Exploiting 162-Nanosecond End-to-End Communication Latency on Anton

Conference Paper · November 2010

DOI: 10.1109/SC.2010.23 · Source: DBLP

14 authors, including:

J.P. Grossman
Citadel Securities
63 PUBLICATIONS **3,619** CITATIONS
SEE PROFILE

Brian Towles
D. E. Shaw Research
59 PUBLICATIONS **11,411** CITATIONS
SEE PROFILE

John Salmon
D. E. Shaw Research
93 PUBLICATIONS **10,149** CITATIONS
SEE PROFILE

Joseph A Bank
D. E. Shaw Research
23 PUBLICATIONS **2,618** CITATIONS
SEE PROFILE

Exploiting 162-Nanosecond End-to-End Communication Latency on Anton

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, and David E. Shaw*

D. E. Shaw Research, New York, NY 10036, USA

***Abstract*—Strong scaling of scientific applications on parallel architectures is increasingly limited by communication latency. This paper describes the techniques used to mitigate latency in Anton, a massively parallel special-purpose machine that accelerates molecular dynamics (MD) simulations by orders of magnitude compared with the previous state of the art. Achieving this speedup required a combination of hardware mechanisms and software constructs to reduce network latency, sender and receiver overhead, and synchronization costs. Key elements of Anton's approach, in addition to tightly integrated communication hardware, include formulating data transfer in terms of *counted remote writes*, leveraging fine-grained communication, and establishing fixed, optimized communication patterns. Anton delivers software-to-software inter-node latency significantly lower than any other large-scale parallel machine, and the total critical-path communication time for an Anton MD simulation is less than 4% that of the next fastest MD platform.**

I. Introduction

In recent years, the rise of many-core architectures such as general-purpose graphics processing units (GPUs), the Cell BE, 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 software-to-software communication latency—the time elapsed between a software-initiated send of a short message on one node and the message's availability to software on another node—have not kept pace. As a result, the extent to which tightly coupled scientific applications can be parallelized across many nodes, and hence the maximum achievable performance for a given problem size, is increasingly limited by inter-node communication latency.

Classical molecular dynamics (MD) simulations, which model molecular systems at an atomic level of detail, become particularly sensitive to communication latency at high levels of parallelism due to the need for frequent inter-node communication. MD is uniquely suited to capture the behavior of biomolecules such as proteins and nucleic acids, but billions or trillions of sequential simulation steps are required to reach the timescales on which many of the most biologically important events occur. The time required to execute a single step is thus a critical determinant of the range of applicability of this technique. As a result, a great deal of effort has gone into accelerating MD both through parallelization across nodes [4, 9, 12, 20, 24] and through the use of GPUs and other multi-core architectures [6, 17, 21, 23, 29, 31, 34]. While a single GPU can perform a simulation significantly faster than a single traditional CPU, the highest simulation speeds attained on GPU clusters for a given chemical system size remain short of those achieved on CPU clusters, because the communication latency between GPUs is currently greater than that between CPUs [15, 34]. More generally, the maximum simulation speed achievable at high parallelism depends more on inter-node communication latency than on single-node compute throughput.

Anton, a massively parallel special-purpose supercomputer that dramatically accelerates MD simulations, has performed all-atom simulations 100 times longer than any reported previously [40]. Anton's speedup is attributable in part to the use of specialized hardware that greatly accelerates the arithmetic computation required for an MD simulation [41]. Without a corresponding reduction in delays caused by latency, however, Anton would deliver only a modest improvement in performance over other highly parallel platforms.

Throughout this paper, we use the term "latency" in a broad sense to refer to several activities associated with inter-node communication that increase the amount of time between useful computation. The raw hardware network latency is one component, but a more useful metric is end-to-end (or software-to-software) latency: the time from when a source initiates a small inter-node message to when the destination has received the message and can perform useful computation on its contents. Anton's smallest end-to-end inter-node latency is 162 nanoseconds—significantly lower than any other large-scale parallel machine of which we are aware. Latency also encompasses any additional communication needed for synchronization, as well as intra-node data marshalling required to form or process messages. Finally, the latency of a collective operation, such as broadcast or all-reduce, is the total time from the initiation of the operation to its completion on all destination nodes. In this paper, we describe the hardware and software mechanisms used on Anton to reduce each of these contributions to overall latency and, where possible, to remove latency from the critical path by overlapping communication with computation.

We begin by detailing the elements of Anton's hardware architecture relevant to the inter-node communication of an MD simulation. We then describe the software and algorithmic approaches used to efficiently map the MD dataflow onto this hardware architecture, including the formulation of almost all of Anton's communication in terms of *counted remote writes*, a paradigm that embeds synchronization within communication with minimal receiver overhead. We provide measurements characterizing Anton's performance both on low-level communication benchmarks and on the communication patterns used for MD.

* David E. Shaw is also with the Center for Computational Biology and Bioinformatics, Columbia University, New York, NY 10032. E-mail correspondence: David.Shaw@DEShawResearch.com.

SC10 November 2010, New Orleans, Louisiana, USA 978-1-4244-7558-2/10/$26.00.

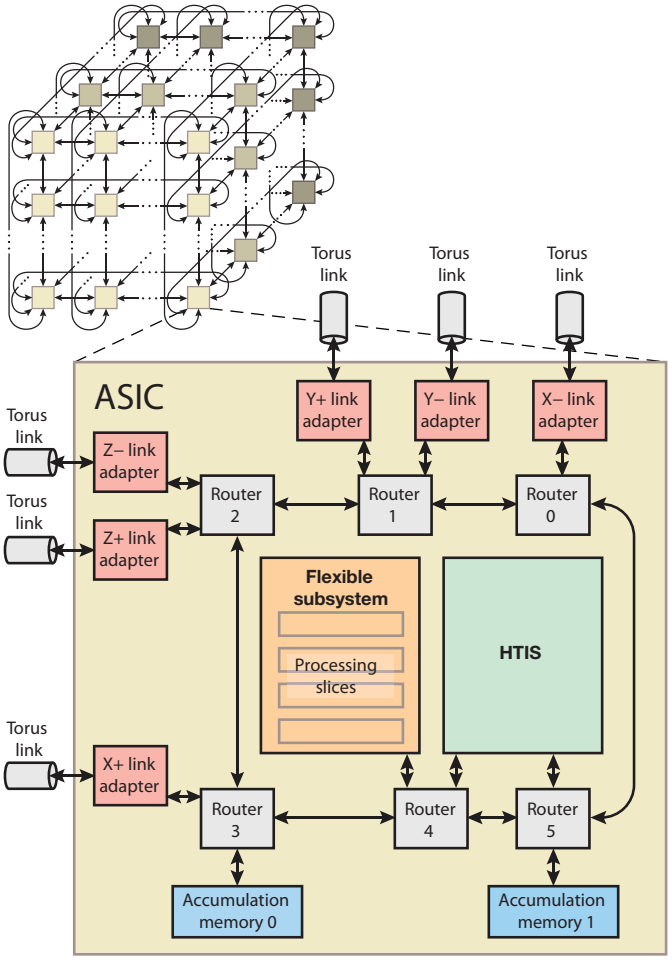

Figure 1. Connectivity of an Anton ASIC. Six on-chip routers form a ring with connections to the computational subsystems, accumulation memories, and inter-node link adapters. Corresponding link adapters on neighboring ASICs are directly connected via passive links to form a three-dimensional torus.

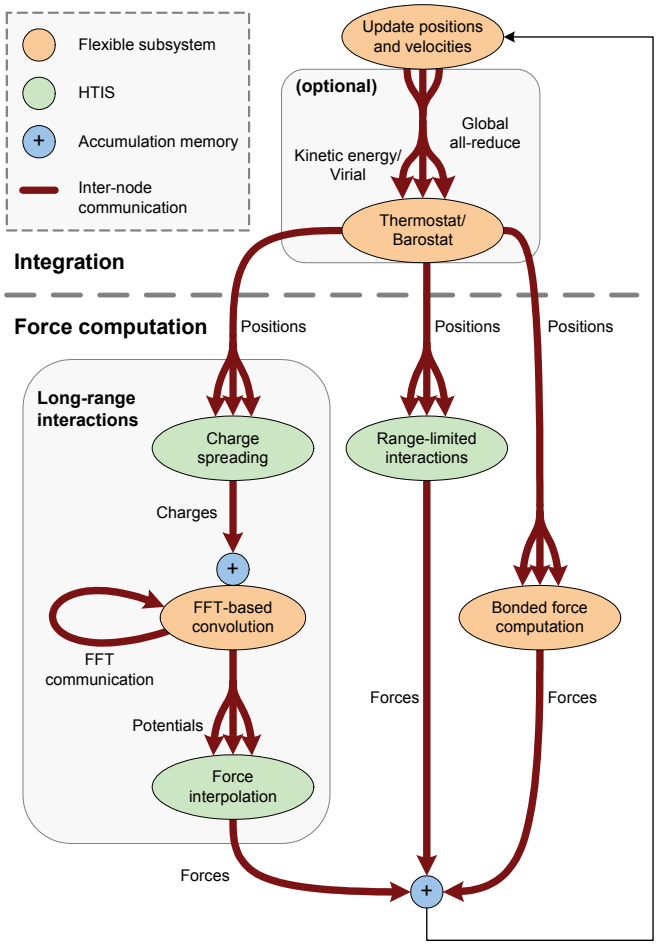

Figure 2. Dataflow of an MD simulation on Anton. Forked arrows indicate communication steps in which each piece of data is sent to multiple recipients; for global all-reduce, the merged arrows indicate that data from multiple sources is combined to produce a result on all nodes. The all-reduce step is omitted for simulations with no temperature or pressure control.

The total communication time on Anton's critical path is less than 4% that of the next fastest platform for MD, an advantage attributable mostly to a reduction in delays associated with latency. As computational density increases, latency will limit the performance of an increasing number of parallel applications, and the combined hardware/software techniques that mitigate the effects of latency on Anton may gain wider applicability.

II. Dataflow of a Molecular Dynamics Simulation

This section reviews the basics of MD simulation and of the Anton architecture, highlighting those portions of the MD computation that require inter-node communication on Anton. An MD simulation steps through time, alternating between a *force calculation* phase that determines the force acting on each simulated atom, and an *integration* phase that advances the positions and velocities of the atoms according to the classical laws of motion. On Anton, the entire computation is mapped to a set of identical MD-specific ASICs, which are connected to form a three-dimensional torus (i.e., a three-dimensional mesh that wraps around in each dimension; Figure 1). The chemical system to be simulated—which might consist, for example, of a protein surrounded by water—is divided into a regular grid of boxes, with each box assigned to one ASIC.

The total force on each atom is computed as a sum of *bonded forces*, which involve interactions between small groups of atoms connected by one or more covalent bonds, and *non-bonded forces*, which include interactions between all pairs of atoms in the system. For purposes of efficient computation, Anton expresses the non-bonded forces as a sum of *range-limited interactions* and *long-range interactions*. The range-limited interactions, involving van der Waals interactions and a contribution from electrostatic interactions, decay rapidly with distance and are thus computed directly for all atom pairs separated by less than some cutoff radius. The long-range interactions, which represent the remaining electrostatic contribution, decay more slowly. They can be computed efficiently, however, by taking the fast Fourier transform (FFT) of the charge distribution on a regular grid, multiplying by an appropriate function in Fourier space, and then performing an inverse FFT to compute the electrostatic potential at each grid point [39]. Charge must be mapped from atoms to nearby grid points before the FFT computation (*charge spreading*), and forces on atoms must be calculated from the potentials at nearby grid points after the inverse FFT computation (*force interpolation*).

Each Anton ASIC contains two major computational subsystems. The *high-throughput interaction subsystem* (HTIS), which includes specialized hardwired pipelines for pairwise interactions, computes the range-limited interactions and performs charge spreading and force interpolation. The *flexible subsystem*, which contains programmable cores, is used for the remaining parts of an MD simulation,

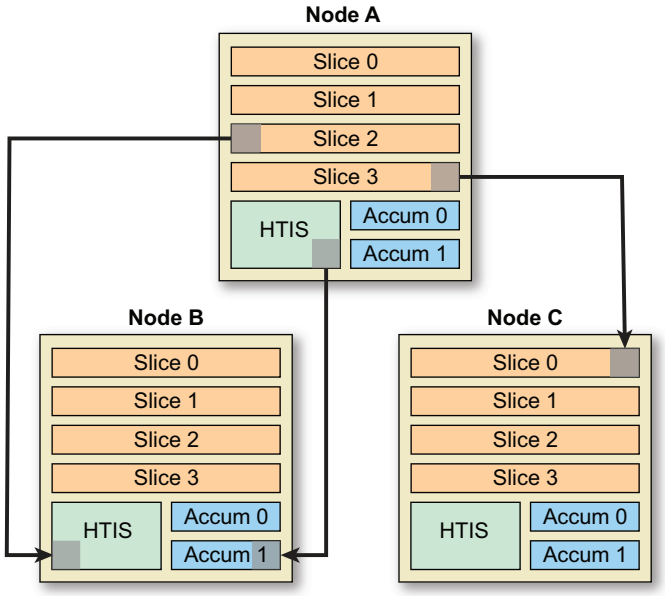

Figure 3. Example of write packets being sent from various network clients on one node directly to the local memories of clients on other nodes. Each node contains seven local memories: a memory associated with each of the four processing slices, a memory associated with the HTIS, and. two accumulation memories.

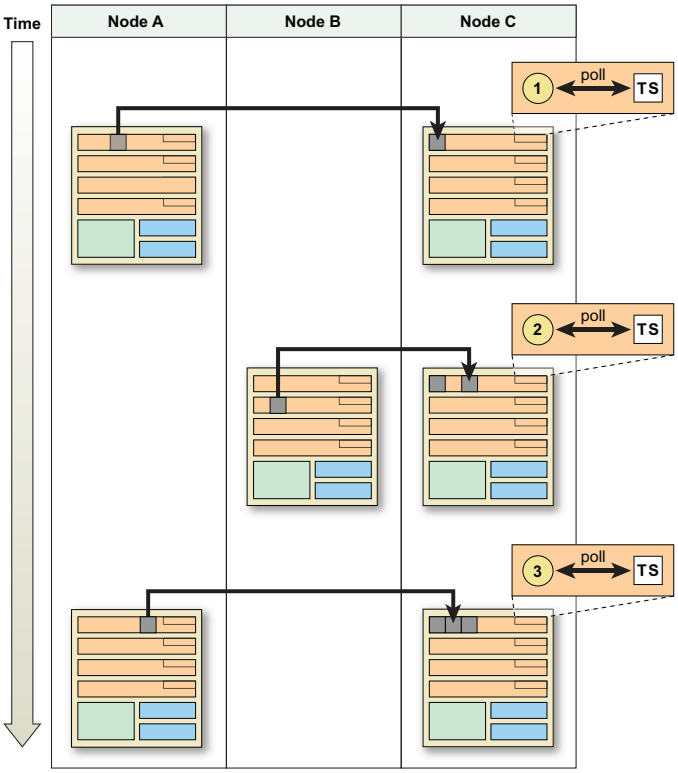

Figure 4. Two source nodes (A and B) use counted remote writes to send data directly to the local memory of a particular processing slice on a target node (C). The packets identify one of the processing slice's synchronization counters, which is incremented as each packet arrives. The Tensilica core (TS) within the processing slice polls the synchronization counter to determine when the expected number of packets has arrived. In this example, the sources and destinations are all processing slices; more generally, the destination may be a processing slice, an HTIS, or an accumulation memory, and the sources may be any combination of local or remote processing slices or HTIS units.

including evaluation of bonded forces, FFT-based convolution, and integration. Each ASIC also includes two accumulation memories, which are used to sum forces and charges, and a six-router ring network for intra-node communication with connections to the inter-node torus network links (Figure 1). Each ASIC constitutes an Anton node.

Figure 2 shows the dataflow of an MD simulation on Anton, including the major computational components and the inter-node communication they require. During integration, each node updates the positions and velocities of the atoms in one spatial region; this region is referred to as the *home box* of that node, and the node is referred to as the *home node* of those atoms. In simulations with a thermostat (temperature control) or barostat (pressure control), global all-reduce operations are required to compute the kinetic energy or virial, which are used to modulate the temperature or pressure (respectively) of the simulated system by adjusting atom velocities or positions. Computation of all three force components—long-range interactions, range-limited interactions, and bonded forces—requires that each node send position information for atoms in its home box to other nodes, and that each node return computed forces to the home nodes of the atoms on which the forces act. Evaluation of long-range interactions requires additional communication steps for the FFT-based convolution and for sending potentials to the HTIS units.

Previously we have described the specialized hardware Anton uses to accelerate the computational components of an MD simulation [27, 28]. Equally important, however, is the acceleration of the communication shown in Figure 2.

III. Communication Architecture and Counted Remote Writes

Anton's communication architecture is designed to provide low latency, small sender and receiver overheads, small synchronization overhead, and support for efficient fine-grained messages. Three distinct types of clients are connected to the Anton network: the HTIS units, the accumulation memories, and *processing slices* within the flexible subsystems. A flexible subsystem contains four processing slices, each of which consists of one Tensilica core, used primarily for communication and synchronization, and two geometry cores, which perform the bulk of the numerical computation. Each network client contains a local memory that can directly accept write packets issued by other clients (Figure 3).

A. Packets and Routing

Both the processing slices and the HTIS units have hardware support for quickly assembling packets and injecting them into the network. Packets contain 32 bytes of header and 0 to 256 bytes of payload; for writes of up to 8 bytes, the data can be transported directly in the header. The accumulation memories cannot send packets, but they accept a special accumulation packet that adds its payload (in 4-byte quantities) to the current value stored at the targeted address.

The inter-node network forms a three-dimensional torus, with each node directly connected to its six immediate neighbors by six 50.6 Gbit/s-per-direction links (one per neighbor); each of these links has an effective data bandwidth of 36.8 Gbit/s in each direction. Nodes are identified by their Cartesian coordinates within this torus; for unicast (single-destination) packets, the node ID is part of a packet's destination address, allowing applications to be designed to maximize three-dimensional spatial locality. Packet routing through the network is lossless and deadlock-free. The network does not, in general, preserve packet ordering, but a software-controlled header flag

can be used to selectively guarantee in-order delivery between fixed source-destination pairs.

The network supports a powerful multicast mechanism that allows a single packet to be sent to an arbitrary set of local or remote destination clients. When a multicast packet is injected into the network or arrives at a node, a table lookup is used to determine the set of local clients and outgoing network links to which the packet should be forwarded. Up to 256 multicast patterns per node can be precomputed and programmed into the multicast lookup tables. Using multicast significantly reduces both sender overhead and network bandwidth for data that must be sent to multiple destinations.

B. *Counted Remote Writes*

Every network client contains a set of synchronization counters, each of which can be used to determine when a pre-determined number of packets has been received. Write and accumulation packets are labeled with 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. The processing slices and HTIS units are able to directly poll their local synchronization counters, resulting in very low polling latencies. The accumulation memory counters are polled by processing slices on the same node across the on-chip network, and thus incur larger polling latencies.

These synchronization counters form the basis of a key communication paradigm on Anton that we call *counted remote writes*. 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. Figure 4 illustrates counted remote writes for an example in which the sources and target are all processing slices.

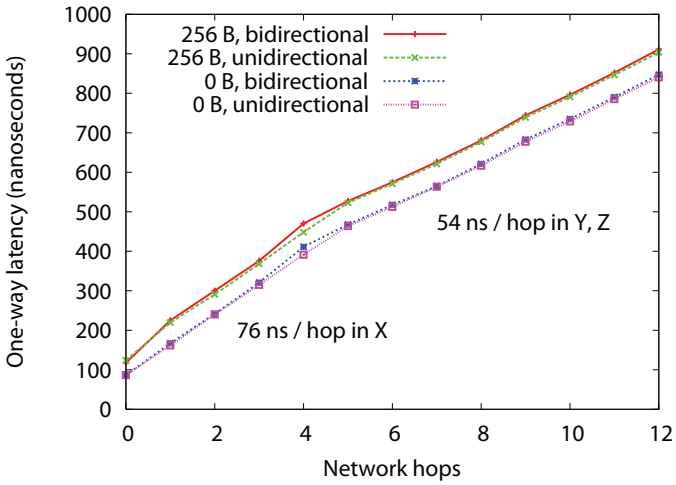

Figure 5. Measurement of one-way counted remote write latency vs. network hops in a 512-node Anton machine. The zero-hop case sends messages between processing slices on the same node. The one- through four-hop cases travel along the X dimension of the torus network; the five- through twelve-hop cases add hops along the Y and Z torus dimensions. The X hops traverse more on-chip routers per node and thus have a higher latency than the Y or Z hops. Twelve hops is the maximum distance between two nodes in an 8×8×8 configuration (shortest-path routing is used along each torus dimension). The latency of a zero-byte message between neighboring nodes along the X dimension is 162 ns.

C. *Message FIFO*

To handle cases where it is difficult or impossible to formulate communication in terms of counted remote writes, each processing slice also contains a hardware-managed circular FIFO within its local memory that can receive arbitrary network messages. The Tensilica core polls the tail pointer to determine when a new message has arrived and advances the head pointer to indicate that a message has been processed. If the FIFO fills then backpressure is exerted into

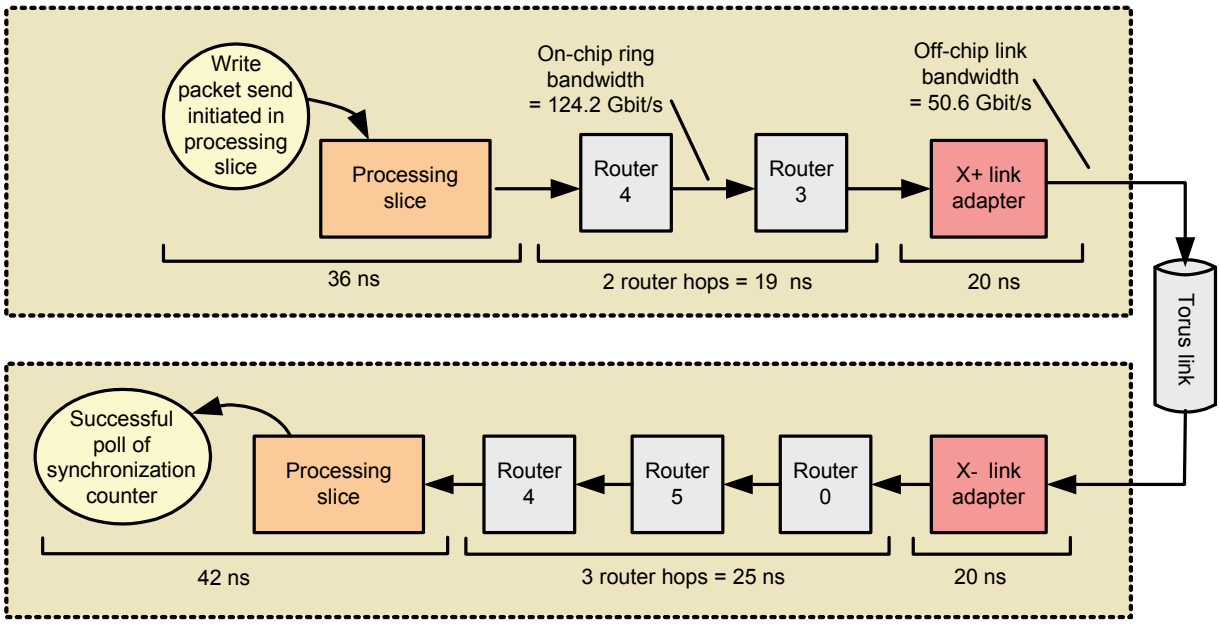

Figure 6. Detailed breakdown of counted remote write latency between processing slices on neighboring nodes along the X dimension of the torus. The wire delay of the passive torus link has been incorporated into the latency shown for the link adapters. The wire delay is up to 4 ns for X-dimension links, up to 8 ns for Y-dimension links, and up to 10 ns for Z-dimension links.

the network; software is responsible for polling the FIFO and processing messages to avoid deadlock conditions.

D. Performance

The combination of efficient packet creation and local polling of a single synchronization counter at the target client results in very low end-to-end message latencies. Figure 5 plots latency as a function of message size and the number of hops in the torus network, measured by both unidirectional and bidirectional ping-pong tests. Figure 6 shows how single-hop latency is broken down among various hardware components as a message travels from source to destination. The total software-to-software latency between two different Anton nodes—from the time a client on the source node issues a send instruction to the time the target node successfully polls the synchronization counter to determine that the message has arrived—is as low as 162 ns. This is significantly lower than the fastest published measurements of which we are aware for inter-node software-to-software latency across a scalable network on any other hardware (Table 1).

Due to the low overhead of message creation and injection, sending many fine-grained messages on Anton's network is nearly as efficient as sending fewer, large messages. This is in contrast to a typical cluster interconnect where latencies grow rapidly as a function of the number of messages, driving software for such clusters to be carefully structured so as to minimize the total message count. Figure 7 shows the modest increase in communication time on Anton as a 2-KB data transfer is divided into many smaller messages, compared to the larger increase in latency over an InfiniBand network. In addition, fine-grained messages are able to make effective use of Anton's network bandwidth: 50% of the maximum possible data bandwidth is achieved with 28-byte messages on Anton, compared with 1.4-, 16-, and 39-*kilobyte* messages on Blue Gene/L, Red Storm, and ASC Purple, respectively [25].

These measurements demonstrate that counted remote writes on Anton provide low latency, low overheads, and efficient support for fine-grained messages. The utility of counted remote writes, however, depends on the ability of the distributed clients to agree on the synchronization counter identifiers, as well as the receiver's ability to pre-compute the total expected number of packets. In the following section we will describe how MD software can be structured so that almost all of its communication meets these requirements.

Machine name	Latency (μs)	Date	Ref.
Anton	0.16	2009	here
Altix 3700 BX2	1.25	2006	[18]
QsNet^{II}	1.28	2005	[8]
Columbia	1.6	2005	[10]
Sun Fire	1.7	2002	[42]
EV7	1.7	2002	[26]
J-Machine	1.8	1993	[32]
QsNET	1.9	2001	[33]
Roadrunner (InfiniBand)	2.16	2008	[7]
Cray T3E	2.75	1996	[37]
Blue Gene/P	2.75	2008	[3]
Blue Gene/L	2.8	2005	[25]
ASC Purple	4.4	2005	[25]
Cray XT4	4.5	2007	[2]
Red Storm	6.9	2005	[25]
SR8000	9.9	2001	[45]

Table 1. Survey of published inter-node software-to-software (ping-pong) latency measurements. This survey excludes measurements of *intra*-node communication that does not traverse a scalable network; it also excludes measurements of one-sided writes that do not include the time required by software on the target node to determine that a message has arrived. The fastest previously published measurement of which we are aware is 1.25 μs, achieved by the SGI Altix 3700 BX2 with NUMAlink4 interconnect [18]. Comparable measurements for Cray's recently released XE6 system have not been published, but are estimated to be around one microsecond (private communication, [38]). In the absence of a scalable switch, a pair of nodes connected back-to-back via InfiniBand network cards can achieve a latency of approximately 1.1 μs [44]. An extrapolation from remote cache access times in scalable shared memory machines such as the EV7 [26] 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.

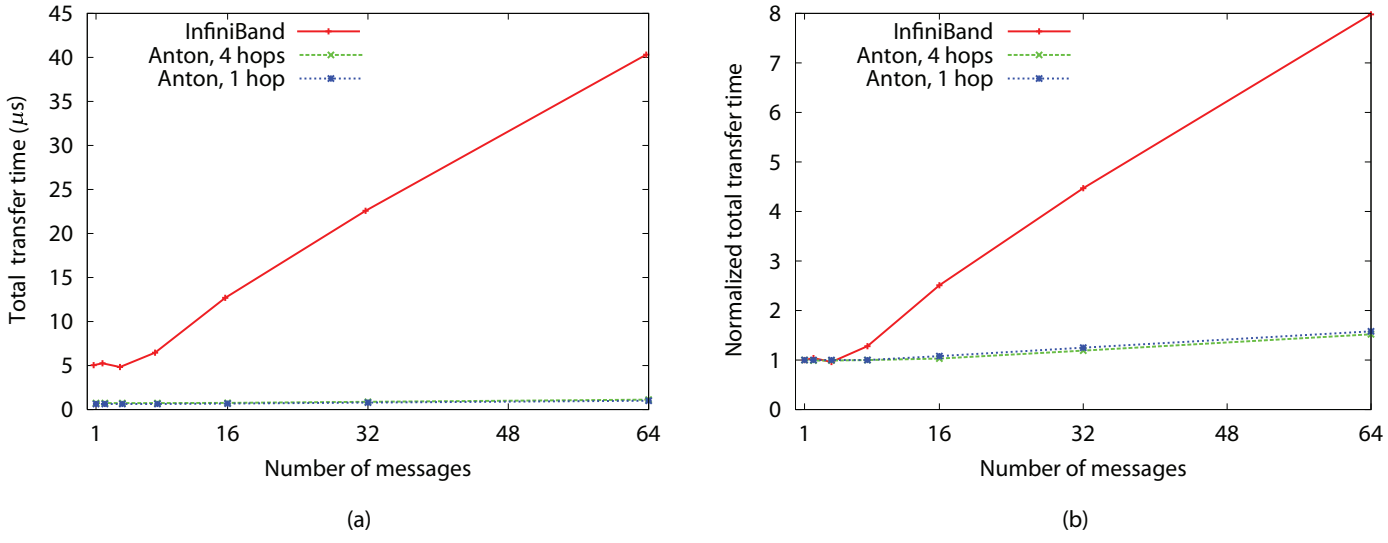

Figure 7. Measurement of total time required to transfer 2 KB of data between nodes as a function of the number of messages used. On many networks, the number of messages has a much larger impact than it does on Anton; here, corresponding figures for a DDR2 InfiniBand network are shown for comparison. Panel (a) shows time on an absolute scale, while in (b), each curve has been normalized to show time in multiples of transfer time for a single 2-KB message.

IV. Mapping the MD Dataflow onto Anton

In order to minimize the impact of communication latency on Anton, we carefully organized the communication in software. We highlight several principles motivating the software design, then discuss how they were applied in implementing each of the major communication pathways on Anton.

A. Design Principles for Reducing the Impact of Communication Latency in Anton

Fix communication patterns so that a sender can push data directly to its destination

A network client on one node can send data directly to the local memory of any other client on any other node (Figure 3). Anton's software leverages this mechanism, relying almost entirely on a choreographed data flow in which a sender pushes data directly to its destination. To avoid additional communication steps, the sender must be able to determine the exact address to which each piece of data should be written. We therefore pre-allocate 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 the course of a simulation.

Encode synchronization within communication using a counting mechanism

One-sided communication requires that a receiver be able to determine when all data required for a given computation has arrived. In Anton, this is typically accomplished by polling a synchronization counter that counts arriving packets (Figure 4). This use of counted remote writes avoids the need to perform additional communication for receiver synchronization, but requires that the receiver know exactly how many packets of a given type to expect. Thus, we strive to fix not only communication patterns, but also the total number of packets sent to each receiver.

Rely on dataflow dependencies to determine when destination buffers are available

Another requirement of one-sided communication is that a sender must be able to determine when the destination buffer is available. For the majority of its communication, Anton is able to make this determination with no additional communication by carefully exploiting dependencies inherent to the MD dataflow. Before a node can send a new atom position to a destination node for force computation, for example, it must receive all forces on the atom from the previous time step, which in turn implies that the destination node has finished processing the atom's previous position and is ready to receive the new position.

Exploit fine-grained communication to reduce data marshalling costs and better overlap communication with computation

On many scalable networks, sending a large number of small messages is much more expensive than sending one large message, whereas on Anton, the difference tends to be small (Figure 7). As a result, Anton can exploit communication patterns that require more messages in order to avoid extra communication rounds or additional processor work in preparing data for transmission. Instead of employing a staged 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 [12, 22, 35]—Anton is able to benefit from the lower latency of a single round of communication in which every node sends many messages directly to their final intended recipients (Figure 8a). Anton's software is also able to avoid local copy or permutation operations by sending additional messages, effectively performing these operations in the network (Figure 8b).

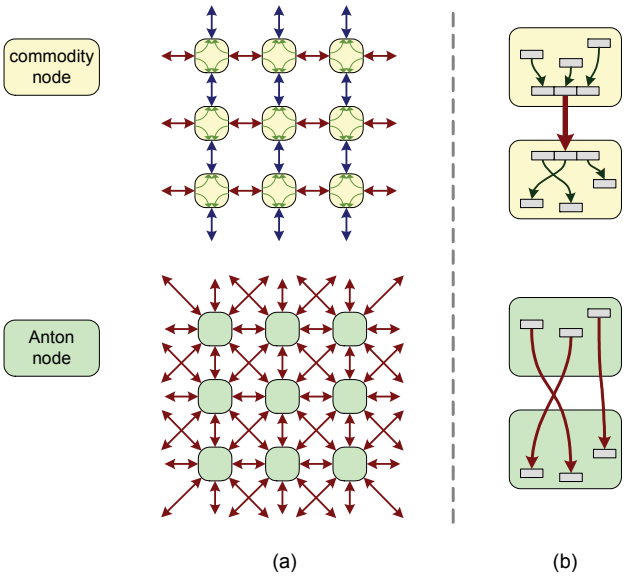

Figure 8. Exploiting fine-grained communication on Anton. (a) Pairwise all-neighbor data exchange, shown in two dimensions. Staged communication (horizontal first, vertical second, with data forwarded from the first stage to the second) is preferable for a commodity cluster in order to reduce the message count; in three dimensions, this technique allows every node to exchange data with its 26 neighbors in three stages using only six messages per node (e.g., [12]). Staging is undesirable in Anton, where performance is optimized with a single round of direct communication. (b) Local sender/receiver data copies are often used to avoid multiple messages between commodity nodes. Direct remote writes eliminate this copy overhead in Anton.

The use of fine-grained communication also allows Anton to 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 ones are available, so that the remainder of the computation is overlapped with communication.

Minimize the number of network hops required for communication

On Anton, the total transmission time of a message is typically dominated by the number of network hops required: communication between the two most distant nodes in an 8×8×8 node machine has a latency five times higher than communication between neighboring nodes (Figure 5). We thus strive to minimize the number of network hops required for most messages. The choice of a toroidal network topology ensures that most communication is local (Anton simulations typically employ periodic boundary conditions, in which atoms on one side of the simulated system interact with atoms on the other side). We also choose parallelization algorithms that reduce the number of sequential network hops, even when those algorithms require sending a larger number of messages. Finally, we attempt to identify the messages requiring the largest number of network hops and to arrange the software so that the latency of these messages is partially hidden behind computation.

B. MD Communication Patterns on Anton

There are four major classes of communication shown in Figure 2: communication with the HTIS units (which requires sending data to multiple HTIS units and returning results for accumulation), bonded force communication (which requires sending atom positions to the appropriate locations for bonded force computation and returning these forces for accumulation), FFT communication, and global all-reduce operations. In addition, as the system evolves, atoms that move beyond their home boxes must be migrated to neighboring nodes. Migration reflects the spatial organization of the MD compu-

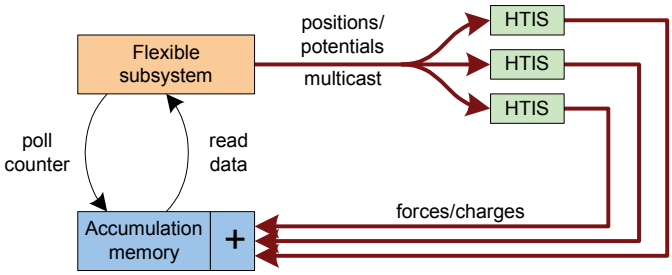

Figure 9. HTIS communication. Positions and potentials are multicast to the HTIS units. Computed forces and charges are returned to the accumulation memories. The flexible subsystem polls the synchronization counter and then retrieves the data.

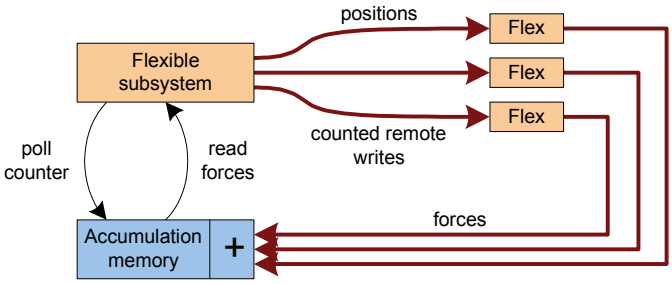

Figure 10. Bond term communication. Positions are unicast to bond destinations. Computed forces are returned to accumulation memories and accumulated with HTIS forces.

tation within Anton and is not captured within the dataflow graph, but it represents an important critical-path communication step.

1) Communication to/from the HTIS

We have chosen a parallelization strategy for the HTIS computation that fixes the communication pattern for both atom positions and grid point potentials. This allows us to use the multicast mechanism to send data to the HTIS units, reducing both sender overhead and bandwidth requirements. These savings are significant, because atom positions are typically broadcast to as many as 17 different HTIS units.

Although the number of atom positions sent to a given HTIS varies due to atom migration, as does the number of forces that the HTIS returns, we fix the number of *packets* communicated, choosing this number to accommodate the worst-case temporal fluctuations in atom density. The number of grid point charge packets generated by the HTIS and the number of grid point potential packets sent to the HTIS are also fixed (trivially so since grid points do not migrate). Fixing the number of packets allows the use of counted remote writes for communication to and from the HTIS, with synchronization embedded in the communication. Synchronization counters in the HTIS count position and potential packets as they arrive, while synchronization counters in the accumulation memory count force and charge packets arriving from HTIS units (Figure 9). In each case, the communication is complete when the appropriate counters have reached their predetermined target values.

The HTIS organizes arriving packets into *buffers* corresponding to the node of origin. For the most part, the order in which the buffers are processed is specified by software running on an embedded controller. A hardware mechanism is also provided to place certain buffers in a high-priority queue so that they can be processed as soon as all of their packets have been received. This high-priority queue is used for the position buffers whose force results must be sent the furthest, allowing the higher latency of these sends to be hidden behind the remaining HTIS computation.

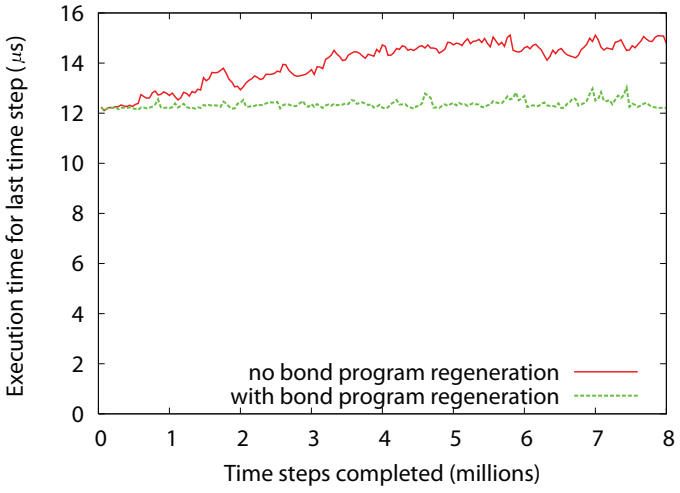

Figure 11. Evolution of time step execution time (lower is better) for a 23,558-atom simulation on an 8×8×8 Anton machine. The upper curve shows performance with no bond program regeneration, while the lower curve shows performance with the bond program regenerated every 120,000 time steps. Bond program regeneration is performed in parallel with the MD simulation, so a bond program is 120,000 time steps out of date when it is installed. Bond program regeneration results in a 14% overall performance improvement for this benchmark.

2) Bonded force communication

On each time step, the atom positions required to compute a given bonded force term must be brought together on one node. We simplify this communication on Anton by statically assigning bonded force terms to nodes, so that the set of destinations for a given atom is fixed. With this strategy we are able to pre-allocate memory on each node to receive incoming positions, and each node knows exactly how many positions it will receive. Atoms can therefore be sent directly to their destinations using fine-grained (one atom per packet) counted remote writes. Bonded forces are sent back to their home node, where they are accumulated along with the HTIS force results (Figure 10). The expected force packet count within an accumulation memory is the total number of expected force packets from all sources.

The assignment of bond terms to nodes (which we refer to as the *bond program*) is chosen to minimize communication latency for the initial placement of atoms, but as the system evolves and atoms migrate this communication latency increases, causing performance to degrade over the course of several hundred thousand time steps. We therefore regenerate the bond program every 100,000–200,000 time steps in order to maintain low communication latencies (Figure 11).

3) FFT communication

The FFT communication patterns are inherently fixed, so they can also be implemented using fine-grained (one grid point per packet) counted remote writes. We have implemented a dimension-ordered FFT, first performing 1D FFTs in the x dimension, then the y dimension, then the z dimension; the inverse FFT is performed in the reverse dimension order. Communication occurs between computation for different dimensions, with per-dimension synchronization counters used to tracking incoming remote writes. The specific assignment of 1D FFTs to nodes defines both the communication pattern and the resulting communication latency. Our FFT communication patterns have been chosen to minimize this latency; see [47] for details.

Number of nodes (torus dimensions)	0-byte reduction (μs)	32-byte reduction (μs)
1024 (8×8×16)	1.56	2.06
512 (8×8×8)	1.32	1.77
256 (8×8×4)	1.27	1.68
128 (8×2×8)	1.24	1.64
64 (4×4×4)	0.96	1.31

Table 2. Global all-reduce times for various Anton configurations. A fast global barrier can be implemented as a 0-byte reduction on Anton, but we avoid global barriers within our MD code by using alternate synchronization methods.

4) Global reductions

Although Anton provides no specific hardware support for global reductions, the combination of multicast and counted remote writes leads to a very fast implementation. We use a dimension-ordered algorithm for global all-reduce operations: the three-dimensional reduction is decomposed into parallel one-dimensional all-reduce operations along the *x*-axis, followed by all-reduce operations along the *y*-axis, and then the *z*-axis. This algorithm, which was also used by QCDOC [13], maps well to a 3D torus network and achieves the minimum total hop count ($3N/2$ for an $N \times N \times N$ machine) with three rounds of communication. By contrast, a radix-2 butterfly communication pattern would require $3\log_2 N$ rounds and $3(N-1)$ hops.

The communication within each dimension is performed by multicasting counted remote writes, with each of N nodes along a dimension broadcasting its data to, and receiving data from, the other $N-1$ nodes. When the data has been received, all N nodes redundantly compute the same sum. Processing slice k receives the remote writes and computes the partial sum for the k^{th} dimension ($k = 0, 1, 2$), so after three rounds slice 2 on each node contains a copy of the global sum, which it shares locally with the other three slices. One could, in principle, perform the partial sums within the accumulation memories, but the overhead of polling the accumulation memory synchronization counters is much larger than the cost of performing the sums in software within the processing slices.

Table 2 lists the latency of a 32-byte all-reduce operation on Anton for several different machine sizes. The 1.77 μs latency for a 512-node Anton represents a 20-fold speedup over the time we measured for the same reduction on a 512-node DDR2 InfiniBand cluster (35.5 μs). Anton's 1.77 μs latency also compares favorably with that of a pure hardware implementation: on BlueGene/L, a 16-byte all-reduce across 512 nodes using the specialized tree-based reduction network has a latency of 4.22 μs [5].

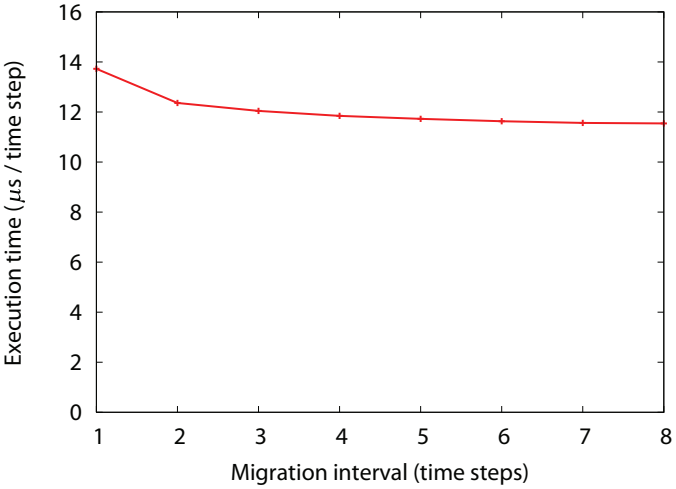

Figure 12. Average execution time (lower is better) for a single time step of a 17,758-particle chemical system on a 512-node Anton machine that performs migrations every N time steps, as N varies from 1 to 8.

5) Migration

Migration is stochastic in nature, and as a result the communication is more difficult to implement efficiently: no node knows in advance how many atoms it will send or receive during a migration phase. This has two negative implications. First, it would be extremely inefficient to pre-allocate memory for all possible migration messages that could be received from all 26 neighbors, so instead migration messages are sent to the hardware message queue, which must be polled and processed by software. Second, an additional synchronization mechanism is required to determine when all migration messages have been received: this is the one instance where we do not directly encode synchronization within the data communication. We implement this additional synchronization by multicasting a counted remote write to all 26 nearest neighbors after all migration messages have been sent, using the network's in-order mechanism to ensure that these writes cannot overtake migration messages within the network. This synchronization step requires 0.56 μs.

As a result of this communication overhead, as well as the additional bookkeeping requirements, migrations are fairly expensive. To reduce the overall performance impact of migration, we relax the spatial boundaries of the home boxes, leveraging the resulting overlap between adjacent home boxes to perform migrations less frequently [40]. Figure 12 shows performance as a function of migration frequency for a benchmark system; reducing the frequency from every time step to every eight time steps results in a 19% performance improvement.

C. Total Communication Time

Table 3 shows the total critical-path communication time during Anton's execution of a typical MD simulation. This communication time was computed by subtracting critical-path arithmetic computation time from total time and thus 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 the *logic analyzer*, an on-chip diagnostic network that allows us to monitor ASIC activity; Figure 13 shows a graphical depiction of the logic analyzer's output. For comparison, Table 3 also lists corresponding timings for the hardware/software configuration that has produced the next fastest reported MD simulations [15]: a high-end 512-node Xeon/InfiniBand cluster running the Desmond MD software [12].

During an average MD time step, Anton's critical-path communication time is approximately 1/27 that of the next fastest MD platform (Table 3). Most modern supercomputers could send and receive only a handful of messages per node during the 12.9-microsecond average duration of an Anton time step; by contrast, the average Anton node sends over 250 messages and receives over 500 messages per time step.

V. Related Work

While we cannot hope to exhaustively list all previously reported hardware and software mechanisms for providing low-latency communication, this section briefly summarizes some of the approaches most relevant to our discussion.

Many architectures have been implemented with hardware support for low-latency remote writes, which are also referred to as *put* operations or one-sided communication, and are one of the important extensions of MPI included in MPI-2 [30]. The Cray T3D and T3E multiprocessors implement global shared memory by providing Alpha microprocessors with memory-mapped access to network

	Anton		Desmond	
	Communication time (μs)	Total time (μs)	Communication time (μs)	Total time (μs)
Average time step	9.8	15.6	262	565
Range-limited time step	5.0	9.0	108	351
Long-range time step	14.6	22.2	416	779
FFT-based convolution	7.5	8.5	230	290
Thermostat	2.6	3.0	78	99

Table 3. Critical-path communication time and total time for simulation of a benchmark system on a 512-node Anton machine and on the fastest MD platform besides Anton: a 512-node Xeon/InfiniBand cluster running the software package Desmond. Long-range interactions and temperature adjustment were performed on every other time step; timings are listed both for time steps that include them (long-range time steps) and for those that do not (range-limited time steps). Timings are also listed for the two most expensive communication steps within a long-range time step (the FFT-based convolution and the global all-reduce required for temperature adjustment). The benchmark system is dihydrofolate reductase (DHFR) with 23,558 atoms; simulation parameters are as specified in [40], except that the MD simulations profiled here included a thermostat. Detailed specifications of the cluster hardware are given in [15].

controllers [37]. The Sun Fire and Quadrics QsNet^{II} interconnects support fast remote writes via programmed I/O [8, 42]. Both the Hitachi SR8000 [45] and QCDOC [13] provide hardware support for remote direct memory access operations. The EV7 AlphaServer and the SGI Altix 3000 both contain specialized networks that directly support cache-coherent shared memory across the nodes of a parallel machine [26, 46]. Remote writes in isolation, however, do not support the design principle of embedding synchronization directly within communication, as separate synchronization is required to inform the receiver that data has arrived.

One approach to integrating synchronization and communication in hardware is to write to a memory-mapped receiver FIFO rather than specific remote memory addresses, so that the arrival of data is indicated by the advancement of the tail pointer. This is implemented in Blue Gene/L [11] and was also proposed for the JNIC Ethernet controller [36]. In the M-Machine, a small receiver FIFO is directly mapped to processor registers [19]. 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 [32]. Anton's counted remote writes are more general and allow a single synchronization event to be associated with the arrival of larger sets of data from multiple locations; this strategy was previously proposed for the Hamlyn interface [14].

A number of architectures provide hardware support for collective

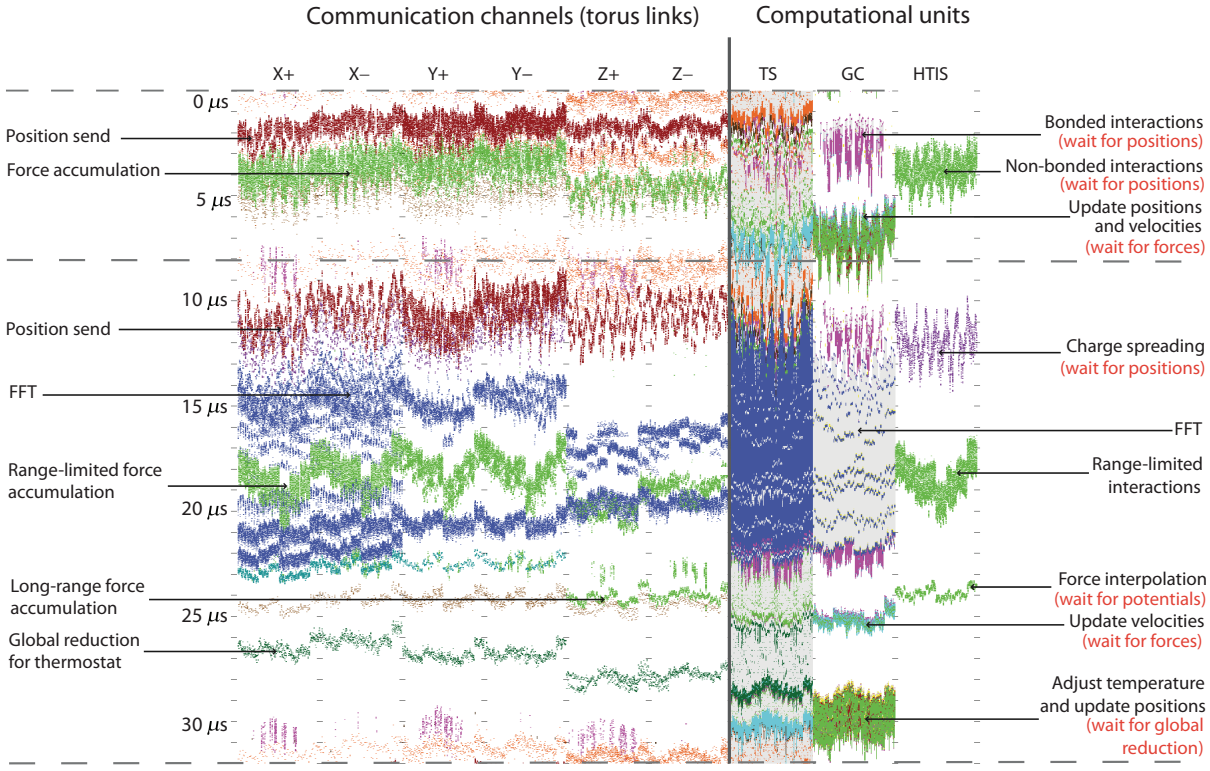

Figure 13. Anton activity for two time steps of a 23,558 atom simulation on a 512-node machine. The first time step (0–8 μs) is a range-limited time step, and the second (8–32 μs) is a long-range time step (see Table 3 caption). The columns on the left show traffic on the 6 inter-node torus links; the columns on the right show activity in the Tensilica cores (TS), geometry cores (GC), and HTIS units. Each column shows activity for all units of the same type across the entire machine. Different colors represent different types of activity, with light gray indicating that a Tensilica core or geometry core is stalled during a computational phase waiting for additional data. The red text on the right-hand side indicates data that must be received before beginning each computational phase. Note that the torus links are occupied for much of the time step, and that the computational units spend a significant amount of time waiting for data.

operations. The Cray T3D includes a specialized four-wire-wide barrier network, while the Cray T3E provides 32 barrier/eureka synchronization units per node that accelerate global and local synchronization operations over the regular interconnect [37]. The SR8000 network supports both broadcasts and barriers involving all nodes [45]. The Quadrics QsNet network supports a limited form of hardware multicast where the set of destination nodes must be contiguous in the global linear order [16], and the QsNet^{II} network has hardware support for both broadcast and barrier operations [8]. Both Blue Gene/L [1] and Blue Gene/P [43] contain two specialized tree networks: one for performing broadcast and global reductions, and the other for global interrupts and barriers. Blue Gene/L supports multicast to a set of consecutive nodes along a single dimension of its three-dimensional torus network [5]; Blue Gene/P extends this with the ability to multicast to a selected "class" of nodes [43]. The QCDOC ASIC contains a Serial Communications Unit that can perform in-network reductions, and supports general multicast patterns in the same manner as Anton: data received from an incoming link can be stored locally and sent to any combination of the outgoing links [13].

VI. Discussion

Anton serves as a case study demonstrating that, through a combination of hardware features and carefully organized software, dramatic reductions can be achieved in the latency-based delays that limit the parallel performance of many scientific applications. Anton's ability to perform all-atom MD simulations that are two orders of magnitude longer than any previously published simulation is due, in no small measure, to a 27-fold reduction in the critical-path communication time relative to the next fastest implementation.

While Anton's hardware is specialized to MD, the combination of hardware and software strategies Anton employs to reduce latency-associated delays may also prove useful in accelerating other parallel applications. In Anton's counted remote write mechanism, for example, each sender pushes data directly to its destination, and the receiver uses a local counter to determine without additional synchronization when data from all senders has arrived. Counted remote writes provide 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 phase of computation. Hardware support for counted remote writes can be implemented efficiently and can dramatically reduce the delays associated with this type of communication.

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. In certain applications, the number of packets that need to be transmitted and/or the destination addresses of these packets cannot be predetermined. Applications based on graph traversal, for example, may defy the use of counted remote writes, because a processor's communication pattern during the next step may only be known after processing the data received during the current step.

In practice, however, even applications whose data structures evolve during a 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 applications based on adaptive mesh refinement, for example, the mesh repartitioning step involves unpredictable communication patterns, but the extensive computational phases between successive refinements typically contain regular patterns of communication. Indeed, parallel MD codes generally contain dynamic data structures, because the set of range-limited interactions to be computed changes from one time step to the next. We formulated Anton's software to ensure that changes in communication patterns occur infrequently, even at the cost of moderate amounts of redundant communication or computation. Other applications may benefit similarly from algorithmic approaches capable of exploiting low-latency communication mechanisms such as counted remote writes.

Anton further reduces the impact of latency by using fine-grained communication to minimize data marshalling costs, avoid staged communication, and better overlap communication with computation. These strategies can be exploited in many parallel applications, but require significant changes to software coupled to changes in network hardware. (Existing software tends to use coarse-grained communication because the per-message overhead on most hardware is much larger than on Anton.) Some of the software changes necessary to take advantage of fine-grained communication would involve simplifications, such as the omission of intermediate data repackaging steps used to reduce message count. Others would tend to make the software more complex; in particular, overlapping computation and communication at a fine-grained level blurs the conceptual division between computation and communication phases, and requires careful software tuning to balance the two.

Finally, Anton takes advantage of inter-time-step data dependencies to determine when destination buffers are available without the use of barriers or library-level handshakes. Many other applications can exploit similar inter-time-step data dependencies. In one time step, for example, a processor computes using received data and then sends the results of its computation to other processors. The processors receiving these new results can infer that the first processor no longer needs the buffers it used to receive data before performing the computation. Iterative algorithms that are not based on time stepping may also have data dependencies that can be exploited in a similar manner. Unlike many of the other latency-reduction strategies we have discussed, this technique does not depend on non-standard hardware support and has been used in certain software packages for commodity hardware (e.g., [12]).

Most of the individual features of Anton's communication hardware have been either implemented or proposed as part of some other parallel machine. By combining these features with carefully designed software and algorithms, however, Anton is able to dramatically reduce delays associated with communication latency in highly parallel MD simulations. As latency limitations become more severe relative to the increasing computational power of modern architectures, the hardware and software techniques Anton uses to circumvent communication bottlenecks may find broader adoption.

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 for editorial assistance.

References

[1] N. R. Adiga, G. Almasi, G. S. Almasi, Y. Aridor, R. Barik, *et al*., "An overview of the BlueGene/L supercomputer," in *Proc. ACM/IEEE Conf. on Supercomputing (SC02)*, Baltimore, MD, Nov. 2002.

[2] S. R. Alam, J. A. Kuehn, R. F. Barrett, J. M. Larkin, M. R. Fahey, *et al*., "Cray XT4: an early evaluation for petascale scientific simulation," in *Proc. ACM/IEEE Conf. on Supercomputing (SC07)*, Reno, NV, Nov. 2007.

[3] S. Alam, R. Barrett, M. Bast, M. R. Fahey, J. Kuehn, *et al*., "Early evaluation of IBM BlueGene/P," in *Proc. ACM/IEEE Conf. on Supercomputing (SC08)*, Austin, TX, Nov. 2008.

[4] F. Allen, G. Almasi, W. Andreoni, D. Beece, B. J. Berne, *et al*., "Blue Gene: a vision for protein science using a petaflop supercomputer," *IBM Systems J.*, vol. 40, no. 2, pp. 310–327, Feb. 2001.

[5] G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, *et al*., "Optimization of MPI collective communication on BlueGene/L systems," in *Proc. 19th Annu. Int. Conf. on Supercomputing*, Cambridge, MA, June 2005.

[6] J. A. Anderson, C. D. Lorenz, and A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units," *J. Comp. Phys.*, vol. 227, no. 10, pp. 5342–5359, May 2008.

[7] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, *et al*., "Entering the petaflop era: the architecture and performance of Roadrunner," in *Proc. ACM/IEEE Conf. on Supercomputing (SC08)*, Austin, TX, Nov. 2008.

[8] J. Beecroft, D. Addison, D. Hewson, M. McLaren, D. Roweth, *et al.*, "QsNet^{II}: defining High-performance network design," *IEEE Micro*, vol. 25, no. 4, pp. 34–47, Aug. 2005.

[9] A. Bhatele, S. Kumar, C. Mei, J. C. Phillips, G. Zheng, and L. V. Kalé, "Overcoming scaling challenges in biomolecular simulations across multiple platforms," in *Proc. IEEE Int. Parallel and Distributed Processing Symp*., Miami, FL, Apr. 2008.

[10] R. Biswas, M. J. Djomehri, R. Hood, H. Jin, C. Kiris, and S. Saini, "An application-based performance characterization of the Columbia supercomputer," in *Proc. ACM/IEEE Conf. on Supercomputing (SC05)*, Seattle, WA, Nov. 2005.

[11] M. Blocksome, C. Archer, T. Inglett, P. McCarthy, M. Mundy, *et al*., "Design and implementation of a one-sided communication interface for the IBM eServer Blue Gene® supercomputer," in *Proc. ACM/IEEE Conf. on Supercomputing (SC06)*, Tampa, FL, Nov. 2006.

[12] K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, *et al*., "Scalable algorithms for molecular dynamics simulations on commodity clusters," in *Proc. ACM/IEEE Conf. on Supercomputing (SC06)*, Tampa, FL, Nov. 2006.

[13] P. A. Boyle, D. Chen, N. H. Christ, M. Clark, S. Cohen, *et al*., "QCDOC: a 10 teraflops computer for tightly-coupled calculations," in *Proc. ACM/IEEE Conf. on Supercomputing (SC04)*, Pittsburgh, PA, Nov. 2004.

[14] G. Buzzard, D. Jacobson, M. Mackey, S. Marovich, and J. Wilkes, "An implementation of the Hamlyn sender-managed interface architecture," in *Proc. 2nd USENIX Symp. on Operating Systems Design and Implementation*, Seattle, WA, Oct. 1996.

[15] E. Chow, C. A. Rendleman, K. J. Bowers, R. O. Dror, D. H. Hughes, *et al*., "Desmond performance on a cluster of multicore processors," D. E. Shaw Research Technical Report DESRES/TR--2008-01, http://www.deshawresearch.com/publications/Desmond%20Performance%20on%20a%20Cluster%20of%20Multicore%20Processors.pdf, July 2008.

[16] S. Coll, D. Duato, F. Petrini, F. J. Mora, "Scalable hardware-based multicast trees," in *Proc. ACM/IEEE Conf. on Supercomputing (SC03)*, Phoenix, AZ, Nov. 2003.

[17] G. De Fabritiis, "Performance of the Cell processor for biomolecular simulations," *Comput. Phys. Commun.*, vol. 176, no. 11–12, pp. 660–664, June 2007.

[18] R. Fatoohi, S. Saini, and R. Ciotti, "Interconnect performance evaluation of SGI Altix 3700 BX2, Cray X1, Cray Opteron Cluster, and Dell PowerEdge," in *Proc. 20th Int. Parallel and Distributed Processing Symp*., Rhodes Island, Greece, Apr. 2006.

[19] M. Fillo, S. W. Keckler, W. J. Dally, N. P. Carter, A. Chang, *et al*., "The M-Machine multicomputer," in *Proc. 28th Annu. Int. Symp. on Microarchitecture*, Ann Arbor, MI, Nov. 1995.

[20] B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. Giampapa, *et al*., "Blue Matter: approaching the limits of concurrency for classical molecular dynamics," in *Proc. ACM/IEEE Conf. on Supercomputing (SC06)*, Tampa, FL, Nov. 2006.

[21] M. S. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. LeGrand, *et al*., "Accelerating molecular dynamic simulation on graphics processing units," *J. Comp. Chem.*, vol. 30, no. 6, pp. 864–72, Apr. 2009.

[22] R. Garg and Y. Sabharwal, "Software routing and aggregation of messages to optimize the performance of HPCC randomaccess benchmark," in *Proc. ACM/IEEE Conf. on Supercomputing (SC06)*, Tampa, FL, Nov. 2006.

[23] M. J. Harvey, G. Giupponi, and G. De Fabritiis, "ACEMD: accelerating biomolecular dynamics in the microsecond time scale," *J. Chem. Theory Comput*., vol. 5, no. 6, pp. 1632–1639, May 2009.

[24] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, "GROMACS 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation," *J. Chem. Theory Comp.*, vol. 4, no. 3, pp. 435–447, Feb. 2008.

[25] A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang, and S. Pakin, "A performance comparison through benchmarking and modeling of three leading supercomputers: Blue Gene/L, Red Storm, and Purple," in *Proc. ACM/IEEE Conf. on Supercomputing (SC06)*, Tampa, FL, Nov. 2006.

[26] D. Kerbyson, A. Hoisie, S. Pakin, F. Petrini, and H. Wasserman, "Performance evaluation of an EV7 AlphaServer machine," in *Proc. LACSI Symp*., Los Alamos, NM, 2002.

[27] J. S. Kuskin, C. Young, J.P. Grossman, B. Batson, M. M. Deneroff, *et al*., "Incorporating flexibility in Anton, a specialized machine for molecular dynamics simulation," in *Proc. 14th Annu. Int. Symp. on High-Performance Computer Architecture*, Salt Lake City, UT, Feb. 2008.

[28] R. H. Larson, J. K. Salmon, R. O. Dror, M. M. Deneroff, C. Young, *et al*., "High-throughput pairwise point interactions in Anton, a specialized machine for molecular dynamics simulation," in *Proc. 14th Annu. Int. Symp. on High-Performance Computer Architecture*, Salt Lake City, UT, Feb. 2008.

[29] E. Luttmann, D. Ensign, V. Vaidyanathan, M. Houston, N. Rimon, *et al*., "Accelerating molecular dynamic simulation on the cell processor and Playstation 3," *J. Comput. Chem*., vol. 30, no. 2, pp. 268–274, Jan 2008.

[30] "MPI-2: extensions to the message-passing interface," Message Passing Interface Forum, http://www.mpi-forum.org/docs/mpi2-report.pdf, Nov. 2003.

[31] T. Narumi, Y. Ohno, N. Okimoto, T. Koishi, A. Suenaga, *et al*., "A 55 TFLOPS simulation of amyloid-forming peptides from yeast prion Sup35 with the special-purpose computer system MDGRAPE-3," in *Proc. ACM/IEEE Conf. on Supercomputing (SC06)*, Tampa, FL, Nov. 2006.

[32] 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, pp. 224–235, May 1993.

[33] F. Petrini, S. Coll, E. Frachtenberg, and A. Hoisie, "Performance evaluation of the Quadrics interconnection network," *Cluster Computing*, vol. 6, no. 2, pp. 125–142, Oct. 2004.

[34] J. C. Phillips, J. E. Stone, and K. Schulten, "Adapting a message-driven parallel application to GPU-accelerated clusters," in *Proc. ACM/IEEE Conf. on Supercomputing (SC08)*, Austin, TX, Nov. 2008.

[35] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *J. Comp. Phys*., vol. 117, no. 1, pp. 1–19, Mar. 1995.

[36] M. Schlansker, N. Chitlur, E. Oertli, P. M. Stillwell, L. Rankin, *et al*., "High-performance ethernet-based communications for future multi-core processors," in *Proc. ACM/IEEE Conf. on Supercomputing (SC07)*, Reno, NV, Nov. 2007.

[37] S. L. Scott, "Synchronization and communication in the T3E multiprocessor," in *Proc. 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, 1996.

[38] S. L. Scott, Cray Inc., Chippewa Falls, WI, private communication, July 2010.

[39] Y. Shan, J. L. Klepeis, M. P. Eastwood, R. O. Dror, and D. E. Shaw, "Gaussian split Ewald: a fast Ewald mesh method for molecular simulation," *J. Chem. Phys*., vol. 122, no. 5, pp. 054101:1–13, 2005.

[40] D. E. Shaw, R. O. Dror, J. K. Salmon, J.P. Grossman, K. M. Mackenzie, *et al*., "Millisecond-scale molecular dynamics simulations on Anton," in *Proc. Conf. on High Performance Computing, Networking, Storage and Analysis (SC09)*, Portland, OR, Nov. 2009.

[41] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, *et al*., "Anton: a special-purpose machine for molecular dynamics simulation," in *Proc. 34th Annu. Int. Symp. on Computer Architecture*, San Diego, CA, June 2007.

[42] S. J. Sistare and C. J. Jackson, "Ultra-high performance communication with MPI and the Sun Fire™ link interconnect," in *Proc. ACM/IEEE Conf. on Supercomputing (SC02)*, Baltimore, MD, Nov. 2002.

[43] C. Sosa and G. Lakner, "IBM system Blue Gene solution: Blue Gene/P application development," IBM Red-Book, SG24--7287, www.redbooks.ibm.com/redbooks/pdfs/sg247287.pdf, 2008.

[44] H. Subramoni, M. Koop, D. K. Panda, "Designing next generation clusters: evaluation of InfiniBand DDR/QDR on Intel computing platforms," in *Proc. 17th IEEE Symp. on High Performance Interconnects*, New York, NY, Aug. 2009.

[45] O. Tatebe, U. Nagashima, S. Sekiguchi, H. Kitabayashi, and Y. Hayashida, "Design and implementation of FMPL, a fast message-passing library for remote memory operations," in *Proc. ACM/IEEE Conf. on Supercomputing (SC01)*, Denver, CO, Nov. 2001.

[46] M. Woodacre, D. Robb, D. Roe, and K. Feind, "The SGI® Altix™ 3000 global shared-memory architecture," http://moodle.risc.uni-linz.ac.at/file.php/50/altix_shared_memory.pdf, 2005.

[47] C. Young, J. A. Bank, R. O. Dror, J.P. Grossman, J. K. Salmon, and D. E. Shaw, "A 32×32×32, spatially distributed 3D FFT in four microseconds on Anton," *Proc. Conf. on High Performance Computing, Networking, Storage and Analysis (SC09)*, Portland, OR, Nov. 2009.