Software-Managed Power Reduction in Infiniband Links

Branimir Dickov*†, Miquel Pericàs‡, Paul 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*—Interconnection network represents the backbone of large-scale supercomputers. While the computation has become more energy proportional, the interconnection network has started to become one of the important consumers in the system. Still, interconnection network has received little attention in terms of efficient power consumption. Several hardware-based schemes have been proposed that function either by turning off the links or lowering the voltage/frequency in links when idle or with low usage. Although effective in certain cases, these schemes share the common drawback that they do not have enough global information about the application to manage network power efficiently. This paper, instead, proposes a software prediction support for shutting down inactive lanes of an Infiniband (IB) link.**

Using *n*-gram extraction technique as a core of the prediction algorithm we are able to predict with high accuracy when links will be unused. The prediction algorithm detects the repeating patterns of MPI calls during the program execution, on which basis the prediction of the following MPI calls can be realized with relatively high precision. This allow us to know on time when links will be active/idle and invoke the explicit turn-off/on lanes calls. Our power saving mechanism is adopted to be executed within MPI allowing a large number of existing MPI programs to be run without any source code modification. Results show an average power savings in IB switches up to around 33% over the representative HPC applications. On the other hand, HPC applications average suffer only up to 1% increase in execution time.

I. Introduction

HPC systems deliver tremendous peak performance for solving problems in many scientific areas. This high performance comes at a cost of increased power and energy consumption in these systems. As a consequence, supercomputers beside their peak performance nowadays are ranked also based on their energy efficiency [1]. Therefore, power and energy consumption have become first-order design constraints in HPC. While there has been a lot of effort to reduce power consumption of processing and memory elements in the system, similar techniques in networks have not reached wide adoption. In interconnection network power consumption does not depend on its utilization and it is near the peak whenever the system is "on". With the power efficient processing elements and larger networks design it is expected that in the future systems power consumption of interconnection network can rise up to 30% of the system's total power [2]. In the case when processors are not highly utilized in non-HPC data centers, it can go up to 50% [3]. The majority of total network power goes to power consumption of interconnection links. It has been shown that links in an IBM Infiniband 8-port 12X switch can take 64% of the switch power [4].

One approach to reduce network power is to turn off the links when they are not in use. The problem with this technique is that state changes of the link from off to active could take up to 10 microseconds [5], which directly adds to latency of transmitted MPI messages. While this mechanism offers a huge power saving potential, performance of HPC application can be severely degraded. The other option would be to lower voltage/frequency in links with low usage. Now we would speed-up the reactivation of links ($\approx 100ns$) but with a cost of much lower power saving potential [3]. Both power aware methodologies typically use hardware prediction schemes for shifting between power modes [6], [7], [8]. These schemes have a common drawback that their effectiveness is largely dependent on their prediction accuracy.

In this paper, we focus on the HPC applications that correspond to the Single Program Multiple Data (SPMD) programming paradigm. All application processes perform the computation part at the same time followed by communication step at the same time. While computation is spent on actual work, communication time is spent for satisfying the remote data dependencies in a parallel application. Therefore, the application developers try to minimize the communication as much as possible. Considering that the current systems use always-on link we see that this optimization is not transferred to power saving in the links and that large portion of power provisioned for communication is wasted. As a result, in the future systems it is essential to save network power by making power and energy consumed related to the usage of the network.

It has been observed that HPC application usually consist of a large number of iterative execution phases between the initialization and finalization phases. This offers a possibility to learn from the previous iteration and apply gained knowledge on the future iteration. Specifically, this means that the patterns of MPI calls that are repeating within each MPI process can be detected. To achieve this, we used an algorithm that is based on *n*-gram extraction techniques, a concept which is widely used in statistical natural language processing [9], [10]. Our pattern prediction algorithm (PPA) allows an on-the-fly detection of repeatable process patterns.

TABLE I
DISTRIBUTION OF LINK IDLE INTERVALS

	N proc	T_idle <20 μs			20 μs <T_idle <200 μs			T_idle >200 μs		
		N of intervals	Intervals [%]	Exec. Time [%]	N of intervals	Intervals [%]	Exec. Time [%]	N of intervals	Intervals [%]	Exec. 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	33	1.26	0.0003	12	0.46	0.0097	2568	98.28	99.99
	16	3440	57.17	0.02	13	0.22	0.01	2564	42.61	99.97
	32	887	25.58	0.02	821	23.67	0.85	1760	50.75	99.13
	64	5995	69.87	0.12	827	9.64	0.89	1758	20.49	98.99
	128	7695	74.91	0.2	1643	15.99	5.06	934	9.09	94.74
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

Therefore, we will get a global view of order and timing of link usages which can allow us to make transition between different link power consumption modes with minimum negative effects on performance. Exploiting new power saving features for IB switches we are able to reduce power consumption in the links without loosing full network connectivity. This feature enables shutting down all but one lane of an 4X-IB link.

In this paper, we propose a power saving mechanism which consist of two parts, the first of which detects the repeatable patterns in a process allowing successful prediction and the second of which actually does the shift and control of link power modes. If we are in the region where prediction is possible, upon an appropriate MPI event the explicit turn-off IB lanes calls are issued (one lane stays always on). At the same time, the predicted idle time is sent to link power controller on the host channel adapter (HCA). Lanes are activated again after a timer elapses which prompts the power controller to change the state of associated lanes. Our system is adaptable to be run within the PMPI profile layer of MPI allowing to avoid any user involvement. Therefore, a large number of existing MPI programs can be run without any source code modification. Specifically, this paper makes the following contributions:

- We demonstrate the large potential for power saving in interconnection network due to idle communication links. The distribution of idle link intervals in our HPC workloads show that in the majority of cases more than 99% all idle times represents the candidate where link power reduction is possible. We also point out how using new features in IB switch architecture this can be done without loosing the connectivity in the network.
- We show that PPA can efficiently predict the pattern appearance in the traces of the studied HPC applications. We obtain large prediction rates of network communication up to 98% in some cases. Also, we provide the complete description of our power saving mechanism enabling be run within the PMPI layer of MPI.
- Using event-driven simulator we evaluate our power saving mechanism using traces taken from production supercomputer. Results on the HPC applications show an average reduction in power up to 33% in IB switch compared to the power-unaware scheme where links are always-on. Also, we show that reduction in power did not come at a large increase in execution time. We got the worst average increase across all application around 1%.

The remaining part of the paper is organized as follows: In Section II, we provide motivation for switching IB links to low-power mode during the iterative computation phases. Next, Section III discusses the design our link power saving mechanism. In Section IV, we describe the simulators used and evaluation of possible low-power mode idle times during the application execution. This evaluation is followed by power-time trade-off analysis in Section IV-B. Section V exposes the related work. We finalize this work by presenting our conclusions in Section VI.

II. BACKGROUND AND MOTIVATION

Computation phases between MPI calls provide an excellent opportunity to reduce the bandwidth to a minimum, thereby saving power in the interconnection network. Considering that turning on/off lanes could take up to 10 microseconds [5], it is desirable to know if inter-communication phases are large enough to provide an opportunity for power savings in IB links. Therefore, the idle link times should be bigger than summed together time to activate and deactivate the lanes. In this work, we will take that both times are equal. This means that the interval should be twice larger than the time necessary for reactivation of the link, $T_{idle} > 2 * T_{react}$. From the Table I, we see that for almost all applications the idle

intervals larger than 20 μs together occupy more 99% of all idle times (times are obtained from execution traces, more details in Section IV-A). What is even more important is that larger idle intervals ($T_{idle} > 200\mu$s), where significant power can be saved, represent in majority of cases more than 90% of total time where link remains idle. Just in case of NAS MG benchmark when run with larger number of processes we get that large idle times are slightly lower then 90% of accumulated link idle time. Although the number of adverse intervals ($T_{idle} < 20\mu$s) rise with the number of MPI process there cumulative sum takes very small part in total idle time. Therefore, all this provides an opportunity to achieve large power savings by activating the power saving mechanism only when idle times are long enough. While the deactivation of the IB lanes can be overlapped with computation, again reactivation could potentially create a delay of the following communication. In an ideal case, the IB link lanes would be turn on sufficiently earlier so that upcoming communication can be done without any latency penalty. To get closer to this ideal case, we propose the use of a prediction algorithm that could supply the system with enough knowledge so that communication can be done without any latency penalty due to not fully operative IB link.

A. Network power management support on Infiniband switches

Recently Mellanox has developed HCA and switches that can save power by optimizing each port's link width and speed. These optimizations are emmbedded in the HCA and switch hardware and are enabled via the firmware. Port link width reduction is done using Width Reduction Power Saving (WRPS) method. For example, using WRPS a 40 Gb/s 4X QDR port can run as 10 Gb/s 1X QDR shutting down three inactive QDR lanes. This reduction in link width allows Mellanox Switch SX6036 to consume only 43% of the nominal power (when all four lanes are active) [11]. We choose this value to correspond to our low-power mode consumption in IB switches.

III. Design

Main goal of our power saving mechanism is to save link power when links are idle with almost identical execution times. In order to identify program regions where lanes can be turn off, T_{react} will be taken as a key parameter. The larger the inter-communication intervals the more power can be saved. From here we see that our power saving mechanism would fit perfectly for compute-intensive programs were weight of compute phase in overall execution time would be high. Therefore, we expect that power saving mechanism would be more effective for weak scaling than for strong scaling runs.

With the aim of finding the link power reducible regions the pattern prediction component of our power saving mechanism is run. Here, the repeatable patterns in a MPI process are detected and if the conditions for the prediction are fulfilled the power mode control component is invoked (see Figure 1). In power mode control component the transition of IB links to low-power mode is done. Upon its activation, pattern prediction part is mainly disabled with the aim to eliminate unnecessary overheads that can come from pattern prediction

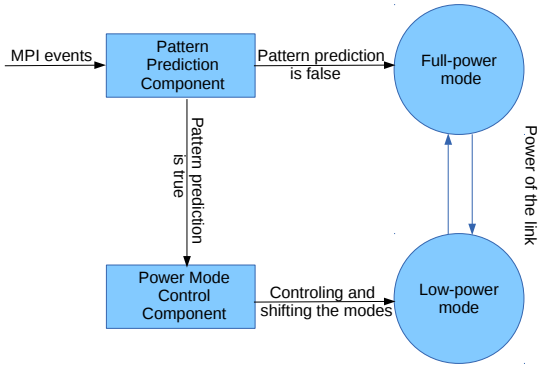


Fig. 1. Simplified diagram of MPI process pattern prediction system that reduces power consumption in interconnection links

algorithm (PPA) execution. Just inter-communication intervals continue to be updated with the new values allowing more accurate transition between power modes. Therefore, we can say that the system is transitioning between pattern prediction component and power mode control component. In the case when pattern misprediction happens the pattern prediction part will be fully activated again while the overhead of reactivation the IB lanes is paid. Now the links are again in the unoptimized full power mode and no transition to low-power mode will occur until some predictable pattern is found again.

As the transition from one power mode to another is happening during the runtime our system is perfectly suitable to be run within PMPI runtime layer. Implemented in such way will enable the current MPI programs to be run without any need for source code modification or recompilation of the MPI runtime library before the execution of the MPI programs. Upon interception of the MPI call the pattern prediction component is executed while after the MPI call if possible power mode control component is run.

A. Pattern Prediction Part

PPA does an on-the-fly detection of repeatable process patterns. It operates on the stream of MPI events that are occuring during the runtime, Figure 2. Also, PPA operates on each MPI process individually.

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 in detecting DNA patterns [12] and patterns in musical notes [13] efficiently. An *n*-gram is defined as a subsequence of *n* items in a sequence. The smallest *n*-gram that can be considered as repeat is a bi-gram (could be two consecutive MPI events).

To start the algorithm the $grams$ on which algorithm will operate need to be formed (Algorithm 1). A $gram$ is constructed of at least one or more consecutive MPI calls, based

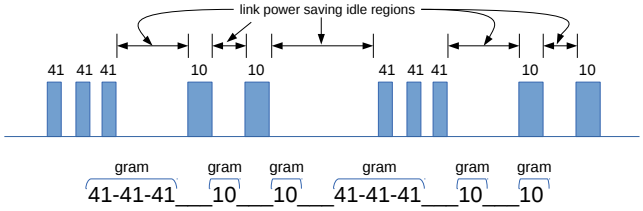


Fig. 2. Stream of MPI events generated from Alya application on which pattern prediction algorithm will operate (41 - ID for MPI_Sendrecv, 10 - ID for MPI_Allreduce)

on the distance between adjacent MPI calls. This distance is determined by T_{react} value which will give the final answer which MPI calls will be grouped together and which one will form distinctive groups of MPI calls. In order to gain benefits with on/off technique, we should have idle time of a link at least two times larger than the time necessary for reactivation of the IB lanes ($T_{idle} > 2 * T_{react}$). As this value has influence on forming groups of MPI calls we name it $grouping\ threshold$ (GT). Therefore, all the adjacent MPI calls that have their inter-communication time less than GT will be grouped together in one gram while MPI calls that have inter-communication time more than GT will belong to different grams. After applying the Algorithm 1 on the stream of MPI events from Alya application, three $MPI\_Sendrecv$ calls are grouped together while two $MPI\_Allreduce$ calls form separate groups which can be seen on Figure 2.

Algorithm 1 Forming Array of Grams

```
1: if previousIdleTime < groupingThreshold then
2:     append eventType to currentGram
3: else
4:     add currentGram to array at pos
5:     pos = +1
6:     clear currentGram
7:     append eventType to currentGram
8: end if
9: if pos < (posNextNGram + patternSize) then
10:    break
11: end if
12: if patternPrediction is false then
13:    call PPA()
14: end if
```

Main goal of PPA is to find continuous repeats of process pattern and to predict with high accurancy if that pattern will continue to emerge. Therefore, we established the following policy:

- *If the same pattern appears three times consecutively, we predict that the 4-th one will be the same as previous three and the power mode control component is activated*
- *If the pattern is mispredicted and in near future the same pattern appears again we don't wait for three consecutive appearances of the pattern but we declare on the first new appearance that the pattern as repeatable and predict that the next one will be the same*

Our PPA described in Algorithm 2 is based on algorithm for detection of patterns proposed by Alawneh [14]. We modified the algorithm and adopted to be able to detect the continuous repetitions of pattern and predict pattern appearance based on previous appearances during the execution of the program.

The PPA uses three main objects: An array which holds formed grams on which algorithm will operate, the pattern object which contains the pattern sequence, its length, its position in the array, its frequency, the time between two grams in a pattern and number of MPI calls in a detected pattern and the pattern list which is a hash table that holds the pattern objects created during the execution. We used $uthash$ [15] hash table to store the pattern objects where pattern is used as a key. Furthermode, two pointers pointing at each moment to current and adjacent n-gram object in the array.

The PPA (Algorithm 2) will start to operate if the number of formed grams located in the $array$ is enough (line 9 in Algorithm 1). All $boolean$ variables used are global and set

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: PatternList(PL) : Hash Table
2: newPattern : Boolean(true)
3: patternPrediction : Boolean(false)
4: checkConsec : Boolean(false)
5: match : Boolean(false)
6: currentPattern : String
7: consecutiveRepeats : Integer(0)
8: patternSize : Integer(2)
9: posCur = position of currentNGram
10: posNex = position of nextNGram
11: if newPattern is false and checkConsec is false
12:      and patternSize < maxPatternSize then
13:    checkPrevious = true
14:    match = false
15:    nextPattern = appendGram(currentPattern)
16:    newPattern = updatePL(nextPattern, curPos)
17:    if newPattern is false then
18:        if PL[nextPattern].detected is true then
19:            patternPrediction = true
20:        end if
21:    end if
22:    match = checkO(nextPattern, currentPattern, posCur)
23:    checkConsec = true
24: end if
25: if checkConsec is true then
26:    checkConsec = checkConsec(posCur, posNex)
27:    if checkConsec is true then
28:        consecutiveRepeats+ = 1
29:        UpdatePL(currentPattern, posNex)
30:        match = true
31:        if consecutiveRepeats > 2 then
32:            maxPatternSize = patternSize
33:            PL[currentPattern].detected = true
34:            patternPrediction = true
35:        end if
36:    end if
37:    if match is false and checkPrevious is true  then
38:        remove nextPattern from PL
39:        patternSize = 2
40:    end if
41: else
42:    currentPattern = array[posCur, patternSize]
43:    newPattern = updatePL(currentPattern, posCur)
44:    checkConsec = true
45: end if
```

at the beginning to $false$ except $newPattern$ which is set to $true$. This is the reason why the algorithm will straightaway go to line 42. Here, the first bi-gram from the $array$ of grams will be read, which is the first pattern to be added to the pattern list. At the next entry in the algorithm, at line 26 we check if the detected pattern is repeated continuously by calling function $CheckConsec$. If not the algorithm will continue to read the next bi-grams from the $array$ and add them to the list until a bi-gram match is detected. If that happens, the $newPattern$ variable at line 43 is set to $false$. This opens the possibility to append next gram from the $array$ on the right of the previously matched bi-gram which will result in a tri-gram (line 15). Before doing it, we again need to check if the current bi-gram has its following pair (line 26). If there are no consecutive repeats of bi-gram and previously bi-gram match has been detected we enter if statement at line 11. Using $appendGram$ function we enlarge pattern to tri-gram ($nextPattern$). Then at line 22 we check if the tri-gram can be constructed from the previous occurrences of its bi-gram by calling the $checkO$ function. If the previous occurrences of the bi-gram can be constructed to match the detected tri-gram, the frequency of the tri-gram pattern will be incremented while the frequency of the bi-gram will be decremented. If the

newly constructed n-gram cannot be detected at any previous position of its prefix n-gram than it will be removed from the pattern list (line 38) and the size of a n-gram will set to the minimal which is 2 (bi-gram). In an opposite case the algorithm continue to add more grams to a pattern each time $match$ variable is set to $true$. Here is the key point in the algorithm, which is that each time we enlarge the pattern and add a new pattern to hash table, we check if that new pattern has consecutive appearances. When that happens and the number of consecutive repeats becomes equal to two (3-th appearance) we can declare the current pattern as the predicted what will consecutively enable the activation of power mode control component of the system. This means that from now, all upcoming inter-communication intervals will be predicted allowing to execute a transition of IB link to low-power mode. Therefore, the boolean $patternPrediction$ variable from now on will be set to $true$. Also, the pattern object variable $detected$ we set to $true$ in order to immediately activate the power reduction procedure upon the previously predicted pattern is found again.

In order to recognize the natural (real) iteration in the application and do the prediction of the next iterations based on the times of the previous ones, we declare the current pattern size as a maximum pattern size at line 32. If not done so the algorithm would merge more and more iteration as one pattern and the later iteration would be predicted based on the values from the iteration that were executed much earlier before. Therefore, the pattern size can vary from the smallest bi-gram to the size defined by $maxPatternSize$ value.

On the Figure 3, we present an example of Alya application and communication events occuring during its execution and how our pattern prediction algorithm operates and predict the next region of communication events. Each time a MPI event is intercepted we try to form a gram on which PPA will operate if conditions to run PPA are met. From Figure 3 we see that for first 8 MPI events, the number of formed grams is not enough to start PPA execution. On the 9th MPI call, we start PPA by reading first bi-gram "41-41-41, 10" at position 0 which we add as a new pattern to the pattern list. On next MPI event, we check if there are contiguous repeats of the pattern. If not next bi-gram will be read and added to pattern list which is the case. On the following MPI events, we repeat these two operations, adding and checking until 15th MPI event appear when find out that new pattern is existing pattern in pattern list. The frequency of the matched pattern is incremented and new position is added. Now we check again for contiguous repeats and again obtain negative answer. Because it is existing pattern on the next MPI event we add one more gram to the pattern and check for contiguous repetitions. Since now there are contiguous repetitions and the same pattern appear more than three times we declare the pattern "41-41-41, 10, 10" as the predicted and now from the position 12 in the array all the idle periods can be predicted allowing to accurately switch the IB link to low-power mode.

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

PPA execution:

#	MPI ID	Array of grams	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	[41-41-41,10,10,41-41-41,10,10,41-41-41]	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

Insertions into Pattern List:

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

predicted pattern	from position
41-41-41_10_10	12

Fig. 3. Pattern prediction example

B. Power Mode Control Part

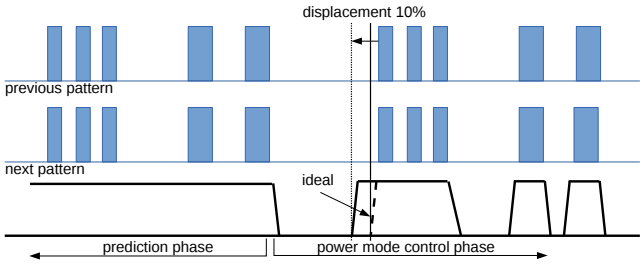
Power saving mechanism itself is based on the available interval times between MPI calls from detected repeatable patterns. Although these times are averaged over previous appearances still, the inter-communication times can vary notably throughout the iteration phases. To be sure that communication link will be available on time, we establish the $displacement\ factor$ as a parameter that can give us higher guarantees that during predicted idle time no MPI calls will be invoked. It is better to power up a link little bit earlier than needed, than to introduce an overhead in the system due to reactivation of a link. We can see this on the Figure 4(a) where with the full line is indicated when in an ideal case the reactivation should start and with the dashed line when we

conservatively start. Therefore, although we will not achieve maximum possible reduction in power, we rather choose to guarantee that the link will be fully operative without any latency penalty. Still, in some cases the misprediction can happen. This can be seen on the Figure 4(b) where we see that everything is now delayed because the reactivation of the IB lanes wasn't started sufficiently earlier.

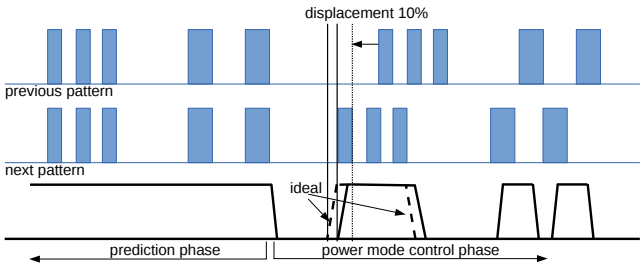
It is clear that as larger is the $displacement\ factor$ the probability for misprediction will diminish considerably. On the other hand with large $displacement\ factor$ we lose an opportunity to save more power by activating the link as late as possible. Therefore, we can trade the time for power but also opposite is possible. The timers that measure the predicted intervals between MPI calls are activated upon the turn off lanes instructions are executed. What is important is to understand that in our system two types of misprediction can happen, the first one may occur during the pattern prediction process when the current pattern is not the one that was predicted and the second type of misprediction can happen for the correctly predicted pattern when idle intervals may not coincide perfectly and the reactivation of the link will be done to late.

Therefore, we can conclude that during the program execution the link can operate in three modes. In the part of program where prediction is successful the links will work in low-power mode. On the other side, in the part that can't be predicted the links will work in full operative mode. The last mode is in the switching part where links are shifted from one mode to another. For the shifting phase, we take that consumed power would equal the power when link is fully operative.

Control of the link power mode is presented in Algorithm 3. If the indicator that prediction is enabled is set on and the group of current MPI calls corresponds with the size and content to the predicted gram, the links are shifted to low-power mode. If not the links will stay at the full speed



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



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

Fig. 4. Controling power mode of IB links during the Alya program execution with $displacement\ factor$ of 10%

Algorithm 3 Managing link power mode

```
1: if patternPrediction is true then
2:     if len(currentGram) Equals len(patternGram) then
3:         if currentGram Equals patternGram then
4:             safetyLimit = idleTime * displacementF + T_react
5:             predictIdleTime = idleTime − safetyLimit
6:             WRPS_method(predictedIdleTime)
7:         else
8:             patternPrediction = false
9:         end if
10:    end if
11: end if
```

consuming maximal power. The predicted low-power mode interval is calculated based on $displacement\ factor$ and the T_{react}. The timer that should be part of the power link controller is informed about the duration of low-power mode interval. Once the time elapses the links are back to unoptimized mode allowing original communication to be done. The managing is one directional, the times that tell how long link would stay in low-power mode are supplied to link power controller on HCA, but any feedback to our system about the correctness of prediction are not needed. Regarding the other misprediction, the pattern misprediction, when its happen the $patternPrediction$ variable is set to $False$ and the pattern prediction part is relaunched again.

C. Hardware Support

On the Figure 5 is illustrated hardware support for IB link power managment. In order to avoid interrupting the CPU when its time to wake up the lanes we propose to add one hardware timer to link structure. Upon the link is shifted to low-power mode the timer is programmed using the predicted idle time. After the programmed delay elapses the timer will generate an interrupt to firmware which will carry out the reactivation of lanes.

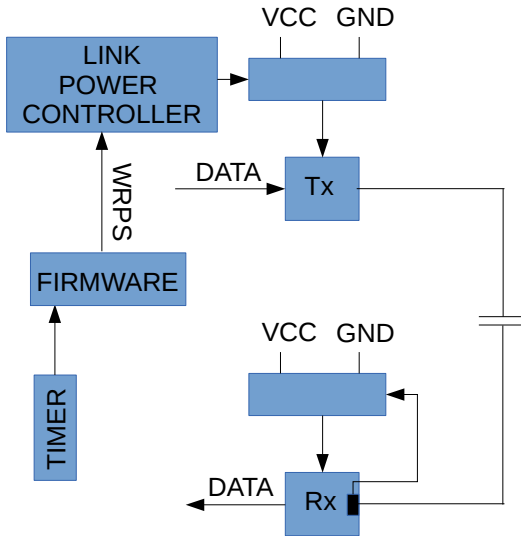


Fig. 5. Link Block diagram

IV. Experimental Evaluation

A. Methodology

In order to quantify achievable power savings in IB links and the corresponding latency penalties that can arise from

our system we use Venus-Dimemas [16], [17] simulator . This simulator can perform full end-to-end simulation of an HPC system. Dimemas simulator allows a replay of the MPI activity in the traces while Venus is doing the detailed network simulation of the communications. To fed the simulator we needed the execution traces of the selected applications. The traces were obtained 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. Configuration with 1 MPI process per processor is used.

In simulation computation operations are not performed, but represented with the actual computation intervals recorded in the traces. Network topology is modeled as a fat-tree with two levels of switches. One MPI application process is allocated per node in our simulator. The detailed simulated system parameters are given in the Table II.

TABLE II
PARAMETERS USED IN SIMULATIONS

Simulator	Dimamas-Venus
Connectivity	XGFT(2;18,14;1,18)
Topologies	Extended Generalizes Fat Trees
Switch technology	Infiniband
Network Bandwidth	40 Gbits/s
Segment Size	2 KB
MPI latency	1 μs
CPU Speedup	1
Routing scheme	Random routing

In order to obtain original execution times we first run the simulation without any modification of the traces. Next, we apply the PPA on the traces and insert the new events which mark where the prediction is possible and events which mark when the links are predicted idle and therefore, inactive lanes can be shut down, thereby saving power. In the traces when misprediction happens the delays due to reactivation of a lanes are inserted. Also, the other overheads associated with our power saving mechanism are inserted (overheads of PPA, overheads of data collection...). These overheads can happen at the beginning and end of each MPI call. After the modification in traces is done, the simulation is relaunched allowing to quantify real effect of our system on the execution times of selected applications and also measure time when IB links are shifted to low-power mode.

Times when IB links are fully active and times when IB links goes to low-power mode are measured using $Paraver$ tool [18]. This can be seen from the Figure 6 where with the darker blue is represented the low-power state of IB links while with the brighter blue is shown when IB links are in power-unaware consumption state. The times used in evaluation are averaged over all MPI processes.

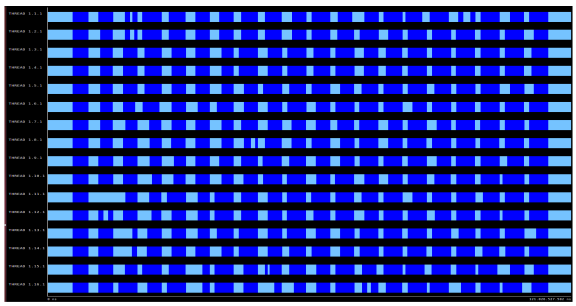
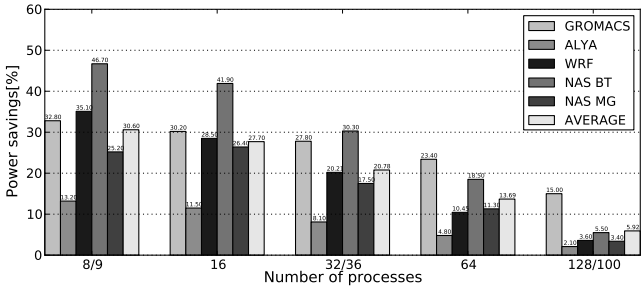


Fig. 6. Execution trace of Gromacs application run with 16 MPI processes showing when IB links enter low-power mode

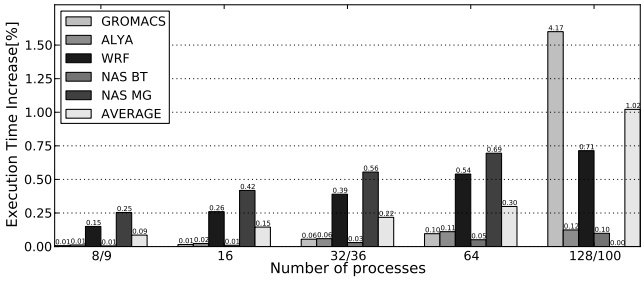
B. Results

In this section, we analyze power reduction achieved by using power saving mechanism proposed in Section III and the corresponding latency penalties for the studied applications.

We show results for runs with 8, 16 32, 64 and 128 MPI processes (NAS BT benchmark we run with 9, 16, 36, 64, 100 processes as it require a square number of processes). Taking in account that inter-communication time could vary throughout the application execution we did the evaluation for different values of $displacement\ factor$. We took $displacement\ factor$ to be 1%, 5% and 10% of the original predicted time for idle region. Beside the displacement we need to take in account reactivation time T_{react} of lanes also. Therefore, we gather this two times and obtain the final prediction for predicted idle region by resting this new summed value from original idle prediction value. By taking large threshold value, we can reduce to minimum the overheads introduced in the system and, therefore, get closer to original execution time. However, the drawback of large threshold values is that consequently we reduce idle low-power region, thereby diminishing total power savings in the IB switch. The results for large displacement of 10% are shown on the Figure 7. Here, we get maximum average power reduction in IB switches of 30.6% with almost negligible increase in execution time. For runs with a larger number of processes, we get smaller benefits in terms of power savings. The reason is that we use strong scaling traces where network communication becomes more dominant in larger scale runs. Therefore, we could expect that our power saving mechanism would be more effective in weak scaling than in strong scaling runs. Also with larger scale runs we get larger increase of

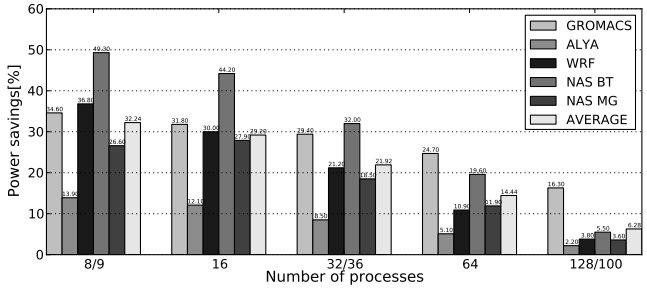


(a) Power savings in IB switches

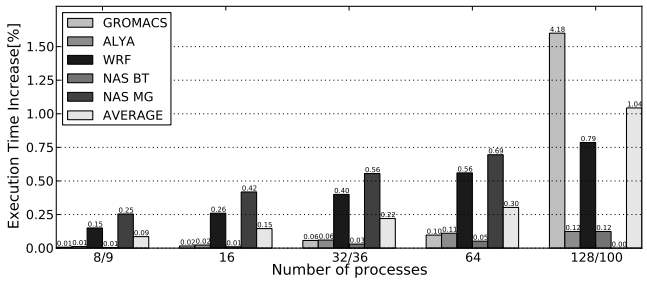


(b) Applications execution time increase

Fig. 7. Results when large $displacement\ factor$ of 10% is employed

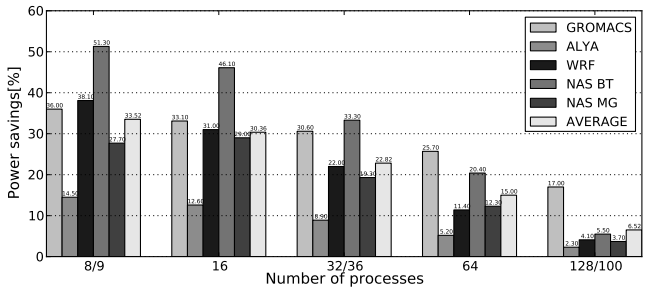


(a) Power savings in IB switches

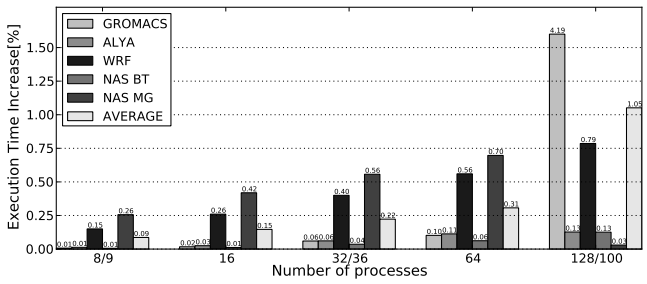


(b) Applications execution time increase

Fig. 8. Results when medium $displacement\ factor$ of 5% is employed



(a) Power savings in IB switches



(b) Applications execution time increase

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

execution time but still the maximum average increase is around 1%. Due to larger inter-process communication the delays introduced in the system coming from our power saving mechanism can propagate between processes. Depending on the communication pattern during the execution this could bring the agglomeration of delays in the system and create global delay of the entire application much larger than what would be expected based on local delays in each MPI process. This can be seen for Gromacs application where in a run with 128 processes we get more than 4% increase in execution time.

On the Figure 8 are shown result for 5% displacement. Here, we get larger average power reduction than with displacement of 10%. On the other side, execution time is not changed dramatically. And finally taking the displacement factor of 1% we get the largest power savings, Figure 9. We get maximum average power reduction in IB switches to be 33.52%. Also, here the execution time increase is within the acceptable limits little more than 1%. We can conclude that with the minimal displacement, we get maximum power savings with the minimum increase in overall execution times of the tested applications. Considering that in the switches beside links exist other power consumption parts like input buffers, crossbars powering down of this parts can further diminish power consumption in the interconnection network. Reactivation times of these elements are much longer and can take up to a millisecond. If the reactivation of the entire switch on time is late we could potentially have delays of up to a millisecond introduced in the system which could lead to unacceptable large increase of execution time. Therefore, 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

Before the prediction algorithm is launched the grams on which will operate should be formed. Grams are created based on the grouping threshold (GT) value. Considering that the distance between two MPI routines can not be less than $2*T_{react}$ GT at least need to be equal or larger than that value in order to have possibilities for power savings in the IB links. The idea is to try to merge all MPI routines together in that way that when we reach iterative parts in the application the grouping stays consistent and avoid that some MPI call ends in another gram. This can happen because the idle times during the iteration can vary and this can provoke the misprediction in algorithm and disable the PPA to achieve maximal power savings in the IB links. Therefore, to evaluate the best possible value for GT we run the PPA taking the GT across the range of values starting with the minimal of $2*T_{react}$. The idea is to try to achieve maximal correct prediction rate on MPI calls but considering also that a large GT value will reduce the number of idle intervals where shifting to low-power mode is possible. In Table III are shown chosen values for GT after an

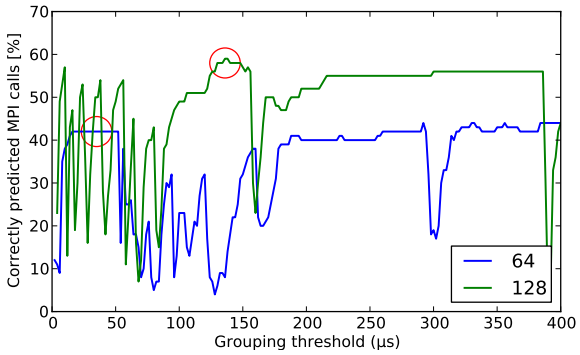


Fig. 10. Evaluation of GT value for Gromacs application when run with 64 and 128 processes

evaluation processes across wide range of values. Evaluation process we can see on Figure 10.

TABLE III
CHOSEN GT ACROSS HPC APPLICATIONS

	N 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

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 around 1 μs. 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 algorithm predict the repeating pattern allowing power saving mechanism to shut down inactive lanes, the core of the prediction part is disabled, waiting for the misprediction when it will be relaunched again. For activation/deactivation of the IB lanes, we took to take maximum 10 μs. While the deactivation will be overlapped with computation, the reactivation penalty in case of misprediction has to be paid. The penalty could be full or smaller than reactivation time depending wheather the reactivation has been previously started but still the communication is not ready on time. The PPA overheads are also varying and depend on pattern size and number of all possible patterns detected during the execution. We used $uthash$ [15] hash table to store the pattern objects where pattern is used as a key. In Table IV are shown the average overheads of PPA through the chosen HPC applications and benchmarks. Although the overhead per MPI call on first sight can seem very big, 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.

TABLE IV
PPA OVERHEADS WHEN RUN WITH 16 MPI PROCESSES AVERAGED OVER ALL MPI PROCESSES

HPC application	MPI calls when PPA is invoked	Overhead per MPI call when PPA invoked (μs)	Overhead per all MPI calls (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

V. RELATED WORK

Power consumption optimization in the interconnection network has become very important target for HPC system designers. Hoefler [5] gives an overview of the power problem, and the related aspects of interconnection power, with focus on supercomputers. Power models for interconnection network, characterizing the power profile of network routers and links, have been proposed enabling further research into power-efficient techniques [19]. Several power reduction techniques have been proposed, where most of them are based on DVS. Shang et al. [7] proposed history-based DVS policy, where past network utilization has been used to predict future traffic. Soteriou at al. [20] propose software techniques that form an extension to the parallelizing compiler flow, which statically generate DVS instructions that later will direct run-time network power reduction.

Other techniques are based on turning off unused communication links or links with low utilization. Alonso et al. [8] propose a power saving mechanism for regular interconnection networks that are built with a high degree switches, where each network dimension is formed of various links in parallel. The idea is to turn off and on the links that compose trunk link as a function of network traffic. All links but one can be turned off. Therefore, connectivity in the network is maintained, which allows the use of same routing algorithm regardless of power reduction level. In the work of Kim et al. [21] DVS technique is complemented with powering down under-utilized links. The use of adaptive routing algorithm is required in order to avoid deadlocks.

Lim et al. [22] use MPI run time system to reduce CPU power consumption during the communication phases. The run time system identifies communication regions and select processor frequency in order to minimize energy-delay product. This work is the most similar with ours although they apply the power reduction on CPU during the communication phases.

Li et al. [23] proposes a compiler-directed communication link shut-down strategy. It determines the last use of communication links at each loop nest and insert link shut-down instructions. Each turned-off link is reactivated upon the next access to it. Our approach is similar in the way that we also issue shut-down instructions after an analization of inter-communication patterns. In our approach this is done during the runtime without requiring any modification in the source code while here explicit turn-off instruction are insterted during the compilation process.

Jian et al. [24] work is focused on non-prediction power-saving techniques. Links are powered up just before they are needed, by relaying 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. This approach is similar to ours in the way of powering on the data links. The difference is that 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], 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, the HPC applications can not afford such performance loss.

Saravanan et al. [25], provided 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

In this paper, we have evaluated software-directed approach to manage power consumption in interconnection links. We propose a power saving mechanism which in combination with runtime support can make network power consumption more proportional to the amount of data it is transmitting. The main goal of the system is to avoid indroducing the toggle delay in the system when switching on/off IB lanes. The idea is to detect on the fly the regions of the program that have repeatable communication patterns, on which basis the prediction of the following regions can be realized with relatively high precision. Also by declaring the minimum GT value we avoid regions where shifting to low-power mode would be useless. This prevents introducing in the system an unnecessary overheads which could provoke siginificant increase in execution times. Our system can be executed within PMPI layer enabling current MPI programs to be run without any source code modification. Also, the principles of our system are not restricted to Infiniband technology, as many modern interconnect technologies are built from multiple lanes (e.g. 40 GbE Ethernet has four lanes at 10 Gb/s each, but there is as yet no standard to turn lanes on and off individually). The overall results show the possiblity for significan power savings in IB switches of up to 33% with the almost neglible increase in the execution time (around 1%).

We have evaluated the possible power savings on strong scaling runs which led to smaller power reduction with larger number of processors. Therefore, we are expecting that our system would benefit more in weak scaling runs. With our system we amortize the negative effects of switching on/off, but to be fully profitable it is important that computation phases are considerably longer than the transition delay.

Creating the mechanism that can efficiently predict when to fully power up a link we open a possibilites for further power savings in the swithes by powering down other elements in the switch. Powering up these elements is much longer and can take up to a millisecond. Therefore, with an accurate system these delays would be in most cases avoided and thus negative effects on overall execution times. For future work, we will try to evaluate this scenario when powering down of other elements in the switch is taken into account.

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

[12] 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.

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

[14] 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.

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

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.