# Energy Efficient HPC Networks with On/Off Links

Francisco J. Andújar[1], Salvador Coll[1], Marina Alonso[1], Juan-Miguel Martínez[1], Pedro López[1], José L. Sánchez[1], Francisco J. Alfaro[1], Raúl Martínez[1]

[a]*Department of Computer Engineering, Universitat Politècnica de València, Valencia, Spain*
[b]*Instituto de Instrumentación para Imagen Molecular (I3M), Universitat Politècnica de València, Valencia, Spain*
[c]*Computing System Department, University of Castilla-La Mancha, Albacete, Spain*
[d]*Amazon Lab126, Cupertino, California, United States*

**Abstract**

Achieving an optimal performance/energy ratio is a challenge for massively parallel computer architects, and in particular for the interconnection network designers. The k-ary n-cube is one of the most popular topologies used in the largest current supercomputers. In this paper, we present a study that considers two alternatives to build k-ary n-cube topologies taking advantage of the high-radix switches currently available: a topology with more dimensions and one NIC per router, or a topology with less dimensions, link aggregation and several NICs per router. The fact of using link aggregation eases the implementation of simple power consumption reduction techniques. Using a simple power model, we evaluate by trace-driven simulation the impact on energy and performance of several network sizes for both topology proposals. In order to do a fair comparison, we keep fixed the theoretical network bandwidth.

*Keywords:* ...

## 1. Introduction

One of the most popular topologies in the largest current supercomputers is the $k$-ary $n$-cube, also known as $n$-dimensional torus [**?** ]. The torus topology has a fixed switch radix that facilitates the network fabric implementation, it is scalable and has linear cost of expansion. Moreover, the torus provides multiple paths for every pair of source/destination nodes in such a

way that fault tolerance and load balancing become feasible. For these reasons, several commercial interconnection systems allow to implement a torus network.

For example, 3D tori are supported by EXTOLL switches [**?** ] and the interconnection network of the Cray CS Series supercomputers [**?** ], while the interconnection network of the IBM Blue Gene/Q [**?** ] implements a 5D torus. In November 2017, there are three supercomputer machines using the torus topology in the top ten Top500 list [**?** ] and eight in the top ten Graph500 list [**?** ].

For a given number of switches, the performance of a torus network directly depends on its number of dimensions. The higher the number of dimensions, the lower the distances among nodes and therefore, the higher the network performance. The path diversity is also increased [**?** **?** ], improving the efficiency of adaptive routing algorithms.

However, as shown in the previous examples, commercial torus networks usually have a low number of dimensions, since wiring becomes more complex as the number of dimensions increases. Indeed, Dally [**?** ] and Agarwal [**?** ] showed that under fixed bisection and chip packaging, lower radix networks offer lower packet latency. Scott and Goodman [**?** ] introduced link pipelining in the network model, favoring a slightly higher dimensionality for large networks. In addition, multiple scientific applications use 3D mathematical models, whose communication patterns naturally fit in a 3-dimensional torus. The combination of ease of wiring with the use of 3D models makes low dimensional tori very appropriate topologies for designing an interconnection network.

Currently, switches with a high number of ports (i.e. high-degree switches) are commercially available [**?** **?** ]. We can take advantage of these high-degree switches to build low dimensional tori using the link aggregation concept (also denoted as link trunking). It consists in connecting every pair of adjacent switches by means of two or more physical channels. Two design approaches are possible. On one hand, we can combine several physical channels to work as a single wider physical link, therefore increasing the channel bandwidth. For example, the 4X QDR links on Mellanox products are composed of four QDR lanes [**?** ] and therefore, four flits of the same packet are transmitted in parallel by the 4X link. On the other hand, we can use the channels as independent links, i.e. the ports of a trunk link transmit different packets, in order to increase path-diversity and routing flexibility. For example, the Cray Gemini [**?** ] uses 10 links for building a 3D torus: 4 links for the X and

Z dimensions, and only 2 links for the Y dimension. In addition, Gemini has two nodes connected to each router, reducing the number of routers in the network. Note that combining both design approaches is also possible. In fact, the internal router of Cray Gemini has 48 ports [**?** ]: 8 ports are used to communicate the router with the 2 Gemini NICs, while the remaining 40 ports comprise the 10 Gemini links, using 4 ports per link.

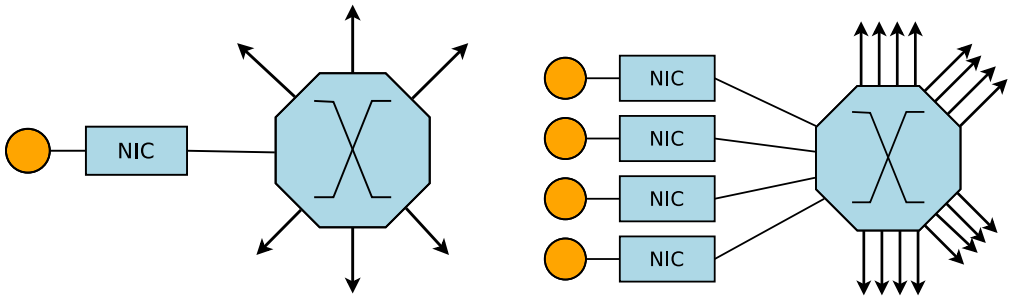
The use of trunk links increases switch to switch bandwidth. We can exploit this increased bandwidth by attaching several NICs to every switch (as stated above, the Cray Gemini attaches two NICs). An interesting observation is that, for a fixed system size (in number of NICs -i.e., computing nodes-), the number of network switches in a torus is reduced by the number of NICs attached to each switch. Of course, switches with a higher number of ports are required.

Network performance is not the only parameter to take into account when designing the network. A trade-off between performance and cost is usually required. Regarding cost, we must take into account not only the cost of network design and deployment but also exploitation costs. Leaving aside the possible network failures, this cost greatly depends on the network power consumption.

From this point of view, for a given network size $N$, two design options are possible. The first one is using a high dimensional network without link aggregation and one NIC per router. The extreme case would be a hypercube[1] with $\log_2 N$ dimensions. The second option is using a lower dimensional network, with link aggregation, several NICs per router and a lower number of routers. The first option requires a higher number of smaller switches, while the second requires a lower number of bigger switches. To do a fair comparison, the bisection bandwidth of both design choices should be the same, in order to keep constant the theoretical network bandwidth. Notice, though, that there may exist several intermediate design options that use different levels of link aggregation.

As an example, let's consider a 64-node network. Using the first approach, we could build a $4 \times 4 \times 4$ 3D torus with 64 switches with one computing node per switch. Each switch has 7 links (one port per each network direction plus one port to attach the computing node). Figure **??** shows this configuration.

[1]Remember that an hypercube is an $n$-dimensional torus with 2 nodes in each dimension. Therefore, the number of dimensions of the hypercube is $\log_2 N$.




(a) 7-port switch for $4 \times 4 \times 4$ torus (b) 20-port switch for $4 \times 4$ torus

Figure 1: Torus node configuration for building a 64-node network.

The network has $64 \times 6 = 384$ links, an average distance of $n \times \frac{k}{4} = 3 \times \frac{4}{4} = 3$ and a bisection bandwidth of $2 \times k^{n-1} = 2 \times 4^2 = 32$ bidirectional links.

Using the second approach, we could build a $4 \times 4$ 2D torus with 16 switches, with 4 trunk links and 4 computing nodes per switch. Each switch has 20 links (4 ports per each network direction plus 4 ports to attach 4 computing nodes). Figure **??** shows this second network configuration. This network has $16 \times 16 = 256$ links, an average distance of $2 \times \frac{4}{4} = 2$ and a bisection bandwidth of $4(\#$ of trunk links$) \times 2 \times k^{n-1} = 4 \times 2 \times 4^1 = 32$ bidirectional links.

Although both networks have the same bisection bandwidth, they will show different behaviors. The network with aggregated links (the 2D torus in the previous example) should have lower message latencies under low traffic loads because it has lower diameter than the network with more dimensions. However, the switch allocator performance decreases when the number of ports increases [**?** ]. After reaching certain traffic load, the network with more dimensions (the 3D torus in the previous example) should obtain better performance since its switches allow to achieve greater throughput as they are smaller. On the other hand, although the 3D network will achieve better performance under heavy traffic loads, the 2D network still has a lower power consumption because it has less network links (see Sections **??** and **??** for details).

The questions we try to answer in this paper are: which network is overall more energy efficient? A high-dimensional network with single links or a low-dimensional network with aggregated links? Is the performance loss for high loads by the network with aggregated links acceptable if it consumes lower energy? Since the energy consumed for running an application depends on both the system power consumption and the execution time, these are not

trivial questions. Although a lower dimensional network with aggregated links has lower power consumption, the execution time could be longer. This might increase the energy consumption of the system.

Another issue that must be taken into account is the fact that interconnection networks can provide dynamic mechanisms to save energy [**?** **?** ]. In particular, the mechanism proposed by Alonso [**?** ] turns off links when low network utilization is detected. This mechanism is very well suited for networks with trunk links, since turning off individual links from trunk links, while keeping at least one operating link, maintains the topology and does not require changes in the routing algorithm. Similar techniques are used in real products for saving energy. For example, in Mellanox switches, a link-level power saving feature reduces the width of the trunk link when a fabric is underutilized [**?** ].

In this paper, we analyze the performance of different k-ary n-cube configurations to determine what configuration is more energy-efficient. To do so, we will analyze both performance and energy consumption applying dynamic power saving techniques. Performance of the different configurations will be determined by using application execution traces. The rest of the document is organized as follows. In Section **??** we describe the power consumption model. Section **??** presents the system and evaluation model. After that, we analyze and discuss network performance and energy evaluation results in Section **??**. Finally, in Section **??** we outline the conclusions and future work.

## 2. A simple power consumption model

As mentioned in Section **??**, our objective is to show the impact of the energy consumption of a high performance computing (HPC) platform (cluster or supercomputer) for different configurations of the interconnection network topology.

The study presented in this paper has been developed by simulation, using a tool that models the nodes and the network that interconnects them. To obtain energy results, we have included in the simulator a simple power consumption model where the contribution of the main components of the interconnection network is considered. Note that we are going to carry out a comparative study, and therefore it is not totally relevant to use the absolute power consumption of each system component, but to determine the fraction of the total power consumed by those components.

Using the simulation tool, we obtain the total energy consumed by an application running on the HPC platform from its execution time and the power consumption during the simulation. As is well known, the power consumption of many building blocks of a computing system is the sum of a static or fixed component, and a dynamic component, that varies according to their utilization. For this reason, during the simulation, data related to dynamic power is collected, in order to calculate the total power consumption at the end of the simulation.

Due to the relevance of the links in the performance and energy efficiency of the network, in order to determine the total power consumption of a switch, our model considers the power consumption of the links and the power consumption of the remaining switch logic. According to the state of the art, we consider the following general hypotheses:

- The switch power consumption increases linearly with the number of ports [**?** ].
- We assume two states for the switch ports: *wake-up* (or *turned on*) and *sleep* (or *turned off*), assuming a power-saving mechanism like Low Power Idle (LPI), proposed on the IEEE **E**nergy-**E**fficient **E**thernet standard (IEEE 802.3az, and from now on, the EEE standard) [**?** ]. LPI freezes transceiver state when it enters in sleep mode and restores it when the port is waked up, drastically reducing the power consumption and allowing to turn on/off the links in a few microseconds [**?** ]. Therefore:
  - Since the transceiver is working independently of the port is transmitting data or not, the port power consumption is 100% when it is turned on.
  - When the port is in sleep mode, it consumes a small part of the total power consumption.
  - During the transitions from one state to another, the port power consumption is 100%.
- We assume the power saving strategy proposed by Alonso et al [**?** ] for the aggregated torus topology. Briefly, this strategy always maintains one active port per aggregated link, turning on/off the remaining ports depending on the aggregated link utilization.

At the end of this section, we provide an equation that allows us to obtain the total energy consumed by the execution of a given application for each network configuration on the HPC platform. Previously, a set of definitions is introduced, related to the system components and their contribution to the total power consumption of such system.

### *2.1. Definitions*

We introduce the parameters we use to quantify both the main components of the system and their contribution to power consumption and total energy.

In order to compare networks with different number of routers/ports, we normalize the power consumption with respect to a "reference" network. Let's consider as an example the networks shown in Figure **??**. Considering the 3D torus network (Figure **??**) as the reference network, its maximum power consumption will be 1. According to our initial hypotheses, switch (and network) power consumption linearly increases with the number of ports. The ratio of ports between the 2D torus (Figure **??**) and the 3D torus is $\frac{16\times20}{64\times7} = 0.714$. Therefore, the maximum power consumption of the 2D torus network will be 0.714, although it could be even lower if power saving strategies were applied (e.g. turning off unused or underutilized links). For this reason, we have included several terms related to the reference network, that are denoted with the prefix REF.

- $N_{ports}$: Number of ports per switch.
- $N_{sw}$: Number of switches in the network.
- $REF_{ports}$: Number of ports per switch in the reference network.
- $REF_{sw}$: Number of switches in the reference network.
- $U^p_{port}$: Fraction of time that a port $p$ is turned on.
- $w^p_{Sport}$: Fraction of the power consumption that a port $p$ consumes while in *sleep* mode.
- $U_{cpu}$: Fraction of time that a CPU is running; i.e. the time that the CPU is not idle because its allocated task is performing communications.

- $w_{Snodes}$: Fraction of the power consumption that a node always consumes, independently of their CPUs load.
- $w^s_{ports}$: Contribution of the ports in a switch $s$ to the total power consumption of that switch.
- $w_{net}$: Contribution of the network to the total power consumption of the system.

*2.2. Power consumption model*

Let $W^s_{ports}$ be the fraction of the power consumption that all the ports consume in a switch $s$ with respect to the maximum power consumption of those ports. When a port $p$ is in sleep mode, only consumes $w^p_{Sport}$ of the total power consumption. Then, a port $p$ always consumes $w^p_{Sport}$, plus $(1-w^p_{Sport})$ when it is turned on. Therefore:

$$W^s_{ports} = \frac{1}{N_{ports}} \sum_{i=1}^{N_{ports}} \left( w^i_{Sport} + (1 - w^i_{Sport}) \cdot U^i_{port} \right)$$

Ports in a switch have the same characteristics and so $w^1_{Sport} = w^2_{Sport} = \cdots = w^{N_{ports}}_{Sport} = w_{Sports}$. Therefore,

$$W^s_{ports} = w_{Sports} + (1 - w_{Sports}) \frac{1}{N_{ports}} \sum_{i=1}^{N_{ports}} U^i_{port}$$

If the average value $U_{ports}$ for all $U^i_{port}$ is considered:

$$U_{ports} = \frac{1}{N_{ports}} \sum_{i=1}^{N_{ports}} U^i_{port}$$

we obtain

$$W^s_{ports} = w_{Sports} + (1 - w_{Sports}) \cdot U_{ports}$$

Once we have the port power consumption, we can calculate the proportion of the switch power consumption with respect its maximum power consumption. In order to simplify the model, we consider that the remaining logic of the switch always consumes the maximum power consumption independently of its utilization, i.e. the switch logic always consumes $(1-w^s_{ports})$. Therefore:

$$W^s_{sw} = (1 - w^s_{ports}) + w^s_{ports} \cdot W^s_{ports}$$

If we consider all switches in the network, we can obtain the network power consumption:

$$W_{net} = \frac{1}{N_{sw}} \sum_{i=1}^{N_{sw}} W_{sw}^{i} = \frac{1}{N_{sw}} \sum_{i=1}^{N_{sw}} ((1 - w_{ports}^{i}) + w_{ports}^{i} \cdot W_{ports}^{i})$$

Without loss of generality, and reasoning at network level in the same way as switch level, we consider that the contribution of switch ports to the total switch power consumption is the same for all switches ($w_{ports}^{1} = w_{ports}^{2} = \ldots w_{ports}^{N_{sw}} = w_{ports}$). Considering $W_{ports}$ as the average value of $W_{ports}^{i}$:

$$W_{net} = (1 - w_{ports}) + w_{ports} \cdot W_{ports}$$

However, this way of obtaining the network power consumption is only valid to compare network topologies using the same type of switch; i.e. the number of ports per switch must be the same in each topology. Since we want to compare networks with different number of switches or ports per switch, we need to normalize the power consumption with respect to a "reference" network.

As our initial hypothesis is that the power consumption increases linearly with the number of ports, once we have chosen the "reference" network, we can calculate the relative network power consumption:

$$W_{net} = ((1 - w_{ports}) + w_{ports} \cdot W_{ports}) \cdot \frac{N_{ports}}{REF_{ports}} \cdot \frac{N_{sw}}{REF_{sw}}$$

In order to obtain the total energy of the HPC platform, we need to consider the power consumption of the computing part, mainly due to the compute nodes. Taking into account the definitions above, and again considering the average behavior of the nodes, the fraction of the power consumption that all the nodes consume in the system, with respect to the maximum power consumption of those nodes, can be expressed as:

$$W_{nodes} = w_{Snodes} + (1 - w_{Snodes}) \cdot U_{cpu}$$

where we assume that the compute nodes are energy proportional, but also there is a fixed part of the total power consumption that is always consumed. The rest of the power consumption is proportional to the CPU utilization.

Then, considering the nodes and network power consumption, we can calculate the HPC platform power consumption:

$$W_{cluster} = w_{net} \cdot W_{net} + (1 - w_{net}) \cdot W_{nodes}$$

Table 1: Power model parametrization

| Parameter | $w_{Sport}$ | $w_{ports}$ | $w_{net}$ | $w_{Snodes}$ |
|---|---|---|---|---|
| Value | 0.1 | 0.65 | 0.15 | Variable |

$W_{cluster}$ provides the fraction of the maximum power (both network and nodes) consumed during the application execution and normalized with respect to the "reference" network. Finally, the energy consumed by an application is:

$$\begin{aligned} E_{net} &= W_{net} \cdot RunTime \\ E_{cluster} &= W_{cluster} \cdot RunTime \end{aligned}$$

*2.3. Parameter characterization*

Once the power model has been defined, we must select the values for the parameters that determine the fraction of power dissipated by the various network building blocks, as defined in our model. Table **??** summarizes all the selected values.

According to the IEEE Energy-Efficient Ethernet standard (IEEE 802.3az), the power consumption of an idle link[2] is estimated to be 10% of the link power consumption [**?** **?** ]. Therefore, we set $w_{Sport}$ to 0.1.

The weight of the link power consumption with respect to the switch power consumption is 65% and 63% for the Dell PowerConnect 5324 (24-port switch) and the Dell PowerConnect 6248 (48-port switch) [**?** ], respectively; 64% for an IBM Infiniband 8-Port 12X switch [**?** ] and 68% for the EXTOLL Tourmalet switch [**?** ]. According to that, we consider that 0.65 is a realistic estimation for $w_{ports}$.

Finally, we set $w_{net}$ to 0.15, since the network power consumption is 10%∼20% [**?** **?** ] of the full system. We have not fixed $w_{Snodes}$, since we want to study the impact of CPU power dissipation in the final results. A realistic estimation of this parameter is 0.5 since even energy-efficient servers still consume half of their power while idle [**?** ].

[2]Note that, in our terminology, an idle link is equivalent to an off link, while an active link is equivalent to an on link.

## 3. System model

After presenting the power model, we describe the system model used as evaluation testbed. Section **??** outlines the switch architecture model while Section **??** shows the topologies selected for our experiments. Finally, Section **??** briefly explains the network load model.

### *3.1. Switch model*

We consider that all the switches in the network use the same technology, independently of their number of ports. The modeled architecture is not based on a single specific system, but it is realistic and representative of current state of the art HPC platforms since the design parameters have been chosen based on several commercial networks [**?** **?** **?** **?** **?** ].

The main specifications of the switch architecture are the following: *IQ* (*Input Queued*) switches [**?** ], *virtual cut-through* switching [**?** ], credit-based flow control and the three-stage allocation algorithm implemented in the IBM Blue Gene L [**?** ], with the only difference that our algorithm employs round-robin arbiters in all the allocator stages.

The data is transmitted in flits of 16 bytes, grouped in 8-flit packets (or 128-byte packets). The switch logic frequency is 625 MHz (i.e. the switch clock resolution is 1.6 ns). Since the switch crossbar can deliver one flit per cycle, each switch port offers a peak bandwidth of 10 Gbytes/s.

Table 2: Case studies

| Nodes | Topology | Dim. | Num. ports | Allocator latency | Port Aggr. | Num. Switches | Network ports | Port Ratio | Name (No PS) | Name (PS) |
|---|---|---|---|---|---|---|---|---|---|---|
| 64 | 3D torus | 4x4x4 | 7 | 3 | 1 | 64 | 448 | – | 3D-1X | – |
| | 2D torus | 4x4 | 20 | 5 | 4 | 16 | 320 | 0.714 | 2D-4X | 2D-4X-PS |
| | 1D torus | 4 | 48 | 6 | 16 | 4 | 192 | 0.428 | 1D-16X | 1D-16-PS |
| | k4-n3 | — | 8 | 3 | 1 | 40 | 320 | 0.714 | k4-n3 | k4-n3-PS |
| | k8-n2 | — | 16 | 4 | 1 | 12 | 192 | 0.428 | k8-n2 | k8-n2-PS |
| 256 | 4D torus | 4x4x4x4 | 9 | 4 | 1 | 256 | 2304 | – | 4D-1X | – |
| | 3D torus | 4x4x4 | 28 | 5 | 4 | 64 | 1792 | 0.777 | 3D-4X | 3D-4X-PS |
| | k4-n4 | — | 8 | 3 | 1 | 224 | 1792 | 0.777 | k4-n4 | k4-n4-PS |
| | k16-n2 | — | 32 | 5 | 1 | 24 | 768 | 0.333 | k16-n2 | k16-n2-PS |

The latency per hop is approximately 50 ns, slightly varying as a function of the number of ports. Since we assume that the switches are implemented

with the same technology, we can maintain fixed the latency of the switching units. The latency of input buffering and the port serializer does not depend on the number of ports. Considering a table-based routing algorithm, the latency mainly depends on the memory technology and the maximum network size supported by the architecture. The only switch unit with a variable latency is the switch allocator. Algorithms like iSLIP [**?** ] or our allocator based on the IBM Blue Gene L allocator [**?** ] are composed of several stages of multiple round-robin arbiters. Increasing the number of ports linearly increases the number of round-robin arbiters (and therefore the hardware complexity) but the number of stages is kept fixed and the increment of the round-robin arbiter latency is only logarithmic [**?** ]. This slightly increases the allocator latency. Table **??** shows the allocator latency, measured in cycles, for each router size.

Regarding the buffer organization, each input port has an input buffer of 1024 flits, or 16 Kbytes, statically split between the four virtual channels (VCs). The use of the VCs depends on the modeled topology:

- In the torus topologies, the virtual channels are employed to avoid deadlocks and to provide adaptiveness. In this case, the switch implements a fully-adaptive routing algorithm [**?** ]. Two of the four VCs are fully-adaptive VCs, while the remaining VCs are used as escape paths, implementing the DOR algorithm using two VCs to avoid deadlocks [**?** ].

- Since VCs are not necessary in the fat-tree topologies for avoiding deadlocks or providing adaptiveness, the VCs are used to implement DBBM [**?** ] and reduce the Head-Of-Line blocking, mapping the packets in the VCs using the function $Destination \ \% \ Number\_of\_VCs$.

Finally, we have implemented the trunk links using the second approach described in Section **??**; i.e. each trunk link comprises several independent ports transmitting independent packets. We have also implemented the power saving strategies described in [**?** ] and [**?** ] for the aggregated-link torus topologies and the fat-tree topologies, respectively[3]. Note that we have not employed a power saving strategy in the reference network (a torus network without port-aggregation). The main reason is to avoid great

[3]Although both topologies implements adaptive routing algorithms, note that an off port can not be chosen by the routing function.

performance penalty caused, not by the topology, but by the power saving strategy. Other strategies like turning-off the ports after a certain unused time has great performance penalties, while the Alonso et al. proposals has an insignificant impact of the performance since the network it is always fully-connected [**?** ].

Although our router is not related to any specific technology, we have used the time values specified in IEEE Energy-Efficient Ethernet standard [**?** ] to configure the delays for turning on (4.16 $\mu$s) and turning off (2.88 $\mu$s) a link. These delays are used in all the power-saving networks, independently of the network topology.

### *3.2. Case studies*

We have selected systems with 64 and 256 compute nodes. Table **??** shows the topologies evaluated for the same amount of compute nodes, indicating the most relevant parameters. Note that all the networks evaluated for each system size have the same bisection bandwidth.

The labels used to identify the topologies use the following nomenclature:

- Torus topologies: **n**D-**p**X, where **n** is the number of dimensions and **p** is port aggregation.
- Fat-tree topologies: k**x**-n**y**, where **x** is the k-arity of the fat-tree (then, the number of ports per switch is $2x$) and **y** is the number of stages.

Note that if a power saving mechanism is used in the network, the corresponding label ends with **-PS**. And finally, note that we have chosen the torus topology with the highest number of dimensions (and no aggregated links) as the "reference" system for both system sizes.

### *3.3. Network load*

The assessment of energy consumption by an HPC platform relies on the correct estimation of power consumption and execution time for a given load. Synthetic traffic patterns, typically used in performance evaluation, are not appropriate for this purpose since the difference among message latency on different network configurations cannot be translated into differences in execution time. As a result, for modeling the network load, we have used an open access trace-driven traffic model, called *VEF trace model* [**? ?** ]. The MPI traffic injected by parallel applications is captured in a trace file which will be later used to generate the traffic in the network simulator. VEF

traces model both MPI point-to-point and MPI collective communication primitives, using the collective communication algorithms implemented in Open MPI [**?** ].

Specifically, we have performed an evaluation using the VEF traces generated by parallel applications run in the supercomputer *GALGO* [**?** ]. For our test, we have selected the following applications, trying to illustrate different realistic scenarios[4]:

- *Namd* is a parallel application for simulating large biomolecular systems [**?** ]. The application maps logically the tasks in a 3D grid and the tasks communicate mainly with their neighbor tasks in the grid. For this reason its traffic pattern shows a great spatial locality. Our traces correspond to the STMV benchmark.

- *Gromacs* [**?** ] is a scientific application to perform molecular dynamics. Similarly to the the previous application, it shows a great spatial locality. We generated the trace using the input "d.poly-ch2" available in the Gromacs benchmark[5].

- *HPCC **MPI** **R**andom **A**ccess* [**?** ] (or MPIRA). Most of the communications are performed by MPI point-to-point primitives. The messages are uniformly distributed among all the tasks, being its traffic pattern very close to an uniform traffic pattern.

- *HPCC Linpack* [**?** ] is used to solve a dense system of linear equations. This application follows a specific pattern in which a specific task use to communicate always with the same tasks, following a ping-pong traffic pattern.

- *Graph500 benchmark* using the *replicated-csr* implementation, a scale factor of 20 and an edge factor of 16 [**?** ]. All the communications are generated by MPI collective communications that generate a great exchange of data among tasks. This application generates the highest network load of the three applications tested.

[4]All the VEF traces described in this work and the software needed to run the VEF traces are available free at the VEF website [**?** ].

[5]`http://www.gromacs.org/About_Gromacs/Benchmarks`

*Single trace evaluation*

In first place, we have evaluated each system executing a single application each time. We have evaluated the five applications shown in the previous section, using 512-task traces for simulate all the applications. In order to represent a more realistic environment, we simulate multicore nodes whenever possible. That is, for testing the 64-node networks, we have simulated 8-core nodes, but for testing the 256-node network, we have simulated only two cores per node. Unfortunately, the trace generation is limited by the size of the supercomputer GALGO, thus we can only simulate nodes with a small number of cores.

*Trace scheduler evaluation*

After the first evaluation, we have developed an oblivious trace scheduler to solve the limitation of the number of tasks per trace and to evaluate the networks under a more realistic environment. Given a set of traces, the operation of the trace scheduler is basic. In first place, the scheduler checks the number of available cores. In second place, it checks the number of tasks of each trace to determine what traces are selectable; that is, what traces can be executed with the available resources. Finally, the scheduler randomly choses a selectable trace and map it in the first free nodes. This process is repeated until all the traces are mapped or there are no free nodes to map another trace. In the second case, the scheduler will be executed again when a trace finishes and frees its resources.

For the trace scheduler evaluation, we consider that all the nodes has 8 cores; i.e. the 64-node network has 512 cores while the 256-node network has 2048 cores. We have evaluated three different set of traces, combining traces of the five applications shown in Section **??** with three different task sizes: 128, and 256 and 512 tasks. All the trace sets has the same number of applications (15 traces: 5 applications multiplied by 3 task sizes). However, since the Graph500 application has the high network load, each set increase the number of Graph500 traces to increase the total network load. The evaluated set of traces are the following:

- **Set A**: Namd, Gromacs, Linpack, MPI Random Access and Graph500.
- **Set B**: Gromacs, Linpack, MPI Random Access and Graph500 (x2).
- **Set C**: Linpack, MPI Random Access and Graph500 (x3).

Finally, note that since the scheduler randomly selects the order of the trace execution, we have performed 30 different executions of each trace set and each topology for taking the results.

## 4. Evaluation

In this section, we show the results of the performance and energy consumption evaluation. Section **??** shows the results for each application individually executed, while Section **??** shows the results obtained using the trace scheduler.

### *4.1. Single trace evaluation*

*64-node network*

In first place, we explain the results obtained for the 64-node network. Figure **??** shows the *Runtime*, $U_{cpu}$ and $E_{net}$ obtained for each application, while Figure **??** shows $E_{cluster}$ as a function of $w_{Snodes}$. Note that *Runtime* and $E_{net}$ are normalized with respect to the reference network, while $U_{cpu}$ shows the CPU utilization obtained in each simulation. Finally, remember that $w_{Snodes}$ indicates the fraction of power consumed by compute nodes while idle. That is, if $w_{Snodes} = 0$, we have the ideal case of totally energy-proportional nodes, while when $w_{Snodes} = 1$ we have the worst case: nodes always consume the same power regardless of their utilization.

Namd, Gromacs, Linpack and MPIRA applications obtain similar results. Since all the network topologies achieve the same performance, the network energy consumption directly depends on port ratio (shown in Table **??**). For this reason, the networks with the same port ratio practically consumes the same energy. The 2D torus and 3-stage fat-tree consumes 71.5% of the reference network energy (or 50% with the power-saving strategies), while the 1D torus and the 2-stage fat-tree only consumes 43% (or 32% with the power-saving). Only the torus networks has a little performance penalty in the MPIRA application (less than 1% in the 2D torus and 3% in the 1D torus), and for this reason, they consume around 1$\sim$3% more energy than the fat-tree with the same port ratio. Moreover, note that the power-saving strategies has no significant impact on performance in these applications.

However, there are some differences when we analyze $E_{cluster}$. In Namd, Gromacs and Linpack, with no significant performance penalty, $E_{cluster}$ depends on the port ratio, getting the 1D torus and the 2-stage fat-tree network the lower energy consumption, followed by the 2D torus and 3-stage fat-tree.

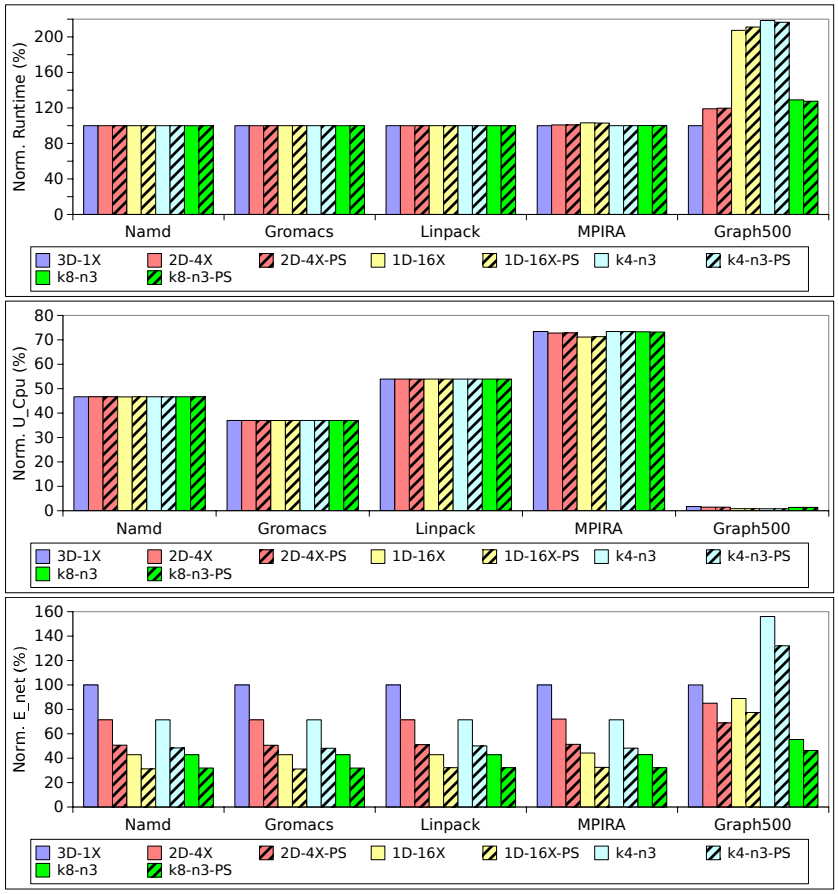
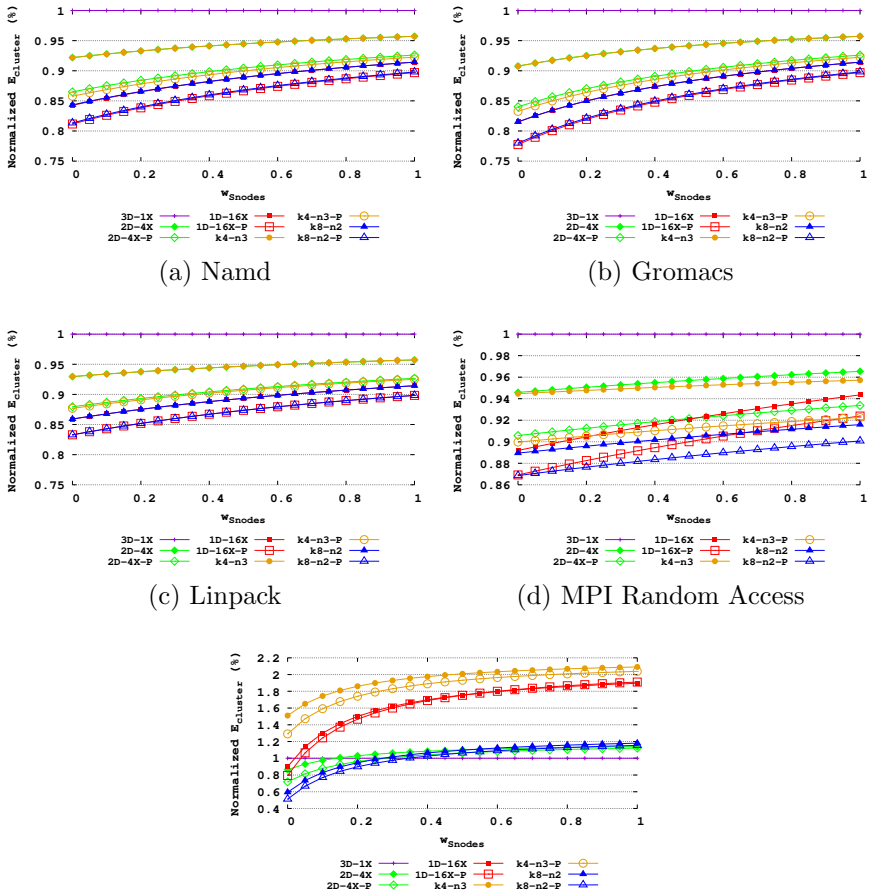



Figure 2: Runtime, $U_{cpu}$ and $E_{net}$ obtained for 64-node networks

But the totally energy saved also depends on the CPU utilization. As seen, the applications with lower $U_{cpu}$ can saving more energy, depending on how energy-proportional are the nodes. This is very reasonable: if $U_{cpu}$ decreases, the node energy consumption also decreases, doing that $E_{net}$ represent a more significant proportion of $E_{cluster}$. Regarding MPIRA, the torus consume more energy than the fat-tree with the same port ratio due to the performance penalty, being the increasing of $E_{cluster}$ higher in the 1D torus.

Finally, Graph500 provides captivating results. As seen in the figures, the network is the bottleneck of the system since the CPU is idle most of the running time. Moreover, the execution time on the 1D torus and the 3-stage fat-tree is unacceptable (around 208∼218%, depending on the case).




(a) Namd (b) Gromacs

(c) Linpack (d) MPI Random Access

(e) Graph500

Figure 3: $E_{cluster}$ for 64-node networks

This performance penalty greatly increases $E_{net}$ and $E_{cluster}$ in the 3-stage fat-tree. In the 1D torus, although $E_{net}$ is reduced due its reduced port ratio, $E_{cluster}$ also greatly increases.

The 2D torus and the 2-stage fat-tree also have a performance penalty: 19% and 28%, respectively. Considering the network subsystem alone, both topologies achieve significant energy savings (14%/31% in 2D torus, 44%/53% in 2-stage fat-tree). But due to he performance penalty, the energy efficiency highly relies on the compute nodes. Both topologies only achieve energy saving if the nodes are energy-proportional. Note that $w_{Snodes}$ must be lower than $0.2 \sim 0.4$, depending on the case, to save energy. In other case, the energy consumption increases.

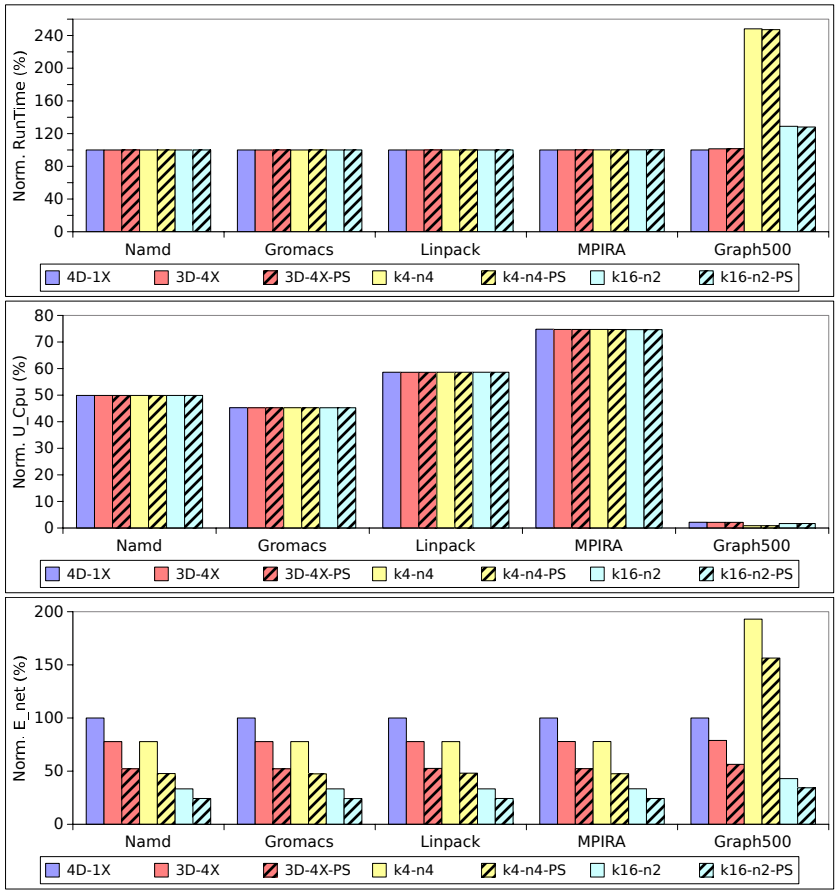
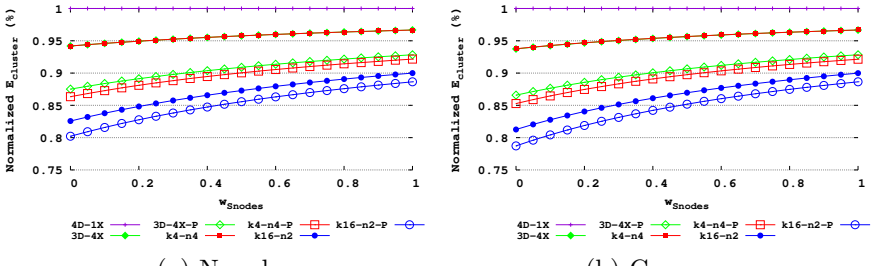



Figure 4: Runtime, $U_{cpu}$ and $E_{net}$ obtained for 256-node networks

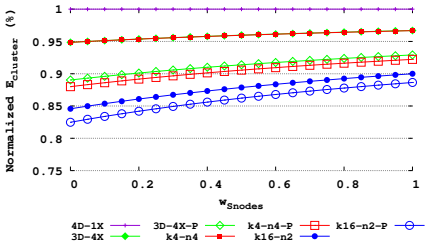
In summary, the 1D torus and the 3-stage fat-tree are the worst design options. Although both networks can save energy under low network loads, the performance penalty and the energy consumption increasing is unacceptable under high loads. Regarding the 2D torus and the 2-stage fat-tree, both topologies obtains significant energy savings under low loads. Under high loads, their energy efficiency depends on the energy-proportionality of compute nodes. However, the energy savings on the low load scenarios could compensate the energy increasing on the high load scenario. Finally, note that the 2-stage fat-tree achieves the greatest energy saving, although under Graph500 load it also has a greater performance penalty than the 2D torus.



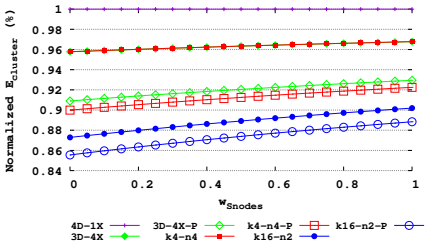

(a) Namd



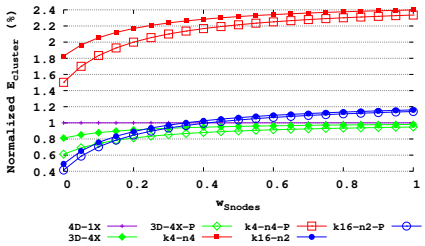

(b) Gromacs




(c) Linpack




(d) MPI Random Access




(e) Graph500

Figure 5: $E_{cluster}$ for 256-node networks

*256-node network*

Now we explain the results obtained for the 256-node network. Figure **??** shows the *Runtime*, $U_{cpu}$ and $E_{net}$ obtained for each application, while Figure **??** shows $E_{cluster}$.

The results are similar than the previous network size. All the topologies achieve the same performance under Namd, Gromacs, Linpack and MPIRA applications, depending the energy consumption on the port ratio. The aggregated-link 3D torus obtains the same power saving than the 4-stage fat-tree, although the last one consumes less energy when power-saving strategies are applied. In every case, the 2-stage fat-tree obtains the greatest power savings.

Regarding the Graph500, all the topologies shows a performance penalty with respect to the reference network. The execution time of 4-stage fat-tree (240%) make its results unacceptable, both in performance and energy consumption. The 2-stage fat-tree has a performance penalty of 28%. In the same way than the 64-node network, its energy efficiency relies on the energy-proportionality of the computes nodes. Only the 3D torus, with only 1.5% performance penalty, reduces the energy consumption independently of the value of $w_{Snodes}$.

In summary, the aggregated-link 3D torus achieves the best results. Although the 2-stage achieves great power savings under low loads, the 3D torus has no significant loss on performance in all the scenarios. And, as in the previous case, using the power saving strategies has no significant impact on performance.

### *4.2. Trace scheduler evaluation*

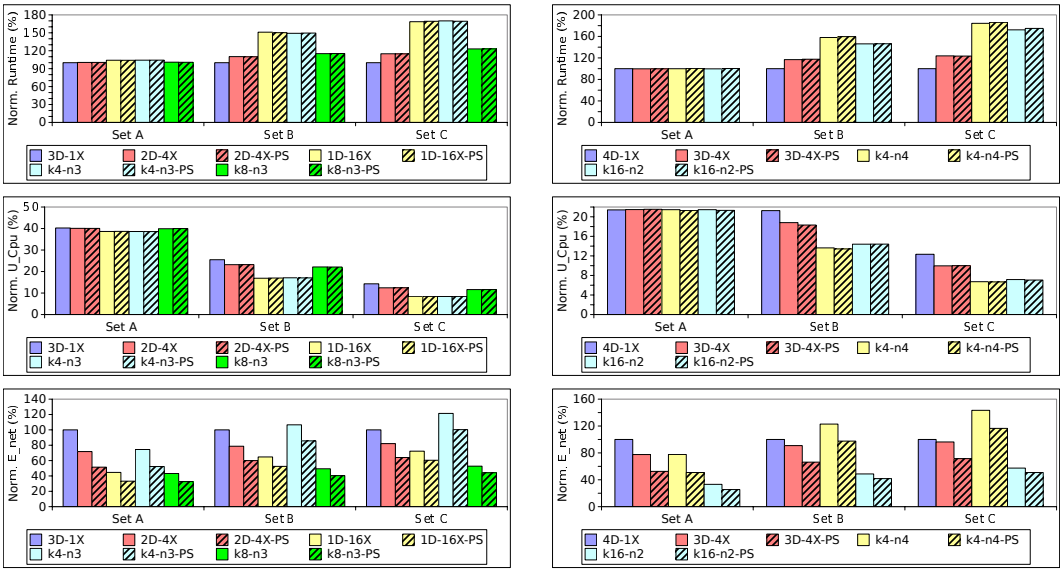
Finally, we show the results obtained using the trace scheduler. Figure **??** shows the *Runtime*, $U_{cpu}$ and $E_{net}$ obtained for each trace set, while Figure **??** shows $E_{cluster}$.

#### *64-node network*

As seen in Figure **??**, the results are consistent with the obtained in the single trace evaluation. In the application Set A, which has the lowest network load, the energy consumption mainly depends on the network port ratio. The 2D torus and the 8-ary 2-tree has no significant impact on performance, while the 1D torus and the 4-ary 3-tree increase the execution time around 4.5%. This performance penalty makes that, the 8-ary 2-tree consumes less energy than 1D torus, and the 2D torus consumes less energy than the 4-ary 3-tree[6].

When the network load increases, the performance of the 4-ary 3-tree and the 1D torus is enormous, being their execution times 150% for Set B and 170% for Set C, approximately. This great performance penalty makes these two network consume many more energy than the reference network. Only in the nodes are very energy-proportional ( $0 \leq w_{Snodes} < 0.15$) these networks can save energy, but the performance penalty is still unacceptable.

[6]Remember that the 8-ary 2-tree and the 1D torus has the same port ratio (0.428), just like the 4-ary 3-tree and the 2D torus (0.777).




(a) 64 nodes (b) 256 nodes

Figure 6: Runtime, $U_{cpu}$ and $E_{net}$ using the trace scheduler

The other two networks also achieves a lower performance. The 8-ary 3-tree increases the execution time 15% and 23% for Sets B and C, while the performance penalty in the 2D torus is 10% for Set B and 15% for Set C. Both networks consume less energy than the reference network, although the fat-tree achieves a lower energy consumption ($40 \sim 44\%$ with power saving) than the 2D torus ($60 \sim 64\%$ with power saving). Since the 8-ary 3-tree has a lower power consumption than the 2D torus, the fat-tree network consumes less energy although it has a worse performance than the 2D torus.

However, this does not happen when consider the entire cluster. As in the single trace evaluation, both topologies can save energy with respect the reference network depending on the value of $w_{Snodes}$. Both networks save energy when $w_{Snodes} < 0.7$ and $w_{Snodes} < 0.45$ for Sets B and C, respectively.

If we compare the 2D torus and the 8-ary 3-tree when there are no power saving strategies implemented, the second always consumes less energy, independently of the value of $w_{Snodes}$. But using power saving, this depends on $w_{Snodes}$. For great energy-proportional nodes, the fat-tree achieves lower energy consumption. However, if the nodes are poorly energy-proportional, the increasing of energy consumption in the nodes is greater than the energy saved in the network, doing that the 8-ary 3-tree consumes more energy than

the 2D torus since it achieves a worse performance.

Summarizing, as happens in the single trace evaluation, the 3D torus (the reference network) is the best on terms on performance. The 1D torus and the 3-stage fat-tree are the worst design options due its performance penalty. Again, the 2D torus and the 2-stage fat-tree achieves significant energy savings under low loads, but their energy efficiency under high loads depends on the energy-proportionality of compute nodes. However, the values required for $w_{Snodes}$ to save energy are more reasonable than the networks evaluated only using the Graph500 benchmark. As expected, the energy savings on the low load applications cab compensate the energy increasing on the high load applications.

*256-node network*

The results are similar to the previous evaluation. For Set A, there are no topologies with a significant performance penalty, and therefore, the energy consumption depends on the network port ratio. In this case, the 16-ary 2-tree is best option in terms on energy and performance.

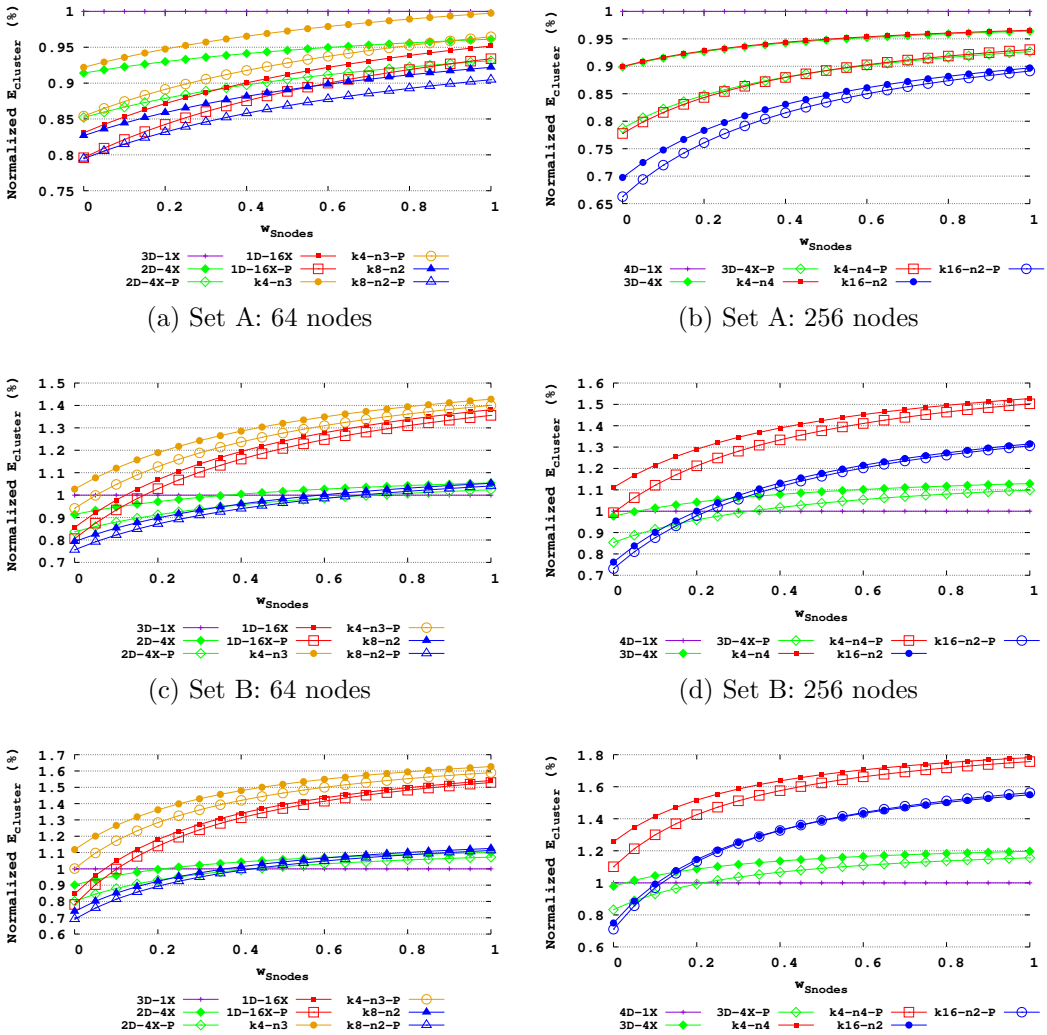
However, all the topologies get important performance penalties for Sets B and C. The 4-ary 4-tree never saves energy, independently of how energy-proportional are the nodes. The 2-stage fat-tree also needs a very energy-proportional nodes ($w_{Snodes} < 0.2$) to save energy. The 3D torus has the lowest performance penalty, but it still requires a energy-proportional nodes to save energy, although requires more reasonable values of $w_{Snodes}$ ($w_{Snodes} < 0.45$).

In summary, 16-ary 2-tree is only a good option under low network loads. For high loads, the 4D torus (the reference network) obtains the better results in terms on energy and performance.

## 5. Related work

Although power consumption in HPC interconnection networks has not received much attention, several works have proposed alternatives to improve energy efficiency of interconnection networks.

Using a dynamic power management (DPM) mechanism is proposed in [**?** ] for mesh topologies. Depending on network utilization, interconnection links are turned on or off. Traffic is redirected using alternative routes when specific links are turned off making use of a deadlock-free, fully-adaptive




(a) Set A: 64 nodes

(b) Set A: 256 nodes

(c) Set B: 64 nodes

(d) Set B: 256 nodes

(e) Set C: 64 nodes

(f) Set C: 256 nodes

Figure 7: $E_{cluster}$ using the trace scheduler

routing algorithm that guarantees packet arrival to destination. Significant power savings with moderate impact on performance indicate that more efficient dynamic switchable link designs would be critical.

Dynamic Adjusting Link Width (DALW) [**?** ] dynamically sets the available network bandwidth as a function of the network traffic. Unlike DPM that completely switch links off when they are not fully utilized, DALW is

based on reducing link bandwidth by narrowing link width. As the network topology is not modified, the same routing algorithm can be used simplifying the router design. Significant power consumption reduction can be achieved but message latency for low loads increases.

Work done in [**?** ], for torus networks, takes advantage of the availability of high-degree switches to connect them through several links (i.e., trunk links) and apply power consumption reduction techniques switching off links for low loads, as long as network connectivity is guaranteed, (i.e. every pair of switches should be connected through at least one active link). A set of utilization thresholds is used to control on/off links. In [**?** ], the authors propose a method to reduce interconnect power consumption by merging two techniques for network topologies based on aggregated links: firstly, dynamically switching on and off network links as a function of traffic; secondly, dynamically reducing the link bandwidth, with low traffic. As in the case of DALW, an advantage with respect to DPM is that the topology of the network is not modified so the same routing algorithm can be used.

Alonso et al. [**?** ] propose a mechanism to reduce power consumption in fat-tree networks while guaranteeing network connectivity. Simulation results show that to obtain a significant network power consumption reduction with minimum performance degradation is possible . In [**?** ] the authors improve the mechanism by defining a dynamic behavior that considers the switch status to modulate the sensitivity to traffic variations. The aggressiveness of the power reduction strategy can also be set. This solution significantly outperforms their previous proposals without additional cost.

Gunaratne et al. [**?** ] investigate adaptive link rate to reduce the energy consumption of an Ethernet link by adaptively varying the link data rate depending on utilization. Output buffer queue length thresholds and utilization monitoring are used to set data rate. An evaluation using an analytic model and simulation using synthetic traffic patterns shows that an Ethernet link can operate at a low data rate most of the time yielding to significant energy savings with small impact on packet delay.

Totoni et al. [**?** ] propose a hardware support for turning off interconnection links in software when they are not used during parallel applications execution and their management using adaptive runtime systems. Their proposal is evaluated by simulation for torus and dragonfly topologies.

PerfBound and DynamicFastwake mechanisms were presented in [**?** ] to minimize interconnect link energy consumption. A performance overhead bound is introduced to dynamically manage on/off based networks. The

techniques use local information already available at network switches and interfaces and require no change to the application and no communication with nodes and switches.

## 6. Conclusion

This paper explores the performance/energy dilemma in current high-performance and massive computing systems. We have provided evidence that an appropriate selection of the interconnect configuration is key to make an efficient use of a large scale computing platform. Our experiments, conducted on a selection of real application traces, representing typical high-demanding computing scenarios. As seen in the results, two main factor will determine what network topology will be more energy efficient: the network load and how energy-proportional all the nodes.

Under low network load scenarios, the best option is to build a fat-tree with a lower number of stages. This topologies has not significant impact on performance and has the lowest power consumption, getting the best results on energy consumption. Moreover, using power saving techniques we can get great power saving without significant performance penalties.

However, the results changes under high loads. In this case, the best performance is obtained by the torus with high number of dimensions; that is the reference network in all the experiments. Taking in account the energy consumption, the fat-tree with lower number of stages requires a very energy-proportional nodes to save energy.

Regarding the torus with aggregated links and lower number of dimensions than the reference network, it gets lower performance penalties than the fat-tree and require reasonable values if $w_{Snodes}$ to save energy. These topologies offer the best trade-off between performance and energy consumption. It achieves lower energy savings than the fat-tree under low network loads, but also achieves lower performance penalties under high network loads. Only in the highest network load scenario the most energy efficient network is the reference topology, but in the remaining scenarios it is the aggregated-link torus. But, in any case, if we are sure that the network load of or cluster will be low, the fat-tree with few stages ill be the best option.

Finally, note that, in all the evaluated topologies, both torus and fat-tree, the power saving strategies implemented reduce significantly the energy consumption without significant performance penalty. That is, independently

of the network topology selected for our cluster, the implementation of these strategies is a good idea.

Note that the achieved power-saving, depending on the topology selected, may look small. But remember that the total energy consumption of a supercomputer is huge. For example, let's consider the MareNostrum, the biggest Spanish supercomputer [**?** ] and currently the number 16 in Top500 list [**?** ]. It consumes 1.632 KW. Considering the price of one KWh as 0.07 euros (an extremely optimistic price in Spain), the cost of one year energy is around 1M euros. Saving only 5% of this bill is still a lot of money.

**Acknowledgment**

This work has been supported by the Spanish MINECO and European Commission (FEDER funds) under project TIN2015-66972-C5-1-R and project TIN2015-66972-C5-2-R. Francisco J. Andújar is also funded by the Spanish MINECO under a Juan de la Cierva grant FJCI-2015-26080.