Power Management of Extreme-scale Networks with On/Off Links in Runtime Systems

Ehsan Totoni, University of Illinois at Urbana-Champaign
Nikhil Jain, University of Illinois at Urbana-Champaign
Laxmikant V. Kale, University of Illinois at Urbana-Champaign

Networks are among major power consumers in large-scale parallel systems. During execution of common parallel applications, a sizeable fraction of the links in the high-radix interconnects are either never used or are underutilized. We propose a runtime system based adaptive approach to turn off unused links, which has various advantages over the previously proposed hardware and compiler based approaches. We discuss why the runtime system is the best system component to accomplish this task, and test the effectiveness of our approach using real applications (including NAMD, MILC), and application benchmarks (including NAS Parallel Benchmarks, Stencil). These codes are simulated on representative topologies such as 6-D Torus and multilevel directly-connected network (similar to IBM PERCS in Power 775 and Dragonfly in Cray Aries). For common applications with near-neighbor communication pattern, our approach can save up to 20% of total machine's power and energy, without any performance penalty.

Categories and Subject Descriptors: D.3.4 [**PROGRAMMING LANGUAGES**]: Processors—*Run-time environments*; C.1.4 [**PROCESSOR ARCHITECTURES**]: Parallel Architectures; B.4.3 [**INPUT/OUTPUT AND DATA COMMUNICATIONS**]: Interconnections (Subsystems)

1. INTRODUCTION

Large-scale parallel computers are becoming much bigger in terms of the number of processors, and larger interconnection networks are being designed and deployed for those machines. The reason is that the demand for performance of supercomputers is escalating, while single-thread performance improvement has been very limited in the past several years. Moreover, the many-core era with on-chip networks is rapidly approaching, which will add another level to the interconnection network of the system [Totoni et al. 2012]. These immense networks are a key factor in the performance and power consumption of the system.

Modern networks are over-provisioned in resources (e.g. links), in order to provide good performance for a range of applications. Since networks with lower latency and higher bandwidth, in comparison to existing popular networks (such as 3D Torus networks), are necessary for some applications executing on multi-petaflop/s systems, higher radix network topologies such as multi-level directly connected ones [Arimilli et al. 2010; Bhatele et al. 2011b; Faanes et al. 2012] and high-dimensional tori [Ajima et al. 2011] are being proposed and used. Although these networks are designed to provide enough bisection bandwidth for the worst case (e.g. all-to-all communication

The authors are members of the Parallel Programming Laboratory in the Department of Computer Science at the University of Illinois at Urbana-Champaign, Urbana, IL, 61801. For contact information, please visit http://charm.cs.illinois.edu/
Preliminary work for this paper was presented at Workshop on High-Performance, Power-Aware Computing (HPPAC 2013) [Totoni et al. 2013].

DOI:http://dx.doi.org/10.1145/0000000.0000000

in FFT), not all applications make use of the abundant bandwidth. Furthermore, the intention is to provide low latency for all applications, hence the network provides small hop count and low diameter for any given pair of nodes. However, the set of communicating node pairs of different applications vary, which may leave some part of the network unused in each application. As evidenced in this paper (Section 3), the net result is that many applications do not use a large fraction of links, especially for high radix networks.

Saving network power is crucial for keeping HPC systems within a reasonable power budget. Power and energy consumption are major constraints for HPC systems and facilities [Kogge et al. 2008], especially at the high end. Interconnection networks are often among the major power consumers for different systems, and many researchers have reported on their power consumption. For example, routers and links are expected to consume about 40% of some server blades' power, which is the same as their processors' power budget [Shang et al. 2003; Soteriou and Peh 2007]. For current HPC systems, using an accurate measurement framework, Laros et.al. [Laros et al. 2012] report more than 25% total energy savings by shutting off some of the network resources of a Cray XT system. In future systems, especially because of the increasing number of cores per chip, and aggressive network designs, the network is expected to consume 30% of the system's total power [Kogge 2008]. From this power consumption, up to 65% is allocated to the links and the resources associated with them [Soteriou and Peh 2007] (and the remaining 35% is mostly consumed by routers). In addition, up to 40% of the many-core processor's power budget is expected to go to its on-chip network [Soteriou and Peh 2007].

In contrast to processors, the network's power consumption does not currently depend on its utilization [Soteriou and Peh 2007], and it is near the peak whenever the system is "on". For this reason, while about 15% of the power and energy is allocated to the network in many current systems [Mahadevan et al. 2009], it can go as high as 50% [Abts et al. 2010] when the processors are not highly utilized in data centers. While, for *HPC* data centers the processors are not usually as underutilized, they are not fully utilized all the time either and energy proportionality of the network is still a problem. Therefore, it is essential to save the network's power and make it energy proportional [Abts et al. 2010], i.e. the power and energy consumed should be proportionate to the usage of the network.

An effective approach to address this problem and improve energy proportionality is to turn off unused links. Thus, we propose addition of hardware support for on/off control of links (links that can be turned on and off), which can be used by the runtime system to save the wasted power and energy consumption. We show how the runtime can accomplish that by observing the applications' behavior. Note that adaptive runtimes have also been shown to be effective for load balancing and power management (using DVFS) [Sarood et al. 2012]; our approach makes use of the same infrastructure. We also discuss why the hardware and compiler cannot perform this task effectively, and why network power management should be done by the runtime system.

Our contributions can be summarized as follows:

— We have evaluated the communication patterns of different HPC applications' and benchmarks' with respect to extreme-scale high-radix networks. The applications and benchmarks we evaluated include NAMD [Kale et al. 2011], MILC [Bernard et al. 2000], ISAM [Jain and Yang 2005], Stencil benchmarks (representing nearest neighbor communication patterns) and some of NAS Parallel Benchmarks [Bailey et al. 1992].

— We have proposed a runtime system based approach to adaptively turn off unused links, which has various advantages over the previously proposed hardware and compiler based approaches.
— We have developed a theoretical model of link utilization of HPC applications, which provides insights about the applications and networks.
— We present a case study demonstrating that system design alternatives (e.g. mappings) with similar performance can have very different power consumption profiles.

Using our basic approach, for commonly used nearest neighbor applications such as MILC [Bernard et al. 2000], 81.5% of the links can be turned off for a multilevel directly-connected network (around 16% of total machine power, assuming 30% network power budget), and 20% for 6D Torus (Sections 3 and 4). Moreover, we demonstrate that approximately 20% of the machine power can potentially be saved for most applications on these networks (Section 6) using a smarter scheduling approach. All these can be realized if the system allows the runtime system to turn off some of the links.

Sections of this paper are organized as follows. Section 2 establish the background by discussing the related work, extreme-scale networks and applications' communication patterns. Section 3 demonstrates, via empirical evidence, that many links are never used on different high-radix networks and proposes a basic approach to turn them off in the runtime system. Subsection 3.2 of this section presents a case study, which shows how much our basic approach can save for two different design alternatives (which have the same performance). Section 4 discusses the implementation of our approach in a runtime system, and methods to handle practical issues that arise. Section 5 develops a theoretical model to estimate the power and energy that can be saved for an application, running on a high-radix network. The insight from this model helps us generalize the idea into a more practical scheduling approach in Section 6, which also considers the on/off transition delay. We conclude the paper in Section 7.

2. BACKGROUND AND MOTIVATION

2.1. Related Work

Power consumption of interconnection networks in supercomputers, distributed systems and data centers has received special attention in recent times. Several techniques have been proposed for reduction of network power in non-HPC data centers [Abts et al. 2010; Mahadevan et al. 2009; Heller et al. 2010]. Intelligent power-aware job allocation and traffic management schemes form the basis of many of these approaches. Laros et.al. [Laros et al. 2012] present results on potential power saving using CPU and network scaling, by post processing the data collected from the monitoring system of Cray XT machines. Their work, using real systems (instead of simulations and projections) and real applications, shows the importance and potential of network power management for supercomputers.

Among hardware based approaches, power management of interconnection networks using on/off links has been studied [Soteriou and Peh 2007; Alonso et al. 2006; Li et al. 2011]. On/off links, which refers to shutting down communication links that are not being used, has shown to be a useful method to save power. However, dependence on hardware for power management may cause considerable delay for some applications. Additionally, hardware does not have enough global information about the application to manage network power effectively.

Soteriou et.al. [Soteriou et al. 2007] show severe possible performance penalty of hardware approaches, and propose the use of parallelizing compilers for power management of the links. However, parallelizing compilers are not widely used because of their limited effectiveness, and most parallel applications are created using explicit

parallel programming models [Becker 2012]. Furthermore, compilers do not have information about input dependent message flow of an application, and cannot manage the power effectively for such applications.

New programming paradigms to perform power management inside the application (by giving more information about the communication) has also been proposed [Hendry 2013]. Although the programmer has more information about the application, involvement of the programmer compromises productivity. In addition, such approaches cannot be applied to legacy code easily. Thus, automatic approaches seem more practical.

As an alternative to hardware, compiler and application driven power management, we advocate network power management by the runtime system. Limited network power management by the runtime system, such as for collective algorithms, has been proposed in the past. Power management using on/off links in the runtime system has also been studied [Conner et al. 2007]. However, that approach is limited to management of network links only during collective operations in MPI. In this paper, we propose the use of an adaptive runtime system to manage the power of network links using on/off control, taking into account all of the communications performed by an application.

2.2. Network Power Management Support on Current Machines

Unfortunately, network power management support on current HPC machines is very limited. For example, it is possible to reduce link and node injection bandwidth on a Cray XT system (effectively turning off some portion of the links), but it requires a reboot of the whole machine [Laros et al. 2012]. Thus, using this feature is impractical. Other recent machines such as IBM Blue Gene/Q, Cray XE6, and Cray XC30 do not seem to have any feature for dynamic network power management either. However, techniques such as on/off links have been implemented before, and it seems feasible to include them for HPC machines as well. For instance, some commercial systems [1] can disable some of the board-to-board and box-to-box links to save power. Currently it takes 10,000 cycles to turn the links on/off, although even this can be improved much further [Soteriou and Peh 2007].

2.3. Extreme-scale Networks

In this section, we briefly describe n-dimensional tori and multilevel directly-connected networks, which have been used in recently developed supercomputers that are predominant in the Top-500 list [top500 2013].

n-dimensional tori have been used in many supercomputers such as IBM Blue Gene series, Cray XT/XE, and the K computer. Given an n-dimensional mesh, a torus is obtained by adding wrap around links in every dimension, i.e., by adding links that connect nodes at one end of a dimension to the nodes at the other end. Tori are symmetric in the sense that the number of links out of every node is the same, with each node being connected to two other nodes in every dimension. An n-dimensional torus strikes a good balance in terms of bisection bandwidth, latency, and the link cost, and have been shown to be scalable. In the past few years, most vendors have increased the torus dimensionality from three (as it is in IBM BlueGene/P and Cray XT/XE) to five (IBM BG/Q) and six (the K computer). This shift is necessary in order to keep latency low, with possible increase in the bisection bandwidth. We present analysis and results for link utilization of an n-dimensional torus, with n varying from 3 to 10.

Multilevel directly-connected networks have been proposed by IBM (PERCS network [Arimilli et al. 2010]), the DARPA sponsored Exascale study report [Kogge et al. 2008] (Dragonfly topology [Kim et al. 2008]), and Cray (Aries network [Faanes

[1] Motorola MC92610 WarpLink 2.5 Gb/s Quad SERDES Transceiver, Motorola Inc., www.motorola.com

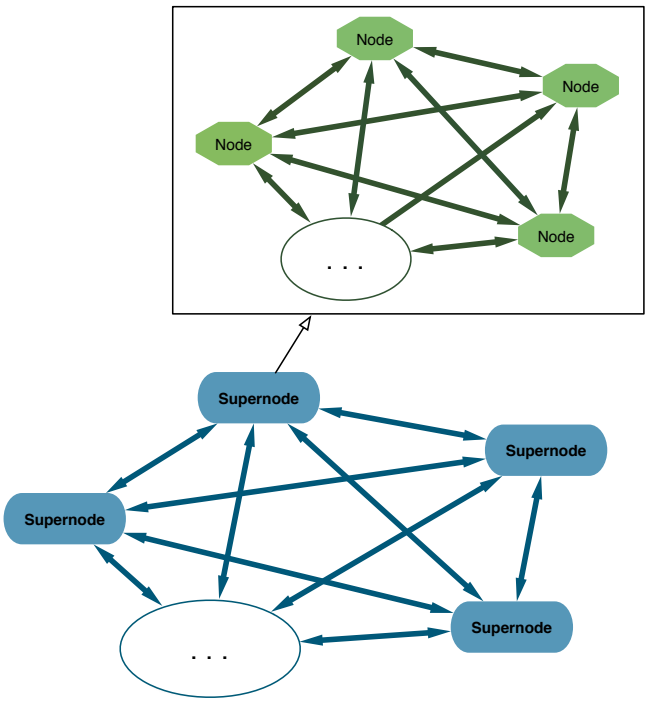


Fig. 1. IBM PERCS - a two-level directly-connected network

et al. 2012]). In all of these proposals, similar multi-level directly connected networks have been described. In these networks, nodes are logically grouped together at multiple (currently two or three) levels. In each level, nodes (or the grouped entities from previous level) are connected in an all-to-all manner. Hence, in the first level a clique of nodes is formed, and in the second level, a clique of cliques (from the first level) is constructed and so on. The resultant network, with its large number of links, boasts of a large bisection width. At the same time, the latency of the entire system is low (few-hop connectivity between any pair of nodes). Currently, these networks are used in some large-scale IBM Power 775 machines [2] and in the Cray XC30 machines. In this study, we use the parameters of PERCS as an instance of multilevel directly-connected networks, since they are readily available. However, the conclusions will apply to other multilevel directly-connected networks as well.

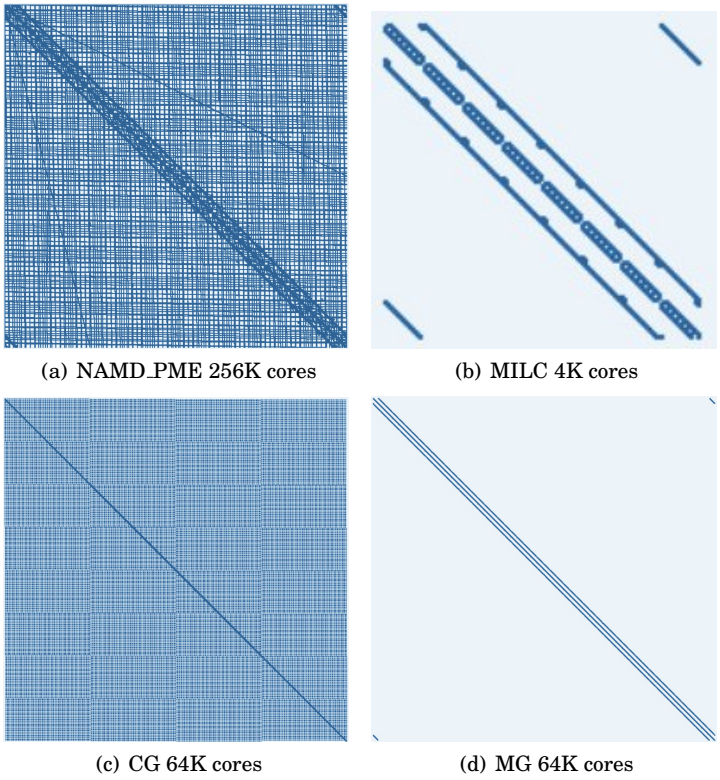
In Figure 1, we present a prototype of the PERCS network (two-level directly connected), in which the nodes are grouped, and connected in an all-to-all manner to form supernodes. These supernodes are further connected in an all-to-all fashion to obtain the entire system. We present link utilization results for these networks as well.

We observe that the two topologies we present results on, multilevel directly-connected networks and tori with high dimensionality, have a large number of links. The presence of these links provides an opportunity for high performance, as well as a challenge for power and energy proportionality.

[2] www.top500.org

2.4. Application Communication Patterns

Interconnection networks are designed to serve a range of communication patterns, in terms of bandwidth and latency. High radix networks, such as multilevel directly connected ones, provide a large number of links to support demanding communication patterns such as all-to-all and varying demands of different applications. In addition, in order to maintain low latency and fewer hops between every node pair, a large number of links are required. However, each application has its own communication pattern, so many node pairs of a system may not communicate during execution of a common application, leaving a large fraction of the network unused.



(a) NAMD_PME 256K cores (b) MILC 4K cores

(c) CG 64K cores (d) MG 64K cores

Fig. 2. Communication pattern of different applications

Figure 2 shows the communication pattern of some of the applications we use in this paper. NAMD [Kale et al. 2011], implemented in CHARM++ [Kale and Zheng 2009], is a prevalent parallel molecular dynamics code designed for high-performance simulation of large bio-molecular systems. Its localized communication pattern represents the pattern of many common particle interaction HPC applications. MIMD Lattice Computation (MILC) [Bernard et al. 2000] is widely used to study quantum chromodynamics (QCD). Similar to many HPC applications, it has a near-neighbor communication pattern. We use CG from NAS Parallel Benchmarks (NPB) [Bailey et al. 1992] to represent expensive many-to-many communication pattern found in some applications and MG from NPB to represent the communication pattern of common numerical solvers.

Both the vertical and horizontal axes of Figure 2 represent the nodes in a system. A point (x, y) is marked if the node y on the vertical axis sends a message to the node

x on the horizontal axis, during the execution of an application. Each marked point has been enlarged for better illustration. It can be seen that many of the node pairs never communicate during execution of various applications. Moreover, the number of pairs that communicate varies significantly with the application. For instance, in NAMD_PME and CG, the number of node pairs that communicate is much larger than in MILC and MG.

Most of the communicating pairs in NAMD_PME are due to the FFT performed in the PME phase, which is done once every four iterations. Without the PME option, NAMD has a near neighbor communication pattern, which can be seen in the dense region around the diagonal of Figure 2(a). CG, on the other hand, has a more uniform and dense communication pattern. Applications like NAMD_PME and CG, that have large number of communicating pairs are more likely to use most of the network.

On the other hand, the number of communicating pairs in MILC (Figure 2(b)) and MG (Figure 2(d)) are few, and concentrated near the diagonal. As such, these applications are expected to make use of a small fraction of the available network links. These applications represent a large class of applications in science and engineering, such as the ones following the nearest neighbor pattern [Bailey et al. 1992].

All illustrated cases have a dense region close to the diagonal of their communication graph, suggesting that nearest neighbor communication constitutes a major part of many applications' communication. This can be used as a clue in understanding a network link's usage. We use Stencil, decomposed in two, three and four dimensions, to study network's link usage for near neighbor communication patterns. From this discussion, there are reasons to expect that there is an extensive opportunity to save the power of the network links in higher-radix topologies in many common cases, since they are designed for the worst cases with many communicating pairs (such as random access benchmark or FFTs).

In summary, there are applications that have intense communication patterns such as all-to-all, but many applications have only nearest neighbor pattern. Additionally, *embarrassingly* parallel applications that essentially do not rely on the network during their computation (e.g. ISAM [Jain and Yang 2005]) represent an extreme set. Since they do not use any of the links, the link power can be saved easily.

3. POTENTIALS OF BASIC NETWORK POWER MANAGEMENT

In the previous section, we observed that the number of communicating pairs for many applications is not large, which indicates that a sizeable fraction of links may be unused. In this section, we discuss and evaluate a basic power-saving approach for links, implemented in an adaptive runtime system. In this approach, the runtime monitors a few iterations and turns off the links that are never used during execution.

Our methodology of evaluation is to emulate an application at scale using BigSim (which has been validated for these networks before [Totoni et al. 2011; Zheng et al. 2004]) and capture the communication traces. These traces are then used to simulate the target network and mark the links that are used.

We assume default mapping for placing processes (ranks) onto processors for all the networks. For a 32 cores per node case, it means that the first 32 ranks are mapped to the first node, next 32 ranks are mapped to the second node and so on. Only direct routing is considered for multilevel directly-connected networks in this section (before Subsection 3.2), which means that the messages are sent directly to the receiver, instead of going through an intermediate supernode (i.e. indirect routing). Effect of different mappings and indirect routing are discussed and evaluated in Subsection 3.2. For tori, we assume minimal dimension order routing, which is used in many of the current supercomputers. The results are easily extensible for adaptive routing as we discuss later.

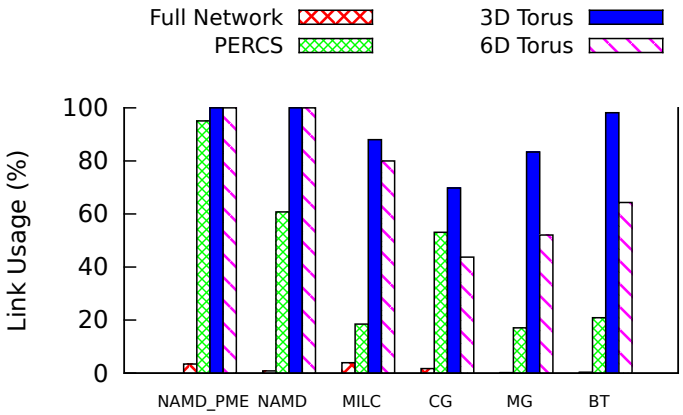


Fig. 3. Fraction of links used during execution of various applications.

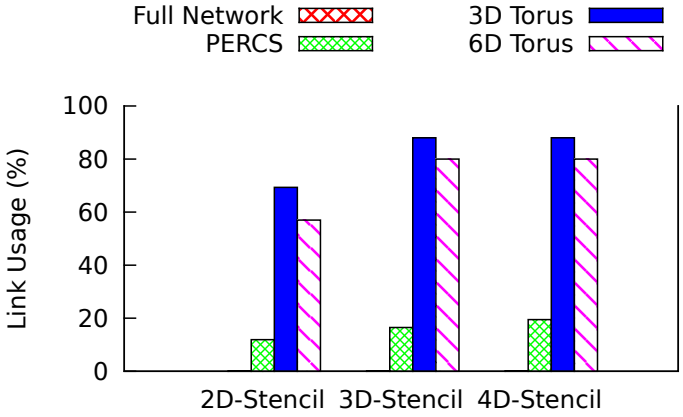


Fig. 4. Fraction of links used during execution of stencil codes.

3.1. Link Usage of Modern HPC Networks

Figure 3 and Figure 4 show link usage of different applications and benchmarks for a fully connected network (a link between every pair of nodes), 3D Torus, 6D Torus and PERCS (two-level fully-connected). In the context of this section, we consider a link as "used" if it is used *at any time during* execution of an application. With this assumption, the specifics (e.g. link bandwidth) of each network other than its topology do not make any difference. These used links may be utilized for only a small fraction of the application execution; we will exploit this property in Sections 5 and 6. Note that the full network is an asymptotic case that is not usually reached by the large-scale networks. However, for example, small jobs (less than 1k cores for PERCS) running on a two-tier system will have a fully connected network. In this case, most of the links can be shut down according to our results, which saves a significant fraction of the system's power.

As shown in Figure 3, link usage of each application is different, and depends on the topology of the system. For example, MILC only uses 3.93% of the links of a fully connected network, while it uses 80% of the 6D Torus links. For most applications, a larger fraction of 6D Torus links will be used in comparison to links on PERCS

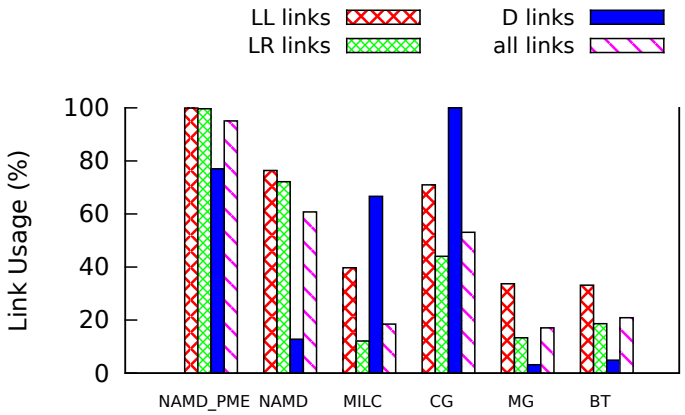


Fig. 5. Fraction of different links used during execution on PERCS

network; an exception is CG that uses a higher fraction of PERCS links. This shows that analysis of the link utilization of different networks is not trivial and depends on various aspects of the topology and the application.

For the applications of Figure 3, from 4.4% to 82.94% of the links are never used during the program's execution on PERCS. In the stencil benchmarks of Figure 4, 2D-Stencil uses only 11.91% of the PERCS links, with similar numbers for other stencil dimensions. This demonstrates a great opportunity for the runtime system to save link power of two-level directly connected networks.

There is an opportunity on 6D Torus to save energy as well. Even though NAMD uses all of the links, MG leaves 47.92% of the links unused. However, many applications can use most of the links of a 3D torus, which has been one of the dominant topologies in the past and in the current supercomputers. There is potential for saving power in some cases (e.g. 30.67% for 2D-Stencil), but the savings are neither high nor common. This happens even with deterministic routing, which uses fewer links than adaptive routing. This shows that implementing on/off links for those networks is not significantly useful, and probably is the reason that they have not been implemented so far. However, for high dimensional tori and multi-level directly connected networks, the benefits justify the implementation cost of software controlled on/off links. If we take MILC to represent a significant set of common HPC applications (which usually have near neighbor communication), 81.51% of PERCS links and 20% of 6D Torus links can be turned off to save power. Assuming that 65% of network power goes to links and the network consumes 30% of the total machine's power, around 16% of total machines power can be saved for PERCS systems and around 4% can be saved for 6D Torus systems.

In NAMD_PME, the communication intensive PME calculation is usually performed every four iterations (which takes around 16 ms assuming about 4 ms per iteration for ApoA1 benchmark on 2K cores of BGQ [Sameer Kumar and Kale 2013]). In this case, many links can be turned off after PME communication is complete, and turned back on right before the next PME communication phase begins (scheduling on/offs is further discussed in Sections 5 and 6).

We observe that even though 3D-Stencil has a 3D communication pattern, when it is mapped to a system with 32 cores per node, the communication between nodes is not an exact 3D pattern anymore. Thus, some fraction of the links (12%) are not used.

So far, for simplicity, we assumed that all the links of the network are the same and have the same power cost. However, networks are usually made of different links for

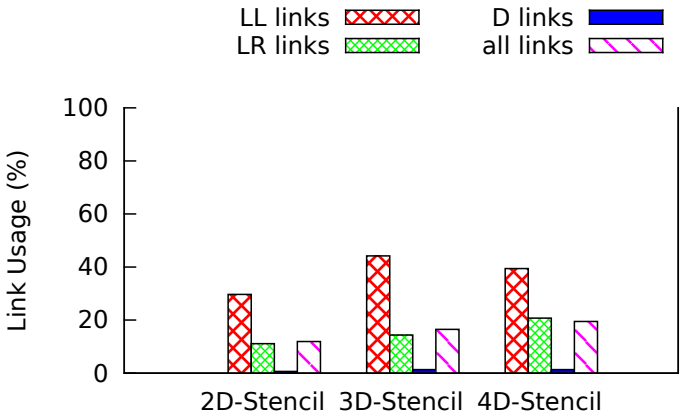


Fig. 6. Fraction of different links used during execution on PERCS

practical purposes. These links even have different technologies (optical vs. electrical). For example, in PERCS network, global links are called D-links and connect clusters of nodes at second level and they use optical technology [Arimilli et al. 2010]. These long links are probably more power hungry than the local ones. On the other hand, LLocal (LL) and LRemote (LR) links connect nodes placed close to each other (LL for nodes in the same drawer, LR for other local links) and use electrical technology. These local links probably consume much less power compared to the global ones.

To find out which type of links are used more often, Figure 5 and Figure 6 show the usage of different types of PERCS links. Overall, D-links are usually less utilized than the local ones. This happens because most of the communication of the applications is either local or near neighbor exchange. Even NAMD_PME, which exhibits limited opportunity for power saving in previous results, does not use 23.09% of D-links; this may improve absolute power saving. MILC shows high usage of D-links because the results are for 4K processors only (4 supernodes), hence there are just 12 D-links. For larger configurations, it should show link usage similar to 4D-Stencil and have a very low D-link utilization. CG is again an exception and uses more of the D-links. This is because its communication is not local but distributed as mentioned earlier. The stencil benchmarks use only around 1% of D-links and most of those links can thus be turned off safely. Thus, using a simple model of same power cost for all links is pessimistic, and the actual savings can be much higher in many cases as the power hungry D-links have less utilization. Note again that these results are with the default (rank-ordered) mapping (we will discuss a case of other mappings in Section 3.2).

Figure 7 and Figure 8 show the link usage of the applications on tori with different dimensions, from 3 to 10. As dimensionality of tori is increased, a smaller fraction of the links are used, which is intuitively expected. For example, 4D-Stencil uses only 53% of the links of a 10D torus network, but more than 80% links are used on a 3D torus. Even NAMD that does not have any savings on low dimensional torus, shows potential for saving power on a torus with sufficiently high dimensionality, starting from 7D. It uses only 65% of the links of a 10D torus, which shows that even such applications have potential of link power savings on high dimensional tori.

Other than these applications, there are cases where the network is virtually unused. Data parallel applications do not have much communication (except during startup, I/O in the beginning and at the end) and do not use the network during execution. For example, ISAM [Jain and Yang 2005], which is a climate modeling application, only uses stored climate data to do the computation (in its standalone mode).

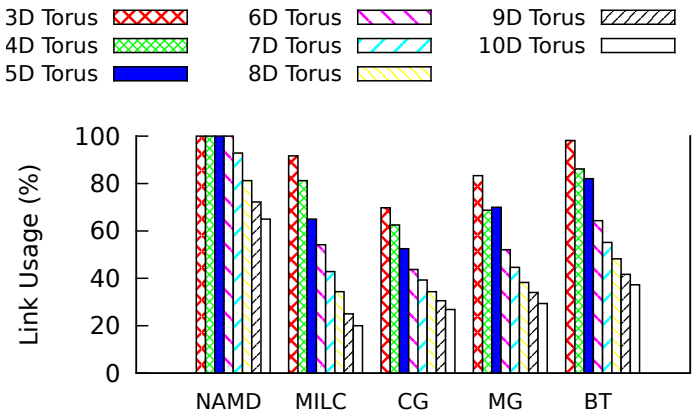


Fig. 7. Fraction of links used during execution

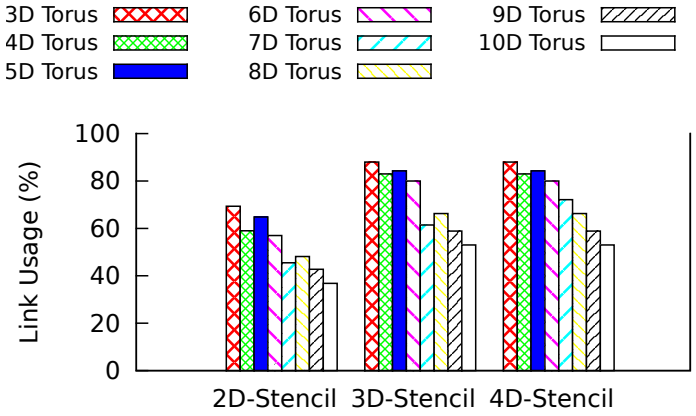


Fig. 8. Fraction of links used during execution

Thus, almost all of the network power can be saved for these applications during the main computation phases (I/O uses the main network in some machines, but it happens usually infrequently).

To summarize, in the common near neighbor applications like MILC, up to 16% of total machines power can be saved (assuming 30% network power budget and 65% of network power associated with links) using a basic power management approach. Since our assumptions are very conservative and only links that are never used (and are not likely to be used) are turned off, the application will not experience a significant performance penalty.

3.2. Different Mappings

In the results so far, we assumed default mapping with direct routing for PERCS network, which implies sending each message directly to the destination supernode. However, it had been shown that this configuration might result in contention in few links of the network for some applications [Bhatele et al. 2011b]. In this case, one might use intelligent application specific mappings, or use other more general alternatives that had been proposed before. A previous study on PERCS network [Bhatele et al. 2011b] suggests using random mapping or indirect routing to avoid contention on few links

and improve performance. Indirect routing uses a random intermediary supernode for each message transfer (dynamically), while random mapping places the processes (e.g. MPI ranks) on random processors (statically). The purpose is to use more links to avoid contention, at the cost of some possible overheads (e.g. due to more hop count).

Figure 9 shows the link usage of these schemes (proposed in [Bhatele et al. 2011b]) compared to the default mapping. Random mapping has higher link usage than default mapping, which is intuitive. It can use 33.18% of the links, which is twice the 16.51% link utilization of the default mapping. However, the overall usage is still low and the possible savings are as high as 67%. Note that this scheme uses many more D-links, which may increase the power consumption significantly. Thus, when choosing among different mappings for future machines, power consumption should also be taken into account, since in addition to performance, mapping can affect power consumption as well.

Indirect routing uses all of the links of the network, since every packet is routed through a random supernode. Therefore, it is very expensive in terms of network power and no energy reduction is possible. On the other hand, random mapping is shown to have similar performance as indirect routing on PERCS networks [Bhatele et al. 2011b]. Thus, indirect routing should be avoided and random mapping should be used to have much less power consumption but the same performance. The routes in random mapping are statically determined during the mapping phase, while indirect routing dynamically changes the routes for every packet. This suggests that different aspects of hardware and software design can affect power consumption of the network significantly, and power should be considered at every stage of the design.

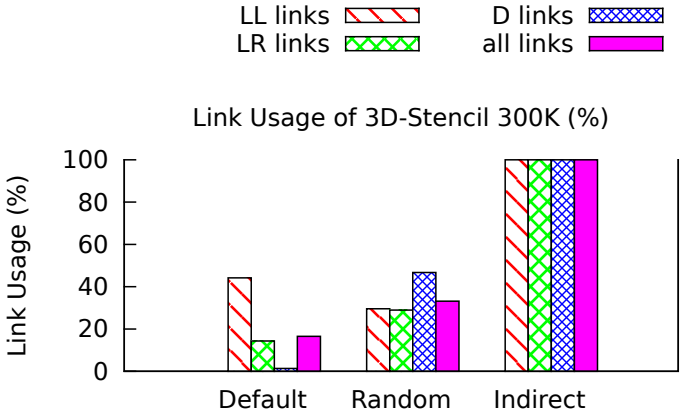


Fig. 9. Fraction of links used with different mappings

4. IMPLEMENTATION IN RUNTIME SYSTEM AND HARDWARE

The large amount of unused links, discussed in the previous section, presents opportunities for power optimization and savings. Although past studies suggest hardware and compiler techniques, we believe that this should be done by the runtime system. Hardware and compiler do not have enough information about the characteristics of the application, hence they may make conservative assumptions or cause unnecessary delays. For example, NAMD's communication depends on the input and previous iterations, and hence the compiler cannot assume any unused links. It is also difficult at the application level since it would hurt portability and programmer productivity.

On the other hand, runtime systems, such as MPI and CHARM++, have enough information about both the application and the hardware to make wise decisions. The runtime system obtains this information about the application by monitoring the communication performed as the application executes (with negligible overhead [Zheng et al. 2011]).

4.1. Runtime System Support

The runtime system mediates all the communications and computation, so it can instrument the application easily. Runtime systems, such as CHARM++, use this information for many purposes such as load balancing [Zheng et al. 2011] and power management [Sarood et al. 2012]. They also obtain characteristics of the network, such as its topology [Bhatele et al. 2011a]. Our approach requires only a small subset of this data to save network power: the communication graph of the application and the topology of the network. Using this information, our approach can turn off unnecessary links as follows. We assume that each node keeps track of the destinations of its messages. At the network power management stage, each node calculates the route for each of its destinations. It sends a notification message to each of the intermediate nodes to have them mark their used links. At the end, when a node has received all notification messages and marked its own links, it turns off all of its unused links. Note that collectives are mostly implemented as point-to-point messages in runtime, so they can be handled similarly. This algorithm, which is executed by every node, can be summarized as follows:

(1) Obtain the destination list of local messages.
(2) For each destination, calculate its route.
(3) Mark the local links used by local messages.
(4) For each intermediate node, instruct it to mark the required links.
(5) Wait for all notifications to be received (possibly using a termination detection algorithm).
(6) Turn off the local links that are never used either by local or non-local messages.

This algorithm needs to be invoked at appropriate times, which is feasible in most cases since scientific and many other parallel applications are usually iterative in nature. For the common case of static communication pattern, which encompasses all of our benchmarks except NAMD, every iteration follows a constant communication pattern. Thus, one invocation of the power management scheme (e.g. after the first iteration) is sufficient. Note that even in this simple case, the hardware cannot make wise decisions on its own, because it is not aware of the iteration time of the application and its window might be too small. In addition, hardware does not see the global picture of the application's message flow, since it usually works at the packet and flit levels.

For NAMD, the communication pattern between objects is static, but the objects may migrate between processors periodically, under the control of a load balancer. Therefore, the actual communication pattern varies. In this case, the new communication pattern can be determined by the runtime system at (or just after) the load balancing steps. Thus, the network power management algorithm needs to be called at every load balancing step. Even in this case, the switching delay of links is of negligible concern, since the runtime will not make any mistake in switching the links' states. In addition, since load balancing is not performed very frequently (usually once in thousands of iterations), our method will not add significant overhead. Many other dynamic (and phase based) applications, such as Adaptive Mesh Refinement (AMR) [Langer et al. 2012], can be handled in the same way.

4.2. Hardware support

For our approach, we only require the network hardware to implement links that can be turned off and on (or any other power saving means such as DVS), along with a software interface to control them. Note that the "off" state should usually be implemented as a very low power (but slow) state. Turning the links completely off may increase the switching delay significantly because the links would need "re-training". Not turning the links fully off also ensures connectivity of the network.

In a simple but robust implementation, the runtime provides hints to the hardware to turn a link off, but the hardware turns it back on if a message (packet) needs the link. That message will pay the penalty of switching delay because of the incorrect prediction by the runtime. In this way, we do not strictly require any change in routing and switching tables. Note that the runtime can measure the iteration times and turn off the power management algorithm, or adjust it, if the performance is degraded. This feedback loop ensures "safety" of our approach for performance if anything about the application or system changes. This safety cannot be provided easily by hardware or compiler approaches.

On some current machines (e.g. from Cray), network interference from other running jobs can decrease the predictability for the runtime and make the power management task more difficult. However, the other jobs running on the machine are most probably also iterative with predictive behavior, so the runtime can take them into account similarly. Note that job interference also has performance penalty [Bhatelé and Kalé 2009; Kerbyson et al. 2012], and many machines (e.g. Cray systems) are exploring new job schedulers for isolated partitions. Some other machines, such as Blue Genes, already allocate isolated partitions (prisms) for every job running on the supercomputer. I/O interference can cause similar issues if I/O is performed on some I/O nodes that are out of the job's allocation, using the same network as the application. Thus, I/O needs to be considered as well.

Impact of Adaptive Routing (for Tori). For many networks (especially tori), dynamic adjustments, such as adaptive routing for performance and fault tolerance, have been proposed before. The dynamic behavior may hinder our approach because it reduces predictability, resulting in performance penalties. However, the support for adaptive routing is still limited in current machines due to practical restrictions. For example, the K computer has a fixed, minimal dimension order routing [Ajima et al. 2011]. Some machines may support a limited form of adaptive routing, such as routing in "zones" on Blue Gene/Q [Chen et al. 2011]. For this case, the runtime needs to know the details of the routing protocol (what links are actually used for communication for messages of an iteration). This information is usually already available to the runtime system for communication performance optimization. Note that even for Blue Gene/Q, minimal dimension ordered routing is used for most messages depending on the system's configuration [Megan Gilge 2013]. Adaptive routing is usually used for demanding cases such as all-to-all, in which case, we do not turn links off. Fault tolerance can also be considered easily, since most faults bring down a whole node, calling the runtime system's fault tolerance protocol. Thus, we call the network power management method after every resiliency action of the runtime. It is notable that such dynamic behaviors will not result in disastrous performance penalties, since the runtime can measure the performance and correct itself.

5. POWER MODEL FOR NETWORK LINKS

Our power management in runtime system approach is very simple but offers substantial link power savings. However, it is useful to know how much saving is possible for each application on a network. Theoretical models that suggest upper bounds of link

power savings need to be developed for this purpose. Furthermore, a simple theoretical model can give insight about the application's utilization of a network and compare different networks. In this section, with these goals in mind, we develop a theoretical model for link utilization of a network while running an application. Our model provides an upper bound on power requirement of an application using a particular network. We make some assumptions that keep the model tractable at the expense of some loss in accuracy. We also suggest some additions to the model to make it more accurate and provide tighter bounds.

Suppose it is possible to switch a link on and off without any delay, and a link has a bandwidth of B. Assume also that there is no zero-size message latency. In this ideal case, each link can be turned on whenever there is some message traffic and can be off otherwise. Thus, a link only consumes power when it has to transfer data. If $B_i = 100\,\mathrm{MB/s}$ for link i, and a program transfers $100\,\mathrm{MB}$ of data through this link during its 10 seconds of execution, then link i is used only for 1 second. Thus, only 0.1 of its capacity was utilized according to this simple calculation:

$$\frac{100\,\mathrm{MB}}{(100\,\mathrm{MB/s}) * 10\,\mathrm{s}} = 0.1$$

Let us generalize this formula, assuming that each link i transfers z_i bytes of data during t seconds of program execution:

$$U_i = \frac{z_i}{(B_i t)}$$

In this formula, U_i represents the utilization of link i. We can derive the whole network's utilization by a sum over all the n links:

$$U = \frac{1}{n}\sum_{i=1}^{n} U_i = \frac{1}{n}\sum_{i=1}^{n} \frac{z_i}{(B_i t)} = \frac{1}{nt}\sum_{i=1}^{n} \frac{z_i}{B_i}$$

We also assume that $B_i = B$ are the same for all the links and derive the upper bound of power savings:

$$M = 1 - \frac{1}{(nBt)}\sum_{i=1}^{n} z_i$$

M is the fraction of network link's power that can be saved, given the hypothetical assumptions, e.g. no on/off delays. In this formula, n, B and t can be determined easily, but z_i depends on the application, mapping, network topology and routing algorithm. Note that we assumed on/off links without Dynamic Voltage Scaling (DVS) support. Using DVS for the links carrying messages that are not on the critical path may result in even greater power savings.

Let us calculate this formula for a simple case of 3D Stencil, running on a 3D torus. Assume that the processes are mapped to processors perfectly, i.e. in a way that communicating processes are neighbors in the network. If each iteration takes $10\,\mathrm{ms}$, each message is $2\,\mathrm{MB}$ and the bandwidth of each link is $1\,\mathrm{GB/s}$, we have:

$$M = 1 - \frac{1}{(n * (1000\,\mathrm{MB/s}) * (10\,\mathrm{ms})}\sum_{i=1}^{n}(2\,\mathrm{MB})$$

Resolving the summation we have:

$$M = 1 - \frac{n * (2\,\mathrm{MB})}{n * (1000\,\mathrm{MB/s}) * (10\,\mathrm{ms})} = 80\%$$

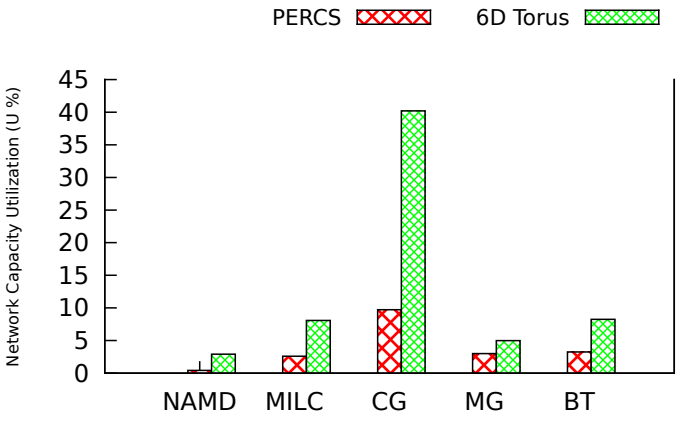


Fig. 10. Network capacity utilization of different applications

Thus, 80% of the links' power can be saved since only 20% of the time the links are being used. By using a perfect schedule for toggling the links, this power can be saved.

Calculating this formula is not usually as simple as this case, since z_i values are not easy to determine in many cases. Therefore, execution or emulation is required for finding the communication volume of each link, while running the application. However, this task is straightforward because the exact time that each link is used is not important for these values. First, communication traces can be obtained, even on a much smaller machine using an emulation approach, such as BigSim [Totoni et al. 2011; Zheng et al. 2004]. Then, a simple counting program can determine the path for each message and keep track of each link's communication volume. Using this method, we have determined the maximum possible link power saving for each application.

Figure 10 shows the network utilization of different applications on 6D Torus and PERCS networks. Except CG on 6D Torus network, the applications utilize less than 10% of the network. Thus, according to our model, more than 90% of the link power can be saved. Our basic approach can realize a significant portion of these savings with low effort for many applications. For example, more than 81.5% of the power can be saved for MILC by the runtime system. Next, we incorporate transition delay in our model and extend our basic approach accordingly for more savings.

6. EFFECT OF ON/OFF TRANSITION DELAY

Our basic model assumes zero on/off transition delay for simplicity. We also do not consider the conservative delay of scheduling in the runtime system to account for system noise and other overheads. We can incorporate this delay if we add one more assumption: *each iteration of an iterative application is divided into long distinct computation and communication phases*. Here, *long* means that a computation stage is considerably longer than the link transition delay, so we can turn the link off in that duration. This assumption is usually valid for common HPC applications, such as the ones with nearest neighbor communication pattern and/or bulk-synchronous parallel (BSP) model.

In addition to this assumption, we also assume that the links consume their full power during their transition and extend our model. If each link i transfers z_i bytes for each iteration that takes t seconds and the transition delay is d, the link consumes full power U fraction of the time:

$$U_i = \frac{z_i}{(B_i t)} + \frac{2p_i d}{t}$$

Here, p_i is a boolean variable that is one if the link is ever used, i.e. $z_i > 0$, and zero otherwise. This variable is needed because if the link is not used at all, it does not need to make any transition. Furthermore, since the runtime system turns the link off after each communication step and turns it back on before the next one, we pay the delay cost twice per iteration. Note that transition delay d should be smaller than half of the iteration time of the application, since utilization cannot be more than one. We define p as the fraction of the network links used for the application and z as total communication volume over links. Thus, we can derive the full network utilization:

$$U = \frac{1}{n}\sum_{i=1}^{n} U_i = \frac{1}{nBt}\sum_{i=1}^{n} z_i + \frac{2d}{nt}\sum_{i=1}^{n} p_i = \frac{z}{nBt} + \frac{2pd}{t}$$

This formula specifies the network utilization as a linear function of transition delay over the iteration time, so we can quantify the effect of the transition delay for each network. For practical cases, the transition delay is not a problem since the iteration time is much longer and the last term of the equation becomes small. For example, a typical short iteration time is around $10ms$, while some current implementations have a transition delay of around $10,000$ cycles ($10\mu s$ at $1GHz$). In this case, the transition overhead is just 1%.

Our approach may lead to overheads due to the software and hardware. In software, in order to capture the communication pattern and link utilization, the runtime system has to monitor the application. However, this should not have significant overheads since the monitoring is usually performed only once (because the applications are iterative) and its results are stored. Other overheads include the system call overheads (context switching overheads, argument verification overheads, etc.) because currently the runtime system is executed in the user space (which is likely to change in the future). Our experiments suggest that these overheads are limited to $20\mu s$ per call. In the hardware, as mentioned earlier, some current implementations have a transition delay of around $10,000$ cycles ($10\mu s$ at $1GHz$) for turning links on/off, and it is projected that it will improve much further (down to just 100 cycles) [Soteriou and Peh 2007]. Hence, the overall overheads should be less than $30\mu s$ per call that turns a link on/off.

Figures 11 and 12 show the link power savings as a function of transition delay (other overheads are also included) for PERCS and 6D Torus networks (using simulation). For many of our applications, we have short iteration times of around $30ms$ and we show the results with up to $15ms$ delay for illustration. Thus, as can be seen from the figures, transition delays and other overheads are not significant problems for our approach.

Note that this approach assumes accurate scheduling of links' on/off transitions by the runtime system, which is achievable since each iteration's message send and receive times are usually very deterministic. To verify this, we ran some of the NPB benchmarks on Blue Gene/P and inspected a sample of processors. We found that the message sends and receives occur with regular intervals and are predictable. The prediction error was usually less than $200\mu s$, while the iteration time is in hundreds of milliseconds. Thus, in our results, we consider $1ms$ conservative delay for the runtime system to incorporate noise and variations in the system. Figure 13 summarizes machine power saving potentials of our approaches for different applications on PERCS and 6D Torus networks. As before, this figure assumes that 30% of the machine power is consumed in the network and 65% of network power is consumed by the links. As

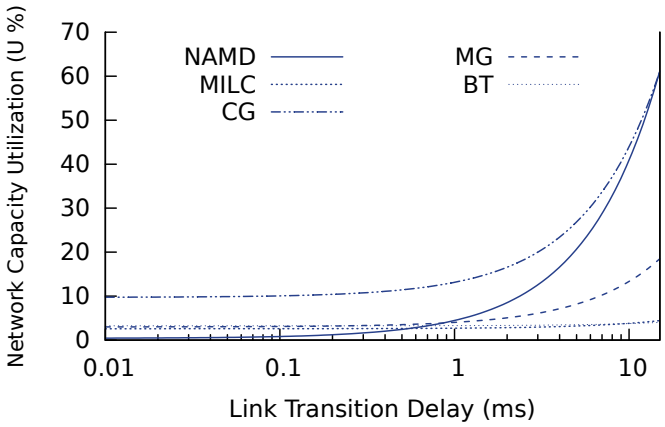


Fig. 11. Potential link power saving on PERCS network

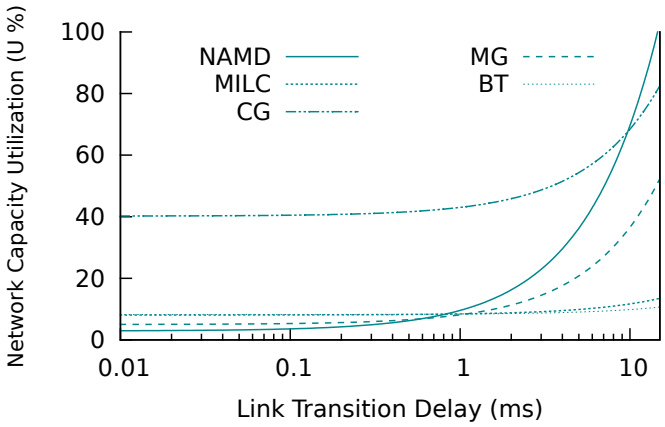


Fig. 12. Potential link power saving on 6D Torus network

can be seen, for most applications, our scheduling approach can save around 20% of the machine power. Our basic approach can also save significant power, usually more than 15% for PERCS and around 10% for 6D Torus. Note that in the case of NAMD_PME, the basic approach cannot save much and the scheduling approach is required.

7. CONCLUSIONS AND FUTURE WORK

With ever increasing communication demands of large-scale parallel systems, multi-level directly connected networks (Dragonfly, PERCS) and high dimensional tori are becoming more appealing. Optimizing the power and performance of these innovative networks presents a new challenge for parallel systems. We showed that many parallel applications do not fully exploit a significant fraction of the network links, which present opportunities for power optimization. Thus, a runtime system can optimize the power consumption of the links by turning off the unused ones, with minimal hardware support. This approach results in up to 20% saving of total system's power for common place applications with near neighbor communication.

For future work, less conservative approaches that turn off more links can be used, which may have some performance penalties. Furthermore, dynamic voltage scaling

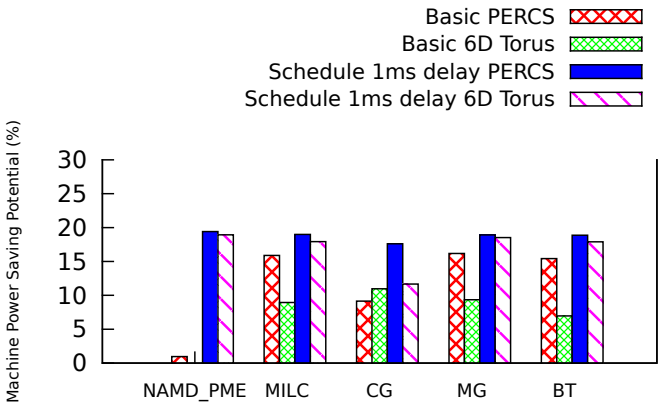


Fig. 13. Potential total machine power saving for different approaches

(or reducing the bandwidth) of the network links can be exploited for the links that do not transfer messages on the critical path. Overall, we suggest that more adaptive power management techniques by the runtime system for the network should be explored further.

REFERENCES

Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. 2010. Energy proportional datacenter networks. In *Proceedings of the 37th annual international symposium on computer architecture (ISCA '10)*. ACM, New York, NY, USA, 338–347. DOI:http://dx.doi.org/10.1145/1815961.1816004

Y. Ajima, Y. Takagi, T. Inoue, S. Hiramoto, and T. Shimizu. 2011. The Tofu Interconnect. In *High Performance Interconnects (HOTI), 2011 IEEE 19th Annual Symposium on*. 87–94. DOI:http://dx.doi.org/10.1109/HOTI.2011.21

Marina Alonso, Salvador Coll, Juan-Miguel Martínez, Vicente Santonja, Pedro López, and José Duato. 2006. Dynamic power saving in fat-tree interconnection networks using on/off links. In *Proceedings of the 20th international conference on Parallel and distributed processing (IPDPS'06)*. IEEE Computer Society, Washington, DC, USA, 299–299. http://dl.acm.org/citation.cfm?id=1898699.1898826

B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, Jian Li, Nan Ni, and R. Rajamony. 2010. The PERCS High-Performance Interconnect. In *2010 IEEE 18th Annual Symposium on High Performance Interconnects (HOTI)*. 75–82.

D.H. Bailey, E. Barszcz, L. Dagum, and H.D. Simon. 1992. NAS Parallel Benchmark Results. In *Proc. Supercomputing*.

Aaron Becker. 2012. *Compiler Support for Productive Message-Driven Parallel Programming*. Ph.D. Dissertation. Dept. of Computer Science, University of Illinois. http://charm.cs.uiuc.edu/media/12-44/.

Claude Bernard, Tom Burch, Thomas A. DeGrand, Carleton DeTar, Steven Gottlieb, Urs M. Heller, James E. Hetrick, Kostas Orginos, Bob Sugar, and Doug Toussaint. 2000. Scaling tests of the improved Kogut-Susskind quark action. *Physical Review D* 61 (2000).

Abhinav Bhatele, Eric Bohm, and Laxmikant V. Kale. 2011a. Optimizing communication for Charm++ applications by reducing network contention. *Concurrency and Computation: Practice and Experience* 23, 2 (2011), 211–222.

Abhinav Bhatele, Nikhil Jain, William D. Gropp, and Laxmikant V. Kale. 2011b. Avoiding hot-spots on two-level direct networks. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*. ACM, New York, NY, USA, 76:1–76:11.

Abhinav Bhatelé and Laxmikant V. Kalé. 2009. Quantifying Network Contention on Large Parallel Machines. *Parallel Processing Letters (Special Issue on Large-Scale Parallel Processing)* 19, 4 (2009), 553–572.

Dong Chen, N.A. Eisley, P. Heidelberger, R.M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D.L. Satterfield, B. Steinmacher-Burow, and J.J. Parker. 2011. The IBM Blue Gene/Q interconnection network and mes-

sage unit. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*. 1–10.

S. Conner, S. Akioka, M.J. Irwin, and P. Raghavan. 2007. Link Shutdown Opportunities During Collective Communications in 3-D Torus Nets. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. 1 –8. DOI:http://dx.doi.org/10.1109/IPDPS.2007.370534

Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard. 2012. Cray cascade: a scalable HPC system based on a Dragonfly network. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12)*. IEEE Computer Society Press, Los Alamitos, CA, USA, Article 103, 9 pages. http://dl.acm.org/citation.cfm?id=2388996.2389136

Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. 2010. ElasticTree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*.

Gilbert Hendry. 2013. Decreasing Network Power with On-Off Links Informed by Scientific Applications.. In *HPPAC'13*.

A.K. Jain and X. Yang. 2005. Modeling the effects of two different land cover change data sets on the carbon stocks of plants and soils in concert with CO2 and climate change. *Global Biogeochem. Cycles* 19, 2 (2005), 1–20.

Laxmikant V. Kale, Abhinav Bhatele, Eric J. Bohm, and James C. Phillips. 2011. NAnoscale Molecular Dynamics (NAMD). In *Encyclopedia of Parallel Computing (to appear)*, D. Padua (Ed.). Springer Verlag.

Laxmikant V. Kale and Gengbin Zheng. 2009. Charm++ and AMPI: Adaptive Runtime Strategies via Migratable Objects. In *Advanced Computational Infrastructures for Parallel and Distributed Applications*, M. Parashar (Ed.). Wiley-Interscience, 265–282.

D.J. Kerbyson, K.J. Barker, A. Vishnu, and A. Hoisie. 2012. Comparing the Performance of Blue Gene/Q with Leading Cray XE6 and InfiniBand Systems. In *Parallel and Distributed Systems (ICPADS), IEEE 18th International Conference on*. 556–563.

John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. *SIGARCH Comput. Archit. News* 36 (June 2008), 77–88. Issue 3.

PM Kogge. 2008. Architectural Challenges at the Exascale Frontier (invited talk). *Simulating the Future: Using One Million Cores and Beyond* (2008).

Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, and Katherine Yelick. 2008. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. (2008).

Akhil Langer, Jonathan Lifflander, Phil Miller, Kuo-Chuan Pan, Laxmikant V. Kale, and Paul Ricker. 2012. A Scalable Mesh Restructuring Algorithm for Distributed-Memory Adaptive Mesh Refinement. In *Proceedings of 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*.

James Laros, Kevin Pedretti, S. Kelly, Wei Shu, and C. Vaughan. 2012. Energy Based Performance Tuning for Large Scale High Performance Computing Systems. In *Proceedings of 20th High Performance Computing Symposium (HPC)*.

Jian Li, Wei Huang, C. Lefurgy, Lixin Zhang, W.E. Denzel, R.R. Treumann, and Kun Wang. 2011. Power shifting in Thrifty Interconnection Network. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*. 156 –167. DOI:http://dx.doi.org/10.1109/HPCA.2011.5749725

P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. 2009. Energy Aware Network Operations. In *INFOCOM Workshops 2009, IEEE*. 1–6. DOI:http://dx.doi.org/10.1109/INFCOMW.2009.5072138

Megan Gilge. 2013. Blue Gene/Q Application Development. http://www.redbooks.ibm.com/abstracts/sg247948.html. (2013).

Yanhua Sun Sameer Kumar and L. V. Kale. 2013. Acceleration of an Asynchronous Message Driven Programming Paradigm on IBM Blue Gene/Q. In *Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Boston, USA.

Osman Sarood, Phil Miller, Ehsan Totoni, and L. V. Kale. 2012. 'Cool' Load Balancing for High Performance Computing Data Centers. In *IEEE Transactions on Computer - SI (Energy Efficient Computing)*.

Li Shang, Li-Shiuan Peh, and N.K. Jha. 2003. Dynamic voltage scaling with links for power optimization of interconnection networks. In *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on*. 91–102. DOI:http://dx.doi.org/10.1109/HPCA.2003.1183527

Vassos Soteriou, Noel Eisley, and Li-Shiuan Peh. 2007. Software-directed power-aware interconnection networks. *ACM Trans. Archit. Code Optim.* 4, 1, Article 5 (March 2007). DOI:http://dx.doi.org/10.1145/1216544.1216548

V. Soteriou and Li-Shiuan Peh. 2007. Exploring the Design Space of Self-Regulating Power-Aware On/Off Interconnection Networks. *Parallel and Distributed Systems, IEEE Transactions on* 18, 3 (march 2007), 393–408. DOI:http://dx.doi.org/10.1109/TPDS.2007.43

top500 2013. Top500 Supercomputing Sites. http://top500.org. (2013).

E. Totoni, B. Behzad, S. Ghike, and J. Torrellas. 2012. Comparing the power and performance of Intel's SCC to state-of-the-art CPUs and GPUs. In *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*. 78–87.

Ehsan Totoni, Abhinav Bhatele, Eric Bohm, Nikhil Jain, Celso Mendes, Ryan Mokos, Gengbin Zheng, and Laxmikant Kale. 2011. Simulation-based Performance Analysis and Tuning for a Two-level Directly Connected System. In *Proceedings of the 17th IEEE International Conference on Parallel and Distributed Systems*.

Ehsan Totoni, Nikhil Jain, and Laxmikant V. Kale. 2013. Toward Runtime Power Management of Exascale Networks by On/Off Control of Links. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2013 IEEE International Symposium on*.

Gengbin Zheng, Abhinav Bhatele, Esteban Meneses, and Laxmikant V. Kale. 2011. Periodic Hierarchical Load Balancing for Large Supercomputers. *International Journal of High Performance Computing Applications (IJHPCA)* (March 2011).

Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V. Kalé. 2004. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *18th International Parallel and Distributed Processing Symposium (IPDPS)*. Santa Fe, New Mexico, 78.