Software-Managed Power Reduction in Infiniband Links

Branimir Dickov*†, Miquel Pericàs‡, Paul M. Carpenter*, Nacho Navarro*†, Eduard Ayguadé*†
* Barcelona Supercomputing Center
†Universitat Politècnica de Catalunya, BarcelonaTech
‡ Tokyo Institute of Technology
{branimir.dickov, paul.carpenter, nacho.navarro, eduard.ayguade}@bsc.es
pericas.m.aa@m.titech.ac.jp

***Abstract*—The backbone of a large-scale supercomputer is the interconnection network. As compute nodes become more energy-efficient, the interconnect is accounting for an increasing proportion of the total system energy consumption. The interconnect's energy consumption is, however, only starting to receive serious attention. Some hardware-based schemes have been proposed that exploit idle periods or low utilisation, either by turning off the links or by lowering the frequency and voltage. Although these schemes are effective in certain cases, they do not have enough global information about the application's communication behaviour to efficiently manage the network power consumption. This paper proposes an alternative approach: moving the intelligence into the PMPI layer of the MPI library, and using prediction to discover repetitive patterns in the application's communication behaviour.**

The core of the prediction algorithm is an *n*-gram extraction technique, which can accurately predict not only when a link will become unused but also when it will become active again, allowing lanes to be switched off during the idle periods and switched back on again in time to avoid incurring a significant performance degradation. Many HPC applications benefit from prediction, since they have repetitive computation and communication phases. By implementing the energy-saving mechanism inside the MPI library, existing MPI programs do not need to be modified. Using an event-driven simulator, driven by representative HPC workloads, we demonstrate average energy savings in Infiniband switches up to around 33%, while the average execution time increase is only up to 1%.

I. Introduction

High-Performance Computing (HPC) is a crucial tool for modern science. There is a constant need for more powerful supercomputers, but increasing performance is leading to excessive peak power demand and total energy consumption. While supercomputers have traditionally been ranked only by performance, now that power and energy are first-order design constraints, systems are also being ranked on energy efficiency [1]. An important characteristic of energy-efficient system components is energy proportionality, which means that energy consumption depends linearly on utilization.

The system's interconnect accounts for an important fraction of its total energy consumption. Nevertheless, although a significant effort has been invested into achieving energy proportionality of processors and memory, similar techniques in networks have not reached wide adoption. 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]. Outside HPC, where data centre processors often have low utilisation, this fraction can reach 50% [3]. Most of this power consumption is due to the interconnection links. For example, the links in an IBM eight-port Infiniband 12$\times$ switch consume 64% of the switch power [4]. High-performance interconnect links are, however, not energy proportional, since their power consumption is always near peak, whether or not they are actually being used for message transmission.

One approach to reduce network energy consumption is to put the links into low-power mode when they are not being used. The problem is that link state changes, from off to active, can take up to 10μs [5]. Since state changes add to the latency of MPI messages, and many HPC applications are highly sensitive to latency, this leads to an unacceptable loss in performance. An alternative is to lower the voltage and bandwidth of links when utilization is low, which has faster link reactivation, at about 100ns, but the potential power saving is much lower [3]. Both mechanisms switch between power modes using low-level hardware schemes [6], [7], [8]. Common drawbacks are the inability to capture significant energy savings, as well as an unknown and uncontrollable performance penalty.

Most HPC applications follow the bulk synchronous programming paradigm, in which application processes are synchronised, either all performing computation at the same time or all involved in communication. In general, application developers view the time spent in communication as overhead, and therefore try to minimize it. This leads to high peak bandwidth demand and latency sensitivity, but low average utilisation, which, as explored in the following section, provides significant opportunities for energy savings. Unfortunately, as mentioned above, current interconnects are not energy proportional, so the potential energy savings are lost.

The majority of execution time in most HPC applications is spent in a large number of iterative execution phases. Since the communication pattern inside each phase is essentially the same, it is possible to observe the communication behaviour in one iteration, and use the knowledge gained to predict the behaviour of the subsequent iterations. Specifically, this means detecting the patterns of MPI calls that are repeating within each MPI process. To achieve this, we use an algorithm based on *n*-gram extraction techniques, a widely used concept from statistical natural language processing [9], [10]. Our Pattern Prediction Algorithm (PPA) allows an on-the-fly detection

TABLE I
DISTRIBUTION OF LINK IDLE INTERVALS WITH STRONG SCALING

	N° proc	**$T_{\text{idle}} < 20\mu s$**			**$20\mu s < T_{\text{idle}} < 200\ \mu s$**			**$T_{\text{idle}} > 200\mu s$**		
		N° intervals	Intervals [%]	Time [%]	N° intervals	Intervals [%]	Time [%]	N° intervals	Intervals [%]	Time [%]
GROMACS	8	3277	58.56	0.001	6	0.11	0.009	2313	41.33	99.99
	16	3595	54.98	0.002	606	9.27	0.078	2338	35.75	99.92
	32	5052	53.72	0.006	1523	16.2	0.304	2829	30.08	99.62
	64	5046	53.68	0.011	2228	23.7	0.779	2126	22.62	99.21
	128	9067	68.5	0.11	2276	17.2	1.01	1893	14.3	98.88
ALYA	8	771	22.57	0.024	82	2.4	0.006	2563	75.03	99.97
	16	1744	34.08	0.013	811	15.85	0.077	2563	50.08	99.91
	32	3642	58.52	0.07	818	13.14	0.99	1763	28.33	98.94
	64	6754	71.97	0.27	827	9.29	0.9	1758	18.73	98.83
	128	8497	76.72	0.4	1644	14.84	5.05	934	8.43	94.55
WRF	8	209357	94.31	0.05	2201	0.99	0.14	10419	4.69	99.81
	16	209423	94.34	0.11	2051	0.92	0.26	10503	4.73	99.63
	32	209414	94.34	0.3	4014	1.81	0.73	8549	3.85	98.97
	64	209284	94.28	1.07	5050	2.28	1.48	7643	3.44	97.45
	128	209442	94.36	1	6697	3.02	0.51	5833	2.63	98.49
NAS BT	9	9664	78.63	0.009	9	0.07	0.001	2618	21.3	99.99
	16	13286	77.63	0.022	5	0.03	0.008	3824	22.34	99.97
	36	20522	76.68	0.031	5	0.02	0.009	6236	23.3	99.96
	64	27750	76.21	0.094	13	0.04	0.006	8648	23.75	99.9
	100	34996	75.98	0.13	161	0.35	0.22	10902	23.67	99.65
NAS MG	8	5468	54.66	0.095	3794	37.92	3.055	742	7.42	96.85
	16	5119	54.85	0.18	3729	39.96	5.87	484	5.19	93.95
	32	5503	58.7	0.46	3600	38.4	11.38	271	2.89	88.16
	64	5775	60.79	0.97	3458	36.4	8.37	267	2.81	90.66
	128	7082	84.65	7.04	1123	13.42	6.71	161	1.92	86.25

of consecutive repeatable MPI communication patterns. This provides a high-level view of the order and timing of link usage, which allows the transition between different link power modes to be made with minimum impact on performance. We apply the algorithm to recently announced power-saving features for Infiniband (IB) switches, reducing link power consumption without losing network connectivity. This feature works by shutting down all but one lane of an 4×IB link.

The proposed energy-saving mechanism consists of two parts. The first part detects consecutive repeatable patterns in MPI communication, leading to successful prediction. The second part uses this prediction to control the shifting between link power modes. Both parts are executed in the PMPI profile layer of MPI, eliminating the need for user involvement, but enabling customisation by the system integrator and by HPC operations. Existing MPI programs benefit from energy savings without needing any source code modification.

Specifically, this paper makes the following contributions:

- We demonstrate a large potential to save energy in the interconnection network. The majority of HPC workloads we tested have long link idle times, allowing overheads in switching power modes to be amortised by large energy savings. We also point out how newly-announced features of IB switches will allow power savings without loss of network connectivity.
- We show that, for the studied HPC applications, the PPA algorithm can successfully exploit patterns in MPI communication. We measure prediction accuracy of up to 98%. We also provide a complete description of our power saving mechanism, enabling it to be run within the PMPI layer of MPI.
- We evaluate our energy-saving mechanism using an event-driven simulator and traces obtained from a production run on a real supercomputer. Results show an average reduction in IB switch energy consumption of up to 33%, compared with the power-unaware scheme where links are "always-on". We also show there is no significant increase in execution time. In particular, the worst average increase was around 1%.

The rest of this paper is structured as follows. Section II provides the motivation and necessary background to understand switching of IB links to low-power mode during iterative computation phases. Section III introduces the design of our link power saving mechanism and the PPA algorithm. 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*

As discussed above, HPC applications typically follow the bulk synchronous programming model, in which network traffic is concentrated into distinct communication phases. It is reasonable to expect that, since the network links are idle during computation phases, there is an automatic opportunity to enter power-saving mode. It is, however, important to take account of the overhead in changing power mode, which is approximately $10\mu s$ [5]. There can be no energy savings from idle periods that are shorter than the total time to turn the link off then back on again. A significant energy saving is only possible if the idle period is much longer than this overhead. For simplicity in exposition, we assume that the time to turn the link off is the same as the time to turn it back on again, and therefore denote both by T_{react}. In summary, energy savings are only possible for idle periods with $T_{\mathrm{idle}} > 2 \times T_{\mathrm{react}}$.

We evaluated the potential for link power reduction by analysing traces of typical HPC applications (Gromacs [11], Alya [12], WRF [13] and two NAS Parallel Benchmarks [14]) running on a production machine. The machine is based on Bull B505 nodes, each with two 6-core Intel Xeon E5649 processors running at 2.53GHz and with 24GB of RAM. We configured the applications to use one MPI process per processor. We used strong scaling, in which the same workload was used irrespective of the number of processors.

The results are shown in Table I. We see that, for almost all applications, 99% of the link idle time is inside idle intervals that are longer than 20μs, which is twice the typical value of T_{react}. Even more importantly, in the majority of cases, more than 90% of the total link idle time is in longer idle intervals of duration $T_{\mathrm{idle}} > 200\mu$s, where significant power can be saved. Since the goal is a reduction in operational costs over the lifetime of the supercomputer, the important consideration is average potential energy savings over all applications. Only the NAS MG benchmark when running with a large number of processes, has a figure lower than 90%. All the results in this paper are for the pessimistic case of strong scaling. Better results are expected for weak scaling. Nevertheless, although, for strong scaling, the number of short intervals ($T_{\mathrm{idle}} < 20\mu$s) rises with the number of MPI processes, short intervals still contribute a small proportion of the total idle time. Since long idle intervals account for most of the idle time, reducing link power only during the long idle intervals is sufficient to obtain most of the potential energy savings, resulting in close to energy proportionality.

While deactivating IB lanes can be overlapped with computation, reactivation may incur a latency in subsequent communication. In an ideal case, the IB link lanes would be turned on in time to avoid a latency penalty on the next message. We solve this problem by providing the necessary knowledge using a prediction algorithm.

B. *Network power management support on IB switches*

Mellanox has recently developed 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. Port link width reduction is done using a method called Width Reduction Power Saving (WRPS). For example, using WRPS a 40 Gb/s 4× QDR port can run as 10 Gb/s 1× 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) [15]. We use this published value of 43% in the evaluation section as the power consumption of an IB switch in low-power mode.

III. Design

This section describes our energy-saving mechanism, which reduces link power consumption during idle periods, with negligible impact on execution time. Figure 1 is a high-level view of our proposal, which consists of two parts. The first

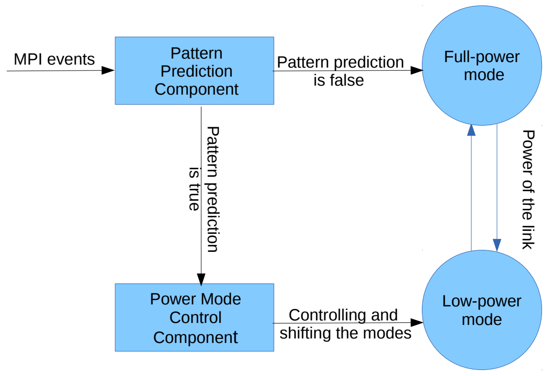

MPI events
Pattern Prediction Component
Pattern prediction is false
Full-power mode
Pattern prediction is true
Power of the link
Power Mode Control Component
Controlling and shifting the modes
Low-power mode

Fig. 1. Simplified diagram of MPI process pattern prediction system that reduces power consumption in interconnection links

part, the Pattern Prediction Component (PPC), is invoked before every MPI event. It contains the Pattern Prediction Algorithm (PPA), described in detail below, which builds a table, known as the *pattern list*. This table contains repeatable MPI communication patterns. It also generates an output flag, *patternPrediction*, which is true whenever PPA has determined that the program is following a known repeatable pattern. If *patternPrediction* is false, then no prediction is active, meaning that the link remains in full-power mode.

When *patternPrediction* is true, however, control of the link's power modes is transferred to the second part, the Power Mode Control Component (PMCC). Whenever this 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 expected long idle intervals, 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 PPA, 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, PMCC sets *patternPrediction* to false. In that case, PPA is reactivated and the link is kept in full-power mode until the next repeatable pattern.

A. *Pattern Prediction Component*

The algorithm uses the concept of *n*-grams, which is extensively used in the area of natural language processing. The *n*-gram extraction approach has been used to efficiently detect DNA patterns [16] and patterns in musical notes [17]. An *n*-gram is defined to be a subsequence of *n* items in a sequence. In our case, the sequence of items, known as grams, is derived from the MPI events in the program's execution. Each *gram* is one or more consecutive MPI events that are separated only by short idle intervals, whereas the idle intervals between different grams are long. An n-gram is a sequence of n consecutive grams. Note that PPA works on the MPI events in a single process. Although it is outside the scope of this paper, if there are multiple MPI processes per node, prediction should be done inside each MPI process separately, with their outputs combined using a single PMCC per node.

Before the PPA algorithm is invoked, the grams need to be formed. Algorithm 1 performs the grouping of MPI events

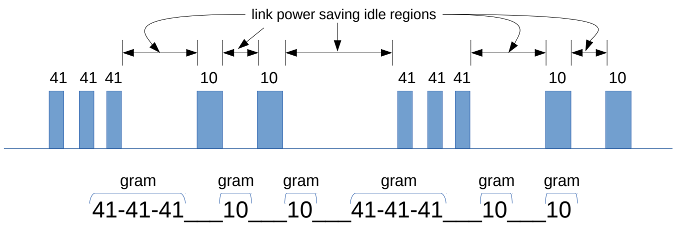


Fig. 2. Forming the array of grams from the MPI event stream (Alya). Event IDs are 41 for *MPI_Sendrecv* and 10 for *MPI_Allreduce*

Algorithm 1 Forming Array of Grams

```
Input: Current MPI event, eventType, and length of preceding idle interval, previousIdleTime.
Output: Predicted pattern of MPI events, predictedPattern, and the current partial gram, currentGram, required by Algorithm 3.
1: if $previousIdleTime > groupingThreshold$ then
                                   ▷ Insert completed gram into array
2:     $array[pos] = (currentGram, previousIdleTime)$
3:     $pos = pos + 1$
4:     clear $currentGram$
5: end if
6: append $eventType$ to $currentGram$
7: if $patternPrediction\ is\ false$ then
8:     if $pos \geq (posNext + patSize)$ then
9:         $call\ PPA()$
10:    end if
11: end if
```

into grams, based on the idle time interval between adjacent MPI events. Two consecutive MPI events are considered to be part of the same gram whenever the idle time separating them is less than a threshold known as the *grouping threshold* (GT). The intention is that the link enters low-power mode between grams but not inside them, so this grouping threshold should be larger than the critical value of $2{\times}T_{\text{react}}$ discussed in Section II-A.

The input to the algorithm is the current MPI event type, *eventType*, and the length of the idle time preceding it, *previousIdleTime*. The output of the algorithm is the predicted pattern of MPI events, *predictedPattern*, and the current partial gram, *currentGram*, required by PMCC.

Here, an array, *array*, of tuples is created. Each tuple in this array holds the list of MPI events in the current gram, as well as the length of the idle interval that *follows* the gram. Note that the current gram can only be inserted into *array* when this latter length is known; i.e. after the idle interval following it, which is on the first MPI event of the next gram.

Figure 2 illustrates the effect of Algorithm 1. Each set of three consecutive *MPI_Sendrecv* calls is grouped together to form a single gram, while each *MPI_Allreduce* call is isolated as a separate gram. These grams will be used as building blocks to construct the repeatable communication patterns. The building of patterns is done by the PPA algorithm, which is invoked on line 9, only when there is no currently repeating pattern and a sufficient number of grams has been seen.

B. Pattern Prediction Algorithm

A *repeating pattern* is a sequence of grams that has been observed to occur at least twice consecutively. We established the following policy to discover these repeating patterns and accurately predict their continuation:

- After observing three consecutive occurrences of the same pattern, it is predicted to continue to repeat for a long time, meaning that the Power Mode Control Component is activated.
- On misprediction, the Power Mode Control Component is deactivated. However, observing the pattern once more causes it to be detected, meaning that the Power Mode Control Component is reactivated.

This policy is implemented by Algorithm 2, the Pattern Prediction Algorithm (PPA). It is based on an algorithm proposed by Alawneh for the detection of process patterns [18]. We modified the algorithm to adapt it to detect continuous repetitions of patterns in program execution and the prediction of pattern appearance based on previous appearances.

The input to the PPA algorithm is the array of tuples, $array$, from Algorithm 1. Each tuple in the array corresponds to a completed gram, holding the list of MPI events inside it, as well as the length of the idle interval that follows it. The PPA algorithm builds a $uthash$ [19] hash table, known as the $pattern\ list$, with key the pattern sequence (list of grams) and value a tuple giving the pattern's length, its positions in the array, its frequency, the list of idle intervals between grams and the total number of MPI calls in the sequence. In addition, there are two indices into $array$, $posCur$, initially zero, which points to the current pattern, and $posNext$, initially equal to $patSize$, which starts with the value two.

The PPA algorithm is best understood using an example. Figure 3 illustrates the execution of the algorithm for the Alya workload. At the top, in Figure 3(a), is the list of MPI events grouped into grams; it is an extension of the example in Figure 2. Next, in subfigure (b), is shown the progress of the algorithm, with each row corresponding to an MPI event. For simplicity, the lengths of the idle intervals have been omitted from $array$. At the bottom, in Figure 3(c) is shown the insertions into the pattern list.

We now follow the progress of the algorithm in Figure 3(b). The PPA algorithm will not be executed until there are sufficient completed grams in $array$ (line 9 of Algorithm 1). Since the initial values of *patSize* and *posNext* are both two, the number of formed grams becomes large enough only on the ninth MPI call (line 9 in the PPA execution in Figure 3(b)). At this point, since *newPattern* is true and *checkConsec* is false, the only action is to insert the current gram into the pattern list (lines 48 to 50). The first bi-gram, 41-41-41_10, is therefore read from $array$, and added to the pattern list (lines 48 and 49). This insertion is shown in Figure 3(c). The return value from $updatePL$ indicates whether this is the first insertion of that particular pattern sequence. It is, so $newPattern$ is true.

On the next MPI event, $newPattern$ and $checkConsec$ are both true, so the first action is to check whether there are two consecutive identical patterns in the array (line 23). The comparison is between the bi-grams 41-41-41_10 and 10_41-41-41 at the beginning of the array. These do not match, so control passes to lines 36 to 40, where $checkConsec$ becomes false, and both $posCur$ and $posNext$ are shifted one position. On the 11th MPI call, the second bi-gram 10_10 is added to the pattern list, in a similar manner to the first. On the 13th MPI call, the third bi-gram is added.

Algorithm 2 Pattern Prediction Algorithm (PPA): Algorithm runs for each MPI process separately identifying consecutive repeating patterns on which basis the prediction is done

```
1: if $newPattern\ is\ false$ and $checkConsec\ is\ false$
2:     and $patSize < maxPatternSize$ then
                                        ▷ Grow the pattern by one gram
3:     $checkPrevious = true$
4:     $match = false$
5:     $nextPattern = currPattern + array[posCur + patSize]$
6:     $patSize = patSize + 1$
7:     $newPattern = updatePL(nextPattern, curPos)$
8:     if $newPattern\ is\ false$ then
9:         if $PL[nextPattern].wasUsed\ is\ true$ then
                                        ▷ Reactivate previously used pattern
10:            $predictedPattern = nextPattern$
11:            $patternPrediction = true$
12:            return
13:        end if
14:    end if
15:    Check all previous positions listed in $PL[currPattern]$
16:    if All can be extended to match $nextPattern$ then
17:        $match = true$
18:    end if
19:    $checkConsec = true$
20:    $currPattern = nextPattern$
21: end if
22: if $checkConsec\ is\ true$ then
                                        ▷ Check whether the pattern is repeated at $posNext$
23:    if $array[posNext : posNext + patSize]$ Equals $currPattern$ then
24:        $consecutiveRepeats{+}= 1$
25:        $UpdatePL(currPattern, posNext)$
26:        $match = true$
27:        if $consecutiveRepeats$ Equals 2 then
28:            $maxPatternSize = patSize$
29:            $PL[currPattern].wasUsed = true$
30:            $predictedPattern = currPattern$
31:            $patternPrediction = true$
32:        end if
33:        $posCur = posCur + patSize$
34:        $posNext = posNext + patSize$
35:    else
36:        $checkConsec = false$
37:        if $newPattern\ is\ true$ and $match\ is\ false$ then
38:            $posCur = posCur + (patSize - 1)$
39:            $posNext = posNext + (patSize - 1)$
40:        end if
41:    end if
42:    if $match\ is\ false$ and $checkPrevious\ is\ true$ then
43:        remove $nextPattern$ from $PL$
44:        $patSize = 2$
45:        $checkPrevious = false$
46:    end if
47: else                                    ▷ Insert current gram into pattern list
48:    $currPattern = array[posCur : posCur + patSize]$
49:    $newPattern = updatePL(currPattern, posCur)$
50:    $checkConsec = true$
51: end if
```

(a) Array of grams of MPI events formed during an MPI process:

0	1	2	3	4	5	6	7	8
41-41-41	10	10	41-41-41	10	10	41-41-41	10	10
41-41-41	10	10	41-41-41	10	10	41-41-41	10	10
9	10	11	12	13	14	15	16	17

(b) PPA execution:

#	MPI ID	Array of grams	Current Gram	Action on MPI event	Pattern prediction
1	41		41	Not enough grams	false
2	41		41-41	Not enough grams	false
3	41		41-41-41	Not enough grams	false
4	10	[41-41-41]	10	Not enough grams	false
5	10	[41-41-41,10]	10	Not enough grams	false
6	41	[41-41-41,10,10]	41	Not enough grams	false
7	41	[41-41-41,10,10]	41-41	Not enough grams	false
8	41	[41-41-41,10,10]	41-41-41	Not enough grams	false
9	10	[41-41-41,10,10,41-41-41]	10	Add pattern to PL	false
10	10	[41-41-41,10,10,41-41-41,10]	10	Check consecutive-no	false
11	41	[41-41-41,10,10,41-41-41,10,10]	41	Add next pattern to PL	false
12	41	[41-41-41,10,10,41-41-41,10,10]	41-41	Check consecutive-no	false
13	41	[41-41-41,10,10,41-41-41,10,10,]	41-41	Add next pattern to PL	false
14	10	[41-41-41,10,10,41-41-41,10,10,41-41 -41]	10	Check consecutive-no	false
15	10	[41-41-41,10,10,41-41-41,10,10,41-41 -41,10]	10	Add next pattern to PL- match detected	false
16	41	[41-41-41,10,10,41-41-41,10,10,41-41 -41,10,10]	41	Check consecutive-no	false
17	41	[41-41-41,10,10,41-41-41,10,10,41-41 -41,10,10,]	41-41	Add gram Consecutive-yes	false
18	41	[41-41-41,10,10,41-41-41,10,10,41-41 -41,10,10]	41-41-41	Not enough grams	false
19	10	[41-41-41,10,10,41-41-41,10,10,41-41 -41,10,10, 41-41-41]	10	Not enough grams	false
20	10	[41-41-41,10,10,41-41-41,10,10,41-41 -41,10,10, 41-41-41,10]	10	Not enough grams	false
21	41	[41-41-41,10,10,41-41-41,10,10,41-41 -41,10,10, 41-41-41,10,10]	41	Check consecutve-yes	true

(c) Insertions into Pattern List:

#	pattern	frequency	idle intervals	position	pattern size	Nº MPI calls
9	41-41-41_10	1	t_1,t_2	0	2	4
11	10_10	1	t_1,t_2	1	2	2
13	10_41-41-41	1	t_1,t_2	2	2	4
15	41-41-41_10	2	t_1,t_2	0, 3	2	4
17	41-41-41_10_10	1	t_1,t_2,t_3	3	3	5
17	41-41-41_10_10	2	t_1,t_2,t_3	3, 6	3	5
21	41-41-41_10_10	3	t_1,t_2,t_3	3, 6, 9	3	5

(d) Prediction possible:

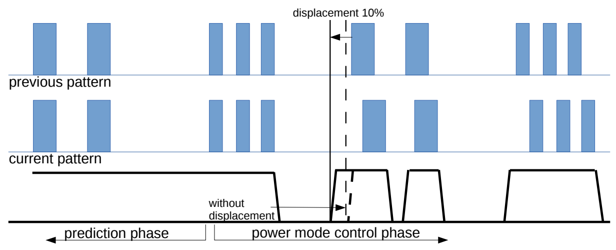
predicted pattern	idle intervals	from position
41-41-41_10_10	[t_1,t_2,t_3]	12

Fig. 3. Example execution of the PPA algorithm for Alya workload

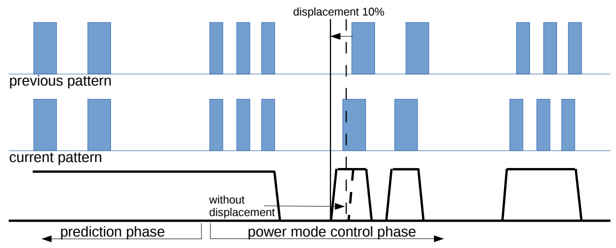
On the 15th MPI event, the 41-41-41_10 bi-gram is encountered for a second time. Since it was already present in the pattern list, *newPattern* is set to false (line 49). Inside *updatePL*, the frequency count, shown in the third column in the insertions list in Figure 3, is increased to two and the list of positions is extended to be [0, 3]. Next, on the 16th MPI event, *newPattern* and *checkConsec* are both true, but, as before, there is no consecutive repeat of the bi-gram 41-41-41_10. Therefore, *checkConsec* is set to false, but, as now $newPattern$ is set to false, it is necessary first to check whether the enlarged pattern can detects its repetitions, before we shift both indices by the $patSize - 1$.

On the 17th MPI event, *newPattern* is still false, and *checkConsec* is now false, since the sequence of grams, 41-41-41_10 has been seen twice, but they are not consecutive. For this reason, the algorithm increases the size of this pattern by one gram (lines 3 to 14); in this case, to the tri-gram 41-41-41_10_10. If this pattern had previously been used for prediction, then it would be immediately reactivated (lines 9 and 11), according to the second statement in the policy at the beginning of this section. This is not the case, so instead, line 15 checks whether all previous occurrences of the bi-gram 41-41-41_10 can be extended to the new tri-gram. If the newly constructed tri-gram cannot be detected at any previous position of its prefix bi-gram and there's no consecutive repeats, than it will be removed from the pattern list (line 43) and the size of a n-gram will be set to the minimal value, 2 (bi-gram). Here, it is not the case, so $match$ is set to true.

Eventually, on the 17th call, the first consecutive repetition of the tri-gram 41-41-41_10_10 is found. At this point, $consecutiveRepeats$ is incremented to 1, and both $posCur$ and $posNext$ are advanced by the pattern size. When PPA is next invoked on the 21st MPI event, the second consecutive repeat is seen. The pattern is assigned to $predictedPattern$ and $patternPrediction$ is set to true (lines 28 to 31), since the PPA algorithm has successfully found the repeating pattern.



(a) Real idle interval turned out to be larger than expected



(b) Real idle interval turned out to be shorter than expected

Fig. 4. Controlling IB link power mode during execution of Alya, with *displacement factor* of 10%

Algorithm 3 Power Mode Control Component

```
Input: Predicted pattern, predictedPattern and the current (partial) gram,
    currentGram
1: $index : Integer(0)$
2: if $patternPrediction\ is\ true$ then
3:     $predictedPatternGram = predictedPattern[index]$
4:     $idleTimeArray = loadFromPL(predictedPattern)$
5:     if $len(currentGram)$ Equals $len(predictedPatternGram)$ then
6:         if $currentGram$ Equals $predictedPatternGram$ then
7:             $idleTime = idleTimeArray[index]$
8:             $safetyLimit = idleTime \times displacementF + T_{\mathrm{react}}$
9:             $predictIdleTime = idleTime - safetyLimit$
10:            $WRPS\_method(predictedIdleTime)$
11:            $index = (index + 1) \bmod len(predictedPattern)$
12:        else
13:            $patternPrediction = false$
14:            $index = 0$
15:        end if
16:    end if
17: end if
```

In order to recognize the natural (real) iteration in the application and predict each iteration based on the behaviour of the previous one, we must avoid merging multiple application iterations into a single pattern. This is done by setting the maximum pattern size to be the length of the current pattern (line 28). If this were not done, and increasing numbers of application iterations were combined into a single pattern, prediction accuracy would suffer, since idle intervals would be predicted based on older values from many iterations previously. The pattern size can therefore vary from the smallest bi-gram to the size defined by $maxPatternSize$ value.

C. Power Mode Control Component

The Power Mode Control Component is responsible for switching between link power modes, according to the current repeatable pattern. The algorithm is presented as Algorithm 3, which is executed only when *patternPrediction* flag is true. The first input to the algorithm is the predicted pattern from Algorithm 2. This pattern is described by two arrays. The first array is the sequence of grams, *predictedPattern*, and the second array is the sequence of idle time intervals following those grams, *idleTimeArray*. The other input to the algorithm is the current gram being built by Algorithm 1.

Algorithm 3 works with the partial gram, and considers it to be complete when it has the correct length (line 5). If, in addition, the MPI events in the actual gram match the prediction (line 6), then the predicted length of the upcoming idle period, *idleTime*, is read from the array. It is modified by the *displacement factor*, as described below, and the T_{react}, obtaining the final prediction, *predictIdleTime*. The resulting value can be passed as the argument to *WRPS_method*, giving the time to remain in low-power mode. If, on the other hand, the actual gram does not match the prediction, then the current pattern has finished, and PPA is reactivated by setting the $patternPrediction$ flag to $false$.

The displacement factor, mentioned above, is a safety factor, used to take account of variability in the link idle intervals. To reduce the likelihood that the link is not turned on too late, the predicted idle time is reduced using the displacement factor (line 8 of Algorithm 3). It is a value between 0 and 1, where 0 means that the predicted idle time is not reduced, and 1 means that it is reduced all the way to zero. For simplicity in presentation, the displacement factor is expressed as a percentage (so a displacement factor of 5% is equivalent to a value of 0.05 in the algorithm).

The function of the displacement factor is illustrated in Figure 4. Figure 4(a) is the case when the current pattern has an idle interval slightly larger than predicted. In this case, a displacement factor of 10% reduces the energy savings by slightly more than 10%, compared with optimal. Figure 4(b) is the case when the current pattern has an idle interval shorter than predicted. In this case, the displacement factor of 10% has avoided the latency penalty that would have been incurred by switching on the link too late. In general, in the context of HPC, it is better to reduce the energy savings than risk a noticeable degradation in performance. Varying the displacement factor exposes a trade-off between the two.

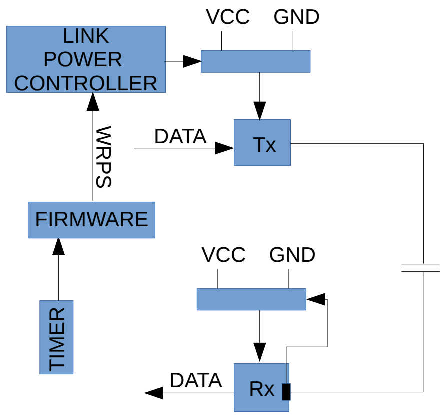


Fig. 5. Link Block diagram

D. Hardware Support

Figure 5 shows the hardware support that is required for IB link power management. A special command is required, which enables user code to request that the link enters low-power mode once any ongoing communication has completed. In order to avoid interrupting the CPU when it is time to wake up, we propose adding one hardware timer associated with the link. This timer is programmed using the predicted idle time. After the programmed delay elapses the timer will generate an interrupt to the firmware, which will reactivate the lanes. Communication between PMCC and the hardware is unidirectional, meaning that there is no feedback to the system on the correctness of prediction.

IV. Experimental Evaluation

A. Methodology

In order to quantify the performance impact and energy savings, we use the Venus–Dimemas [20], [21] simulator. Dimemas is an event-driven simulator, which replays a trace of the application's computation bursts and MPI activity, preserving casual 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 modelled by recording their durations in the trace. We obtained traces of five representative HPC applications on a machine based on Bull B505 nodes, each with two 6-core Intel Xeon E5649 processors running at 2.53 GHz and with 24 GB of RAM. The applications were configured with one MPI process per node and strong scaling (i.e. a fixed workload). The parameters of the simulated system are given in Table II.

TABLE II
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	$1\mu s$
CPU Speedup	1
Routing scheme	Random routing
Switch power consumption	43% when in low-power mode [15]

We first ran the simulations without modifying the traces, in order to check that the original execution times were reproduced. Next, we apply PPA 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 happen delays due to reactivation of a lanes are inserted in the traces. All other overheads associated with the power saving mechanism are inserted, including the time to execute the PPA algorithm, as well as the overheads of data collection. Finally, we simulate the new traces on Venus–Dimemas, in order to quantify the resulting performance and energy savings.

Using the Paraver tool [22], we measure the total amount of time for which the IB links are fully active, as well as the time that the links are in low-power mode. Figure 6 shows a trace from Paraver. The dark blue regions represent durations during which the IB links are in low-power state, and bright blue regions show when IB links are in full-power state. Energy savings are somewhat different for the various MPI processes. The times used for evaluation are averaged over all MPI processes.

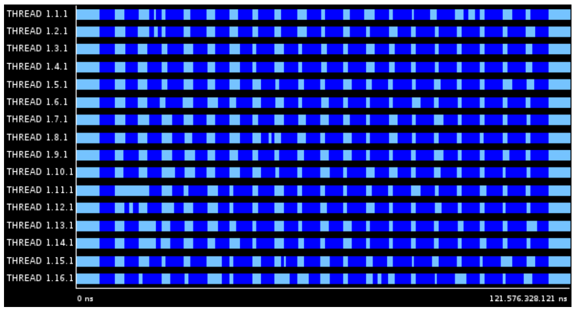
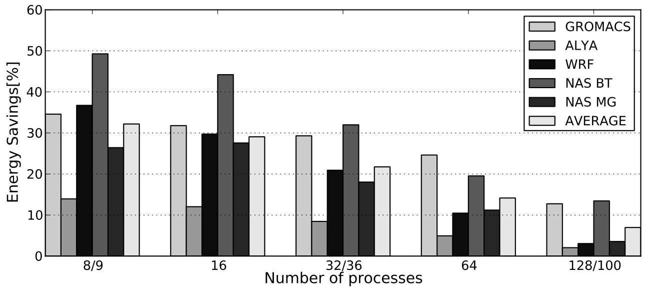


Fig. 6. Execution trace of the Gromacs application with 16 MPI processes, showing when IB links enter low-power mode

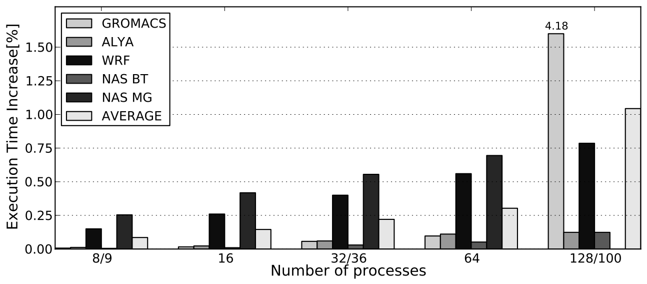
B. Results

This section presents and analyzes the experimental results, in terms of execution time and energy savings. For all benchmarks except NAS BT, we show results for runs with 8, 16, 32, 64 and 128 MPI processes. Since NAS BT requires the number of processors to be square, we instead run it with 9, 16, 36, 64, and 100 MPI processes.

Figure 7 shows the energy savings and performance impact for a medium value of the *displacement factor* equal to 5%. Since we used strong scaling workloads, the amount of communication relative to computation increases with the number of nodes, inevitably reducing the opportunities for energy savings. We expect this problem to not occur with weak scaling. For the same reason, larger scale runs suffer from a larger increase in execution time, but still the maximum average increase, across applications, is around 1%. Due to larger inter-

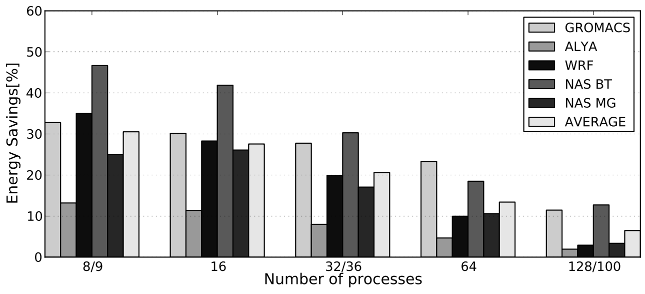


(a) Energy savings in IB switches

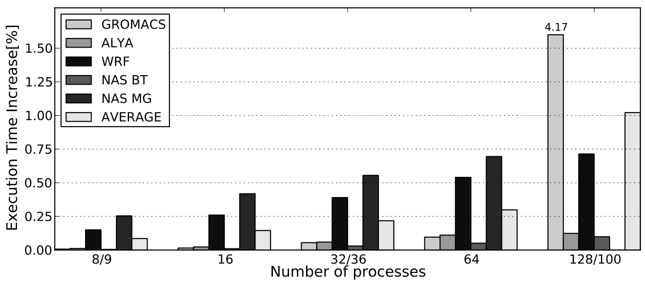


(b) Applications execution time increase

Fig. 7. Strong scaling results with medium *displacement factor*

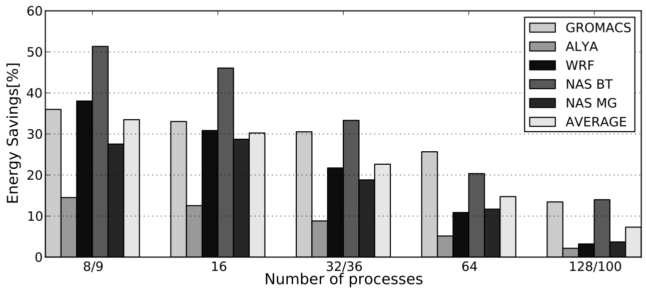


(a) Energy savings in IB switches

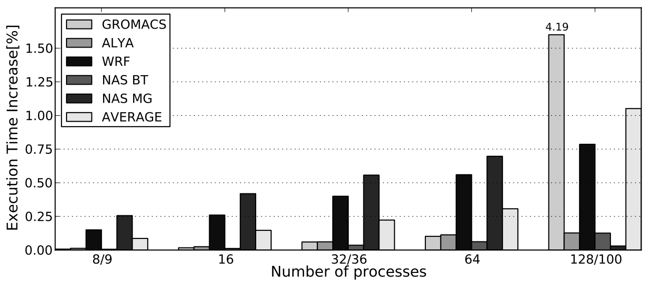


(b) Applications execution time increase

Fig. 8. Results when large $displacement\ factor$ of 10% is employed



(a) Energy savings in IB switches



(b) Applications execution time increase

Fig. 9. Results when small $displacement\ factor$ of 1% is employed

process communication the delays introduced in the system coming from our power saving mechanism can accumulate between processes. Depending on the communication pattern during execution, this could bring the agglomeration of delays and create a total delay in the entire application that is much larger than a single local delay on one MPI process. This can be seen for the Gromacs application, where in a run with 128 processes, we see more than 4% increase in execution time.

Figures 8 and 9 explore the trade-off in varying the *displacement factor*. Choosing a larger *displacement factor* reduces the overheads incurred by waking the link up too late, at the cost of reduced time in the low-power mode. The results for a large displacement of 10%, in Figure 8 show that the average energy reduction is lower at 30.6%, with an almost negligible increase in execution time, compared with the original. Using the smaller displacement factor of 1%, in constrast, shown in Figure 9 gives the largest average energy savings of 33.5%, at the cost of potentially larger impact on execution time.

The energy consumption of the interconnection network can be reduced further if other components in the switches can be turned off; e.g. the input buffers and crossbars. The reactivation times of these elements are much longer, at up to a millisecond, which could cause an unacceptably large increase in execution time. We expect that our power saving mechanism can better amortize larger reactivation times and allow switches to go to deeper low-power modes without any major negative effect on the execution times.

C. Grouping Threshold (GT) Value

An important parameter in the PPA algorithm is the grouping threshold (GT) value, which determines whether two consecutive MPI calls should be considered as part of the same gram. Since there are no opportunities for power savings during idle periods shorter than $2\times T_{\mathrm{react}}$, the value of GT should be greater than this value.

Table III shows the values of the grouping threshold that were used for evaluation, as well as the resulting prediction accuracy. Prediction accuracy is averaged over all MPI calls, including those outside the iterative parts of the application, which correspond to less predictable initialization and finalization phases. This is an important consideration for WRF and partially for Gromacs, while for Alya, NAS BT and NAS MG, the majority of calls are inside the iterative phase and the prediction accuracy is rather large. It is interesting that although the WRF application has the lowest prediction accuracy, it has the second-largest power savings; see Figure 9(a). This is because the majority of large idle intervals are inside the iterative phase, while idle intervals in the other parts of the application are quite small. The opposite is true for the Alya application, where the prediction accuracy is large but the power savings are smaller. Here, the mayority of the large idle intervals are not in the iterative part of the application.

D. System Overheads

To measure overheads we relied on the system clock using the *gettimeofday* system call. The costs of overheads associated with interception of the MPI call and reading the system time are approximately 1μs. These overheads occur every MPI call while overheads that come from power saving system are different and do not occur on every MPI call. When the algorithm predicts the repeating pattern allowing power saving mechanism to shut down inactive lanes, the PPA is disabled, waiting for pattern misprediction to be relaunched again. Also, if the number of necessary grams is not enough the PPA will not be invoked. For activation/deactivation of the IB lanes, we chose a typical latency of 10μs. While the deactivation will be overlapped with computation, the reactivation penalty in case of misprediction has to be paid. The penalty could be equal or smaller than reactivation time if reactivation has already been started. The PPA overheads are also varying and

TABLE III
Chosen GT across HPC applications

	Num proc	Grouping Threshold, GT (μs)	MPI call hit rate (%)
GROMACS	8	20	42
	16	222	44
	32	20	48
	64	22	44
	128	136	59
ALYA	8	20	93
	16	72	93
	32	36	93
	64	36	93
	128	20	93
WRF	8	56	25
	16	30	33
	32	30	32
	64	36	31
	128	22	31
NAS BT	9	20	97
	16	22	98
	36	46	98
	64	20	98
	100	50	98
NAS MG	8	300	74
	16	382	79
	32	300	70
	64	290	74
	128	150	74

TABLE IV
Average PPA overheads: 16 MPI processes

HPC workload	MPI calls when PPA is invoked	Overhead when PPA invoked (μs)	Average overhead per MPI call (μs)
Gromacs	4.7%	25.1	2.1
Alya	1.2%	16.1	1.2
WRF	0.4%	7.8	1.1
NAS BT	3.7%	6.9	1.1
NAS MG	0.5%	26.4	1.05
Average	2.1%	16.5	1.3

depend on pattern size and number of all possible patterns detected during the execution. We used *uthash* [19] hash table to store the pattern objects where pattern is used as a key. Table IV shows the average overheads of PPA through the HPC applications. Although the overhead per MPI call on first sight can seem very large, it only occurs on small number of MPI calls (average 2.1%). The overheads associated with PPA can be further reduced by using faster hash tables.

V. Related Work

Optimization of the interconnection network power consumption optimization is an important target for HPC system designers. Hoefler [5] gives an overview of the power problem and related aspects of interconnect power, with a focus on supercomputers. Power models for the interconnection network, which characterize the power profile of network routers and links, have been proposed, enabling further research into power-efficient techniques [23]. Several power reduction techniques have also been proposed, with most of them based on DVS. Shang et al. [7] proposed a history-based DVS policy, where past network utilization is used to predict the future traffic. Soteriou at al. [24] propose software techniques that form an extension to a parallelizing compiler flow, statically generating DVS instructions that later will direct run-time network power reduction.

Other techniques are based on turning off communication links that are either idle or have low utilization. Alonso et al. [8] propose a power-saving mechanism for regular interconnection networks built with a high degree switches, where each network dimension is formed from multiple links in parallel. The idea is to turn off and on the links that compose trunk link, as a function of the network traffic. All links but one can be turned off. Therefore, connectivity in the network is maintained, which allows the use of the same routing algorithm regardless of the power reduction level. In the work of Kim et al. [25], a DVS technique is complemented with powering down under-utilized links. The use of an adaptive routing algorithm is required, in order to avoid deadlocks.

Li et al. [26] proposes a compiler-directed communication link shut-down strategy. The compiler determines the final use of communication links within each loop nest, and inserts a link shut-down instruction. The link is reactivated upon the next access to it. Our approach is similar, in that shut-down instructions are issued from the CPU in a way that depends on an analysis of communication patterns. In our approach, these instructions are issued by the runtime, without requiring any modification in the source code, while here the turn-off instructions are inserted during the compilation process.

The work of Lim et al. [27] is complementary to ours, since the MPI run-time system dynamically reduces CPU power consumption during communication phases. The run-time system identifies communication regions and selects the processor frequency in order to minimize the energy-delay product, without profiling or training.

Jian et al. [28] work is focused on non-prediction power-saving techniques. Links are powered up just before they are needed, by relying on hints from the built-in system events or from macros in MPI source code. Here, separate control network is needed which is always on, to enable link activation messages to flow through. In our approach, we rely on IB architecture with 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.

Saravanan et al. [29] provide a detailed evaluation on Energy Efficient Ethernet (EEE) from the perspective of HPC. They propose a technique to further increase power savings of HPC systems by leaving the link in active mode until a threshold time is reached.

VI. Conclusions

High-performance computing is an increasingly important tool for modern science. There is a constant demand for more powerful supercomputers, but increasing performance is leading to excessive peak power demand and energy consumption. Now that energy consumption is beginning to account for a significant fraction of an HPC system's total cost of ownership, there is pressure for all system components to become more energy efficient. An important characteristic of energy-efficiency is energy proportionality. Although processors and memories are now close to energy proportional, high-performance interconnects are not.

This paper presents a software-directed mechanism for interconnect link energy proportionality. We propose the PPA algorithm, to be executed within the PMPI layer of MPI. Putting the intelligence in the MPI library allows differentiation by the system integrator and customisation by the operations department, while avoiding any need for modifications to the user's source code. This allows energy savings to be achieved for unmodified existing MPI applications.

The PPA algorithm detects the repetitive communication patterns that are typical of modern scientific applications, and it uses this knowledge to predict the durations of the link idle periods. The links are put into low-power mode during idle periods until a short time before they are expected to become active again, leading to a significant reduction in the average link energy consumption at negligible loss in performance.

We evaluated the possible power savings with strong scaling runs, which gives pessimistic results, since network utilization, which should be proportional to energy consumption, increases with the number of nodes. In addition, strong scaling leads to shorter computation periods, meaning that constant overheads in changing power mode are amortized over short idle periods. Weak scaling runs would therefore lead to larger observed energy savings. Nevertheless, the results show the possibility for significant energy savings in IB switches of up to 33%, with a negligible increase in execution time of around 1%.

We also show possibilities for further switch energy savings, by powering down other elements in the switch. Such elements take much longer to change power state, requiring up to a millisecond to wake, increasing the need for accurate prediction mechanisms such as the PPA algorithm. We will evaluate this scenario in future work, by taking into account the powering down of other elements in the switch.

Finally, it is important to note that the principles of our system are not restricted to Infiniband. Many modern interconnect technologies, like Infiniband, have multiple lanes at the physical layer. For example, 40GbE Ethernet has four lanes at 10 Gb/s each, although there is currently no standard for turning lanes on and off individually. Proposals like ours may have an impact on future standardisation efforts.

References

[1] "Ranking the world's most energy-efficient supercomputers," 2014, http://www.green500.org/.

[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] "Ibm. ibm infiniband 8-port 12x switch," 2011, http://www-3.ibm.com/chips/products/infiniband.

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

[6] V. Soteriou and L.-S. Peh, "Design-space exploration of power-aware on/off interconnection networks," in *ICCD*, oct. 2004, pp. 510 – 517.

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

[8] 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.

[9] 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.

[10] 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.

[11] "Gromacs(GROningen MAchine for chemical simulations)," University of Groningen, The Netherlands, 2011, http://www.gromacs.org.

[12] "Alya system - large scale computational mechanics," Barcelona Supercomputing Center, Barcelona, Spain, 2011, http://www.bsc.es/plantillaA.php?cat_id=552.

[13] J. Michalakes, "The weather research and forecast model: Software architecture and performance," in *In proceedings of the 11th ECMWF Workshop on the Use of HPC In Meteorology,*, October 2004.

[14] "Nas parallel benchmarks 3.3," 2014, http://www.nas.nasa.gov/Resources/Software/npb.html.

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

[16] J. Zheng and S. Lonardi, "Discovery of repetitive patterns in DNA with accurate boundaries," in *Fifth IEEE Symposium on Bioinformatics and Bioengineering (BIBE)*, Oct 2005, pp. 105–112.

[17] N. Patel and P. Mundur, "An n-gram based approach for finding the repeating patterns in musical data." in *EuroIMSA*, Grindelwald, Switzerland, 2005.

[18] L. Alawneh and A. Hamou-Lhadj, "Pattern recognition techniques applied to the abstraction of traces of inter-process communication," in *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, 2011, pp. 211–220.

[19] T. D. Hanson, "*uthash*: A hash table for c structures," 2013, http://troydhanson.github.io/uthash/.

[20] 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.

[21] 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.

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

[23] H.-S. Wang, L.-S. Peh, and S. Malik, "A power model for routers: modeling alpha 21364 and infiniband routers," in *High Performance Interconnects, 2002. Proceedings. 10th Symposium on*, 2002, pp. 21–27.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.