Overcoming Communication Latency Barriers in Massively Parallel Scientific Computation

Ron O. Dror
J.P. Grossman
Kenneth M. Mackenzie
Brian Towles
Edmond Chow
John K. Salmon
Cliff Young
Joseph A. Bank
Brannon Batson
Martin M. Deneroff
Jeffrey S. Kuskin
Richard H. Larson
Mark A. Moraes
David E. Shaw
D.E. Shaw Research

Anton, a massively parallel special-purpose machine that accelerates molecular dynamics simulations by orders of magnitude, uses a combination of specialized hardware mechanisms and restructured software algorithms to reduce and hide communication latency. Anton delivers end-to-end internode latency significantly lower than any other large-scale parallel machine, and its critical-path communication time for molecular dynamics simulations is less than 3 percent that of the next-fastest platform.

In recent years, the rise of many-core architectures, such as general-purpose GPUs, the Cell Broadband Engine, and Larrabee, has led to a dramatic increase in the compute throughput achievable on a single chip. At the same time, the communication bandwidth between nodes on high-end networks has continued to increase, albeit at a slower rate. Unfortunately, improvements in end-to-end communication latency—the time from when application software on a source node initiates a message send to when application software on the receiving node begins using that data for computation—have not kept pace. As a result, internode communication latency increasingly limits the extent to which tightly coupled applications can be parallelized across many nodes, and thus the maximum achievable performance for a given problem size.

Classical molecular dynamics simulations, which model molecular systems at an atomic level of detail, become particularly sensitive to communication latency at high levels of parallelism owing to the need for frequent internode communication. A molecular dynamics simulation steps through time, alternating between a phase that determines the force acting on each simulated atom and a phase that advances the atoms' positions and velocities according to Newton's laws of motion. Such simulations are uniquely suited for capturing the behavior of biomolecules such as proteins and nucleic acids, but billions or trillions of sequential simulation steps are required to reach the time scales on which many of the most biologically important events occur. The time required to execute a single step is therefore a critical determinant of this technique's range of applicability. As a result, much effort has gone into accelerating molecular dynamics through parallelization across nodes using traditional CPUs, GPUs, and

Published by the IEEE Computer Society

other architectures,[1-5] but the maximum achievable simulation speed depends more on internode communication latency than on single-node compute throughput.

Anton, a massively parallel special-purpose supercomputer that dramatically accelerates molecular dynamics simulations, has performed all-atom simulations 100 times longer than any previously reported.[6] Anton's speedup is partly attributable to the use of specialized hardware that greatly accelerates the arithmetic computation required for a molecular dynamics simulation. Without a corresponding reduction in delays caused by latency, however, Anton would deliver only a modest performance improvement over other highly parallel platforms.

This article describes the architectural and algorithmic approaches[7] that reduce the total communication time on Anton's critical path to less than 3 percent of that of the next-fastest molecular dynamics platform.[8] First, we introduce Anton's techniques for reducing end-to-end latency of individual messages, which includes hardware network latency, sender and receiver overhead, and any additional communication needed for synchronization. Anton's shortest end-to-end internode latency is 162 ns, significantly lower than that of any other large-scale parallel machine of which we are aware. Second, we describe Anton's use of fine-grained communication—sending many small messages instead of a few big ones—to mitigate the effect of latency on overall application execution time by reducing aggregate communication time and better overlapping communication with computation. As computational density increases, latency will limit the performance of an increasing number of parallel applications, and combined hardware/software techniques of the sort Anton uses to mitigate the effects of latency could gain wider applicability.

Reducing end-to-end latency of individual messages

A message's end-to-end latency includes both the time required to communicate the data and the synchronization overhead required to inform the receiver that the data has arrived. Anton reduces the communication time by providing a tightly integrated high-performance network, and it minimizes synchronization overhead by using a mechanism we call *counted remote writes*.

Anton also reduces latency by minimizing the network distance a typical message traverses. Anton maps the entire molecular dynamics computation to a set of identical application-specific integrated circuits (ASICs), each of which constitutes an Anton node. The chemical system to be simulated—which might consist, for example, of a protein surrounded by water—is divided into a regular grid, with each grid box assigned to one node. The nodes connect to form a 3D torus, a 3D mesh that wraps around in each dimension (see Figure 1), and neighboring grid boxes are assigned to neighboring nodes. Because much of the communication in a molecular dynamics simulation involves neighboring grid boxes, this arrangement allowed us to design Anton's software such that most messages traverse only one or two network hops. Nevertheless, internode latency constitutes much of the critical path of each time step and thus remains a major determinant of overall performance.

Tight network integration

Every computational subunit on an Anton ASIC connects to an on-chip network router; the routers handle communication both within and between ASICs. Each ASIC contains seven such computational subunits, which we call network clients: four flexible subsystem processing slices, each including three programmable cores; one high-throughput interaction subsystem (HTIS), which includes specialized hardwired pipelines that compute interactions between pairs of atoms; and two accumulation memories, which are used to sum forces. The on-chip routers are connected into a ring, and each ASIC is directly connected to its six immediate neighbors within the 3D torus by six 50.6 gigabits-per-second-per-direction links (Figure 1).

Each network client contains a local, on-chip memory that can directly accept write packets issued by other clients. Both the processing slices and the HTIS units have specialized hardware support for quickly

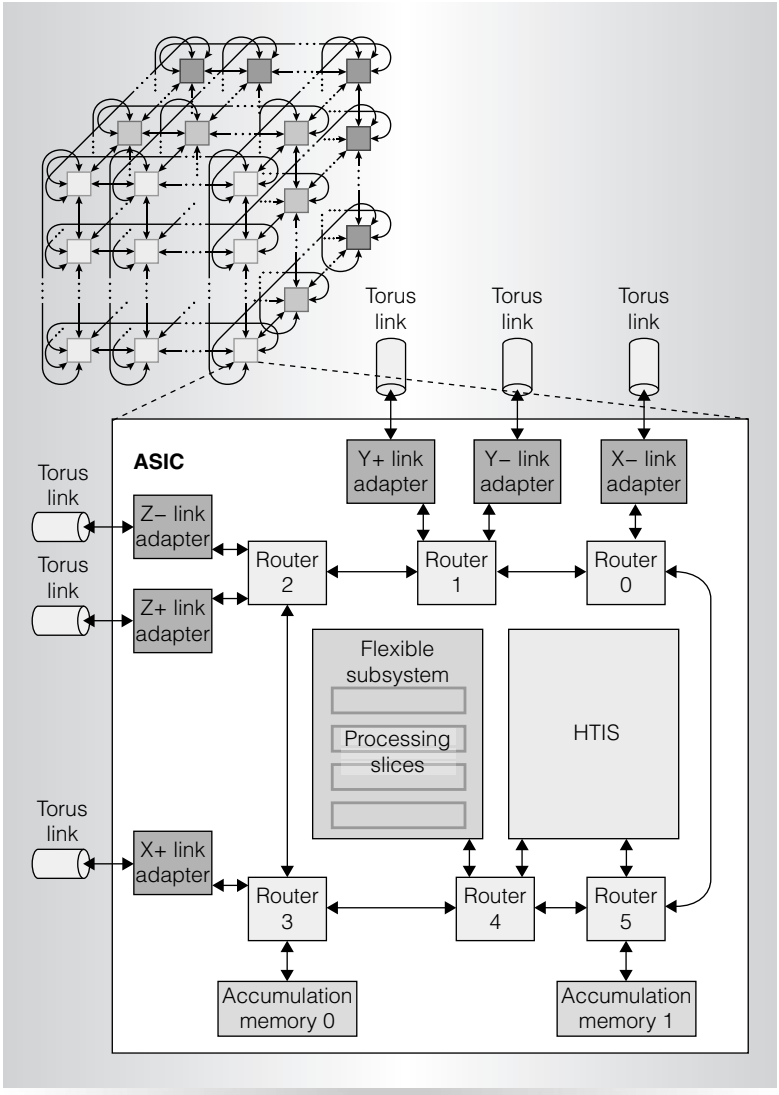

Figure 1. Connectivity of an Anton application-specific integrated circuit (ASIC). Six on-chip routers form a ring with connections to the computational subunits and internode link adapters. Corresponding link adapters on neighboring ASICs are directly connected via passive links to form a 3D torus.

assembling packets and injecting them into the network. The accumulation memories can't send packets, but they accept a special accumulation packet that adds its payload to the current value stored at the targeted address. Each packet's header specifies not only that packet's destination node, but also the specific local memory and memory address to which it will be written.

This combination of features allows for remote-write operations with unusually low sender and receiver overhead. Unlike the remote direct memory access (RDMA) protocols that many modern general-purpose machines support, Anton enables remote writes directly to each computational subunit's local, on-chip memory rather than to a separate DRAM chip. Anton thus eliminates the overhead associated with fetching data from DRAM before application software on the receiving node can operate on it.

Counted remote writes

When using remote writes, some form of synchronization is required to inform the receiver that all the data required for a given computation has arrived. Every network client on an Anton ASIC contains a set of synchronization counters, which can be used to determine when a predetermined number of packets has been received. Each packet's header includes a synchronization counter identifier; once the receiver's memory has been updated with the packet's payload, the selected synchronization counter is incremented. Clients can poll these counters to determine when all data required for a computation has been received, allowing synchronization with no additional communication. The processing slices and HTIS units can directly poll their local synchronization counters, resulting in low polling latencies.

These synchronization counters form the basis of a key communication paradigm on Anton that we call counted remote writes (Figure 2). When one or more network clients must send a predetermined number of related packets to a single target client, space for these packets is preallocated within the target's local memory. The source clients write their data directly to the target memory, labeling all write packets with the same synchronization counter identifier. This operation is logically equivalent to a gather (a set of remote reads) performed by the target client, but it avoids explicit synchronization between the source clients and the target—the sources simply send data to the target when the data is available.

By combining tight network integration, efficient packet creation, and local polling of synchronization counters at the target client, Anton's counted-remote-write

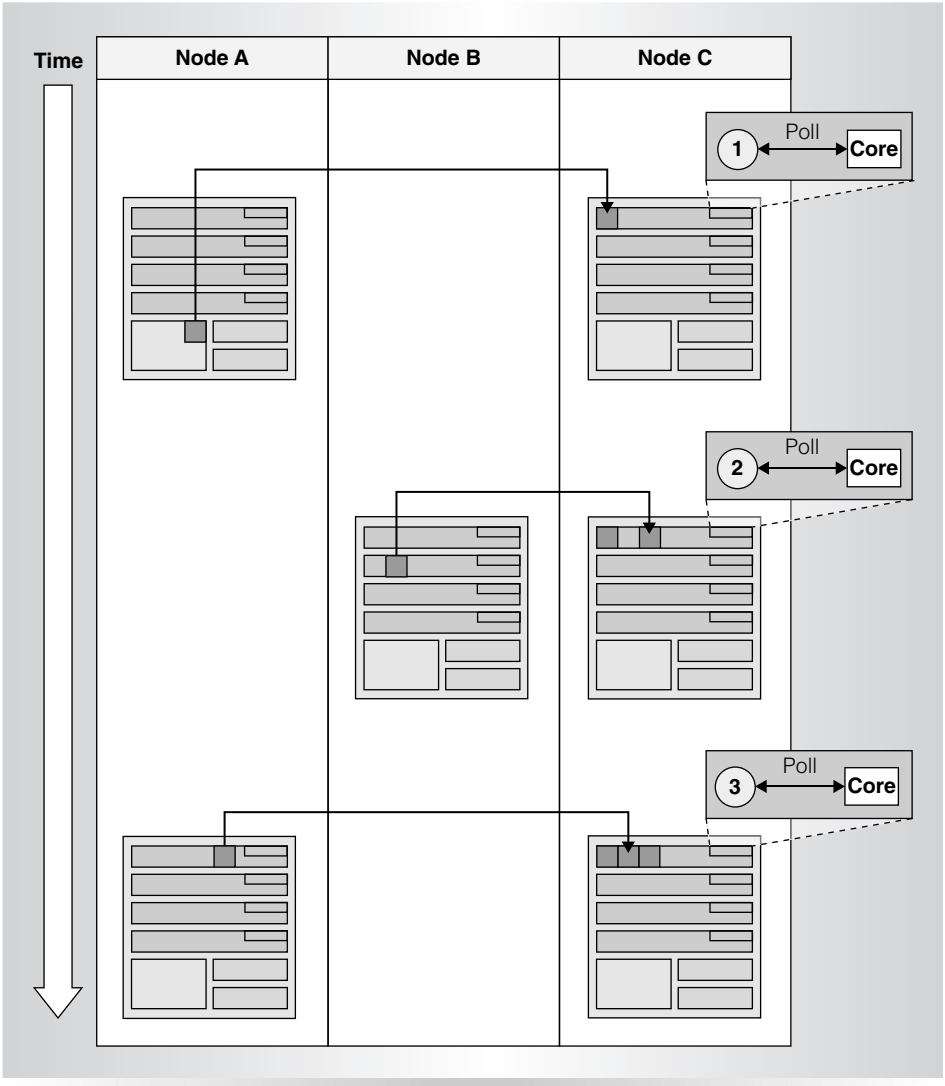

Figure 2. Two source nodes (A and B) use counted remote writes to send data directly to the local memory of a particular processing slice on a target node (C). Each node contains seven local memories associated with computational subunits of three types, depicted by rectangles within each node; the four uppermost rectangles correspond to processing slices. The arriving packets identify one of the processing slice's synchronization counters, which is incremented as each packet arrives. A core within that processing slice polls the synchronization counter to determine when the expected number of packets has arrived.

mechanism results in low end-to-end message latencies. Figure 3 plots end-to-end latency as a function of the number of hops across the torus network. Figure 4 shows how single-hop latency breaks down among various hardware components as a message travels from the source to the destination. The total end-to-end latency between two Anton nodes—from the time that the application software on a source node initiates a send to the time that the application software on the target node begins using the received data—is as low as 162 ns. This is significantly lower than the fastest published measurements of which we are aware for internode end-to-end latency across a scalable network on any other hardware.

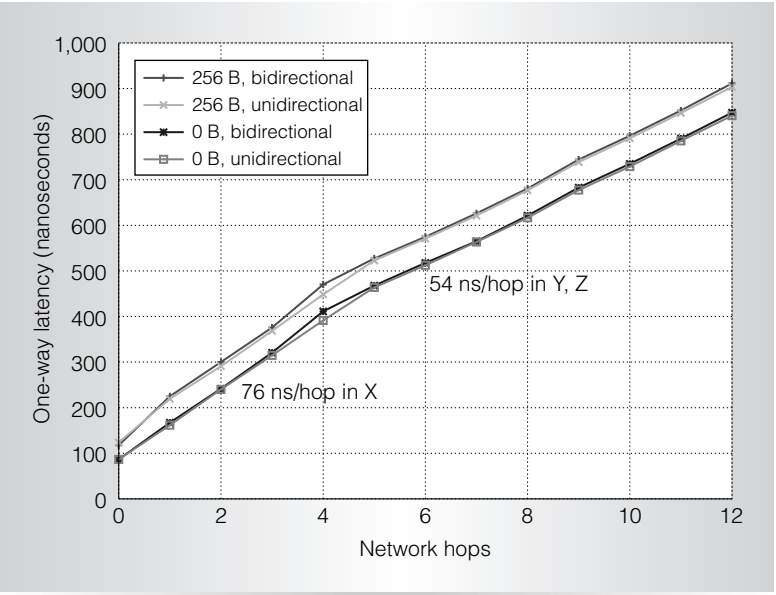

Figure 3. One-way counted-remote-write latency versus network hops in a 512-node Anton machine configured as an 8 × 8 × 8 torus, measured by unidirectional and bidirectional ping-pong tests for two message sizes. The zero-hop case sends messages between processing slices on the same node, the one- through four-hop cases travel along the torus network's *X* dimension, and the five- through 12-hop cases add hops along the *Y* and *Z* dimensions. The *X* hops traverse more on-chip routers per node and thus have a higher latency than the *Y* or *Z* hops. Anton's software is tuned to minimize the number of network hops required for most messages; no internode message on Anton traverses more than eight hops, and the majority traverse only one or two hops.

Table 1 shows a survey of published internode end-to-end latency measurements. We define end-to-end latency as the time from when application software on a source node initiates a message send to when application software on the receiving node begins using that data for computation; end-to-end latency is generally measured by a half-round-trip ping-pong time. This survey excludes measurements of one-sided writes that do not include the time required by the target node software to determine that a message has arrived, as well as measurements of communication that do not traverse a scalable internode network. Comparable measurements for Cray's recently released XE6 system haven't been published but are estimated to be approximately one microsecond (S.L. Scott, Cray, private communication, July 2010). An extrapolation from remote cache access times in scalable shared memory machines such as the EV7[9] would project latencies of a few hundred nanoseconds for messaging based on cache-to-cache communication, but to our knowledge, a performance analysis of this approach has not been publicly documented.

Fixing communication patterns

The use of counted remote writes requires carefully choreographed communication. To push data directly to its destination, senders must determine the node, network client, and exact memory address to which each piece of data should be written. To detect transfer completion by polling a local synchronization counter, each receiver must know exactly how many packets of a given type to expect. Anton supports alternative communication mechanisms for cases in which these conditions can't be satisfied, but these mechanisms have significantly higher end-to-end latency.[7]

We tailored Anton's software and algorithms to support the use of fixed communication patterns, and thus counted remote writes, in almost all cases. We preallocate receive-side storage buffers before a simulation begins for almost every piece of data to be communicated and, whenever possible, avoid changing these addresses during a simulation. We also fix the total number of packets sent to each receiver, choosing the number to accommodate worst-case temporal fluctuations in local atom density (typically about 1.5 times the average density).

As an example, consider the computation of bonded force terms, which involve small groups of atoms connected by chemical bonds. At each time step, the atom positions required to compute a given bonded force term must be brought together on one node. In most molecular dynamics software packages designed for parallelism, the assignment of bonded force terms to nodes changes as the atoms involved move through space. On Anton, however, we statically assign bonded force terms to nodes so that a given atom's set of destinations is fixed. With this strategy, we can preallocate memory on each node to receive incoming positions, ensuring that each node knows exactly how many positions it will receive. Atoms can be sent directly to their

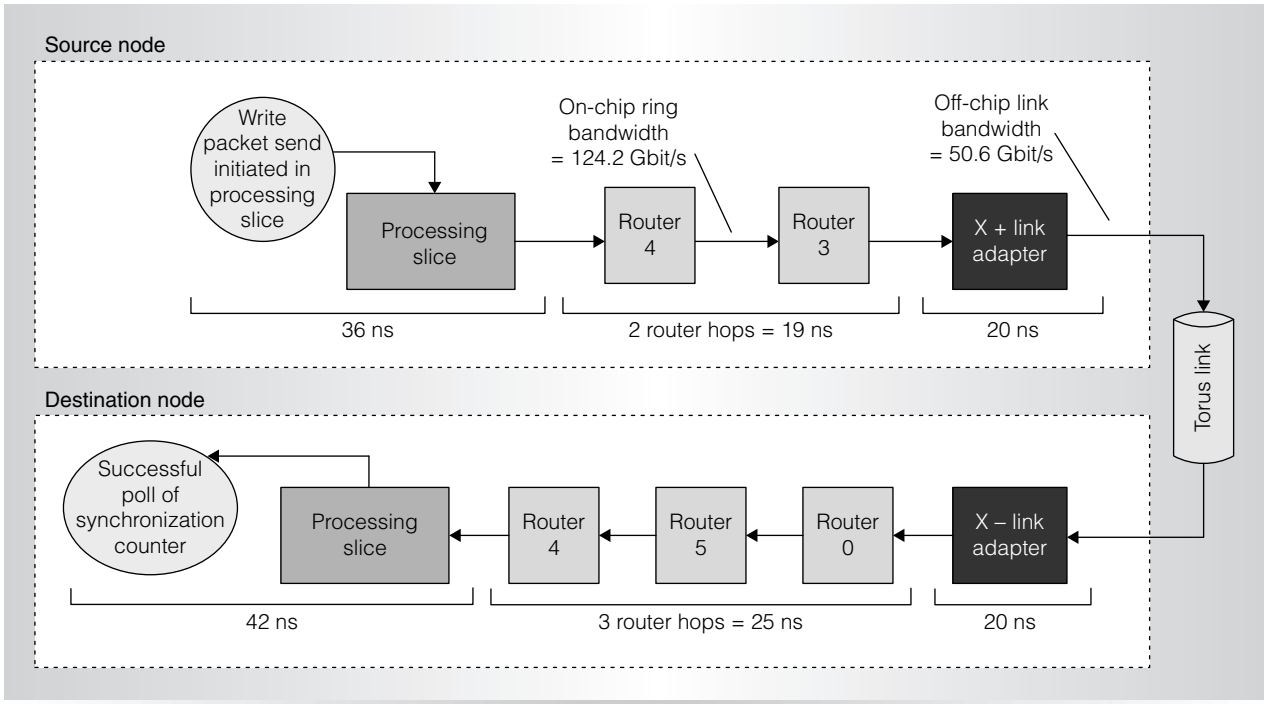

Figure 4. Detailed breakdown of the 162-ns counted-remote-write latency between processing slices on neighboring nodes along the *X* dimension of the torus. The passive torus link's wire delay has been incorporated into the latency shown for the link adapters.

destinations using fine-grained (one atom per packet) counted remote writes.

Given the initial placement of atoms, we assigned bonded force terms to nodes in order to minimize the required number of network hops for most messages. As the atoms move, the hop counts increase, causing performance to degrade over the course of several hundred thousand time steps. To maintain low communication latencies, we therefore compute a new assignment of bonded force terms to nodes every 100,000 to 200,000 time steps. We avoid pausing the simulation by computing the new assignment in the background as the simulation proceeds.

Table 1. Survey of published internode end-to-end latency measurements across scalable networks.

Machine	Latency (μs)	Date
Anton[7]	0.16	2010
Altix 3700 BX2[10]	1.25	2006
QsNetII[11]	1.28	2005
Columbia[12]	1.6	2005
EV7[9]	1.7	2002
J-Machine[13]	1.8	1993
Blue Gene/P[14]	1.9	2008
Roadrunner (InfiniBand)[15]	2.16	2008
Cray T3E[16]	2.75	1996
Blue Gene/L[17]	2.8	2005
ASC Purple[17]	4.4	2005
Red Storm[17]	6.9	2005

Fine-grained communication

On most scalable networks, sending one long message is much cheaper than splitting the same data into many small messages, but on Anton, the difference is small. Anton exploits fine-grained communication to reduce latency's negative effects on overall application performance, both by reducing aggregate communication time and by better overlapping communication with computation.

Achieving low per-message overhead

Anton reduces per-message overhead in two ways. First, the network uses small message headers, which are only 32 bytes

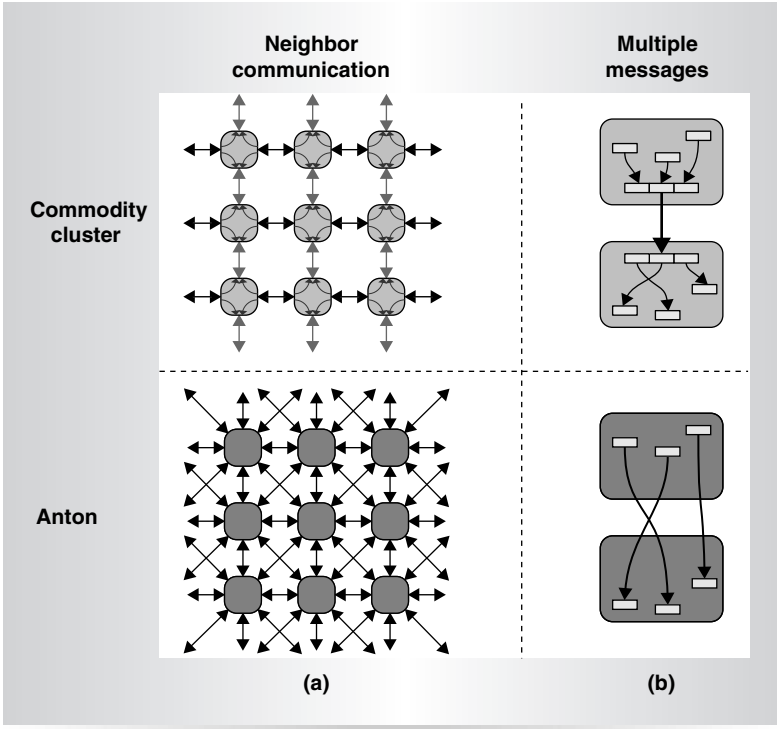

Figure 5. Comparison of fine-grained communication on Anton to coarse-grained communication on a commodity cluster. Two algorithms for pairwise all-neighbor data exchange are shown in two dimensions (a). Staged communication (horizontal first, vertical second, with data forwarded from the first stage to the second) is preferable for a commodity cluster to reduce the message count (a, top); in three dimensions, this technique lets every node exchange data with its 26 neighbors in three stages using only six messages per node.[2] Staging is undesirable in Anton, where performance is optimized with a single stage of direct communication (a, bottom). Next, two methods for sending multiple data items from a sender to a receiver are shown (b). Local sender/receiver data copies are often used to avoid multiple messages between commodity nodes (b, top). Multiple direct remote writes eliminate this copy overhead in Anton (b, bottom).

in length. Second, sender and receiver overheads are minimized. Each sender has hardware support for sending raw data directly from its local memory and can initiate a new message in a single cycle. There is no per-message receiver overhead, aside from receiver input bandwidth requirements, because the data is delivered directly to the local memory, and the receiver can use a single counter to track the receipt of many counted remote writes. Thus, the overall cost of implementing communication using a large number of fine-grained messages on Anton is extremely low.

As an example, the total time required to transfer 2 Kbytes of data between two nodes in 64 separate messages is only 50 percent higher than the time required to send that data as a single message; on a double-data rate (DDR) InfiniBand network, the difference is 700 percent.[7] Additionally, fine-grained messages can effectively use Anton's network bandwidth: 50 percent of the maximum possible data bandwidth is achieved with 28-byte messages on Anton, compared with much larger 1.4-, 16-, and 39-Kbyte messages on Blue Gene/L, Red Storm, and ASC Purple, respectively.[17]

Exploiting fine-grained communication

Anton exploits communication patterns involving many small messages to avoid extra communication stages or additional processor work in preparing data for transmission. Instead of employing a multistage approach in which each node recombines or processes the data in messages received at one stage in preparation for transmission at the next stage—an important technique for improving performance on commodity clusters[2,18]—Anton can benefit from the lower latency of a single communication stage in which every node sends many messages directly to their final intended recipients (Figure 5a). Anton's software can also avoid local copy or permutation operations by sending additional messages, effectively performing these operations in the network (Figure 5b).

The use of fine-grained communication also lets Anton better overlap computation and communication. Certain computations start as soon as the first message has arrived, while other messages are still in flight. Likewise, transmission of results typically begins as soon as the first results are available so that the remainder of the computation is overlapped with communication.

Our implementation of global summation (adding a quantity across all nodes such that each node obtains the global sum) illustrates Anton's strategy for exploiting fine-grained communication. We use a dimension-ordered algorithm in which the 3D summation is decomposed into parallel 1D summation operations along the *x*-axis, followed by summation operations along

the y-axis, and then the z-axis. Each 1D summation operation involves a set of N nodes connected in a loop; Figure 6 illustrates the communication among these nodes for $N = 8$. On Anton, each node sends a message to all seven others in a single communication stage (using a multicast mechanism to avoid redundant communication). Optimized software for other networks, by contrast, would typically use a three-stage approach, with each node sending only a single message at each stage. Anton's approach requires seven messages per node, whereas the alternative approach requires only three.

On a network that supports efficient fine-grained communication, Anton's communication pattern yields two substantial advantages. First, it involves a single communication stage rather than three, thus allowing overlapped communication of all the individual messages. Second, Anton's approach requires only four sequential hops (the diameter of an eight-node loop), whereas the multistage approach requires seven (one in the first stage, two in the second, and four in the third).

The latency of a 32-byte global summation operation (also called an *all-reduce operation*) on a 512-node Anton machine is 1.77 μs, representing a 20-fold speedup over the time we measured for the same reduction on a 512-node DDR InfiniBand cluster (35.5 μs). Anton's 1.77-μs latency also compares favorably with that of a pure hardware implementation: on Blue Gene/L, a 16-byte global summation across 512 nodes using a specialized tree-based network designed for this purpose has a latency of 4.22 μs.[19]

For a discussion of other approaches that have been used to reduce latency and support fine-grained communication, see the "Related Work in Reducing Communication Latency and Exploiting Fine-Grained Communication" sidebar.

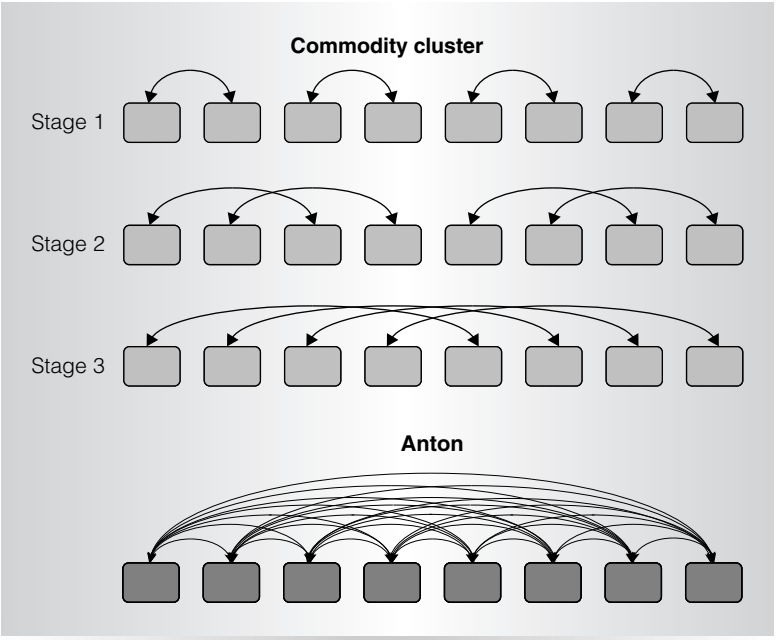

Figure 6. Two strategies for summing a value across eight nodes connected in a loop such that each node obtains the final result. A three-stage approach is preferable on most networks, while a single-stage approach is preferable on Anton, which supports efficient fine-grained communication.

Table 2. Performance benchmarks for Anton and the next-fastest molecular dynamics platform.

	Anton	Next-fastest platform
Average time step duration (μs)	13.5	565
Critical-path communication time (μs)	7.7	262

Total communication time for molecular dynamics

Table 2 shows the total critical-path communication time during Anton's execution of a typical molecular dynamics simulation. We computed this communication time by subtracting critical-path arithmetic computation time from total time, so it includes all sender, receiver, and synchronization overhead, as well as the time required for on-chip data movement. Measurements were performed on a 512-node Anton machine using an on-chip diagnostic network that lets us monitor ASIC activity.[7]

For comparison, Table 2 also lists corresponding times for the hardware/software configuration that produced the next-fastest reported molecular dynamics simulations:[8] a high-end 512-node Xeon/InfiniBand cluster running the Desmond molecular dynamics software.[2] (Detailed specifications of the cluster hardware are given elsewhere.[8]) The benchmark system for all measurements in the table is dihydrofolate reductase in

Related Work in Reducing Communication Latency and Exploiting Fine-Grained Communication

Anton's software relies on several architectural features—including tight network integration, counted remote writes, and efficient fine-grained communication—to reduce critical-path communication time. One important implication of tight network integration is that it gives rise to fast software-initiated remote writes. Many other architectures have been implemented with varying types of hardware support for remote writes (also called put operations or one-sided communication). The Cray T3D and T3E multiprocessors implement global shared memory by providing Alpha microprocessors with memory-mapped access to network controllers.[1] The Quadrics QsNetII interconnect supports fast remote writes via programmed I/O.[2] QCDOC provides hardware support for remote direct memory access (RDMA) operations.[3] Shared memory can also be used to implement remote writes; the EV7 AlphaServer contains a specialized network that directly supports cache-coherent shared memory across up to 64 nodes of a parallel machine.[4] All these mechanisms require separate synchronization to inform the receiver that data has arrived.

Anton avoids the need for separate synchronization by using counted remote writes. Another approach is to write to a memory-mapped receiver first-in, first-out rather than a specific remote-memory address, so that the data's arrival is indicated by the tail pointer's advancement. This is implemented in Blue Gene/L.[5] Yet another approach is to provide word-level synchronization for remote writes; in the J-Machine, each memory word has an associated "present" bit for this purpose.[6] Anton's counted remote writes are more general and allow a single synchronization event to be associated with the arrival of larger data sets from multiple locations; a similar strategy was previously proposed for the Hamlyn interface.[7]

Anton's support for efficient fine-grained communication allows the use of certain communication patterns that would be prohibitively expensive on commodity hardware, and it enables fast software-based global summations. Certain other architectures provide specific hardware support for global summations: Blue Gene/L[8] and Blue Gene/P[9] each contain secondary specialized networks for this purpose, while QsNetII[2] and the QCDOC application-specific integrated circuit[3] each provide hardware units to accelerate global summations over the primary network.

References

1. S.L. Scott, "Synchronization and Communication in the T3E Multiprocessor," *Proc. 7th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* ACM Press, 1996, pp. 26-36.
2. J. Beecroft et al., "QsNetII: Defining High-Performance Network Design," *IEEE Micro,* vol. 25, no. 4, 2005, pp. 34-47.
3. P.A. Boyle et al., "QCDOC: A 10 Teraflops Computer for Tightly-Coupled Calculations," *Proc. ACM/IEEE Conf. Supercomputing* (SC 04), IEEE CS Press, 2004, doi:10.1109/SC.2004.46.
4. D. Kerbyson et al., "Performance Evaluation of an EV7 AlphaServer Machine," *Int'l J. High Performance Computing Applications,* vol. 18, no. 2, 2004, pp. 199-209.
5. M. Blocksome et al., "Design and Implementation of a One-Sided Communication Interface for the IBM eServer Blue Gene Supercomputer," *Proc. ACM/IEEE Conf. Supercomputing* (SC 06), ACM Press, 2006, doi:10.1145/1188455.1188580.
6. M.D. Noakes, D.A. Wallach, and W.J. Dally, "The J-Machine Multicomputer: An Architectural Evaluation," *ACM SIGARCH Computer Architecture News,* vol. 21, no. 2, 1993, pp. 224-235.
7. G. Buzzard et al., "An Implementation of the Hamlyn Sender-Managed Interface Architecture," *Proc. 2nd USENIX Symp. Operating Systems Design and Implementation* (OSDI 96), ACM Press, 1996, pp. 245-259.
8. G. Almási et al., "Optimization of MPI Collective Communication on BlueGene/L Systems," *Proc. 19th Ann. Int'l Conf. Supercomputing* (ICS 05), ACM Press, 2005, pp. 253-262.
9. C. Sosa and G. Lakner, "IBM System Blue Gene Solution: Blue Gene/P Application Development," *IBM Redbook,* SG24-7287, 2008, www.redbooks.ibm.com/redbooks/pdfs/sg247287.pdf.

water, with 23,558 atoms; simulation parameters are as specified elsewhere,[6] except that the molecular dynamics simulations profiled here included a thermostat applied at every other time step.

During an average molecular dynamics simulation time step, Anton's critical-path communication time is approximately 1/34 that of the next-fastest molecular dynamics platform (Table 2). Most modern supercomputers could send and receive only a handful of messages per node during the 13.5 μs average duration of an Anton time step; by contrast, the average Anton node sends (and receives) over 500 messages per time step.

How general is Anton's approach?

Although Anton's hardware is specialized to molecular dynamics, the combination of hardware and software strategies Anton uses to reduce latency-associated delays could prove useful in accelerating other parallel applications. Anton's counted-remote-write mechanism, for example, provides a natural way to represent data dependencies in applications parallelized using domain decomposition, where a processor associated with a

subdomain must wait to receive data from other processors associated with neighboring subdomains before it can begin a given computation phase. This mechanism also accelerates nonlocalized communication patterns, such as global summations. Hardware support for counted remote writes can be implemented efficiently and can dramatically reduce the delays associated with this type of communication. It is particularly appealing for architectures with an out-of-order network (such as Anton) because it solves the problem of ensuring that data has arrived before the receiver is notified of its availability.

The use of counted remote writes depends on fixed communication patterns in which a predetermined number of packets are transmitted to predetermined locations at each step and thus generally requires restructuring of software and algorithms. In certain applications, such as those based on graph traversal, predetermining the number of packets that need to be transmitted and their destination addresses might be impossible. In practice, however, even applications with data structures that evolve during computation can often be implemented such that a large proportion of the communication is predictable and thus amenable to acceleration by counted remote writes. In adaptive mesh-refinement methods, for example, the mesh-repartitioning step involves unpredictable communication patterns, but the extensive computational phases between successive refinements typically contain regular communication patterns.

Anton further reduces the impact of latency by using fine-grained communication to eliminate local copy operations, avoid staged communication, and better overlap communication with computation. These strategies can be exploited in many parallel applications, but they require significant changes to software as well as network hardware. Existing parallel software is usually structured around the use of coarse-grained communication because the per-message overhead on most hardware is much larger than it is on Anton.

Anton serves as a case study demonstrating that, through a combination of hardware features and carefully organized software, one can achieve dramatic reductions in the latency-based delays that limit many scientific applications' parallel performance. Anton's ability to perform all-atom molecular dynamics simulations that are two orders of magnitude longer than any previously published simulation is due in no small measure to a 34-fold reduction in the critical-path communication time compared to the next-fastest implementation. As latency limitations become more severe relative to the increasing computational power of modern architectures, the combined hardware and software techniques Anton uses to circumvent communication bottlenecks could find broader adoption. MICRO

Acknowledgments

We thank Ed Priest, Larry Nociolo, Chester Li, and Jon Peticolas for their work on high-speed analog circuits and signal integrity in Anton's communication channels; Stan Wang for design verification and validation; Amos Even for an array of system software tools; Kevin Bowers for analysis of factors determining communication latency on commodity clusters; and Rebecca Kastleman and Mollie Kirk for editorial assistance. This article is based on previous work.[7]

References

1. A. Bhatelé et al., ''Overcoming Scaling Challenges in Biomolecular Simulations Across Multiple Platforms,'' *Proc. IEEE Int'l Symp. Parallel and Distributed Processing,* IEEE Press, 2008, doi:10.1109/IPDPS.2008.4536317.
2. K.J. Bowers et al., ''Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters,'' *Proc. ACM/IEEE Conf. Supercomputing* (SC 06), IEEE, 2006, doi:10.1145/1188455.1188544.
3. B.G. Fitch et al., ''Blue Matter: Approaching the Limits of Concurrency for Classical Molecular Dynamics,'' *Proc. 2006 ACM/IEEE Conf. Supercomputing* (SC 06), ACM Press, 2006, doi:10.1145/1188455.1188547.
4. T. Narumi et al., ''A 55 TFLOPS Simulation of Amyloid-Forming Peptides from Yeast Prion Sup35 with the Special-Purpose Computer System MDGRAPE-3,'' *Proc.*

ACM/IEEE Conf. Supercomputing (SC 06), ACM Press, 2006, doi:10.1145/1188455.1188506.

5. J.C. Phillips, J.E. Stone, and K. Schulten, ''Adapting a Message-Driven Parallel Application to GPU-Accelerated Clusters,'' *Proc. ACM/IEEE Conf. Supercomputing* (SC 08), IEEE Press, 2008, no. 8.
6. D.E. Shaw et al., ''Millisecond-Scale Molecular Dynamics Simulations on Anton,'' *Proc. Conf. High Performance Computing Networking, Storage and Analysis,* ACM Press, 2009, doi:10.1145/1654059.1654099.
7. R.O. Dror et al., ''Exploiting 162-Nanosecond End-to-End Communication Latency on Anton,'' *Proc. 2010 ACM/IEEE Int'l Conf. High Performance Computing, Networking, Storage and Analysis,* IEEE CS Press, 2010, doi:10.1109/SC.2010.23.
8. E. Chow et al., *Desmond Performance on a Cluster of Multicore Processors,* tech. report DESRES/TR-2008-01, D.E. Shaw Research, 2008.
9. D. Kerbyson et al., ''Performance Evaluation of an EV7 AlphaServer Machine,'' *Int'l J. High Performance Computing Applications,* vol. 18, no. 2, 2004, pp. 199-209.
10. R. Fatoohi, S. Saini, and R. Ciotti, ''Interconnect Performance Evaluation of SGI Altix 3700 BX2, Cray X1, Cray Opteron Cluster, and Dell PowerEdge,'' *Proc. 20th Int'l Parallel and Distributed Processing Symp.* (IPDPS 06), IEEE Press, 2006, doi:10.1109/SC.2005.11.
11. J. Beecroft et al., ''QsNetII: Defining High-Performance Network Design,'' *IEEE Micro,* vol. 25, no. 4, 2005, pp. 34-47.
12. R. Biswas et al., ''An Application-Based Performance Characterization of the Columbia Supercomputer,'' *Proc. ACM/IEEE Conf. Supercomputing* (SC 05), IEEE CS Press, 2005, doi:10.1109/SC.2005.11.
13. M.D. Noakes, D.A. Wallach, and W.J. Dally, ''The J-Machine Multicomputer: An Architectural Evaluation,'' *ACM SIGARCH Computer Architecture News,* vol. 21, no. 2, 1993, pp. 224-235.
14. S. Kumar et al., ''The Deep Computing Messaging Framework: Generalized Scalable Message Passing on the Blue Gene/P Supercomputer,'' *Proc. 22nd Ann. Int'l Conf. Supercomputing* (ICS 08), ACM Press, 2008, pp. 94-103.
15. K.J. Barker et al., ''Entering the Petaflop Era: The Architecture and Performance of Roadrunner,'' *Proc. ACM/IEEE Conf. Supercomputing* (SC 08), IEEE Press, 2008, no. 1.
16. S.L. Scott, ''Synchronization and Communication in the T3E Multiprocessor,'' *Proc. 7th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* ACM Press, 1996, pp. 26-36.
17. A. Hoisie et al., ''A Performance Comparison through Benchmarking and Modeling of Three Leading Supercomputers: Blue Gene/L, Red Storm, and Purple,'' *Proc. 2006 ACM/IEEE Conf. Supercomputing* (SC 06), ACM Press, 2006, doi:10.1145/1188455.1188534.
18. S. Plimpton, ''Fast Parallel Algorithms for Short-Range Molecular Dynamics,'' *J. Computational Physics,* vol. 117, no. 1, 1995, pp. 1-19.
19. G. Almási et al., ''Optimization of MPI Collective Communication on BlueGene/L Systems,'' *Proc. 19th Ann. Int'l Conf. Supercomputing* (ICS 05), ACM Press, 2005, pp. 253-262.

Ron O. Dror has managed and contributed to various research projects involving algorithms, computer architecture, and structural biology since joining D.E. Shaw Research as its first hire. Dror has a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology.

J.P. Grossman works on architecture, design, simulation, and implementation for Anton at D.E. Shaw Research. Grossman has a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology.

Kenneth M. Mackenzie works on architectural simulation and software for Anton at D.E. Shaw Research. Mackenzie has a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology.

Brian Towles works on architecture, design, and implementation for Anton at D.E. Shaw Research. Towles has a PhD in electrical engineering from Stanford University.

Edmond Chow previously worked on software for Anton at D.E. Shaw Research. He is an associate professor at the Georgia Institute of Technology. Chow has a PhD in computer science from the University of Minnesota.

John K. Salmon is involved in algorithm design, software development, and computer architecture for Anton at D.E. Shaw Research. Salmon has a PhD in physics from the California Institute of Technology.

Cliff Young works on architectural simulation, architecture design, and software for Anton at D.E. Shaw Research. Young has a PhD in computer science from Harvard University.

Joseph A. Bank is involved in architectural simulation, architecture design, and software for Anton at D.E. Shaw Research. Bank has an MEng in electrical engineering and computer science from the Massachusetts Institute of Technology.

Brannon Batson works on hardware architecture, design, and implementation for Anton at D.E. Shaw Research. Batson has an MS in electrical engineering from Purdue University.

Martin M. Deneroff previously led the hardware design and engineering team at D.E. Shaw Research. Deneroff has a BS in electrical engineering from the Massachusetts Institute of Technology.

Jeffrey S. Kuskin works on hardware architecture, design, and implementation for Anton at D.E. Shaw Research. Kuskin has a PhD in electrical engineering from Stanford University.

Richard H. Larson works on hardware architecture, design, and implementation for Anton at D.E. Shaw Research. Larson has a BS in electrical engineering from the University of Illinois at Urbana-Champaign.

Mark A. Moraes heads the Anton software development and systems teams at D.E. Shaw Research. Moraes has an MASc in electrical engineering from the University of Toronto.

David E. Shaw serves as chief scientist at D.E. Shaw Research, where he works on the development of new algorithms and machine architectures for high-speed biomolecular simulations and on the application of such simulations to basic scientific research and computer-assisted drug design. He is also a senior research fellow at the Center for Computational Biology and Bioinformatics at Columbia University. Shaw has a PhD from Stanford University.

Direct questions and comments about this article to David E. Shaw, D.E. Shaw Research, New York, NY 10036; David.Shaw@DEShawResearch.com.