Link Shutdown Opportunities During Collective Communications in 3-D Torus Nets

S. Conner, S. Akioka, M. J. Irwin, P. Raghavan

The Pennsylvania State University
Computer Science and Engineering Department
University Park, PA 16802, USA
{*sconner*, tobita, mji, raghavan}@cse.psu.edu

Abstract

As modern computing clusters used in scientific computing applications scale to ever-larger sizes and capabilities, their operational energy costs have become prohibitive. While it is an emerging trend in modern cluster design to optimize for low energy consumption in the individual computational nodes, little attention has been paid to reducing the energy used by the communication network that connects the nodes. In this work. we consider a 3-D torus network similar to the one in BlueGene/L to explore opporunities for link shutdown during collective communication operations. For example, we demonstrate that in the case of all-to-one reduce codes, approximately 99% of the total network link time can be spent in a shutoff state on a 64-node toroidal network, thus reducing the overall system energy by approximately 15–28%

1 Introduction

The modern world of scientific computing has increasingly turned to larger and larger supercomputing clusters in order to meet the community's ever-growing need for more capable computers. As codes scale in complexity and the demand for computational resources increases, the compute clusters available to the community have grown, too, and have begun to demonstrate significant challenges in large energy consumption and high levels of power dissipation. To combat this trend, the current generation of compute cluster designers have become much more mindful of the need for controlling energy consumption in clusters. The current state-of-the-art clusters are, consequently, many times more energy efficient than older clusters, but there are still significant areas available for improvement.

Current energy optimizations in clusters are focused on reducing the energy used per bit of computation in the actual processing nodes—the processors, memory, and other associated hardware used to compute results with available data. This trend largely has ignored the approximately 37% of energy budget that is consumed in a cluster by its interconnection network [4]. The interconnection network, in this work a three-dimensional torus, is responsible for efficiently distributing and collecting data across the nodes. It is responsible for providing high bandwidth, low-latency communication, and many possible network energy optimizations come at an unacceptable cost in terms of one of those two performance metrics.

In this work, we consider a technique that has been explored in other application domains and apply this to several common kernels that are used in many scientific computing applications. Link shutdown, the practice of electrically disabling and re-enabling network links through software controls during the run time of an application, is a tested method of saving significant energy in interconnection networks where networks can be shown to have predictable or bursty traffic patterns [7]. These patterns give the system ample opportunity to reactively disable and predictively re-enable links without causing significant performance loss at the application level [7]. We show that scientific codes that employ realistic usage of collective communications, special group communications with well-defined and

1-4244-0910-1/07/$20.00

studied patterns, demonstrate promising opportunities for link shutdown.

While it is often left unstated in literature in the scientific computing world, collective communications are a central and important aspect of almost all scientific computing codes [8]. In practice, virtually all scientific codes employ collective communications for many reasons, two of which are that collective communications are available as simple, pre-tested library calls and that collective communications reduce the needs for explicit barrier synchronization in parallel codes.

We demonstrate that link shutdown opportunities for common collective communication operations on practical systems can surpass 99% while observing network conditions that would allow inexpensive on/off links to be used [7]. This means that, in this context, 99% of all link-cycles during collective communications, the clock cycles where a link would normally be enabled, may be instead spent in an electrically disabled state. Even at the minimum, these algorithms bound at 50% potential shutdown. By allowing the application layer to inform the link layer, the link shutdown process can be made non-predictive. This allows for at least fully half of the collective communication link energy to be saved, representing approximately 15–28% of total system energy.

This paper is organized as follows. In section 2, we introduce the custom simulation platform used to complete the experimental trials. In section 3, we describe our experimental procedure and conditions. In section 4, we present and discuss the experimental results. In section 5, we briefly discuss implementation details for real-world systems. In section 6, we discuss relevant previous work in this field. In section 7, we draw conclusions and mention possible future work. In section 8, we offer acknowledgments.

2 Simulation Platform

Because there was a lack of available preconstructed simulation environments that suited our needs for this work, we developed a simulation tool, TorusSim, to perform our experimental evaluation. In this section, we introduce TorusSim and explain its validation.

2.1 TorusSim Overview

TorusSim is an offline, modular torus and mesh simulator that has been developed by the authors. It uses an event-based simulation engine to route simulated packets through arbitrarily-sized torus and mesh networks in a fashion that is accurate enough to produce meaningful results while being fast enough to simulate large networks and large trace files captured from production hardware.

TorusSim presents the user with a number of important features for this work, including arbitrarily sized one-, two-, and three-dimensional toruses of any size from the trivial case to a large $256 \times 256 \times 256$ network—far larger than any modern supercomputer. TorusSim offers configurable tracking of send, receive, buffer, processing-per-bit, start-message, and receive-message energy, along with customizable tracking of opportunities for link shutdown by recording significant gaps in link traffic at runtime. Buffer sizes, link speeds, and other network parameters are also configurable.

Data input to the simulator is accomplished by recording application-level traces from actual codes on toroidal clusters; converting them into a simple, text-based file format; and providing them as input to TorusSim. Input files are organized according to message traces, with one complete message per input line. Each message description contains a start node, an end node, the message size in 64-bit units, and start and receive time stamps. The unit of time, typically nanoseconds or microseconds, is configurable between milliseconds and nanoseconds. Internally, TorusSim simulates using nanosecond granularity, meaning that all events take an integral number of nanoseconds to complete.

For standard communications, TorusSim implements a statically-routed, minimum-length, uniformly distributed, random-path routing algorithm. All packets within a message follow the same route, and behave according to wormhole-like routing, where all of the packets move in a chain without waiting for each packet to be received before the next is sent. At trace-read time, a static path is generated for each message, and that path is followed by all of the packets in the message. This provides a very simple routing model that is made deadlock-free by the addition of virtual channels in the torus, while still remaining realistic enough to capture network behavior—eg. link activity, congestion, queue utilization, and other statistics. Additionally, routing simulation is simplified further by restricting all links to be point-to-point. This means that each node-to-node connection is statically allocated a dedicated link, or, equivalently, a portion of a shared link where each connection is given a static bandwidth and latency guarantee.

2.2 The TorusSim Event Model

In TorusSim, all events in the network are handled through a central event-management priority queue. Events are prioritized on the basis of event time, event virtual channel/priority, and event serial number, in decreasing order. These criteria provide a deterministic and correct event ordering. To support this, TorusSim requires time-sorted input files and reads a trace from the input file to determine the next input time stamp, then iterates over the existing events until time "catches up" before placing the new event from the file into the queue. This permits the preservation of causality without mandating complicated event-reordering schemes.

It is assumed that all simulated torus links have the same performance characteristics. This technique is accomplished without difficulty in production systems by interleaving the torus nodes and folding the torus into itself, eliminating long cables [1].

2.3 Link Shutdown Analysis

TorusSim implements a simple, effective method of tracking opportunities for link shutdown at runtime. Based on the fact that the network under test is not dynamically routing, it becomes useful to simply track when links are active and when they are not. Every time a packet is transmitted across a link, the times that the transmission starts and completes are recorded. Between packets, the analysis engine looks for gaps by comparing these timestamps, and records significant gaps by adding them to a counter. To determine if a gap is significant, they are compared to a runtime-defined static time—the "link shutdown cutoff"—, which can be varied for each simulation run by the user. Each link in the system has its own counter for link shutdown opportunity.

2.4 Validation

Many approaches were considered for validating TorusSim to verify its correct simulation of networks. Unfortunately, the authors were unable to locate any other project or product that was designed specifically for this purpose and the challenge of re-creating a torus simulator in another framework was prohibitive and would provide only an un-verified benchmark to compare against. Therefore, validation of TorusSim has been accomplished by hand-verification of representative, small simulations.

Several different approaches to validation have been performed on simplified networks. First, to verify that the network is deadlock-free, artificial traces that would otherwise deadlock were given to the simulator, which successfully cleared the network using the escape virtual channel functionality. This functionality breaks down with traces that saturate every buffer in the network at once, but in this situation, no virtual channel system will succeed in clearing the network, so this is expected behavior.

Analysis of the above simulations also provides evidence that the event ordering and congestion models work properly. Events are queued at the right time and execute in the proper order, and when contention occurs in the network, the proper message is blocked according to realistic behavior for real systems.

While no experimental evaluation against existing implemented networks or other, previously-validated, network simulators are available, the hand-validation permits us to assert our confidence that the simulator, as operating, functions properly and accurately enough to demonstrate the broad trends presented in this paper.

3 Experimental Procedure

The effort of simulating real-world data using captured traces from actual machines introduces concerns about the validity and usefulness of captured traces. Additionally, because the simulated network and physical network that traces are drawn from cannot be identical, it is useful to describe both in detail, as we do in this section.

3.1 The Simulated Network

For this work, we configured TorusSim to simulate a BlueGene/L-like 3-D torus network consisting of seven queues per node (X+/-, Y+/-, Z+/-, Exit) [3]. The BlueGene/L is designed as a low-power and high-performance supercomputer, however, most of the power retrenchment is achieved around CPU cores. We conjecture that further energy savings may be possible if collective communications afford ample opportunity for link shutdown. In TorusSim, individual messages between nodes, in this configuration, consist of atomic packets carrying eight 256-bit flits, which with headers results in 1952 data bits per packet. The first packet of a message takes 6000ns to its first hop to account

for application-layer and network-layer setup. All other packet hops, including the first hops of subsequent packets in a message, take 90ns to complete. Links are full duplex, meaning that each torus edge consists of two links, one in each direction such that two nodes, side by side, can communicate at full bit rate without interleaving their communications.

While TorusSim is able to simulate networks ranging from small, one-dimensional rings to very large, three-dimensional superclusters, for this work the range of explored network sizes is constrained. Simulations were performed on a 64-node torus arranged in a logical $4 \times 4 \times 4$ structure embedded in a larger BlueGene/L torus, which provides shortcut links to guarantee proper message timing despite the logical node embedding. Timings in the network were assumed to match the published figures for the full-scale BlueGene/L network at Los Alamos National Labs [3].

3.2 Source and Validity of Traces

Traces for this work were provided by hand-written MPI codes executed on a small BlueGene/L installation at Argonne national Labs. Codes were written specifically to perform minimal calculation between communications, thereby minimizing the opportunity for the application layer to artificially increase the potential for link shutdown.

We experimentally collected traces for all-to-one reduce and all-to-all scatter operations for a wide variety of specifications. By varying message sizes from 8 KiB to 128 KiB, we loaded the network differently to provide a good cross section of behavior. For this work, however, we chose to focus on 32 KiB traces. As the message sizes increase, the trend shows even more profound opportunities for link shutdown. Additionally, we explored sinking to various torus nodes for reduce, but because toruses are always self-similar from the view of any node, the sink node does not matter significantly for any algorithm that involves all-to-one or all-to-all communication. As shown in Figure 1, the variances in opportunity are experimentally small. Therefore, we focus our results on specific sets of nodes.

4 Results and Analysis

In general, results show that applications that depend heavily on collective communication primitives will show great opportunity for exploiting link shutdown (LS) to save energy. Because of the structured nature of primitive collective operations, we see that many links in a three-dimensional torus are never used at all for some codes. For the cases of both all-to-all scatter and all-to-one reduce, approximately 50% of links go *completely* unutilized. This pattern demonstrates that these algorithms operate inherently by embedding a minimal tree structure into the torus, and any link that does not become a tree edge will go unused.

4.1 Scatter Results

Scatter was chosen for this work because it heavily loads the network by distributing data from each node to each other node. At the end of an all-to-all scatter operation, all nodes contain all of the data sent from every node. This has the effect of minimizing the LS opportunities given that it is a well-structured algorithm. Of the collective communication primitives, all-to-all scatter is among the most link-using. The intention of this was to provide a worst-case collective communication demonstration.

For the specific case of 32 KiB messages on a 64-node $4 \times 4 \times 4$ torus, we find that all-to-all scatter opportunities fall off from over 99.999% of total runtime to 50% gradually as the LS cutoff enlarges. From these data, we see that LS opportunities at a cutoff of 0.05s are still 99.9%, at 0.1s are 89.3%, and only reach 50% at 0.8s, never dropping below that percentage. These results are shown in Figure 2.

4.2 Reduce Results

In addition to all-to-all scatter, we investigated all-to-one reduce for this work. All-to-one reduce collects a reduction of data that is spread across the network into a single sink node. Each node reduces the data sent to it by its downstream nodes, such that a node will produce only a single message to its upstream neighbor. This has the effect of loading the network less heavily than almost any other collective communication primitive, because the data is not only being sent to a single node instead of distributed, it is being condensed at each hop. This provides an example of a highly exploitable collective communication operation, as opposed to scatter, which was chosen for its less exploitable behavior.

For the analogous case of reduce as described above for scatter, with 32 KiB messages on the same torus, we see that reduce presents a very different LS profile from scatter. While in the best case we see only about 99.991% opportunity, the curve falls off more smoothly, to 99.1% at 0.05s, down to 92.2%

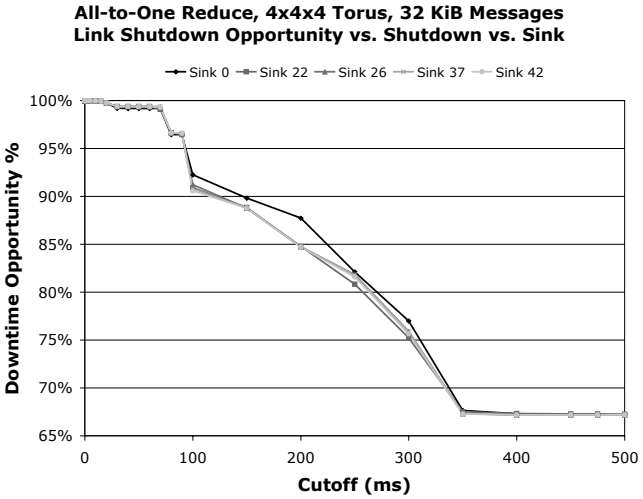

Figure 1: Insignificant LS Opportunity Variance Due to Sink Location (Zoomed in to significant region)

at 0.1s, and bottoming out only at 67.2% at 0.45s. These results are shown graphically in Figure 3, and meant for comparison with those in Figure 2.

In noting that the all-to-one reduce operation reaches a minimum at a higher opportunity than all-to-all scatter, we can see a general trend that all-to-one operations tend to make much less use of the network, and therefore provide more significant opportunities for link shutdown.

5 Link Shutdown Implementations

For the majority of this work, we have avoided the discussion of implementing a real-world system to implement the ideas presented here. In the literature, many works consider the various ways of implementing link shutdown, from which two major techniques for implementation have emerged. We discuss those methods and how they apply to this specific problem domain here.

The technique that is most simple and also presently available commercially is the use of "on/off" links. As the name implies, these are links that switch, via some technique, between entirely enabled and entirely disabled, doing so as needed during program runtime. These typically use a counter to count consecutive cycles of links going unused and disable a link once it has reached a threshold, only to non-predictively re-enable the link when it is needed again. This is inexpensive and not complex in terms of hardware implementation, but a naive approach can cause latency loss in the single digit percents [7]. Systems with highly redundant link structures such as fat-trees [2] and the torus networks considered here, however, fare much better with basic on/off links. Commercially, on/off links that exhibit maximal latency increases of tens of microseconds were available in 2004 [7], and we have demonstrated here that codes showing thousands of microseconds of inactivity show very high shutdown opportunities.

By moving to a more complex hardware implementation, designers may opt to use dynamic voltage scaling (DVS) and dynamic link shutdown (DLS) in combination. This method is comprised of two techniques. Firstly, DVS is used to electrically slow down links that are necessary but experiencing minimal load, thereby saving energy and making messages traveling on those links experience higher latency while still allowing connectivity. Then, network-level heuristics predictively disable and re-enable links that are historically not highly used, attempting to predict when they will be necessary. This has the potential to reduce the latency cost to sub-microsecond times [7] and to save more energy than on/off links. DVS and DLS links are, however, more complex and expensive to implement [5].

We propose that a method for exploiting the link shutdown opportunities in toroidal networks is to use non-predictive on/off links. In real-world codes, collective communication primitives may be han-

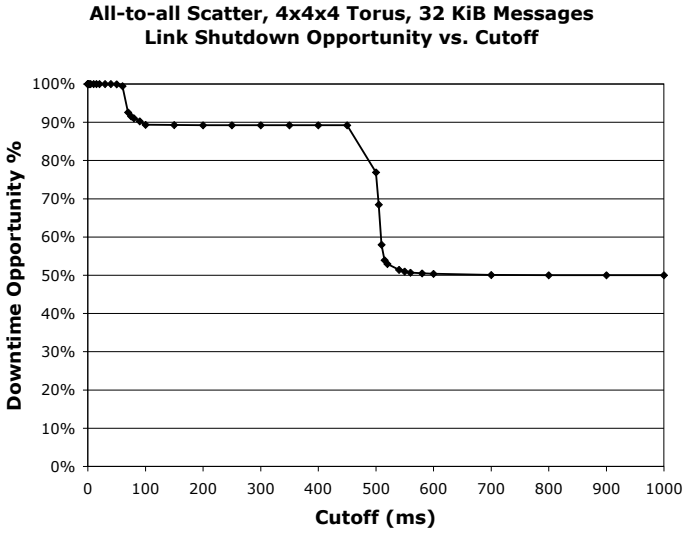

Figure 2: LS Opportunity vs. LS Cutoff for 64-Node Scatter

dled by optimized software libraries that may inform the network structure of impending periods of link activity or inactivity. Because such large potentials for shutdown exist, very accurate on/off timing may be achieved. By combining this with traditional on/off links, we predict that significantly near-zero latency costs may be achieved without significantly increasing hardware cost beyond what is already commercially available.

6 Prior Work

There is a limited body of work that discusses large-scale network simulation with an eye toward accurate energy analysis and link shutdown opportunity.

Alonso, et al., analyzed the effects of network link shutdown using on/off links for general applications running on fat trees, a similar but more specialized network topology than those considered here. In that work, they found that significant energy can be saved with very little performance cost by exploiting redundancy in the fat tree interconnection when the network was not heavily loaded [2].

A number of groups have investigated on-chip mesh energy for network-on-chip (NoC) computers. Orion from Princeton University [9] is a tool that performs cycle-accurate energy and performance modeling of NoCs. Due to differences in the way that meshes and toruses operate and the architectural differences between simulating small NoCs and large cluster systems, this work is not directly applicable to modeling computing clusters. It has, however, significantly informed the field of mesh network energy analysis.

In [5], Kim, et al., investigated practical energy-saving techniques on individual cluster links. By performing an energy analysis of a practical dynamic voltage scaling (DVS) and link shutdown (LS) system for cluster-style interconnections, the authors of that work concluded that DVS and LS are useful and effective energy-saving techniques. In that work, the authors analyzed a synthesizable network link chip and associated link energy from a real-world system and exposed the trend that, as technology continues to scale, the need for network-link-energy saving techniques like DVS and LS will only increase.

7 Conclusions

Energy consumption of the interconnection networks in large-scale compute clusters has become a significant, under-addressed problem in modern cluster design. Because energy usage in a network is not strongly dependent on the operational speed of the network, reducing the energy consumed by the network per computation must rely on utilizing the network less. Without adapting scientific codes or routing algorithms to use networks in more efficient

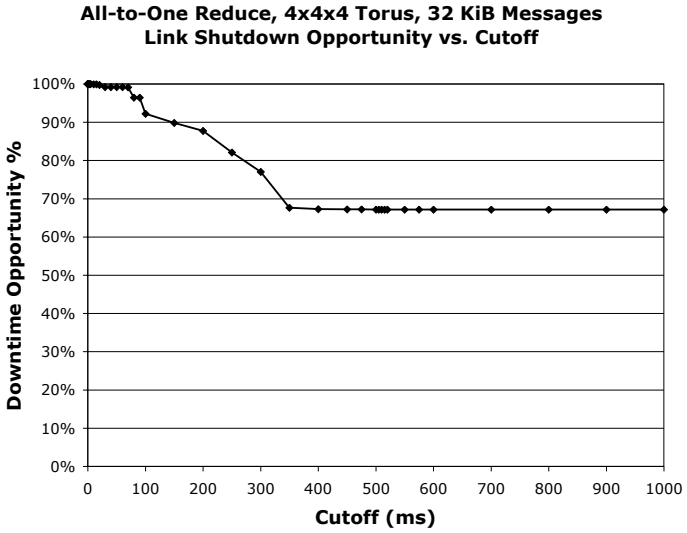

Figure 3: LS Opportunity vs. LS Cutoff for 64-Node Reduce

ways, we still see that using current, commonly used collective communication primitives provides significant opportunities to save energy with link shutdown. In this work, we demonstrated that with a link shutdown timer of 0.5s, a time easily managed through explicit software calls, we can save 50–67% of link energy. By decreasing the time to 0.1s, we increase the potential energy savings considerably to 92+%. This could represent considerable savings over link shutdown for applications such as parallel dense linear algebra libraries which rely on collective communications with large message sizes [6].

Unfortunately, simply finding the opportunities for link shutdown will not, in itself, save energy. Instead, a practical system must implement link shutdown and wake-up in a way that preserves system efficiency while effectively utilizing the opportunities presented. Thankfully, on/off links may be used in conjunction with application level support to simply enable and disable links non-predictively, when it is known that the application will not need them. This has the potential to achieve very high efficiencies with minimal latency cost, but requires applications to be conscious of their link usage [7].

In the future, we will extend this work and analysis to encompass detailed consideration of multiple types of networks running heterogeneous application codes. By focusing on how real-world benchmarks, when properly coded, are crafted to include collective communication primitives, we will analyze how network energy in overall systems will be affected when using link shutdown methods that focus on the primitives. Finally, many potential avenues for work including TorusSim exist, including analysis of on-chip networks; heterogeneous, multi-chip NoC systems; and very large scale supercomputing networks beyond those currently implemented in real systems.

8 Acknowledgements

The authors would like to thank Dr. Greg Link of the York College of Pennsylvania for his early input into this work and the staff of Argonne National Laboratories for the use of their BlueGene/L mini-cluster to run our experimental codes on. This work has been supported in part by NSF Grant 0444345 and by MARCO/DARPA GSRC.

References

[1] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas. Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 49(2/3):265–276, March/May 2005.

[2] M. Alonso, S. Coll, J. M. ad V. Santonja, P. Lopez, and J. Duato. Dynamic power saving in fat-tree interconnection networks using on/off links. In *Proceedings of the 2006 Parallel and Distributed Processing Symposium, IPDPS 06*, April 2006.

[3] A. Gara, M. Blumrich, D. Chen, G.-T. Chiu, P. Coetus, M. Giampapa, R. Haring, P. Heidelberger, D. Hoenicke, G. Kopcsay, T. Liebsch, M. Ohmacht, B. Steinmacher-Burow, T. Takken, and P. Vranas. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49(2/3):195–212, March/May 2005.

[4] E. Kim, K. Yum, G. Link, N. Vijaykrishnan, M. Kandemir, M. Irwin, M. Yousif, and C. Das. Energy optimization techniques in cluster interconnects. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pages 459–464, August 2003.

[5] E. J. Kim, G. M. Link, K. H.Yum, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and C. R. Das. A holistic approach to designing energy-efficient cluster interconnects. *IEEE Transactions on Computers*, 53(6):660–671, June 2005.

[6] P. Mitra, D. Payne, L. Shuler, R. van de Geijn, and J. Watts. Fast collective communication libraries, please. In *Proceedings of the Intel Supercomputing Users' Group Meeting*, 1995.

[7] V. Soteriou and L.-S. Peh. Design-space exploration of power-aware on/off interconnection networks. In *ICCD '04: Proceedings of the IEEE International Conference on Computer Design (ICCD'04)*, pages 510–517, Washington, DC, USA, 2004. IEEE Computer Society.

[8] J. S. Vetter and F. Mueller. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 96, Washington, DC, USA, 2002. IEEE Computer Society.

[9] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. In *Proceedings of MICRO 35*, November 2002.