Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap

Christian Bell[1,2], Dan Bonachea[1], Rajesh Nishtala[1], Katherine Yelick[1,2]

[1]University of California, Berkeley
Computer Science Division
Berkeley, CA 94720 USA

[2]Lawrence Berkeley National Laboratory
Computational Research Division
Berkeley, CA 94720 USA

{csbell,bonachea,rajeshn,yelick}@cs.berkeley.edu

Abstract

This paper demonstrates the one-sided communication used in languages like UPC can provide a significant performance advantage for bandwidth-limited applications. This is shown through communication microbenchmarks and a case-study of UPC and MPI implementations of the NAS FT benchmark. Our optimizations rely on aggressively overlapping communication with computation, alleviating bottlenecks that typically occur when communication is isolated in a single phase. The new algorithms send more and smaller messages, yet the one-sided versions achieve $> 1.9\times$ *speedup over the base Fortran/MPI. Our one-sided versions show an average* 15% *improvement over the two-sided versions, due to the lower software overhead of one-sided communication, whose semantics are fundamentally lighter-weight than message passing. Our UPC results use Berkeley UPC with GASNet and demonstrate the scalability of that system, with performance approaching 0.5 TFlop/s on the FT benchmark with 512 processors.*

1. Introduction

The one-sided communication model is typically viewed as advantageous for unstructured computations and irregular communication patterns for both performance and programmability [11]. One-sided communication is the primary mode of communication in Partitioned Global Address Space (PGAS) languages and has been integrated into the 2.0 revision of the Message-Passing Interface (MPI). Although these instantiations of the one-sided model differ semantically and operationally in the mechanisms used to enforce synchronization, both aim to improve performance by decoupling synchronization from data movement. The benefits of the one-sided model are most pronounced for small message data transfers where the synchronization and software overhead is not amortized by transfer time, however we argue that one-sided communication can also be beneficial in improving the performance of applications that are bandwidth-bound. In particular, we show that replacing large bulk transfers with more frequent smaller messages allows a bandwidth-bound application written using UPC's one-sided model to outperform versions written with MPI two-sided message-passing.

Conventional wisdom holds that communication costs should be minimized by sending a small number of large messages, especially for cluster networks where the per-message cost can be high. This practice is motivated by the observation that large messages have traditionally been required to achieve peak communication bandwidth. Many applications therefore adopt a bulk-synchronous communication paradigm, dividing program execution into mutually exclusive global phases of computation and communication. Two recent trends are challenging that wisdom: network vendors are increasingly offloading communication protocol processing onto network hardware; and the emergence of one-sided communication offers unique opportunities to further reduce overhead by decoupling synchronization from data transfer. In many applications the computational data dependencies could actually permit a large fraction of the communication to be initiated earlier or completed later. If one can find sufficient independent computation to overlap the communication latency such that negligible time is spent waiting for communication completion, then the primary cost of communication becomes the software overhead required to initiate and synchronize non-blocking communication. Once the latency component of communication has been entirely overlapped in this manner, the transfer bandwidth achieved by an individual message becomes less important, making it feasible to trade off smaller message size for improved overlap.

This paper explores the hypothesis that communication can be effectively overlapped in bandwidth-sensitive applications using one-sided communication. We use a series of communication microbenchmarks and the NAS Fourier Transform (FT) benchmark as a case study to compare one-sided and two-sided communication models in order to validate this hypothesis. Our approach is to spread the communication of the all-to-all transpose step required by the 3D FFT throughout the computation of the local slabs, sending data as soon as it becomes available. This optimization alleviates bottlenecks in the communication and aggressively overlaps it behind the computation. Although the total volume of data communicated is constant, the number of messages per thread increases from $O(T)$ in the all-to-all version to $O(n)$ where T is the total number of threads and n is the size of the maximum dimension. The promising results motivate an even more aggressive overlap strategy that sends $O(\frac{n^2}{T})$ messages while still keeping the total volume of data constant. As the results will show, the two-sided versions that used the $O(n)$ algorithm achieve nontrivial gains (over $1.75\times$) compared to the traditional all-to-all version. However the versions which utilize one-sided communication achieve an additional speedup using the algorithm that sends $O(\frac{n^2}{T})$ messages. These algorithms are consistently the best performers with speedups of up to $1.9\times$ over the traditional all-to-all versions. We also implement the $O(\frac{n^2}{T})$ version in MPI, but show it cannot achieve the same performance benefits as UPC due to the higher communication overhead in MPI. We argue that this added overhead is at least partly fundamental to the two-sided model.

We use Berkeley UPC [7] and MPI v1.1 [29] as representatives of the one and two-sided communication models, respectively. Although the MPI 2.0 standard [30] adds a one-sided communication interface, this interface has several semantic limitations that hinder its use in practice [10], and therefore we do not consider it further in this paper. Instead, we use use the Berkeley UPC implementation with GASNet [8] as our representative for one-sided communication. UPC [36], along with Co-Array Fortran [32] and Titanium [25], are modern examples of the Partitioned Global Address Space (PGAS) language approach to parallel computing. They expose language semantics that induce a one-sided communication model: processors logically issue direct loads and stores to the memory of remote processors using reads and writes to logically shared variables. Our one-sided implementations of the benchmark are written from scratch in UPC, and leverage some minor library extensions to UPC for non-blocking bulk memory operations provided by the Berkeley UPC compiler. Our two-sided versions are written in Fortran and C with MPI v1.1, starting with the standard NAS release of the FT benchmark.

The remainder of this paper is organized as follows: section 2 gives a brief introduction to PGAS languages and UPC. Sections 3 and 4 present the GASNet communications layer and show its bandwidth and latency performance compared to MPI. Section 5 details how we leverage one-sided communication to obtain significant performance improvements over MPI in the NAS FT.

2. PGAS Languages and UPC

Partitioned Global Address Space languages combine a Single Program Multiple Data (SPMD) programming model with a global address space, which is logically partitioned to give each thread a portion of shared memory to which it has affinity. The study in this paper is based on Unified Parallel C (UPC), although the observations on communication techniques are more broadly applicable to the entire family of PGAS languages and other parallel systems providing one-sided communication.

There are many commercial and open-source compilers available for UPC [35]. In this paper we used the portable, high-performance Berkeley UPC compiler [7]. On a shared memory machine, accesses to the UPC shared address space translate into conventional load/store instructions. On distributed memory machines (which are of interest in this paper) such accesses translate into calls to the Berkeley GASNet layer [23]. Some of the application-level optimizations presented in this paper make use of Berkeley-specific extensions to the UPC language [9]. The results in this paper demonstrate their benefits and motivate their likely inclusion in the next language revision.

3. GASNet Communication System

GASNet provides a portable, language-independent communication interface designed as a compilation target for PGAS languages. GASNet delivers communication performance very close to the raw hardware peak across many interconnects, effectively leveraging platform and network-specific features such as RDMA support and block transfer engines. The GASNet API provides point-to-point data transfers that are fully one-sided and decoupled from inter-thread synchronization, with no relative ordering constraints between outstanding operations (in contrast to other one-sided communication interfaces such as ARMCI [31]). GASNet's point-to-point communication API includes simple blocking gets/puts, and several flavors of non-blocking data transfers with a flexible and expressive set of synchronization primitives crafted to support sophisticated communication optimizations. The GASNet implementation is designed in layers for portability: a small set of core functions constitute the basis for portability, and there is a reference implementation of the complete API written entirely in terms of this core. In addition, the implementation for a

given network (the *conduit*) can be tuned by implementing any appropriate subset of the general functionality directly upon the hardware-specific primitives, bypassing the core-based reference implementation. Our research has shown that the layered design approach is effective at providing robust portability as well as high-performance, with UPC performance comparable to vendor-provided compilers on architectures ranging from loosely-coupled clusters with a near-commodity network [13] to tightly-coupled MPP systems with a hardware-supported global memory system [6].

The GASNet interface has been natively implemented on Myrinet (GM), Quadrics QsNetI/QsNetII (Elan3/4), InfiniBand (Mellanox VAPI), IBM SP Colony/Federation (LAPI), Dolphin (SISCI), Cray X1 (shmem) and SGI Altix (shmem). Aside from these high-performance instantiations of the GASNet interface, there are also fully portable GASNet conduits over MPI 1.1 (for any MPI-enabled HPC system not natively supported), GASNet on UDP (for any TCP/IP network, eg. Ethernet), and GASNet for shared-memory SMP's lacking interconnect hardware. Our GASNet implementation is written in standard C and is very portable across architectures and operating systems – thus far it has been successfully used on over fifteen different CPU architectures, twelve different operating systems, and twelve different C compilers, and porting existing GASNet conduits to new UNIX-like systems is nearly effortless. See [23] for further implementation details.

4. Performance Advantages of One-Sided Communication in Microbenchmarks

Partitioned Global Address Space languages are sometimes considered suitable only for shared memory hardware such as Symmetric Multiprocessors (SMPs), Distributed Shared Memory machines (e.g., the SGI Altix), or machines with global address space support integrated into the processor (e.g., the Cray X1). However in this paper we demonstrate that the one-sided communication model underlying these languages is also a more effective match to modern cluster network hardware than two-sided message passing interfaces such as MPI.

The disadvantages of the MPI two-sided message-passing model are summarized in the following points:

- Message sends and receives must be matched to complete a transfer. The implementation is responsible for matching the MPI communicator, message tag and sender id between the sender and receiver, and the overhead of this matching can impose a significant penalty for small to medium sized messages.
- MPI guarantees point-to-point message ordering, despite the fact that many current and future networks lack such ordering guarantees in hardware. Studies [27, 28, 38] have shown there is a non-trivial cost associated with enforcing ordering semantics upon fundamentally unordered network hardware.
- The semantic requirement for active participation from application-level code on both sides of the communication implies that observed latency in an MPI application may be significantly longer than predicted by a best case scenario - i.e., an application that is inattentive to the network may perform poorly even on a system with low best-case MPI latency.

High-quality MPI implementations on cluster hardware generally use a combination of algorithms to provide the required message-matching semantics and also provide good performance over a large range of message sizes. The *eager* algorithm (which is generally used for small messages) optimistically sends the data and messaging metadata to an anonymous buffer on the target process, which later performs message matching and copies the data to the user buffer. This approach minimizes the wire-time latency, but imposes CPU and memory bus overheads for the extraneous data copy operation and hence is unsuitable for sufficiently large messages where the copy costs would dominate. The *rendezvous* algorithm (generally used for larger messages) initially sends only the metadata to the remote process, which performs the matching and later initiates a zero-copy transfer of the data. This approach minimizes data copying overheads, but imposes the latency of at least one additional roundtrip on the wire, and hence is unsuitable for small messages.

A key advantage to the one-sided communication model is that this tradeoff is not required, because the initiator always provides complete information describing the data transfer to be performed. There are no overheads imposed by matching, ordering or synchronization semantics, and the implementation is free to perform the data transfer using the most efficient mechanism available. On modern cluster networks, this usually translates into Remote Direct Memory Access (RDMA) operations supported by the hardware. This allows efficient remote access without intervention by the remote host CPU.

4.1. Latency Advantages of One-Sided

One of the key advantages of a one-sided communication model is that all relevant information about a communication operation is provided by the initiator – information is never required from the remote user code to complete communication. GASNet's one-sided data transfer operations are entirely decoupled from inter-process synchronization, allowing data transmission to begin immediately upon operation initiation (subject only to network congestion) and proceed autonomously from any action at the target process. For example, in a put operation the initiator can always transmit the precise destination address along with the data,

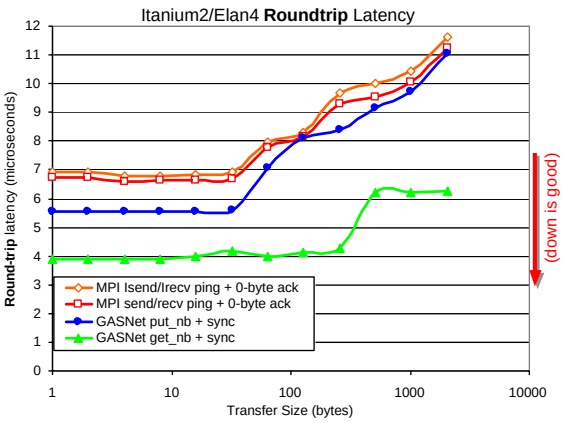

Figure 1. Latency of GASNet vs. MPI on Quadrics Elan4

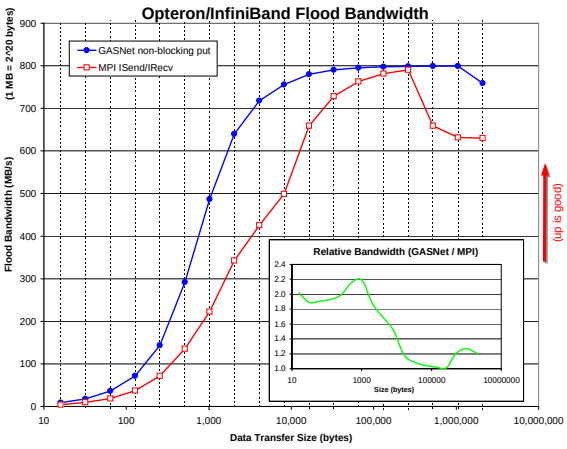

Figure 2. Bandwidth of GASNet vs. MPI on InfiniBand

providing a close semantic match to the requirements of high-bandwidth, zero-copy RDMA hardware. Conversely, a similar operation in MPI message-passing requires retrieving the destination address from a matching receive operation posted by the target user process (possibly at some point in the future) before the transfer can be completed. This matching operation often dictates the performance of MPI implementations, and thus vendors invest significant effort in optimizing its cost.

Although the FT kernel is a bandwidth-limited problem, the effective use of overlap in our approach depends crucially on the per-message CPU overheads associated with initiating and completing non-blocking operations. These overheads can be difficult to measure directly, however comparisons of small-message latency performance provide insight into the effects of software overhead, as it tends to comprise a large fraction of small-message latency. In network processor-based solutions such as Quadrics QsNetI/II, the network interface is capable of autonomously completing MPI message matching. Such approaches generally outperform host-based solutions that require attention from the remote host CPU (e.g., Myrinet and InfiniBand). We expect such MPI protocol offload to become increasingly prevalent, and therefore examine the Quadrics network in this section, since it represents a best-case for MPI latency.

Figure 1 compares the *round-trip* latency performance over varying data transfer size of GASNet's elan-conduit with Quadrics MPI on an Itanium2/Elan4 system. The GASNet tests measure the round-trip latency to issue a GASNet put or get operation and block for round-trip completion. The MPI tests measure the round-trip latency for a ping-pong test where the initiator sends a message of the given size, and the remote side issues a 0-byte acknowledgement message. Performance is measured using both the blocking (`MPI_Send/MPI_Recv`) and non-blocking (`MPI_Isend/MPI_Irecv`) message passing primitives.

The Quadrics network hardware offloads MPI message matching onto the NIC processor via the Elan Tports interface, freeing the host processor from most duties associated with the MPI message queue. However as evidenced by the figure, there is still a pronounced latency difference between this interface and the performance achievable through the lighter-weight, raw RDMA elan interfaces (`elan_put/elan_get`) targeted by the GASNet put/get implementation on Quadrics. One-sided communication is a better semantic match to RDMA-enabled hardware, and thus induces lower software overhead and delivers better latency performance for small and medium-sided messages.

4.2. Bandwidth Advantages of One-Sided

Figure 2 compares the flood bandwidth performance over varying transfer size of GASNet's InfiniBand/VAPI conduit with OSU MVAPICH [28], an extensively tuned implementation which is widely considered to be the best available MPI on InfiniBand. GASNet consistently and significantly outperforms MVAPICH on InfiniBand because the GASNet one-sided put/get semantics are fundamentally a better match for the capabilities of the underlying RDMA hardware than MPI's two-sided message passing semantics. GASNet's put/gets translate to simple, fully one-sided RDMA operations in the common case, and therefore reap the hardware peak performance, whereas MVAPICH pays in performance for enforcing MPI's ordering and message matching semantics. The flood bandwidth performance drop-off beyond the 256KB data transfer size is an artifact caused by a performance bug in the Mellanox hardware, which GASNet has been tuned to avoid. GASNet uses a novel distributed algorithm called Firehose [4] to efficiently manage memory registration on pinning-based RDMA networks, such as Myrinet and InfiniBand. Firehose effectively delegates the control of registration resources to the RDMA initiators, successfully exposing one-sided, zero-copy com-

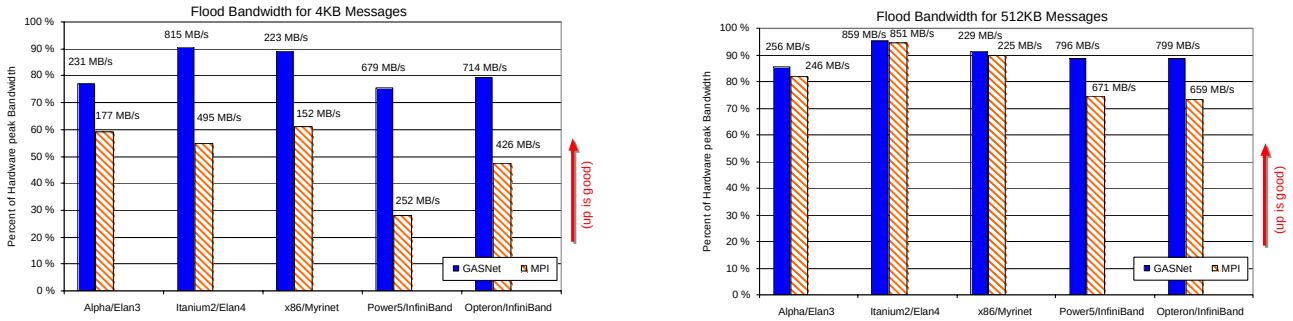

Figure 3. Flood Bandwidth of GASNet vs. MPI for 4KB and 512KB Messages

munication as a common case, while minimizing the number of host-level synchronizations required to support remote memory operations and amortizing the cost of synchronization and pinning over multiple remote memory operations.

The semantically-induced bandwidth performance gap between MPI's message passing and GASNet's one-sided communication is observable across a number of modern RDMA-enabled cluster interconnects. Figure 3 compares the flood bandwidth achievable with GASNet's one-sided put/get primitives against `MPI_Isend/MPI_Irecv` message-passing for 4KB and 512KB data transfer sizes across a number of production cluster supercomputers (as detailed in [5]). The bar height is normalized to the theoretical peak bandwidth of the system (a minimum of the interconnect link speed and the I/O bus speed), and absolute bandwidth performance is also shown. The figure demonstrates that for "large" messages, both the one-sided and message-passing mechanisms typically saturate to similar peak bandwidth values (the only exception being due to the InfiniBand hardware performance bug described above). However, one-sided communication consistently provides a significant performance advantage at "mid-ranged" sizes of about 1KB - 100KB, where the raw payload transmission times are often too short to fully amortize or overlap the MPI per-message overheads. Similar patterns have been observed on other systems for "mid-ranged" message sizes. Recent studies [24] of MPI usage across a range of real-world scientific applications have found that "mid-ranged" message sizes dominate many production applications and become even more prevalent at larger scales, motivating the importance of these message sizes. These message sizes can also be crucial in achieving efficient communication overlap, as described in the subsequent sections.

5. Optimizing Bandwidth-Limited Apps

In this section we consider a problem that is often hailed as the canonical example of a problem limited by bisection bandwidth, the three-dimensional Fourier Transform (FT). Superficially, none of the latency advantages of a one-sided model would appear to be relevant because the key to performance is the efficiency of a cross-processor transpose. As typically coded, the messages are all large and have a fixed size that is known in advance since it is a simple function of the problem size. The FT kernel is used in many scientific applications and is a critical operation in its own right, but it also reflects a more general class of algorithms that are a challenge to scalability and performance. Machines with inadequate bisection bandwidth typically suffer relative to those with full crossbars on applications requiring a large volume many-to-many communication [33].

5.1. NAS FT Benchmark

The NAS FT benchmark [2] solves a partial differential equation using a series of forward and inverse Fourier Transforms over three dimensions using the Fast Fourier Transform (FFT). Since the 3-D prism is laid out linearly in memory, the sets of 1-D FFTs must be transposed to calculate the complete 3-D FFT. A transpose is required for two out of the three dimensions to perform the 3-D FFT. The data can be decomposed across parallel threads along one or more of the dimensions. The reference implementation of the NAS FT benchmark uses a 1-D layout, where two of the dimensions are computed and transposed locally while third dimension incurs a global exchange among all processors, after which the FFT along the remaining dimension can be computed. It is implemented in Fortran with MPI, and the only significant communication step in this version is performed using the MPI All-to-all collective, a bulk communication operation where each thread exchanges its portion of the domain with every other thread. The existing exchange (All-to-all) version separates communication and computation into distinct phases: after computing the FFT over all its planes, every thread locally transposes the computed data into an ordering suitable for the exchange operation. After the global exchange, the data is re-transposed to complete the remaining FFTs. The communication can be placed between a local 2D-FFT and a local 1D-FFT or vice-versa.

5.2. Expressing NAS FT with One-sided Communication in UPC

A one-sided implementation of NAS FT modeled directly after the original Fortran/MPI code could perform the exchange using either point-to-point bulk *put* operations or alternatively use the collective operations in UPC [36]. Since large exchange operations are bandwidth-bound and are not noticably optimized beyond the performance of point-to-point communication, we expect UPC performance to at least match the performance of the original version given the point-to-point performance results presented in section 4. The data movement and communication patterns are similar in both the one-sided and two-sided variants of this implementation; since each communication call provides complete information to the communication library in the one-sided approach. Unlike the two-sided model where the target must provide the destination address, one-sided communication maps well to networks that can autonomously delivery data – the entire communication can proceed without involving the target processor. However, the size of the messages and overall communication in the exchange is sufficiently large to hide the implied synchronization costs imposed by the two-sided model.

5.3. Optimizing FT on Modern Networks

Issuing a single collective communication to globally exchange all FFT planes makes use of large messages with the goal of maximizing the available bandwidth and simplifies the programmer's task in identifying local dependencies: two of the three FFTs are complete prior to the exchange and the last FFT can begin once the exchange completes. The performance downside arises from the increasing monetary cost and complexity of providing full network bisection bandwidth as the amount of nodes involved in the exchange increases. Networks that do not provide full bisection bandwidth at high node counts can benefit from any operation that can amortize the cost of a global exchange operation. Those that do can still reduce the cost of a global exchange if the communication network supports asynchronous communication, because portions of the exchange can be hidden behind computation. Since our target networks support such operations, our approach is to decompose the FFT computation and communication into smaller pieces that permit overlap. In doing so, we have implemented the FT benchmark by decomposing the 2-D planes of the 1-D layout into smaller contiguous pieces. These implementations are summarized by the following two approaches:

- **Overlap Slabs**. Overlapping slabs is a method of decomposing the 3D-FFT to reduce the amount of time spent in communication-bound operations by overlapping the communication of previously computed slabs with the computation of remaining slabs. A slab is defined to be the portion of each FFT plane that has affinity to a single thread and will be sent to a single remote thread, such that communication incurs only a single *put* operation. In the original implementation, slabs destined for each remote thread are packed into contiguous buffers prior to the global exchange.
- **Overlap Pencils** The second method further reduces the granularity of the overlap by sending more, smaller-sized messages. The key observation is that each slab, which is destined to a single remote processor, is composed of $\frac{n}{THREADS}$ rows, or *pencils*. Instead of computing the FFT on all the rows destined to a remote thread before moving on to the rows for another thread, the pencils-based approach cycles through the slabs, and thus the remote thread, to choose the row to compute and send. This simultaneously increases the message injection rate, decreases the message size, and more aggressively intersperses the communication events with the computation.

FT Implementation	*Messages / thread*	*Overlap Efficiency*
Exchange	THREADS	0
Overlap Slabs	n	$1 - \frac{1}{n}$
Overlap Pencils	$\frac{n^2}{\text{THREADS}}$	$1 - \frac{\text{THREADS}}{n^2}$

Table 1. Summary of FT Algorithms as a function of the FFT cube dimensions (n^3)

Table 1 summarizes the implementations and provides a measure of overlap efficiency, the fraction of the computation that can potentially be overlapped with communication. The single exchange approach used by the original implementation is represented by *Exchange*. Both overlap algorithms have communication startup costs where the first and last units of communication (either a slab or pencil) cannot be overlapped with additional computation. Also shown is the total number of network messages sent by each thread for the global exchange. As can be expected, the finer the data decomposition, the greater the message count (up to a factor of the square of the input cube's dimensions for the Pencils algorithm). Also, since the volume of the data exchanged is constant across all the implementations, sending more messages also implies smaller messages.

6. Results

Performance results in this section are shown for three popular RDMA-based interconnect technologies: InfiniBand, Quadrics/Elan and Myrinet. The FT benchmark is very computationally intensive, and since each system uses different processors the overall MFlops rate is not intended to be directly compared across systems. The variety of interconnect hardwares are graphed to demonstrate that the

communication optimizations employed by FT UPC generalize to a large class of high performance system configurations. The descriptions of the platforms can be found in [5].

To measure the effectiveness of our approach, we wrote UPC and MPI/C versions of Exchange, Overlapped Slabs and Overlapped Pencils in order to compare them to the original Fortran/MPI implementation. To prevent serial FFT performance variances across different languages, all benchmarks use a 1-D decomposition of the domain and compute the 1-D FFTs using FFTW [22] (which consistently outperforms the Stockholm FFT used in the original NAS Fortran implementation by a small margin on all the platforms we used). The MPI-C and UPC versions of the benchmark are similar except for the language and/or library features they employ for non-blocking communication. UPC uses GASNet's non-blocking operations whereas MPI-C uses `MPI_Isend` combined with advance preposting of receive buffers such that no overheads resulting from unexpected messages were incurred by the underlying MPI communication layer. All but the original Fortran with MPI version of the benchmark employ a configurable padding parameter that allows one of the power-of-two dimensions in the FT class to be padded for more effective use of the memory hierarchy. We have found the padding to be most effective when computing FFT over the non-unit-stride dimensions. In addition, we implemented the benchmark using FFTW's built-in distributed 3D FFT which uses MPI and found that the Fortran with MPI version (and hence all our MPI and UPC versions) outperform this FFTW with MPI version. We thus only show results for the default MPI with Fortran implementation and our new implementations.

The MPI wallclock timer is used to profile the MPI versions, and hardware cycle counters of sub-microsecond accuracy were used to time the UPC versions. The data reported is the maximum performance across five trials - the standard deviation across the various trials was very low.

6.1. UPC Non-blocking Slabs and Pencils

The results in Figure 4 show the performance speedup of the UPC implementation of Exchange, Slabs and Pencils over the original Fortran implementation. Clearly, the approach of overlapping communication and computation over smaller units of the FFT is beneficial on all the tested interconnects and actually improves over newer generations of the same interconnect (Elan/Quadrics). Average speedups are on the order of 80%, with the most recent interconnects (InfiniBand and Elan4) showing 90% speedups. These speedups demonstrate that overlap can produce higher overall efficiency on systems that allow networking and computational resources to be used concurrently and independently.

In comparing UPC Pencils to Slabs, all platforms show

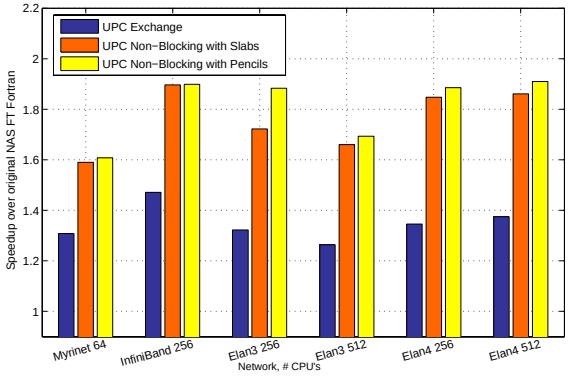

Figure 4. Speedup of UPC Exchange, Overlap Pencils & Slabs vs. Original Fortran FT

Pencils to be slightly faster. While the improvement is never beyond 10%, Pencils notably differ from Slabs by sending many more messages. This is contrary to the typical approach of sending fewer larger messages and validates that GASNet can effectively maintain or improve communication performance as it decreases the message size and increases the messaging rate.

In order to measure Pencil's impact on the increase in messaging rate, we identified two areas where Slabs and Pencils produced noticable differences across all platforms. The final 1-D FFT that occurs after all rows are communicated requires each received row to be reordered for a non-unit stride FFT. With Slabs, consecutive elements in the non-unit stride appear on different slabs, whereas Pencils allow the rows to be sent into an ordering that anticipates the remaining non-unit stride FFT and effectively reduces the stride to the length of a pencil. On systems such as the Alpha with a small TLB and/or high TLB miss penalty, the smaller stride improved computational performance by reducing pressure on the memory system and minimizing the number of TLB misses. Pencils therefore reduces the amount of time spent computing the final FFT, at a cost of a higher total message count. These combined costs are shown in Figure 5 for each platform, which illustrates that the time Pencils recovers in reordering is greater than the increased overhead of sending more messages. The overheads imposed by the increase in message rate and the decrease in message size are well amortized by the network through the use of GASNet. These overheads include initiation and completion costs as well as the inter-operation gap which represents the minimum amount of time between two consecutive message injections. Given the pronounced increase in message count and decrease in message size with Pencils, these overheads are kept relatively low. For example, for FT's Class D problem size at 256 processes, each process sends 1024 messages of 128KBytes with Slabs and 8192 messages of 16KBytes with Pencils.

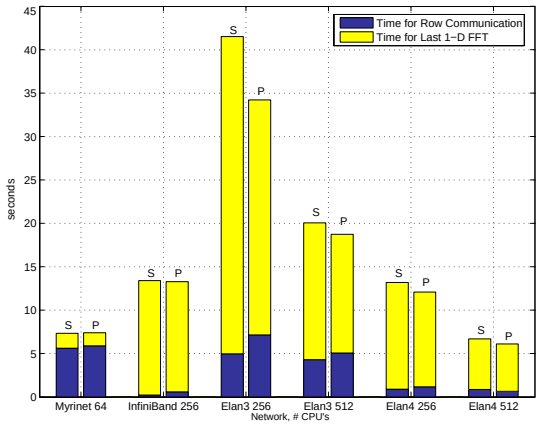

Figure 5. UPC Slabs (S) and UPC Pencils (P): Communication overhead and resulting computation performance

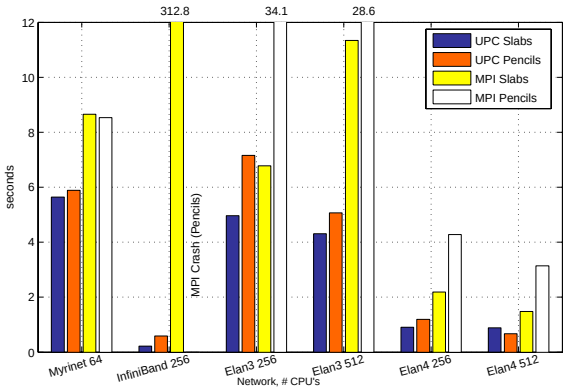

Figure 6. Time spent in communication calls for Overlapped UPC and MPI

6.2. UPC and MPI Overlap Comparison

In order to evaluate the effectiveness of our overlapping techniques with regards to one and two-sided communication, we also compare the Pencils and Slabs algorithms in MPI. The MPI implementations prepost receive buffers in a communication phase before non-blocking communication is initiated. This maximizes the potential for communication overlap and avoids unexpected MPI messages (which can degrade the performance of the MPI implementation). Under UPC, all communication is one-sided – the initiator provides both the source and destination addresses, and the non-blocking operations return an explicit handle which is later synchronized. Results comparing the UPC and MPI implementations of these non-blocking techniques are shown in Figure 6 in terms of total time each version of the benchmark spends in communication (all versions are always within 10% of each other for the times spent in computation). Perfect overlap would be represented as 0 seconds, and any time above 0 seconds represents the combined, non-overlapped cost of initiating and completing the non-blocking operations.

The Myrinet and Elan4 systems are the only configurations where the MPI implementations remain relatively competitive with those measured on UPC's FT implementations. The MPI configurations on other platforms either spent unacceptable time dealing with non-blocking communication messages or simply crashed. Although the MPI implementation of Overlapped Slabs and Pencils is compliant with the MPI 1.1 specification, the aggressive use of non-blocking point-to-point communication is not representative of the way MPI applications are typically written. While preposting buffers is always a good general strategy to prevent unexpected message costs, preposting and initiating several hundred communication operations on networks capable of highly asynchronous operation relies on correct and efficient MPI receive queue handling. The MPI results raise some obvious scalability concerns with regards to the number of point-to-point messages and the total amount of nodes exchanging messages. For example, in FT's Class D problem size at 256 processes using Pencils, each node sends and receives 2K messages for each plane it owns in the 3D cube decomposition and this leads to a sharp increase in the length of the MPI receive queues relative to the Slabs or Exchange algorithms. Performance problems are particularly apparent in the Opteron/InfiniBand system, where the MVAPICH implementation has scalability and correctness problems: the Pencils approach causes the application to crash and the Slabs approach causes the library to spent all of its time completing asynchronous sends and receives in MPI's *Waitall* primitive. On this same InfiniBand platform, the one-sided UPC version achieves the best communication overhead times for both Slabs and Pencils, indicating that the hardware is certainly capable of producing significant speedups using either technique.

UPC's overlapped approaches fare best on the more recent interconnect technologies we have benchmarked, where the commmunication times demonstrate a high messaging rate and low overhead for small-sized messages. Figure 7 presents a summary of the best result we could obtain on each platform for the original Fortran implementation, the best overall MPI and best overall UPC implementations. All platforms use the largest FT problem size (Class D) with the exception of Myrinet where the problem size was too large to fit at 64 processors and on InfiniBand where the MPI implementation's unreliable handling of the largest class required the smaller Class C to be used. In all cases, the Best MPI constitutes of the better of either the Exchange-based MPI/C, Slabs MPI/C or Pencils MPI/C and happens to be Slabs MPI/C each time. This is not surprising since all the networks are capable of some form of asynchronous operation, and any overlap is better than no over-

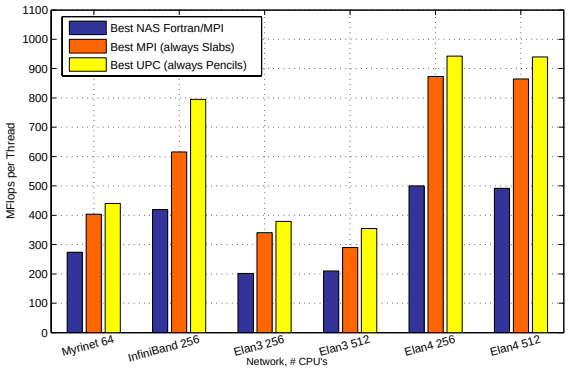

Figure 7. Best MPI and UPC Results

lap. The Best UPC happens to always be the Pencils version when compared to the Exchange and Slabs approaches. In many cases, the overlapped versions of the code nearly double the performance of the original implementation. The original NAS FT code differs from the other versions since it is written in Fortran, however the other UPC and MPI/C variants share 90% of the code base. The UPC Slabs and Pencils approaches differ in only about 10 lines of code, indicating the relative ease of programming for performance using finer-grained overlap in the one-sided model.

GASNet's low overhead for initiating many non-blocking communication operations allows a performance advantage through computation/communication overlap, trading latency associated with large bandwidth-bound operations such as the global exchange for minimal small message initiation and completion costs. The synchronization and messaging overheads imposed by the two-sided model are shown to come at a noticeable cost, and with some MPI implementations, seriously limit the effectiveness of our overlap optimizations. The one-sided model successfully reduces these overheads and delivers good small message pipelining rates which largely determine the overlap efficiency of the benchmark on RDMA-capable networks.

7. Related Work

As we have shown, large performance improvements are possible by distributing the communication operations throughout an application rather than segmenting it into computation and communication phases. Analyzing and optimizing the all-to-all communication pattern needed for a large parallel Fourier transform has been the subject of many papers. The analyses have ranged from modeling the performance on small commodity clusters [1, 12, 19] through using highly specialized networks [14, 18, 34].

There has been much work on tuning the communications libraries. Automatic tuning efforts [21, 37] create portable high performance implementations through an exhaustive search of the implementation space. Network vendors have also spent considerable efforts tuning their communication libraries so that users can leverage hardware-supported collectives [3]. Unlike these methods that focus primarily on the communication phases, we analyze the communication and computation patterns together showing that the communication can be spread through out the computation to relieve communication bottlenecks.

Iancu et. al [26] consider the benefits of breaking large messages into smaller ones to automate overlap. Their segmentation is automated by the compiler and thus is subject to the limitations of static analysis. Danalis et. al [16] describe techniques similar to our own to explicitly spread the communication across the computation – however their main focus is on applications written for a two-sided model, which is similar to our MPI/C Slabs and Pencils implementations. Danalis et. al also leverage the advantages RDMA and communication-computation overlap to show that a lower level communication library can yield better performance. We argue that applications written using one-sided semantics can realize further improvements because of the inherent advantages of the one-sided model, as shown by the gains of UPC Pencils over MPI Slabs. In addition, our work demonstrates the effectiveness of these techniques on a variety of cluster interconnects and shows that this approach scales to large processor counts and large problem sizes, further extending and validating their findings.

Finally, previous work on implementing the NAS parallel benchmarks in UPC [20] and Co-Array Fortran [15] was based on translating the MPI or OpenMP versions. In this work, the UPC implementations were written from scratch using a one-sided paradigm and thus are able to more effectively leverage the communication features of UPC/GASNet and demonstrate the capabilities of the system. We've applied some of the ideas from this work to our implementation of NAS FT in Titanium [17] (which also uses GASNet for communication) and achieved similar speedups over the MPI versions.

8. Conclusions

We have presented a detailed investigation into the relative performance of one-sided and two-sided communication, using UPC on GASNet for one-sided communication and MPI v1.1 for two-sided message-passing. Our microbenchmarks demonstrate that GASNet significantly outperforms MPI in latency performance and small to mid-size message bandwidth. As expected, both models reach the same asymptotic bandwidth at large message sizes on most platforms, but GASNet reaches the peak for smaller message sizes than MPI.

Our results suggest that one-sided communication offers an opportunity to revisit some commonly shared beliefs induced by two-sided message passing. One typical assump-

tion is that performance is optimized by sending fewer and larger messages to asymptotically approach peak bandwidth on cluster networks. The one-sided model provides alternative mechanisms whereby small messages can provide lower startup and completion costs, and the programmer can retain explicit control over synchronization by separating it from data movement. Applying these techniques to the well-known NAS FT benchmark, we have shown improvements in two dimensions. First, the low startup and completion costs that determine the potential for efficient communication and computation overlap have produced almost 2x speedups over the existing reference NAS FT Fortran implementation. Second, by aggressively pipelining smaller-sized messages and not imposing any unnecessary synchronization or ordering constraints over these messages, the one-sided approach as implemented in Berkeley UPC/GASNet has produced more efficient and consistent results than the two-sided approach and various MPI implementations. These results are consistent across four different cluster networks (Myrinet, Infinband, and two generations of Quadrics) and highlight the viability of UPC as a high performance programming model for clusters.

These results provide evidence that the bulk-synchronous, message-passing style of communication popularized by MPI may not be the most effective use of cluster networking hardware. As the number of processors grows in future machines, and networks become a more significant component of system cost, optimizations such as communication and computation overlap, use of small messages to increase the depth of message pipelines, and reductions in communication overhead through one-sided communication models are likely to increase in importance.

References

[1] R. C. Agarwal, F. G. Gustavson, and M. Zubair. A high performance parallel algorithm for 1-d fft. In *SC*, pages 34–40, 1994.

[2] D. H. Bailey, et al. The NAS Parallel Benchmarks. *The Int'l J. of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[3] J. Beecroft , et al. QSNETII: Defining high-performance network design. *IEEE Micro*, 25(4):34–47, 2005.

[4] C. Bell and D. Bonachea. A new DMA registration strategy for pinning-based high performance networks. In *Communication Architecture for Clusters (CAC03), Nice, France*, 2002.

[5] C. Bell, D. Bonachea, R. Nishtala, and K. Yelick. Optimizing bandwidth limited problems using one-sided communication and overlap. Technical Report LBNL-59207, Berkeley National Lab, 2005.

[6] C. Bell, W. Chen, D. Bonachea, and K. Yelick. Evaluating Support for Global Address Space Languages on the Cray X1. In *19th Annual Int'l Conference on Supercomputing (ICS)*, June 2004.

[7] The Berkeley UPC Compiler, 2002. http://upc.lbl.gov.

[8] D. Bonachea. GASNet specification. Technical Report CSD-02-1207, University of California, Berkeley, October 2002.

[9] D. Bonachea. Proposal for extending the UPC memory copy library functions and supporting extensions to GASNet, v1.0. Technical Report LBNL-56495, Berkeley National Lab, October 2004.

[10] D. Bonachea and J. C. Duell. Problems with using MPI 1.1 and 2.0 as compilation targets. In *Proc. Support for High Performance Scientific and Engineering Computing (SHPSEC-03)*, 2003.

[11] F. Cantonnet, Y. Yao, M. Zahran, and T. El-Ghazawi. Productivity Analysis of the UPC Language. In *IPDPS*, 2004.

[12] S. Chalasani and P. Ramanathan. Parallel FFT on ATM-based networks of workstations. In *Proc. of the 6th Int'l Symposium on High Performance Distributed Computing (HPDC)*, 1997.

[13] W. Chen, D. Bonachea, J. Duell, P. Husband, C. Iancu, and K. Yelick. A Performance Analysis of the Berkeley UPC Compiler. In *Proc. of Int'l Conference on Supercomputing (ICS)*, June 2003.

[14] C. Y. Chu. Comparison of two-dimensional FFT methods on the hypercube. In *Proc. of Hypercube concurrent computers and applications*, pages 1430–1437, 1988.

[15] C. Coarfa, Y. Dotsenko, J. Eckhardt, and J. Mellor-Crummey. Co-array Fortran performance and potential: An NPB experimental study. In *16th Int'l Workshop on Languages and Compilers for Parallel Processing (LCPC)*, October 2003.

[16] A. Danalis, K.-Y. Kim, L. Pollock, and M. Swany. Transformations to parallel codes for communication-computation overlap. In *Supercomputing 2005*, November 2005.

[17] K. Datta, D. Bonachea, and K. Yelick. Titanium performance and potential: an NPB experimental study. In *Proc. of Languages and Compilers for Parallel Computing*, 2005.

[18] L. Díaz, M. Valero-García, and A. González. A method for exploiting communication/computation overlap in hypercubes. *Parallel Computing*, 24(2):221–245, 1998.

[19] P. Dmitruk, et al. Scalable parallel FFT for spectral simulations on a beowulf cluster. *Parallel Computing*, 2001.

[20] T. El-Ghazawi and F. Cantonnet. UPC performance and potential: A NPB experimental study. In *Supercomputing*, 2002.

[21] A. Faraj and X. Yuan. Automatic generation and tuning of MPI collective communication routines. In *Proc. Supercomputing*, 2005.

[22] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proc. of the IEEE*, 93(2):216–231, 2005.

[23] GASNet home page. http://gasnet.cs.berkeley.edu/.

[24] D. Han and T. Jones. Survey of MPI call usage. In *SciComp*, 2004.

[25] P. Hilfinger, D. Bonachea, D. Gay, S. Graham, B. Liblit, G. Pike, and K. Yelick. Titanium language reference manual. Tech Report UCB/CSD-01-1163, U.C. Berkeley, November 2001.

[26] C. Iancu, P. Husbands, and W. Chen. Message strip mining heuristics for high speed networks. In *Proc. High Performance Computing for Computational Science (VECPAR)*, 2004.

[27] J. Liu, A. Vishnu, and D. K. Panda. Building multirail Infiniband clusters: MPI-level design. In *SuperComputing*, 2004.

[28] J. Liu, J. Wu, and D. K. Panda. High performance RDMA-based mpi implementation over Infiniband. *Int'l J. of Parallel Prog.*, 2004.

[29] MPI: A message-passing interface standard, v1.1. Technical report, University of Tennessee, Knoxville, June 12, 1995.

[30] MPI–2: a message-passing interface standard. *Int'l J. of High Performance Computing Applications*, 12:1–299, 1998.

[31] J. Nieplocha and B. Carpenter. ARMCI: A portable remote memory copy library for distributed array libraries and compiler run-time systems. In *Proc. RTSPP IPPS/SDP'99*, 1999.

[32] R. Numrich and J. Reid. Co-array fortran for parallel programming. In *ACM Fortran Forum 17, 2, 1-31.*, 1998.

[33] L. Oliker, et al. Scientific computations on modern parallel vector systems. In *Proc. of Supercomputing*, 2004.

[34] P. Swartztrauber and S. Hammond. A comparison of optimal FFTs on torus and hypercube multicomputers. *Parallel Computing*, 2001.

[35] UPC consortium home page. http://upc.gwu.edu/.

[36] UPC language specifications, v1.2. Technical Report LBNL-59208, Berkeley National Lab, 2005.

[37] S. S. Vadhiyar, G. E. Fagg, and J. Dongarra. Automatically tuned collective communications. In *Supercomputing*, 2000.

[38] V. Velusamy, et al. Programming the Infiniband network architecture for high performance message passing systems. In *ISCA*, 2003.