# Performance-aware Energy Optimizations in Networks for HPC

by

KARTHIKEYAN P. SARAVANAN

A thesis submitted in fulfillment of the requirements for the degree of

*Doctor of Philosophy*
in
Computer Architecture

Departament d'Arquitectura de Computadors (DAC)
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain


PhD Supervisors: DR. ALEX RAMIREZ & DR. PAUL M. CARPENTER

2016

*This page has been intentionally left blank.*

# Performance-aware Energy Optimizations in Networks for HPC

KARTHIKEYAN P. SARAVANAN
Universitat Politècnica de Catalunya (UPC)

**(ABSTRACT)**

Energy efficiency is an important challenge in the field of High Performance Computing (HPC). High energy requirements not only limit the potential to realize next-generation machines but are also an increasing part of the total cost of ownership of an HPC system. While at large HPC systems are becoming 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 thesis explores interconnect energy proportionality from a performance perspective. First a characterization of HPC applications makes a case for the enormous potential for interconnect energy proportionality with HPC applications. Next, an HPC interconnect with on/off based links, modeled after the IEEE 802.3az Energy Efficient Ethernet protocol, is evaluated. This evaluation, while presenting a relationship between performance impact and energy over HPC applications, also emphasizes the need for performance focused designs in energy efficient interconnects. Next, an adaptive mechanism, *PerfBound*, is presented that saves link energy subject to a bound on application performance overheads. Finally this evaluation structure is applied into an intermediate link power state - *Fast-Wake*, in addition to the traditional on and off states. Results of this study, over 15 production HPC applications show that, compared to current day always-on HPC interconnects, link energy can be reduced by unto 70% while application performance overhead is bounded to only 1%.

*This page has been intentionally left blank.*

*Dedicated to*
*Amma, Appa, Tangi, Thambi and my Sai*

*This page has been intentionally left blank.*

# Acknowledgments

I start by thanking my advisors Dr. Alex Ramirez and Dr. Paul M. Carpenter for their support and guidance. I consider myself lucky to have had advisors whom, as I finish my PhD still make me wonder how much more I could have learned from them. I will always be grateful to Alex for instilling in me a methodology for research and critical thinking which has driven this thesis and will continue to guide my career forward. Alex is an inspiring and visionary supervisor, being under him has taught me to be ambitious and work on hard problems. He has always provided my PhD with a sense of purpose and why it matters.

I cannot thank Paul enough for the last five years of weekly meetings and guidance which has morphed into every word in this thesis. His guidance, generosity with his time, willingness to engage and patience during our discussions when it sometimes takes me literally minutes to get to his level of thinking, is behind all ideas in this thesis. Working on my PhD has been often times hard, with many paper rejects and many more ideas that didn't work, sometimes with months of investigation amounting to nothing, it's Paul's encouragement and support that has been vital in helping me persist and push through.

I want to thank Barcelona Supercomputing Center (BSC) and its director Prof. Mateo Valero for giving me an opportunity to be part of such a prestigious institution. During a time of economic recession in Spain, the mental freedom and financial security BSC provided me cannot be understated. I thank BSC for providing an environment filled with brilliant and accomplished professors and peers, especially with regards to my research group HetroArch. I will always fondly remember my time with the members of HetroArch, especially, Rico, Carlos, Rajo, Puzo, Renan, Dani, Milan, Thomas, Petar, Victor and Ugi, with whom I've had both intellectually challenging conversations as well as IQ-dulling parties. I also thank BSC for all of its resources from support teams to HR to its cutting-edge facilities and access to one of the world's fastest supercomputers.

I thank the Department of Computer Architecture (DAC) at the Universitat Politecnica de Catalunya (UPC) and its administration for their support in all my PhD requirements. I thank my funding institutions Universitat Politecnica de Catalunya (UPC) and the Generalitat de Catalunya for providing me with the FPI-UPC and the FI-AGAUR grants for PhD research. I thank the institution HiPEAC for their various workshops and internship opportunities, which lead to an internship and my first job at Samsung Electronics, UK.

I thank my Guru, undergraduate research advisor, Prof. Waran, whom I will always

be indebted to for my interest in research that drove me into taking up a PhD.

My time in Barcelona could not have been half as fun or memorable without my friends. I thank in no particular order, Shrikanth, Vinoth, Ramnath, Hasini, Aswin, Rajagopal, Dharshan, Sundaramoorthy, Sudhanshu, Manish, Rahul, Gaurang and Martha for memories I will always cherish. I can in specific never forget my first years living in Avinguda de Madrid with Shrikath and Aswin; and living in Rossend Arus with Vinoth Sudhanshu, Hasini and Marta; and with Raja, Dharshan and Sundhar visiting. Going back home after a hard day's work or the disappointment due to the lack thereof, has been so much easier due to these guys. I want to give special thanks to my friend Javed back home for his support and especially for enduring through my rants and complaining when I didn't get the results I wanted from my research.

I want to thank Sai, my fiance, for she's been incredibly supportive especially when I've been down and the going got tough. She's often times the first to celebrate my success and the first to cheer me up when I'm down. I want to give special thanks to my cousin Yogesh and my sister-in-law Lans, for I've never made a major decision or celebrated a success without telling them or turning to their guidance. I want to thank Suresh mamma, Jayanthi athai and Ashwin for their constant encouragement throughout my PhD.

I finally want to thank my family, for without their support none of this would have been possible (and I would have found myself a real job sooner). I want to thank my wonderful brother and sister, Kathir and Krithigha, for their love and support, for making it so much fun to visit India after a year's work. I thank my father and mother, Mr. P.M. Saravanan and Mrs. P.S. Rupavathy, who are my pillars of support. They believed in me when I didn't believe in myself. My father's emphasis on facing challenges head-on and bravely gave me the confidence to pursue and persevere during my PhD. I cannot express the encouragement and confidence my mother can impart after just a phone call with her. For their selfless sacrifice, love and encouragement throughout my education, I dedicate this thesis.

# Table of Contents

# List of Figures

# List of Tables

*This page has been intentionally left blank.*

# Chapter 1
# Introduction

Theory and experimentation are often known as the pillars of the scientific method. Advancements pushing the limits of the traditional experimental methods have lead to computational sciences now constituting a *third pillar* to scientific inquiry. Many fields including physics, chemistry, biology, astronomy, climatology, economics, social science and engineering rely on computational sciences to explore and test uncharted possibilities where traditional experimentation is either not possible or too expensive in cost or time. The success of computational sciences is driven by High Performance Computers (HPC) or Supercomputers. Exponential improvement in processor technology has in turn made it possible to exponentially improve the computing throughput of supercomputers for decades. This improvement translates to simulating ever larger models and problems to solve large questions in science.

Since the 1990s, HPC performance has increased at an exponential rate, doubling roughly every 1.2 years. The fastest machine in the November 2015 TOP500 list [3], Tianhe-2 achieves 33 PF (Peta-FLOPS) on the TOP500's HPL benchmark, which is 33 quadrillion floating point operations per second. The next milestone for HPC performance is 1 EF (Exa-FLOP which is 1000 PF), which is expected to be achieved around the year 2018 [9, 11, 12]. Several challenges lie ahead in reaching Exa-FLOP performances, however, one of the key challenges is energy efficiency and proportionality[1].

This thesis makes a case for a *performance-aware* approach to energy proportionality in HPC interconnects. Research focusing energy proportionality has made several strides over the compute components of the HPC system, making interconnect energy prominent. Interconnects are especially far from energy proportionality because their links (which constitute a significant proportion of the overall network energy) are always-on consuming energy regardless of their usage. Although several proposals have been made for energy proportionality in interconnects, they have not been adopted in HPC due to their possible unknown performance overheads. While energy

[1]Energy proportional systems consume energy proportional to their usage

efficiency is a key challenge, performance is still the primary design objective in HPC and hence proposals are unlikely to be accepted unless the performance overheads are limited and well understood. This thesis explores HPC applications and interconnect energy savings from a performance perspective, building and analyzing techniques not only for energy saving but also for containing limiting performance overheads within acceptable bounds.

## 1.1 Grand Challenge Problems and Impact

HPC based research has a profound impact on science and technology advancements. Grand challenge problems are fundamental problems in science or engineering with broad applications, with solutions aimed to be obtained by HPC. [21] The following are few examples of Grand Challenge Problems and their potential Impact. However, besides the examples below, HPC also has wide applications in engineering fields in the research of more efficient manufacturing, automobile, aerospace, transportation management, in energy with nuclear research, efficient wind turbines and oil discovery, in Astronomy, in material sciences, etc. Considering the above, as discussed in the following sections and chapters, the benchmarks and applications used in this research are actual production applications from the fields of biomechanics, quantum chemistry, weather modeling, molecular dynamics, particle interactions, N-body simulation, etc.

**Prediction of weather, climate global change**

Research over the decades has provided unequivocal evidence for the warming the global climate system. Compelling evidence for rapid climate change comes from several sources that include, records of sea level rise, global temperature rise, warming oceans, shrinking ice sheets, etc. [20] From sea level rise threatening coastal regions to warming oceans and melting ice sheets changing entire ecosystems and food producing patterns, understanding and accurately predicting climate change is more important now than ever.

HPC is one of the foremost technologies in use towards the fight against global climate change. Supercomputers are widely used by multiple research centers around the world to run climate and weather models that predict global climates many decades

into the future to even predicting the odds of rain the next day. [20] These models extrapolate current and past patterns to give insights as to the impact of climate change in the future. Supercomputers are also widely used in number crunching (*Big Data*) and visualization, in obtaining meaning from data gathered by climate sensors and satellites. Modeling and simulating hurricanes and wind patterns are crucial to preparing for and averting natural disasters. Climate and weather modeling research while using the fastest supercomputers in the world, still require even faster machines for more accurate predictions and results [20]. Similar to the above, the aplication WRF - Weather Research and Forecasting Model, a numerical weather prediction system designed for both atmospheric research and operational forecasting needs is a part of the list of applications evaluated in this thesis.

**Human Brain Project, Molecular Dynamics and Biomechanics**

The Human Brain Project (HBP) [16] is a European Commission Future and Emerging Technologies flagship project that is primarily aimed at gaining a deep understanding of the workings of the Human Brain [2]. Understanding the Human Brain is critical to solving neurological disorders and brain diseases. HPB involves many sub-projects and research divisions, however primary objectives include modeling and simulating the brain. The project also includes research on HPC since the computing infrastructure required for simulating a human brain does not currently exist.

Simulating the brain over a computer provides for a non-invasive method of drug testing and discovery. While drug testing on animals is the norm, extensive drug testing on human brains involve legal and ethical complications. A computational model of the brain will speed up the process of drug testing and a better understanding of disease progression. Pattern in the data or biological signatures of diseases such as Alzheimer's will led to better classification, more accurate diagnoses and a more personalized medicine [16].

Several workloads evaluated in this thesis have applications in biology, specifically ALYA, GROMACS and NAMD. ALYA is a HPC based computational Biomechanics model for Supercomputers. Its designed to simulate and study the working of the human heart, Respiratory System, Skeletal Muscles among others. GROMACS stands for GROningen MAchine for Chemical Simulations and is a molecular dynamics package designed for simulating proteins, lipids, and nucleic acids. It is used to study protein

[2] A similar project at the United States is the BRAIN Initiative [19]

folding, which in itself has applications in understanding and solving diseases. NAMD is a scalable molecular dynamics program used to study chemical interactions and for biomolecular modeling.

**CERN and HPC in Physics**

Research in CERN with its Large Hadron Collider (LHC) run experiments that probe into the fundamental nature of the universe. the Large Hadron Collider is the largest and the most powerful particle collider in the world. It is also the largest experimental facility ever built and is the largest machine in the world. Built by CERN - European Organization for Nuclear Research, LHC allows physicists to test predictions and theories from particle and high energy physics. It was instrumental in the discovery of the Higgs boson particle, which explains the existence of mass in the universe. Along with experiments into understanding the nature of the Higgs boson, many other particle predictions and unsolved questions in physics are expected to be solved with the use of the LHC [17].

The data produced by particle collisions at LHC is enormous and unprecedented and hence has one of the world's largest high performance computing facilities in the world, to store, distribute and analyze points of interest. While CERN by itself does not have the computing resources to compute the produced data, the data is computed by *Worldwide LHC Computing Grid*, a distributed computing infrastructure with supercomputers from around the world.

The above example in relation to CERN and LHC provides for one example in the role of HPC in the field of Physics. However, several applications and modelled to simulate and test theories in physics and cosmology. This thesis evaluates applications GADGET and MILC which both have their applications in Physics. GADGET specifically is a N-body simulation package used to simulate dark matter and gas interactions. MILC is used to study Quantum ChromoDynamics (QCD), the theory of the strong interactions of subatomic physics.

Besides the above, several other applications are used in this work and are discussed in the following chapters. Reaching exa-flop performances is key to solving several key challenges and problems. The following is a brief introduction to the evolution of supercomputers, leading up the the problem of exa-scale and energy as a key challenge.

## 1.2 Supercomputers - Past, Present and Future

Grand challenge problems have perpetually inspired faster and higher throughput supercomputers. Faster machines usually enable better and more accurate results and a possibility to simulate larger problem sets.

Although supercomputing dates back to 1960's, they only used a few processors [15] until 1990's which marked the beginning of massively parallel computers with thousands of processors. Performance increased from about 120 GF in 1993 with Fujitsu's Numerical Wind Tunnel machine to about 2.3 TF with the Intel ASCI machines from DoE-Sandia National Labs. The 1990s also marked the shift from vector processors to the use of commodity microprocessors in supercomputers, due to their better price to performance ratio of the latter.

Exponential improvement in performance continued through the 2000s. The years between 2000 and 2010 brought about several advances in technology with 1000x performance increase reaching Tera-FLOP to Peta-FLOP performances. The previous generation counterparts and the introduction of several of current day flagship machines where introduced during this period. The NEC Earth Simulator was the first supercomputer to reach 35 TFLOPS in throughput. Earth Simulator consisted of 640 nodes with 8 vector processors and 16 gigabytes of memory in each node. It was the fastest supercomputer in the world from 2002 and was surpassed by the Blue Gene/L machine in 2004.

The IBM's Blue Gene/L launched its landmark Blue Gene series of machines towards the end of 2004. The Blue Gene series introduced machines that had unprecedented scaling powered by a large number of relatively low performance but highly energy efficient processors. The Blue Gene/L introduced several unique innovations including, trading performance per core for lower power, dual processors per node, System-on-a-chip design, highly scalable machine that scales up to 65 thousand processors enabled by a 3-D torus interconnect and a light weight operating system. With unprecedented scaling, Blue Gene/P ranked, first in the Top500 list of the worlds fastest supercomputers, from November 2004 with 70 TFLOPS until November 2007 reaching nearly half a PFLOPS performance[3].

Following the BlueGene/P, the IBM Roadrunner was released in 2008, the first supercomputer to reach the PETA-FLOPS performance milestone. The Roadrunner was the first machine to use accelerators along with conventional processors, calling it then, a "hybrid approach". Alongside the Roadrunner, Cray Jaguar in 2009 and the Chinese

Tianhe-1A from 2010 reached 1.7 PFLOPS and 2.5 PFLOPS respectively.

In 2011, the K-Supercomputer from Japan was the first to reach 10 PFLOPS. Between 2011 and 2015, three other supercomputers BlueGene/Q, Titan and Tianhe-2 surpassed the K-Supercomputer with performance higher then 10 PFLOPS, with 17.1, 17.5 33.8 PFLOPS respectively (as of November 2015). The years between 2010 to 2015 also saw the emergence of GPU accelerator based supercomputers. From only nine supercomputers in the Top500 list[3] in 2010, that were built with accelerators, GPU based accelerator adoption has seen a steady increase with 89 machines as of November 2015.

Apart from performance improvements, last 10 years have also seen an increased emphasis on energy and power efficiency. The Green500 [4] list was established in 2006 to list 500 of the most energy and power efficient supercomputers in the world measured in FLOPS/WATT. From BlueGene/P ranking first with 357 MFLOPS/W in the first Green500 list in November 2007, energy/power efficiency has come a long way with the top machines in November 2015 measuring in with up to 7000 MFLOPS/W in the list. One key factor in driving increases in power/energy efficiency can be attributed to the use of accelerators. For example all of the top 40 systems in November 2015 Green 500 list use accelerators namely NVIDIA GPUs, AMD FirePro GPUs, Intel Phi or PEZY-SC. Even following the top 40, the next 40-100 systems are still primarily a mix of the above mentioned accelerators based systems or the BlueGene/Q.

The increase in emphasis in energy and power efficiency is critical to building future HPC. Past trends in exponential growth in HPC performance point to a 1 Exa-FLOP machine by the year 2018. Several challenges lie ahead of building exa-scale machines as seen in the subsequent sections.

## 1.3 The Problem of Energy

The Top500 trend-line for the fastest machine in the world, in Figure 1.1 [3] predicts Exa-FLOP HPC performance at about 2018-2020 time period. Several reports including DARPA [11] in 2008 and PRACE[18] in 2012 have called for addressing the several challenges and barriers that require crossing before reaching Exa-scale performance.

[3] Obtained from the Top500 statistics page [3]

Architecture, micro-architecture, energy efficiency, power consumption, memory bandwidth/capacity, system and interconnect scaling, I/O bandwidth, resilience and reliability, algorithm and application scaling to state a few, require several innovations to allow for Exa-scale performance [11, 18].

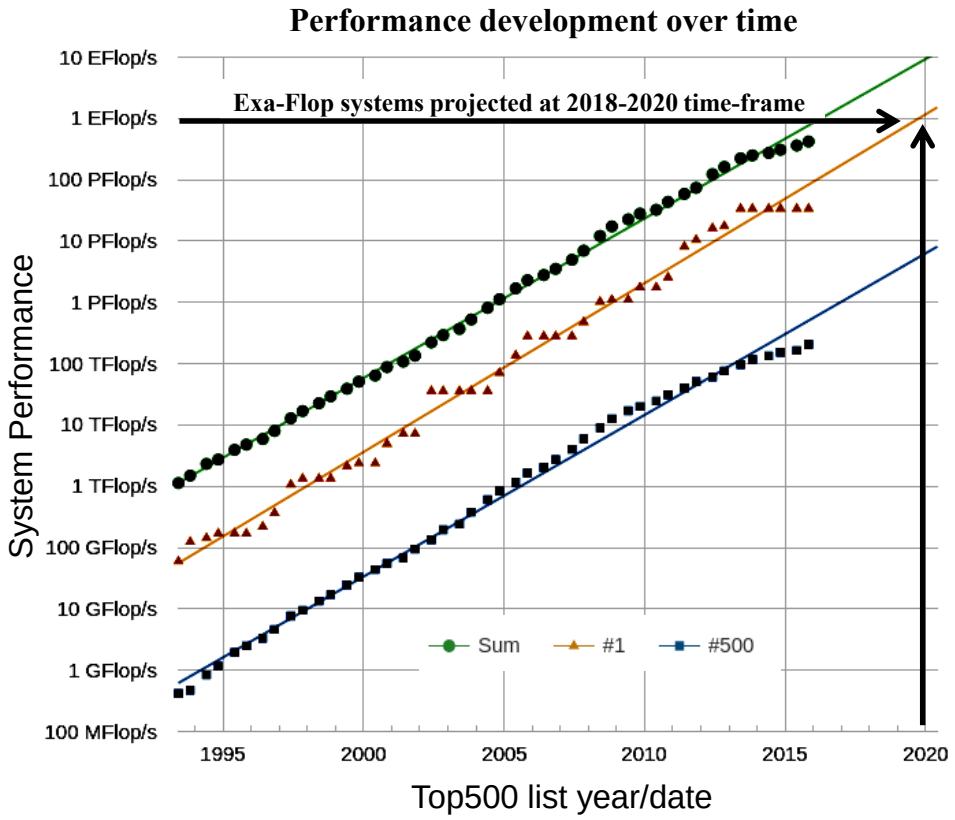



Figure 1.1: System share of Interconnects in TOP500 supercomputers.

Of the mentioned challenges energy/power efficiency, in particular, is one of the key design criterion for modern supercomputers. Today's fastest supercomputer (as of June 2015, at 33.8 PF/ 17.8 MW [3] gives 1.9 GF/W), if extrapolated to a sustained ExaFlop performance, would have a total power consumption of about 500 MWatts. In contrast, US DOE, DARPA and other exa-scale HPC programs target the deployment of an ExaFlop machine by 2018 with power consumption of 20 MWatts [11, 12]. Building future supercomputers at such stringent power budgets requires eliminating every inefficiency throughout the system. Much effort in bringing about energy efficiency and energy proportionality to systems, has gone into optimizing the compute elements and memory, which constitute the majority of the system power consumption. However, recent trends show an increased emphasis on interconnects.

**Energy Proportionality in HPC Interconnects**

With compute nodes being power optimized and energy proportional, the interconnection network power has become increasingly significant. Power consumption of HPC interconnects can contribute to up to 12% of the overall energy costs of the system [34, 48] and even more when the system is not fully utilized. As with supercomputers, reducing interconnect power has become a pressing issue for Internet and IT infrastructure as well as servers and high-end data centers. Studies suggest an annual power consumption of 6 TWh consumed by networking devices alone in the US, and this figure is expected to further increase [55]. Since Ethernet is the dominant interconnect technology in commercial and IT infrastructure, improvements in its energy efficiency are estimated to bring about energy savings of over 3 TWh [55].

Interconnection links can be attributed to consuming a substantial portion of the total interconnect power. Links take up 64% of the power budget of the IBM InfiniBand 8-port 12X switch [32, 71] and 63%, 65% of the Dell PowerConnect 5324 and 6248 respectively [71]. In addition, conventional network links are essentially always on, thereby dissipating power, regardless of whether or not data is being transmitted. The average power consumption of the IBM InfiniBand 12X link, for example, is almost identical to its worst-case power [54, 71]. In the case of Ethernet, its design require both the transmitters and receivers to operate continuously to keep the link aligned. In order to mitigate such waste, the IEEE 802.3az Energy Efficient Ethernet (EEE) task force was setup and an energy efficiency standard for Ethernet was approved in September 2010. This thesis in specific, discusses interconnect energy proportionality as modeled based on Energy Efficient Ethernet for reasons discussed below.

## 1.4 HPC Fabrics - Is Ethernet Relevant?

This research specifically focuses on evaluating the use of Energy Efficient Ethernet for energy proportionality in HPC interconnects, its challenges and their solutions. Since Energy Efficient Ethernet is based on Ethernet technology, the question of whether Ethernet is relevant to HPC is discussed below.

Over the period of this research, between 2011 and 2015, HPC has seen the use of numerous interconnect technologies, broadly falling into the following categories, Ethernet based, InfiniBand based, proprietary and custom interconnects. Both proprietary

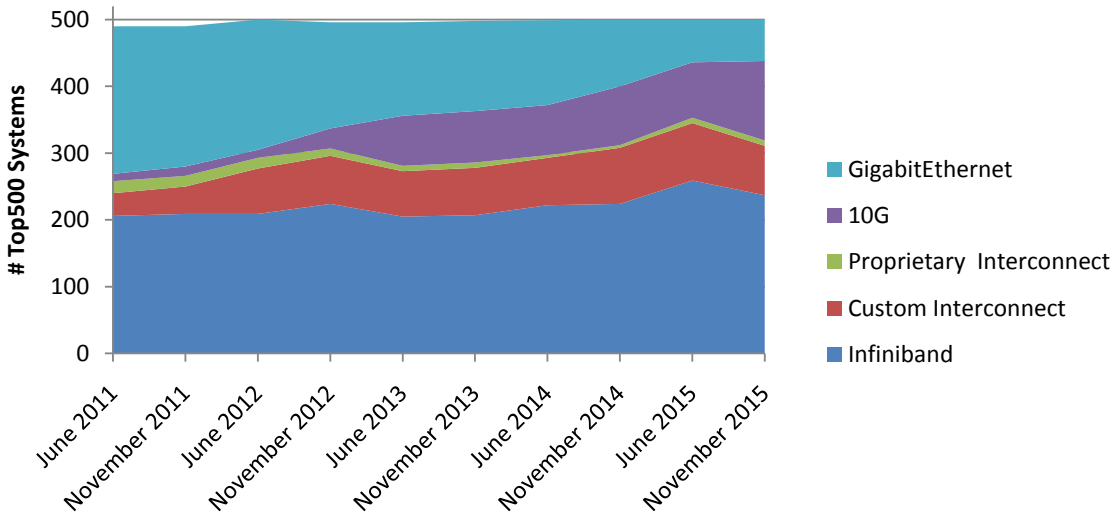



Figure 1.2: System share of Interconnects in TOP500 supercomputers stacked

and custom interconnects are as their name implies interconnects specifically built for their respective machines rather than off-the-shelf solutions such as Ethernet and InfiniBand. The K-Supercomputer from Japan's Tofu interconnect [38] is one such example of a custom as well as a proprietary interconnect. The Tofu is a 6-D mesh/torus built specifically for the K-Supercomputer. Its six dimensions interconnect, racks scaling the entire system at the lower dimension, boards at the middle and the processors on each system board at the highest dimensions of the network. Interconnects of the Cray systems - Aries and Gemini and the interconnects of Blue Gene series machines fall into this category.

Figures 1.2 and 1.3 lists the system share of interconnects in the Top500 systems list. The Figure 1.2 is stacked showing the evolution of the above mentioned categories of interconnects over the years. In June 2011 GigaBit Ethernet, 10G Ethernet, InfiniBand, custom and proprietary interconnects add up to 490 systems, while the other 10 systems contained networks such as Myrinet [6], NUMAlink [7] and Quadrics [8].

### InfiniBand

InfiniBand, over the years has held a large system share in the Top500 systems. InfiniBand is widely used in HPC for featuring very low latency, very high throughput high scalability and quality of service. Similar to Ethernet, InfiniBand is operated by a consortium with its standards organization called the InfiniBand Trade Association (IBTA)

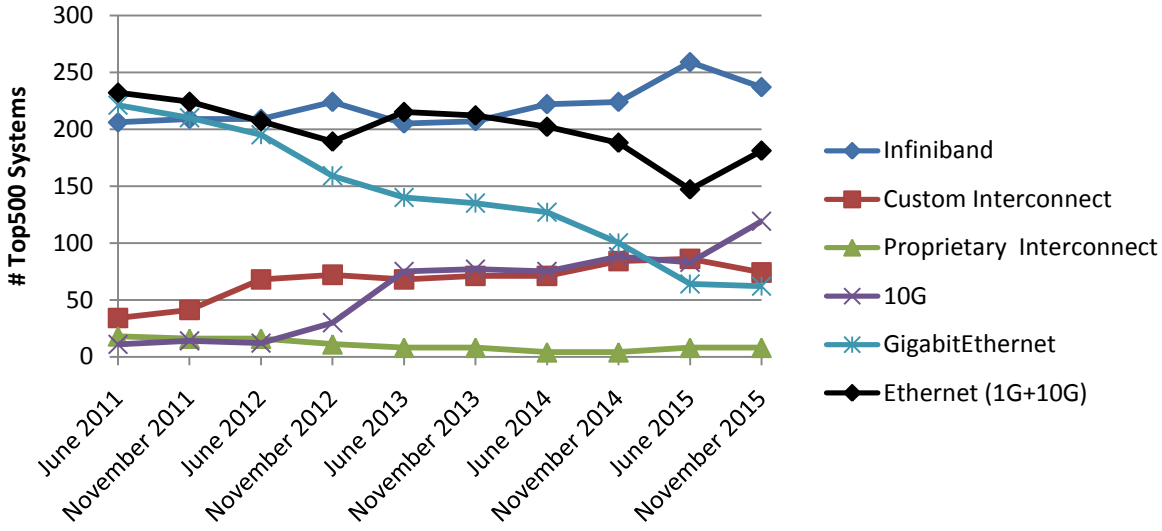



Figure 1.3: System share of Interconnects in TOP500 supercomputers - Ethernet based machines constitute about 40% every year

publishing its first InfiniBand architecture in the year 2000. Since 2000, InfiniBand has seen steady increase in bandwidth and decrease in latency. The signaling rate has increased from 2.5 Gb/s in 2001 to 5, 10, 14 and 25Gb/s with SDR (Single Data Rate), DDR (Double Data Rate), QDR (Quadruple Data Rate), FDR (Fourteen Data Rate), and EDR (Enhanced Data Rate) respectively. These signaling rates are the theoretical throughputs for a single link (1x) and can be paired with 4x or up to 12x links, known as **Link Aggregation**, reaching up to 290 Gb/s with 12xEDR. The latencies from SDR to EDR has decreased over the years from 5$\mu$s to 0.5$\mu$s respectively. The Top500 system share of InfiniBand has steadily increased from about 200 in June 2011 to about 250 systems in November 2015.

### Ethernet - Gigabit Ethernet and 10G

Ethernet compared to InfiniBand is an older and most widely used interconnect technology commonly used in local area networks (LANs) and metropolitan area networks (MANs). Dating back to first introduction as early as 1980s, Ethernet has become the standard for networking. Ethernet is a part of the IEEE 802 family of standards maintained by the IEEE 802 LAN/MAN Standards Committee (LMSC). Ethernet was introduced in HPC and featured in the Top500 machines list in the early 2000s with 10/100Mb/s and soon followed by the Gigabit Ethernet in 2002. Both Infiniband and

Ethernet saw increasing adoption in HPC markets over the years with the rise of commodity components based supercomputers. Ethernet specifically is popular for its wide spread use in LANs and MANs and its extremely cost- effective solutions, i.e., Ethernet based networking solutions are cheaper than the above mentioned Infiniband, proprietary and custom solutions. Traditionally Ethernet has lagged behind Infiniband and custom networks in latency and bandwidth, however, the gap between them has reduced over the years with the introduction of technologies such as iWarp [23] and RoCE (RDMA over Converged Ethernet) [24] and the recent standardization efforts targeting up to 400Gbit Ethernet. The most compelling reason for Ethernet's large adoption in HPC is its low cost and high performance.

Figure 1.3 presents system share of interconnects in the Top500 machines and as shown, Gigabit Ethernet has seen a steady decline over the years in its use. First featuring in the Top500 systems list in 2002, and by 2011, Gigabit Ethernet was used by more than 200 systems. As seen in the Figure 1.3, Gigabit Ethernet has seen a steady decline in its use between 2011 to 2015, however 10G Ethernet as shown has seen consistent rise in adoption. Combined, both Gigabit and 10G Ethernet (shown in black) is the most popular interconnect technology along side Infiniband. The share of Ethernet based machines over the years from 2011 to 2015 is shown in Figure 1.3. It is clear from the figure that a large proportion (about 40%) of HPC and Top500 machines are deployed with Ethernet.

## 1.5 Energy Efficiency in Ethernet (EEE)

As mentioned previously network links are traditionally always-on consuming power regardless of their usage. The ubiquitous nature of Ethernet with its wide spread use throughout the computing spectrum meant that optimization of the power consumption of Ethernet links translated directly to global energy savings from network infrastructure. The IEEE 802.3az Energy Efficient Ethernet (EEE) task force formed in 2007 aimed to make Ethernet links energy proportional. In considering several proposals, two approaches where most popular, Adaptive Link Rate (ALR) and Low Power Idle (LPI). The task force eventually adopted Low Power Idle (LPI) as the standard for Energy Efficient Ethernet (EEE). The Low Power Idle (LPI) converts the always-on link to an on/off based link where the link saves power at its low power off state. The standard published in 2010 provided mechanisms for implementing on/off based links to Ethernet switches.

The original Energy Efficient Ethernet (EEE) protocol, discussed thus far, provides specifications for energy savings on 100Mb, 1Gb and 10Gb links (Gigabit and 10G Ethernet). The success of EEE, current and recent efforts for the standardization of 40Gb, 100Gb and 400Gb backplanes and optical Ethernet, have opted to include and have included energy savings mechanisms from EEE. While incorporating EEE for 40Gb, 100Gb and 400Gb links, the standard further introduced alongside the older on/off based EEE, an additional intermediate sleep state. The intermediate sleep state is targeted towards a faster transition to fully active state in order to avoid excessive performance degradation due to state transitions.

These protocols, specifically IEEE 802.3bj and 802.3bm, providing standards for 40Gb and 100Gb backplanes and optical Ethernet, respectively, were ratified as recently as March 2015, and IEEE 802.3bs for 400Gb is expected to be ratified in 2017. As with the original EEE protocol, products based on the recent standards, may be deployed for HPC within a year, but it is likely that EEE will be disabled by default. With about 40% of Top500 HPC machines using Ethernet based interconnects every year, that could potentially have these protocols in their switches; it is imperative to understand the need for EEE, its performance impact and possible configuration parameters in the context of HPC applications.

### 1.5.1 Performance-aware Energy Proportionality

HPC applications require a high-performance interconnect to support their peak communications demand, but the average utilization of the network is low. Considering this, it is thus especially wasteful in HPC for its network links to remain always-on when its average the network utilization is low. A number of studies have proposed mechanisms to save interconnect link energy [13, 31, 48, 49, 63, 68]. These proposals fall into one of two main categories. Firstly, *on/off links* are powered down during idle periods. An important example is the above mentioned IEEE 802.3az Energy Efficient Ethernet (EEE) [2, 54]. 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 a variable number of active $1\times$ links.

In both cases, changing power state incurs a delay; for example, EEE on a 10Gbps link requires about $4\mu s$ to switch states. A similar delay in changing bandwidth has

been reported for Infiniband [48]. 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. In EEE specifically, the protocol does not specifically define the heuristics behind state transitions. This is an area of active research where different heuristics targeting different workloads (including data centers) have been proposed [64]. These state change heuristics 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.

## 1.6 Contributions and Organization of this Thesis

This thesis discusses energy proportionality in HPC networks from a performance perspective. There are several challenges as discussed above that this thesis addresses in this context. The following are the broad novel contributions of this thesis, specific contributions are further discussed in the following chapters.

1. Evaluation of the Energy Efficient Ethernet protocol on HPC is presented. This thesis was the first to evaluate the potential of Energy Efficient Ethernet for link energy savings in the context of HPC. The presented analysis concludes that HPC applications have high potential for network energy savings however the wake-up and sleep delay introduced by EEE can cause performance overheads unless properly controlled.

2. Any technique used for energy savings can only be used if their performance overheads are controlled. The second contribution of this thesis proposes the use of PerfBound - A technique that automatically manages on/off links such that performance overheads are limited, while savings link energy. An extension to PerfBound, PerfBoundRatio is presented that distributes an accepted the application overhead across all links of the network. This work also discusses a prediction method for saving link energy by predicting idle periods between messages.

3. Third contribution of this thesis evaluates and extension to the original Energy Efficient Ethernet protocol, Fast-Wake. Fast-Wake was added to EEE as a faster wake-up mechanism but that which also saves energy. This work shows that Fast-Wake on its own does not provide substantial benefits however, used along side the original on/off states proposed by EEE, can provide better energy savings.

4. The evaluation of Fast-Wake showed energy savings benefits in using an intermediate energy state. This benefit however cannot be realized unless the on/off states are properly managed. For this reason, the final contribution of this thesis presents DoublePerfBound that extends the previously proposed PerfBound to work for an intermediate state, alongside the original on/off states. DoublePerfBound automatically manages on/off links to obtain pareto-optimal energy savings to performance values.

The organization of the rest of this thesis is as follows. The next chapter presents background and literature, which discusses the necessary background behind Energy Efficient Ethernet and relevant related work. Chapter 3 presents the methodology used in the work. Chapters 4, 5, 6 and 7 discuss the four contributions of this thesis as presented above. Finally Chapter 8 concludes this work.

# Chapter 2
# Background and Literature

By the year 2010, the Internet and specifically data centers accounted for 1.1% to 1.5% of global energy consumption, with this percentage having doubled since 2005 [46]. The growing energy cost of Internet infrastructure and data centers have pushed for energy proportionality [1] across the computing spectrum. Addressing the above, in 2010, the IEEE 802.3az Energy Efficient Ethernet Task Force published its standard for Ethernet energy efficiency [2, 54]. The goal of the task force was to reduce the significant contribution of network devices to the national power budget, especially since large sections of the Internet and data center infrastructure are built using Ethernet [54]. While a number of proposals for energy efficiency were considered, Adaptive Link Rate and Low Power Idle [2, 54] were the most popular choices. In following sections, we briefly discuss Adaptive Link Rate and Low Power Idle.

## 2.1 Adaptive Link Rate

From data centers to HPC to Internet communications, Interconnects in general are sparsely used. However, the power consumption of links (which takes up the majority of the interconnect power) remains the same regardless of whether or not there is data transmission [2, 54]. This led to studies focusing on building energy proportional links whose energy consumption roughly proportional to link usage. Many initial proposals in energy proportional interconnects were concepts similar to Adaptive Link Rate (ALR) [40, 41, 42, 43]. The idea behind ALR comes from the fact that a link's power increases with bandwidth. Example, a 100 Mb/s transceiver consumes about $0.25$ W, whereas 1 Gb/s and 10 Gb/s transceivers consume about $0.7$ W and $6$ W respectively [35]. Furthermore, since these links typically consume the same power whether active and idle, it is beneficial to reduce the Ethernet link rate from 10 Gb/s link to 100 Mb/s link (say) during periods of inactivity.

[1] Energy proportionality signifies energy consumption proportional to usage.

Although ALR promised large energy savings by making links energy proportional, its process of changing link rate required time in the range of milliseconds to a few seconds, which is far too large for many applications. This is in agreement with results presented in Chapter 4 which evaluates HPC applications for their sensitivity to increases in latency. While other alternatives to accelerate rate change were proposed [44], ALR still contained two major disadvantages. Firstly, despite the proposals to accelerate rate change, the time taken for rate change was still higher than acceptable ranges. The second disadvantage is that, although lesser power, Adaptive Link Rate still consumes power during idle periods of the interconnect. To illustrate, if a 10Gb/s link is rate adjusted to 1Gb/s during periods of inactivity, the inactive 1Gb/s link consumes power, even though no data is being transmitted.

## 2.2 Energy Efficient Ethernet: Low Power Idle/ Deep-Sleep

To solve the above conundrum, an alternative known as Low Power Idle [2, 54] was proposed. Low Power Idle (LPI) essentially switches *off* the links during periods of inactivity and turns them back *on* when needed. The key point is that switching *on* and *off* links with LPI is relatively much faster (order of microseconds) and the link speed is not changed. This scheme was chosen to be the standard accepted to be a part of IEEE 802.3az Energy Efficient Ethernet standard.

The Low Power Idle (LPI) mode of EEE proposes the use of "Sleep" and "Wake" modes to conserve power during periods of inactivity. Unlike complex mechanisms required to change the link speed in the case of ALR, Low Power Idle freezes the states of the transceiver when it enters the low power mode and restores it when links are powered back up. This operation can be performed in a few microseconds compared to milliseconds required for ALR [2]. Figure 2.1 shows a state transition example of a link that uses Low Power Idle. Here, the $T_s$, $T_w$ and $T_r$ are the time taken to put the link to sleep, wake the link and refresh the link respectively. The periodic refresh in Figure 2.1 is to ensure the receiver elements are aligned with the channel during low power mode.

With regard to energy efficiency, link's power consumption during $T_s$, $T_w$ and $T_r$ consumes the same power as when a link is in its *on* state and $T_q$ consumes about 10% of the total power consumption of the link (here, $T_r << T_q$) [2, 55]. Table 2.1 shows

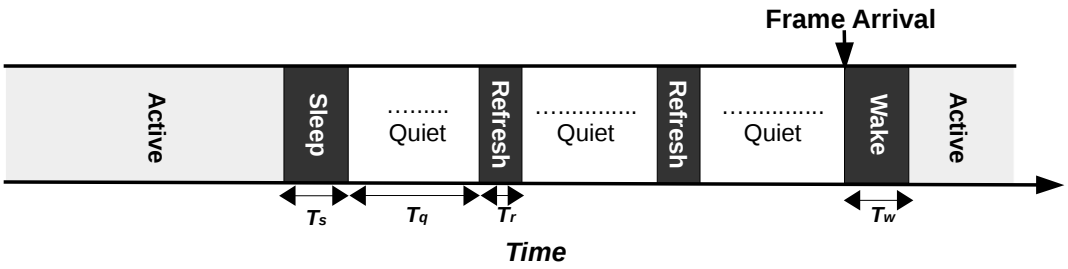



Figure 2.1: State transitions between active and low-power modes in Energy Efficient Ethernet (EEE): Low Power Idle (LPI)

| Protocol | $T_w$ | $T_s$ | $T_{Frame}$ (1500B) | Frame eff. |
|---|---|---|---|---|
| 100Base-Tx (100 Mb/s) | 30 $\mu s$ | 100 $\mu s$ | 120 $\mu s$ | 48 % |
| 1000Base-T (1 Gb/s) | 16 $\mu s$ | 182 $\mu s$ | 12 $\mu s$ | 5.7% |
| 10GBase-T (10 Gb/s) | 4 48 $\mu s$ | 2 88 $\mu s$ | 1 2 $\mu s$ | 14.6 % |

Table 2.1: Link parameters (wake, sleep, frame transmission time)

the values for $T_s$, $T_w$ and frame transmission efficiencies based on IEEE 802.3az draft [1]. It is to be noted from Table 2.1 that the sleep and wake times are relatively large for small frames. This is to show that, EEE would typically work best for large bursts of communication activity followed by large periods of inactivity.

P. Reviriego et al., published an evaluation of Energy Efficient Ethernet [55]. Their results, as shown in figure 2.2, suggested that power savings with EEE links decrease quickly with an increase in link utilization above zero. According to their results, when the link utilization is at 20%, the corresponding link power consumption is greater than 70%. The results obtained were for 1000-bit frames arriving into the link following a Poisson process. The poor performance is shown to be a result of large wake-up and sleep overheads compared to actual frame transmission. Essentially, a majority of the time is being spent switching *on* and *off* the links leading to decreased performance. To solve the above problem, frame buffering was proposed which holds frames (without waking up the link) up to a certain number of frames or a time-out period. With an increase in time-out periods and/or by increasing the number of frames held in buffers, the results can be made more energy proportional. However this method comes with the cost of increased packet delay leading to performance degradation of latency sensitive applications. Results show that a time-out period of $120\mu s$ [54, 55] is required for the link to be energy proportional with increased link utilization. In the later chapters, we analyze the impact of such delays on HPC application performance.

## 2.3 Moving forward - EEE on 40, 100, 400Gb Ethernet

**Energy Efficient Ethernet (EEE) with Fast-Wake:** the 1Gb and 10Gb variant of Energy Efficient Ethernet (EEE) ratified in 2010, subsequent standardization efforts, which focused on building specifications for 40Gb, 100Gb and 400Gb backplane and optical Ethernet, adopted the energy savings mechanisms from EEE. In the process of integrating the aforementioned Low Power Idle or **Deep-Sleep** (as it is currently known and renamed), an additional sleep state, **Fast-Wake**, was first introduced by the IEEE 802.3bj task force in charge of the 100Gb Backplane and copper cable standardization effort.

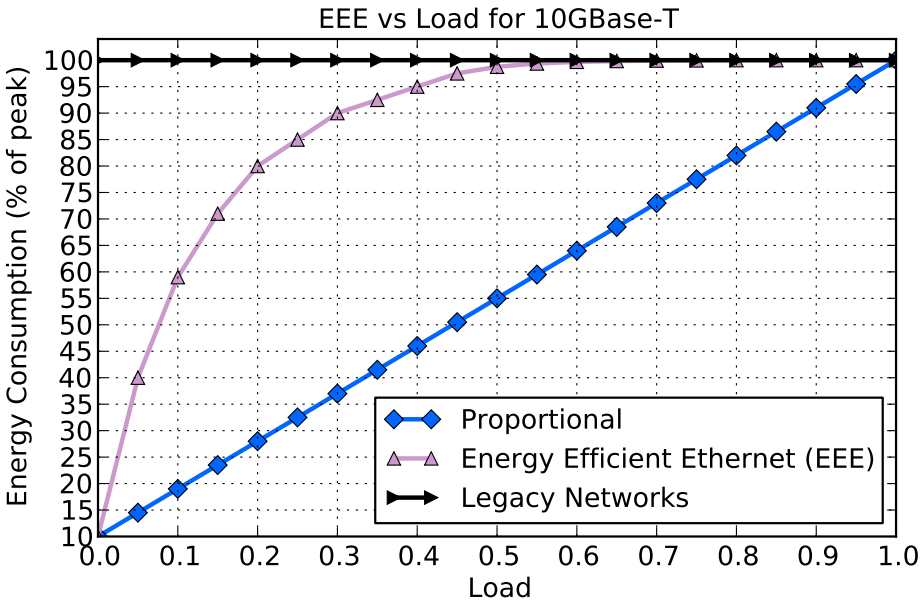



Figure 2.2: Energy consumption vs load for 10Gb/s Link on data center-like workload (figure reproduced from literature (23)); Figure shows how energy consumption of Energy Efficient Ethernet links quickly increases with increase in load.

The motivation for a Fast-Wake mode came from the relatively high wake-up time of the Deep-Sleep mode, whose effect on performance is more pronounced at higher link speeds. The possible increase in latency due to Deep-Sleep is considered to be too high, so it is expected to be disabled for 100Gb links. The long wake-up delay in the original EEE standard comes from signaling components, specifically the Physical Medium Attachment (PMA) and Physical Medium Dependent (PMD) components in the PHY,

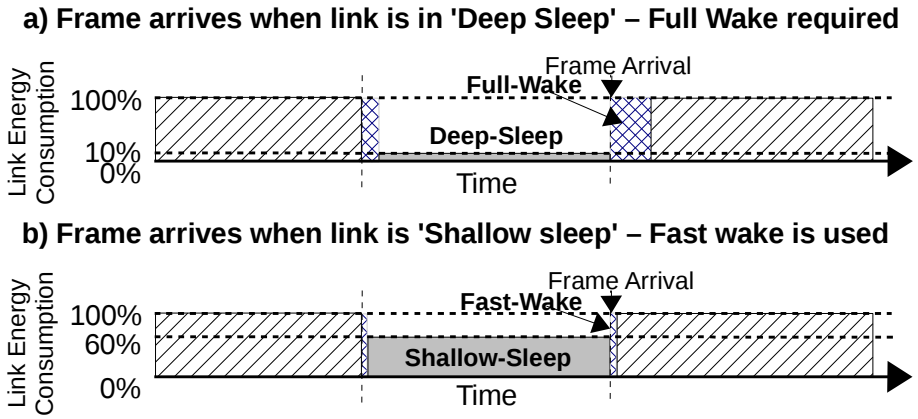



Figure 2.3: Example timeline illustrating various low power states of an EEE link

that are required to synchronize the transmitting and receiving links before data transmission. In the case of Fast-Wake, while some components are powered down, the PMA and PMD remain active, continually transmitting signals between transceiver and receiver, hence maintaining synchronization. This allows for a link wake-up in a few hundred nanoseconds, rather than microseconds. Specifically, studies show that the link could wake up from Fast-Wake mode in $250\,\text{ns}$ to $500\,\text{ns}$, with power savings of 20–40%, compared to an active link [58]. Recent specifications show that Deep-Sleep may not be currently compatible with Optical Transport Network (OTN) based optical networks, however, the standards expect to incorporate Deep-Sleep in these devices in the future.

Figure 2.3 illustrates the working of the above described mechanisms using an EEE-based on/off link. In Figure 2.3(a), the link remains active during frame transfer, which is followed by a state change that turns off the link, reducing power consumption to 10%. The later frame arrival during Deep-Sleep requires a **Full-Wake** ($4.48\,\mu s$) to power up the link to 100% before transmission. Figure 2.3(b) is similar to that of the Figure 2.3(a) except that the link powers down to **Shallow-Sleep**, which reduces power consumption to 60%. The link powers back to 100% faster with Fast-Wake, since it only requires a few hundred nanoseconds to do so.

Products supporting Fast-Wake may be deployed in HPC systems within a year. The standards supporting Fast-Wake are IEEE 802.3bj and 802.3bm, for 40Gb/100Gb backplanes and optical Ethernet, respectively, ratified as recently as March 2015, and IEEE 802.3bs for 400Gb, which is expected to be ratified in 2017. Switches that support EEE, targeting data centers, were commercially available within a year from the date

of standardization. Without investigation and understanding of Deep-Sleep and Fast-Wake on HPC applications with especially emphasis on their performance impact, HPC vendors are likely to disable these mechanisms by default. In the following chapters, this thesis discusses the performance and energy impact and benefits of these mechanisms respectively.

## 2.4 Energy Saving in Infiniband

Infiniband, as in the case of Ethernet is widely used in HPC. While the system share of Infiniband is about the same as Ethernet, its performance share is usually higher. Typically Infiniband supports two mechanisms that can be used for energy savings and they both focus on changing its operating bandwidth to save energy. First example is similar to the above mentioned Adaptive Link Rate (ALR) with link operating at different data rates, example Quad Data Rate at 10Gbit/s per serial lane and Single Data Rate (SDR) at 2.5 Gbit/s per lane. Lower data rates as in ALR require lower energy. Secondly, it is common for Infiniband to aggregate several serial lanes to form a single link as a means to increase effective bandwidth [56, 61]. In both the above cases, when the traffic is low, the idea behind saving link energy is to lower the operating frequency or number of lanes to their respective minimum since lower bandwidth and fewer operating lines consume lower power. This thesis specifically focuses on the Energy Efficient Ethernet for the following two reasons, 1. the results obtained are directly applicable to Ethernet vendors building EEE for HPC and 2. at the lowest energy state, Low Power Idle (LPI) and/or Deep-Sleep of Energy Efficient Ethernet consumes 10% power while even at lowest bandwidth Infiniband links consume 40% energy [48, 56, 61]. However, the results presented and approaches in turning links from active state to sleep state with EEE may directly be applicable to Infiniband in changing between fewer active links and a fully active link.

## 2.5 Other Related Work

In this section, we discuss other proposals related to Energy Efficient Ethernet (EEE) and energy-proportional interconnects. Jian Li et al.,[49] proposed a similar on/off based interconnect, which makes use of system events to overlap link *on* transaction delay. Their approach requires a control network which sends messages that switch

*on* the links prior to sending the message and thereby overlaps the link *on* transaction delay with the pre-processing of messages. The control network in their approach is required to be always powered *on*, in order to receive and transmit control signals. They evaluate their methodology over three power *on* modes, each with 1$\mu$s, 2$\mu$s and 1ms as their wake-up delays over a simulated PowerPC-like 32 node machine. In our approach, we evaluate four different machines, over corresponding EEE standard defined wake-up and sleep delays, their effect on power and performance. Our model uses power estimates and energy efficiency schemes as proposed by EEE which does not assume the need for extra control networks. Alonso et al.,[66] in their work propose the use of traffic information to switch *on* or *off* networks to save power. Similar to the above, their approach requires an always-on control link to maintain connectivity.

Dannis et al.,[48] proposed energy proportional interconnects based on a similar method of reducing link power consumption. In their work on energy proportional interconnects, they evaluate the idea of reducing power consumption during periods of inactivity. However, their approach towards energy efficiency involves reducing the link rates of aggregated links. Aggregated links are typically networks built using multiple links of lower rate, aggregated to a single logical high bandwidth link. In their approach, during periods of inactivity, link rates are reduced to a lower link bandwidth to save power. This work is similar to Adaptive Link Rate proposals for EEE, mentioned in the section. We do not analyze Adaptive Link Rate due to the scheme's above mentioned technology disadvantages.

In the work by Vassos et al.,[65], they provide a design space analysis for On/Off based links. They evaluate two different machines for the power saving capability of On/Off interconnects. For their first machine, they use a fast interconnect similar to an on-chip interconnect, using corresponding on-chip benchmarks. In their second machine, they evaluate a slow network, similar to cluster networks using synthetic benchmarks, with messages arrival following a Poisson process. Similar Ethernet evaluation reports [54, 55] also use synthetic benchmarks to evaluate on/off networks.

Work by Kim, et al., [68] 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

S Miwa et al., [62]propose an evaluation method for EEE supported HPC systems. They show that their model can be used to anticipate performance of future systems that use EEE. Taylor et al., [59] explore dynamic energy saving features in Infiniband

networks and show potential for power savings.

Gupta, et al., in their work [67], show opportunistic sleeping of links is possible, their technique however increases mean latency. Totoni, et al., [72] 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.

R. Bertran et al., [13], evaluate the power and performance trade- offs on the Blue Gene/Q. Among other results, their work shows that the links of Blue Gene/Q are also 'always-on' similar to that of Ethernet and Infiniband. Their work clearly makes a case of how the work presented in this thesis can also be applicable to non-Ethernet but on/off based networks.

J.A. Maestro et al., in their work [35] evaluate EEE for its potential over industrial Ethernet based systems using analytical models. Similarly, Sergio et al., in their work, analytically model and analyze the potential of EEE for energy savings[36]. They provide models for Ethernet standards with frame buffering. Their results suggest that frame buffering offers increased energy savings, however, with the cost of increased packet delays.

Relevant work on Energy Efficient Ethernet [2, 40, 41, 42, 43, 44, 54, 55, 57, 71] provide detailed evaluations on EEE for its potential for desktop and IT based systems; however, they do not give a sense of its performance over HPC.

Yoshi, et al., [70], 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.

# Chapter 3
# Methodology

This chapter presents the experimental methodology used in this work. Briefly, the simulation infrastructure used in this work, Dimemas Cluster simulator is discussed, followed by the model and configuration of the network and simulation parameters. HPC application traces were used to evaluate ideas presented in this thesis and its instrumentation and collection methods are discussed.

## 3.1 Experimental Infrastructure

An extension of *Dimemas* [26, 27] cluster simulator is used for this work. Dimemas has been found to be accurate to within 10% and validated against production supercomputers, including Blue Gene/L, P, Q, and three generations of the MareNostrum supercomputer [27, 28, 29]. The simulation infrastructure supports the execution of *Paraver* [26] traces that for this work were obtained from a production supercomputer - MareNostrum II [53] built with PowerPC970MC blades. The simulator reconstructs the behavior of the actual application from traces that contain CPU intervals and MPI/communication event information (message size, identifiers, type, source-destination etc) from the original execution.

The simulator models simple node modules that contain CPUs, memory and on-board interconnect. Simulated CPUs based on their performance parameters operate relative to the actual application CPU (PowerPC970) intervals recorded in the traces. The default cluster network is modeled as a point-to-point bus based network with duplex independently operating ports, simulating a fully- connected mesh. The communication is based on a linear performance model, however non-linear effects are also considered [27]. More specifically, network conflicts are modeled, with messages queued before transmission when resources are unavailable. Any network conflict intern affects subsequent execution of compute events over CPUs delaying overall execution. Essentially, traces replayed in Dimemas contain inherent communication and

execution dependencies that are used to maintain coherency and correctness in simulations. Dimemas supports the execution of both point- to-point and collective MPI events however our simulations are based on traces while collective communication were mapped to represent their individual point-to- point messages. When representative network resources are allocated, messages transmission time is calculated using the latency, message size and bandwidth. Similar parameters are used in projecting the performance of CPU, Node and Memory.

The first work on Energy Efficient Ethernet, presented in Chapter 4 uses the above mentioned point-to-point network modified support on/off links modeled for after Energy Efficient Ethernet. Carrying this work forward, work on all subsequent chapters 5, 6 and 7 uses a hierarchical network as discribed below. 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-rack switch, which is in-turn connected to a two-level fat tree (4-ary 2-tree) [30]. The architecture of the network used is shown in Figure 3.1.

The network was modified to support static routing with cut-through flow- control and full-duplex links. Each switch in the network routes incoming messages to corresponding outgoing links and requires a switching latency to be configured for the same. Routing decision is statically determined to be the shortest path between source and destination and when multiple paths of equal distance are available the route chosen is simply the modulo of destination times the number of available paths. The system and network parameters were chosen to emulate a high-end HPC system based on analysis of systems in the TOP500 list. Each node is a two-socket high-end CPU with $225$ GF (based on June 2012 TOP500 list machines with two Intel Xeon sockets). The switch latency is configured at $320$ ns for the first hop and $80$ ns for subsequent hops to emulate about $1\,\mu$ss in end-to-end worst-case network latency. Edge links are configured at 20 Gb/s, and the higher two levels are 40 Gb/s and 100 Gb/s respectively. Wake-up and sleep timers for low power states were obtained from Energy Efficient Ethernet specifications and were set to $4.48\,\mu$s and $3.88\,\mu$s for Deep- Sleep [2, 54] and $250$ ns for Fast- Wake [58] respectively.

Each link module is independently implemented low power modes discussed in the rest of this thesis. Each duplex link contains its corresponding states for low power modes. EEE is implemented such that either send or receive can trigger its own low power states within the link. Both sending and receiving links are independently woken up or put into sleep state. The network presented in Figure 3.1 makes for six-hops in the

worst case between source to destination. With EEE each hop along the way requires the links to wake-up before transmission and are otherwise queued until wake-up.

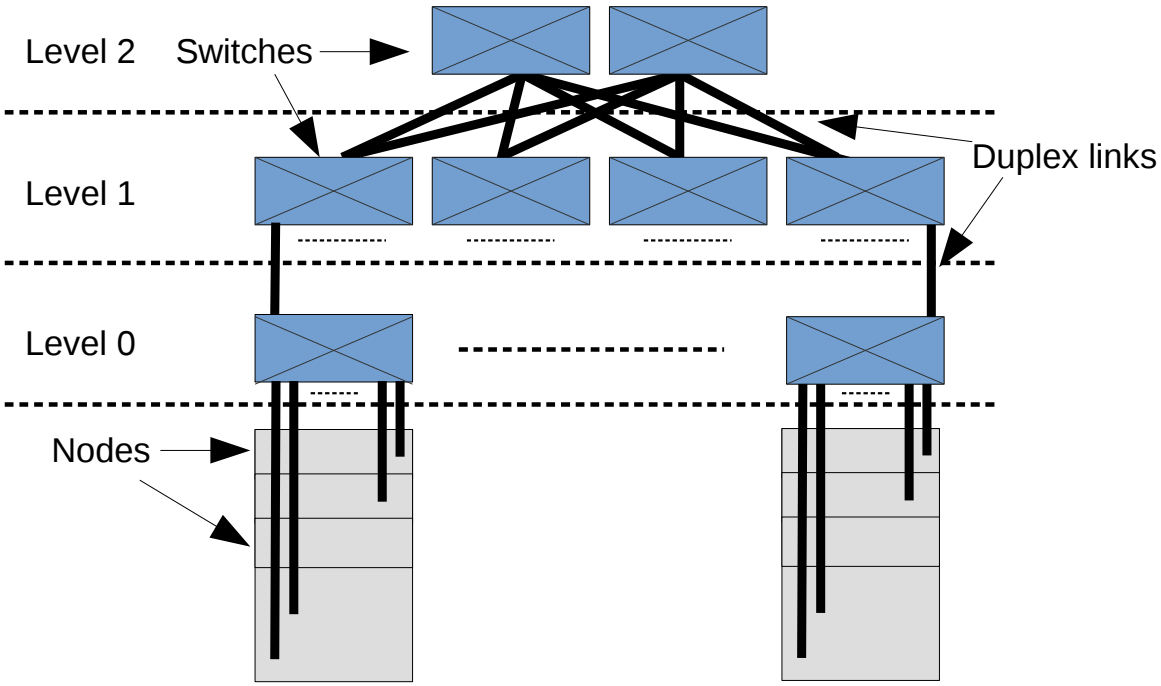



Figure 3.1: Network organization used in the discussion and analysis of PerfBound, Fast-Wake and Double PerfBound discussed in Chapters 5, 6 & 7 respectively.

## 3.2 HPC Applications and Benchmarks

Table 3.1 provides a description of the HPC applications and workloads used in this study. Fifteen workloads in total were used of which, BT, CG, MG, SP and LU are from the NAS Parallel Benchmarks, the rest of the applications are production HPC applications. This thesis uses traces of these applications and the tools and methodology behind obtaining them are discussed below.

### 3.2.1 Trace collection

To obtain traces, the Extrae Instrumentation Tool [14] from Barcelona Supercomputing Center (BSC) was used. Extrae produces trace files for application post-execution

Table 3.1: HPC WORKLOADS USED IN SIMULATIONS

| Name | Nodes Executed | Class | Description |
|---|---|---|---|
| ALYA [73] | 256 | Biology | Biomechanics |
| BT, CG, MG [74] | 256 | Fluid Dynamics | NAS Parallel Benchmarks |
| SP, LU [74] | 64 | Fluid Dynamics | NAS Parallel Benchmarks |
| CPMD [75] | 128 | Chemistry | Molecular Dynamics |
| GADGET [76] | 128 | Astro-Physics | Dark-matter simulation |
| GROMACS [77] | 128 | Biology | Biomolecular dynamics |
| LINPACK [78] | 256 | Benchmark | Linear algebra solver |
| MILC [79] | 128 | Physics | Sub-Atomic Interactions |
| NAMD [80] | 64 | Biology | Biomolecular simulations |
| PEPC [81] | 64 | Mathematical | Parallel Coulomb Solver |
| QUANTUM [82] | 128 | Chemistry | Nanomaterials modeling |
| WRF [83] | 128 | Meteorology | Weather Forecasting Model |

analysis, compatible with the Dimemas Cluster Simulator. Extrae can be used to obtain traces of programming languages and models such as MPI, OpenMP, CUDA, OpenCL, pthreads, OmpSs, Java and Python. Extrae can be configured to instrument various types of information from a program running on a system. The most popular information gathered are, Timestamps, Performance and other counter metrics and references to the source code.

Timestamps are collected in up to nanosecond granularity to provide context to information instrumented. Extrae provides a set of clock functions that accurately gather timestamps with low cost. For performance metrics, Extrae uses PAPI and PMAPI interfaces to collect information regarding the microprocessor. With PAPI, the Extrae tool can be configured to collect information from not only the microprocessor but also the disk, network, OS, among others. Furthermore, entry and exit points of routines can also be instrumented. References to source code imply that Extrae can be used to collect information such as function name and line number at specific points in the program.

Traces obtained for this thesis were from HPC production applications with Extrae tool configured to trace high-level CPU events, synchronization events and detailed MPI communication events. The produced traces are time-stamped and annotated communication events such as MPI_SEND, MPI_Wait, etc. These traces are also annotated with hardware performance counters and memory usage information associated with each CPU event. The target platform used for obtaining the traces is the MareNostrum Supercomputer [53], which contains a cluster of JS21 blades (nodes), each with four

IBM PowerPC 970MP processors at 2.3 GHz. The network connecting the system is a high-speed Myrinet type M3S-PCIXD 2-I port as well as two GigaBit Ethernet ports. Tracing with Extrae produces CPU timing information that are extrapolated within the Dimemas simulator, however, the communication is not annotated with timestamps. The network events produced are entirely reproduced within the Dimemas simulator to reconstruct communication behavior and delays.

### 3.2.2 Trace processing

Traces obtained from Extrae are processed using an in-house tool, Paraver, from the Barcelona Supercomputing Center (BSC). The Paraver [26] visualization tool is generally used for trace analysis. The traces produced by Extrae are compatible with Paraver and are used to visualize application execution in-order to identify performance bottlenecks or potential improvements to the parallel code or MPI communication structures. A sample Extrae generated trace of application FT [74] running with 256 processes is shown in Figure 3.2. In Figure 3.2, the yellow lines represent MPI communication, red representing communication waits and blue representing compute or CPU events.

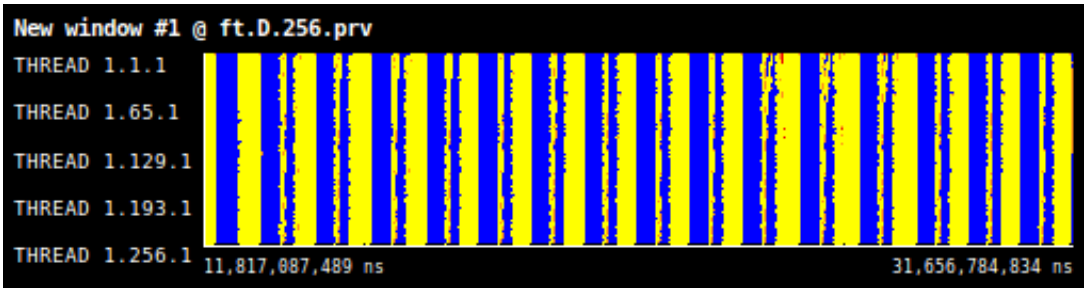



Figure 3.2: Visualization of application FT, a sample Extrae generated trace on Paraver.

The communication behavior of HPC applications, as seen in these traces, are discussed in detail, the following sections. However, in this specific case, the trace shown is an execution between time-periods 11.81 s to 31.65 s. Typical HPC application execution times last many hours and even at small problem sizes can typically take minutes in real world execution. Hence the traces produced generally tend to be tens to hundreds of gigabytes in size, making them hard to simulate or work with.

To solve the above problem, Paraver offers as a feature, trace filtering and cropping. Traces can be cut and filtered to reduce it's size and consequently simulation time. In

this thesis Paraver was used to select events that are relevant, such as MPI communication, filtering out others that are less relevant (example, detailed main memory access). Furthermore, traces are cut in specific sections to produce a smaller or a subsection of a given trace. The tool can be used to visualize and obtain various statistics from a given trace and application run. The Dimemas simulator is also capable of producing Paraver traces, hence the simulated trace can be compared with the trace of the original execution on the actual machine.

Trace shown in Figure 3.2 would make for a typical cut containing many iterations application's outer loop execution. This iterative execution pattern seen in Figure 3.2 typically continues and is only different during the application initialization and finish. However, since those application start and end phases are generally small and since most of the application consists of iterations as observed in Figure 3.2, this cut makes for a good representation of the application characteristics as a whole.

All applications used in this study have diverse network requirements; their network latency and bandwidth sensitivity are discussed in the Chapter 4. The analysis for Energy Efficient Ethernet - Low Power Idle, timing information used is provided by the IEEE 802.3az EEE[1] draft (Table 2.1).

# Chapter 4
# Energy Efficient Ethernet on HPC

This chapter presents an analysis of Energy Efficient Ethernet over HPC. A summary of the chapter and a recap of the problem statement as discussed in the introduction and background chapters is presented below followed by actual analysis and findings.

## 4.1 Summary

Interconnect links can be attributed to consuming a substantial portion of the total interconnect power. Links take up to 60% of the power budget of the network infrastructure [71]. Conventional network links are essentially always on, thereby dissipating power, regardless of whether or not data is being transmitted. The average power consumption of the IBM InfiniBand 12X link, for example, is almost identical to its worst-case power [54, 71]. In the case of Ethernet, its design require both the transmitters and receivers to operate continuously to keep the link aligned. In order to mitigate such waste, the IEEE 802.3az Energy Efficient Ethernet (EEE) standard (approved in Sept 2010) brought about energy saving schemes that make the energy consumption of the link proportional to its utilization.

While numerous interconnects for HPC exist, Ethernet is a popular choice due to its high performance to cost. About 40% of all supercomputers in the Top500 list use the Ethernet family as their interconnection network. Although the IEEE's Energy Efficient Ethernet standard is being incorporated into the network infrastructure in the areas of commercial IT, desktop, servers and high end data centers, the question of how EEE would perform under HPC workloads requires answering.

Although EEE as a standard promises energy proportionality by design, previous literature do not discuss its behavior from the perspective of HPC workloads. The biggest justification for the deployment of the EEE standard came from desktop and

IT infrastructure, which remain idle for the majority of the time. EEE saves power by turning off links when they are not in use - a feature known as Low Power Idle (LPI). However, the time involved in the turning *off* and *on* links, adds additional latency to the messages transmitted. This added latency is known to be destructive to performance and power of data center workloads (whose communication is modeled as a Poisson process) and hence, frame buffering as a solution was suggested to tackle this conundrum. However, it is uncertain as to whether these solutions would also be beneficial to HPC.

The specific contributions in this chapter are as follows:

1. The first evaluation of Energy Efficient Ethernet under HPC workloads to determine the potential of EEE for energy proportional supercomputer interconnects.

2. Latency sensitivity analysis on HPC workloads to project performance estimates for plausible interconnect latencies determined by EEE's energy savings schemes.

3. A proposal for the use of textitPower-Down Threshold as a technique to reduce overhead involved with EEE's additional latencies. This work experimentally compares and demonstrates how the proposed textitPower-Down Threshold scheme significantly reduces the on/off transition overheads, compared to out-of-the-box Energy Efficient Ethernet. Finally, based on this analysis design recommendations are proposed for vendors intending to deploy EEE for HPC systems.

## 4.2 Methodology

**Simulation Infrastructure specifics**

The simulation infrastructure is discussed in the methodology chapter 3, however, here certain specifics are summarized below. The work uses the Dimemas cluster simulator modified to support energy savings schemes from EEE. A notable difference (as previously mentioned in Chapter 3) is that this chapter evaluates EEE over a modeled as a point-to-point mesh with duplex links as opposed to a hierarchical network evaluated in the following sections.

| Test Machine | Node Perf. | Interconnect |
|---|---|---|
| Machine-Low | 62 GFlops | 2x10Gbps links with 1$\mu$s latency |
| Machine-Mid | 112 GFlops | 40Gbps links with 1$\mu$s latency |
| Machine-High | 224 GFlops | 100Gbps links with 1$\mu$s latency |
| Machine-Acc | 348 GFlops | 100Gbps links with 1$\mu$s latency |

Table 4.1: Network and node parameters for EEE evaluation

**Experimental Setup and HPC applications specifics**

The evaluation models four different machines, each with varying network bandwidth and node performance as mentioned in the methodology section and as shown in Table 4.1. The choice of node performance for *Machine-Low*, *Machine-Mid*, *Machine-High* and *Machine-Acc* comes from the top four machines of TOP500 (June'12) that use Ethernet. The second fastest Ethernet based machine, Amazon EC2 Cluster, contains Xeon 8-core processor. The sustained node performance of the above machine is estimated to be 112 GFlops/node with one socket/node and 225 GFlops/node with 2 sockets/node (Calculated based on system information provided at June'12 TOP500 list). The fastest Ethernet based system contains Xeon processors with NVIDIA 2090 GPUs; whose node performance is estimated based on a similar machine Tianhe-1A (TOP500 Rank 4) at 349 GFlops/node, for the accelerator based machine. Similarly, for *Machine-Low* performance is estimated based on a machine ranking 4th among the fastest Ethernet based TOP500 machines at 62GFlops. Node-to-node latency is assumed to be of 1$\mu$s across test machines based on relevant literature and specification sheets of products.

The workloads uses in this work are CPMD, GADGET, GROMACS, LINPACK, MILC, NAMD, PEPC, QUANTUM and WRF. The specifics of the number of nodes executed and trace information is the same as discussed in the methodology chapter.

Analysis of Low Power Idle mode of Energy Efficient Ethernet is setup with timing information provided by the IEEE 802.3az EEE[1] draft (Table 2.1). EEE standard does not provide timing information on the wake and sleep modes for Ethernet links with bandwidth upwards of 10Gbps. Since the network links considered for this study are 10Gbps and upwards, timing information of 10Gbps links is assumed for for 40 and 100 Gbps links.

Energy consumption is modelled based on data available in the industry and relevant literature [32, 33, 34, 49]. Energy consumption of links is assumed to be 10% of

the full link power in idle mode based on the IEEE 802.3az EEE [1] draft. To calculate system power consumption, it is assumed that links consume 65% of the network power and consider total network power to be about 12-20% of the system power[34, 48, 49]. Consequently energy consumption of each of the machines (Low, Med, High and Acc) is calculated based on the above model and network utilization. Similar to previous literature [33, 48] other network elements are assumed to consume power proportional to utilization.

## 4.3 Interconnect sensitivity analysis

The following sections presents evaluation of the IEEE Energy Efficient Ethernet standard on actual HPC workloads. First evaluation of HPC workloads for their bandwidth and latency sensitivity is presented to draw conclusions for EEE based on the same. Following which, link activity is examined for their potential for power savings with EEE and finally performance and power evaluation of Energy Efficient Ethernet is discussed. To evaluate Energy Efficient Ethernet, its links on various test machines are simulated and results are obtained for various workloads. Further, EEE is compared with the proposed *Power-Down Threshold* and power estimates for the same are drawn based on the results obtained. Although this study focuses on Energy Efficient Ethernet, the evaluation and discussions can apply to design decisions of other On/Off networks for HPC.

HPC workloads are typically characterized by alternating compute and communication phases (as shown in Figure 4.1). These applications most often contain very large compute phases followed by relatively smaller communication phases, usually bursts of MPI messages. Further, these compute and communication phases usually have dependency patterns characteristic of the HPC applications. These dependencies may cause design decisions such as Frame buffering in EEE [37, 54, 55] to be destructive to both application performance and power (more discussed in later sections). Further, since HPC workloads in general do not overlap communication and compute periods, the time involved in communication is destructive to performance. This results in HPC applications being heavily optimized to keep communication time at its minimum, translating to low interconnect usage. However, HPC systems running these applications with conventional always-on links, do not benefit from this inherent power saving opportunity. Figure 4.1 shows alternating compute and communication phases of application WRF[83] along with its corresponding interconnect usage. The figure clearly shows that the interconnect links are predominantly idle. Since these links are

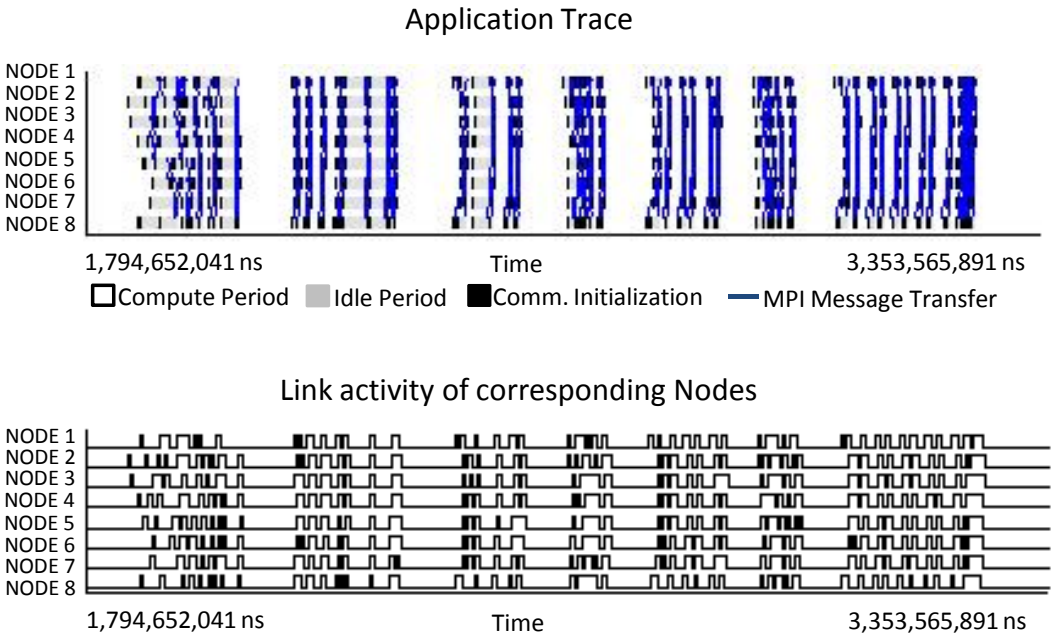



Figure 4.1: Execution and Link activity pattern of WRF application. Sub-figure a) A cut of the actual application run showing execution patterns. Sub-figure b) Link activity corresponding to execution - peaks represent data transmission. Note that communication occurs in phases and are correlated.

typically always-on, continually consuming power, they would benefit greatly if built to be energy proportional.

As shown in Figure 4.2, the requirements of the presented workloads vary widely in terms of their sensitivity to network latency and bandwidth. Figure 4.2 presents normalized execution time as a function of latency (Figure 4.2(a,b)) and bandwidth (Figure 4.2(c,d)). Figure 4.2(a,c) shows the execution time for each application, on a fixed machine, *Machine-Low* and Figure 4.2(b,d) shows the execution time for each machine (Low, Med, High and Acc), with *Max, Min* and *Avg* corresponding to the most sensitive and least sensitive application and the average across applications. The presented graphs point out the diversity of HPC applications with their bandwidth and latency requirements. Furthermore, this work shows how the choice of interconnect bandwidth and latency for the corresponding systems fall within a 10% drop in performance in comparison to infinite bandwidth and zero latency, to which the graphs are normalized. A 10% drop in application performance is generally not ideal for state-of-the- art HPC designers. However, the diversity of applications and large interconnect

capital costs play a factor, often making it difficult to build the perfect interconnection network for all applications. However, when faster interconnects are used, since EEE is intended to be energy proportional, higher performance will translate to lower average power consumption.

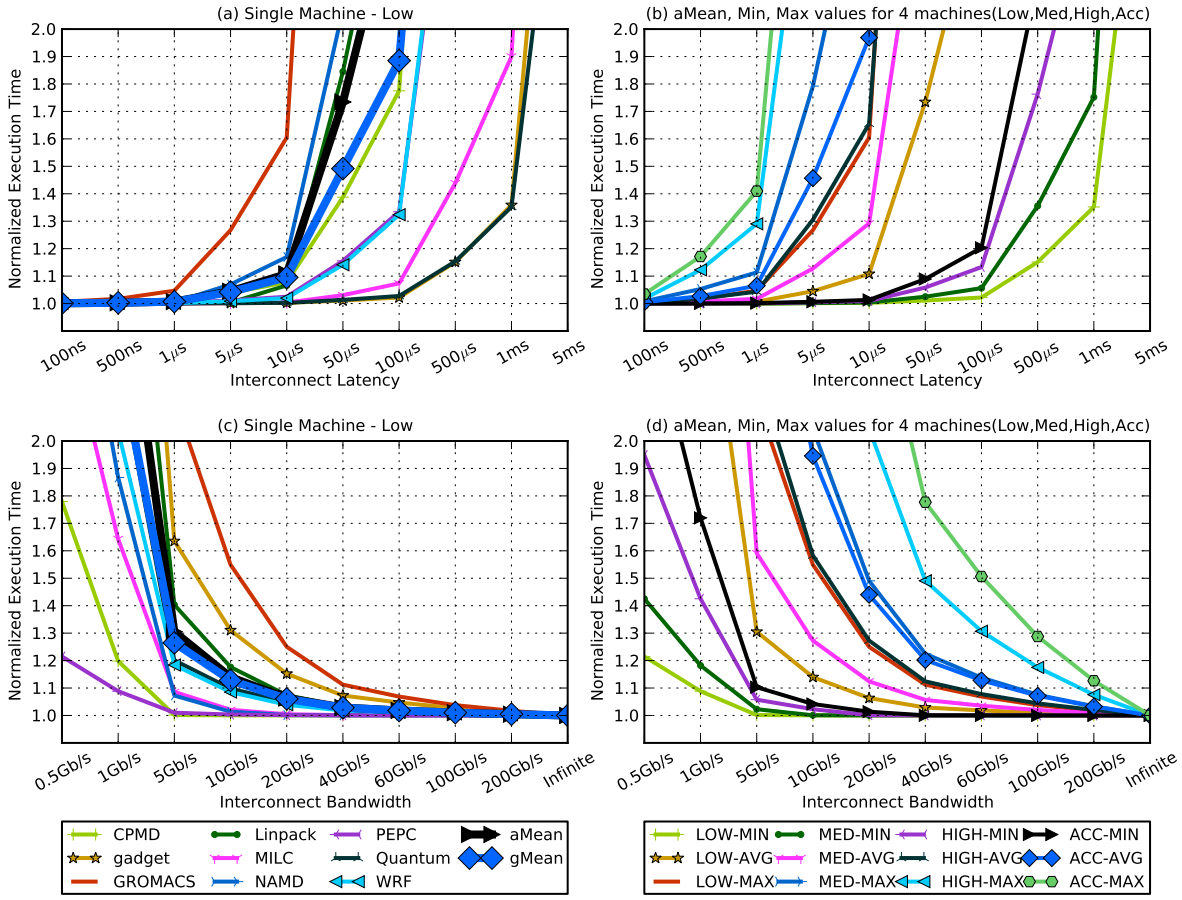



Figure 4.2: Bandwidth and Latency sensitivity of the applications executed on the four test machines (Low, Med, High and Acc)

## 4.4 Idle link event time analysis

Energy Efficient Ethernet uses Low Power Idle (LPI) as a proposed mechanism for bringing about energy proportionality. LPI works on the basis that links are turned off during

periods of inactivity. While this proposal seems inherently energy efficient, its power saving potential is heavily dependent on the frequency at which the link becomes inactive. Due to a non-negligible increase in latency imposed by LPI every time a link needs to *wake-up* from low power mode, the frequency at which these links are turned *on* and *off* is crucial to power savings. In this regard, the time required for powering the link back to operational mode determines power savings as well as performance overheads. Conventional wisdom suggests that deeper power saving modes require correspondingly large *wake* times. Note that a large *wake* time introduces large latencies, which in turn results in large performance overheads. To keep performance overheads at a minimum, networks that require large *wake-up* times would rarely be put into low power modes. To understand and evaluate power saving benefits offered by On/Off based interconnects with specified *wake* times applications are analyzed for their link idle time distribution.

Figure 4.3 presents histograms to illustrate and describe intervals during which links remained idle, running application *GADGET* over *Machine-Mid*. Histogram 1, shows *idle link events*[1] of various time intervals. The x-axis of Histogram 1 shows the time distribution and correspondingly for each point in the x-axis the number of events along the y-axis. The time distribution corresponds to the time interval during which the link remained idle. To illustrate, there exists a peak at 100$\mu$s with approximately 3,750 events. The 3,750 events in this peak correspond to 3,750 separate intervals during which the link was idle, each of whose duration is close to 100$\mu$s. The peaks between 1$\mu$s to 4$\mu$s and 100$\mu$s to 400$\mu$s in Histogram 1 of Figure 4.3 are a result of self- similarity or repetitive patterns with these applications. This essentially shows that the distribution that holds true for one or more iterations of a cut of the *GADGET* application, will remain similar throughout all iterations of the application execution.

Histogram 2 of Figure 4.3, shows the above Link Idle Event distribution in actual time periods. Here the bars of the histogram represent the total idle time throughout the application's execution, from all idle intervals of length close to the value on the x-axis. To illustrate, consider the bar at 100 ns which contains an y-axis equivalent of approximately 1$\mu$s. This shows that the total time in all idle link events that lasted close to 100ns, accumulated over the whole trace execution is 1$\mu$s. Although a majority of the idle link events come from the range of 1$\mu$s to 4$\mu$s, as shown in histogram 2, their accumulated link idle time does not exceed a few milliseconds. Histogram 3, converts Histogram 2 into its cumulative distribution graph to determine the cumulative time in all idle periods of length less than the value on the x-axis. For example, the total time

[1] An *idle link event* is assumed to be any interval during which no data is being transmitted on the link

in all intervals of length 100$\mu$s or less is above 1 second. Histogram 3, calculates the interval length for which 90%, 99% or 99.9% of the total idle link time is in idle intervals that are longer than that value. In this case, for application *GADGET* on *Machine-Med*, 99.9% of the total application execution time that links remained idle, comes from events when a link remained idle for 52$\mu$s or longer. The corresponding values for the other applications and machines are listed in table 4.2. With the exception of GROMACS, all applications on Machine-Mid have 90% of their idle link times in the ranges of milliseconds or above.

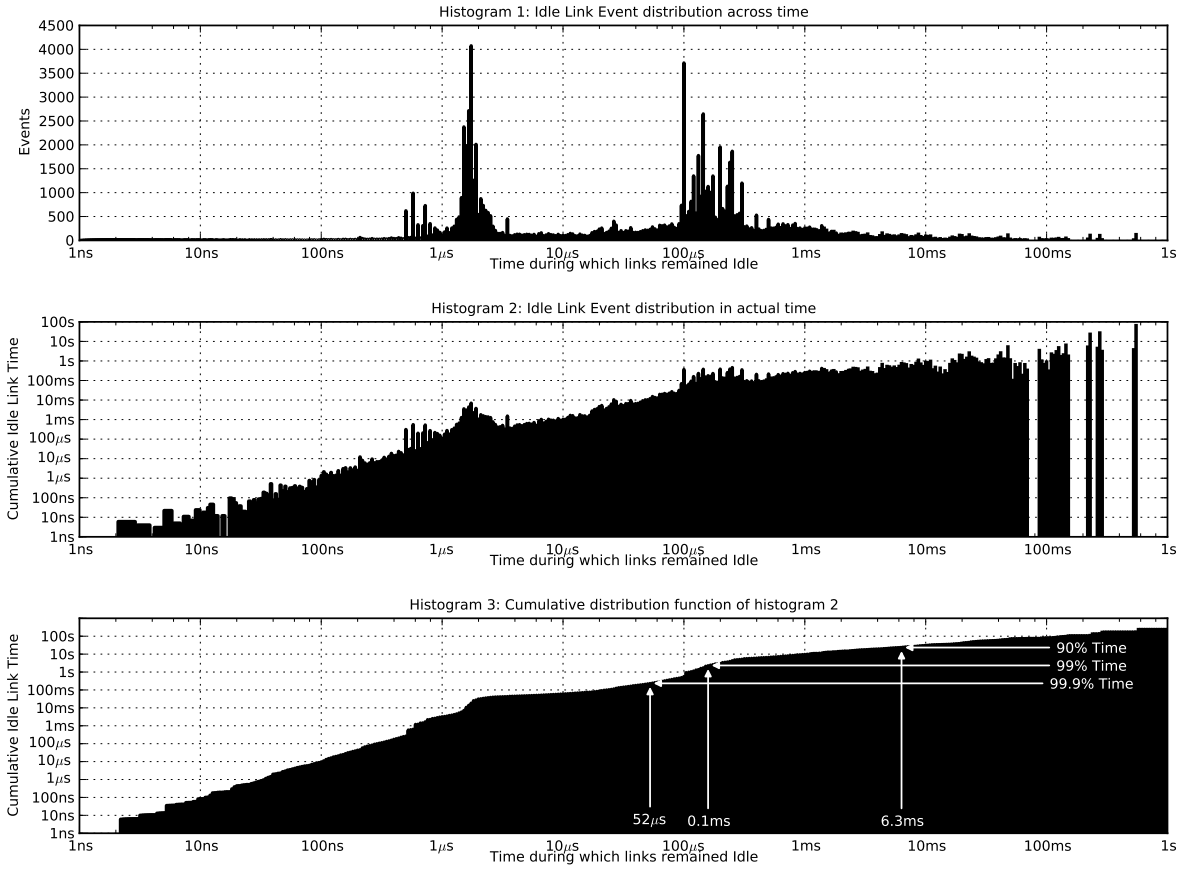



Figure 4.3: Histograms show Idle Link Event distribution for application *GADGET* running on Machine-Mid. Histogram 1 (top) shows the number of idle link events as a function of time. Histogram 2 shows a product of total idle link events by idle link time for the corresponding events. Histogram 3 shows the cumulative distribution graph of Histogram 2

From the design perspective of power savings modes for On/Off based interconnects such as Energy Efficient Ethernet, these numbers are relevant in designing *wake* timings and to gauge potential power savings using the above. For 10Gb Ethernet, the link *wake-up* and *sleep* times proposed by the IEEE 802.3az EEE [1] draft are 4.48$\mu$s and 2.88$\mu$s respectively. This is to show that an EEE link requires approximately 7$\mu$s in total to switch off and back on. For applications such as *GADGET*, which contains 99% of its link's total idle time coming from events where the link remains idle for beyond 100$\mu$s, 7$\mu$s potentially offers enough time to go into and back from low power mode without significant loss in performance.

Large gaps in the ranges of 70ms to 800ms shown Histogram 2 of Figure 4.3 represent lengths of idle intervals that never occur. Essentially, if we consider a gap near 100ms (assume the gap is between 70-80ms), this is to show that no link that has gone idle, has had an idle time that has lasted between 70 and 80ms. Note that these gaps occur in the range of 70ms to 800ms, which contribute to a large portion of the total link idle time (82% in the case of the histogram 2 in Figure 4.3). The reason behind these gaps may be that the majority of the execution time contains distinctively large computation periods unhindered by communication events (during which all links remain idle). The analysis on experiments over various machines suggests the existence of these patterns in all applications except GROMACS, NAMD and Quantum Espresso. In the case of the Histogram 2 in Figure 4.3, the gap between 70 to 80ms suggests that, when a link has been idle beyond 70ms, it is certain that it would remain idle for another 10ms (until 80ms) in time. These results suggest that even a simple history based prediction algorithms, that dynamically records this information to switch On or Off links would greatly benefit from deeper sleep states of the network, however, this work is beyond the scope of this work.

## 4.5 Performance and power analysis

This section evaluates Energy Efficient Ethernet, its power and performance with and without the proposed *Power-Down Threshold*. As mentioned in the previous sections, *wake-up* latency of 4.48$\mu$s and a *sleep* latency of 2.88$\mu$s is assumed for the simulations. This latency is the standard proposed wake-up and sleep latency of 10Gbps which we use for all test machines.

**Power-Down Threshold**: From the analysis of HPC workloads, it is clear how extremely latency intensive HPC workloads can be. The previous proposals suggest the

| Applications | Machine-Low | | | Machine-Mid | | | Machine-High | | | Machine-Acc | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Idle Link Time | 90% | 99% | 99.9% | 90% | 99% | 99.9% | 90% | 99% | 99.9% | 90% | 99% | 99.9% |
| CPMD | 10s | 0.4s | 50ms | 5.7s | 0.22s | 22ms | 2.8s | 0.11s | 0.2ms | 1.8s | 72ms | 10$\mu$s |
| GADGET | 10ms | 0.2ms | 91$\mu$s | 6.3ms | 0.1ms | 52$\mu$s | 3.3ms | 72$\mu$s | 19$\mu$s | 1.9ms | 52$\mu$s | 13$\mu$s |
| GROMACS | 3.9$\mu$s | 1$\mu$s | 0.3$\mu$s | 2$\mu$s | 0.7$\mu$s | 0.2$\mu$s | 1.6$\mu$s | 0.5$\mu$s | 0.18$\mu$s | 1.3$\mu$s | 0.4$\mu$s | 0.1$\mu$s |
| LINPACK | 22ms | 0.3ms | 2.75$\mu$s | 11ms | 0.1ms | 1.8$\mu$s | 6.6ms | 31$\mu$s | 1$\mu$s | 9.1ms | 3.6$\mu$s | 72$\mu$s |
| MILC | 1.3ms | 0.1ms | 25$\mu$s | 0.7ms | 79$\mu$s | 13$\mu$s | 0.3ms | 34$\mu$s | 6.3$\mu$s | 0.2ms | 30$\mu$s | 4.7$\mu$s |
| NAMD | 0.1ms | 26$\mu$s | 1.4$\mu$s | 0.1ms | 12$\mu$s | 91$\mu$s | 45$\mu$s | 4.1$\mu$s | 1$\mu$s | 28$\mu$s | 1.9$\mu$s | 75$\mu$s |
| PEPC | 0.6ms | 45$\mu$s | 4.1$\mu$s | 0.3ms | 21$\mu$s | 2.0$\mu$s | 0.1ms | 9.1$\mu$s | 1.1$\mu$s | 0.1ms | 5.2$\mu$s | 1.1$\mu$s |
| QUANT.ESP | 1.8ms | 0.9ms | 0.1ms | 1ms | 0.5ms | 87$\mu$s | 0.4ms | 0.3ms | 41$\mu$s | 0.3ms | 0.1ms | 28$\mu$s |
| WRF | 0.2ms | 27$\mu$s | 7.2$\mu$s | 0.1ms | 15$\mu$s | 3.9$\mu$s | 79$\mu$s | 7.5$\mu$s | 1.3$\mu$s | 52$\mu$s | 5.2$\mu$s | 95$\mu$s |

Table 4.2: Distribution of link Idle times: The numbers indicate that 90%, 99% or 99.9% of the total time a link remains idle during the entire application execution, comes from events where the link remains idle for a period above the table specified time for their respective machines

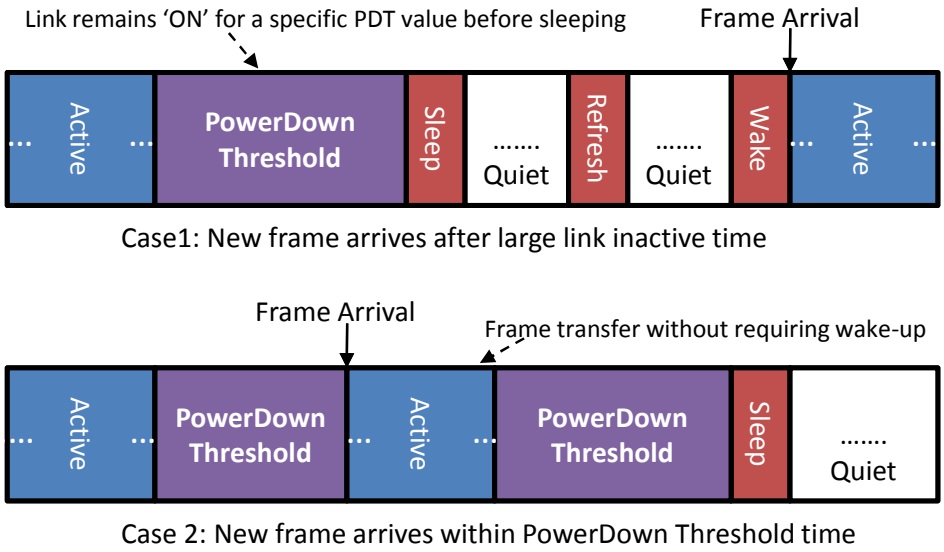



Figure 4.4: State transactions of Power-Down Threshold (PDT) for Low Power Idle in Energy Efficient Ethernet - Link remains in 'on' state after frame transmission for specific PDT time. Frame arrival within PDT time does not incur additional latencies for the turning *off* and *on* the link.

use of Frame Buffering, where frames are held back up to 100$\mu$s before transmission to improve energy savings. However, with regard to HPC, as shown in Figure 4.2, a 10$\mu$s added latency (due to frame buffering) reduces performance on average by about 7% to 40%. This reduction in performance would essentially negate any power benefits obtained by the use of Energy Efficient Ethernet. Hence, as a solution to maintaining performance, as well as saving power, *Power-Down Threshold* as a scheme for EEE for HPC is proposed. While in the case of Frame Buffering, frames are buffered and the link does not switch *on* for transmission until time-out or a specific number of frames are obtained. In the proposed *Power-Down Threshold*, as shown in figure 4.4, after the transmission of a frame, EEE remains in *on* state until a threshold time before shifting back to low power mode. During this duration, frames arriving at the link is transmitted without the need for powering the link back up. This added threshold time would result in increased power consumption, since the link remains *on*, longer than required in the case when no messages arrive before the threshold. However, this work argues that this increased power consumption (in an already energy proportional interconnect) is rather small in comparison to power savings achieved by negating any performance overheads due to the use of EEE.

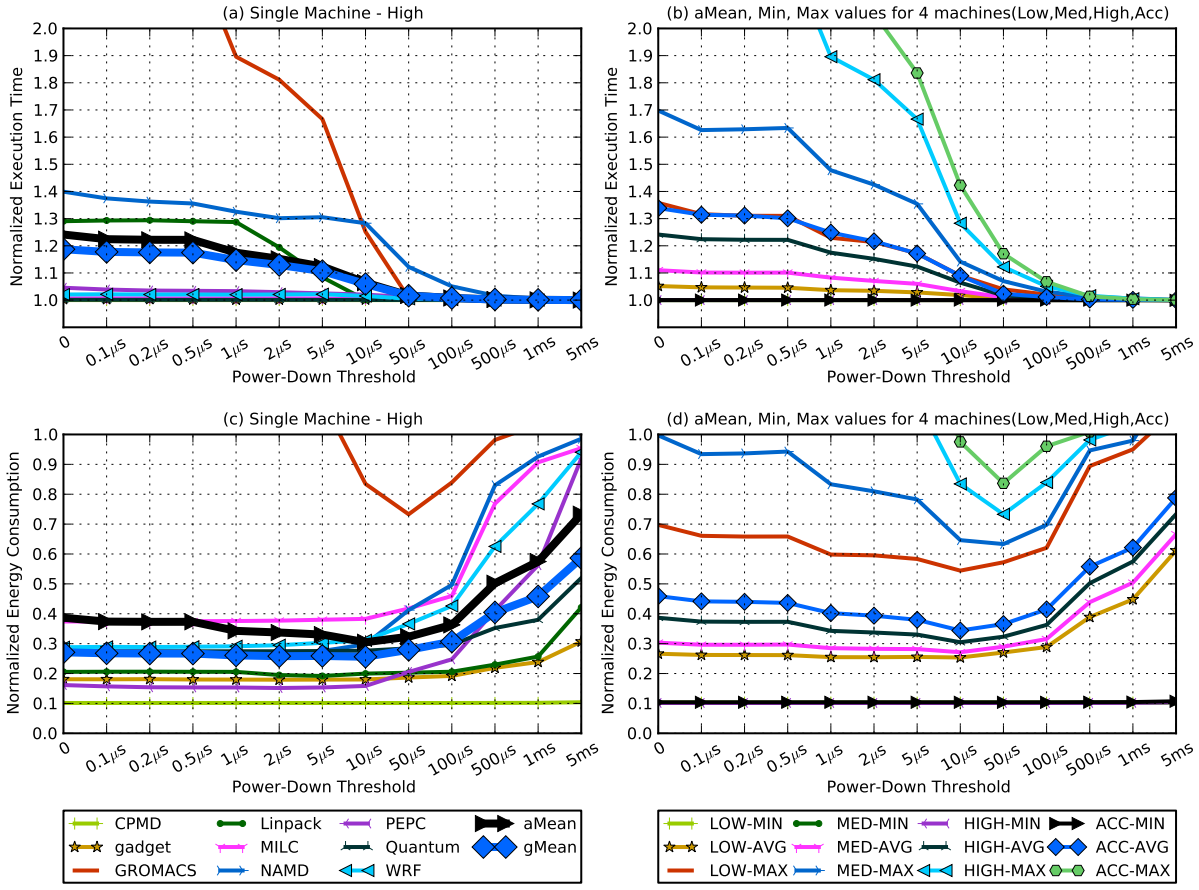



Figure 4.5: Performance and power graphs of Energy Efficient Ethernet over proposed *Power-Down Threshold*

**Experimental Results**: Figure 4.5 shows the experimental results on the evaluation of EEE and *Power-Down Threshold*. In Figure 4.5, sub-figures (a) and (b) corresponds to performance graphs and sub-figures (c) and (d) correspond to the average energy consumption of all links. Further, sub-figures (a) and (c) refer to results for all applications over a single machine and sub-figure (b) and (d) shows results for all 4 machines, with the averages, minimum and maximum values of the applications. In Figure 4.5(a,b), at *Power-Down Threshold* value zero corresponds to Energy Efficient Ethernet where for every message, links are required to be powered *on* and *off* introducing wake-up and sleep transition delays. As seen, from the figure, EEE introduces an average of approximately 25% reduction in performance for Machine-High, 35% for Machine-Acc and 10% and lower for Machine-Mid and Machine-Low. Correspondingly, on an average, there is 60% reduction in the link energy consumption. However, with increase in

*Power-Down Threshold* values, at about 50$\mu$s, the performance overhead of EEE drops down to less than 2% with savings in link energy consumption at 70%.

**Total System Power Estimates**: The above savings in power correspond to link power; here the overall system power consumption is calculated based on the methodology specified in Chapter 3. On considering two cases, where *Power-Down Threshold* is zero (Energy Efficient Ethernet) and 50$\mu$s threshold values (Energy Efficient Ethernet + Power-Down Threshold) the overall power consumption is calculated across the system. As mentioned above, power estimates are projected assuming that node and network elements operate at full power during the entire application execution. However, during real execution this is not necessarily the case, many components including CPUs can enter sleep mode, so the increase in energy may be lower than estimates presented in this work. For Machine-High, shown in Figure 4.5 (a), there is a 25% reduction in performance, with power savings in the interconnect of about 60%. This translates to an overall system power consumption at 115%, suggesting that deploying Energy Efficient Ethernet on Machine-High would lead to a 15% increase in power and 25% decrease in performance. However, with a *Power-Down Threshold* of about 50$\mu$s, the performance overhead drops down to less than 2% with interconnect power savings of 70%, translating to overall system power savings of 7.3%.

## 4.6 Conclusions

This chapter presented the first analysis of Energy Efficient Ethernet for HPC workloads. The *Power- Down Threshold* as a technique was proposed to further increase power savings of HPC systems where the links remained *on* but idle until a threshold is reached. Based on this analysis of HPC workloads on EEE, various recommendations were proposed to further increase power savings. Based on results obtained, this work concludes that out-of-the-box Energy Efficient Ethernet for HPC does not provide sufficient power savings to justify its use. However, with further EEE enhancements such as the proposed *Power-Down Threshold* makes using Energy Efficient Ethernet for HPC a very promising solution for energy proportionality. This work suggests that existing complimentary proposals such as frame buffering that increase the latency of messages are harmful to performance, negating any power benefits obtained by such schemes. The *Power-Down Threshold* however is application dependent and must be dynamically chosen, hence in the next chapter, a dynamic and application independent mechanism is discussed.

*This page has been intentionally left blank.*

# Chapter 5
# PerfBound: Bounding Performance Overheads

The previous chapter introduced Power-Down Threshold (PDT) as a timer to reduce performance overhead while still achieving high link energy savings. It also showed how optimal PDT is application dependent and that a trade-off exists between energy savings and power consumption. This introduces an interesting problem with regard to building a energy savings mechanisms into the link. Any energy savings mechanism introduced for HPC should ensure minimal performance overhead (as discussed in Chapter 1) while obtaining energy savings. In this regard, this chapter introduces PerfBound, that automatically manages the PDT of a link for energy savings but with a bound of possible performance overheads.

## 5.1 Summary

This chapter 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. Therefore *PerfBoundRatio* is proposed, 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, this chapter introduces *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 systems to provide energy proportional interconnects, it is crucial that the incurred performance overheads caused by the same are controlled. In this regard, the key novelty behind this work the approach to on/off link management mechanisms from a performance overheads perspective. To be specific, the novel contributions of this chapter are as follows:

1. A detailed analysis of the communication behavior in HPC applications provides insights on the correct management of on/off links. This work shows 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. A case is made for how the energy savings depend on making the right choice of messages to delay.

2. The above insights are used 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. PerfBoundRatio is proposed, which respects the same bound on the total overhead in a hierarchical network.

3. Finally, a prediction mechanism is discussed 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 chapter is organized as follows. The following section discusses the

specifics in methodology used in the work presented in this chapter. Following which, an investigation the causes of performance degradation is discussed. Based on insights gathered a case for performance bounding in interconnects is presented. Finally the final sections, discuss the evaluation of the proposed techniques and conclude.

## 5.2 Methodology

This chapter uses the methodology outlined in chapter 3, a brief specifics of the same is outlined below. An extension to Dimemas cluster simulator modified to support a hierarchical network is used in this work. The network models EEE based on/off links controlled by the techniques discussed in this chapter. The simulation infrastructure is driven by traces, as discussed in chapter 3, 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 modeled as 100% when "on" or during transition between on/off states, and 10% when "off". All energy figures are normalized to percentage of original energy- to-solution.

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-rack switch, which is in-turn connected to a two-level fat tree (4-ary 2-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 (to represent a High-End system). The switch latency is configured at $320\,$ns for the first hop and $80\,$ns 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. The wake-up and sleep times of $4\,\mu$s and $3\,\mu$s for Energy Efficient Ethernet [2, 54] is used.

This work uses 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. The applications used are as follows, ALYA[73], LINPACK[78], BT [74], CG[74], FT[74], MG[74], QUANTUM[82], WRF[83] MILC[79], GROMACS[77], GADGET[76], NAMD[80], PEPC[81], SP and LU[74].

## 5.3 Motivation

The motivation behind this work comes from the evaluation presented the previous chapter where a application dependent PowerDown Threshold (PDT) ( which is called the LinkOFF threshold in the context of this work) was required for optimal energy and performance. This chapter expands this motivation and presents application analysis to support ideas presented in the following sections.

The chapter 4 discussed how HPC applications require a high-performance interconnect, to support their peak communications demand, but the average utilization of the network is low. Much of the interconnect's idle time is contributed by relatively long idle periods [50, 69]. Figure 5.1 shows the communication behavior of LINPACK [78], BT [74], and NAMD [80]. Fourteen of the fifteen applications as further disccussed in this chapter exhibit regular patterns similar to Figures 5.1(a) and 5.1(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 5.1(c), appears irregular, but it still exhibits low network utilization and considerable interconnect energy savings. Considering this, switching from current-day always-on interconnects in HPC to energy proportional networks presents an opportunity for large savings in energy costs.

Energy Efficient Ethernet uses on/off links, and switching between states require about $4\mu$s. While the physical layer specification provides the underlying mechanisms, the decisions as to when to enter and leave power-saving states are left to the vendor. The previous chapter makes a clear case to show how 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. 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 5.2 shows a sensitivity analysis of application performance to wake-up latency, assuming the naive management technique. The $x$-axis is the wake-up

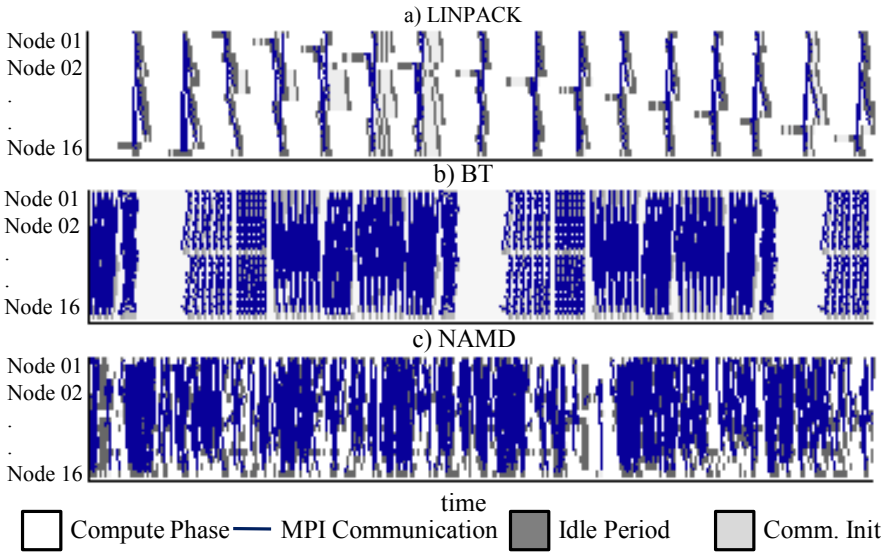



Figure 5.1: Communication behavior in HPC applications - LINPACK[78], BT[74] and NAMD[80]

latency (which for EEE is about 4$\mu$s/link). Applications that are least sensitive, including Quantum[82] and BT[74], can potentially tolerate an aggressive energy saving technique, since the naive approach incurs only about 2% performance overhead. In contrast, GROMACS[77] and NAMD[80] 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 attempts to obtain 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 as presented in the previous chapter, that can work well, is to turn the link off only after a specific duration of idle time (**LinkOFF threshold**), and to turn it back on when the next message arrives. The naive approach described

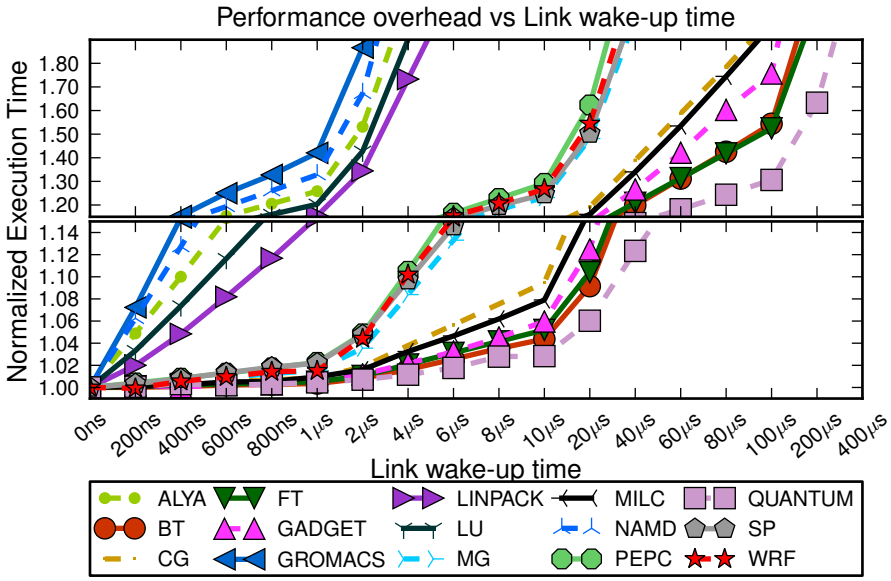



Figure 5.2: Application performance overhead as a function of wake-up delay.

above corresponds to LinkOFF=0. The previous chapter clearly showed that this mechanism can work well in HPC. Since different applications have different sensitivities to increases in latencies, the optimal value of LinkOFF threshold depends on the application. This chapter therefore 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. This work therefore proposes 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 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 is incorrect, the total performance degradation is still controlled.

The 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 runtime 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. On balance, the PerfBound approach gives the right compromise, especially since the results, as presented below, 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.

### 5.3.1 Understanding the overhead behind link wake-up

In order to make performance overhead-aware decisions for link energy savings, it is important to first understand how wakeup latencies translate to performance overheads. Figure 5.2, showed that different applications have different sensitivities to wake-up latencies. To look at this in more detail, this section examines application overheads by applying the wake-up delays selectively. From this point in this chapter, "message inter-arrival periods" are referred to as **idle link events**.[1]

Figure 5.3 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 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

[1]This chapter defines any duration during which no data is transmitted over a link as an *idle link event*

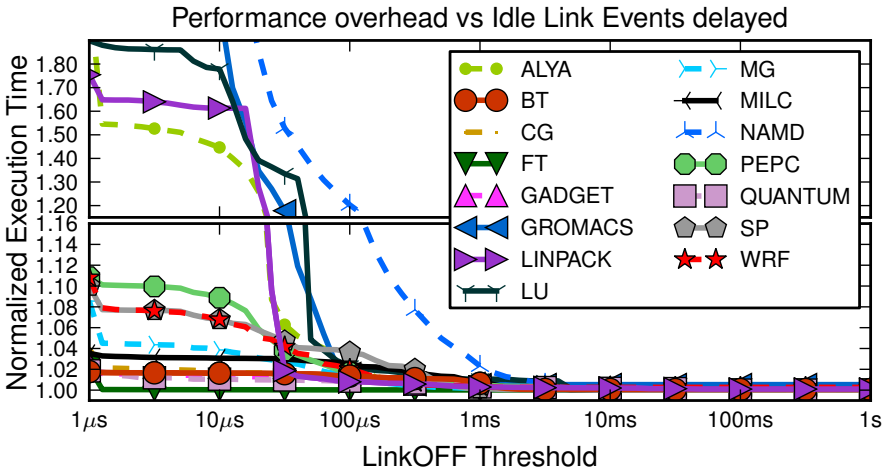



Figure 5.3: Application performance overhead as LinkOFF threshold is varied - normalised to execution over an always-on network.

zero. If an acceptable level of performance overhead for the application is 5%, for example, then Figure 5.3 can be used to determine an application-dependent static value for the LinkOFF threshold. For application LU, for instance, it is clear that an appropriate value of the LinkOFF threshold would be 80$\mu$s. In this case, the link remains on for the first 80$\mu$s 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 5.4. 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[78], BT[74] and NAMD[80] according to the title. Figures 5.4(a), 5.4(c) and 5.4(e) are colored according to the *number of events*, whereas Figures 5.4(b), 5.4(d) and 5.4(f) are colored according to the total idle time contributed by those events. That is, if in Figure 5.4(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 5.4(a) and 5.4(c) is in the clustering of idle link events. In the case of LINPACK, in Figure 5.4(a), the events are clustered at around 10$\mu$s, while the events in BT are clustered at around 1ms. Another key difference

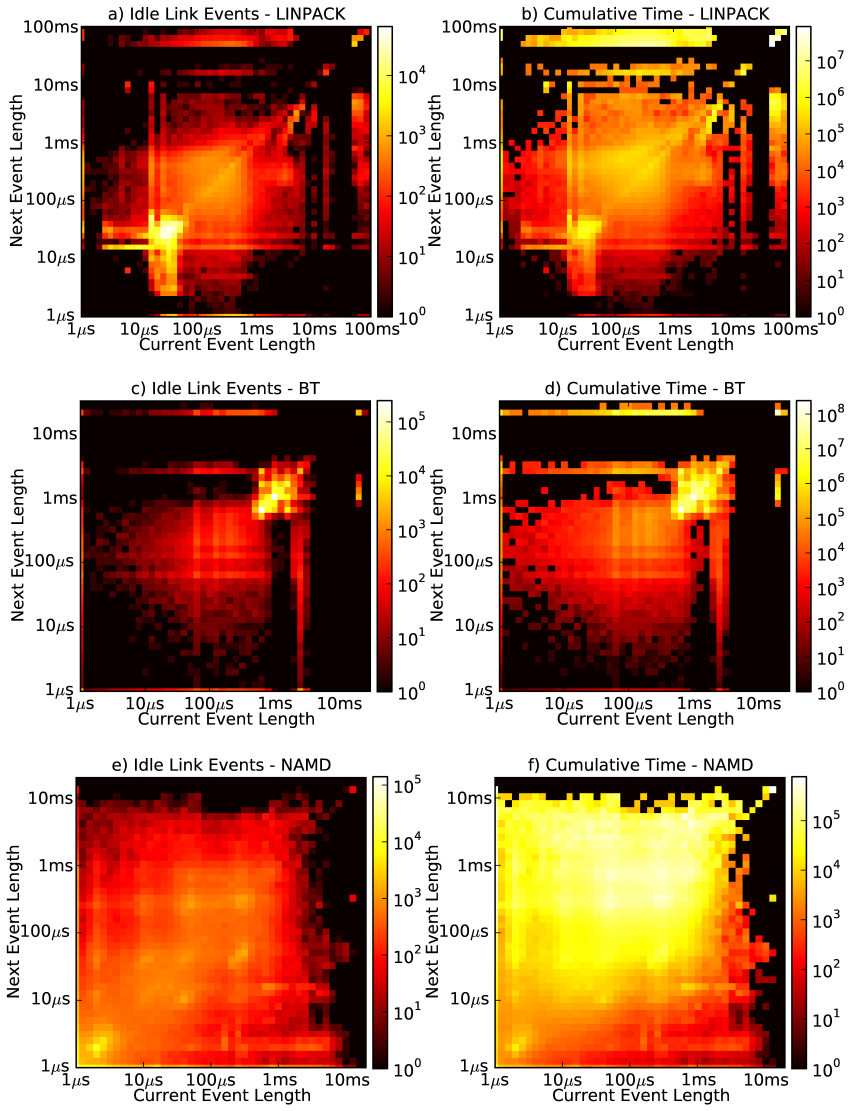



Figure 5.4: Idle Link Event distributions of LINPACK(a,b), BT(c,d), NAMD(e,f) - (a),(c),(e) Heat map of the idle link event duration; (b),(d),(f) Heat map of total idle time (Number events $\times$ duration)

between these applications is clearly seen in Figures 5.4(b) and 5.4(d): the majority of the idle link events of LINPACK are of 10$\mu$s, 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 the set of applications. In Figures 5.4(e) and 5.4(f), it is clear that the application is irregular. In the context of predictability, for any current idle link event of 5.4(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 5.4(a) and 5.4(c), in contrast, LINPACK and BT show reasonable predictability. In the case of LINPACK, for any event of size between 10$\mu$s and 100$\mu$s, 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 5.3 and 5.4, explains the observed performance overheads. Firstly, Figure 5.3 shows the performance overhead of LINPACK remains at about 60% until the LinkOFF threshold is increased to 10$\mu$s, where it drops to about 2% between 10$\mu$s and 100$\mu$s. Comparing that to Figure 5.4(a), the performance overhead has clearly dropped as the LinkOFF threshold crossed the cluster between 10$\mu$s and 100$\mu$s. In other words, if the link remains on for about 100$\mu$s, none of the events in the cluster in Figure 5.4(a) would incur performance overheads. Similarly, in the case of BT, comparing Figures 5.3 and 5.4(c), it is clear 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 5.4(c). Finally, in the case of NAMD, since there are no clusters and the distribution of events is uniform, a gradual decrease in performance overhead can be observed as LinkOFF is increased, falling below 1% above a threshold of 2ms, which correlates with Figure 5.4(e). Note that the clustering observed in Figures 5.4(a) and 5.4(c) are due to repetitive patterns in these applications (as seen in Figure 5.1), 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 5.4(c), no events exist to incur performance delays. Since, for BT, the number of events that exist between 1$\mu$s 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 1$\mu$s is applied to an event of 1ms, the added delay corresponds to 0.1%. For LINPACK, most events are clustered at 10$\mu$s, so if a 1$\mu$s delay is added to them, each delay adds 10% of the idle time, translating to large performance overheads.

## 5.4 PerfBound: Bounding performance overheads in on/off HPC links

The application analysis in Section 5.3.1 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 LinkOFF threshold must be controlled by an algorithm that is dynamic, adaptive and application independent. Finally, from an energy standpoint, it is best to delay the events of longest duration. Based on the above, this section presents PerfBound and PerfBoundRatio. The only parameter to the algorithms is a limit on the performance degradation, which is set to 1% in the evaluation. For the purpose of the exposition, it is assumed that this 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.

**Calculating the the number of events that can be delayed:**

First, for simplicity, a network with only a single hop between two nodes is assumed, i.e., two nodes connected to each other by a single link. 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 duration *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 or hierarchical networks, 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

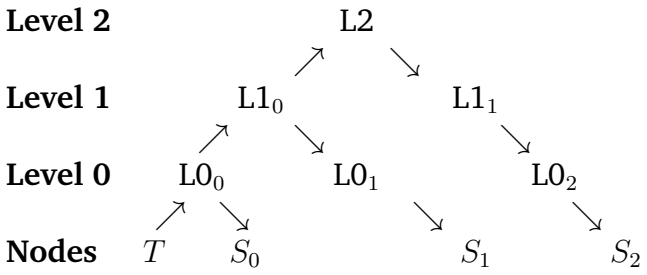



Figure 5.5: Example network topology

| 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 5.1: PerfBoundRatio: Example calculation of local state, when 50%, 40% and 10% of messages reach levels 0, 1 and 2, respectively.

local performance bound of 0.166%.

This is, however, unnecessarily conservative. An application that mainly communicates at Level 0 of the network(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.

The solution for hierarchical networks or multi-hop networks is **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 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.

Here PerfBoundRatio is explained using the example three-level network in Figure 5.5. 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 5.1. 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 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 5.1) for that link is given by

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

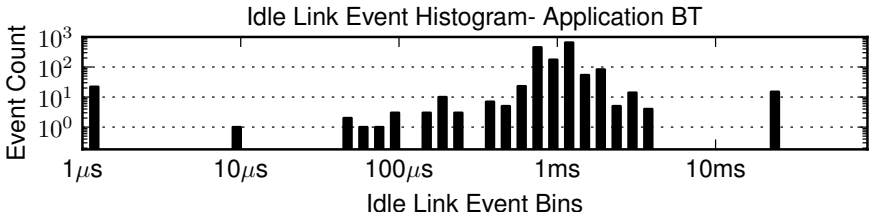



Figure 5.6: Snapshot of an Idle Link Event Histogram

**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 5.6, 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 the experiments presented, 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 5.7 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

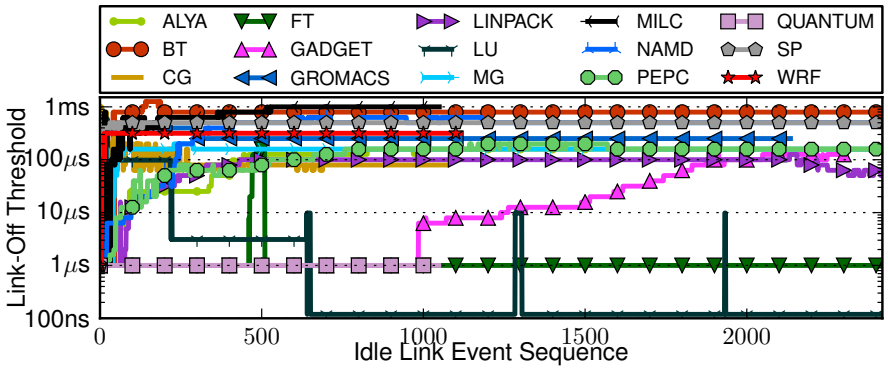



Figure 5.7: LinkOFF threshold convergence over time

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 5.3 where the overhead drops below 1%. Thirdly, for some benchmarks, most clearly LU, LinkOFF threshold is seen to adapt to varying application phases.

Although this section discusses the presented 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 5.7 shows 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 4$\mu$s and 6 hops/message incurred on all 20,000 events in the first refresh cycle, the overhead is $\leq$480ms, which is negligible compared with typical application execution times.

## 5.5 PerfBoundPredict: Prediction over PerfBound for On/Off networks

It is clear, both from the above analysis and from the traces in Figure 5.1, that HPC applications exhibit repetitive behavior. This repetitive behavior translates to periodic and predictable idle link events. This insight is used 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.

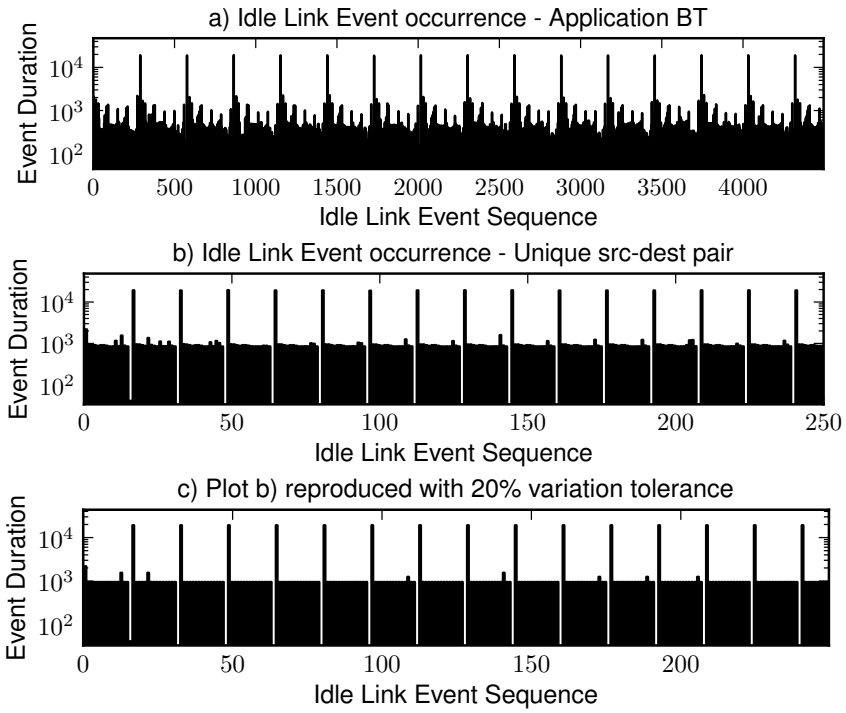



Figure 5.8: Idle link events sequence of occurrence

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.

**PerfBoundPredict** borrows from ATPT [70], in 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 $\pm$20%. This tolerance is proposed based on experiments that showed a steep reduction in the number of unique idle link events up to $\pm$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, **PerfBoundPredict** ignores 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.

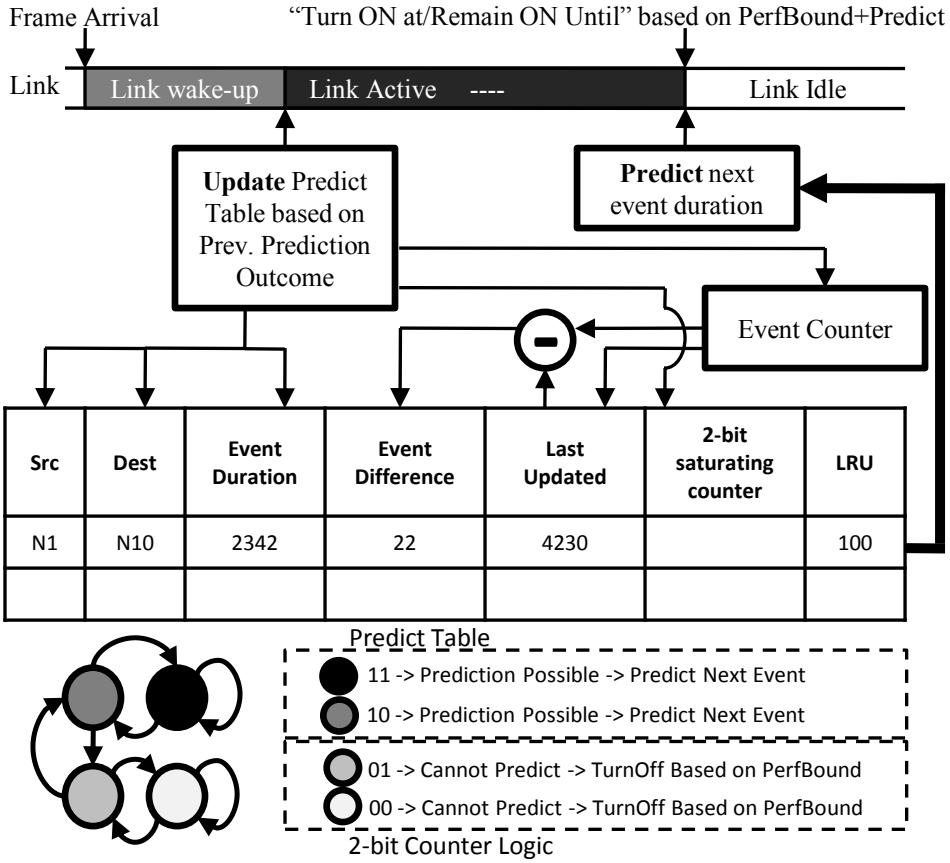



Figure 5.9: Block diagram of PerfBoundPredict

Figure 5.8 shows how classifying idle link events by the src- dest pair for the preceding message helps to identify repetitive behavior. Figure 5.8(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 5.8(a). First, large events occur periodically, but smaller events are more sporadic. Figure 5.8(b) shows only those idle link events that follow a message on a specific src-dest pair. Note that not all src-dest pairs attach to large idle events. Most of the other src-dest pairs examined had no large idle events at all. In fact, the src-dest pair is specifically choses to contain all of the large events visible in Figure 5.8(a). In Figure 5.8(b), all large events are separated from the preceding one by exactly the same number of idle link events. In Figure 5.8(c) accounts for variation in the idle event durations by applying $\pm$ 20% variation tolerance. above, the proposed idle period predictor

Figure 5.9, 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 $\pm$ 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*

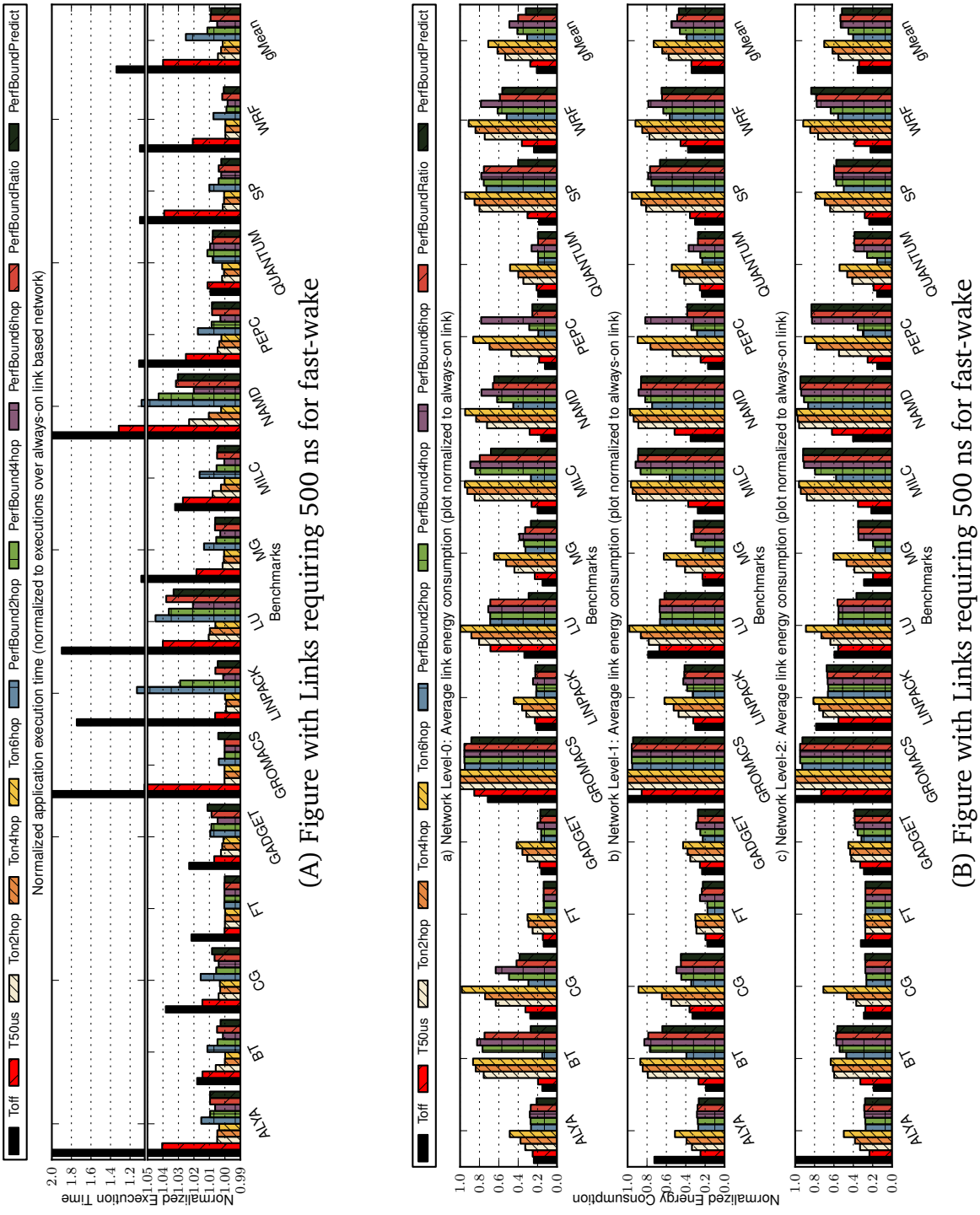



Figure 5.10: Application incurred performance overheads over techniques proposed

*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.

## 5.6 Results

Figure 5.10 (A) 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 5.10 (B) 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 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 5.10 (A), proposed mechanisms PerfBoundRatio and PerfBoundPredict both remain within the assigned performance degradation bound of 1%. In comparison to Toff, the mechanisms presented (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 5.10 (B), on average, Toff and T50us gives the highest link energy savings, followed by PerfBoundPredict and PerfBoundRatio. Note that the difference in energy savings between the 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 the lower prediction accuracy 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 overheads greater than 1%, but their energy consumption is more than twice that of PerfBoundRatio and PerfBoundPredict, at 77% rather than 37%. A sweep is also presented to find the best static LinkOFF threshold. The analysis showed that a value of 500us or larger is needed to reduce the worst-case overhead, for benchmarks considered, 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 5.10 (A) and (B) 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 is because PerfBoundRatio works very well for LINPACK leaving little scope for improvement. 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 Figures 5.4(a) and 5.4(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 5.4). Since LinkOFF threshold for BT is large, turning off the link immediately results in larger power savings. Interestingly, in Figure 5.10 (B) shows that PerfBound2hop performs better than PerfBoundPredict. The reason for this can be seen in Figure 5.10 (A), 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 5.4(c). 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 5.1(c), NAMD does not have patterns to exploit for energy savings. Note that in both cases, performance overhead is still less than 4% with link energy savings up to 70%.

## 5.7 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. This chapter presents three techniques towards the above in the context of on/off links - PerfBound, PerfBoundRatio and PerfBoundPredict. This work showed that significant energy savings can be obtained while performance overhead is bounded. Proposed techniques do not require modifications to the application/compilers nor does it introduce extra traffic into the network. The key novelty of this work is the analysis of link energy from a *performance perspective* - linking application performance degradation with link energy savings. Furthermore, a detailed analysis and insights on HPC application behavior with respect to link idle periods was presented. With performance bounded to acceptable levels this work could enable EEE and thereby link energy proportionality in HPC where performance is crucial.

# Chapter 6
# FastWake: Intermediate Power State

The two previous chapters primarily presented two key ideas. First, HPC application present opportunity for link energy savings but performance can suffer if links are not properly managed. A mechanism, PowerDown Threshold (or LinkOFF Threshold), was proposed that leaves the link on but idle, reduced performance overheads with significant link energy savings. Secondly, the previous chapter presented PerfBound. The motivation for PerfBound comes from the need for a dynamic and application independent PowerDown Threshold mechanism. PerfBound works by automatically managing the links to provide energy savings, but within a bound on performance overheads. This chapter presents an analysis of Fast-Wake, an intermediate power state of EEE in the context of HPC. A summary of this work is presented below.

## 6.1 Summary

The original EEE protocol provides specifications for energy savings in 100Mb, 1Gb and 10Gb links. The success of EEE pushed for current and recent efforts for the standardization of 40Gb, 100Gb and 400Gb backplanes and optical Ethernet, into including energy savings mechanisms from EEE. While incorporating EEE for 40Gb, 100Gb and 400Gb links, the standard introduced an additional sleep state known as **Fast- Wake**. In Fast-Wake mode the link does not turn off all its components, but only a few to trade-off higher energy consumption for a faster wake-up time.

Ethernet protocols, specifically IEEE 802.3bj and 802.3bm, providing standards for 40Gb and 100Gb backplanes and optical Ethernet, respectively, were ratified as recently as March 2015, and IEEE 802.3bs for 400Gb is expected to be ratified in 2017. As with the original EEE protocol, products based on the recent standards, which include Fast-Wake, when deployed for HPC is likely that Fast-Wake will be disabled by default. About 40% of Top500 HPC machines use Ethernet based interconnects every year, that

could potentially have these protocols in their switches; it is imperative to understand the need for Fast-Wake, its performance impact and possible configuration parameters in the context of HPC applications.

This chapter describes a comprehensive analysis of Energy Efficient Ethernet with Fast-Wake using traces from fifteen HPC applications of various domains obtained from production supercomputers. This analysis answers the following important questions regarding the use of Fast-Wake, specifically:

1. How useful is Fast-Wake or an intermediate sleep state for HPC networks, or would a single Deep-Sleep mode be sufficient?
2. How long should the link remain in the intermediate Fast-Wake state before entering Deep-Sleep?
3. What is the reduction in performance overhead and the energy savings on using Fast-Wake, compared with EEE without Fast-Wake?

The answers to these questions could potentially benefit interconnect vendors designing interconnects with EEE and Fast-Wake targeting HPC.

The main contributions of the work presented in this chapter are,

1. A detailed analysis of the Fast-Wake mode over HPC applications to understand its energy saving potential is presented. To this end, this chapter first presents a comparative analysis of the older Deep-Sleep and the newer Fast-Wake.
2. A hybrid approach of using both Deep-Sleep and Fast-Wake and their energy performance trade-offs are discussed. This show how using both the low power modes together offers a better energy to performance trade-off than using either alone.
3. The presented analysis also covers a design space analysis of wake-up timings and other energy consumption levels of Fast-Wake.

## 6.2 Approach to investigating Fast-Wake in HPC

Before presenting the analysis of Fast-Wake, the approach used to investigate Fast-Wake as well as a brief summary of terminology discussed in the Background Chapter

is discussed below.

Figure 6.1 presents the working of Deep-Sleep and Fast-Wake. In Figure 6.1(a), the link remains active during frame transfer, which is followed by a state change that turns off the link, reducing power consumption to 10%. The later frame arrival during the low power state requires a **Full-Wake** ($4.48\,\mu$s) to power up the link to 100% before transmission. Figure 6.1(b) is similar to that of the Figure 6.1(a) except that the link powers down to **Shallow-Sleep**, which reduces power consumption to 60%. The link powers back to 100% faster with Fast-Wake, since it only requires a few hundred nanoseconds to do so.

Figure 6.1(c) and (d) present the working of Stall- Timers [1]. In both, the link remains on at 100% after frame transmission, as opposed to Figure 6.1(a) where the link drops to Deep-Sleep immediately. In Figure 6.1(d), the frame that arrives before the Stall-Timer expires can be transmitted immediately, with no wake-up delay.

Figure 6.2 presents a hybrid approach with both Deep-Sleep and Fast-Wake. Figure 6.2(a), illustrates an example to present a link that actively transmits data, following which remains on at full (or 100%) power until the **Stall-to-Shallow** timer expires. After which the link switches to **Shallow-Sleep** consuming 60% energy until the original Stall-Timer expires. The end of the original Stall-Timer drops the link to Deep-Sleep consuming 10% power. The arrival of a frame during Deep-Sleep triggers a full wake-up, before the link can transmit the frame. Figure 6.2(b), shows the case where the next frame arrives during the Shallow-Sleep period, presenting a case where Fast-Wake is used to quickly power up the link. The above mentioned terminology can be described as follows,

- **Deep-Sleep** is the original Energy Efficient Ethernet (EEE) low power state consumes 10% link energy and requires $3.88$ and $4.48\,\mu$s to which states between active and low-power and back respectively. Deep-Sleep was also previously known as Low Power Mode (LPI).
- **Full-Wake** is the wake-up process or a state transition between low power to active of a link in Deep-Sleep.
- **Fast-Wake** is the newly introduced low power state, with faster wake-up periods, requiring only a few hundred nanoseconds. However the link consumes about

[1] This chapter and the next, uses the term *Stall-Timer* to describe the previously mentioned LinkOFF Threshold or PowerDown Threshold

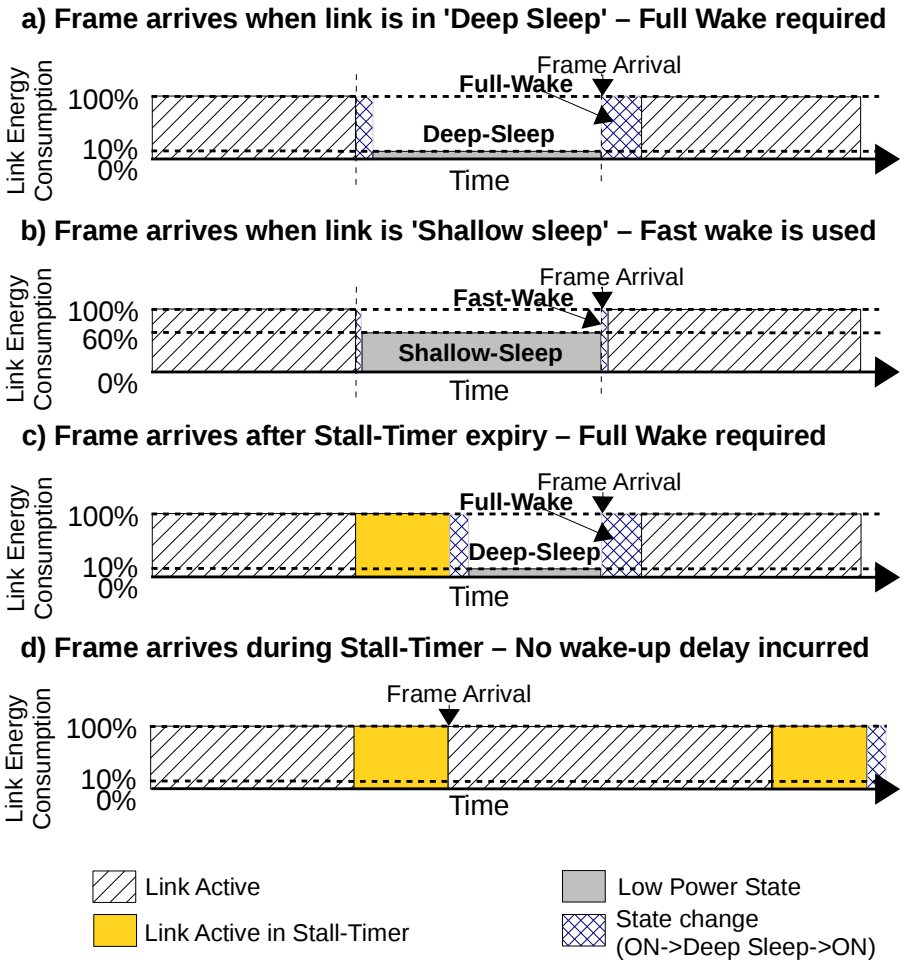



Figure 6.1: Example timeline illustrating various low power and Stall-Timer states of an EEE link

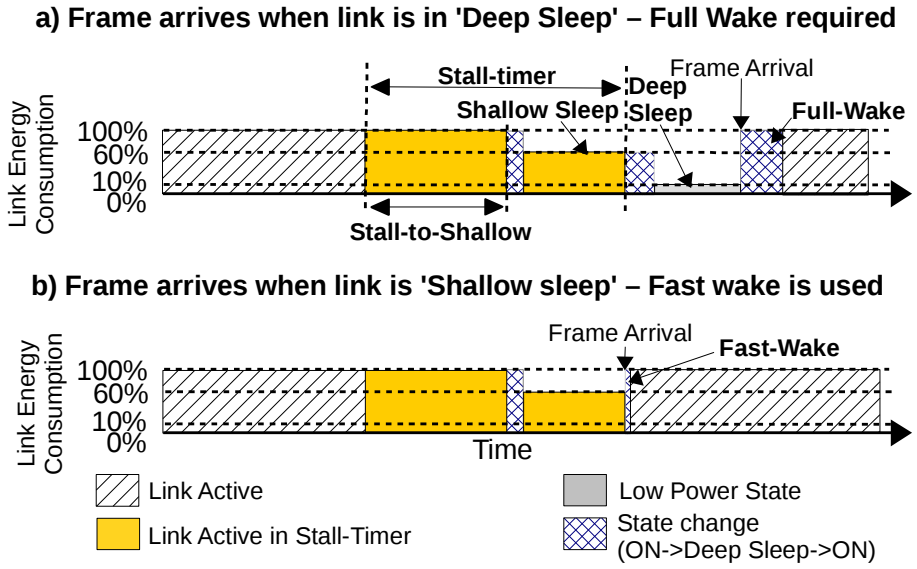



Figure 6.2: Timeline illustrating hybrid Fast-Wake+Deep-Sleep

60% of the active energy in the low power state. This chapter uses the term Fast-Wake to infer quick state transition process from low power (60%) to active (100% power).

- **Shallow-Sleep** is when the link only requires a Fast-Wake to turn active, i.e., the link is in low power mode consuming 60% power as opposed to 10% of the Deep-Sleep.

- **Stall-to-Shallow** is used with the hybrid approach as shown in Figure 6.2. Stall-to-Shallow is the duration link remains on but idle, before entering Shallow-Sleep to consume 60% power.

- **Stall-Timer** as discussed in previous chapters, leaves the link on but idle for a duration, after which the link enters Deep-Sleep. In the hybrid approach discussed in Figure 6.2, the only difference is that the Stall-Timer contains an intermediate power state, where the link transitions to Shallow-Sleep. Note that for this reason, the Stall-Timer must be larger than or equal to Shallow-Sleep.

With respect to the above, the approach towards investigating Fast-Wake is as follows. First a comparison between the individual power and performance characteristics of Fast-Wake and Deep-Sleep is drawn. Based on this analysis, a case for a combined/hybrid approach that uses both Fast-Wake and Deep-Sleep is explored. Further, the following parameters are investigated: 1. Energy consumption during Fast-Wake mode and 2. Link wake-up duration for Fast-Wake. The relationship and trade-offs between optimal power and performance and careful adjustment of respective Stall-timers is also discussed in the following sections. The experiments were performed with a constant Stall-Timer/Stall-to-Shallow across all links of the network, since investigating optimal Stall-Timers for every link in the network is not feasible for the sensitivity analysis presented below.

### 6.2.1 Methodology specifics

Note that the experimental methodology used in this chapter is identical to the previous chapter 5. The experiments were done over the Dimemas simulator. The network specifics, experimental configuration and applications used are the same as discussed in the previous chapter 5. The key addition is that this work modifies the simulator to add the above mentioned mechanisms (Fast-Wake and two different Stall-Timers along slide Deep-Sleep) into the links.

## 6.3 Deep-Sleep and Fast-Wake vs Stall-time

Figure 6.3(A) presents an analysis of the performance and energy consumption using only the Deep-Sleep mechanism, as a function of its corresponding Stall-Timer. Specifically, Figure 6.3(A)(i), shows the normalized execution time for each application referred to in the previous section, relative to the same with an always-on interconnect. Figures 6.3(A)(ii), (iii) and (iv) are similar in that they show the average energy consumption of links at each level of the network, where level-0 is connected to nodes and level-2 is the highest.

Varying the Stall-Timers of Fast-Wake and Deep-Sleep presents interesting trade-offs between power and performance, as seen in Figures 6.3(A) and (B), respectively. Figure 6.3(A)(i) shows that, for all applications, the performance overhead reduces as the Stall-Time increases. This is because, as the Stall-Timer increases to infinity, the

network tends towards being always-on. This means that the Stall-Timer would eventually be larger than all idle periods between frame arrivals, and hence no subsequent frame arrival would ever incur a wake-up delay, rendering zero performance overhead. This however increases the energy consumption in all levels of the network, since an always-on network consumes 100% power relative to its baseline. Figure 6.3(A) shows that lower values of the Stall-Timer (0 to $100\,\mu$s) have relatively large performance overheads (greater than 10%) but high energy savings (70% on average). For higher values of the Stall-Timer ($100\,\mu$s to $1$ ms), the performance overhead drops to below 1%, while the energy savings are still about 50% to 60% (average), for all levels of the network.

Figure 6.3(B) shows a similar analysis, but for Fast-Wake. Here, the Stall-Timer represents Stall-to-Shallow, where the link drops from 100% power to 60% power, and requires a Fast-Wake to wake-up. Figure 6.3(B)(i) is similar to Figure 6.3(A)(i) in that performance overheads decrease with an increase in the Stall-Timer. Figure 6.3(B)(i), however, shows the performance overhead drops below 1% (say) at a much lower value of the Stall-Timer compared with Figure 6.3(A)(i). Correspondingly the energy consumption as shown in Figure 6.3(B) begins at 60% and increases towards 100%. The link energy consumption of applications in Figures 6.3(B)(ii), (iii) and (iv) start at 60% since the idle power of Fast-Wake is 60% of that of an active link.

On comparing the two mechanisms for application NAMD, for example, it can be observed that for a performance overhead of less than 1% with Deep-Sleep (Figure 6.3(A)), a Stall-Timer larger than $1$ ms is required, for which the energy consumption is about 85–90% (for Level-0 links). With Fast-Wake, as seen in Figure 6.3(B), application NAMD falls below 1% at about $100\,\mu$s, where the energy consumption is about 65–70%, for the same Level-0 links. Similar behavior can be seen for higher network level links. Clearly for NAMD, Fast-Wake is the better mechanism. In contrast to the above, application PEPC requires a Stall-Timer of about $100\,\mu$s for an overhead of less than 1%, as shown in Figure 6.3(A), but it only consumes 20% link energy. In Figure 6.3(B) however, that PEPC requires a Stall-Timer of less than $10\,\mu$s for overheads less than 1%, but its energy consumption is now above 60%. In the case of PEPC therefore, Deep-Sleep is the better mechanism for energy savings and low performance overhead.

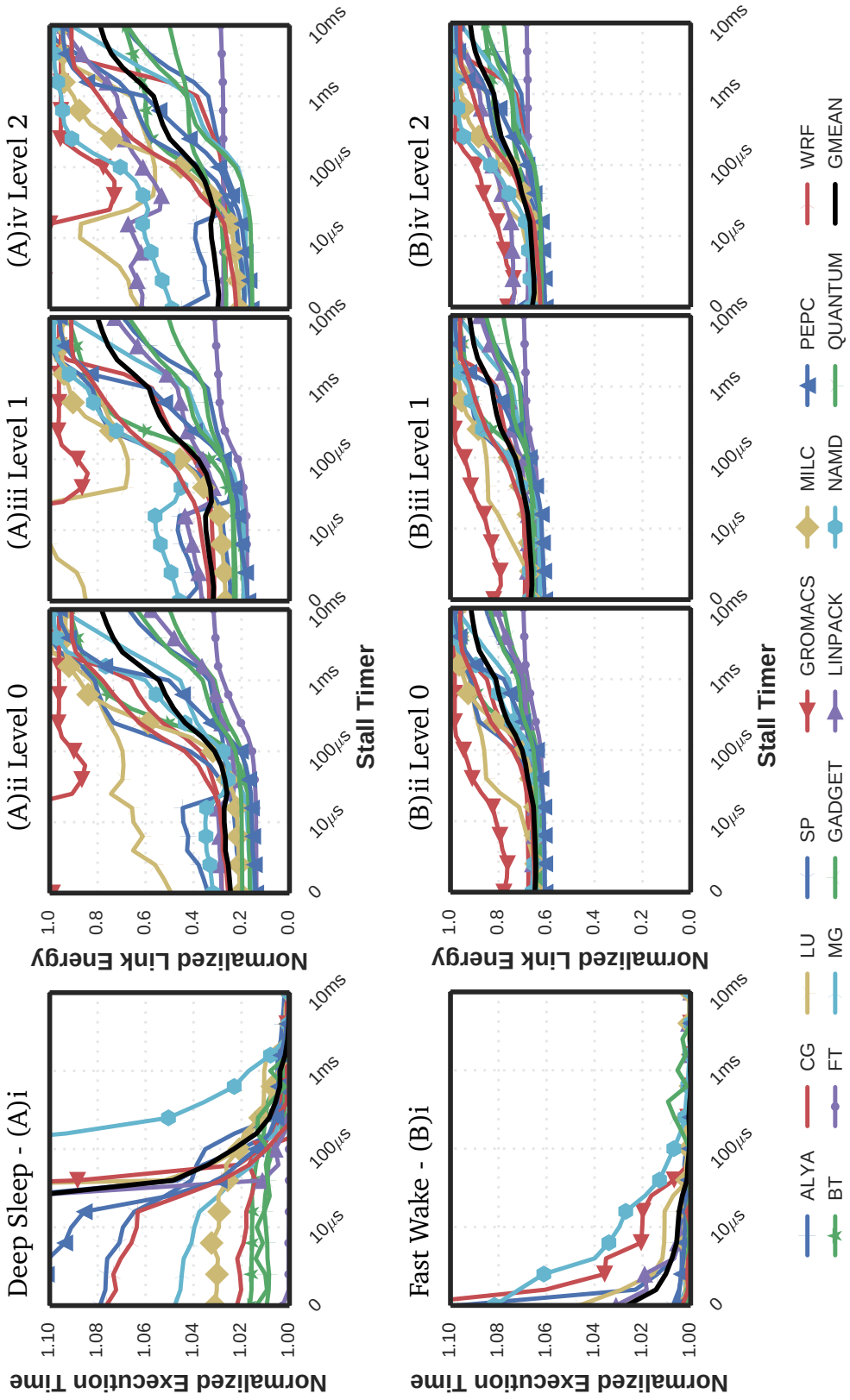



Figure 6.3: Power and Performance of using Deep-Sleep and Fast-Wake as a function of their Stall-Timers

Figures 6.3(A) and (B) show that, for both Deep-Sleep and Fast-Wake, adjusting the Stall-Timer varies power and performance, through an increase in energy savings at the expense of performance, or vice-versa. It is clear that since Fast-Wake powers back the link at a faster rate, relatively small Stall-Timers are required for low performance overheads. However, with Deep-Sleep, larger Stall-Timers are required to reduce performance overheads. It is interesting to see that for a fixed performance overhead point (1% say), some applications such as NAMD, GROMACS, SP and MILC tend to have better energy savings with Fast-Wake while others such as GADGET, FT, QUANTUM and PEPC are better with Deep-Sleep. Specifically, applications that have low network utilization and that are not latency sensitive are better with Deep-Sleep compared to Fast-Wake. These applications do not require the faster wake-up time, so they can benefit from Deep-Sleep's lower energy consumption.

## 6.4 Pareto optimal analysis of Fast-Wake and Deep-Sleep

The analysis with Figures 6.3(A) and (B) clearly shows the need for a combined approach. As shown in Figure 6.2 using both Deep-Sleep and Fast-Wake would require adjusting the original Stall-Timers, which expires to Deep-Sleep and Stall-to-Shallow, which expires to Shallow-Sleep. Together, these two Stall-Timers create a 2-dimensional search space.

An exhaustive search of this 2D search space is presented in Figure 6.4. The $x$-axis is the performance overhead, up to 5%, as larger overheads would be unacceptable. The $y$-axis is the energy consumption of all links in Level-0 of the network (the behavior of higher network layers is discussed in the next section). The scatter points labeled "Deep Sleep + Fast Wake + Stall timer sweep" show all combinations found by varying both Stall-Timer values. Specifically, Stall-Timers were varied in multiples of 10, example 10 ns, 100ns and so on. To explore all combinations, for every Stall-Timer *'X'*, Stall-to-Shallow was varied between 10 ns to *'X'*. The curve labeled "Deep Sleep + Fast Wake + Stall timer - Pareto Optimal" is the Pareto-optimal curve derived from only the points that are Pareto-optimal (which is roughly speaking only the points with lowest energy consumption for a given performance overhead). The other curve, labeled "Deep Sleep + Stall Timer", presents, for comparison, the power and performance obtained from varying Stall-Timer using only the Deep-Sleep mode. This approach is compared with Deep-Sleep because it represents the older approach to energy savings available in EEE switches before the introduction of Fast-Wake. This figure therefore presents the potential benefits of using both Fast-Wake and Deep-Sleep.

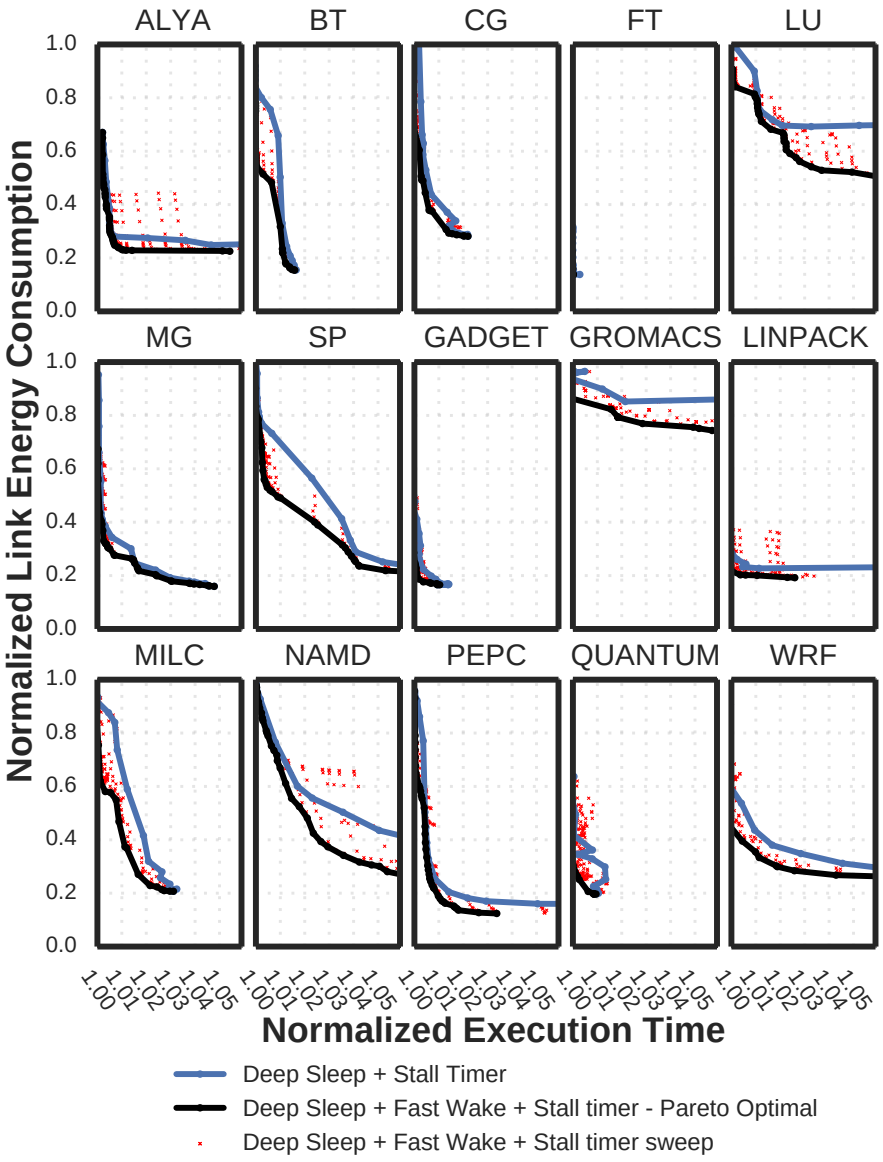



Figure 6.4: Pareto-optimal analysis of average (level-0) link energy and performance using both Deep-Sleep and Fast-Wake along with their corresponding Stall-Timers compared to only using Deep-Sleep and its Stall-Timer

Figure 6.4 makes a strong case for using a combination of Deep-Sleep and Fast-Wake, since the Pareto-optimal curve is clearly below or overlapping the Deep-Sleep curve for all applications. As seen, this technique is less useful for applications that are not latency sensitive as in the case of FT, QUANTUM and GADGET, where simply using Deep-Sleep is sufficient. However, with applications BT, CG, LU, SP, GROMACS, MILC, NAMD, PEPC and WRF, a combined approach clearly improves energy and performance overhead. With BT at 0.5% performance overhead has 45% energy savings with the Pareto optimal curve compared to only 20% with Deep-Sleep. Applications such as SP and MILC benefit most from the combined approach at low performance overheads, where a further reduction of performance overhead is possible at the same or similar energy savings. With GROMACS and WRF 5%–20% energy savings are obtained compared with all points of Deep-Sleep.

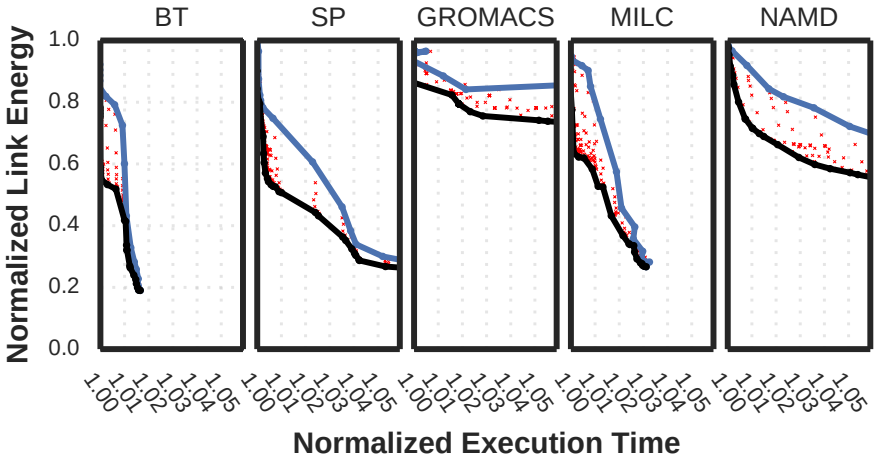
Here, it is to be noted that a Pareto-optimal curve of the hybrid Fast-Wake and Deep-Sleep is compared to the Stall-Timer sweep of Deep-Sleep. This is to say that the scatter points in Figure 6.4 clearly show that for many combinations of Stall-to-Shallow and Stall-Timer, higher performance overhead or much lower energy savings are obtained. Specifically for ALYA, choosing an incorrect combination of Stall-Timers values is highly likely since most other values perform worse compared to Deep-Sleep. Dynamically and automatically choosing these Stall-Timers for Fast-Wake and Deep-Sleep during runtime could be an interesting problem for further research.

## 6.5 Fast-Wake on higher level links - L1, L2

In Figure 6.5 and all following figures a subset of the applications seen in Figure 6.4 is discussed. Specifically, applications BT, SP, GROMACS, MILC and NAMD are chosen as representatives to further the following discussions. These applications specifically are discussed since other applications either behave similarly, or, in the case of FT, GADGET and QUANTUM, do not benefit from Fast-Wake.

Figure 6.5 is the same as Figure 6.4, except that it shows the normalized energy consumption of network Level-1 for Figure 6.5(A) and network Level-2 (highest) for Figure 6.5(B). Comparing Figures 6.5(A) and (B) with Figure 6.4 shows that the benefits of Fast-Wake plus Deep-Sleep at the higher levels of the network are similar to those at Level-0. NAMD particularly has increasingly better trade-offs with higher link levels compared to Deep-Sleep. Since NAMD utilizes its network sporadically rather than in bursts, higher benefits from Fast-Wake can be seen. At higher levels of the network,

(A) Figure 6.4 presented for links in level-1




(B) Figure 6.4 presented for links in level-2

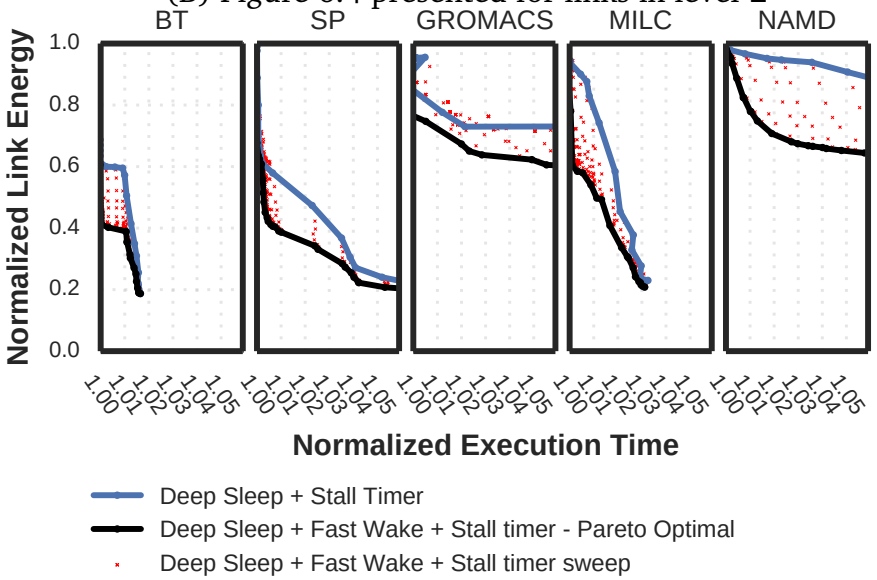



Figure 6.5: Fast-Wake analysis on upper layer network links

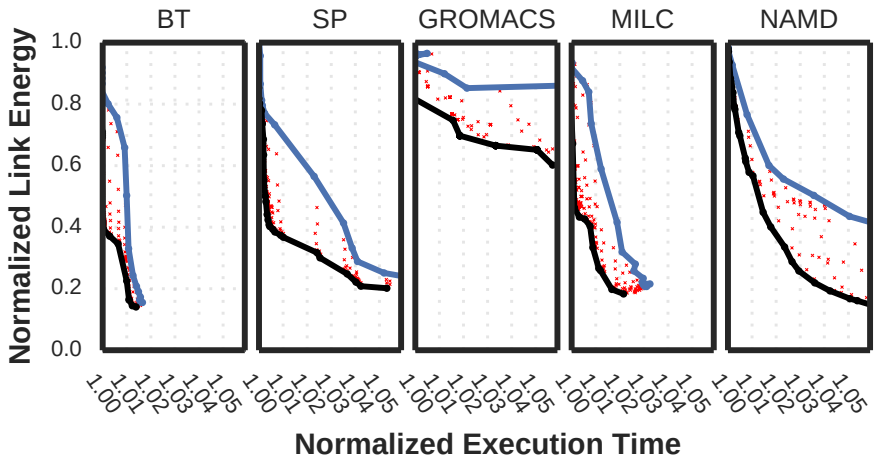
where the traffic is sporadic rather than uniform, large Stall-Timers for Deep-Sleep are required to maintain acceptable overheads. Having an intermediate state between these large Stall-Timers produces the better energy trade-off.

## 6.6 Fast-Wake energy ratio - 40/60/80% and Fast-Wake timing analysis - 500 ns

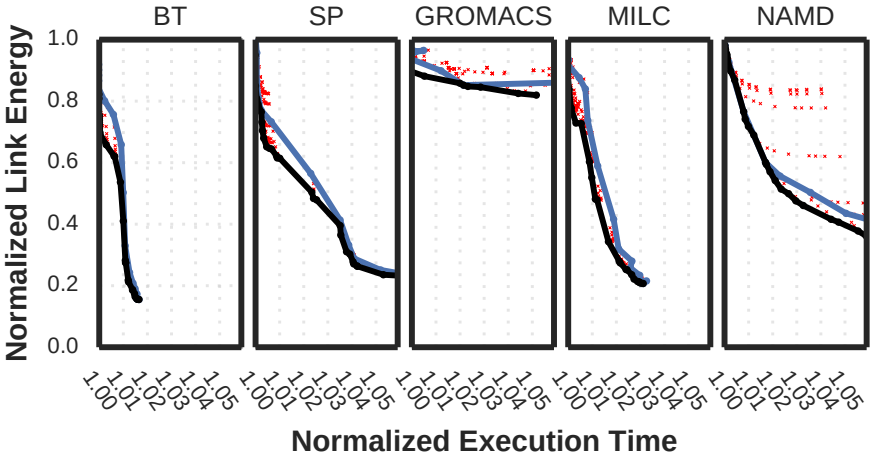
This section investigates how the conclusions in this chapter would change, when varying the power consumption during Shallow-Sleep and its wake-up time. The power consumption during Shallow-Sleep has been shown to be 60% compared to when the link is active [58], but the precise value cannot be known for sure until products arrive on the market. Figures 6.6(A) and (B), are similar to Figure 6.4 in that they have a Fast-Wake time of $250$ ns, and they show the energy and performance trade-offs for Level-0 links. However, unlike Figure 6.4, Figures 6.6(A) and (B) model the power during Fast-Wake to be 40% and 80%, respectively, compared to when the link is active. In Figure 6.6(A), where Shallow-Sleep only consumes 40% energy, a higher energy to performance trade-offs can be observed compared to Shallow-Sleep with 60% shown in Figure 6.4. Note that for a given performance overhead of 1%, the link energy savings for GROMACS increases from 15% (with 60% Shallow-Sleep as seen in Figure 6.4) to about 25%. These benefits indicate that a number of points in the Pareto-optimal curve have Stall-Timer values configured so that the link spends significant time in Shallow-Sleep. Consuming lower energy during Fast-Wake's Shallow-Sleep therefore contributes to significantly higher link energy savings. In Figure 6.6(B), however, where the power consumption in Shallow-Sleep is 80%, the curve is similar to Deep-Sleep only.

In Figure 6.6(C) the Shallow-Sleep energy consumption is modeled back to 60%, but the Fast-Wake time is modeled to be $500$ ns, instead of $250$ ns (used in all previous figures). Fast-Wake with $1\,\mu$s and $2\,\mu$s was also investigated. As expected, the results with other values of Fast-Wake time show that the gap between the two curves of the Pareto-optimality study, reduces, as the Fast-Wake time increases. Since Deep-Sleep requires $4.48\,\mu$s, increasing Fast-Wake time towards $4.48\,\mu$s, tends to make Fast-Wake closer to that of Deep-Sleep. However, between $250$ ns and $500$ ns, only a small reduction in energy savings or performance compared to Figure 6.4 can be seen. It is conceivable that a higher wake-up time for Fast-Wake could allow for more components to be turned off, further reducing its energy consumption.

(A) Figure 6.4 with Link at 40% energy in shallow-sleep




(B) Figure 6.4 with Link at 80% energy in shallow-sleep




(C) Figure 6.4 with Links requiring 500 ns for fast-wake

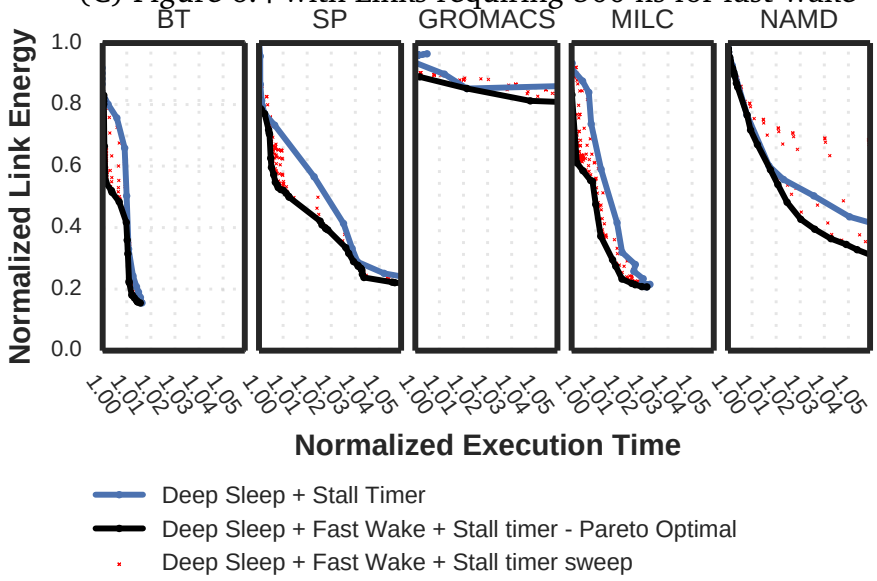



Figure 6.6: Fast-Wake analysis on link energy consumption and link wake-up time

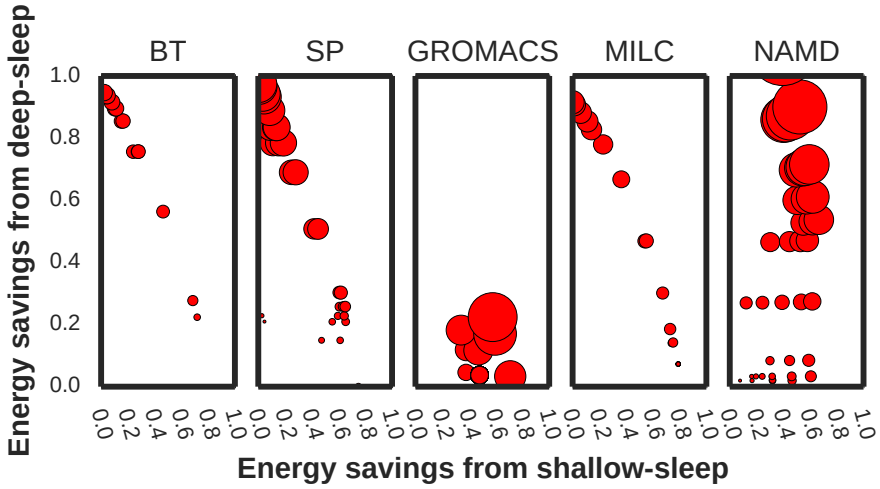



Figure 6.7: Contribution of energy savings with Pareto-optimal points from Figure 6.4 between Shallow-Sleep and Deep-Sleep

## 6.7 Ratio of energy savings between Fast-Wake and Deep-Sleep

Figure 6.7, shows the proportion of energy savings from Fast-Wake and Deep-Sleep respectively. The figure shows a scatter plot of points obtained from the Pareto-optimal curve in Figure 6.4 for each corresponding application. The $x$-axis shows the proportion of energy savings obtained while the application is in Shallow-Sleep and the $y$-axis shows the proportion in Deep-Sleep. The size of the scatter point is proportional to its performance overhead, where larger points represent a higher overhead.

Figure 6.7, application NAMD, for example, shows that the higher the energy savings from Deep-Sleep, the higher the performance overhead. The size of the scatter points reduces with the reduction in overhead, as the contribution of energy savings from Deep-Sleep reduces. Reduced energy savings from Deep-Sleep also means that the total interconnect energy increases, due to higher Stall-Timer values. The same behavior can be seen with Shallow-Sleep where overhead correspondingly reduces with a reduction in energy savings from Shallow-Sleep. It is interesting to see that Stall-Timer values correspond to points in Figure 6.7. Choosing the correct Stall-Timers is critical to maximizing energy savings and reducing overheads.

## 6.8 Conclusions

This chapter presents an analysis of Fast-Wake for HPC workloads, examining its potential for energy savings and possible performance overheads. This analysis showed that using both an hybrid approach with both Fast-Wake and Deep-Sleep offers higher energy savings in links with lower average performance overhead than their use as standalone mechanisms. This work also showed that for optimal energy and performance, the corresponding Stall-Timers of Fast-Wake and Deep-Sleep must be chosen carefully. Further, higher network layers are shown to benefit more from the use of the hybrid approach compared to edge links. With the ratification of Fast-Wake in March 2014 and 2015 for 40/100Gb Ethernet, and the ongoing standardization effort for 400Gb Ethernet, this analysis could could help HPC interconnect vendors build Ethernet solutions that integrate Fast-Wake.

# Chapter 7
# Double PerfBound: PerfBound on Fast-Wake

The previous chapter presented an analysis of Fast-Wake, this chapter presents DoublePerfBound - an approach that uses PerfBound for performance-aware energy savings with Fast-Wake.

## 7.1 Summary

In extending EEE beyond 10 Gbit/s, an additional sleep state, known as Fast-Wake, was added, in order to trade higher energy consumption for a faster wake-up time. The previous chapter showed that a combination of Fast-Wake and the original Deep-Sleep together delivered a better performance–energy trade-off than either on its own, with up to 20% higher energy savings at the same performance overhead. The key to these higher energy savings however is in finding two different Stall-Timers on or below the Pareto-optimal curve. This is hard since the optimal Stall-Timers vary for every application. The challenge here is further compounded by the fact that with Fast-Wake, now two Stall-Timers require dynamic adjustment over a two-dimensional search space, while the original PerfBound with Deep-Sleep only required one.

This chapter introduces **DoublePerfBound**, a self- contained technique to manage Fast-Wake + Deep-Sleep to minimize energy subject to a bound on the performance degradation. The algorithm dynamically switches between active, Fast-Wake and Deep-Sleep modes, adapting to application characteristics. The key to achieving low performance overhead, as discussed in Chapter 5 that discusses PerfBound, is to ensure that not too many frames incur a link wake-up delay. Just as in PerfBound, the approach to lower performance overhead here is to use the Stall-Timer, in keeping the link on for a period after transmitting a frame, so that any subsequent frames can be transmitted

without incurring a wake-up delay. With a single sleep mode, as in PerfBound, changing the Stall-Timer either increases execution time and decreases energy consumption, or vice- versa. This gives a simple trade-off between performance and energy. In this case, for a given performance overhead bound, the Stall-Timer with the highest energy savings is the smallest value that respects the bound. With two sleep modes, each sleep mode has its own independent Stall-Timer, and varying these values gives a two-dimensional search space. Not only is this much larger than in the one-dimensional case, but a given performance overhead now translates to a curve of Stall-Timer pairs, each with the same performance overhead, but a different energy consumption. Finding the optimal solution can only be done based on an estimate of the energy consumption. This is a much more complex problem, but as we show, it can be solved using the relatively inexpensive DoublePerfBound algorithm.

The contributions and novelty of this chapter are as follows:

1. This work introduces DoublePerfBound, a novel algorithm that manages the Stall-Timers of both Fast-Wake and Deep-Sleep, maximising energy consumption subject to a performance overhead bound. DoublePerfBound is self- contained and does not require any changes to applications and only uses information available at the NICs and switches. Furthermore, it does not introduce additional communication messages and is compatible with multi-hop networks. The evaluation of DoublePerfBound shows that heuristic finds Pareto- optimal performance–energy points and saves up to 70% link energy. This work also compares DoublePerfBound with the previous proposal PerfBound 5, that controls a single Stall-Timer, and shows, on average, 10% better EDP.

2. This work presents a low-complexity feed-forward control mechanism that improves the accuracy of PerfBound and in extension DoublePerfBound in finding Stall-Timers closer the given performance overhead bound.

## 7.2 Motivation

DoublePerfBound automatically adjusts two stall-timers subject to a bound on performance overhead. Analysis of stall-timers for multiple link sleep states discussed in the previous chapter shows its benefits. The previous chapter however does not provide a mechanism for automatically controlling two stall- timers such that Pareto-optimal energy–performance can be achieved.

This section briefly discusses relevant information from the previous work presented in Chapters 4, 5 and 6 to show how DoublePerfBound extends from the above results. The methodology and experimental setup of this chapter is identical to the previous chapter except for DoublePerfBound modeled within EEE enabled links that control the stall-timers, as opposed to the static control of the previous chapter.

From the previous chapters, the following key attributes of HPC application behavior on on/off based links is clear. First, increase in Stall-Timer values causes a decrease in execution time and increase in link energy. This is obvious since Stall-Timer causes links to remain on for longer durations after turning idle, and hence they delay fewer messages and thereby reduce overheads, but increase link energy. However, for a given Stall-Timer, there are large differences in trade-offs for each application with regard to energy and performance. Second, as presented in chapter 6, some applications benefit more from either using Fast-Wake or only Deep-Sleep, i.e., they have lower performance overhead and higher energy savings with only either one of them. Analysis presented in chapter 6 clearly makes a case for using a combined approach, in using both Deep-Sleep and Fast-Wake.

Figure 7.1 shows the energy–execution time trade-off for the hybrid scheme. Figure 7.1 is obtained from the results presented in the previous chapter. As in the previous chapter, the $x$-axis is execution time and the $y$-axis is average link energy (both normalized to non-EEE). The blue line is for Deep-Sleep only, and is a one- dimensional sweep of the single Stall- Timer. The scatter plot is an exhaustive search for the hybrid Fast- Wake + Deep-Sleep scheme, covering the two-dimensional space given by the two Stall- Timers. The dark line connects the Pareto-optimal points from the scatter plot, which are roughly the points with lowest energy consumption for a given performance overhead.

Figure 7.1 presents results for three applications as motivation but results of all applications compared to that of the proposed technique DoublePerfBound is presented in Section 7.5. Figure 7.1 can be used to identify the following conclusions of the previous work, first, applications benefit from 5% to 20% with the use of the hybrid Fast-Wake + Deep-Sleep as opposed to only using Deep-Sleep. Second, this benefit is only possible if the right stall-timers are chosen, since, as seen in ALYA and SP, many combinations of stall-timers can make the performance and energy significantly worse than even Deep-Sleep. A bad choice of stall-timer for ALYA could result in up to 20% higher energy consumption compared to the Pareto-optimal curve, and 15% worse than Deep-Sleep.

In summary, while significant energy savings are available from using Deep- Sleep

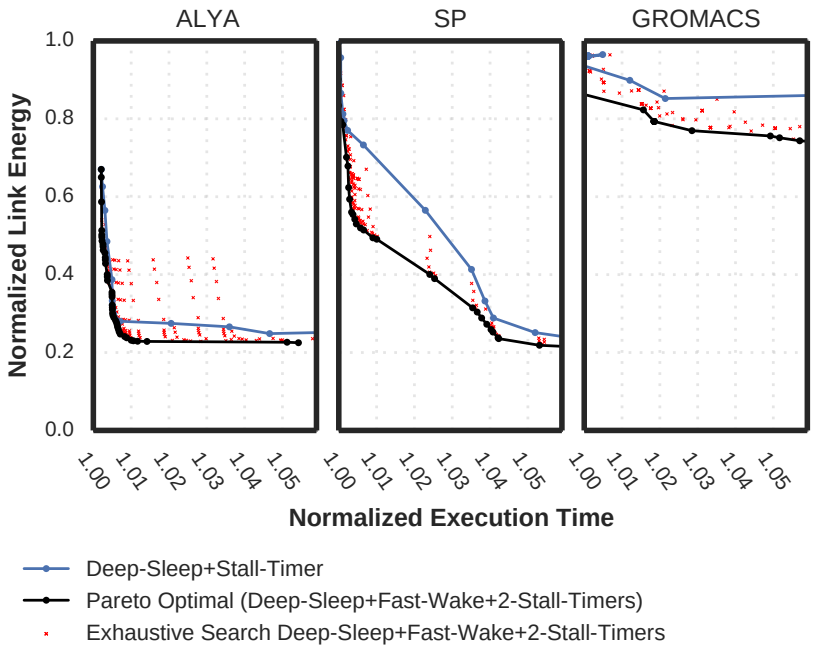



Figure 7.1: Pareto-optimal analysis of performance/energy using both Deep-Sleep and Fast-Wake vs. only using Deep-Sleep with Stall-Timers

and Fast-Wake, Stall-Timers must be carefully chosen. This is not obvious since optimal values are application-dependent, and cover a two- dimensional space that includes points that are far from Pareto optimal. Moreover, for a given performance overhead there are many choices of stall- timer pairs, each consuming different amounts of energy. Hence, finding the Pareto–optimal solution can only be done based on an estimate of the energy consumption. The following sections show how DoublePerfBound automatically finds optimal stall-timer pairs for a given performance overhead bound such that energy consumption is at its the lowest.

## 7.3 Relating link wake-ups to the Stall-Timer

Section 7.4 will discuss the DoublePerfBound scheme that adjusts Stall-Timers to minimise energy subject to a bound on the performance overhead. Doing this requires that the performance overhead is actually known at runtime, which is in general difficult.

PerfBound, as discussed in Chapter 5, however, has shown, that the performance overhead introduced by link wake-ups is approximately proportional to the number of times that the link is switched off then back on. This is an approximation that does not track dependency chains among nodes, which would determine whether the introduced delays are on the critical path or not. Tracking dependency chains would require either that the user or compiler modifies or annotates the application, or that additional messages are sent at run time and monitored by the NICs and switches. Both approaches add complexity, and as previously argued, such a proposal would be unlikely to be adopted in practice. This work follows along the same lines as PerfBound because, on balance, as seen in the Section 7.5, it gives the right compromise.

Below is a brief summery of relevant information from PerfBound. The previous PerfBound algorithm was proposed adjust a single Stall-Timer such that maximum energy savings could be achieved, while bounding the performance overhead. PerfBound first relates the given performance bound to the number of messages that can be permitted to incur wake-up delays. Secondly, it adjusts a single Stall-Timer value using a histogram in order to delay this target number of messages.

For DoublePerfBound, the same calculation from PerfBound for the maximum number of messages to delay is used. As in PerfBound, the overhead is assumed to come only from delayed messages, so the maximum number of them that can be tolerated, within a bound of $B$%, in a period of length $X$ is $lX/T_{\mathrm{W}}$, where $T_{\mathrm{W}}$ is the wakeup delay and $l = 0.01B$ is the percentage overhead as a fraction. As $X$ increases over time, the number of messages that can be delayed increases in proportion.

For multi-hop networks, the extension to PerfBound, the PerfBoundRatio essentially calculates a local performance bound to apply at the link, replacing the original (global) performance bound. The local bound depends only on information already available locally at the switch. A global performance bound of 1% could therefore be translated to a local bound of say $B = 0.3$% at a particular link.

## 7.4 DoublePerfBound: Fast-Wake + PerfBound

This section introduces DoublePerfBound, a self-contained technique to manage hybrid Fast-Wake + Deep-Sleep, optimizing energy subject to a bound on the performance degradation. The motivation for DoublePerfBound, as outlined in Section 7.2, comes from benefits in using an hybrid Deep-Sleep + Fast-Wake approach and their need for

optimal Stall- Timers. This chapter presents and describes DoublePerfBound from its behavior at a single link however the multi-hop network extension of PerfBound, its PerfBoundRatio is directly applicable. The results section present results with ideas from PerfBoundRatio integrated even though the following discussion focuses on a single link.

### 7.4.1 Single Stall-Timer case

The previously mentioned PerfBound heuristic discusses how a target number of delayed messages can be mapped to a single Stall-Timer value. This is done using a histogram that records for each link, the lengths of their idle periods[1] A snapshot of this idle period histogram for application BT is shown in Figure 7.2(A). The histogram is updated after every message or frame on a link, by inserting the length of the previous idle period. It therefore represents the distribution of inter-message idle periods.

The Stall-Timer is adjusted to selectively delay only the longest idle periods, in order to get the greatest energy savings for a fixed number of wakeups. Its value is found by searching from the right-hand side of the histogram, starting from the longest idle period, until the target number of messages has been counted. If the histogram has been collected for a total time $X$, the link's local performance bound is $l$, and the time to wake from Deep-Sleep is $T_{\text{W}}$, then Section 7.3 gave the number of messages to delay as $N_{\text{D}} = lX/T_{\text{W}}$. The Stall-Timer is the midpoint of the smallest bin with at most $N_{\text{D}}$ messages in all bins to the right. As indicated in Figure 7.2(A), only intervals longer than the Stall- Timer, which appear on the right-hand side of the histogram, benefit from energy savings and incur wake-up overheads.

### 7.4.2 Extending PerfBound to hybrid Fast-Wake + Deep-Sleep

In the case of hybrid Fast-Wake + Deep-Sleep, the *number* of messages to delay is not fixed, as it depends on how many messages cause a wake from Fast-Wake, each incurring a delay of $T_{\text{FW}}$, and how many wake from Deep-Sleep, each incurring a delay of $T_{\text{DS}}$. The first step should therefore identify the total of the wakeup times, $T_{\text{D}} = lX$, rather than the number of messages to delay, $N_{\text{D}}$.

[1] After active transmission of all frames in the frame- buffer, links remain idle awaiting new frames for transmission. This duration of inactivity is termed an "idle link period".

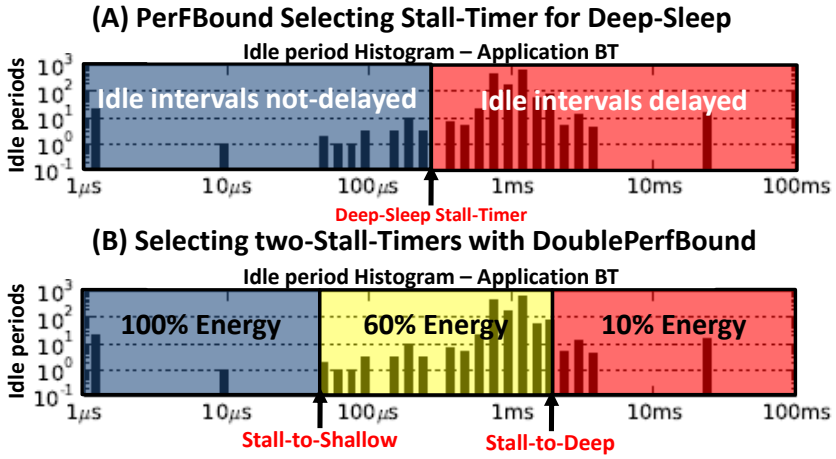



Figure 7.2: Idle period histograms for Application BT, illustrating effect of Stall-Timer placement on link delay and energy savings

Figure 7.2(B) shows how the idle period histogram is used by DoublePerfBound. Instead of a single Stall-Timer, the algorithm must determine two, Stall-to-Shallow and Stall-to-Deep. Similarly to Figure 7.2(A), the left of the histogram has short idle intervals, of length less than Stall-to-Shallow, during which the link remains active, at 100% power, and there is no wake-up penalty. In the middle of the histogram are intervals larger than Stall-to-Shallow but less than Stall-to-Deep, during which the link (eventually) enters Shallow-Sleep, consuming 60% power but later incurring a Fast-Wake penalty of $T_{\text{FW}}$. The right of the histogram has intervals longer than Stall-to-Deep, during which the link enters Deep-Sleep, consuming just 10% power but later incurring the full penalty of $T_{\text{DS}}$.

The search for the optimal pair of Stall-Timer values is illustrated in Figure 7.3. The two-dimensional space on the left of the figure contains all potential pairs of Stall-Timer values. Only the pairs in the lower-right triangle are valid, since Stall-to-Shallow must always be less than or equal to Stall-to-Deep. The (pink) region in the lower-left contains valid Stall-Timer pairs that should be excluded because the estimated performance overhead is greater than the link's local performance bound, $l$. The algorithm then chooses the acceptable solution with the greatest estimated energy savings. As indicated at the right, and explained in detail below, the performance overhead and energy savings are estimated using histograms derived from that of Figure 7.2(B). Also, two "monotonicity" properties, also described below, mean that the search is one dimensional, not two, and this is suggested by the (green) arrow.

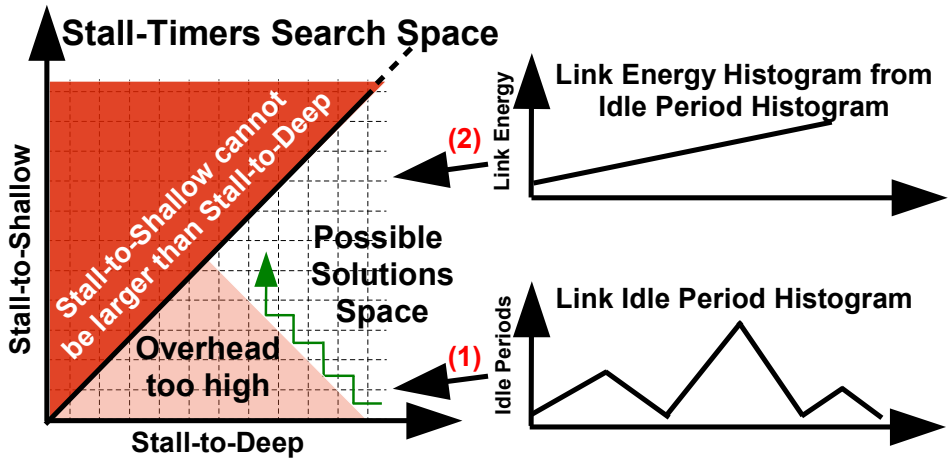



Figure 7.3: Stall-Timer search logic of DoublePerfBound; (1) Idle period histogram used to find possible solutions and (2) for each possible solution, energy histogram is used to find lowest energy solution

The performance overhead is estimated using the idle period histogram of Figure 7.2(B), illustrated at the bottom- right of Figure 7.3 and marked (1). Given the two Stall-Timer values, the histogram is used to determine the number of idle periods that are followed by a Fast-Wake, denoted $M_{\text{FW}}$, and the number of idle periods followed by a Full-Wake, denoted $M_{\text{DS}}$. The estimated overhead is then $T_{\text{overh}} = M_{\text{FW}} \cdot T_{\text{FW}} + M_{\text{DS}} \cdot T_{\text{DS}}$. The first step of DoublePerfBound calculated the total acceptable wakeup time as $T_{\text{D}} = lX$, so the Stall-Timer pair is excluded whenever this bound is exceeded; i.e. if $T_{\text{overh}} > T_{\text{D}}$. Otherwise, the pair of Stall- Timers give an acceptable solution, although it may not be optimal in terms of energy.

The energy savings are estimated as shown at the top-right of Figure 7.3 and marked (2). This step is done using the *energy histogram*, which measures the total idle time in each bin, and is proportional to the energy savings, rather than the idle period histogram, which measures the *number* of idle intervals in the bin, and was proportional to the performance overhead. In fact, instead of separately collecting the energy histogram, we found that it was sufficient to approximate it from the idle period histogram in Figure 7.2(B), simply by multiplying the number of idle periods in each bin by the mid-point of the bin. If $I_{\text{FW}}$ is the total idle time in idle periods followed by a Full-Wake, determined from the energy histogram, and $I_{\text{DS}}$ is the total idle time in idle periods followed by a Deep-Sleep, also from the histogram, then the energy savings are estimated to be $E_{\text{sav}} = \frac{40}{100} I_{\text{FW}} + \frac{90}{100} I_{\text{DS}}$.

**Algorithm 1** DoublePerfBound search algorithm

```
B_D = D_MAX                                   ▷ Stall-to-Deep bin index
B_S = 0                                       ▷ Stall-to-Shallow bin index
E_BEST = 0                                    ▷ Best energy savings so far
B_BEST = (B_D, B_S)                           ▷ Best Stall-Timers so far
do
    Calculate T_overh and E_sav
    if T_overh < T_D then                     ▷ Acceptable, so Move left
        if E_sav > E_BEST then
            E_BEST = E_sav
            B_BEST = (B_D, B_S)
        end if
        B_D = B_D − 1
    else                                      ▷ Rejected, so Move up
        B_S = B_S + 1
    end if
while B_S ≤ B_D
return B_BEST
```

Combining the above, the optimal pair of Stall-Timers is chosen to maximise $E_{\text{sav}}$ subject to $T_{\text{overh}} \leq T_{\text{D}}$, and this can be naively done using a two-dimensional search over all pairs of valid Stall-Timers.

The optimised one-dimensional search is given in Algorithm 1. This algorithm manipulates the bin indexes for the two Stall-Timers, labelled $B_{\text{D}}$ and $B_{\text{S}}$, rather than the Stall-Timers themselves. It starts with Stall-to-Deep at its maximum value ($B_{\text{D}} = D_{\text{MAX}}$) and Stall-to-Shallow at zero ($B_{\text{S}} = 0$). Whenever the performance overheads are acceptable, then the best energy savings is updated if necessary, and Stall-to-Deep is reduced (by decrementing $B_{\text{D}}$), moving to the left in Figure 7.3. If the overheads are too high, then Stall-to-Shallow is increased (by incrementing $B_{\text{S}}$), moving up. The algorithm terminates when the Stall-Timers cross, which is when the solution enters the invalid upper-left triangle in Figure 7.3. Since it is always true that $0 \leq B_{\text{S}} \leq B_{\text{D}} \leq D_{\text{MAX}}$, both Stall-Timers stay in bounds.

This algorithm can be proved to be correct, using two "monotonicity" properties. These are easier stated if the values of Stall-to-Shallow and Stall-to-Deep are abbreviated as $S_{\text{S}}$ and $S_{\text{D}}$, respectively. Firstly, if $(S_{\text{D}}, S_{\text{S}})$ is excluded because the overhead is too high, then clearly all $(S'_{\text{D}}, S'_{\text{S}})$ with $S'_{\text{D}} \leq S_{\text{D}}$ and $S'_{\text{S}} \leq S_{\text{S}}$ have smaller Stall-Timers, and therefore larger overheads, so they must be excluded also. Secondly, if $(S_{\text{D}}, S_{\text{S}})$ has acceptable overheads and the energy savings have been estimated, then all $(S'_{\text{D}}, S'_{\text{S}})$ with $S'_{\text{D}} \geq S_{\text{D}}$ and $S'_{\text{S}} \geq S_{\text{S}}$ have larger Stall-Timers, and therefore smaller energy

savings, so they cannot be closer to optimal.[2]

Using these properties, the algorithm can be shown to be equivalent to scanning the two-dimensional space in Figure 7.3, starting from the lower-right, and moving right-to-left then bottom-to-top, excluding pairs that do not need to be checked due to the monotonicity relationships. When a pair is rejected because the performance overheads are unacceptable, then performance monotonicity implies that all values to the left can be ignored, as the performance overheads will also be unacceptable. Moreover, all values one row up and strictly to the right can also be excluded due to energy monotonicity applied to the previously visited Stall-Timer pair. Hence the next potentially optimal pair of Stall-Timers is found by moving up.

Since $B_{\mathrm{D}} - B_{\mathrm{S}}$ starts at $D_{\mathrm{MAX}}$, and it decreases by one in each iteration of the loop, the number of iterations is simply $D_{\mathrm{MAX}}$. This is the same as the worst case for the original PerfBound, though maybe twice its average case. It is also worth noting that it is not necessary to recalculate $T_{\mathrm{overh}}$ and $E_{\mathrm{sav}}$ from scratch on each iteration, since, each iteration, only the contributions from a single bin are changed. This optimization is not included in Algorithm 1 for the sake of simplicity, though it is the reason why $E_{\mathrm{sav}}$ is calculated every iteration instead of only when needed.

### 7.4.3 Stall-Timer Error Correction

This section describes a low-complexity feed-forward control mechanism that improves the accuracy of PerfBound and in extension of DoublePerfBound in reaching a target performance overhead bound. As described above, PerfBound translates its performance overhead bound to a target number of messages that are allowed to be delayed and then chooses Stall-Timers to match this target. Although the algorithm is reasonably accurate, the target messages that can be delayed does not always exactly match the actual messages delayed, as observed at the links. This is mainly because of the discrete nature of histograms. In particular, the Stall-Timers are always chosen to be at a mid-point of one of the bins. Increasing the number of bins reduces the error, at the cost of increased computation time.

Figure 7.4 (A) illustrates this tracking error, for application BT. The dashed black

[2]The phrase “has acceptable overheads” is necessary because when the overheads are large, increasing Stall-Timers may actually save energy indirectly through a decrease in execution time.

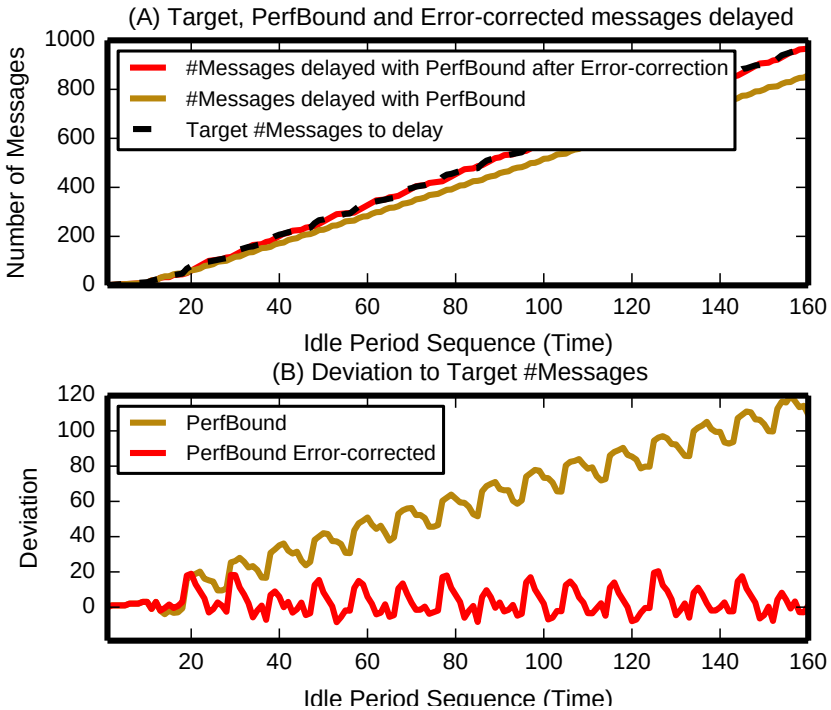



Figure 7.4: Stall-Timer Error Correction and #Target messages to actual messages delayed by PerfBound and DoublePerfBound for BT

line is the target number of delayed messages, and the yellow line is the actual number of messages delayed by PerfBound. As seen, there is a consistent error of about 20%, with the actual number of delayed messages lower then the target, meaning that PerfBound achieves lower energy savings than it could have. This tracking error was similarly observed in other examined applications.

Figure 7.4 (B) plots error difference between actual messages delayed and target messages delayed as presented in Figure 7.4 (A). Here, zero represents no deviation between target and actual messages delayed. It is clear from Figure 7.4 (B) that this deviation increases without bound for the previous PerfBound (in red).

This is solved with the use of simple feed-forward controller that monitors the difference between the actual and target numbers of delayed messages. If this difference exceeds an arbitrary but fixed value of twenty messages, it applies an offset to the Deep-Sleep Stall-Timer to force the difference towards zero. In Figure 7.4(A), the red line is the number of messages delayed after this error correction and in Figure 7.4(B), the red line shows the deviation from the target. As shown, the error after correction stays close to zero.

## 7.5 Results

The performance–energy trade-offs for the DoublePerfBound algorithm are shown in Figure 7.5. These results were obtained using the methodology and applications described in the methodology section of Chapter 6. For each application, the $x$-axis is the execution time and the $y$-axis is the average link energy, both normalized to the corresponding values for an always-on network. In these plots, moving down and/or left is preferred, since this corresponds to decreasing energy and/or improving performance, respectively. In practice, the mechanism offers a trade-off between performance and energy; hence the curves.

The baseline results are as follows. The dashed blue line is a static sweep of a single Stall-Timer, and the large yellow triangle is the result from PerfBound, configured for a performance overhead of 1%. The smaller yellow triangles are results for PerfBound configured for performance overheads of 0.5%, 2%, 3% and 4%.

The Fast-Wake results are as follows. The dashed black line is the Pareto-optimal curve from an exhaustive static search over all pairs of Stall-Timers, obtained as explained in the description of Figure 7.1. Finally, the large red square is the result from DoublePerfBound configured for a performance overhead of 1%. The small red squares are obtained by varying the performance overhead parameter as above. This section focuses on discussing the 1% performance overhead configurations, which were highlighted by the larger points. The conclusions from the other configurations are similar.

The first observation from Figure 7.5 is **for ten of the fourteen applications, DoublePerfBound finds a result on or below the static Pareto-optimal curve**. Specifically, the DoublePerfBound curve overlaps the static Pareto-optimal curve. Applications SP and MILC has DoublePerfBound values below the Pareto-optimal curve, indicating higher energy and lower performance overheads than obtainable with a sweep. The reason for this is that all links of a sweep are all a configured to a single fixed Stall-Timer. An ideal sweep would vary for all links all possible Stall-Timers, but this would have translated to an enormous number of simulations so it was not feasible. Since PerfBound and DoublePerfBound are independent and local to each link, the heuristics naturally configure different Stall-Timers for each link, depending on locally visible traffic and can thus be better than the sweeps.

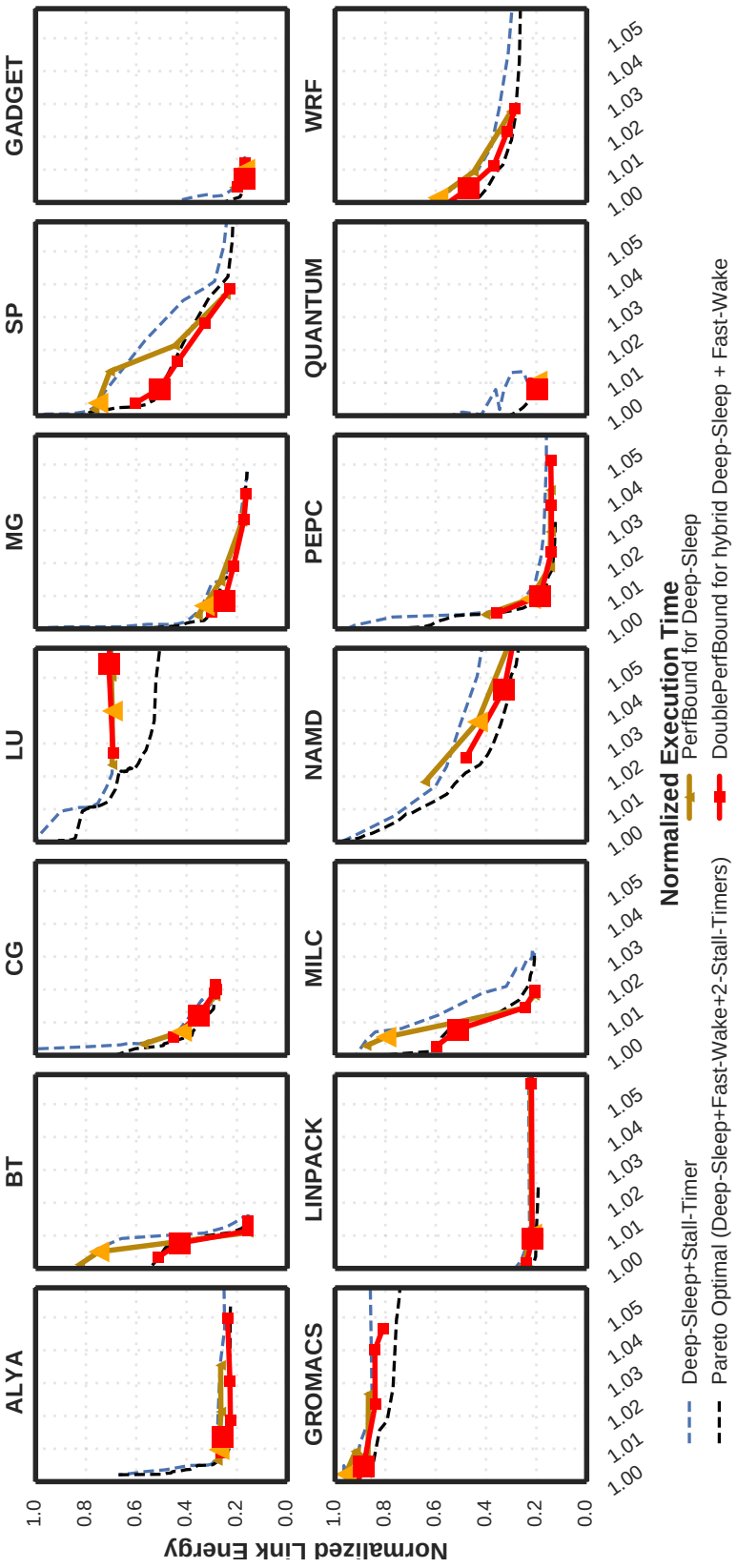



Figure 7.5: Energy and Performance of various configurations of DoublePerfBound and PerfBound (0.5%, 1% (large square and triangle), 2%, 3% and 4%) compared to sweep analysis of Deep-Sleep and the Pareto-optimal curve of the hybrid Deep-Sleep+Fast-Wake mechanisms

In Figure 7.5, for a 1% overhead, SP shows about 20% improvement in energy from DoublePerfBound. This was the dangerous example discussed in Section 7.2, for which the Stall- Timer speed contains values far from the Pareto-optimal curve. DoublePerfBound for SP has its value on the Pareto-optimal curve itself. Similarly, BT and MILC also show high energy savings from DoublePerfBound. The possible 5% higher energy saving potential observed for MG is captured by DoublePerfBound. For application ALYA however only 2% is saved from its possible 5%. Although even ALYA, for performance overheads larger than 1%, have DoublePerfBound values overlapping the Pareto-optimal curve. Section 7.2, specifically discusses ALYA for its many combinations of possible Stall-Timer values that are significantly worse than the Pareto-optimal curve or even the Deep-Sleep sweep. To this end, for ALYA, both DoublePerfBound and PerfBound have nearly optimal energy to configured performance overheads. Similar to ALYA and MG are applications, CG, WRF and GROMACS with the similar 5% improvement in energy savings. Some applications, specifically GADGET, LINPACK and QUANTUM, have no opportunity to obtain energy savings from Fast-Wake. Therefore, for the three applications where Fast-Wake offers no additional energy savings, DoublePerfBound found the same Pareto-optimal solution as PerfBound.

Figure 7.5 also presents the accuracy of DoublePerfBound in maintaining its configured performance overhead bound. **For twelve out of fourteen applications, the deviation from the configured performance overhead was less than 1%**. This error is potentially due to the inherent application behavior. For example, when an application ends immediately after a network long idle period, with no further messages, an opportunity to delay messages is lost. This is because, with the PerfBound heuristic, the number of messages delayed and in extension incurred performance overhead is proportional to time. A large idle period introduces time that could potentially be used to delay more messages for energy savings. This however cannot be exploited if the application immediately ends with no new impending messages. Hence in this scenario expected overhead is slightly lower than the configured overhead bound. Similarly a large burst of messages towards the end of an application, could introduce many delayed messages, and may not give the heuristic time to react or adjust its Stall-Timers to correspondingly control overheads caused by the same. This may cause a small increase in overhead compared to the bound. Two applications, specifically NAMD and LU have higher overhead than the configured bound but still is less than 5% for the configured 1% bound. These applications have high inter-message dependencies that PerfBound does not account for in its calculation.

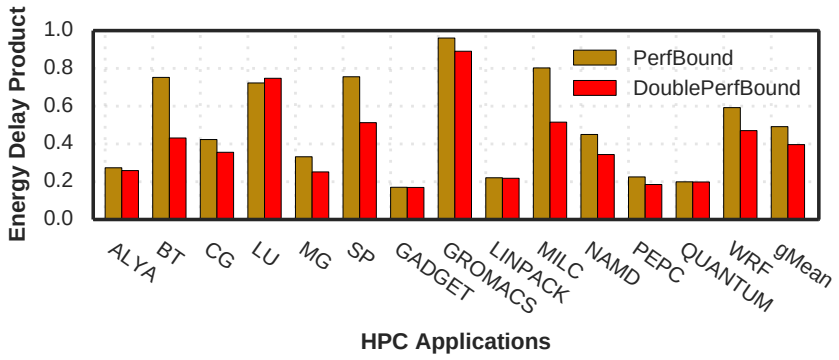



Figure 7.6: Energy Delay Product obtained from Figure 7.5

Out of fourteen, DoublePerfBound saves 70% energy for six applications and 40-70% for another six, with a performance overhead bounded to only 1%. The two applications that have lower than 40% energy saving, LU and GROMACS both have high network usage. GROMACS in specific is also latency sensitive and a bad choice of Stall-Timers have the potential for large performance overheads. While DoublePerfBound does not save energy when the opportunity is limited, it still bounds performance overheads.

Figure 7.6 shows for the energy–delay product (EDP) for all points in Figure 7.5 with applications configured at a 1% performance overhead bound. On average **DoublePerfBound has EDP 10% better than PerfBound**.

### 7.5.1 Discussion

This thesis assumes the possibility of having PerfBound/Double PerfBound implemented within each NIC. A brief hardware requirement/feasibility discussion is presented below. The Double PerfBound heuristic itself requires two histograms, hardware counters and a simple 1-pass logic as discussed in Section 7.4. The histograms were set to only 100 bins and since analyzes on larger histograms did not show any major benefit. All idle periods below 1us were ignored, i.e., the algorithm is not invoked since stall- timers operate at hundreds of microsecond values. In calculating stall- timers, the heuristic decides when to turn off the link and this is not time critical. A simple hardware logic could compute the stall- timers in few hundred cycles but the decisions are only required in microsecond ranges, making the overhead of this heuristic very low. Furthermore since the the heuristic is not time critical, it can implemented over slower but power efficient hardware.

## 7.6 Conclusions

Energy consumption is a key challenge in high-performance computing, but the primary goal will continue to be performance. Mechanisms to reduce system energy consumption will, therefore, only be employed if the impact on performance is known and small. Switches and NICs implementing the new Energy Efficient Ethernet standards, with support for Fast-Wake, will soon be deployed in HPC systems, but, without continued investigation, these features will likely be disabled by default.

This chapter introduced DoublePerfBound, a self-contained technique to manage hybrid Fast-Wake + Deep-Sleep to minimise the interconnect link energy consumption, subject to a bound on the performance degradation. DoublePerfBound was evaluated using traces from a production supercomputer and for twelve of fourteen applications, DoublePerfBound finds a result on or below the static Pareto-optimal curve. Overall, DoublePerfBound achieves an energy–delay product 10% better than the single stall-timer PerfBound technique. No changes are required to the application, and, since it uses only local information already available at the switches and NICs, there is no additional communication. It is also compatible with multi-hop networks. With the ratification of Fast-Wake in March 2014 and 2015 for 40/100Gb Ethernet, and the ongoing standardization effort for 400Gb Ethernet, this work could help interconnect vendors to build Fast-Wake into energy-efficient switches and NICs that target HPC.

# Chapter 8
# Conclusions

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

High performance interconnects consume a significant portion of system energy. Typical interconnects consume up to 12% of the total system energy at full load and more when the application does not fully utilize the CPU and memory. Interconnects, in particular, are not energy efficient because their links, which consume up to 65% of the total interconnect power, are essentially always-on, consuming energy even when idle. Links traditionally remain always- on, continually transmitting signals, for link alignment and synchronization. Advances in energy proportionality of compute and memory elements further increases the proportion of system energy by interconnects.

Recognizing the need for energy proportional interconnects, the IEEE 802.3az Energy Efficient Ethernet (EEE) standard was ratified providing specifications to turn off links during periods of inactivity. While the standard provides mechanisms to turn on and off links, the mechanisms that govern power management are vendor specific and are an active area of research. Although Ethernet switches with EEE currently exist in the market, HPC vendors typically do not enable energy savings mechanisms due to their largely unknown effect on performance.

This thesis is among the first to evaluate the IEEE 802.3az Energy Efficient Ethernet (EEE) standard for its potential use in HPC. The results showed that while EEE promised energy savings, its wake-up delay results in performance degradation for latency sensitive applications. To control performance overheads, Stall-Timer or Power-Down Threshold as a mechanism was shown to reduce overhead while still ensuring

significant energy savings. One problem with Stall-Timers however is that they are application dependent, each with its own optimal value.

PerfBound as a solution was discussed to mitigate this issue. PerfBound automatically adjusts the Stall-Timer such that maximum energy savings can be obtained while bounding performance overhead. PerfBound is a self–contained mechanism that does not require any changes to the application and only uses information locally present in NICs and hence no additional communication is added.

This thesis further discusses an addition to EEE, Fast-Wake an intermediate link state. Evaluation of Fast-Wake showed that a hybrid approach with both Fast-Wake and Deep-Sleep showed higher energy savings as opposed using only either of them alone. This opportunity for higher energy savings can only be obtained with careful adjustment of the two Stall-Timers of Deep-Sleep and Fast-Wake respectively. An extension from PerfBound, DoublePerfBound was proposed, that which finds Pareto-optimal solutions with regard to energy– performance.

Combined these proposed techniques and evaluation make a strong case for the use of EEE and similar energy saving mechanisms in HPC links. The results presented in this thesis show that up to 70% of the link energy can be saved with these mechanisms with only a 1% performance overhead. With the ratification of Fast-Wake in March 2014 and 2015 for 40/100Gb Ethernet, and the ongoing standardization effort for 400Gb Ethernet, this thesis could help interconnect vendors to build EEE into energy-efficient switches and NICs that target HPC.

## 8.1 General discussion and Potential extensions

**Multiple application traffic flows:** This thesis presented results evaluating a single application per simulation but in actual HPC systems many 10s or 100s of applications may execute at a given time with complex traffic in the network. In regards to this, note that for in any indirect network, edge links between a node and a lowest-level switch are only ever used by one application. In multi-level networks typical of Ethernet, e.g. fat trees, since jobs are generally scheduled to groups of adjacent nodes, we believe that relatively few higher-level links see traffic from multiple applications.

Furthermore, the proposed algorithm in bounding performance overheads, would in fact work well in the case of concurrent applications. The 1% bound is still respected,

for each application separately, but the energy savings on the (few) shared links is somewhat conservative. To see this, imagine that a particular link is shared between application A and application B. Then the heuristic introduces wakeup overheads that total at most 1% of the elapsed time. The heuristic does not distinguish the messages of application A from those of application B, so these overheads are divided among the applications in some way; e.g. 0.3% on A and 0.7% on B. Since the total overhead is still 1%, both applications necessarily have overhead $<= 1\%$. A more aggressive algorithm could push the overhead to 1% on each application, to achieve greater energy savings on the shared links. Since we observed that the number of shared links is small, this was seen as a small optimization that would not be worth the extra complexity.

A possible extension of this work could be to extend the PerfBound heuristic to adapt to multiple application flows. This may particularly useful for higher level network links that may have shared links where traffic flow from multiple applications can make PerfBound take a conservative approach.

**Routing and Topology in EEE:** This thesis discussed Energy Efficient Ethernet over static routing and a fat-tree based hierarchical network topology. Different adaptations of EEE may be required for specific topologies and routing techniques. It is still important however to restrict performance overheads and build techniques similar to PerfBound in ensuring a bound on overheads. A possible extension of this work could be in analyzing how PerfBound and/or EEE could be applied to topologies such as Torus or DragonFly.

**Asynchronous communication patterns:** The applications discussed in this thesis showed iterative and predictable communication behavior. There is however an increased push in the community for applications that overlap communication and computation. Having overlapped communication and computation or asynchronous communication comes with various advantages. Firstly, communication overlapped with computation means waits for messages are reduced and hence performance increases. Secondly, overlapping communication removes communication from the critical path. This could make applications more latency tolerant. The use of such communication models brings about interesting challenges for energy savings. Applications with overlapped communication and computation phases may have MPI messages spread throughout the application. Thus traffic patterns could be more sporadic making it harder to build energy saving heuristics.

## 8.2 Work published

**[ISPASS 2013]** **Karthikeyan P. Saravanan**, Paul M. Carpenter and Alex Ramirez, "*Power/Performance evaluation of Energy Efficient Ethernet (EEE) for High Performance Computing*" in proceedings of 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2013) [50]

**[ICS 2014]** **Karthikeyan P. Saravanan**, Paul M. Carpenter and Alex Ramirez, "*A Performance Perspective on energy efficient HPC links* " in proceedings of 28th ACM International Conference on Supercomputing (ICS), 2014 [51]

**[ICCD 2015]** **Karthikeyan P. Saravanan**, Paul M. Carpenter and Alex Ramirez, "*Exploring multiple sleep modes in on/off based energy efficient HPC networks*" in proceedings of 33rd IEEE International Conference on Computer Design (ICCD), 2015 [52]

**[- *2016*]** **Karthikeyan P. Saravanan**, Paul M. Carpenter and Alex Ramirez, "*Dynamic-Fastwake: Conserving energy with bounded performance overheads in On/Off-based HPC interconnects*", Under Review

# References


[1] IEEE Draft P802. 3az/D2. 3, Energy Efficient Ethernet, 2010. 17, 28, 31, 32, 37

[2] Active/Idle Toggling with Low Power Idle, IEEE 802.3az Task Force. 12, 15, 16, 22, 24, 44

[3] Top500 list, June 2012,
url: *http://www.top500.org/list/2012/06/100* 1, 5, 6, 7

[4] Green500: List of most energy efficient Supercomputers.
*http://www.green500.org* 6

[5] Mont-Blanc: European approach towards energy efficient high performance.
*http://www.montblanc-project.eu/*

[6] Myrinet overview
http://www.myricom.com/scs/myrinet/overview/ 9

[7] SGI NUMAlink Industry Leading Interconnect Technology. White paper. April 2013 9

[8] Petrini, F., Feng, W.C., Hoisie, A., Coll, S. and Frachtenberg, E., The quadrics network (qsnet): High-performance clustering technology. In Hot Interconnects 9, 2001. (pp. 125-130). IEEE. Vancouver 9

[9] A. Sodani. Race to exascale: Opportunities and challenges. Keynote Speech, MICRO 44, Dec, 2011. 1

[10] D. Turek, Challenges on the road to exascale computing (invited talk), Simulating the Future Workshop, 2008.

[11] Peter Kogge et al. ExaScale Computing Study: Technology Challenges in achieving Exascale Systems, June'08. 1, 6, 7

[12] Scientific grand challenges: Crosscutting technologies for computing at the exascale. Report from the Workshop Held Feb'10, by U.S.DOE. 1, 7

[13] R. Bertran, Y. Sugawara, H. M. Jacobson, A. Buyuktosunoglu and P. Bose, "Application-level power and performance characterization and optimization on IBM Blue Gene/Q systems," in IBM Journal of Research and Development, vol. 57, no. 1/2, pp. 4:1-4:17, Jan.-March 2013. 12, 22

[14] Extrae Tool from Barcelona Supercomputing Center.
https://www.bsc.es/computer-sciences/extrae 25

[15] 35 Years Ago - the Control Data 6600
http://ref.web.cern.ch/ref/CERN/CNL/2001/001/cdc6600/ 5

[16] Human Brain Project
https://www.humanbrainproject.eu/ 3

[17] The Large Hadron Collider
http://home.cern/topics/large-hadron-collider 4

[18] The Scientific Case for High Performance Computing in Europe 2012 - 2020. From PetaScale to ExaScale. Prace Report, Oct 2012. http://www.prace-ri.eu/IMG/pdf/prace_-_the_scientific_case_-_full_text_-.pdf 6, 7

[19] The BRAIN Initiative
http://www.braininitiative.nih.gov 3

[20] NASA climate report
http://www.nccs.nasa.gov/ 2, 3

[21] Grand Challenges: High Performance Computing and Communications, The FY 1992 U.S. Research and Development Program. 2

[22] Worldwide LHC Computing Grid
http://wlcg-public.web.cern.ch/about

[23] RDMA - iWARP
http://www.chelsio.com/nic/rdma-iwarp/ 11

[24] RoCE vs iWARP Competitive Analysis
http://www.mellanox.com/related-docs/whitepapers/WP_RoCE_vs_iWARP.pdf 11

[25] Unveiling parallelization strategies at undergraduate level.
*http://pm.bsc.es/SC12_training_session/material/SlidesTareador.pdf*

[26] Jesus Labarta *et al.*, "DiP: A Parallel Program Development Environment", In proceedings of the Second International Euro-Par Conference on Parallel Processing, 1996 23, 27

[27] Badia, R. M., Labarta, J., Gimenez, J., and Escale, F. (2003, June). DIMEMAS: Predicting MPI applications behavior in Grid environments. In Workshop on Grid Applications and Programming Tools (GGF8) (Vol. 86, pp. 52-62). 23

[28] Girona, S., Labarta, J., and Badia, R. M. (2000). Validation of Dimemas communication model for MPI collective operations. In Recent Advances in Parallel Virtual Machine and Message Passing Interface (pp. 39-46). Springer Berlin Heidelberg. 23

[29] Gonzalez, J., Gimenez, J., Casas, M., Moreto, M., Ramirez, A., Labarta, J., and Valero, M. (2011). Simulating whole supercomputer applications. IEEE Micro, (3), 32-45. 23

[30] F. Petrini and M. Vanneschi, "k-ary n-trees: high performance networks for massively parallel architectures," In proceedings of the 11th International Parallel Processing Symposium, 1997 24

[31] L. Shang, L.-S. Peh, and N. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In proceedings of the 9th International Symposium on High-Performance Computer Architecture, 2003 12

[32] IBM InfiniBand 8-port 12x Switch,
url: *http://www-3.ibm.com/chips/ products/infiniband* 8, 31

[33] H.-S. Wang *et al.*, A power model for routers: modeling alpha 21364 and infiniband routers. In proceedings of 10th Symposium on High Performance Interconnects, 2002. 31, 32

[34] P. Kogge. Architectural challenges at the exascale frontier (invited talk), Simulating the Future Workshop.2008. 8, 31, 32

[35] J. Maestro *et al.*, Energy efficiency in industrial ethernet: The case of powerlink. IEEE Transactions on Industrial Electronics, Aug. 2010. 15, 22

[36] S. Herreria et al.,How efficient is energy efficient ethernet, in 3rd International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2011 22

[37] M. Mostowfi and K. Christensen. Saving energy in lan switches: New methods of packet coalescing for energy efficient ethernet, in proceedings of the International Green Computing Conference and Workshops, 2011 32

[38] Ajima *et al.*, "Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers," in journal of Computer, 2009. 9

[39] Liao XK *et al.*, The TianHe-1A Supercomputer: Its Hardware and Software, in Journal of Computer Science and Technology.

[40] K. J. C.Chamara Gunaratne. Ethernet adaptive link rate: System design and performance evaluation, in proceedings of 31st IEEE Conference on Local Computer Networks, 2006. 15, 22

[41] Anand, Himanshu, Casey Reardon, Rajagopal Subramaniyan, and Alan D. George. "Ethernet adaptive link rate (ALR): Analysis of a MAC handshake protocol." In Proceedings. 2006 31st IEEE Conference on Local Computer Networks, IEEE, 2006. 15, 22

[42] Baoke Zhang, K. Sabhanatarajan, A. Gordon-Ross and A. George, Real-time performance analysis of Adaptive Link Rate. 33rd IEEE Conference on Local Computer Networks (LCN), Montreal, Que, 2008, pp. 282-288. 15, 22

[43] C. Gunaratne *et al.*, Reducing the energy consumption of ethernet with adaptive link rate (ALR). IEEE Trans. Comput. Apr08. 15, 22

[44] Blanquicet, Francisco, and Ken Christensen. "An initial performance evaluation of rapid PHY selection (RPS) for energy efficient Ethernet." In Local Computer Networks, 2007. LCN 2007. 32nd IEEE Conference on, pp. 223-225. IEEE, 2007. 16, 22

[45] Arthur Marris, Comments on EEE operation, IEEE P802.3bj Task Force, September 2013.
ieee802.org/3/bj/public/sep13/marris_3bj_01_0913.pdf

[46] Growth in Data center electricity use 2005 to 2010. http://www.analyticspress.com/datacenters.html 15

[47] Thomas Ludwig and Manuel Dolz, Total Cost of Ownership in High Performance Computing,

[48] Dennis Abts and Mike Marty and Philip Wells and Peter Klausler and Hong Liu Energy Proportional Datacenter Networks. In *Proc. International Symposium on Computer Architecture (ISCA)*, 2010. 8, 12, 13, 20, 21, 32

[49] Li, J., Huang, W., Lefurgy, C., Zhang, L., Denzel, W. E., Treumann, R. R., and Wang, K. Power shifting in thrifty interconnection network. In IEEE 17th International Symposium on High Performance Computer Architecture (HPCA), 2011 12, 20, 31, 32

[50] K. P. Saravanan, P. M. Carpenter, and A. Ramirez, Power/performance evaluation of energy efficient Ethernet (EEE) for High Performance Computing. In Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on 2013 Apr 21 (pp. 205-214). IEEE. 45, 99

[51] Saravanan, K. P., Carpenter, P. M., and Ramirez, A. (2014, June). A performance perspective on energy efficient HPC links. In Proceedings of the 28th ACM international conference on Supercomputing (pp. 313-322). ACM. 99

[52] Saravanan, K. P., Carpente, P. M., and Ramirez, A. (2015, October). Exploring multiple sleep modes in on/off based energy efficient HPC networks. In Computer Design (ICCD), 2015 33rd IEEE International Conference on (pp. 54-61). IEEE. 99

[53] The MareNostrum II System Architecture. http://www.bsc.es/marenostrum-support-services/marenostrum-system-architecture 23, 26

[54] Christensen, K., Reviriego, P., Nordman, B., Bennett, M., Mostowfi, M., and Maestro, J. A. IEEE 802.3 az: the road to energy efficient ethernet. IEEE Communications Magazine, 2010. 8, 12, 15, 16, 17, 21, 22, 24, 29, 32, 44

[55] Reviriego P. et al. Performance evaluation of energy efficient ethernet. *Comm. Letters.*, 13(9):697–699, September 2009. 8, 16, 17, 21, 22, 32

[56] Branimir Dickov. MPI Layer Techniques to Improve Network Energy Efficiency. PhD Thesis, 10 December 2015. 20

[57] Renan Fischer e Silva, Paul M. Carpenter, Exploring Interconnect Energy Savings Under East-West Traffic Pattern of MapReduce Clusters, 40th IEEE Conference on Local Computer Networks (LCN), 2015 22

[58] Hugh Barrass, Options for EEE in 100G IEEE 802.3bj 19, 24, 76

[59] Groves, Taylor, and Ryan Grant. "Power Aware, Dynamic Provisioning of HPC Networks." Sandia National Labs report. 21

[60] Soteriou, Vassos, Noel Eisley, and Li-Shiuan Peh. "Software-directed power-aware interconnection networks." ACM Transactions on Architecture and Code Optimization (TACO), (2007)

[61] B. Dickov, M. PericÃăs, P. Carpenter, N. Navarro and E. AyguadÃľ, "Software-Managed Power Reduction in Infiniband Links," 2014 43rd International Conference on Parallel Processing, Minneapolis MN, 2014, pp. 311-320. 20

[62] Miwa, Shinobu, Sho Aita, and Hiroshi Nakamura. "Performance estimation of high performance computing systems with Energy Efficient Ethernet technology." Computer Science-Research and Development 29.3-4 (2014): 161-169. 21

[63] Soteriou, Vassos, and Li-Shiuan Peh, Dynamic power management for power optimization of interconnection networks using on/off links. In *High Performance Interconnects*, pages 15–20. IEEE, 2003. 12

[64] R. F. e Silva and P. M. Carpenter Exploring Interconnect Energy Savings Under East-West Traffic Pattern of MapReduce Clusters In Proc. 40th IEEE Conference on Local Computer Networks, 2015. 13

[65] Soteriou, Vassos, and Li-Shiuan Peh. "Design-space exploration of power-aware on/off interconnection networks." In Proceedings. IEEE International Conference on. Computer Design: VLSI in Computers and Processors, ICCD 2004. 21

[66] Alonso, Marina, Salvador Coll, Juan-Miguel Martinez, Vicente Santonja, Pedro Lopez, and Jose Duato. "Dynamic power saving in fat-tree interconnection networks using on/off links." In Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, pp. 8-pp. IEEE, 2006. 21

[67] Gupta, M. and Singh, S., 2007, June. Dynamic ethernet link shutdown for energy conservation on ethernet links. In Proc. IEEE International Conference on Communications, 2007. 22

[68] Kim, Eun Jung, Ki Hwan Yum, Greg M. Link, Narayanan Vijaykrishnan, M. Kandemir, Mary Jane Irwin, M. Yousif, and Chita R. Das. "Energy optimization techniques in cluster interconnects." In Proceedings of the 2003 international symposium on Low power electronics and design, pp. 459-464. ACM, 2003. 12, 21

[69] S Conner et al. Link shutdown opportunities during collective communications in 3-D torus nets. In *IPDPS 2007*. 45

[70] YS-C Huang et al. Application-driven end-to-end traffic predictions for low power NoC design. In *VLSI Systems*, 2013. 22, 57

[71] Koibuchi, M., Otsuka, T., Matsutani, H., and Amano, H. (2009, May). An on/off link activation method for low-power ethernet in PC clusters. In Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE. 8, 22, 29

[72] Totoni, E., Jain, N., and Kale, L. V. (2013, May). Toward runtime power management of exascale networks by on/off control of links. In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International (pp. 915-922). IEEE. 22

[73] Alya Red:Computational Biomechanics for Supercomputers. 26, 44

[74] NAS parallel benchmarks.
http://www.nas.nasa.gov/publications/npb.html xii, 26, 27, 44, 45, 46, 49

[75] D. Marx and J. Hutter. Ab-initio Molecular Dynamics: Theory and Implementation, NIC. Forschungszentrum JÃijlich, 2000. 26

[76] "The cosmological simulation code GADGET-2." 26, 44

[77] Hess, Berk, Carsten Kutzner, David Van Der Spoel, and Erik Lindahl. "GROMACS 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation." Journal of chemical theory and computation 4, no. 3 (2008): 435-447. 26, 44, 46

[78] Davies, Teresa, Christer Karlsson, Hui Liu, Chong Ding, and Zizhong Chen. "High performance linpack benchmark: a fault tolerant implementation without checkpointing." In Proceedings of the international conference on Supercomputing, 2011. xii, 26, 44, 45, 46, 49

[79] MIMD Lattice Computation (MILC) Collaboration. 26, 44

[80] Brunner, R. K., Phillips, J. C., and Kale, L. V. (2000, November). Scalable molecular dynamics for large biomolecular systems. In ACM/IEEE Conference on Supercomputing, 2000 xii, 26, 44, 45, 46, 49

[81] PEPC: Pretty Efficient Parallel Coulomb-solve, Interner Bericht Zentralinstitut fur Angewandte Mathematik. 26, 44

[82] Giannozzi, Paolo, et al. "QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials." Journal of Physics: Condensed Matter, 2009. 26, 44, 46

[83] Dudhia, J., Gill, D., Henderson, T., Klemp, J., Skamarock, W., and Wang, W. (2005). The weather research and forecast model: software architecture and performance. In Proc. 11th ECMWF Workshop on the Use of High Performance Computing in Meteorology. 26, 32, 44