Self-Tuned Software-Managed Energy Reduction in InfiniBand Links

Branimir Dickov*† Paul M. Carpenter* Miquel Pericàs‡ Eduard Ayguadé*†
*Barcelona Supercomputing Center †Universitat Politècnica de Catalunya ‡Chalmers University of Technology
{branimir.dickov, paul.carpenter, eduard.ayguade}@bsc.es miquelp@chalmers.se

***Abstract*—One of the biggest challenges in high-performance computing is to reduce the power and energy consumption. Research in energy efficiency has focused mainly on energy consumption at the node level. Less attention has been given to the interconnect, which is becoming a significant source of energy-inefficiency. Although supercomputers undoubtedly require a high-performance interconnect, previous work has shown that network links have low average utilization. It is therefore possible to save energy using low-power modes, but link wake-up latencies must not lead to a loss in performance.**

This paper proposes the Self-tuned Pattern Prediction System (SPPS), a self-tuned algorithm for energy proportionality, which reduces interconnect energy consumption without needing any application-specific configuration parameters. The algorithm uses prediction to discover repetitive patterns in the application's communication, and it is implemented inside the MPI library, so that existing MPI programs do not need to be modified. We build on previous work, which showed how the application structure can be successfully exploited to predict the communication idle intervals. The previous work, however, required the manual adjustment of a critical idle interval length, whose value depends on the application and has a major effect on energy savings. The new technique automatically discovers the optimal value of this parameter, resulting in a self-tuned algorithm that obtains large interconnect energy savings at little performance cost.

We study the effectiveness of our approach using ten real applications and benchmarks. Our simulations show average energy savings in the network links of up to 21%. Moreover, the link wake-up latencies and additional computation times have a negligible effect on performance, with an average penalty less than 1%.

***Keywords*-Energy Efficiency; Interconnection Network; InfiniBand; MPI Programs; Idle Time Prediction**

I. Introduction

High-Performance Computing (HPC) is a crucial tool for modern science and engineering. There is a constant demand for more powerful supercomputers, leading to increasing levels of energy consumption. The industry is working toward a target of 1 EF in 20 MW, which implies a ten-fold improvement in energy efficiency, compared with today's most energy-efficient machine.[1] Such a large reduction in system energy consumption is only possible if there are significant improvements in the energy efficiency across all subsystems. Power-saving techniques for processors and memory are progressing quickly, but there is less progress on reducing the power consumption of the interconnect [1]. With energy-efficient processing elements and larger networks, the interconnection network is expected to account for up to 30% of the system's total power [2]. This fraction can even reach 50% [3] for data center servers since the CPUs operate at lower levels of utilization.

InfiniBand and Ethernet, the two most widely used interconnect technologies in HPC today, are introducing power-saving features. Recent Mellanox Host Channel Adapters (HCAs) support Width Reduction Power Saving (WRPS), which saves power by optimizing each port's bandwidth and number of active lanes. For example, a 40 Gbit/s 4× port can run at 10 Gbit/s by shutting down three of its four QDR lanes. This paper is targeted primarily at InfiniBand networks, but it is worth noting that Ethernet also has similar power saving capabilities. IEEE 802.3az Energy Efficient Ethernet (EEE), introduced in 2010 [4], defines the low-level mechanisms for entering and leaving a low-power sleep mode known as Low Power Idle (LPI).

Although interconnects for HPC systems are dimensioned to provide high peak bandwidth, the interconnect is mostly idle during long computation phases [5], [6], [7]. It should, therefore, be possible to use the above-mentioned features to reduce the interconnect's energy consumption. Such a scheme must work autonomously, without ongoing intervention from operations or assistance from the application developer, and there must be no noticeable performance overhead.

As HPC applications often involve a large number of iterative execution phases, each repeating a similar communication pattern, it is possible to observe the communication behavior in one iteration, and use the knowledge gained to predict the behavior of the subsequent iterations. Specifically, this means detecting the patterns of MPI calls that are repeating within each MPI processes.

Observing the structure of the communication phases, we notice that the vast majority of them are of very short length; the number of long ones is considerably less, but they occupy most of the idle link time. In our previous work, we found that more than 90% of the total link idle time is inside relatively long idle periods, of 200 µs or more. We simplify the prediction problem by ignoring the shorter idle intervals, which would have made a negligible contribution to the energy savings, allowing the pattern prediction algorithm to more efficiently predict the remaining (longer) idle intervals. We therefore propose a mechanism to automatically determine the critical idle interval, which distinguishes short and long idle intervals. We refer to this value as the Grouping Threshold (GT), and it has an important influence on the level of prediction and therefore on energy savings, depending on the application in

[1]The L–CSC machine at GSI Helmholtz Center leads the November 2014 Green500 list with 5.2 GF/W.

a complex way. The resulting mechanism obtains significant energy savings with negligible performance overhead.

This paper proposes the Self-Tuned Pattern Prediction System (SPPS), which is a software-managed interconnect energy saving technique. The SPPS energy-saving mechanism consists of a front-end and a back-end component. The front-end component finds the Grouping Threshold value, and merges each group of consecutive MPI calls separated only by short idle intervals into a single event. The back-end component detects consecutive repeatable patterns in these events using the Pattern Prediction System (PPS) algorithm [7]. Upon successful prediction, the predicted idle lengths are used to shift between link power modes. The whole algorithm is executed in the PMPI profile layer of the MPI library, which provides more context to the algorithm than is available to the OS or NIC firmware. Placing the algorithm in the MPI library allows all MPI programs to benefit from energy savings without needing any changes to their source code.

Specifically, this paper makes the following contributions:

- We give detailed insight into constraints that must be obeyed to obtain successful detection of the optimal length of the idle interval, known as GT, whose value has a critical effect on energy savings.
- We provide a complete description of the Self-Tuned Pattern Prediction System (SPPS), a self-guided prediction-based mechanism that exploits repetitive patterns in the application's communication behaviour to reduce link energy consumption.
- We evaluate the self-tuned energy-saving mechanism using an event-driven simulator with traces obtained from a production run on a real supercomputer. Results show an average reduction in link energy consumption of up to 21%, compared with the power-unaware scheme where links are "always on". We also show that on average the increase in execution time is less than 1%. We show that the Grouping Threshold value selected by the SPPS has energy savings similar to the best value found by a manual exploration of the possible values.

The rest of this paper is structured as follows. Section II describes the existing Pattern Prediction System (PPS) and explains why it is hard to automatically find the Grouping Threshold. Section III introduces the design of our automatic power saving mechanism known as SPPS. Section IV describes the methodology and experimental evaluation, and it explores the energy-time tradeoff. Section V compares with the related work. Finally, Section VI presents the most important conclusions from this work.

II. Background

A. Motivation

Most HPC applications require a high-performance network that provides high bandwidth and low latency. Nevertheless, much of their time is dedicated to computation, during which the network is idle, so the average network utilisation is low [5], [6], [7]. Moreover, more than 90% of the total idle time is inside relatively long idle intervals, of 200 µs or more [7]. The regular structure of scientific applications leads to predictable patterns of communication [7].

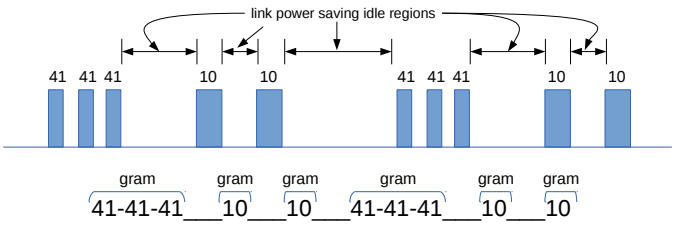


Fig. 1. Forming the array of grams from the MPI event stream of the Alya application. Event IDs are 41 for *MPI_Sendrecv* and 10 for *MPI_Allreduce*

These characteristics provide an opportunity to save a significant amount of energy, since the interconnect can be put into a low-power mode during the long idle intervals. It is important, however, to ensure that there is no visible impact on performance, as switching power mode (off to on, or vice-versa) requires about 10 µs [1]. Although deactivating InfiniBand lanes can usually be overlapped with computation, reactivating the link only on-demand; i.e. when it is needed, will incur a latency penalty. We avoid this overhead and achieve maximum energy savings using a prediction algorithm.

B. Pattern Prediction System (PPS)

The Pattern Prediction System (PPS) [7] is an intelligent link power saving algorithm that can be implemented in the PMPI layer of the MPI library. It detects patterns of repeating MPI calls inside the same MPI process, using an algorithm based on n-gram extraction techniques, a widely used concept from statistical natural language processing [8], [9]. Each *gram* represents one or more consecutive MPI events in the program's execution trace. An MPI event that is preceded by a long idle interval becomes the first MPI event in a new gram. Subsequent MPI events that are preceded by short idle intervals are added, in turn, to the same gram. The threshold that determines whether an interval is short or long is known as the Grouping Threshold (GT), and is discussed further in Section II-C. This process of grouping MPI events into grams is shown in Figure 1. In this figure, each set of three consecutive *MPI_Sendrecv* calls is grouped together to form a single gram while each *MPI_Allreduce* call becomes a separate gram. Finally, an n-gram is a sequence of n consecutive grams, which is used as a pattern in the prediction algorithm.

The PPS has two parts. The first part, the Pattern Prediction Component (PPC), is invoked before every MPI event. It monitors the communication behaviour, and it builds a table known as the *pattern list*, which contains repeatable MPI communication patterns. The PPC detects these patterns based on the MPI event type and the (idle) interval between events. If the PPC determines that the program is following a known repeatable pattern, it sets an output flag that transfers control to the second part, the Power Mode Control Component (PMCC).

Whenever the PMCC component is active, it is invoked *after* every MPI event. It compares the actual MPI events with those expected from the pattern. So long as they continue to match, the length of the next idle interval can be read from the pattern.

At the start of an idle interval that is predicted to be sufficiently long, the link is put into low-power mode for the appropriate amount of time. As long as the program continues to follow the pattern, there is no need to invoke PPC, since the pattern is already known. It is only necessary to continue updating the idle intervals with recent values, allowing some adaptation to varying application characteristics. If the current MPI event does not match the pattern, however, PPC is reactivated and the link is kept in full-power mode until the next repeatable pattern.

The PPS is executed separately for each process, each potentially with its own GT value. If multiple processes on the same node are sharing the same physical network links, then the network links should enter low-power mode only when requested to do so by all processes.

C. Effect of Grouping Threshold (GT) value

The Grouping Threshold (GT) defines the threshold between short and long idle intervals. An MPI event that is preceded by a long idle interval; i.e. one longer than the GT, becomes the first MPI event in a new gram. All consecutive MPI events that are preceded by short intervals will be included in this same gram. This grouping of MPI events into grams was shown in Figure 1.

Figure 2 shows the effect of varying the Grouping Threshold for one process of the MG NAS benchmark. The y-axis is the fraction of time that the link stays in low-power mode, and higher values are clearly better. The x-axis is the value of the GT, varying from the minimum value of $20\,\mu s$, which is the total time required by the hardware to enter and leave the low-power mode, to an upper value of around $56\,ms$. After $32\,ms$, the curve reaches zero as the largest idle interval (the one that can be predicted) has been declared as short one i.e. below the GT value. We observe that the amount of time that the link stays in low-power mode remains steady around GT values of $400\,\mu s$, $2\,ms$ and $15\,ms$.

We wish to find the best value of GT automatically, but the erratic behavior of the function in Figure 2 implies that it will be difficult to optimize, without additional context. There is, however, a connection between the sharp dips in this function and the frequency histogram of the idle interval lengths, as described in Section III-A, which can be used for optimization.

D. InfiniBand switch power management

Mellanox has recently announced it is working on developing Host Channel Adaptors (HCAs) and switches that save power by optimizing each port's link width and speed. These optimizations are embedded in the HCA and switch hardware, and are enabled via the firmware. An InfiniBand port typically has four physical lanes, each providing one quarter of the total bandwidth. The number of active lanes can be reduced using a method called Width Reduction Power Saving (WRPS). For example, using WRPS, a $40\,$Gbit/s $4\times$ QDR port can run as $10\,$Gbit/s $1\times$ QDR by shutting down three of its four QDR lanes.

This reduction in link width reduces the power consumption of Mellanox Switch SX6036 to only 43% of its nominal power (when all four lanes are active) [10]. We use this published value of 43% in the evaluation section as the power consumption of an InfiniBand switch in low-power mode.

III. SPPS Design

This section describes the Self-Tuned Pattern Prediction System (SPPS), a software-managed interconnect energy saving mechanism executed in the PMPI profiling layer of the MPI library. The overall structure of SPPS is shown in Figure 3. The front-end groups the MPI events into grams, each of which contains one or more MPI events separated only by short idle intervals (see Section II-B). The first task is to determine the Grouping Threshold (GT) which distinguishes between long and short intervals. For reasons given in Subsection III-A, idle intervals of length close to the GT may cause problems with prediction. The front-end therefore checks whether any idle intervals appear close to the GT, in a region known as the *dangerous zone*, and, if so, the GT must be recalculated. Otherwise, the MPI events are processed according to PPS [7], by grouping them into grams and passing them to the back-end. In the back-end, the Pattern Prediction Component detects repeatable communication patterns and it uses these patterns to predict the link's idle intervals. If prediction is possible, then the Power Mode Control Component (PMCC) shifts between link power modes.

A. GT detection algorithm approach

This section gives an informal description of the intuition behind the SPPS algorithm. As already described, the Grouping Threshold (GT) is used to classify each idle interval as short or long. In order for the prediction algorithm to function well, this classification must be consistent, meaning that each time the pattern repeats, corresponding MPI events should always be classified in the same way.

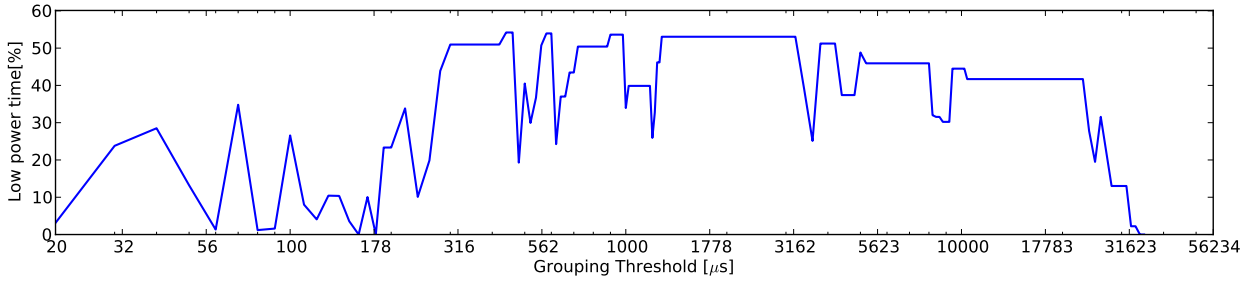


Fig. 2. Effect of Grouping Threshold on time in low power mode, for NASMG benchmark with 32 MPI processes, showing data of process 0

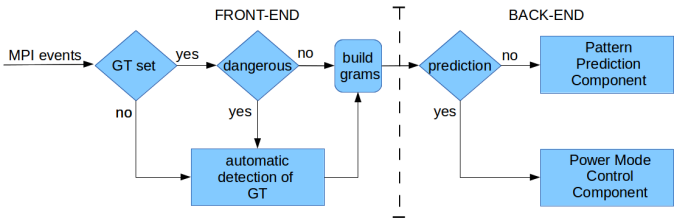


Fig. 3. Simplified block diagram of Self-Tuned Pattern Prediction System (SPPS)

Figure 4(a) is a histogram of the idle intervals, known as $Histogram_f$, for the same example used in Figure 2. Each bin counts the number of idle intervals in the relevant range. For example, there are 268 idle intervals of length between 100 µs and 178 µs. There are a large number of idle intervals of length less than about 300 µs, with no clear internal structure. There are also three smaller peaks, with clear gaps separating them from the other peaks. The region below 300 µs is the same region, visible in Figure 2, for which the GT gives poor and unreliable results. We see similar behaviour across many of the benchmarks.

This suggests that the algorithm should avoid values of the GT close to many idle intervals. If the GT is close to a large number of idle intervals, it is likely that when a pattern is repeated, some of the idle intervals will be classified differently each time, i.e. they will be on different sides of the GT. Such behaviour will mean that the pattern is not recognised, leading to a low prediction rate. To avoid this we should aim to set the GT in an area without idle intervals, which will cover the region of at least GT$\pm$10% (in the rest of the text we refer to this region as the *dangerous zone*). The reason for the size of $\pm$10% is that the lengths of idle link intervals are always subject to noise, which makes corresponding idle intervals have slightly different lengths. The tolerance of $\pm$10% is proposed based on offline clustering experiments, which have shown that idle intervals tend to be in groups of at most this size.

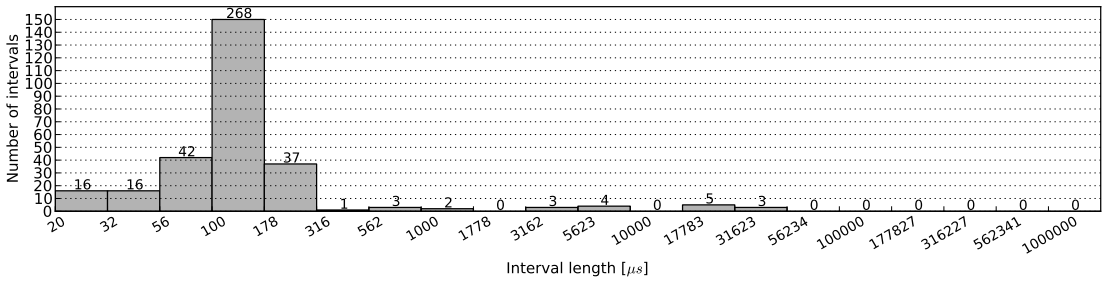
Figure 4(b) is a cumulative histogram of the idle time, which is known as $Histogram_{cumt}$. Each bin is the percentage of the total idle time that is found in idle intervals of length less than or equal to the upper bound of the bin. That is, each bin gives the percentage of the total idle time mapped to either the given bin or a "smaller" bin. The behaviour in the example is typical, as most of the idle time is contained in long intervals. This supports the decision to put the interconnect into low-power mode only during comparatively long intervals. The GT should however be small enough to capture most of the energy savings.

The approach is therefore to use the histogram of the number of idle events, given in Figure 4(a), to choose only values of GT for which there are zero or few idle intervals of a similar length, and then to use the cumulative histogram of the total idle time, shown in Figure 4(b), to choose a GT value with acceptable energy savings.

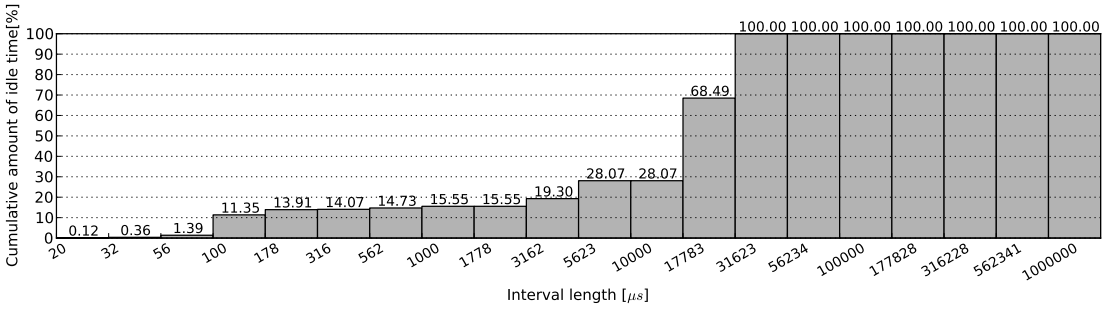
B. Description of the GT detection algorithm

The algorithm that finds the best Grouping Threshold (GT) value is known as Automatized Detection of Grouping Threshold (ADGT) algorithm, and given in pseudocode in Algorithm 1. This algorithm is divided into three phases, which are described in the following subsections.

Configuration and Build-Histogram Phase: The first phase collects a sample of the initial idle interval lengths, and it builds the two histograms defined in Section III-A. Since many applications start with an initialization phase significantly different from the rest of the application, line 1 skips the first few idle intervals. The number of skipped idle



(a) $Histogram_f$: Histogram of number of IDLE intervals



(b) $Histogram_{cumt}$: Cumulative histogram of IDLE interval time

Fig. 4. Idle time histograms for NASMG benchmark with 32 MPI processes, showing data of process 0

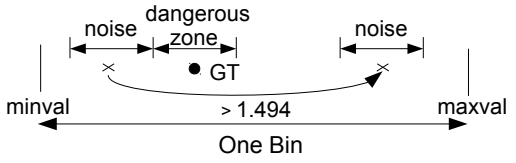


Fig. 5. Necessary gap in the finalization stage for successful GT allocation

intervals is given by *firstIdleIntervals*, and is set to 100 in the evaluation (in a more sophisticated system, the initialization phase could be determined dynamically). Line 2 collects the sample of idle intervals, and lines 5 and 6 build the two histograms. The number of bins for the histograms is given by a common rule of thumb, which uses the square-root of the number of idle intervals in the sample. Both histograms should use a log-scale representation of the idle intervals, due to an empirically observed skewness towards shorter intervals. In both histograms, the last bin, corresponding to the longest idle intervals, should include all idle intervals longer than its lower bound; i.e. they should also include the idle events longer than the histogram's nominal upper bound of 1.7 s. Lastly, we set the demanded time of a link in low-power mode, $demCumIdleTime$, to the initial value of 90%. This value was chosen based on our previous study [7], which found that 90% of idle time is spent in relatively large idle intervals. This value is, however, decreased if no solution is found, as elaborated in the next phase.

Main Phase: The main phase finds a histogram bin containing at most a few idle intervals that still leads to acceptable energy savings. Line 14 excludes all values of GT that would lead to unacceptable energy savings, i.e. those where the energy savings would be less than $demCumIdleTime$. This is done by pointing $indexBin$ to the "smallest" bin that does not have acceptable energy savings. There is at least one such bin, since the construction of $Histogram_{cumt}$ ensures that its "largest" bin always has the value 100%. Hence, the set of values given to the **min** operator is non-empty, and since all elements are valid, the value of $indexBin$ is valid. Line 18 determines the bin with acceptable energy savings that contains the smallest number of idle intervals. The resulting value of Bin is always valid because the **argmin** operator always returns a non-empty set of valid bins. It is not empty because the case $indexBin = 0$, which would cause **argmin** to be evaluated on an empty set, is specifically excluded by line 15. If several bins count the same number of idle intervals, then the "largest" bin is chosen.

Line 19 checks the number of intervals in the bin, and if it is sufficiently low, then there is a high chance that the finalization phase will be able to find a valid GT value inside the bin. The bin is accepted if the number of intervals is less than a value known as *dangNum*, whose calculation is explained in Section III-C. When a satisfactory bin has been found, the algorithm proceeds to the finalization phase by setting $binGT$ to $True$ in Line 20. Otherwise, in Line 22, the value of $demCumIdleTime$ is decreased by 5%, and the loop is repeated. It is possible, though in practice unlikely, that no bin has less than $dangNum$ idle intervals, in which case, line 12 causes the algorithm to fail.

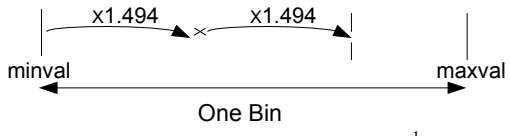
Algorithm 1 Automatized Detection of Grouping Threshold (ADGT) value

Input: *firstIdleIntervals*, *numberIdleIntervals*, *demCumIdleTime*, α
Output: Grouping Threshold (GT).

▷ Configuration and Build-Histogram Phase
1: **skip** $firstIdleIntervals$
2: $idleIntervals =$ **collect** $numberIdleIntervals$
3: $nBin = \lfloor\sqrt{numberIdleIntervals}\rfloor$
▷ Use Coarse Grain Log-Scale (20μs, 1s)
4: $dangNum = \lfloor\alpha \times (28.6 \div nBin)\rfloor$
5: $Histogram_f =$ **freqhistogram**$(idleIntervals, nBin)$
6: $Histogram_{cumt} =$ **cum-timehistogram**$(idleIntervals, nBin)$
▷ Main Phase
7: $aFind = false$
8: **do**
9: **do**
10: $binGT = false$
11: **if** $demCumIdleTime < 0.1$ **then**
12: **return None**
13: **end if**
14: $indexBin$ = **min** b
$b \in \{0, \cdots, nBin - 1\}$
$Histogram_{cumt}[b] \geq (100\% - demCumIdleTime)$
15: **if** $indexBin = 0$ **then**
16: **return None**
17: **end if**
18: $Bin =$ **max argmin** $Histogram_f[b]$
$b \in \{0, ..., indexBin - 1\}$
19: **if** $Histogram_f[Bin] < dangNum$ **then**
20: $binGT = true$
21: **else**
22: $demCumIdleTime = demCumIdleTime \times 0.95$
23: **end if**
24: **while** $(binGT\ is\ false)$
▷ Finalization Phase
25: $minval = Histogram_f.minval[Bin]$
26: $maxval = Histogram_f.maxval[Bin]$
27: $idleIntervalsInBin = \{x \in idleIntervals,$
$minval < x < maxval\}$
28: $sIdleIntervals =$ **sort**$(idleIntervalsInBin)$
29: $sIdleIntervals = [minval] +\!\!+ sIdleIntervals +\!\!+ [maxval]$
30: **for** $i \leftarrow 0$ to $len(sIdleIntervals) - 2$ **do**
31: $dist = (sIdleIntervals[i + 1] \div sIdleIntervals[i])$
32: **if** $dist \geq 1.494$ **then**
33: $aFind = true$
34: $idx = i$
35: **break**
36: **end if**
37: **end for**
38: **if** $aFind = false$ **then**
39: $demCumIdleTime = demCumIdleTime \times 0.95$
40: **end if**
41: **while** $(aFind\ is\ false)$
42: $GT = 1.1 \times sIdleIntervals[idx] \div 0.9$
43: **return** GT

Finalization Phase: When an acceptable bin has been found, the final stage is to choose the value of GT inside that bin. Lines 25 and 26 determine the shortest and longest idle interval lengths mapped to the bin. This defines the acceptable range for the GT value. Line 29 calculates a sorted list of idle interval samples, with the $minval$ and $maxval$ delimiting the endpoints of the bin. The loop on lines 30 to 37 searches for the first sufficiently large gap with no idle intervals. As shown in Figure 5, the gap between two consecutive idle interval lengths (represented with crosses) should be large enough to contain the *dangerous zone* around the GT, as well as the $\pm 10\%$ noise margin around the adjacent idle interval lengths. It is therefore necessary that the larger idle interval length

a.) with n idle intervals (n=1):



$$\left(maxval = minval \times (100.000)^{\left(\frac{1}{nBin}\right)}\right)$$

always have large enough gap if:

$$maxval \geq 1.494^{(n+1)} \times minval$$

$$1.494^{(n+1)} \leq (100.000)^{\left(\frac{1}{nBin}\right)}$$

$$n+1 \leq \left\lfloor \left(\frac{1}{nBin}\right) \frac{(\log 100.000)}{(\log 1.494)} \right\rfloor$$

$$n < \left\lfloor \left(\frac{1}{nBin}\right) \frac{(\log 100.000)}{(\log 1.494)} \right\rfloor \approx \left(\frac{28.6}{nBin}\right)$$

b.) Simplifying and introducing tolerance factor α:

$$n < \lfloor \alpha \times \left(\tfrac{28.6}{nBin}\right) \rfloor := dangNum$$

Fig. 6. Value of *dangNum* used in main phase of ADGT. α is tolerance factor, in our tests $\alpha = 4$

should be at least 1.494 times the smaller idle interval length.

If there is such a gap, then line 42 calculates the GT value, as illustrated in Figure 5, setting it to a safe distance above the shorter idle interval lengths, rather than placing it in the middle, in order to maximise the potential energy savings. If it is not possible to find such a gap, then line 39 reduces $demCumIdleTime$ by 5%. This is an error condition that returns to the main phase. If no such gap is ever found, then line 12 will eventually cause the algorithm to fail.

C. Calculation of DangNum

As remarked in the previous section, a candidate bin is only accepted if the number of idle intervals that it contains is less than a value known as $dangNum$. This value is determined as described below. Figure 6(a) shows a single histogram bin containing n idle intervals. As explained above, a value of GT can be successfully chosen whenever the ratio of two consecutive idle intervals is greater than or equal to 1.494. There is always at least one such gap whenever $maxval \geq 1.494^{n+1} \times minval$. This is illustrated in Figure 6(a), which shows that if the first n gaps are smaller than 1.494, then the last gap must be larger than this value—so they cannot all be smaller. The rest of Figure 6(a) simplifies this equation, using the correct value of the bin's width.[2] We find, however, that using this value is too restrictive, so the value is increased slightly using a tolerance factor, α, obtaining the final formula for *dangNum* shown in Figure 6(b). For example, if the sample contains 400 idle intervals, then $nBin = \sqrt{400} = 20$, so if $\alpha = 4$, then *dangNum* equals five.

[2]Referring to Figure 4, there are $nBins$ bins in the histogram, which covers the range from 17 µs at the bottom of the first bin (although the figure specifies 20 µs since intervals less than 20 µs are ignored) to 1.7 s at the top of the last bin (this is unmarked, since the last bin contains all idle intervals longer than 1 s). Hence the ratio between the upper and lower limits of each bin is given by $(1.7\,\mathrm{s}/17\,\mu\mathrm{s})^{1/nBin} = (100,000)^{1/nBin}$.

D. Restart mechanism

The restart mechanism causes the Grouping Threshold (GT) to be recalculated when doing so is likely to be beneficial. A restart is needed when the application changes phase. It is also needed when random factors cause the ADGT algorithm to choose a bad value of the GT.

As mentioned in Subsection III-A, the GT values should not be close to many idle intervals. If several idle intervals are seen within GT$\pm$10%, which is known as the *dangerous zone*, then the prediction algorithm is likely to become unstable. In order to restore the proper functioning of the prediction algorithm, it is restarted, which causes a new automatic detection of the GT. An important question is how sensitive to make the restart mechanism since, on the one hand, no energy savings are possible during restart, but on the other hand, it is worthless to continue using a bad value of GT for too long. On balance we tolerate a single idle interval in the dangerous zone, and restart if a second one appears within an interval of three times the current pattern size.

IV. Experimental Evaluation

A. Methodology

The link-level power management described in Subsection II-D is still under development by the InfiniBand switch and NIC vendor, so it is not yet supported in the devices' firmware. We therefore evaluate our self-guided prediction-based mechanism and its impact on performance and energy savings using a simulation environment. We decided to use the Venus–Dimemas [11], [12] simulator. Dimemas is an event-driven simulator, which replays a trace of the application's computation bursts and MPI activity, preserving its causal relationships and timings. Venus is a detailed network simulator, which models the complete network architecture including topology, routing, and an accurate switch/adapter model. Computation bursts are not actually performed, but represented by the time the actual computation would last. The traces of ten representative HPC workloads were collected on the MinoTauro machine. The MinoTauro machine is comprised of 126 Bull B505 compute nodes, each with two six-core Intel Xeon E5649 processors running at 2.53 GHz and with 24 GB of RAM. The workloads include the NAS Parallel Benchmarks for HPC, and applications used by academia and industry, running daily in production use at HPC facilities. The applications were configured with one MPI process per node. The system parameters used in the simulation are given in Table I.

We first fed the Venus–Dimemas simulator with unmodified traces, in order to check that the original execution times were reproduced. Next, we applied SPPS to the traces, inserting new events that mark when prediction is possible and events that mark when links are in low-power mode. When mispredictions occur, delays are inserted into the traces to model the link wake-up times. All other overheads associated with the SPPS mechanism are also inserted, including overheads originating

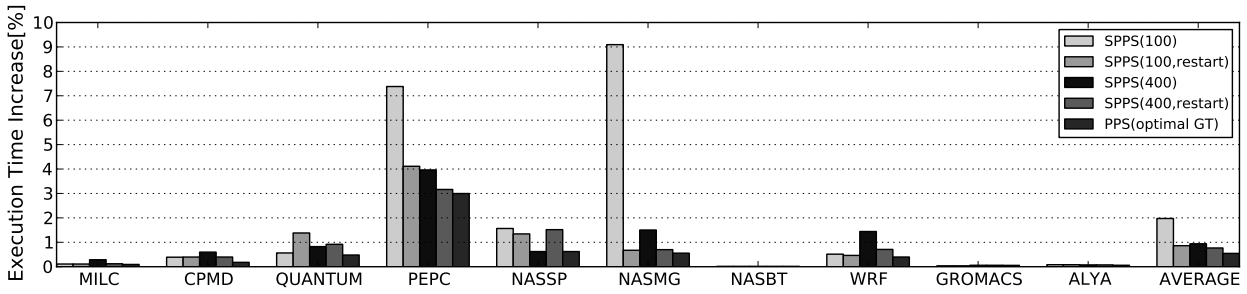


Fig. 7. Applications execution time increase

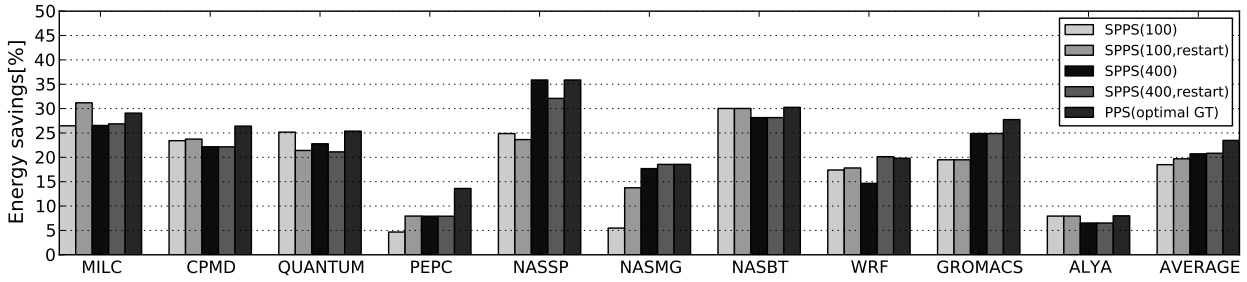


Fig. 8. Energy savings in InfiniBand edge switch links

TABLE I
PARAMETERS USED IN SIMULATIONS

Parameter	Value
Simulator	Dimemas-Venus
Connectivity	XGFT(2;18,14;1,18)
Topology	Extended Generalized Fat Trees (two levels of switches)
Switch technology	InfiniBand
Network Bandwidth	40 Gbits/s
Segment Size	2 KB
MPI latency	1 µs
CPU Speedup	1
Routing scheme	Random routing
Switch power consumption	43% when in low-power mode [10]

from Pattern Prediction Component (PPC), the overheads of data collection and the overheads spent on finding the GT. Finally, we replay the new traces on the Venus–Dimemas simulator, simulating the new structure of events in the trace and obtaining the final performance overheads and energy savings.

Using the Paraver tool [13], we measure the total amount of time for which the InfiniBand links are fully active, as well as the time that the links are in low-power mode. Energy savings are somewhat different for the various MPI processes. The times used for evaluation are averaged over all MPI processes.

The benchmarks cover a broad range of scientific workloads, with the number of MPI processes varying between 16 and 128. MILC and QUANTUM were executed with 16 MPI processes, GROMACS, ALYA, WRF and NASMG with 32 MPI processes, and NASSP and NASBT had 36 MPI processes. PEPC and CPMD had larger runs with 64 and 128 MPI processes respectively.

We measured the execution time overheads for the SPPS algorithm on a real system, by reading the system clock using the `gettimeofday()` system call. Our previous work [7] analysed in detail the execution time overheads that arise from the Pattern Prediction Component (PPC), which were on average about 1 µs to 2 µs per MPI call. We measured the additional overheads that arise from collecting the idle intervals to be far less than 1 µs. We also measured the time to calculate the GT value, following the algorithm in Section III-B, which was between 5 µs and 10 µs. We included all these overheads in the traces by increasing the lengths of the relevant computation bursts.

B. Results

This section discusses the experimental results, including a comparison with the existing PPS algorithm [7]. The results for PPS use a single best value of GT taken from a manual sweep. Since the SPPS algorithm introduced in the current paper extends the previous algorithm to automatically determine the GT value, the optimal PPS algorithm results should be seen as ideal results to try to match, rather than the current state-of-the-art to improve upon.

The main experimental results are shown in Figure 7, which gives the increase in execution time, and Figure 8, which gives the energy savings. For each benchmark we show five results. The first four results are for various configurations of the SPPS algorithm, with a sample size of 100 or 400 idle intervals, and with or without the restart mechanism. The final result, "PPS (optimal GT)", shows the best possible result from the existing PPS algorithm.

The results for a sample size of 400 have a worst case overhead of about 4% and an average overhead of less than 1% (Figure 7), both only marginally above that of PPS–Optimal. Regarding the energy savings, the average interconnect edge link energy savings is 21% (Figure 8), which is within 3% of the average from the PPS–Optimal algorithm. The difference

between a sample size of 100 and 400 idle intervals is, for most benchmarks, small, but there are two benchmarks (PEPC and NASMG) for which the smaller sample size initially chooses a bad value of the GT. These benchmarks, however, benefit from restart, which allows the algorithm to recover by calculating a new GT value, in which case it quickly chooses a good value. These benchmarks also benefit from restart when the sample size is 400, but to a lesser extent.

When comparing the dynamic SPPS algorithm against the results for the static PPS–Optimal algorithm, there are two effects to bear in mind. Firstly, the static PPS–Optimal algorithm has the advantage of using the best global value of GT. Secondly, the dynamic SPPS algorithm has the ability to dynamically adjust the GT value taking into account application phases, so long as restart is enabled. Among our benchmarks, only MILC has multiple identifiable phases, but the available benefit is small, amounting to about 5% in energy for the 100-interval case. This is the only result for which the SPPS algorithm has better energy savings than PPS–Optimal. We expect this dynamic adaptability to lead to a larger benefit when running large workloads with multiple phases.

The results show that restart should be enabled, mainly in order to avoid commiting to a poorly-chosen GT value. As already remarked, PEPC and NASMG show a large reduction in overheads, when restart is enabled, especially with a 100-interval sample. These overheads are caused by the Pattern Prediction Component (PPC), which has larger overheads than the Power Mode Control Component (PMCC), together with the large number of mispredictions. The remaining benchmarks show little difference in performance or energy, although QUANTUM and NASSP show a drop in energy savings of about 5%. The average results show that restart provides a large improvement in performance, although this effect is mostly due to the large benefit for PEPC and NASMG.

V. Related Work

Several network power-saving techniques have been proposed, most of which are hardware approaches based on the runtime behaviour at a single switch or link. Shang et al. [14] proposed a history-based dynamic voltage scaling (DVS) policy, where past network utilization is used to predict future traffic. Alonso et al. [15] propose a power-saving mechanism for regular interconnection networks built with high-degree switches and port trunking (also known as link aggregation). Each trunk link is composed of multiple links that can be individually turned on and off depending on the load. There is at least one link on, at all times, in order to maintain connectivity. Kim et al. [16] use both dynamic voltage scaling and the powering down of under-utilized links. This technique requires adaptive routing, in order to avoid deadlocks. Saravanan et al. [6] proposes an algorithm to turn links on and off using an idle period predictor, which detects repetitive behaviour at the packet level. Our technique, in contrast, is done at the application level, and before injecting data into the network. By predicting end-to-end traffic we are able to predict bursty data transmission more accurately than if prediction is done at the packet level.

Although previous work shows that hardware-based schemes can be effective, they share a common drawback that they may be to slow to adjust to sudden changes in the network traffic. Soteriou et al. [17] show that hardware-based approaches can incur large performance overheads and propose a compile-time technique, as part of a parallelizing compiler flow, that generates instructions to dynamically control network power reduction. Li et al. [18] propose another compiler-based technique, which inserts instructions to turn off communication links at the point in each loop nest when they are last used. The link is reactivated on-demand the next time it is used.

Jian Li et al. [19] are focused on non-prediction power-saving techniques. Links are powered up just before they are needed, based on hints from the built-in system events or from macros in MPI source code. Here, a separate control network is needed which is always on, in order to enable link activation messages to flow through. In our approach, we take advantage of InfiniBand links that offer a dynamic range in terms of performance and power.

In the work of Abts et al. [3], the authors propose energy-proportional datacenter networks. Link data rates are selected on the basis of traffic intensity in the network. They use the congestion sensing heuristic to sense traffic intensity, dynamically activating links as they are needed. While this work is focused on datacenter applications, which can tolerate small changes in latency, HPC applications cannot afford such performance loss.

Huang et al [20] propose a table-based traffic predictor (ATPT) which can predict the amount of data injected in the network allowing the suitable working frequency and the corresponding voltages of the links to be set in advance. The main problem lies in the unknown time interval in tracking and predicting traffic which if not chosen correctly can affect the prediction acccurancy. Also the amounts of data transmitted are quantized according to the number of discrete voltage-frequency (VF) levels implying that more adjustable voltage levels will incur more costs.

Finally, the work of Lim et al. [21] is complementary to ours, in the sense that both are power saving techniques in the MPI run-time system. Whereas our technique turns off communication links during computation periods, Lim et al. reduces CPU power consumption during the communication phases. The run-time system identifies communication regions and adjusts the processor frequency to minimize the energy–delay product, without needing profiling or training.

VI. Conclusions

One of the biggest challenges in high-performance computing is to reduce the power and energy consumption. Significant improvements in energy efficiency are required across all subsystems, and, as processors and memory have become more energy efficient, attention is turning to the interconnect. Previous research has shown that although HPC systems require a high-performance interconnect, the average utilisation is low, implying a large benefit from power-saving mechanisms aimed at energy proportionality.

This paper proposes the Self-Tuned Pattern Prediction System (SPPS), an automatic software-directed mechanism for interconnect link energy proportionality. SPPS automatically recognises and exploits repetitive patterns in the application's communication behaviour. It is implemented in the MPI library, which allows all MPI programs to benefit, without needing changes to their source code.

Our automatic mechanism tries to identify and predict those link idle intervals where the majority of idle link power is spent. Discovering the critical idle interval length, defined by the Grouping Threshold (GT) value, allows to filter out irrelevant idle intervals in the communication patterns. We propose to detect GT for an application process based on its characteristics. We analyze and find the GT value at runtime without knowing any details about the application. We also show that the SPPS system is capable of adjusting to different application regions selecting the new customized GT for each region, while keeping the overheads at the minimum.

We evaluate SPPS using an event-driven simulator with traces from a production supercomputer. When compared to baseline unmanaged execution, our simulations show average energy savings in the network edge links of up to 21%, with an average execution time penalty of less than 1%.

The obtained results show that our runtime self-tuned algorithm can work cooperatively with the on-the-fly dynamic management. Experimental results confirm the effectiveness of our pattern prediction approach on large number of applications and benchmarks. Therefore, we believe that our solution, with its significant energy-saving potential and low performance overheads, can accelerate the adoption of runtime power management techniques in HPC systems.

VII. Acknowledgements

This work has been supported by the Spanish Severo Ochoa award SEV-2011-00067 and project TIN2012-34557 from the Ministry of Science and Innovation, by the Generalitat de Catalunya 2014-SGR-1051 award and finally Swedish Foundation for Strategic Research (SSF) under grant number 37252706 as part of the project SCHEME. Branimir Dickov was supported by Catalan Government with Pre-doctoral scholarship FI (reference no. 2009FI-B00077).

References

[1] T. Hoefler, "Software and hardware techniques for power-efficient hpc networking," *Computing in Science Engineering*, vol. 12, no. 6, pp. 30–37, 2010.

[2] P.M.Kogge, "Architectural challenges at the exascale frontier(invited talk)," *Simulating the Future: Using One Million Cores and Beyond*, 2008.

[3] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 338–347, Jun. 2010.

[4] "Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 3: Csma/cd access method and physical layer specifications amendment 5: Media access control parameters, physical layers, and management parameters for energy-efficient ethernet," *IEEE Std 802.3az-2010 (Amendment to IEEE Std 802.3-2008)*, pp. 1–302, Oct 2010.

[5] K. Saravanan, P. Carpenter, and A. Ramirez, "Power/performance evaluation of Energy Efficient Ethernet (EEE) for high performance computing," in *ISPASS*, 2013, pp. 205–214.

[6] K. P. Saravanan, P. M. Carpenter, and A. Ramirez, "A performance perspective on energy efficient hpc links," in *Proceedings of the 28th ACM international conference on Supercomputing*. ACM, 2014, pp. 313–322.

[7] B. Dickov, M. Pericàs, P. Carpenter, N. Navarro, and E. Ayguadé, "Software-managed power reduction in infiniband links," in *Parallel Processing (ICPP), 2014 43rd International Conference on*. IEEE, 2014, pp. 311–320.

[8] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Comput. Linguist.*, vol. 18, no. 4, pp. 467–479, Dec. 1992.

[9] J. Y. Kim and J. Shawe-taylor, "Fast string matching using an n-gram algorithm," *Software - Practice and Experience*, vol. 24, pp. 79–83, 1994.

[10] "Power savings features in mellanox products," Mellanox, Sunnyvale, CA, USA, january 2013, http://www.mellanox.com/related-docs/whitepapers/.

[11] C. Minkenberg and G. Rodriguez, "Trace-driven co-simulation of high-performance computing systems using omnet++," in *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, ICST, Brussels, Belgium, 2009.

[12] J. Labarta, S. Girona, V. Pillet, T. Cortes, and L. Gregoris, "Dip: A parallel program development environment," in *Euro-Par'96 Parallel Processing*. Springer Berlin Heidelberg, 1996, pp. 665–674.

[13] "Performance tools," Barcelona Supercomputing Center, Barcelona, Spain, 2014, http://www.bsc.es/computer-sciences/performance-tools/.

[14] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *HPCA*, 2003.

[15] M. Alonso, S. Coll, J.-M. Martínez, V. Santonja, P. López, and J. Duato, "Power saving in regular interconnection networks," *Parallel Comput.*, vol. 36, no. 12, pp. 696–712, Dec. 2010.

[16] E. J. Kim, K. H. Yum, G. M. Link, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, M. Yousif, and C. R. Das, "Energy optimization techniques in cluster interconnects," in *ISLPED 2003*. New York, NY: ACM, pp. 459–464.

[17] V. Soteriou, N. Eisley, and L.-S. Peh, "Software-directed power-aware interconnection networks," *ACM Trans. Archit. Code Optim.*, vol. 4, no. 1, Mar. 2007.

[18] F. Li, G. Chen, M. Kandemir, and M. Karakoy, "Exploiting last idle periods of links for network power management," in *EMSOFT*. New York, NY, USA: ACM, 2005, pp. 134–137.

[19] J. Li, W. Huang, C. Lefurgy, L. Zhang, W. Denzel, R. Treumann, and K. Wang, "Power shifting in thrifty interconnection network," in *HPCA*, feb. 2011.

[20] Y.-C. Huang, K.-K. Chou, and C.-T. King, "Application-driven end-to-end traffic predictions for low power noc design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 2, pp. 229–238, Feb 2013.

[21] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs," in *SC*, New York, NY, 2006.