward progress and re-executing tasks may interfere with one another; and (3) tasks that are re-executing may generally end up with a state that is inconsistent with the rest of the application. Achieving a consistent state requires careful design of the execution model and potentially restrictions on the types of algorithms that can be guaranteed consistency.

7.1 Containment Domains (UT Austin)

We carefully reviewed prior material and a partial prototype implementation of a CD runtime (by Cray) and developed a plan for a lowest-common-denominator implementation of CDs targeting GNU libc. Our main accomplishments are itemized below and described with greater detail in the subsections below:

1. Developed an initial version of a containment domains-based generic resilience model for PGAS applications.
2. Identified required system support for implementing the initial CD-based PGAS resilience model.
3. Conducted a detailed and critical review of available prototype code that was developed by Cray and which implements a subset of CD runtime functionality.
4. Developed detailed plans for an initial “least common denominator” CD runtime implementation.

7.1.1 Containment Domains for PGAS Resilience

We started by extending the containment domains resilience model to a generic PGAS context. With containment domains, the application is logically partitioned into a tree of nested CDs. Each CD provides means to (potentially partially) preserve data necessary for its execution and a mechanism for recovery. Recovery is typically achieved by re-executing a CD (and all its children) or escalating the failure to its parent for handling. By carefully balancing the overheads of recovery and re-execution using knowledge of system parameters and failure models, a CD-enabled application can achieve maximal resilience with minimal overhead and express a rich set of resilience mechanism. The CD

model, originally developed in the context of an explicit communication, defines two CD flavors: strict and relaxed. With strict CDs, parallel tasks may y only communicate when they are all within the same CD context. Two (or more) relaxed CDs, however, can exchange data directly. Strict CDs are simpler to reason about and implement, while relaxed CDs offer increased flexibility of trading off preservation and recovery overheads.

The strict flavor of containment domains, achieves the goals of the generic resilience PGAS model stated in the beginning of this section. However, based on initial experimentation we believe that the strict CD model is too restrictive and cannot be used to express efficient resilience for a large class of applications. Our main effort was thus targeted at developing the model semantics and specifying system support for relaxed CDs in a PGAS context. As mentioned earlier, the challenge lies in the fact that CD recovery, by nature, is hierarchical and uncoordinated, which leads to a globally inconsistent view of globally shared memory. Our initial model, which we are currently verifying for completeness of correctness is summarized in the minimal list of requirements below:

1. The application is deterministic, in that all re-executions of any CD results in the same values stored to globally shared memory as in the original execution of that CD (to within acceptable precision).

2. All read operations from data that is shared between relaxed CDs are logged for re-execution; or alternatively a versioned memory system can provide originally read values.

3. Write operations to potentially-shared data from any re-executing CD that has previously committed are squashed (they already were potentially observed and consistency demands that they not be re-executed).

4. Write operations to globally-visible addresses that are temporarily private to the re-executing CD may re-execute to a shadow memory space to avoid corrupting previously communicated global state.

5. All synchronization operations between relaxed CDs are logged and squashed during re-execution.

The implications of this model are that logging and replay mechanisms are required for shared data, whether global or local, and that system calls must be logged to ensure deterministic re-execution. In addition, and related to deterministic re-execution, memory management operations also require special handling to ensure consistent use of pointers (or data identifiers) throughout the application and to prevent memory allocation leaks. An important aspect of such logging is the need to define semantically consistent and safe points in the execution were logged information can be discarded. Our conservative approach is to define the release point as the least-common-ancestor strict CD for all

relaxed CDs that share particular data or that encompass a allocate/potential-use/free sequence.

7.1.2 PGAS Containment Domains Prototype

One of our long-term goals for the DEGAS project is to demonstrate a hierarchical PGAS system with support for containment domains. Achieving this goal requires careful planning and a multi-year implementation effort. Because the CD runtime must rely on some aspects of the underlying DEGAS runtime support for task control, communication, synchronization, and memory management, we decided to focus our initial effort on support for CDs within the GNU libc context and on detailed reviews of existing codebases and mechanisms for containment domains specifically and state preservation in general. GNU libc (or equivalent) is the most common base system used to implement parallel runtime systems. Thus, this initial work sets the groundwork for the full implementation. Also, despite this restrictive first target, many of the features and challenges of the full CD system are apparent within libc.

7.1.3 Containment Domains within GNU libc

We reviewed the majority of libc methods that are stateful and thus require special handling, including methods for memory management, file I/O, random number generation, and interactions with the hardware system (e.g., the gettimeofday() method). Based on this review our current primary focus is memory management methods (i.e. malloc, calloc, realloc, and free), which we believe are the most critical for the success of the runtime. While not yet implemented, we are confident that this design ensures correct and consistent execution and re-execution, while adding negligible runtime overhead; only memory management method calls are augmented whereas memory accesses are unmodified. The downside is that free operations are deferred, potentially increasing the memory footprint of applications that heavily utilize dynamic memory management. While we do not expect this to be a problem with any of our current target applications, we are also considering alternative designs that reduce the potential impact on memory footprint.

7.1.4 Analysis of the Cray Research Prototype Implementation of CDs

We conducted a detailed and critical review of all prior CD literature and the GPU/CPU-targeted CD preservation prototype developed by Cray. It is important to understand that the Cray prototype was developed with the very specific goals of demonstrating that CDs can be implemented, that they provide a benefit for heterogeneous architectures, and that the execution overheads added by CDs are small. The Cray code achieved these specific goals and provides an important and useful first-cut design for a CD runtime. However, we concluded that significant additional implementation and design work is necessary.

Specifically:

1. The Cray design should be augmented to be more general and support greater diversity in infrastructures and programming models.
2. Many of the API methods currently implementation require renaming, refactoring, and extensions to simplify their use.
3. Additional functionality is required to correctly handle memory management and other low-level system functionality.
4. Support is required for PGAS, as the Cray runtime focused on MPI and GPU/CPU hybrid nodes.
5. The CD model implemented by Cray is not sufficiently flexible and does not efficiently support all CD hierarchy manipulation required in the general case.
6. Only a single preservation method was implemented.
7. The prototype is purposefully limited to only recover from GPU failures and assumes that the system, node, process, memory, and libc threads are perfectly reliable.

Despite its incomplete and initial state, the Cray code base and design documents provide an excellent starting point and will save significant effort in our implementation.

7.2 BLCD (LBNL)

We are using BLCR as part of our development platform for DEGAS. BLCR is a system-level implementation of checkpoint/restart. It is designed so that applications can be checkpointed or restarted

- *without modifications* to the application code,
- at *arbitrary times* during their execution, and
- by confining changes required for recovery to the parallel runtime.

BLCR may be used to implement rollback-recovery from node crashes or other errors. Although BLCR is supported by all MPI implentations of interest to the DEGAS project, BLCR is not yet supported by any PGAS languages. As part of our early work on this project, we updated BLCR to run on newer Linux kernels and modern Linux distributions. These changes were released as BLCR 0.8.5. BLCR 0.8.5 fixes the following user-visible bugs and issues:

- Support for "vanilla" Linux kernels up to 3.7.x
- Fixes "vdso remap failed" errors on x86 platforms.
- Fixes support for Xeon Phi
- Fixes support for ARMv7 CPUs and for ARM THUMB2 kernels
- Fixes to work with more recent autotools, glibc, and rpmbuild.
- Fixes for memory regions larger than 4G
- Now supports "rpmbuild --define 'with_multilib 0' ..." to disable building of 32-bit libs on 64-bit targets.
- "make rpms" now works correctly when run in non-English locales
- Numerous other minor bugs

7.2.1 Analytic Models of Availability and Efficiency

As part of the design of the resilience infrastructure, and in the analysis of possible fault tolerance models, we have explored the use of probabilistic models, adopted from reliability engineering, to estimate availability and efficiency for different resilience schemes. With these models, we have been able to explore questions such as:

- How reliable does the resilience framework itself need to be?
- How many faults do applications need to handle to achieve a specific level of reliability?
- What is the cost of dropping support for forward recovery? Backwards recovery?
- How effective must predictive resilience schemes be in order to match the periodic (preventive) schemes used today?
- Is there an optimal interval for containment domains?

We have explored these questions with the use of continuous Markov chains and validated the results with discrete event simulation. Although approximate, we believe the analytic models provide valuable insight into the behavior of these resilience schemes, and intend to publish our results in an appropriate forum.

7.2.2 Resilience and Persistence

The DEGAS runtime requires two distinct forms of resilient storage, optimized for two different cases. Case 1 is a logging service designed for small, high-frequency, and low latency updates. Case 2 is a bulk service designed for large, low-frequency, high bandwidth storage. The logging service is designed to be used during normal forward execution of the program, to store information required to recover the resilience runtime itself, whereas the bulk service is designed to be used mainly during checkpoint and restart operations. A development plan for the bulk data service is in preparation. We have started implementing log-based recovery through the logging service.

7.2.3 Fault and Recovery Scenarios

We've identified what we believe will be an effective scheme to relate system-level checkpoint/restart to the application-level resilience provided by Containment Domains. Containment Domains provide *backwards recovery* by allowing applicaions to rollback to a past state and reexecute to avoid an error, but loss of the volatile program state requires either reexecution of either the outermost domain or the entire program. In contrast, Checkpoint/Restart provides *forwards recovery* from a loss of volatile state (i.e. a processor, memory, or node failure). We expect to use this model "CD moves backwards, CR moves forwards" to become the baseline for later research.

7.2.4 GASNET-EX and Resilience

As described in Section 6, work has begun to gather requirements for the resilience support that is to be added to the GASNet-EX communications library. Since GASNet-EX is required to implement our consistency model, the GASNet-EX work is a key part of our resilience strategy.

8. References

[1] Sanjay Chatterjee, Zoran Budimli, Vincent Cavé, Milind Chabbi, Max Grossman, Sarnak Tarlar, Yonghong Yan, Vivek Sarkar, "Integrating Asynchronous Task Parallelism with MPI," International Parallel and Distributed Processing Symposium (IPDPS), May 2013. To appear.

[2] James Demmel, David Eliahu, Armando Fox, Shoaib Kamil, Benjamin Lipshitz, Oded Schwartz, Omer Spillinger, "Communication-Optimal Parallel Recursive Rectangular Matrix Multiplication," International Parallel and Distributed Processing Symposium (IPDPS), May 2013. To appear.

[3] A Communication-Optimal N-Body Algorithm for Direct Interactions Michael Driscoll (UC Berkeley, USA); Evangelos Georganas (University of California, Berkeley, USA); Penporn Koanantakool (UC Berkeley, USA); Edgar Solomonik (University of California at Berkeley, USA); Katherine Yelick (University of California at Berkeley, USA), International Parallel and Distributed Processing Symposium (IPDPS), May 2013. To appear.

[4] Evangelos Georganas, Jorge González-Domínguez, Edgar Solomonik, Yili Zheng, Juan Touriño, Katherine A. Yelick, "*Communication Avoiding and Overlapping for Numerical Linear Algebra*," Proceedings of the ACM/IEEE Conference on Supercomputing (SC12), Salt Lake City, UT, November 2012.

[5] Hongzhang Shan, Brian Austin, Nicholas J. Wright, Erich Strohmaier, John Shalf and Katherine Yelick, "*Accelerating Massively Parallel Applications Using One-Sided Communication*," Proceedings of the Partitioned Global Address Space Conference, Santa Barbara, CA, October 2012.

[6] Kamesh Madduri, Jimmy Su, Samuel Williams, Leonid Oliker, Stephane Ethier, Katherine Yelick, "Optimization of Particle-to-Grid Interpolation on Leading Multicore Platforms," IEEE Transactions on Parallel and Distributed Systems, Vol. 99, ISSN 1045-9219, 2012.

[7] Michael Driscoll, Amir Kamil, Shoaib Kamil, Yili Zheng and Kathy Yelick; PyGAS: A Partitioned Global Address Space Extension for Python; Poster in the PGAS Conference, August 2012.