A Performance Perspective on Energy Efficient HPC Links

Karthikeyan P. Saravanan, Paul M. Carpenter, Alex Ramirez
Barcelona Supercomputing Center (BSC)/ Universitat Polit‘ecnica de Catalunya (UPC)
karthikeyan.palavedu@bsc.es, paul.carpenter@bsc.es, alex.ramirez@bsc.es

ABSTRACT

Energy costs are an increasing part of the total cost of ownership of HPC systems. As HPC systems become increasingly energy proportional in an effort to reduce energy costs, interconnect links stand out for their inefficiency. Commodity interconnect links remain “always-on”, consuming full power even when no data is being transmitted. Although various techniques have been proposed towards energy-proportional interconnects, they are often too conservative or are not focused toward HPC. Aggressive techniques for interconnect energy savings are often not applied to HPC, in particular, because they may incur excessive performance overheads. Any energy-saving technique will only be adopted in HPC if there is no significant impact on performance, which is still the primary design objective.

This paper explores interconnect energy proportionality from a performance perspective. We characterize HPC applications over on/off links and propose PerfBound, a technique that reduces link energy, subject to a bound on the application’s performance degradation. We also propose PerfBoundRatio, which maintains the same performance bound across an entire hierarchical network. Finally, we propose PerfBoundPredict, which improves energy savings using an idle time prediction mechanism. Even when predictions are inaccurate, the performance degradation is still bounded. The techniques require no changes to the application and add no communication between nodes and/or switches. We evaluate our techniques using HPC traces from production supercomputers. Our results show that, configured with a 1% performance bound, 13 out of 15 applications are inside the bound, and average link energy savings is 60% for PerfBound and 68% for PerfBoundPredict.

Keywords

Performance overheads bounding, Energy Efficient Ethernet, Energy Proportional Interconnects, Idle time prediction

1. INTRODUCTION

Over the past decade, the field of HPC has become increasingly concerned by power consumption and energy efficiency. This is especially true in the design of future exascale systems, which will only be practicable through a dramatic improvement in energy efficiency[1].Successive technological advances in micro-architecture and process technology have not only sustained tremendous performance scaling, but has also considerably increased performance per watt. With energy optimized processors and memory, attention is moving towards the interconnect.

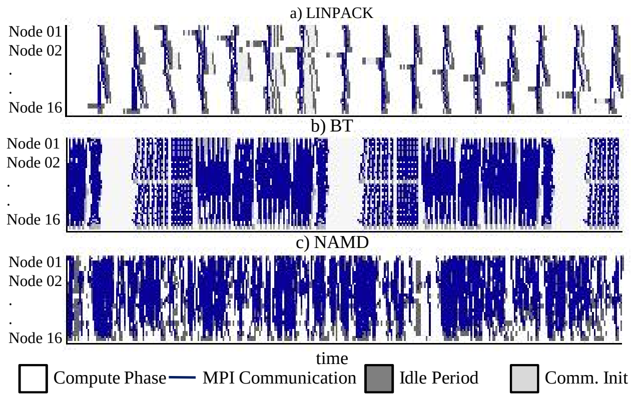

Figure 1: **Communication behavior in HPC applications - LINPACK[2], BT[3] and NAMD[4]**

Interconnects contribute a significant portion of the system’s energy consumption. D. Abts et al. [5] recently showed that a typical interconnect consumes 12% of the total system power at full load, and more when the application does not fully utilize the CPU and memory. Improved energy proportionality in the compute elements naturally increases the proportion of energy consumed by the interconnect. The main reason for a lack of energy proportionality is that interconnect links, which consume up to 65% of the total interconnect power, are essentially “always-on”, continually transmitting signals, even when idle, in order to maintain alignment and synchronization [6, 7, 8].

HPC applications require a high-performance interconnect, to support their peak communications demand, but the average utilization of the network is low. Moreover, much of the interconnect’s idle time is contributed by relatively long idle periods [9, 10]. Figure 1 shows the communication behavior of LINPACK [2], BT [3], and NAMD [4]. Fourteen of the fifteen applications examined in this work exhibited regular patterns similar to Figures 1(a) and 1(b). These applications have short intensive communication bursts, separated by long computation phases, during which the interconnect is idle. The final application, NAMD, shown in Figure 1(c), appears irregular, but it still exhibited low network utilization and considerable interconnect energy savings.

Several proposals attempt to exploit the above-mentioned opportunities to save energy [5, 11, 12, 13, 14]. These proposals are built upon one of the following underlying mechanisms. Firstly, *on/off links* are powered down during idle periods. An important example is IEEE 802.3az Energy Efficient Ethernet (EEE) [8, 15], approved in 2010, which is primarily designed to save network power consumption in homes, offices and data centres. Alternatively, *bandwidth tunable links* adapt the network bandwidth to the communication requirements, reducing the frequency or the number of channels when demand is low, and therefore also reducing the power consumption. An important example is InfiniBand, which implements variable bandwidth as well as variable active 1× links. Effective use of on/off links is especially important, given that it is the mechanism implemented in Energy Efficient Ethernet, and together, 1Gb and 10Gb Ethernet account for 43% of the systems in the November 2013 TOP500 list.

In both cases, changing power state incurs a delay; for example, EEE on a 10Gbps link requires 3μs to sleep and 4μs to wake. Also, the physical layer specification provides the underlying mechanisms, but the decisions as to when to enter and leave power-saving states are left to the vendor. These decisions are critical, especially in HPC, for which, although energy-efficiency is increasingly important, the primary design objective is still performance. Any proposed energy-saving technique will only be adopted if there is no significant reduction in performance.

This paper introduces *PerfBound*, a link energy saving technique for on/off links that reacts to performance overheads. The only parameter is a limit on the performance degradation, which was set to 1% in the evaluation. PerfBound is self-contained, in that the application is not modified and decisions are taken using local state, without any additional communication between nodes and/or switches.

In a multi-hop network, each link in the route may implement power-saving techniques, each of which may incur latency, multiplying the performance overheads. We therefore propose *PerfBoundRatio*, which maintains the same application performance target, across the whole hierarchical on/off network, by automatically adjusting to the application's communication locality. As for PerfBound, the application is not modified, decisions are taken using local state only, and there are no application-dependent parameters.

Finally, we propose *PerfBoundPredict*, which adds an idle time prediction mechanism, based on techniques used in CPU branch predictors. PerfBoundPredict exploits the fact that HPC application communication patterns are typically repetitive, and, when the idle period is predicted to be long, it enables the link to enter sleep mode without first waiting for the timer to elapse, which would otherwise incur unnecessary energy consumption. It also allows the link to be turned back on in time for the next message, avoiding the wake overhead. The interaction with PerfBound or PerfBoundRatio ensures that, even though prediction may be incorrect, the total performance degradation is still controlled.

In summary, in order for HPC to adapt energy proportional interconnects, its crucial that performance overheads caused by the same are controlled. In this regard, the key novelty behind our work is that we examine and propose on/off link management mechanisms that account for performance degradation. To be specific, the novel contributions of this paper are as follows:

1. A detailed analysis of the communication behavior in HPC applications provides insights on the correct management of on/off links. We identify that, for the application to remain within a given performance overhead bound, a certain number of messages, per unit time, can be allowed to incur wakeup delays. We show how the energy savings depend on making the right choice of messages to delay.
2. We use the above insights to propose PerfBound, a technique that saves energy, subject to a bound on the performance degradation. PerfBound monitors the number of wake-up delays and it adjusts the internal parameters to become more or less aggressive, optimizing energy savings subject to the performance overhead bound. We also propose PerfBoundRatio, which respects the same bound on the total overhead in a hierarchical network.
3. Finally, we propose a prediction mechanism for predicting link idle period durations. Knowing the duration of the next idle period allows the link to be turned off immediately, when doing so is appropriate, and it allows the link to be turned back on in time for the next message, avoiding overheads. Prediction is disabled when idle periods are unpredictable. In addition, prediction is always controlled by PerfBound, and disabled when mis-prediction could breach the performance bound.

The rest of this paper is organized as follows. The next section discusses the background and prior work. In Section 3, we investigate the causes of performance degradation and, based on insights gathered, make a case for performance bounding in interconnects. We also propose our algorithms and discuss their technical details. In section 4 & 5, we discuss the evaluation results of our techniques and conclude.

2. BACKGROUND AND RELATED WORK

As discussed in the introduction, interconnect energy proportionality can be supported in two main ways: either through on/off links; e.g. Energy Efficient focuses on on/off links, for two main reasons. Firstly, bandwidth tunable Ethernet, or through varying the bandwidth; e.g. InfiniBand. This paper links at low power mode are still "always-on" at their lowest bandwidth. Recent work [5] found that in the lowest energy state, bandwidth tunable links consume 40% of their maximum power consumption. In contrast, when an on/off link is switched off, it typically consumes about 10% of peak power. Secondly, on/off links are used in Energy Efficient Ethernet, as described below. Although this paper does not discuss bandwidth tunable links any further, the contributions, specifically idle time characterization and prediction, can also be applied to bandwidth tunable links.

Energy Efficient Ethernet (EEE): In 2010, the IEEE 802.3az Energy Efficient Ethernet Task Force published its standard for Ethernet energy efficiency [8, 15]. Since Internet infrastructure is primarily built using Ethernet, the mandate of the task force was to reduce the significant contribution of these network devices to the national power budget[8, 11]. After considering various proposals, including adaptively changing the link rate, the task force adopted the proposal known as Low Power Idle (LPI)[8, 15].

Low Power Idle (LPI) proposes modifications to existing Ethernet standards that allow the link to switch between "sleep" and "wake" modes on demand, to save energy. At low power mode, the link is still periodically refreshed and awaits frames, hence is not completely off. Arrival of a frame triggers the signaling of a wake up transition to turn on the link. Frame transmission starts when both the transceiver and receiver PHY are both active. At the subsequent hop, arriving frames are buffered while a subsequent link is signalled for wake-up. LPI was considered straightforward to implement, since it freezes the state of the transceiver when the link enters sleep and it restores the state when it wakes[15]. Switches that support EEE, targeting data centers, are already commercially available.

The Ethernet family of interconnects is used in the largest share of systems in Top500; specifically, 42.5% of the systems in the Nov. 2013 list use 1Gb/10Gb Ethernet. The growing popularity of Ethernet in HPC, coupled with the need for energy proportionality, makes a strong case for performance-aware techniques for EEE. We believe that the insights presented in this paper could help vendors in designing EEE technology for HPC and the standardization effort in the upcoming EEE for 40/100Gb links (IEEE P802.3bm). To this end, as discussed in the methodology section, the figures presented in this paper use timing information from the EEE specification.

2.1 Related Work

Jian Li, et al., [11] discuses the on/off networks that use snoop messages that arrive at the NICs as an indication of an impending message. In nodes that have snoop-based coherence, snooping messages would arrive at the link before an impending message, which can be used to trigger the link on, before the actual arrival of the message. They also propose the use of an always-on control network that sends control signals through the routing path of a message to wake up other links. They further propose software enhancements which would have programmers annotate the code before impending messages. Similarly, Soteriou, et al., [16] show that on/off networks incur a large performance penalty and hence, they propose software mechanisms such as parallelizing compilers for network power savings.

Gupta, et al., in their work [17], show opportunistic sleeping of links is possible, their technique however increases mean latency. Vassos, et al., [18] discusses a design space analysis for on/off based links. They propose using multiple routing paths available in torus like networks to shut down parts of the network during periods of low load. They evaluate their proposal with message arrivals following a Poisson process. Similarly, Alonso, et al., [19] propose shutting down redundant links (sub-trees) in their fat-tree system to save energy. These proposals do not discuss performance impact of bursty communication that is typical of HPC. Similarly Ethernet evaluation reports [7, 8] also use synthetic benchmarks to evaluate on/off networks. Relevant work on Energy Efficient Ethernet [7, 8, 15, 20, 21, 22, 23] provide detailed evaluations on EEE for its potential for desktop and IT based systems; however, do not target HPC workloads. Totoni, et al., [24] show that not all links of a network executing an HPC application are utilized hence propose runtime techniques to find links in networks that are never utilized and turn them off. However, their work does not adaptively turn on/off links.

D. Abts et al., [5] proposed energy proportional interconnects based on reducing the link rates of aggregated links. In their approach, during periods of inactivity, link rates are reduced to a lower link bandwidth to save energy. Work by Kim, et al., [14] evaluate energy proportional networks and compare links based on dynamic voltage scaling and on/off links. They show that dynamic voltage scaling in links causes significant increase in latency and show that on/off based techniques perform comparatively better. Our previous work [10], presents a case for Energy Efficient Ethernet. We show that however, increase in latencies due to wake-up could be harmful to certain HPC applications. We show that having the link on for a static period of time after link becomes idle, reduces performance overheads. However, the previous work is a static approach and cannot with all applications. The difference between our work and the above proposals is that, our work targets adaptive link energy savings accounting for application performance degradation.

Yoshi, et al., [25], propose ATPT - a prediction mechanism to find message sizes with src-dest pairs. The show that src-dest pairs could be used to improve prediction accuracy. When the size of the next message size is known, they tune the network frequency to the requirements of the next message size. Their work however does not predict idle link periods which are required for on/off based networks.

3. ENERGY EFFICIENT LINKS FOR HPC – A PERFORMANCE PERSPECTIVE

In this section, we discuss our performance-centric approach to energy efficiency in on/off based networks. We present methodology and motivation and our proposed techniques in this section.

3.1 Methodology

We used an extension of the Dimemas cluster simulator, which has been found to be accurate to within 10% and validated against production supercomputers, including Blue Gene/L, P, Q, and three generations of MareNostrum [26, 27, 28]. We modified the network model to support a hierarchical network, with on/off links controlled by our proposed techniques. The simulation infrastructure is driven by traces, which record CPU intervals and MPI events, independent of the network configuration, measured from a real execution on MareNostrum. The CPU intervals are scaled by relative CPU performance. MPI events imply dependencies, which ensure correctness. Link energy consumption is modelled as 100% when "on" or during transition between on/off states, and 10% when "off". All figures are normalised to percentage of original energy-to-solution, rather than joules.

The simulator is configured to model a cluster with a three-level hierarchical network. Applications are executed on 64, 128 or 256 nodes, grouped into 8, 16, or 32 nodes per rack, respectively, forming eight racks in total. Nodes are connected to the top-of-the-rack switch, which is in-turn connected to a two-level, four-node 2-ary fat tree. The network is statically routed, cut-through flow-control with fully duplex links. Each node is a two-socket high-end CPU with 225GFlops (based on Top500 machines with two Intel Xeon sockets). The switch latency is configured at 320ns for the first hop and 80ns for subsequent hops. Edge links are configured at 20Gb/s, while the higher two levels are 40Gb/s and 100Gb/s respectively. The two directions of the full-duplex links can be turned on and off separately. We use the wake-up and sleep times of 4μs and 3μs for Energy Efficient Ethernet [8, 15].

We used fifteen HPC applications. The original traces were large, on the order of hundreds of gigabytes, so simulation was done for a few iterations of the outer loop, as shown in Figure 1. Traces obtained for ALYA[29], LINPACK[2], BT [3], CG[3], FT[3] and MG[3] where executed on 256 nodes, QUANTUM[30], WRF[31] MILC[32], GROMACS[33] and GADGET[34] on 128 nodes, and NAMD[4], PEPC[35], SP and LU[3] on 64 nodes respectively.

3.2 Motivation

Although the EEE standard defines mechanisms for entering and leaving the sleep mode (low power idle), it does not define how to decide when to do so. A naive, and aggressive, technique is to always turn the link off as soon as it becomes idle and to turn it back on only on demand. There is, however, a fundamental trade-off between energy savings and performance overhead: aggressive techniques, such as the above save more energy but may introduce too much network latency, whereas conservative techniques incur a low performance overhead but they achieve little energy savings.

One difficulty in HPC is that different applications react differently to increases in latency. Figure 2 shows a sensitivity analysis of application performance to wake-up latency, assuming the naive management technique. The x-axis is the wake-up latency (which for EEE is about 4μs/link). Applications that are least sensitive, including Quantum[30] and BT[3], can potentially tolerate an aggressive energy saving technique, since the naive approach incurs only about 2% performance overhead. In contrast, GROMACS[33] and NAMD[4] have unacceptable performance degradation, with their execution time roughly doubled, so they require a rather conservative energy saving scheme.

There are two questions related to the management of on/off links: when to turn the link off, and when to turn it back on. An ideal solution, which obtains maximum energy savings, is to turn the link off immediately after each message and to turn it back on at the correct time in anticipation of the next message (if the idle period is shorter than the sum of the sleep and wake times, then the link is, of course, not switched off). This scheme, however, requires an accurate and precise prediction of the arrival time of the next message. If the prediction is wrong, then, either the link is woken up too late, incurring a performance overhead, or too soon, wasting potential energy savings.

A simple mechanism that can work well is to turn the link off only after a specific duration of idle time, which we call the **LinkOFF threshold**, and to turn it back on when the next message arrives. The naive approach described above corresponds to LinkOFF=0. A previous study [10] found that this approach can work well in HPC. Since different applications have different sensitivities to increases in latencies, the optimal value of LinkOFF depends on the application. This paper proposes PerfBound, which determines the correct LinkOFF threshold to obtain maximum energy savings subject to a performance bound. It also proposes PerfBoundRatio, which extends the scheme to cover hierarchical on/off networks.

Using the LinkOFF timer works well for both short idle periods, for which the link correctly remains on throughout, and long idle periods, for which its disadvantages are negligible: the energy consumption before the timer elapses is small, and so is the performance overhead of waking on demand. It works less well if there are a large number of idle periods of intermediate duration. We therefore propose PerfBoundPredict, which adds an idle time predictor. Since HPC application communication patterns are often repetitive, the idle time predictor is often able to provide an accurate prediction of the length of the idle period. If the idle period is predicted to be large enough, the link is switched off immediately and switched back on in time to avoid the wake overhead on the following message. This method avoids the energy consumption otherwise incurred

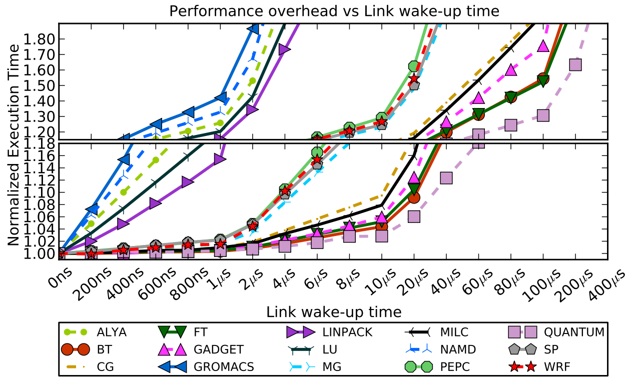

Figure 2: **Application performance overhead as a function of wake-up delay. Note: Y-axis scale varies from 1.00 to 1.15 at 0.02 increments and 1.15 to 2.00 at 0.10 increments.**

before the LinkOFF threshold has elapsed, and it allows the link to be switched off inside much shorter idle periods, since the associated wake up latency is usually avoided. An important disadvantage of prediction is the potential performance impact of mis-prediction. Interaction with the performance bounding mechanism of PerfBound ensures that even when the prediction is not possible or incorrect, the total performance degradation is still controlled.

Our first key insight, in the development of PerfBound, is that, since every time the link is switched off, one message will later be delayed by the wake-up time, the performance overhead is approximately proportional to the number of times the link is switched off. This is an approximation, since the method cannot track chains of dependencies among nodes. Tracking dependency chains requires either that the user or compiler annotates the application, or that additional messages are sent by the run-time system and monitored by the switches. Either approach adds complexity, with the result that such a proposal would be unlikely to be adopted in practice. We believe, on balance, that our approach gives the right compromise, especially since the results, described in Section 4, show that this approximation is generally sound. In summary, the performance overhead bound translates to a fixed number of messages, per unit time, that can be delayed. The following heuristics ensure that this number of delayed messages is not exceeded, and that the right choice of messages to delay is made, to get the maximum energy savings.

3.3 Understanding the overhead of link wake-up and idle time predictability

In order to make overhead-aware decisions for link energy savings, it is important to first understand how wakeup latencies translate to performance overheads. In Figure 2, we showed that different applications have different sensitivities to wake-up latencies. To look at this question in more detail, we examine the application overheads by applying the wake-up delays selectively. From this point in this paper, we refer to "message inter-arrival periods" as **idle link events**.[1]

Figure 3 shows a sensitivity analysis plot relating the LinkOFF threshold, on the x-axis, to application performance. As mentioned previously, the LinkOFF threshold controls the time for which the link must be idle, but kept

[1]In this paper, we term any duration during which no data is transmitted over a link as an *idle link event*

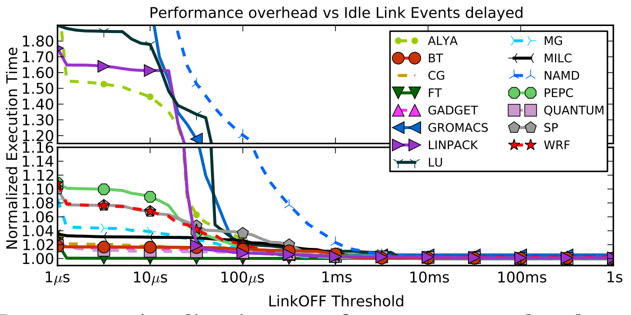

Figure 3: **Application performance overhead as LinkOFF threshold is varied - normalised to execution over an always-on network.**

on, before it is turned off. At low values of the LinkOFF threshold, the links turn off after many short idle periods, which translates to high performance overheads, due to the latency of frequently turning back on when required. As the LinkOFF threshold is increased, the performance overhead drops, eventually approaching zero. This is because, as the threshold is increased, the number of idle link events that exceed the threshold decreases towards zero. If an acceptable level of performance overhead for the application is 5%, for example, then Figure 3 can be used to determine an application-dependent static value for the LinkOFF threshold. For application LU, for instance, we can see that an appropriate value of the LinkOFF threshold would be 80μs. In this case, the link remains on for the first 80μs in each idle period, saving power on all idle link events that are longer than this, but maintaining performance overhead inside the specified bound of 5%.

The application behavior can be understood in greater detail, from the perspective of idle link events, by looking at the heatmaps in Figure 4. All sub-figures show the length of the current idle link event on the x-axis and the length of the next idle link event on the y-axis, for LINPACK[2], BT[3] and NAMD[4] according to the title. Figures 4(a), 4(c) and 4(e) are colored according to the *number of events*, whereas Figures 4(b), 4(d) and 4(f) are colored according to the total idle time contributed by those events. That is, if in Figure 4(a) there are 100 events in position (2ms, 2ms), then their total idle time would be 200ms. The idle link event heatmap gives a sense of the most common idle durations, which is helpful for prediction, and the total idle time helps understand how the idle time translates to energy savings. The results in these figures are averages across all edge links in the network.

Both applications LINPACK and BT are typical examples of HPC applications and shown, the difference between Figures 4(a) and 4(c) is in the clustering of idle link events. In the case of LINPACK, in Figure 4(a), the events are clustered at around 10μs, while the events in BT are clustered at around 1ms. Another key difference between these applications is clearly seen in Figures 4(b) and 4(d): the majority of the idle link events of LINPACK are of 10μs, but most of its total idle time comes from events that are longer than 10ms, even though there are few of them. A similar behavior can be seen in BT, where a small number of events longer than 10ms also contribute to a significant amount of total idle time. The main difference for BT is that its most common idle link event duration also contributes significantly to the total idle time. Further, as mentioned in the introduction, NAMD is an outlier in our set of applications. In

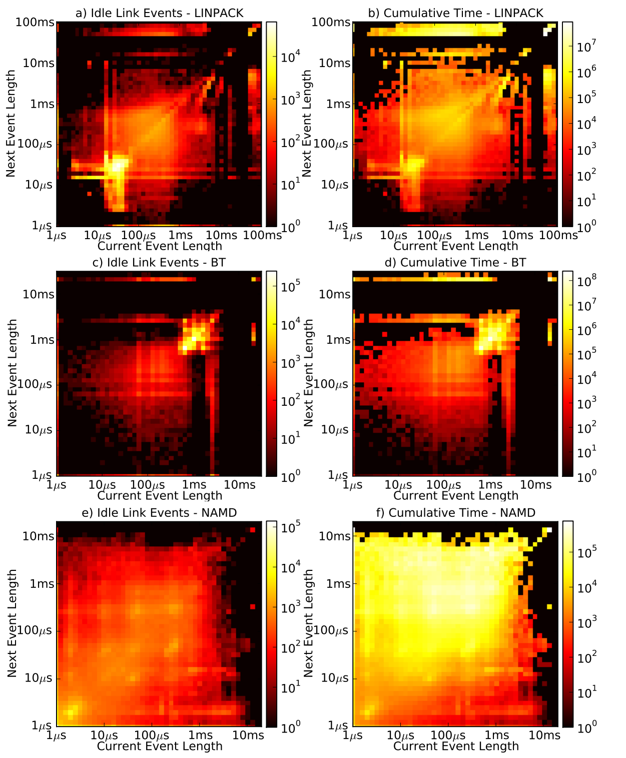

Figure 4: **Idle Link Event distributions of LINPACK(a,b), BT(c,d), NAMD(e,f) - (a),(c),(e) Heat map of the idle link event duration; (b),(d),(f) Heat map of total idle time (Number events $\times$ duration)**

Figures 4(e) and 4(f), it is clear that the application is irregular. In the context of predictability, for any current idle link event of 4(e) there are no event clusters that have an especially high probability. In other words, in the case of NAMD, given knowledge of the current event's length, the next event could have any length. As shown in Figures 4(a) and 4(c), in contrast, LINPACK and BT show reasonable predictability. In the case of LINPACK, for any event of size between 10μs and 100μs, there exists a high probability that the next event is the same size; similarly in the case of BT, for events of between 1ms and 10ms.

Comparing Figures 3 and 4, we can explain the observed performance overheads. Firstly, note from Figure 3, that the performance overhead of LINPACK remains at about 60% until the LinkOFF threshold is increased to 10μs, where it drops to about 2% between 10μs and 100μs. Comparing that to Figure 4(a), the performance overhead has clearly dropped as the LinkOFF threshold crossed the cluster between 10μs and 100μs. In other words, if the link remains on for about 100μs, none of the events in the cluster in Figure 4(a) would incur performance overheads. Similarly, in the case of BT, comparing Figures 3 and 4(c), we can see that the performance overhead of BT, starting from 2%, drops to near zero at about 1ms; this correlates to the clustering found at 1ms in Figure 4(c). Finally, in the case of NAMD, since there are no clusters and the distribution of events is uniform, we find a gradual decrease in performance overhead, as LinkOFF is increased, falling below 1% above a threshold of 2ms, which correlates with Figure 4(e). Note that the clustering observed in Figures 4(a) and 4(c) are due to repetitive patterns in these applications (as seen in Figure 1), which are not seen in irregular application NAMD.

Furthermore, BT has low performance overhead, even at low values of the LinkOFF threshold, because, first, at low threshold values in Figure 4(c), no events exist to incur performance delays. Since, for BT, the number of events that exist between 1μs and 1ms is low, the reduction in the performance overhead is gradual. Secondly, for large events, the ratio of event size to delay incurred is very low. To illustrate, when a delay of 1μs is applied to an event of 1ms, the added delay corresponds to 0.1%. For LINPACK, most events are clustered at 10μs, so if a 1μs delay is added to them, each delay adds 10% of the idle time, translating to large performance overheads.

3.4 PerfBound: Bounding performance overheads in on/off HPC links

The application analysis in Section 3.3 provides several key insights. First, the application overhead is roughly proportional to the number of delayed idle link events. Secondly, the application overhead can be adjusted using the LinkOFF threshold. Thirdly, the best LinkOFF threshold depends on the application, so the algorithm itself must be dynamic, adaptive and application independent. Finally, from an energy standpoint, it is best to delay the events of longest duration. Based on the above, we propose PerfBound and PerfBoundRatio.

The only parameter to the algorithms is a limit on performance degradation, which we set to 1% in the evaluation. For the purpose of the exposition, we assume that the limit is 1%, but it should be clear how to make the bound into a parameter. The approach is to first determine how many idle link events can be delayed per unit time before the overhead reaches 1%, and then to ensure that the right number of events are delayed and that they are the longest ones. The latter is done by dynamically adjusting the LinkOFF threshold.

3.4.1 Calculating the #events that can be delayed:

We first analyse the case where there is a single hop between two nodes. Since the overhead is assumed to come only from delayed wakeup events, the maximum number of them that can be tolerated, within a 1% bound, in a period of length X is simply $0.01X \,/\, T_w$, where T_w is the wakeup delay and 0.01 corresponds to the 1% bound. As X increases, the total number of events that can be delayed also increases, in proportion. This is the value used by **PerfBound**, when configured with a local performance bound of 1%. The next section will describe how the LinkOFF threshold is adjusted to delay the correct number of events.

In a multi-hop network, each link in the route may implement PerfBound, multiplying the resulting performance overhead. A three-level network has a maximum hop count of six, so a single message may incur cumulative wakeup delays on three upward links and three downward links. Using the above equation directly leads to a total overhead of up to 6%. The simplest solution is to divide the global 1% performance bound equally among the links, so that each link uses PerfBound with a local performance bound of 0.166%.

This is, however, unnecessarily conservative. An application that mainly communicates at Level 0 (say), would rarely incur overheads at the upper levels, meaning that the overhead is actually being constrained to 0.33%. Although a lower overhead is better, all else being equal, the configured 1% performance bound would probably have led to greater energy savings. Our solution for multi-hop networks is

$$\begin{tikzcd}
\textbf{Level 2} & & & & & L2 \arrow[dr] & & & \\
\textbf{Level 1} & & & & L1_0 \arrow[ur] \arrow[dr] & & L1_1 \arrow[dr] & & \\
\textbf{Level 0} & & & L0_0 \arrow[ur] \arrow[dr] & & L0_1 \arrow[dr] & & L0_2 \arrow[dr] & \\
\textbf{Nodes} & & T \arrow[ur] & & S_0 & & S_1 & & S_2
\end{tikzcd}$$

Figure 5: **Example network topology**

Link	Total messages	Messages/level			Proportion to level		
		L0	L1	L2	L0	L1	L2
T to L0$_0$	1000	500	400	100	0.5	0.4	0.1
L0$_0$ to S_0	500	500	0	0	1.0	0	0
L0$_0$ to L1$_0$	500	0	400	100	0	0.8	0.2
L1$_0$ to L0$_1$	400	0	400	0	0	1.0	0
L0$_1$ to S_1	400	0	400	0	0	1.0	0
L1$_0$ to L2	100	0	0	100	0	0	1.0
L2 to L1$_1$	100	0	0	100	0	0	1.0
L1$_1$ to L0$_2$	100	0	0	100	0	0	1.0
L0$_2$ to S_2	100	0	0	100	0	0	1.0

Table 1: **PerfBoundRatio: Example calculation of local state, when 50%, 40% and 10% of messages reach levels 0, 1 and 2, respectively.**

PerfBoundRatio, which is configured with a global performance bound. It adapts dynamically to the locality of the application's communication pattern, using only the information that is already available locally at the switch. In order to use PerfBound, each switch must be given enough information about the network topology to be able to calculate the level of the highest switch in the route between any pair of source and destination IP addresses. This may require specific configuration, but for an HPC system, such configuration is tolerable.

We explain PerfBoundRatio using the example three-level network in Figure 5. The switches are labeled with the level and a unique number; e.g. L1$_1$ is one of the switches in level 1 and nodes are labeled T and S_0 to S_2. Let us assume, node T transmits 1000 messages in total, to S_0, S_1 and S_2, in proportion 50%, 40% and 10%, respectively. In a real application, all nodes will transmit, with different distributions to various nodes, but the total counts are simply the sums of the various contributions, and the algorithm still works. It can be best understood by looking at a simple case.

Each link has four counters, one that counts the total number of messages over the link, and three messages/level counters, each corresponding to a level in the network. The messages/level counter for level n counts the number of messages seen whose highest level in the network is exactly n. This information is summarized, for all links, in Table 1. This table also shows the proportion of messages that go to each level, found by dividing by the total number of messages. For example, the link between L0$_0$ and L1$_0$ sees all messages from T that go to either S_1 or S_2. There are 500 such messages, of which 400 go to S_1, reaching level 1, and 100 go to S_2, reaching level 2. The ratios of messages that reach levels 0, 1 and 2, respectively, are 0, 0.8 and 0.2.

The key idea is to divide the global performance bound according to the behavior of the communication traffic. Of the 500 messages that are seen over the link between L0$_0$ and L1$_0$, 80% of the messages that reach network level 1, have four hops on their route, whereas the 20% of messages that reach network level 2, have six hops. The local performance bound is therefore given by $0.8 \times \frac{0.01}{4} + 0.2 \times \frac{0.01}{6}$. In this equation, the weighing factors of 0.8 and 0.2 are given by

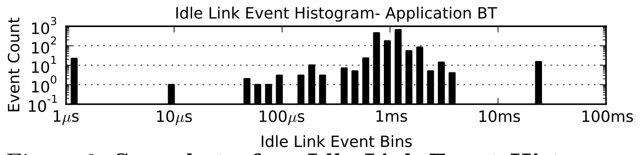

Figure 6: **Snapshot of an Idle Link Event Histogram**

the message statistics, 0.01 is the global performance bound, and the denominators are the numbers of hops on the routes.

In general, for a particular link, let $LC0$ be the total number of messages that reach maximum level 0, $LC1$ be the total number that reach maximum level 1, and $LC2$ the total number that reach level 2. Let $LC = LC0 + LC1 + LC2$ be the local total message counter. Then the local performance bound (as shown in Table 1) for that link is given by

$$l = \frac{LC0}{LC} \cdot \frac{0.01}{2} + \frac{LC1}{LC} \cdot \frac{0.01}{4} + \frac{LC2}{LC} \cdot \frac{0.01}{6}$$

3.4.2 *Calculating the LinkOFF threshold:*

After calculating the total number of events that can be delayed, per unit time, the final step is to determine the LinkOFF threshold. As described in Section 3.2, the LinkOFF threshold is the duration of time that the link must remain idle before it is switched off.

The LinkOFF threshold is determined from a histogram of idle link events. In detail, at the end of every idle link event, one new data point is available. This data point is the length of the previous idle link event. As shown in Figure 6, the bin corresponding to this length is determined and its histogram value is incremented. The histogram therefore keeps track of the distribution of link idle interval lengths, and its total mass increases over time. The LinkOFF threshold is found by searching from the right-hand side of the histogram; i.e. from the longest idle intervals, until the correct total number of messages has been found. That is, if the histogram has been collected for total time X, then the previous section gives the number of messages to delay as $N = lX \ / \ T_w$, where l is the local performance bound. The threshold is given by the midpoint of the smallest bin that has a total of at most N messages in all bins to its right.

The amount of work per message is constant and rather small, since it is only necessary to update the histogram and search for the correct value of LinkOFF. In our experiments, the LinkOFF threshold value is updated after every idle link event, but clearly it can be updated less frequently if desired. Alternatively, the algorithm can easily be optimized to take advantage of the fact that the correct value of LinkOFF seldom moves by more than one bin at a time.

Figure 7 shows three important characteristics of the algorithm. The x-axis is time, or more accurately a sequence number for the idle link event, and the y-axis is the value of the LinkOFF threshold, measured for a particular, but arbitrary, edge link (other edge links had similar behavior). Firstly, the correct value of the LinkOFF threshold differs dramatically between benchmarks—notice the logarithmic scale on the y-axis. Secondly, most applications rapidly converge to a stable value of the LinkOFF threshold, within just 200 events. This stable value can be compared with the point in Figure 3 where the overhead drops below 1%. Thirdly, for some benchmarks, most clearly LU, LinkOFF threshold is seen to adapt to varying application phases.

Although we discuss our mechanisms per *unit-time*, structures are refreshed in *idle link events*, e.g. every 20,000 idle link events, irrespective of the elapsed time or application.

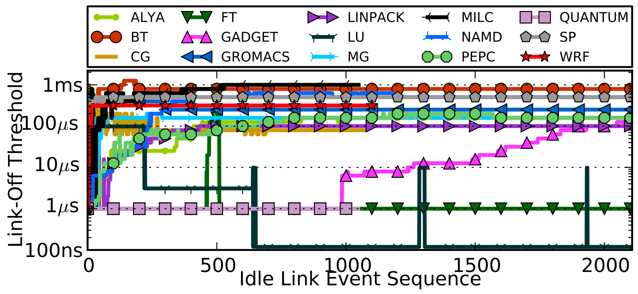

Figure 7: **LinkOFF threshold convergence over time**

In Figure 7, we show that, for all applications, the algorithm converges within 200 events, which is only 1% of 20,000, hence a negligible fraction. When a new application begins, only the first refresh cycle has events from the old application. In the worst case, at 4μs and 6 hops/message incurred on all 20,000 events in the first refresh cycle, the overhead is ≤480ms, which is negligible compared with typical application execution times.

3.5 PerfBoundPredict: Prediction over PerfBound for On/Off networks

It is clear, both from the above analysis and from the traces in Figure 1, that HPC applications exhibit repetitive behavior. This repetitive behavior translates to periodic and predictable idle link events. We use this insight to propose an idle period predictor that detects repetitive idle link events in order to turn off the link immediately as opposed to waiting for the LinkOFF threshold to expire.

This section describes **PerfBoundPredict**, an idle period predictor, whose performance overhead is controlled by PerfBound. Whenever the length of the upcoming idle period cannot be predicted with high confidence, the algorithm defaults to PerfBound. In addition, whenever the predictor mis-predicts, the performance overhead of one additional wakeup delay is compensated for: either by throttling prediction or by adjusting the LinkOFF threshold.

We borrow from ATPT [25], by predicting based on source-destination pairs. ATPT predicts the total amount of data transferred, whereas PerfBoundPredict is concerned with idle link durations. One challenge in predicting the lengths of idle periods is that there is always some noise; i.e., no two idle link events have *exactly* the same duration. This is handled by effectively quantizing the idle link events; that is, more accurately, by considering two idle link events to be the same if they differ by less than ±20%. We propose this tolerance based on experiments that showed a steep reduction in the number of unique idle link events up to ±20%. This means that the wakeup time must be up to 20% before the predicted event, since that prediction could correspond to a value as small as that. Finally, we ignore all events that are smaller than twice the time required for link wake-up and sleep, since there are many such events but they do not provide significant benefits for energy savings.

Figure 8 shows how classifying idle link events by the src-dest pair for the preceding message helps to identify repetitive behavior. Figure 8(a) plots the event duration on the y-axis for all idle link events, arranged along the x-axis. The data is for a fixed but arbitrary edge link, for application BT; other edge links have similar behavior. Two things are apparent in Figure 8(a). First, large events occur periodically, but smaller events are more sporadic. Figure 8(b) shows only those idle link events that follow a message on a spe-

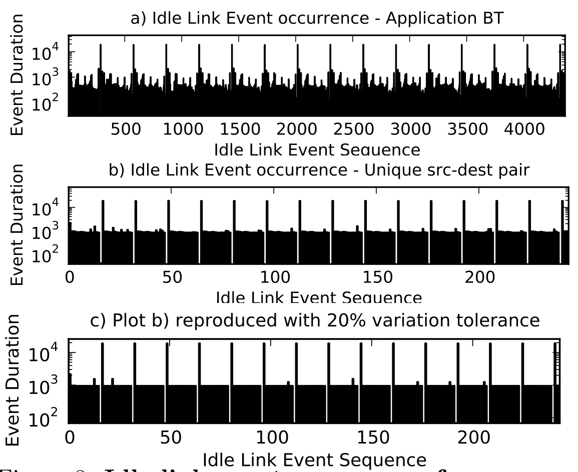

Figure 8: **Idle link events sequence of occurrence**

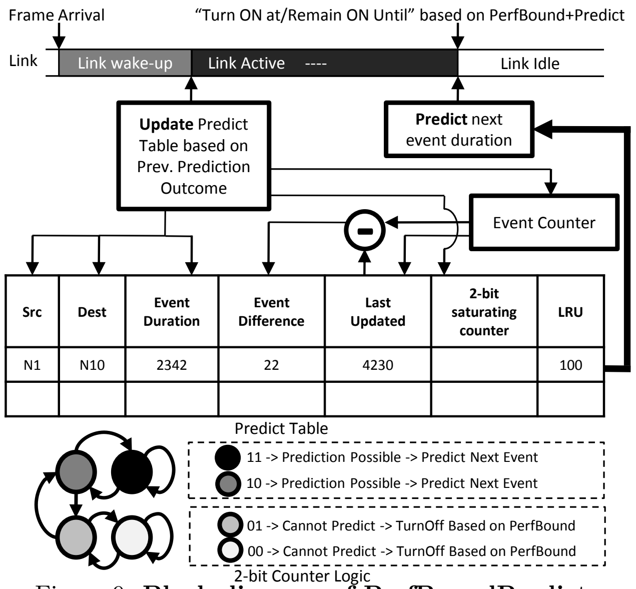
Frame Arrival
"Turn ON at/Remain ON Until" based on PerfBound+Predict
Link
Link wake-up
Link Active ----
Link Idle
Update Predict Table based on Prev. Prediction Outcome
Predict next event duration
Event Counter
Src
Dest
Event Duration
Event Difference
Last Updated
2-bit saturating counter
LRU
N1
N10
2342
22
4230
100
Predict Table
11 -> Prediction Possible -> Predict Next Event
10 -> Prediction Possible -> Predict Next Event
01 -> Cannot Predict -> TurnOff Based on PerfBound
00 -> Cannot Predict -> TurnOff Based on PerfBound
2-bit Counter Logic

Figure 9: **Block diagram of PerfBoundPredict**

cific src-dest pair. Note that not all src-dest pairs attach to large idle events. Most of the other src-dest pairs we examined had no large idle events at all. In fact, we chose the src-dest pair that contains all of the large events visible in Figure 8(a). In Figure 8(b), all large events are separated from the preceding one by exactly the same number of idle link events. In Figure 8(c), we account for variation in the idle event durations by applying ± 20% variation tolerance.

Figure 9, shows a block diagram of the proposed prediction mechanism. The predictor has two functions, **update** state and **predict** state. As shown in the figure, update state is invoked whenever a link is woken up. During update state, all fields in the predict table are updated with the previous idle link event. Predict state is invoked whenever the link becomes idle. The predict table is accessed, based on the recent src-dest, to make an idle time prediction. When prediction is not possible, algorithm defaults to remain on until LinkOFF threshold.

The predict table contains many entries, each of which contains the following: src-dest, the previous idle link *Event Duration*, a 2-bit counter for prediction confidence and a *Least Recently Used* (LRU) value. It also contains the *Event Difference* and *Last Updated* values. An *Event Counter* tracks the total number of idle link events that has progressed. The *Last Updated* value is updated, as described below, to contain a previous value of the *Event Counter*. The *Event Difference* is the period between successive similar idle link events.

The update stage after the link is woken up, updates predict table with the duration of the previous idle link event and the src-dest addresses of the previous message. The predict table is indexed using the src-dest and idle link event duration, to find any already existing entries. If such a src-dest exists, and its event duration falls within ± 20% of the original entry, then an *Event Difference* is calculated as the difference between the current *Event Counter* value and the *Last Updated* value in the entry. If this *Event Difference* matches the entry in the predict table, then a repetitive pattern is found, and the 2-bit counter is incremented. If it does not match, then the 2-bit counter is decremented. In either case, the *Last Updated* field is updated to equal the current *Event Counter*. If, on decrementing, the 2-bit counter reaches zero, then the new *Event Difference* replaces the old one. In any case, the LRU is refreshed, moving the entry to the top of the table. Finally, if there is no matching event, a new one is added, overwriting the entry with the oldest LRU value.

Prediction is done, whenever the link becomes idle based on the src-dest pair of the recent message. First, the src-dest pair is used to obtain a list of all matching entries in the predict table. Each entry in the predict table is checked in turn, starting from the most recent, by first calculating the *Current Event Difference* as the difference between *Event counter+1*, which corresponds to the event counter after this idle period, and the *Last Updated Value* in the entry. If the *Current Event Difference* matches the *Event Difference* in the table, then the entry is a tentative match. In this case, the 2-bit counter is checked for confidence. If it indicates a reliable prediction, then the link is immediately turned off, and scheduled to turn back on after a time given by the *Event Duration* minus 20%. If no tentative match is found whose 2-bit counter indicates a reliable prediction, then the algorithm defaults to PerfBound or PerfBoundRatio, by remaining on for a time given by LinkOFF threshold.

Overheads: Note that the techniques discussed have low overheads and are not time critical since, decisions regarding link shutdown or LinkOFF threshold are only computed after message transmission while the link remains 'on' but idle. Secondly, we expect our heuristics to require compute time in ns while links are always left on for at least the on/off switching time (7μs) after every message transmission.

4. RESULTS

Figure 10 shows the normalized application execution time, for each application referenced in the previous section, relative to the same system with an always-on interconnect. Figure 11 is similar, but for *link energy* savings, separately for each level of the network, where level 0 is connected to the nodes and level 2 is the highest. The energy saving mechanisms are identified as follows: Toff turns off each link as soon as it becomes idle. T50us is an arbitrary static LinkOFF threshold which has the link on for 50us before turning off. Since worst-case static LinkOFF threshold and PerfBound are described only for a single hop, results for the 3-level hierarchical network are given for three variants - allocating the 1% performance overhead equally among two, four or six hops. For example Ton4hop is static LinkOFF threshold tolerating a 0.25% overhead per hop; since the

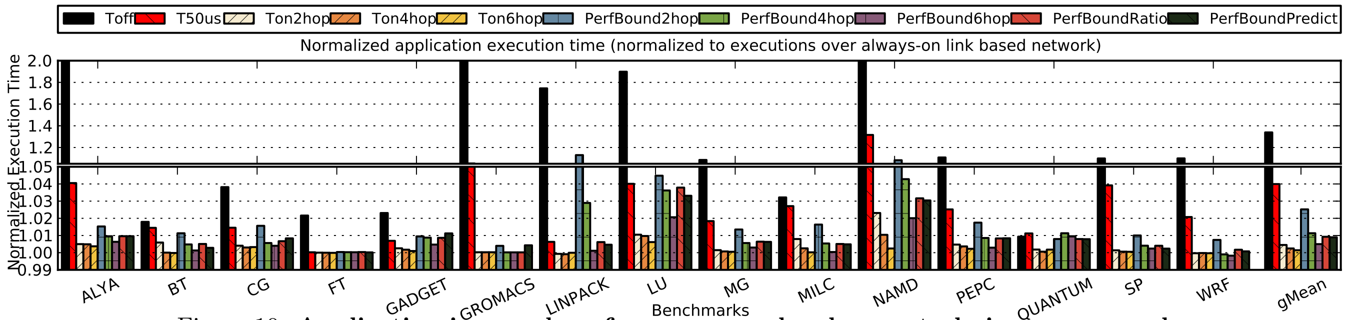

Figure 10: **Application incurred performance overheads over techniques proposed**

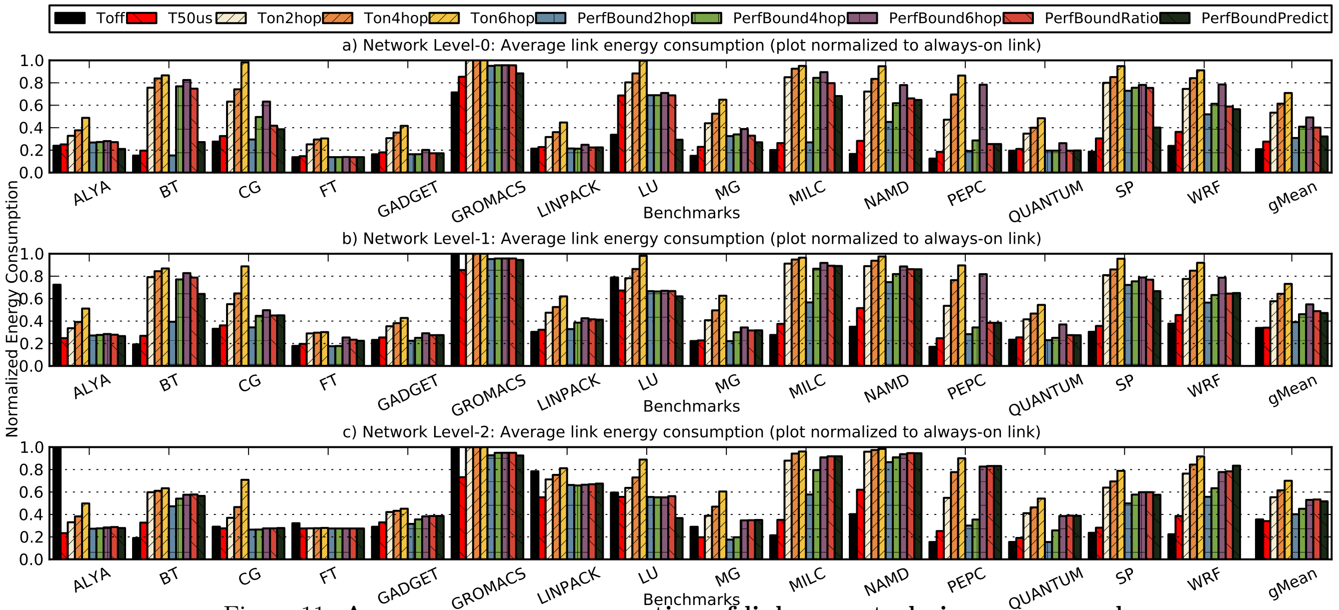

Figure 11: **Average energy consumption of links over techniques proposed**

wake-up latency is $T_w = 4\mu s$, this bound is enforced whenever the LinkOFF threshold is at least $T_w/0.0025$. In consequence, Ton2hop has greater energy savings, but, since it allocates 0.5% potential overhead to each hop, total overhead may reach 3%, while Ton6hop is conservative. PerfBound(2,4,6)hop are similar, but they use PerfBound instead of static LinkOFF threshold. Finally, PerfBoundRatio and PerfBoundPredict are the proposed algorithms. Since they naturally support hierarchical networks, the typical number of hops does not need estimating. The results include the full execution time, including warm-up periods for training the predictor and PerfBound mechanisms.

On average, as shown in Figure 10, proposed mechanisms PerfBoundRatio and PerfBoundPredict, both remain within assigned performance degradation bound of 1%. In comparison to Toff, our mechanisms (on average) reduce possible performance degradation from about 40% to assigned 1%. PerfBound2hop, expectedly exceeds assigned PerfBound to about 2.5% while PerfBound6hop is well within 1% at about 0.5%. As mentioned above it is clear from the results that PerfBound2hop and PerfBound4hop are too optimistic while PerfBound6hop is too conservative and PerfBoundRatio performs better at maintaining the assigned PerfBound. Static LinkOFF threshold values Ton(2,4,6)hop and T50us have performance degradation of 0.5% and 4% respectively.

With respect to energy, as shown in Figure 11, on average, Toff and T50us gives the highest link energy savings, followed by PerfBoundPredict and PerfBoundRatio. Note that the difference in energy savings between our proposed PerfBoundRatio or PerfBoundPredict and the naive Toff technique is less than 20%, while on average PerfBound reduces performance degradation by about 40%. On average, PerfBoundPredict produces 8.5% higher energy savings compared to PerfBoundRatio and overall, PerfBoundPredict saves link energy by 68.5% compared to an always-on network followed by PerfBoundRatio which saves 60% in network Level-1. Similarly, at higher levels 2 and 3, PerfBoundPredict saves 55% and 49% and PerfBoundRatio saves 51% and 48% respectively. Note that higher levels of the network tend to have higher traffic reducing possible opportunity for energy savings. Prediction technique works best in the lower levels (which contain the highest proportion of links in the network) and less well at the higher levels which are subject to more noise. Note that however, decreased prediction has not contributed to higher performance degradation. PerfBound(2,4,6)hop have lower/higher energy savings respective their performance degradation.

Four of the bars in the figures are for static values of the LinkOFF threshold. The 50us static LinkOFF threshold (T50us) achieves good link energy savings, but the worst-case performance degradation of 30% is unacceptable. In comparison, the worst-case overhead for PerfBoundRatio is 4% and for PerfBoundPredict 3.5%. On the other hand, Ton(2,4,6)hop have <=1% overhead, but their energy consumption is more than twice that of PerfBoundRatio and PerfBoundPredict, at 77% rather than 37%. We also performed a sweep to find the best static LinkOFF threshold. We found that a value of 500us or larger is needed to reduce the worst-case overhead, for our benchmarks, to 4%. At this point the average energy consumption increases to 49% of the original, compared with 40% for PerfBoundRatio and 35% for PerfBoundPredict. Moreover, to be confident that the worst case overhead in production is reasonable, a prudent system designer should choose an even larger LinkOFF threshold, similar to the values used by Ton(2,4,6)hop. As previously described, this would lead to link energy consumption roughly double that of PerfBoundRatio/Predict.

In Figures 10 and 11 it is clear that some applications, more than others, benefit from predictability. Application LINPACK, for example, has no benefit from prediction. This lack of benefit in this case however has to do with how well PerfBoundRatio works for LINPACK rather than the prediction mechanism itself. PerfBoundPredict saves energy by switching off the link immediately without having to wait for LinkOFF threshold timer to expire. Hence consequently if LinkOFF timer is small, relative benefit from PerfBoundPredict mechanism is low. LINPACK, as explained in Section 3.3 and as seen in Figure 4a&b, contains few events that contribute to majority of the idle time while most events are small and fit into a rather small LinkOFF Threshold value.

Contrary to the above, BT appears to benefit by about 60% from prediction. Unlike LINPACK, BT contains a large number of events that are large and contribute significantly to idle time of the application (Figure 4). Since LinkOFF threshold for BT is large, turning off the link immediately results in larger power savings. Interestingly, in Figure 11, we find that PerfBound2hop performs better than PerfBoundPredict. The reason for this can be seen in Figure 10, since, unlike other mechanisms, PerfBound2hop exceeds the PerfBound value of 1% by a small amount. This small amount is essentially the difference between a LinkOFF threshold larger than or smaller than that of the large cluster of events observed in Figure 4c. When LinkOFF threshold is larger, as in PerfBoundRatio, 1% PerfBound is maintained, however lesser energy is saved, when smaller, 1% PerfBound is not maintained, as in PerfBound2hop, however higher link energy is saved. Similar behavior can be observed at a smaller scale in applications CG and MILC.

The two outliers whose overhead are not bounded are NAMD and LU, due to dependencies in their messages i.e., messages in these applications are not transmitted until the reception of dependent messages. Further, as shown in Figure 1c, NAMD does not have any pattern to exploit for energy savings. Note that in both cases, performance overhead is still less than 4% with link energy savings up to 70%.

5. CONCLUSIONS

Interconnect inefficiency is a growing problem in HPC. While HPC applications have potential for link energy savings, techniques can only be employed if performance degradation is controlled. In this paper, we presented three techniques towards the above in the context of on/off links - PerfBound, PerfBoundRatio and PerfBoundPredict. We showed that significant energy savings can be obtained while performance overhead is bounded. Our techniques do not require modifications to the application/compilers nor does it introduce extra traffic into the network. The key novelty of our work is the analysis of link energy from a *performance perspective* - linking application performance degradation with link energy savings. Furthermore, we presented detailed analysis and insights on HPC application behavior with respect to link idle periods. We believe that our techniques and analysis could be useful in the upcoming standardization of Energy Efficient Ethernet for 40/100Gb links.

6. ACKNOWLEDGMENTS

This research was supported by the Ministry of Economy and Competitiveness of Spain under the contract TIN2012-34557, HiPEAC-3 Network of Excellence (ICT-287759), European Union's $7th$ Framework Programme [FP7/2007-2013] under project Mont-Blanc (288777), Generalitat de Catalunya (FI-AGAUR 2012 FI B 00644) and finally Severo Ochoa Program (SEV-2011-00067) of the Spanish Government.

References

[1] DARPA. Ubiquitous High Performance Computing (UHPC) Broad Agency Announcement (BAA). 2010.
[2] D. Teresa et al. High performance linpack benchmark: a fault tolerant implementation without checkpointing. In *International Supercomputing Conference (SC)*, 2011.
[3] NAS Parallel Benchmarks.
[4] R. K. Brunner et al. Scalable Molecular Dynamics for Large Biomolecular Systems. In *Supercomputing Conference*, 2000.
[5] D Abts et al. Energy Proportional Datacenter Networks. In *International Symposium on Computer Architecture*, 2010.
[6] Ripduman Sohan et al. Characterizing 10 Gbps network interface energy consumption. In *LCN*, 2010.
[7] Reviriego P. et al. Performance evaluation of energy efficient ethernet. *Comm. Letters.*, 13(9):697–699, September 2009.
[8] Christensen Ken et al. IEEE 802.3az: the road to energy efficient ethernet. *Comm. Mag.*, 48(11):50–56, nov 2010.
[9] S Conner et al. Link shutdown opportunities during collective communications in 3-D torus nets. In *IPDPS 2007*.
[10] K P Saravanan et al. Power/Performance Evaluation of Energy Efficient Ethernet (EEE) for High Performance Computing. In *IEEE ISPASS*, 2013.
[11] Li Jian et al. Power shifting in Thrifty Interconnection Network. In *High Performance Computer Architecture*, 2011.
[12] Li Shang et al. Dynamic voltage scaling with links for power optimization of interconnection networks. *HPCA*, 2003.
[13] V Soteriou et al. Dynamic power management for power optimization of interconnection networks using on/off links. In *High Performance Interconnects*, pages 15–20. IEEE, 2003.
[14] Eun Jung Kim et al. Energy optimization techniques in cluster interconnects. In *ISLPED*, 2003.
[15] Active/Idle Toggling with Low Power Idle, IEEE 802.3az.
[16] Vassos Soteriou et al. Software-directed power-aware interconnection networks. *TACO*, 2007.
[17] Maruti Gupta et al. Dynamic ethernet link shutdown for energy conservation on ethernet links. In *ICC'07*.
[18] Soteriou et al. Design-space exploration of power-aware on/off interconnection networks. In *ICCD*, 2004.
[19] Alonso Marina et al. Dynamic power saving in fat-tree interconnection networks using on/off links. IPDPS, 2006.
[20] Chamara Gunaratne et al. Ethernet adaptive link rate (alr): Analysis of a buffer threshold policy. In *GLOBECOM*, 2006.
[21] Baoke Zhang et al. Real-time performance analysis of Adaptive Link Rate. In *Local Computer Networks (LCN)*, 2008.
[22] Blanquicet Francisco et al. An Initial Performance Evaluation of Rapid PHY Selection (RPS) for Energy Efficient Ethernet. Local Computer Networks (LCN), 2007.
[23] Koibuchi M. et al. An on/off link activation method for low-power ethernet in PC clusters. *IPDPS*, 2009.
[24] Totoni et al. Toward Runtime Power Management of Exascale Networks by On/Off Control of Links. In *IPDPS-W'13*.
[25] YS-C Huang et al. Application-driven end-to-end traffic predictions for low power NoC design. In *VLSI Systems*, 2013.
[26] Rosa M. Badia et al. Dimemas: Predicting MPI applications behaviour in Grid environments. GGF8 Workshop, 2003.
[27] Sergi Girona et al. Validation of dimemas communication model for mpi collective operations. In *EuroPVM/MPI'00*.
[28] J. Gonzalez et al. Simulating whole supercomputer applications. In *IEEE MICRO*, 2011.
[29] Alya Red:Computational Biomechanics for Supercomputers.
[30] QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials.
[31] Michalakes et al. The Weather Reseach and Forecast Model: Software Architecture and Performance. In *ECMWF*, 2004.
[32] MIMD lattice computation collaboration.
[33] Hess et al. GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of chemical theory and computation*, 2008.
[34] The cosmological simulation code gadget-2.
[35] PEPC: Pretty Efficient Parallel Coulomb-solve, Interner Bericht Zentralinstitut fur Angewandte Mathematik.