DEGAS:
Dynamic Global Address Space programming environments

Katherine Yelick, Principal Investigator[1]
Lawrence Berkeley National Laboratory, Berkeley, California

Vivek Sarkar, Co-Principal Investigator[2]
Rice University, Houston, Texas

James Demmel, Co-Principal Investigator[3]
University of California, Berkeley, California

Mattan Erez, Co-Principal Investigator[4]
University of Texas, Austin, Texas

Abstract: The Dynamic, Exascale Global Address Space programming environment (DEGAS) project will develop the next generation of programming models and runtime systems to meet the challenges of Exascale computing. Our approach is to provide an efficient and scalable programming model that can be adapted to application needs through the use of dynamic runtime features and domain-specific languages for computational kernels. We address the following technical challenges:

Programmability: Rich set of programming constructs based on a Hierarchical Partitioned Global Address Space (HPGAS) model, demonstrated in UPC++.
Scalability: Hierarchical locality control, lightweight communication (extended GASNet), and efficient synchronization mechanisms (Phasers).
Performance Portability: Just-in-time specialization (SEJITS) for generating hardware-specific code and scheduling libraries for domain-specific adaptive runtimes (Habanero).
Energy Efficiency: Communication-optimal code generation to optimize energy efficiency by reducing data movement.
Resilience: Containment Domains for flexible, domain-specific resilience, using state capture mechanisms and lightweight, asynchronous recovery mechanisms.
Interoperability: Runtime and language interoperability with MPI and OpenMP to encourage broad adoption.

1 DEGAS Publications

We will organize the summary of DEGAS publications according to each of our major research thrusts. Full citations appear for each in the bibligraphy.

1. Hierarchical Programming Models
2. Communication-Avoiding Compilers and Libraries
3. Adaptive Interoperable Runtime
4. Lightweight One-sided Communication
5. Resilience
6. Applications and Benchmarks

[1]PI Email: kayelick@lbl.gov; Phone: (510) 495-2431
[2]Rice-PI Email: vsarkar@rice.edu; Phone: (713) 348-5304
[3]UCB-PI Email: demmel@cs.berkeley.edu; Phone: (510) 643-5386
[4]UT-PI Email: mattan.erez@utexas.edu; Phone: (512) 471-7846

1.1 Hierarchical Programming Models

1.1.1 UPC++: A PGAS Extension for C++ [1]

We designed and implemented UPC++, a C++ template library realizing the DEGAS programming concepts and techniques for C/C++ applications. While very convenient for moving data around the system, PGAS languages have taken different views on the model of computation, with the static Single Program Multiple Data (SPMD) model providing the best scalability. In this paper we presented UPC++, a PGAS extension for C++ that has three main objectives: 1) to provide an object-oriented PGAS programming model in the context of the popular C++ language; 2) to add useful parallel programming idioms unavailable in UPC, such as asynchronous remote function invocation and multidimensional arrays, to support complex scientific applications; 3) to offer an easy on-ramp to PGAS programming through interoperability with other existing parallel programming systems (e.g., MPI, OpenMP, CUDA).

We implemented UPC++ with a compiler-free approach using C++ templates and runtime libraries. We borrowed heavily from previous PGAS languages and describe the design decisions that led to this particular set of language features, providing significantly more expressiveness than UPC with very similar performance characteristics. To demonstrate the performance of UPC++, we ported a large collection of application benchmarks to UPC++, including miniGMG, HPGMG, LULESH, MILC, NAS Parallel Benchmarks and more. We evaluated the programmability and performance of UPC++ using five benchmarks on two representative supercomputers, demonstrating that UPC++ can deliver excellent performance at large scale up to 32K cores while offering PGAS productivity features to C++ applications.

1.1.2 HabaneroUPC++: A Compiler-free PGAS Library [2]

The Partitioned Global Address Space (PGAS) programming models combine shared and distributed memory features, providing the basis for high performance and high productivity parallel programming environments. UPC++ is a very recent PGAS implementation that takes a library-based approach and avoids the complexities associated with compiler transformations. However, this implementation does not support dynamic task parallelism and only relies on other threading models (e.g., OpenMP or pthreads) for exploiting parallelism within a PGAS place.

In this paper, we introduced a compiler-free PGAS library called HabaneroUPC++, which supports a tighter integration of intra-place and inter-place parallelism than standard hybrid programming approaches. The library makes heavy use of C++11 lambda functions in its APIs. C++11 lambdas avoid the need for compiler support while still retaining the syntactic convenience of language-based approaches. The HabaneroUPC++ library implementation is based on a tight integration of the UPC++ library and the Habanero-C++ library, with new extensions to support the integration. The UPC++ library is used to provide PGAS communication and function shipping support using GASNet, and the Habanero-C++ library is used to provide support for intra-place work-stealing integrated with function shipping. We demonstrated the programmability and performance of our implementation using two benchmarks, scaled up to 6K cores. The insights developed in this paper promise to further enhance the usability and popularity of PGAS programming models.

1.1.3 Optimized Distributed Work-Stealing [3]

Work-stealing is a popular approach for dynamic load balancing of task-parallel programs. However, as has been widely studied, the use of classical work-stealing algorithms on massively parallel and distributed supercomputers introduces several performance issues. One such issue is the overhead of failed steals (communicating with a victim that has no work), which is far more severe in the distributed context than within a single SMP node. Prior work has demonstrated that load- aware victim processor selection can reduce the number of failed steals and improve the performance.

However, this too does not guarantee that the victim processor can always fulfill the thiefs request and steal attempts are still prone to failure.

In this paper, we present a distributed work-stealing algorithm, which introduces a new policy for moving work from busy to idle processors. As in past work, our approach relies on global workload information, but unlike past work, every remote steal attempt is guaranteed to eventually return work (unless the program terminates first). This strategy also avoids querying the same processor multiple times with failed steals. Our approach is applicable to any asynchronous task-based PGAS programming system implemented on a distributed system, e.g., Chapel, HabaneroUPC++, and X10. The results in this paper were obtained using HabaneroUPC++ and GASNet, and include a detailed study of our approach on dynamic irregular applications, as exemplified by the UTS and NQueens benchmarks. Our approach demonstrates low overheads and improved performance (relative to MPI and UPC versions) for up to 12288 cores on the NERSC Edison system.

1.1.4 Hierarchical Coarray Fortran [4, 5]

We produced a detailed design for the HCAF programming language, which extends the PGAS model for massively manycore computers at the exascale. Current systems already contain NUMA hierarchies that have profound consequences for communication costs in time and energy. In exascale systems, we anticipate that NUMA hierarchies will be deeper and more complex. For acceptable performance, such hierarchies must be exploited rather than ignored. We believe that today's approach of having programmers micromanage locality is unsustainable. However, past experience with compilers for HPF has taught us that completely automatic methods are also unlikely to succeed. Our design for HCAF offers an alternative to both of these approaches. HCAF provides constructs for describing and controlling hierarchical locality at a high level, enabling programmers to exploit hierarchy by writing locality-aware code somewhat painlessly. Since hardware topologies can vary greatly across machines and even jobs on one machine, there is a tension between locality awareness and performance portability. HCAF addresses this with a mechanism for specifying and using programmer-defined abstract topologies chosen for to suit application needs; the HCAF runtime would automatically map these application hierarchies to available hardware. The programmer handles strategies for exploiting hierarchal locality at a portable high level, while the language handles tactical details via static optimization and runtime adaptation.

HCAF is based on a notion of hierarchy, mappings between hierarchies, hierarchical objects and operations, and relative locality for exploiting hierarchy. A single model of hierarchy is used to describe hardware topology, execution resources, data distribution, and parallelism across all hardware levels from node interconnect down to hardware threads. HCAF hierarchies are trees with each node's children arranged in a Cartesian grid, where tree nodes denote sets of co-located resources possibly tiled into subsets. Hierarchy maps are locality-preserving correspondences between hierarchies and HCAF can automatically compute a relatively good map between any two hierarchies. The hierarchical objects are locales (hardware topologies), teams (execution resources), and coarrays (distributed data), while the hierarchical operations are mapping, subscripting, data-parallel looping, and asynchronous tasking. Relative locality generalizes the PGAS local/remote distinction by applying it to each node of a hierarchy: resources within a given hierarchy node are relatively local while other resources are relatively remote. By indicating a "current" hierarchy node the programmer chooses a level of detail at which to control locality; this currency is dynamic, supporting modular locality control across components and libraries. Data and task parallelism constructs partition computation by the current view of locality and automatically adjust that view for nested constructs within them.

1.1.5 Portable, MPI-interoperable Coarray Fortran [6]

The past decade has seen the advent of a number of parallel programming models such as Coarray Fortran (CAF), Unified Parallel C, X10, and Chapel. Despite the productivity gains promised by these models, most parallel scientific applications still rely on MPI as their data movement model. One reason for this trend is that it is hard for users to incrementally adopt these new programming models in existing MPI applications. Because each model use its own runtime system, they duplicate resources and are potentially errorprone. Such independent runtime systems were deemed necessary because MPI was considered insufficient in the past to play this role for these languages.

The recently released MPI-3, however, adds several new capabilities that now provide all of the functionality needed to act as a runtime, including a much more comprehensive one-sided communication framework. In this paper, we investigate how MPI-3 can form a runtime system for one example programming model, CAF, with a broader goal of enabling a single application to use both MPI and CAF with the highest level of interoperability.

1.1.6 Managing Asynchronous Operations in Coarray Fortran 2.0 [7]

As the gap between processor speed and network latency continues to increase, avoiding exposed communication latency is critical for high performance on modern supercomputers. One can hide communication latency by overlapping it with computation using non-blocking data transfers, or avoid exposing communication latency by moving computation to the location of data it manipulates. Coarray Fortran 2.0 (CAF 2.0)–a partitioned global address space language—provides a rich set of asynchronous operations for avoiding exposed latency including asynchronous copies, function shipping, and asynchronous collectives. CAF 2.0 provides event variables to manage completion of asynchronous operations that use explicit completion. This paper describes CAF 2.0s finish and cofence synchronization constructs, which enable one to manage implicit completion of asynchronous operations.

`finish` ensures global completion of a set of asynchronous operations across the members of a team. Because of CAF 2.0's SPMD model, its semantics and implementation of finish differ significantly from those of finish in X10 and Habanero-C. `cofence` controls local data completion of implicitlysynchronized asynchronous operations. Together these constructs provide the ability to tune a programs performance by exploiting the difference between local data completion, local operation completion, and global completion of asynchronous operations, while hiding network latency. We explore subtle interactions between cofence, finish, events, asynchronous copies and collectives, and function shipping. We justify their presence in a relaxed memory model for CAF 2.0. We demonstrate the utility of these constructs in the context of two benchmarks: Unbalanced Tree Search (UTS), and HPC Challenge RandomAccess.

We achieve 74%77% parallel efficiency for 4K32K cores for UTS using the T1WL spec, which demonstrates scalable performance using our synchronization constructs. Our cofence micro-benchmark shows that for a producer-consumer scenario, using local data completion rather than local operation completion yields superior performance.

1.1.7 A New Vision for Coarray Fortran [8]

In 1998, Numrich and Reid proposed Coarray Fortran as a simple set of extensions to Fortran 95 [8]. Their principal extension to Fortran was support for shared data known as coarrays. In 2005, the Fortran Standards Committee began exploring the addition of coarrays to Fortran 2008, which is now being finalized. Careful review of drafts of the emerging Fortran 2008 standard led us to identify several shortcom- ings with the proposed coarray extensions. In this paper, we briefly critique the coarray extensions proposed for Fortran 2008, outline a new vision for coarrays in Fortran language that is far more expressive, and briefly describe our strategy for implementing the language extensions that we propose.

1.1.8 Integrating Asynchronous Task Parallelism with MPI [9]

Effective combination of inter-node and intra-node parallelism is recognized to be a major challenge for future extreme-scale systems. Many researchers have demonstrated the potential benefits of combining both levels of parallelism, in- cluding increased communication-computation overlap, improved memory utilization, and effective use of accelerators. However, current hybrid programming approaches often require signif- icant rewrites of application code and assume a high level of programmer expertise.

Dynamic task parallelism has been widely regarded as a programming model that combines the best of performance and programmability for shared-memory programs. For distributed- memory programs, most users rely on efficient implementations of MPI. In this paper, we propose HCMPI (Habanero-C MPI), an integration of the Habanero-C dynamic task-parallel pro- gramming model with the widely used MPI message-passing interface. All MPI calls are treated as asynchronous tasks in this model, thereby enabling unified handling of messages and tasking constructs. For programmers unfamiliar with MPI, we introduce distributed data-driven futures (DDDFs), a new data-flow programming model that seamlessly integrates intra- node and inter-node data-flow parallelism without requiring any knowledge of MPI.

Our novel runtime design for HCMPI and DDDFs uses a combination of dedicated communication and computation specific worker threads. We evaluate our approach on a set of micro-benchmarks as well as larger applications and demon- strate better scalability compared to the most efficient MPI implementations, while offering a unified programming model to integrate asynchronous task parallelism with distributed-memory parallelism.

1.1.9 Hierarchical Computation in the SPMD Programming Model [10]

Large-scale parallel machines are programmed mainly with the single program, multiple data (SPMD) model of parallelism. While this model has advantages of scalability and simplicity, it does not fit well with divide-and-conquer parallelism or hierarchical machines that mix shared and distributed memory. In this paper, we defined the recursive single program, multiple data model (RSPMD) that extends SPMD with a hierarchical team mechanism to support hierarchical algorithms and machines. We implemented this model in the Titanium language and described how to eliminate a class of deadlocks by ensuring alignment of collective operations. We presented application case studies evaluating the RSPMD model, showing that it enables divide-and-conquer algorithms such as sorting to be elegantly expressed and that team collective operations increase performance of conjugate gradient by up to a factor of two. The model also facilitates optimizations for hierarchical machines, improving scalability of particle in cell by 8x and performance of sorting and a stencil code by up to 40% and 14%, respectively.

1.1.10 A Local-View Array Library for Partitioned Global Address Space C++ Programs [11]

Multidimensional arrays are an important data structure in many scientific applications. Unfortunately, built-in support for such arrays is inadequate in C++, particularly in the distributed setting where bulk communication operations are required for good performance. In this paper, we presented a multidimensional library for partitioned global address space (PGAS) programs, supporting the one-sided remote access and bulk operations of the PGAS model. The library is based on Titanium arrays, which have proven to provide good productivity and performance. These arrays provide a local view of data, where each rank constructs its own portion of a global data structure, matching the local view of execution common to PGAS programs and providing maximum flexibility in structuring global data. Unlike Titanium, which has its own compiler with array-specific analyses, optimizations, and code generation, we implemented multidimensional arrays solely through a C++ library. The main goal of this effort was to provide a library-based

implementation that can match the productivity and performance of a compiler-based approach. We implemented the array library as an extension to UPC++, a C++ library for PGAS programs, and we extended Titanium arrays with specializations to improve performance. We evaluated the array library by porting four Titanium benchmarks to UPC++, demonstrating that it can achieve up to 25% better performance than Titanium without a significant increase in programmer effort.

1.1.11 PyGAS: A Partitioned Global Address Space Extension for Python [12]

High-level, productivity-oriented languages such as Python are becoming increasingly popular in HPC applica- tions as glue and prototyping code. The PGAS model offers its own productivity advantages [6], and combining PGAS and Python is a promising approach for rapid development of parallel applications. We discuss the potential benefits and challenges of a PGAS extension to Python, and we present initial performance results from a prototype implementation called PyGAS.

1.1.12 PGAS for Distributed Numerical Python Targeting Multi-core Clusters [13]

In this paper we propose a parallel programming model that combines two well-known execution models: Single Instruction, Multiple Data (SIMD) and Single Program, Multiple Data (SPMD). The combined model supports SIMD-style data parallelism in global address space and supports SPMD-style task parallelism in local address space. One of the most important features in the combined model is that data communication is expressed by global data assignments instead of message passing. We implement this combined programming model into Python, making parallel programming with Python both highly productive and performing on distributed memory multi-core systems. We base the SIMD data parallelism on DistNumPy, an auto-parallel zing version of the Numerical Python (NumPy) package that allows sequential NumPy programs to run on distributed memory architectures. We implement the SPMD task parallelism as an extension to DistNumPy that enables each process to have direct access to the local part of a shared array. To harvest the multi-core benefits in modern processors we exploit multi-threading in both SIMD and SPMD execution models. The multi-threading is completely transparent to the user – it is implemented in the runtime with Open MP and by using multi-threaded libraries when available. We evaluate the implementation of the combined programming model with several scientific computing benchmarks using two representative multi-core distributed memory systems – an Intel Nehalem cluster with Infini band interconnects and a Cray XE-6 supercomputer – up to 1536 cores. The benchmarking results demonstrate scalable good performance.

1.2 Communication-Avoiding Compilers and Libraries

The research in compilers was divided into demonstrations of communication avoiding compiler optimizations and a set of foundational results to understand how these ideas arise in various algorithms and libraries, and what benefit, both theoretically and experimentally, are achievable.

1.2.1 Communication avoiding algorithms: Analysis and code generation for parallel systems [14]

Data movement is a critical bottleneck for future generations of parallel systems. The class of communication-avoiding .5D algorithms were developed to address this bottleneck. These algorithms reduce communication and provide strong scaling in both time and energy. As a first step towards automating the development of communication-avoiding libraries, we have developed Maunam—a prototype compiler for communication avoiding algorithms.

Maunam generates efficient parallel code from a high-level, global view sketch of .5D algorithms that are expressed using symbolic data sizes and numbers of processors. It supports the expression of data movement and communication through high-level global operations such as TILT and CSHIFT as well as through element-wise copy operations. Wrap-around communication patterns can also be achieved using subscripts based on modulo operations.

Maunam employs polyhedral analysis to reason about the communication and computation

present in the input .5D algorithm. It partitions data and computation then inserts point-to-point and collective communication as needed. Maunam also analyzes data dependence patterns and data layouts to identify reductions over processor subsets. Maunam-generated Fortran+MPI code for 2.5D matrix multiplication running on 4096 cores of a Cray XC30 supercomputer achieves 59 TFlops/s (76% of the machine peak).

1.2.2 Communication lower bounds and optimal algorithms for numerical linear algebra [15]

This invited survey article of our work appeared in *Acta Numerica*. The traditional metric for the efficiency of a numerical algorithm has been the number of arithmetic operations it performs. Technological trends have long been reducing the time to perform an arithmetic operation, so that it is no longer the bottleneck in many algorithms, rather *communication*, or moving data, is the bottleneck. This motivates us to seek algorithms that move as little data as possible, either between levels of a memory hierarchy or between parallel processors over a network. In this paper we summarize recent progress in three aspects of this problem. First we describe lower bounds on communication. Some of these generalize known lower bounds for dense classical ($O(n^3)$) matrix multiplication to all direct methods of linear algebra, to sequential and parallel algorithms, and to dense and sparse matrices. We also present lower bounds for Strassen-like algorithms, and for iterative methods, in particular Krylov subspace methods applied to sparse matrices. Second, we compare these lower bounds to widely used versions of these algorithms, and note that these widely used algorithms usually communicate asymptotically more than necessary. Third, we identify or invent new algorithms for most linear algebra problems that do attain these lower bounds, and demonstrate large speedups in theory and practice.

1.2.3 Communication Avoiding and Overlapping for Numerical Linear Algebra [16]

To efficiently scale dense linear algebra problems to future exascale systems, communication cost must be avoided or overlapped. Communication-avoiding 2.5D algorithms improve scalability by reducing inter-processor data transfer volume at the cost of extra memory usage. Communication overlap attempts to hide messaging latency by pipelining messages and overlapping with computational work. We study the interaction and compatibility of these two techniques for two matrix multiplication algorithms (Cannon and SUMMA), triangular solve, and Cholesky factorization. For each algorithm, we construct a detailed performance model which considers both critical path dependencies and idle time. We give novel implementations of 2.5D algorithms with overlap for each of these problems. Our software employs UPC, a partitioned global address space (PGAS) language that provides fast one-sided communication. We show communication avoidance and overlap provide a cumulative benefit as core counts scale, including results using over 24K cores of a Cray XE6 system.

1.2.4 Accuracy of the s-step Lanczos Method for the symmetric eigenproblem [17]

The s-step Lanczos method is an attractive alternative to the classical Lanczos method as it enables an $O(s)$ reduction in data movement over a fixed number of iterations. This can significantly improve performance on modern computers. In order for s-step methods to be widely adopted, it is important to better understand their error properties. Although the s-step Lanczos method is equivalent to the classical Lanczos method in exact arithmetic, empirical observations demonstrate that it can behave quite differently in finite precision.

In this paper, we demonstrate that bounds on accuracy for the finite precision Lanczos method given by Paige [Lin. Alg. Appl., 34:235-258, 1980] can be extended to the s-step Lanczos case assuming a bound on the condition numbers of the computed s-step bases. Our results confirm theoretically what is well-known empirically: the conditioning of the Krylov bases plays a large role

in determining finite precision behavior. In particular, if one can guarantee that the basis condition number is not too large throughout the iterations, the accuracy and convergence of eigenvalues in the s-step Lanczos method should be similar to those of classical Lanczos. This indicates that, under certain restrictions, the s-step Lanczos method can be made suitable for use in many practical cases.

1.2.5 A massively parallel tensor contraction framework for coupled-cluster computations [18]

Precise calculation of molecular electronic wavefunctions by methods such as coupled-cluster requires the computation of tensor contractions, the cost of which has polynomial computational scaling with respect to the system and basis set sizes. Each contraction may be executed via matrix multiplication on a properly ordered and structured tensor. However, data transpositions are often needed to reorder the tensors for each contraction. Writing and optimizing distributed-memory kernels for each transposition and contraction is tedious since the number of contractions scales combinatorially with the number of tensor indices. We present a distributed-memory numerical library (Cyclops Tensor Framework(CTF)) that automatically manages tensor blocking and redistribution to perform any user-specified contractions. CTF serves as the distributed-memory contraction engine in Aquarius, a new program designed for high-accuracy and massively-parallel quantum chemical computations. Aquarius implements a range of coupled-cluster and related methods such as CCSD and CCSDT by writing the equations on top of a C++ templated domain-specific language. This DSL calls CTF directly to manage the data and perform the contractions. Our CCSD and CCSDT implementations achieve high parallel scalability on the BlueGene/Q and Cray XC30 supercomputer architectures showing that accurate electronic structure calculations can be effectively carried out on top of general distributed-memory tensor primitives.

1.2.6 Communication Costs of Strassen's Matrix Multiplication [19]

This paper appeared as in invited Research Highlight in the CACM. Algorithms have historically been evaluated in terms of the number of arithmetic operations they performed. This analysis is no longer sufficient for predicting running times on todays machines. Moving data through memory hierarchies and among processors requires much more time (and energy) than performing computations. Hardware trends suggest that the relative costs of this communication will only increase. Proving lower bounds on the communication of algorithms and finding algorithms that attain these bounds are therefore fundamental goals. We show that the communication cost of an algorithm is closely related to the graph expansion properties of its corresponding computation graph.

Matrix multiplication is one of the most fundamental problems in scientific computing and in parallel computing. Applying expansion analysis to Strassens and other fast matrix multiplication algorithms, we obtain the first lower bounds on their communication costs. These bounds show that the current sequential algorithms are optimal but that previous parallel algorithms communicate more than necessary. Our new parallelization of Strassens algorithm is communication-optimal and outperforms all previous matrix multiplication algorithms.

1.2.7 An Efficient Deflation Technique for the Communication-Avoiding Conjugate Gradient Method [20]

By fusing s loop iterations, communication-avoiding formulations of Krylov subspace methods can asymptotically reduce sequential and parallel communication costs by a factor of O(s). Although a number of communication-avoiding Krylov methods have been developed, there remains a serious lack of available communication-avoiding preconditioners to accompany these methods. This

has stimulated active research in discovering which preconditioners can be made compatible with communication-avoiding Krylov methods and developing communication-avoiding methods which incorporate these preconditioners. In this paper we demonstrate, for the first time, that deflation preconditioning can be applied in communication-avoiding formulations of Lanczos-based Krylov methods such as the conjugate gradient method while maintaining an O(s) reduction in communication costs. We derive a deflated version of a communication-avoiding conjugate gradient method, which is mathematically equivalent to the deflated conjugate gradient method of Saad et al. [SIAM J. Sci. Comput., 21 (2000), pp.19091926]. Numerical experiments on a model problem demonstrate that the communication-avoiding formulations can converge at comparable rates to the classical formulations, even for large values of s. Performance modeling illustrates that O(s) speedups are possible when performance is communication bound. These results motivate deflation as a promising preconditioner for communication-avoiding Krylov subspace methods in practice.

1.2.8 A residual replacement strategy for improving the maximum attainable accuracy of s-step Krylov subspace methods [21]

Krylov subspace methods are a popular class of iterative methods for solving linear systems with large, sparse matrices. On modern computer architectures, both sequential and parallel performance of classical Krylov methods is limited by costly data movement, or communication, required to update the approximate solution in each iteration. These motivated communication-avoiding Krylov methods, based on s-step formulations, reduce data movement by a factor of $O(s)$ by reordering the computations in classical Krylov methods to exploit locality. Studies on the finite precision behavior of communication-avoiding Krylov methods in the literature have thus far been empirical in nature; in this work, we provide the first quantitative analysis of the maximum attainable accuracy of communication-avoiding Krylov subspace methods in finite precision. Following the analysis for classical Krylov methods, we derive a bound on the deviation of the true and updated residuals in communication-avoiding conjugate gradient and communication-avoiding biconjugate gradient in finite precision. Furthermore, an estimate for this bound can be iteratively updated within the method without asymptotically increasing communication or computation. Our bound enables an implicit residual replacement strategy for maintaining agreement between residuals to within $O(\epsilon)\|A\|\|x\|$. Numerical experiments on a small set of test matrices verify that, for cases where the updated residual converges, the residual replacement strategy can enable accuracy of $O(\epsilon)\|A\|\|x\|$. with a small number of residual replacement steps, reflecting improvements of up to seven orders of magnitude.

1.2.9 Contention Bounds for Combinations of Computation Graphs and Network Topologies [22]

Network topologies can have significant effect on the costs of algorithms due to inter-processor communication. Parallel algorithms that ignore network topology can suffer from contention along network links. However, for particular combinations of computations and network topologies, costly network contention may inevitably become a bottleneck, even for optimally designed algorithms. We obtain a novel contention lower bound that is a function of the network and the computation graph parameters. To this end, we compare the communication bandwidth needs of subsets of processors and the available network capacity (as opposed to per-processor analysis in most previous studies). Applying this analysis we improve communication cost lower bounds for several combinations of fundamental computations on common network topologies.

1.2.10 Tradeoffs between synchronization, communication and work in parallel linear algebra computations [23]

This work was reported last time. Since then it has appeared in SPAA'14, and been invited to appear in ACM Trans. Par. Comp. This paper derives tradeoffs between three basic costs of a parallel algorithm: synchronization, data movement, and computational cost. Our theoretical model counts the amount of work and data movement as a maximum of any execution path during the parallel computation. By considering this metric, rather than the total communication volume over the whole machine, we obtain new insight into the characteristics of parallel schedules for algorithms with nontrivial dependency structures. The tradeoffs we derive are lower bounds on the execution time of the algorithm which are independent of the number of processors, but dependent on the problem size. Therefore, these tradeoffs provide lower bounds on the parallel execution time of any algorithm computed by a system composed of any number of homogeneous components each with associated computational, communication, and synchronization payloads. We first state our results for general graphs, based on expansion parameters, then we apply the theorem to a number of specific algorithms in numerical linear algebra, namely triangular substitution, Gaussian elimination, and Krylov subspace methods. Our lower bound for LU factorization demonstrates the optimality of Tiskins LU algorithm answering an open question posed in his paper, as well as of the 2.5D LU algorithm which has analogous costs. We treat the computations in a general manner by noting that the computations share a similar dependency hypergraph structure and analyzing the communication requirements of lattice hypergraph structures.

1.2.11 A Communication-Optimal N-Body Algorithm for Direct Interactions [24]

We considered the problem of communication avoidance in computing interactions between a set of particles in scenarios with and without a cutoff radius for interaction. Our strategy, which we showed to be optimal in communication, divides the work in the iteration space rather than simply dividing the particles over processors, so more than one processor may be responsible for computing updates to a single particle. Similar to a *force decomposition* in molecular dynamics, this approach requires up to $\sqrt{p}$ times more memory than a *particle decomposition*, but reduces communication costs by factors up to $\sqrt{p}$ and is often faster in practice than a particle decomposition. We examined a generalized force decomposition algorithm that tolerates the memory limited case, i.e. when memory can only hold c copies of the particles for $c = 1, 2, ..., \sqrt{p}$. When $c = 1$, the algorithm degenerates into a particle decomposition; similarly when $c = \sqrt{p}$, the algorithm uses a force decomposition. We presented a proof that the algorithm is communication-optimal and reduces critical path latency and bandwidth costs by factors of c^2 and c, respectively. Performance results from experiments on up to 24K cores of Cray XE-6 and 32K cores of IBM BlueGene/P machines indicate that the algorithm reduces communication in practice. In some cases, it even outperformed the original force decomposition approach because the right choice of c striked a balance between the costs of collective and point-to-point communication. Finally, we extended the analysis to include a cutoff radius for direct evaluation of force interactions. We showed that with a cutoff, communication optimality still holds. We sketched a generalized algorithm for multi-dimensional space and assess its performance for 1D and 2D simulations on the same systems.

1.2.12 Computation- and Communication-Optimal Parallel Direct 3-Body Algorithms [25]

Traditional particle simulation methods are used to calculate pairwise potentials, but some problems require 3-body potentials that calculate over triplets of particles. A direct calculation of 3-body interactions involves $O(n^3)$ interactions, but has significant redundant computations that occur in a nested loop formulation. In this paper we explored algorithms for 3-body computations

that simultaneously optimize three criteria: computation minimization through symmetries, communication optimality, and load balancing. We presented a new 3-body algorithm that is both communication and computation optimal. Its optional replication factor, c, saves c^3 in latency (number of messages) and c^2 in bandwidth (volume), with bounded load-imbalance. We also considered the k-body case and discussed an algorithm that is optimal if there is a cutoff distance of less than 1/3 of the domain. The 3-body algorithm demonstrated 99% efficiency on tens of thousands of cores, showing strong scaling properties with order of magnitude speedups over the naïve algorithm.

1.2.13 Communication-Avoiding Matrix Multiplication on a Ring Architecture [26]

An additional effort has explored the development of a new communication-avoiding 1.5D matrix multiplication algorithm for ring architecture. This version has been implemented with OpenMP and tested on the Intel MIC-based Stampede architecture. The 1.5D algorithm is found to be beneficial, compared with the baseline 2.5D approach, in matrix sizes that fit in half the L2 cache (i.e. allows at least one replication), especially on Intel MIC due to its much larger aggregate L2 cache.

1.2.14 Subdivision Surface Evaluation as Sparse Matrix-Vector Multiplication [27]

Another activity is focusing on techniques for developing high-performance via PGAS-aware code generators via domain-specific languages, runtime code generation, and automatic performance tuning (autotuning). To motivate this work, we have identified a class of problems that can be cast as sparse matrix-vector products, but where the element-wise operations are generalized version of multiplication and addition. These problems include matrix-vector multiplication, matrix-matrix multiplication, path queries on graphs (shortest, most reliable, highest capacity, etc.), direct particle simulations, and dataflow analyses in compilers. Because the underlying data structure in all of these problems is a graph (or sparse matrix), the optimal implementation is data-dependent and difficult to predict; therefore, we plan to use code generation and autotuning to discover the best configuration. We are in the process of building an AST library to serve as an intermediate representation between the domain-specific language and LLVM IR, the input to the LLVM JIT compiler. To date, we have implemented support for a common subset of C-level operations, and have released our code online at https://github.com/mbdriscoll/ctree. Our longer-term goal is to enable the code generator technology to target distributed memory machines within the DEGAS software stack.

We demonstrated present an interpretation of subdivision surface evaluation in the language of linear algebra. Specifically, the vector of surface points can be computed by left-multiplying the vector of control points by a sparse subdivision matrix. This "matrix-driven" interpretation applies to any level of subdivision, holds for many common subdivision schemes (including Catmull-Clark and Loop), supports limit surface evaluation, allows semi-sharp creases, and complements feature-adaptive optimizations. It is primarily applicable to static meshes undergoing deformation (i.e. animation), in which case the subdivision matrix is invariant over time and the surface can be evaluated at each frame with a single sparse matrix-vector multiplication (SpMV). We describe techniques for building subdivision matrices on-the-fly using the recursive definition of the subdivision scheme and sparse matrix-matrix multiplication (SpMM) routines. The performance of our approach thus reduces to that of SpMV and SpMM, both of which have been studied extensively and are available in common packages for numerical linear algebra. We implemented our approach as an extension to Pixars OpenSubdiv library using routines from Intels Math Kernel Library and Nvidias CUSPARSE library to target multicore CPUs and GPUs, respectively. We present performance results from off-the-shelf routines and our own SpMV-like routines that achieve 1.7-4.8x better performance than existing techniques on both platforms.We conclude by describing

two major limitations of matrix-driven evaluation, namely difficulty computing vertex normals and complications in the presence of hierarchical edits, and suggest workarounds for both.

1.3 Adaptive Interoperable Runtime

1.3.1 Runtime systems for extreme scale platforms [28]

Future extreme-scale systems are expected to contain homogeneous and heterogeneous many-core processors, with $O(10^3)$ cores per node and $O(10^6)$ nodes overall. Effective combination of inter-node and intra-node parallelism is recognized to be a major software challenge for such systems. Further, applications will have to deal with constrained energy budgets as well as frequent faults and failures. To aid programmers manage these complexities and enhance programmability, much of recent research has focused on designing state-of-art software runtime systems. Such run- time systems are expected to be a critical component of the software ecosystem for the management of parallelism, locality, load balancing, energy and resilience on extreme-scale systems. In this dissertation, we address three key challenges faced by a runtime system using a dynamic task parallel framework for extreme-scale computing. First, we address the challenge of integrating an intra-node task parallel runtime with a communication system for scalable performance. We present a runtime communication system, called HC-COMM, designed to use dedicated communication cores on a system. We introduce the HCMPI programming model which integrates the Habanero-C asynchronous dynamic task parallel language with the MPI message passing communication model on the HC-COMM runtime. We also introduce the HAPGNS model that enables data flow programming for extreme-scale systems in which the user does not require knowledge of MPI. Second, we address the challenge of separating locality optimizations from a programmer with domain specific knowledge. We present a tuning framework, through which performance experts can optimize existing applications by specifying runtime operations aimed at co-scheduling of affinitized tasks. Finally, we address the challenge of scalable synchronization for long running tasks on a dynamic task parallel runtime. We use the phaser construct to present a generalized tree-based synchronization algorithm and support unified collective operations at both inter-node and intra-node levels. Overcoming these runtime challenges are a first step towards effective programming on extreme-scale systems.

1.3.2 Distributed Phasers [29]

A phaser is an expressive synchronization construct that unifies collective and point-to-point coordination with dynamic task paral- lelism. Each task can participate in a phaser as a signaler, a waiter, or both. The participants in a phaser may change over time as dynamic tasks are added and deleted. In this paper, we present a highly concurrent and scalable design of phasers for a distributed memory environment that is suitable for use with asynchronous partitioned global address space programming models.

Our design for a distributed phaser employs a pair of skip lists augmented with the ability to collect and propagate synchronization signals. To enable a high degree of concurrency, addition and deletion of participant tasks are performed in two phases: a "fast single-link-modify" step followed by multiple hand-over-hand "lazy multi-link-modify" steps. We also present complexity analysis of the cost of synchronization and structural operations.

Verifying highly-concurrent protocols is difficult. We analyze our design for a distributed phaser using the SPIN model checker. A straight-forward approach to model checking the operation of a distributed phaser requires an infeasibly large state space. To address this issue, we employ a novel "message-based" model checking scheme to enable a non-approximate complete model checking of our phaser design. We guarantee the semantic properties of phaser operations by ensuring that a set of linear temporal logic formulae are valid during model checking.

1.3.3 Experimental analysis of space-bounded schedulers [30]

The running time of nested parallel programs on shared memory machines depends in significant part on how well the scheduler mapping the program to the machine is optimized for the organization of caches and processors on the machine. Recent work proposed space-bounded schedulers for scheduling such programs on the multi-level cache hierarchies of current machines. The main benefit of this class of schedulers is that they provably preserve locality of the program at every level in the hierarchy, resulting (in theory) in fewer cache misses and better use of bandwidth than the popular work-stealing scheduler. On the other hand, compared to work-stealing, space-bounded schedulers are inferior at load balancing and may have greater scheduling overheads, raising the question as to the relative effectiveness of the two schedulers in practice.

In this paper, we provide the first experimental study aimed at addressing this question. To facilitate this study, we built a flexible experimental framework with separate interfaces for programs and schedulers. This enables a head-to-head comparison of the relative strengths of schedulers in terms of running times and cache miss counts across a range of benchmarks. (The framework is validated by comparisons with the Intel Cilk Plus work-stealing scheduler.) We present experimental results on a 32-core Xeon 7560 comparing work-stealing, hierarchy-minded work-stealing, and two variants of space-bounded schedulers on both divide-and-conquer micro-benchmarks and some popular algorithmic kernels. Our results indicate that space-bounded schedulers reduce the number of L3 cache misses compared to work-stealing schedulers by 2565% for most of the benchmarks, but incur up to 7% additional scheduler and load-imbalance overhead. Only for memory-intensive benchmarks can the reduction in cache misses overcome the added overhead, resulting in up to a 25% improvement in running time for synthetic benchmarks and about 20% improvement for algorithmic kernels. We also quantify runtime improvements varying the available bandwidth per core (the bandwidth gap), and show up to 50% improvements in the running times of kernels as this gap increases 4-fold. As part of our study, we generalize prior definitions of space-bounded schedulers to allow for more practical variants (while still preserving their guarantees), and explore implementation tradeoffs.

1.4 Lightweight One-sided Communication

Nearly all of the programming model and runtime publication implicitly cover the communication work as well, since this is part of the DEGAS multi-node runtime.

1.4.1 Evaluation of PGAS Communication Paradigms with Geometric Multigrid [31]

Partitioned Global Address Space (PGAS) languages and one-sided communication enable application developers to select the communication paradigm that balances the performance needs of applications with the productivity desires of programmers. In this paper, we evaluated three different one-sided communication paradigms in the context of geometric multigrid using the miniGMG benchmark. Although miniGMGs static, regular, and predictable communication does not exploit the ultimate potential of PGAS models, multigrid solvers appear in many contemporary applications and represent one of the most important communication patterns. We used UPC++, a PGAS extension of C++, as the vehicle for our evaluation, though our work is applicable to any of the existing PGAS languages and models. We compared performance with the highly tuned MPI baseline, and the results indicate that the most promising approach towards achieving performance and ease of programming is to use high-level abstractions, such as the multidimensional arrays provided by UPC++, that hide data aggregation and messaging in the runtime library.

1.4.2 Accelerating Applications at Scale Using One-Sided Communication [32]

The lower overhead associated with one-sided messaging as compared to traditional two-sided communication has the potential to increase the performance at scale of scientific applications by increasing the effective network bandwidth and reducing synchronization overheads. In this

work, we investigate both the programming effort required to modify existing MPI applications to exploit one-sided messaging using PGAS languages, and the resulting performance implications. In particular, we modify the MILC and IMPACT-T applications to use the one-sided communication features of Unified Parallel C (UPC) and coarray Fortran (CAF) languages respectively. Only modest modifications to the source code are required where fewer than 100 lines of the source code out of 70,000 need to be changed for each application. We show that performance gains of more than 50% can be achieved at scale, with the largest benefits realized at the highest concurrencies (32,768 cores).

1.4.3 A preliminary evaluation of the hardware acceleration of the Cray Gemini interconnect for PGAS languages and comparison with MPI [33]

The Gemini interconnect on the Cray XE6 platform provides for lightweight remote direct memory access (RDMA) between nodes, which is useful for implementing partitioned global address space languages like UPC and Co-Array Fortran. In this paper, we perform a study of Gemini performance using a set of communication microbenchmarks and compare the performance of one-sided communication in PGAS languages with two-sided MPI. Our results demonstrate the performance benefits of the PGAS model on Gemini hardware, showing in what circumstances and by how much one-sided communication outperforms two-sided in terms of messaging rate, aggregate bandwidth, and computation and communication overlap capability. For example, for 8-byte and 2KB messages the one-sided messaging rate is 5 and 10 times greater respectively than the two-sided one. The study also reveals important information about how to optimize one-sided Gemini communication.

1.5 Resilience

Our work on Resilience includes both the concept of containment domains for global address space programming and stack capture mechanisms, e.g., checkpointing.

1.5.1 Containment Domains Semantics version 0.2 [34]

We produced a specification for both strict and relaxed Containment Domains in the PGAS model. Correctly specifying the semantics and use of uncoordinated and distributed resilience in the PGAS context is very challenging because remote accesses to shared state occur without notification in a one-sided manner. This included finding gaps in recent published work on the topic of resilience with one-sided communication. We are currently implementing Containment Domains for UPC++. This implementation includes initial support for logging communication and runtime events, support for preservation in memory and in files, and the design of the inter-thread coordination and preservation runtime components. Our effort also contributed to the first Containment Domains API specification [35].

1.5.2 Containment Domains (API) [35]

The DEGAS team worked with others in the community, in particular at Cray, Inc., to contribute to the first public API for Containment Domains. This is an API described in C++ syntax, but that is general enough to serve as a baseline for implementation and for development of language-specific variants and extensions. The API includes PGAS-specific concepts that we are specifically exploring as a means to provide hints and directives for optimizing the execution of relaxed CDs.

1.5.3 Containment Domains: A Scalable, Efficient, and Flexible Resilience Scheme for Exascale Systems [36]

This paper describes and evaluates a scalable and efficient resilience scheme based on the concept of containment domains. Containment domains are a programming construct that enable applications to express resilience needs and to interact with the system to tune and specialize error detection, state preservation and restoration, and recovery schemes. Containment domains have weak transactional semantics and are nested to take advantage of the machine and application

hierarchies and to enable hierarchical state preservation, restoration, and recov- ery. We evaluate the scalability and efficiency of containment domains using generalized trace-driven simulation and analytical analysis and show that containment domains are superior to both checkpoint restart and redundant execution approaches. Another version of this work also appeared in the Scientific Computing journal [37].

1.5.4 Affinity-Aware Checkpoint Restart [38]

Current checkpointing techniques employed to overcome faults for HPC applications result in inferior application performance after restart from a checkpoint for a number of applications. This is due to a lack of page and core affinity awareness of the checkpoint/restart (C/R) mechanism, i.e., application tasks originally pinned to cores may be restarted on dierent cores, and in case of non-uniform memory architectures (NUMA), quite common today, memory pages associated with tasks on a NUMA node may be associated with a dierent NUMA node after restart. This work contributes a novel design technique for C/R mechanisms to preserve task-to-core maps and NUMA node specic page anities across restarts. Experimental results with BLCR, a C/R mechanism, enhanced with affinity awareness demonstrate signicant performance benets of 37%-73% for the NAS Parallel Benchmark codes and 6-12% for NAMD with negligible overheads instead of up to nearly four times longer an execution times without affinnity-aware restarts on 16 cores.

1.5.5 Affinity-Aware Checkpoint Restart (Master's Thesis) [39]

This work (described in the previous abstract) was also presented in the Master's Thesis by Ajay Saini at North Carolina State University under the supervision of Frank Mueller.

1.6 Applications and Benchmarks

1.6.1 Parallel De Bruijn Graph Construction and Traversal for De Novo Genome Assembly [40]

De novo whole genome assembly reconstructs genomic sequence from short, overlapping, and potentially erroneous fragments called reads. We study optimized parallelization of the most time-consuming phases of Meraculous, a state-of-the-art production assembler. First, we present a new parallel algorithm for k-mer analysis, characterized by intensive communication and I/O requirements, and reduce the memory requirements by 6.93. Second, we efficiently parallelize de Bruijn graph construction and traversal, which necessitates a distributed hash table and is a key component of most de novo assemblers. We provide a novel algorithm that leverages one-sided communication capabilities of the Unified Parallel C (UPC) to facilitate the requisite fine-grained parallelism and avoidance of data hazards, while analytically proving its scalability properties. Overall results show unprecedented performance and efficient scaling on up to 15,360 cores of a Cray XC30, on human genome as well as the challenging wheat genome, with performance improvement from days to seconds.

1.6.2 merAligner: A Fully Parallel Sequence Aligner [41]

Aligning a set of query sequences to a set of target sequences is an important task in bioinformatics. In this work we present merAligner, a highly parallel sequence aligner that implements a seedandextend algorithm and employs parallelism in all of its components. MerAligner relies on a high performance distributed hash table (seed index) and uses one-sided communication capabilities of the Unified Parallel C to facilitate a fine-grained parallelism. We leverage communication optimizations at the construction of the distributed hash table and software caching schemes to reduce communication during the aligning phase. Additionally, merAligner preprocesses the target sequences to extract properties enabling exact sequence matching with minimal communication. Finally, we efficiently parallelize the I/O intensive phases and implement an effective load balancing scheme. Results show that merAligner exhibits efficient scaling up to thousands of cores on a Cray XC30 supercomputer using real human and wheat genome data while significantly outperforming

existing parallel alignment tools.

1.6.3 A whole-genome shotgun approach for assembling and anchoring the hexaploid bread wheat genome [42]

Polyploid species have long been thought to be recalcitrant to whole-genome assembly. By combining high-throughput sequencing, recent developments in parallel computing, and genetic mapping, we derive, de novo, a sequence assembly representing 9.1 Gbp of the highly repetitive 16 Gbp genome of hexaploid wheat, Triticum aestivum, and assign 7.1 Gb of this assembly to chromosomal locations. The genome representation and accuracy of our assembly is comparable or even exceeds that of a chromosome-by-chromosome shotgun assembly. Our assembly and mapping strategy uses only short read sequencing technology and is applicable to any species where it is possible to construct a mapping population.

1.6.4 HipMer: An Extreme-scale De Novo Genome Assembler [43]

De novo whole genome assembly reconstructs genomic sequences from short, overlapping, and potentially erroneous DNA segments and is one of the most important computations in modern genomics. This work presents HipMer, the first high-quality end-to-end de novo assembler designed for extreme scale analysis, via efficient parallelization of the Meraculous code. First, we significantly improve scalability of parallel k-mer analysis for complex repetitive genomes that exhibit skewed frequency distributions. Next, we optimize the traversal of the de Bruijn graph of k-mers by employing a novel communication-avoiding parallel algorithm in a variety of use-case scenarios. Finally, we parallelize the Meraculous scaffolding modules by leveraging the one-sided communication capabilities of the Unified Parallel C while effectively mitigating load imbalance. Large-scale results on a Cray XC30 using grand-challenge genomes demonstrate efficient performance and scalability on thousands of cores. Overall, our pipeline accelerates Meraculous performance by orders of magnitude, enabling the complete assembly of the human genome in just 8.4 minutes on 15K cores of the Cray XC30, and creating unprecedented capability for extreme-scale genomic analysis.

1.6.5 Parallel Hessian Assembly for Seismic Waveform Inversion Using Global Updates [44]

UPC++ has already enabled new science results by allowing application scientists (French and Romanwicz at UC Berkeley) to tackle a previously intractable seismic imaging problem. The solution was made feasible by the specific combination of PGAS features provided by UPC++, particularly remote memory management and asynchronous task execution, while still presenting a familiar language (C++) and allowing interoperation with MPI/OpenMP components of the original code. This application incorporated observational data into a simulation to improve both the quality and speed of the simulation. The performance, shown on the right side in figure demonstrated 90% parallel efficiency on 12,000 cores. With UPC++, the code can solve problems that were previously impossible due to memory size constraints, enabling scientists to extend their earlier upper-mantle tomographic imaging work to compute the first ever whole-mantle global tomographic model using numerical seismic wavefield computations. Following the UPC++ work, the code was also written in MPI-3's one-sided model for a reference implementation, whose current performance lags the UPC++ version on the Edison system by up to 6x at larger scales.

1.6.6 Optimization of geometric multigrid for emerging multi- and manycore processors [45]

Multigrid methods are widely used to accelerate the convergence of iterative solvers for linear systems used in a number of different application areas. In this paper, we explore optimization techniques for geometric multigrid on existing and emerging multicore systems including the Opteron- based Cray XE6, Intel^R Xeon^R E5-2670 and X5550 processor-based Infiniband clusters, as well as the new Intel^R Xeon Phi^{TM} coprocessor (Knights Corner). Our work examines a va-

riety of novel techniques including communication-aggregation, threaded wavefront-based DRAM communication-avoiding, dynamic threading decisions, SIMDization, and fusion of operators. We quantify performance through each phase of the V-cycle for both single-node and distributed-memory experiments and provide detailed analysis for each class of optimization. Results show our optimizations yield significant speedups across a variety of subdomain sizes while simultaneously demonstrating the potential of multi- and manycore processors to dramatically accelerate single-node performance. However, our analysis also indicates that improvements in networks and communication will be essential to reap the potential of manycore processors in large-scale multigrid calculations. (This work was primarily funded by the ExaCT Co-Design Center, but with some involvment of DEGAS-funded researchers.)

1.6.7 Optimization of parallel particle-to-grid interpolation on leading multicore platforms [46]

We are now in the multicore revolution which is witnessing a rapid evolution of architectural designs due to power constraints and correspondingly limited microprocessor clock speeds. Understanding how to efficiently utilize these systems in the context of demanding numerical algorithms is an urgent challenge to meet the ever growing computational needs of high-end computing. In this work, we examine multicore parallel optimization of the particle-to-grid interpolation step in particle-mesh methods, an inherently complex optimization problem due to its low computation intensity, irregular data accesses, and potential fine-grained data hazards. Our evaluated kernels are derived from two important numerical computations: a biological simulation of the heart using the Immersed Boundary (IB) method, and a Gyrokinetic Particle-in-Cell (PIC)-based application for studying fusion plasma microturbulence. We develop several novel synchronization and grid decomposition schemes, as well as low-level optimization techniques to maximize performance on three modern multicore platforms: Intel's Xeon X5550 (Nehalem), AMD's Opteron 2356 (Barcelona), and Sun's UltraSparc T2+ (Niagara). Results show that our optimizations lead to significant performance improvements, achieving up to a 5.6 speedup compared to the reference parallel implementation. Our work also provides valuable insight into the design of future autotuning frameworks for particle-to-grid interpolation on next-generation systems.

1.7 Implementing High-Performance Geometric Multigrid Solver With Naturally-GrainedMessages [47]

Structured grid linear solvers often require manually packing and unpacking of communication data to achieve high performance. Orchestrating this process efficiently is challenging, labor-intensive, and potentially error-prone. In this paper, we explore an alternative approach that communicates the data with naturally grained message sizes without manual packing and unpacking. This approach is the distributed analogue of shared-memory programming, taking advantage of the global address space in PGAS languages to provide substantial programming ease. However, its performance may suffer from the large number of small messages. We investigate the runtime support required in the UPC++ library for this naturally grained version to close the performance gap between the two approaches and attain comparable performance at scale using the High-Performance Geometric Multigrid (HPGMG-FV) benchmark as a driver.

2 Software Products and Web Sites

- We released the UPC++ programming system as Open Source Software with a BSD license. https://bitbucket.org/upcxx
- We released the UPC++ miniGMG implementation in 2014 as Open Source Software under a BSD license. https://bitbucket.org/upcxx/minigmg/downloads
- We released UPC++ HPGMG-FV implementation in 2015 as Open Source Software under a

BSD license.
https://bitbucket.org/upcxx/hpgmg/downloads

- We released the Habanero-UPC++ PGAS library.
  http://habanero-rice.github.io/habanero-upc
- We released the GASNet PGAS communications library in November 2014 (and will again in April 2015) as Open-Source Software with a BSD license. These releases include prototype implementations of backward-compatible features from GASNet-EX.
  http://gasnet.lbl.gov
- We released Berkeley Lab Checkpoint/Restart (BLCR), in January 2013 as Open-Source Software with a GPL license. We added support for additional Linux kernels in development releases during late 2014.
  http://ftg.lbl.gov/checkpoint
- We released the CTree implementation of the Selected Embedded Just-in-Time Specializations (SEJITS), via github:
  https://github.com/mbdriscoll/ctree
- We released an API specification for Containment Domains, including features specifically for a global address space currently developing a UPC++ instantiation of this API.
  http://lph.ece.utexas.edu/users/CDAPI
- We have developed an HPGAS version of most components of the Meraculous genome assembly pipeline. This has been shared and used by our JGI collaborators, and will be release later once the full pipeline is complete and optimized.
- We developed a CAF2.0 translation of CGPOP—a mini-applicaiton that represents the conjugate gradient solver from LANL's POP 2.0. http://svn.rice.edu/r/caf/caf-compiler/tests/benchmarks-forthcoming/cgpop-caf2/caf2D
- We developed a CAF2.0 translation of Sandia's miniGhost—a finite difference mini-application that implements a difference stencil across a homogenous three dimensional domain. http://svn.rice.edu/r/caf/caf-compiler/tests/benchmarks-forthcoming/minighost/miniGhost-mgr-caf2

2.1 Tutorials and Outreach

1. K. Mohror, N. DeBardeleben, E. Roman, and L. Kale. Practical Fault Tolerance on Today's HPC Systems. In *SC '13*. International Conference for High Performance Computing, Networking, Storage and Analysis. Denver CO. November 17, 2013. Tutorial.
2. K. Mohror, N. DeBardeleben, E. Roman, and L. Kale. Practical Fault Tolerance on Today's Supercomputing Systems. In *SC '14*. International Conference for High Performance Computing, Networking, Storage and Analysis. New Orleans, LA. November 16, 2014. Tutorial.
3. C. Iancu, K. Yelick, and Y. Zheng. Advanced PGAS Programming in UPC. In *SC '13*. International Conference for High Performance Computing, Networking, Storage and Analysis. Denver CO. November, 2013. Tutorial.
4. Y. Zheng. PGAS Programming in UPC and UPC++. In *PGAS '14*. International Conference on Partitioned Global Address Space Programming Models. Eugene, Oregon. October, 2014. Tutorial.
5. K. Yelick, and Y. Zheng. Developing Parallel C++ Applications with Modern PGAS Features in UPC++. In *PGAS '15*. International Conference on Partitioned Global Address Space Programming Models. Washington D.C.. October, 2015. Tutorial.

3 Bibliography of DEGAS Publications

References

[1] Y. Zheng, A. Kamil, M. Driscoll, H. Shan, and K. Yelick, "UPC++: A PGAS Extension for C++," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2014.

[2] V. Kumar, Y. Zheng, V. Cavé, Z. Budimlić, and V. Sarkar, "HabaneroUPC++: A compiler-free PGAS library," in *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, PGAS '14, (New York, NY, USA), pp. 5:1–5:10, ACM, 2014.

[3] V. Kumar, V. Sarkar, and Y. Zheng, "Optimized distributed work-stealing," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IPDPS '16, under review.

[4] S. K. Warren, "Hierarchical Locality in HCAF (Hierarchical Coarray Fortran)." http://caf.rice.edu/publications/HCAF-Hierarchical-Locality.pdf, August 13, 2014.

[5] S. K. Warren, "Introduction to HCAF: Hierarchical Coarray Fortran." http://caf.rice.edu/publications/HCAF-Introduction.pdf, 2014.

[6] C. Yang, W. Bland, J. Mellor-Crummey, and P. Balaji, "Portable, MPI-interoperable Coarray Fortran," in *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '14, (New York, NY, USA), pp. 81–92, ACM, 2014.

[7] C. Yang, K. Murthy, and J. Mellor-Crummey, "Managing asynchronous operations in Coarray Fortran 2.0," in *Proceedings of the 27th International Symposium on Parallel Distributed Processing (IPDPS)*, pp. 1321–1332, May 2013.

[8] J. Mellor-Crummey, L. Adhianto, W. N. Scherer III, and G. Jin, "A new vision for coarray fortran," in *Proceedings of the Third Conference on Partitioned Global Address Space Programing Models*, p. 5, ACM, 2009.

[9] S. Chatterjee, S. Tasirlar, Z. Budimlić, V. Cavé, M. Chabbi, M. Grossman, Y. Yan, and V. Sarkar, "Integrating Asynchronous Task Parallelism with MPI," in *IPDPS '13: Proceedings of the 2013 IEEE International Symposium on Parallel&Distributed Processing*, IEEE Computer Society, 2013.

[10] A. Kamil and K. Yelick, "Hierarchical computation in the SPMD programming model," in *The 26th International Workshop on Languages and Compilers for Parallel Computing.*, 2013.

[11] A. Kamil, Y. Zheng, and K. Yelick, "A Local-View Array Library for Partitioned Global Address Space C++ Programs," in *Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ARRAY'14, (New York, NY, USA), pp. 26:26–26:31, ACM, 2014.

[12] M. Driscoll, A. Kamil, S. Kamil, Y. Zheng, and K. Yelick, "Pygas: A partitioned global address space extension for python," in *Poster in the PGAS Conference*, Citeseer, 2012.

[13] M. R. B. Kristensen, Y. Zheng, and B. Vinter, "Pgas for distributed numerical python targeting multi-core clusters," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 680–690, IEEE, 2012.

[14] K. Murthy and J. Mellor-Crummey, "Communication avoiding algorithms: Analysis and code generation for parallel systems," in *Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques*, IEEE, 2015.

[15] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz, "Communication lower bounds and optimal algorithms for numerical linear algebra," *Acta Numerica*, vol. 23, pp. 1–155, 2014.

[16] E. Georganas, J. González-Domínguez, E. Solomonik, Y. Zheng, J. Tourino, and K. Yelick, "Communication avoiding and overlapping for numerical linear algebra," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, p. 100, IEEE Computer Society Press, 2012.

[17] E. Carson and J. Demmel, "Accuracy of the s-step Lanczos method for the symmetric eigenproblem," tech. rep., Technical Report UCB/EECS-2014-165, EECS Department, University of California, Berkeley, 2014. To appear in SIAM J. Mat. Anal. Appl.

[18] E. Solomonik, D. Matthews, J. R. Hammond, J. F. Stanton, and J. Demmel, "A massively parallel tensor contraction framework for coupled-cluster computations," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3176–3190, 2014.

[19] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Communication costs of strassen's matrix multiplication," *Communications of the ACM*, vol. 57, no. 2, pp. 107–114, 2014.

[20] E. Carson, N. Knight, and J. Demmel, "An efficient deflation technique for the communication-avoiding conjugate gradient method," *Electronic Transactions on Numerical Analysis*, vol. 43, pp. 125–141, 2014.

[21] E. Carson and J. Demmel, "A residual replacement strategy for improving the maximum attainable accuracy of s-step Krylov subspace methods," *SIAM Journal on Matrix Analysis and Applications*, vol. 35, no. 1, pp. 22–43, 2014.

[22] G. Ballard, J. Demmel, A. Gearhart, B. Lipshitz, O. Schwartz, and S. Toledo, "Contention bounds for combinations of computation graphs and network topologies," in *CSC14: The Sixth SIAM Workshop on Combinatorial Scientific Computing*, p. 30, 2014.

[23] E. Solomonik, E. Carson, N. Knight, and J. Demmel, "Tradeoffs between synchronization, communication, and computation in parallel linear algebra computations," in *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pp. 307–318, ACM, 2014.

[24] P. Koanantakool and K. Yelick, "A computation-and communication-optimal parallel direct 3-body algorithm," in *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pp. 363–374, IEEE, 2014.

[25] M. Driscoll, E. Georganas, P. Koanantakool, E. Solomonik, and K. Yelick, "A communication-optimal n-body algorithm for direct interactions," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pp. 1075–1084, IEEE, 2013.

[26] E. Georganas, P. Koanantakool, E. Solomonik, S. Williams, K. Yelick, and Y. Zheng, "Communication-avoiding matrix multiplication on ring architectures." unpublished manuscript.

[27] M. Driscoll, "Subdivision surface evaluation as sparse matrix-vector multiplication," Master's thesis, EECS Department, University of California, Berkeley, Dec 2014.

[28] S. Chatterjee, *Runtime systems for extreme scale platforms*. Ph.D. dissertation, Rice University, Houston, 2013.

[29] K. Murthy, S. R. Paul, K. S. Meel, and J. Mellor-Crummey, "Distributed phasers." Submitted to International Conference on Parallel Architectures and Compilation Techniques, 2015, Under review.

[30] H. V. Simhadri, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, and A. Kyrola, "Experimental analysis of space-bounded schedulers," in *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pp. 30–41, ACM, 2014.

[31] H. Shan, A. Kamil, S. Williams, Y. Zheng, and K. Yelick, "Evaluation of PGAS Communication Paradigms with Geometric Multigrid," in *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, PGAS '14, (New York, NY, USA), pp. 8:1–8:12, ACM, 2014.

[32] H. Shan, B. Austin, N. J. Wright, E. Strohmaier, J. Shalf, and K. Yelick, "Accelerating applications at scale using one-sided communication," in *Proceedings of the Conference on Partitioned Global Address Space Programming Models (PGAS12)*, 2012.

[33] H. Shan, N. J. Wright, J. Shalf, K. Yelick, M. Wagner, and N. Wichmann, "A preliminary evaluation of the hardware acceleration of the cray gemini interconnect for pgas languages and comparison with mpi," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 2, pp. 92–98, 2012.

[34] M. Sullivan, I. Lee, J. Chung, S. Zhang, S.-L. Gong, D. Liu, M. LeBeane, K. Lee, and M. Erez, "Containment domains semantics version 0.2," Tech. Rep. Tr-LPH-2014-001, LPH Group, Department of Electrical and Computer Engineering, The University of Texas at Austin, February 2014.

[35] Containment Domains Team, "Containment domains C++ API rev 0.1." lph.ece.utexas.edu/users/CDAPI, 2014.

[36] J. Chung, I. Lee, M. Sullivan, J. H. Ryoo, D. W. Kim, D. H. Yoon, L. Kaplan, and M. Erez, "Containment domains: A scalable, efficient, and flexible resilience scheme for exascale systems," in *the Proceedings of SC'12*, November 2012.

[37] J. Chung, I. Lee, M. Sullivan, J. H. Ryoo, D. W. Kim, D. H. Yoon, L. Kaplan, and M. Erez, "Containment domains: A scalable, efficient, and flexible resilience scheme for exascale systems," *Scientific Programming*, vol. 21, pp. 197 – 212, January 2013.

[38] A. Saini, A. Rezaei, F. Mueller, P. Hargrove, and E. Roman, "Affinity-aware checkpoint restart," in *Middleware*, Dec. 2014.

[39] A. Saini, "Affinity-aware checkpoint restart." Master's Thesis, North Carolina State University, 2014.

[40] E. Georganas, A. Buluç, J. Chapman, L. Oliker, D. Rokhsar, and K. Yelick, "Parallel de bruijn graph construction and traversal for de novo genome assembly," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, (Piscataway, NJ, USA), pp. 437–448, IEEE Press, 2014.

[41] E. Georganas, A. Buluç, J. Chapman, L. Oliker, D. Rokhsar, and K. Yelick, "merAligner: A Fully Parallel Sequence Aligner," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2015.

[42] J. A. Chapman, M. Mascher, K. Barry, E. Georganas, A. Session, V. Strnadova, J. Jenkins, S. Sehgal, L. Oliker, J. Schmutz, K. Yelick, U. Scholz, R. Waugh, J. Poland, G. Muehlbaur, N. Stein, and D. S. Rokhsar, "A whole-genome shotgun approach for assembling and anchoring the hexaploid bread wheat genome," *Genome biology*, vol. 16, no. 1, p. 26, 2015.

[43] E. Georganas, A. Buluç, J. Chapman, S. Hofmeyr, C. Aluru, R. Egan, L. Oliker, D. Rokhsar, and K. Yelick, "Hipmer: An extreme-scale de novo genome assembler," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, (New York, NY, USA), pp. 14:1–14:11, ACM, 2015.

[44] S. French, Y. Zheng, B. Romanowicz, and K. Yelick, "Parallel Hessian Assembly for Seismic Waveform Inversion Using Global Updates ," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2015.

[45] S. Williams, D. D. Kalamkar, A. Singh, A. M. Deshpande, B. Van Straalen, M. Smelyanskiy, A. Almgren, P. Dubey, J. Shalf, and L. Oliker, "Optimization of geometric multigrid for emerging multi- and manycore processors," in *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, IEEE Computer Society Press, 2012.

[46] K. Madduri, J. Su, S. Williams, L. Oliker, S. Ethier, and K. Yelick, "Optimization of parallel particle-to-grid interpolation on leading multicore platforms," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 10, pp. 1915–1922, 2012.

[47] H. Shan, S. Williams, Y. Zheng, A. Kamil, and K. Yelick, "Implementing High-Performance Geometric Multigrid Solver With Naturally-Grained Messages," in *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, PGAS '15, 2015.