COUNTDOWN: A Run-Time Library for Performance-Neutral Energy Saving in MPI Applications

COUNTDOWN: a Run-time Library for Performance-Neutral Energy Saving in MPI Applications

Daniele Cesarini, Andrea Bartolini, *Member, IEEE,* Pietro Bonfà, Carlo Cavazzoni and Luca Benini, *Fellow, IEEE*

Abstract—Power and energy consumption are becoming key challenges for the supercomputers' exascale race. HPC systems' processors waist active power during communication and synchronization among the MPI processes in large-scale HPC applications. However, due to the time scale at which communication happens, transitioning into low-power states while waiting for the completion of each communication may introduce unacceptable overhead.

In this paper, we present COUNTDOWN, a run-time library for identifying and automatically reducing the power consumption of the CPUs during communication and synchronization. COUNTDOWN saves energy without penalizing the time-to-completion by lowering CPUs power consumption only during idle times for which power state transition overhead is negligible. This is done transparently to the user, without requiring labor-intensive and error-prone application code modifications, nor requiring recompilation of the application. We test our methodology on a production Tier-1 system. For the NAS benchmarks, COUNTDOWN saves between 6% and 50% energy, with a time-to-solution penalty lower than 5%. In a complete production — Quantum ESPRESSO — for a 3.5K cores run, COUNTDOWN saves 22.36% energy, with a performance penalty below 3%. Energy saving increases to 37% with a performance penalty of 6.38%, if the application is executed without communication tuning.

Index Terms—HPC, MPI, profiling, power management, energy/power saving, idleness, DVFS, DDCM, C-states, P-states, T-states.

1 Introduction

In today's supercomputers, the total power consumption of CPUs limits practically achievable performance. This is a direct consequence of the end of Dennard's scaling, which in the last decade has caused a progressive increase in the power density required to operate each new processor generation at its maximum performance. Higher power density implies more heat to be dissipated and increases cooling costs. These altogether worsen the total costs of ownership (TCO) and operational costs: limiting de facto the budget for the supercomputer computational capacity.

Low power design strategies enable CPUs to trade-off their performance for power consumption employing low power modes of operation. These states obtained by Dynamic and Voltage Frequency Scaling (DVFS) (also known as performance states or P-states [1]), clock gating, or throttling states (T-states), and idle states which switch off unused resources (C-states [1]). Power states transitions are controlled by hardware policies, operating system (OS) policies, and with an increasing emphasis in recent years, at user-space by the final users [2], [3], [4], [5] and at execution time [6], [7].

While OS policies try to maximize the usage of the CPU — increasing the processor's speed (P-state) proportionally to the processor's utilization, with a specific focus on server and interactive workload — two main families of power control policies are emerging in scientific computing. The first is based on the assumption that the performance penalty can be tolerated to reduce the overall energy consumption [2], [3], [4], [8]. The second is based on the assumption that it is possible to slow down a processor only when it does not execute critical tasks: to save energy without penalizing application performance [5], [6], [7], [9]. Both approaches are based on the concept of application slack/bottleneck (memory, IO, and communication) that can be opportunistically exploited to reduce power and save energy. However, there are drawbacks which limit the usage of these concepts in a production environment. The first approach causes overheads in the application time-to-solution (TTS) limiting the supercomputer throughput and capacity. The second approach depends on the capability of predicting the critical tasks in advance with severe performance loss in case of mispredictions.

A typical HPC application is composed of several processes running on a cluster of nodes that exchange messages

- *D. Cesarini is with the Department of SuperComputing Applications and Innovation, CINECA, 40033 Casalecchio di Reno (BO), Italy (e-mail: d.cesarini@cineca.it).*
- *A. Bartolini is with the Department of Electrical, Electronic and Information Engineering "Guglielo Marconi", University of Bologna, 40136 Bologna, Italy (e-mail: a.bartolini@unibo.it).*
- *P. Bonfà is with the Department of Mathematical, Physical and Computer Sciences, University of Parma, 43121 Parma, Italy (e-mail: pietro.bonfa@unipr.it).*
- *C. Cavazzoni is with the Department of Chief Technology and Innovation Officer, Leonardo S.p.A., 00195 Roma, Italy (e-mail: carlo.cavazzoni@leonardocompany.com).*
- *L. Benini is with the Department of Information Technology and Electrical Engineering, Swiss Federal Institute of Technology in Zurich, 8092 Zurich, Switzerland and with the Department of Electrical, Electronic and Information Engineering "Guglielo Marconi", University of Bologna, 40136 Bologna, Italy (e-mail: lbenini@iis.ee.ethz.ch).*

through a high-bandwidth, low-latency network. These processes can access the network sub-system through a software interface that abstracts the network level. The Message-Passing Interface (MPI) is a software interface for communication that allows processes to exchange explicit messages abstracting the network level. Usually, when the scale of the application increases, the time spent by the application in the MPI library becomes not negligible and impacts the overall power consumption. By default, when MPI processes are waiting in a synchronization primitive, the MPI libraries use a busy-waiting mechanism. However, during MPI primitives the workload is primarily composed of wait times and IO/memory accesses for which running an application in a low power mode may result in lower CPU power consumption with limited or even no impact on the execution time.

MPI libraries implement idle-waiting mechanisms, but these are not used in practice to avoid performance penalties caused by the transition times into and out of low-power states [10]. As a matter of fact, there is no known low-overhead and reliable mechanism for reducing energy consumption selectively during MPI communication slack.

In this paper, we present COUNTDOWN[1], a run-time library, analysis tool, and methodology to save energy in MPI-based applications by leveraging the communication slack. The main contribution of this manuscript are:

i) An analysis of the effects and implications of fine-grain power management in today's supercomputing systems targeting energy saving in the MPI library. Our study shows that in today's HPC processors there are significant latencies in the hardware (HW) to serve low power states transitions. We show that this delay is at the source of inefficiencies (overheads and saving losses) in the application for fine-grain power management in the MPI library.

ii) Through the first set of benchmarks running on a single HPC node we show that: (a) there is a potential saving of energy with negligible overheads in the MPI communication slack of today's HPC applications; (b) these savings are jeopardized by the time that HW takes to perform power state transitions; (c) when combined with low-power states, Turbo logic can help improving execution time.

iii) The COUNTDOWN, which consists of a run-time library able to automatically track at fine granularity MPI and application phases to inject power management calls. COUNTDOWN can identify MPI calls with energy-saving potential for which it is worthwhile to enter a low power state, leaving low-wait-time MPI calls unmodified to prevent overheads caused by low power state transitions. We show that COUNTDOWN's principles can be used to inject DVFS calls as well as to configure the MPI run-time library correctly and take advantage of MPI idle-waiting mechanisms. COUNTDOWN works at execution time without requiring any off-line knowledge of the application, and it is completely plug-and-play ready: it does not require any modification of the source code and compilation toolchain. COUNTDOWN can be dynamically linked with the application at loading time: it can intercept dynamic linking to the MPI library instrumenting all the application calls to MPI functions before the execution workflow jumps to the library. The run-time library also provides a static version of the library which can be connected with the application at linking time. COUNTDOWN supports C/C++ and Fortran HPC applications and most of the open-source and commercial MPI libraries.

iv) We evaluate COUNTDOWN with a wide set of benchmarks and low power state mechanisms. In large HPC runs, COUNTDOWN leads to savings of 23.32% on average for the NAS [11] parallel benchmarks on 1024 cores and to 22.36% for an optimized QuantumESPRESSO (QE) [12] on 3456 cores. When we run QE without communication tuning the savings increases to 37.74%.

The paper is organized as follows. Section 5, presents the state-of-the-art in power and energy management approaches for scientific computing systems. Section 2 introduces the key concepts on power-saving in MPI phases of the application. Section 3 explains the COUNTDOWN run-time library and the characterizations of real HPC applications. Section 4 characterizes the COUNTDOWN library and report experimental results in power saving of production runs of applications on a Tier-1 supercomputer.

2 Background

In this section, we show the implications and challenges of transitioning into low power states (P/C/T-states) during synchronization and communication primitives for energy-savings on two practical examples.

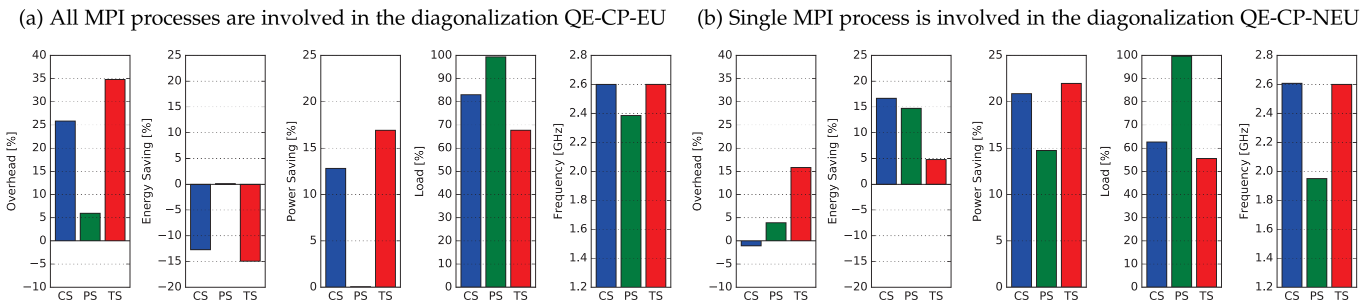
As a test platform we have used a compute node equipped with two Intel Haswell E5-2630 v3 CPUs, with 8 cores at 2.4 GHz nominal clock speed and 85W Thermal Design Power (TDP) and the production software stack of Intel systems. We use Intel *MPI Library 5.1* coupled with Intel *ICC/IFORT 18.0* as our toolchain. We choose the Intel software stack because it is currently used in our target systems as well supported in most HPC machines based on Intel architectures. We use a single compute node for the following exploration because this is a worst-case scenario for energy-saving strategies in MPI applications due the communications happen in a very short time.

For all the tests in this Section, we have been used a real scientific application, namely QuantumESPRESSO [12], which is a suite of packages for performing Density Functional Theory based simulations at the nanoscale and it is widely employed to estimate ground state and excited state properties of materials *ab initio*. For these single nodes tests, we used the CP package parallelized with MPI. We use QE because it is a paradigmatic application that shows the typical behaviors of HPC codes. QE main computational kernels include dense parallel linear algebra (diagonalization) and 3D parallel FFT, which makes the following exploration work relevant for many HPC codes[2].

To exploit the system behavior for different workload distribution in a single node evaluation, we focused the computation of the band structure of the Silicon along with

[1] Github Repository: https://github.com/EEESlab/countdown

[2] QE mostly used packages are: (i) *Car-Parrinello* (CP) simulation, which prepares an initial configuration of a thermally disordered crystal of chemical elements by randomly displacing the atoms from their ideal crystalline positions; (ii) PWscf (Plane-Wave Self-Consistent Field) which solves the self-consistent Kohn and Sham (KS) equations and obtain the ground state electronic density for a representative case study [13].



(a) All MPI processes are involved in the diagonalization QE-CP-EU

(b) Single MPI process is involved in the diagonalization QE-CP-NEU

Fig. 1. Overhead, energy/power saving, average load and frequency for QE-CP-EU (a) and QE-CP-NEU (b). Legend: C-state (CS), P-state (PS) and T-state (TS) mode. Baseline is busy-waiting mode (default mode) of MPI library.

the main symmetry. When executed by a user with no domain expertise and with default parameters, QE runs with a hybrid MPI parallelization strategy: one MPI process performs the matrix diagonalization kernel and the remaining ones perform the FFT kernel. We will later refer to this case as *QuantumESPRESSO CP Not Expert User* (QE-CP-NEU). Differently, when an expert user runs the same problem, he changes the parameters to better balance the workload by using multiple MPI processes to parallelize also the diagonalization kernel. We will later refer to this case as *QuantumESPRESSO CP Expert User* (QE-CP-EU). In the QE-CP-NEU case, when a single process works on the linear algebra kernel, the other ones remain in busy waiting on the MPI call. In the following text, we will compare fine-grain power management solutions with the busy-waiting mode (default mode) of the MPI library, where processes continuously poll the CPU for the whole waiting time in MPI synchronization points.

2.1 Wait-mode/C-state MPI library

Usually, MPI libraries use a busy-waiting policy in collective synchronizations to avoid performance penalties. This is also the default behavior of the Intel MPI library. This library can also be configured to release the control to the idle task of the operating system (OS) during the waiting time to leverage the C-states of the system. This allows cores to enter in sleep states and being woken up by the MPI library when the message is ready through an interrupt routine. In the Intel MPI library, it is possible to configure the wait-mode mechanism through the environment variable *I_MPI_WAIT_MODE*. This allows the library to leave the control to the idle task, reducing the power consumption for the core waiting in the MPI. The transitions in and out from the sleep mode induce overheads in the execution time.

In figure 1 are reported the experimental results, the wait-mode strategy is identify with *CS*. From it, we can see the overhead induced by the wait mode w.r.t. the default busy-waiting configuration, which worsens by 25.85% the execution time. This is explained by the high number of MPI calls in the QE application which leads to frequent sleep/wake-up transitions and high overheads. From the same figure, we can also see that the energy saving is negative, which is -12.72%, this is because the power savings obtained in the MPI primitives do not compensate for the overhead induced by the sleep/wake-up transitions. Indeed, the power reduction is of 12.83%. This is confirmed by the average load of the system, which is 83.02% as the effect of the C-states activity in the MPI primitives. The average frequency is 2.6GHz, which is the standard turbo frequency of our target system.

Surprisingly, the QE-CP-NEU case has a negative overhead (-1.08% overhead is a speedup). This speedup is given by the turbo logic of our system. Indeed, we can see that the average frequency is slightly higher than 2.6GHz, which means that the process doing the diagonalization can leverage the power budget freed by the other processes not involved in the diagonalization while they are waiting in a sleep state in the MPI run-time library. In figure 2, we report the average frequency of the process working on the diagonalization and the average frequencies of all the other MPI processes. In the target system, a single core can reach up to 3.2 GHz if only one core is running, this is what happens when all cores are waiting in a sleep state for the termination of the diagonalization workload. The benefit of this frequency boosting unleashed by the idle mode on the MPI library and the unbalanced workload can save up to 16.69% of energy with a power saving of 20.86%.

As a conclusion of this first exploration, we recognize that it is possible to leverage the wait mode of the MPI library to save power without increasing the execution time, but energy savings and impact on the TTS depends on the MPI calls granularity which can lead to significant penalties if the application is characterized by frequent MPI calls.

2.2 DVFS/P-state MPI library

To overcome the overheads of C-state transitions, we focus our initial exploration of the active low power states (C-state) and DVFS (P-state). Intel MPI library does not implement such a feature, so we manually instrumented all the MPI calls of the application with a *epilogue* and *prologue* function to scale down and raise up the frequency when the execution enters and exits from an MPI call. To avoid interference with the power governor of the operating system, we disabled it in our compute node granting the complete control of the frequency scaling. We use the MSR driver to change the current P-state writing *IA32_PERF_CTL* register with the highest and lowest available P-state of the CPU, which corresponds to the turbo and 1.2GHz operating points. In figure 1 we report the results of this exploration, where the P-state case is labelled with *PS*.

In the overhead plot, in figure 1.a, we can see that the overhead is significantly reduced w.r.t C-state mode, reduc-

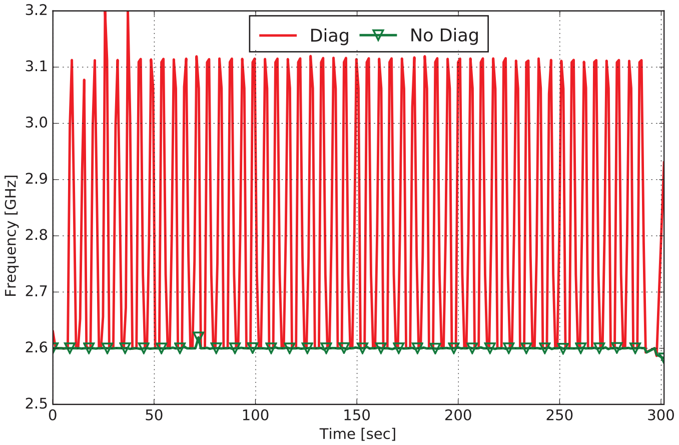


Fig. 2. Time plot of frequency for QE-CP-NEU identifies the frequency of the MPI process working on the diagonalization, *No Diag* is the average frequency of MPI processes not involved in the diagonalization.

ing the 25.85% overhead obtained previously to 5.96%. This means that the overhead of scaling the frequency is lower respect to the sleep/wake-up transitions cost. However, the energy and power savings are almost zero. Similarly to QE-CP-EU, this happens because all the MPI processes participate in the diagonalization, thus we have a high number of MPI calls with a very short duration. This is also confirmed by the average frequency, which does not show significant variations w.r.t. the busy waiting, with a measured average frequency of 2.4GHz. The load bar reports 100% of activity, which means that there is no CPU idle time as expected.

Focusing in the QE-CP-NEU case, in figure 1.b, the overhead is 3.88% which is reduced w.r.t. QE-CP-EU. In addition, in this case, we have significant energy and power saving, respectively of 14.74% and 14.75%. These savings are due to the workload unbalance and to the long time spent in the MPI calls from the processes not involved in the diagonalization. This is confirmed by the lower average frequency (1.95GHz). The load is unaltered as expected.

In conclusion, using DVFS for fine-grain power management instead of the idle mode allows controlling the overhead for both balanced and unbalanced workload better. However, the overhead is still significant and in HPC the TTS is the prime goal.

2.3 DDCM/T-state MPI library

One crucial question is: are the overheads of fine-grain power management strategies induced by the specific power management states? To answer this question, we considered duty-cycling low power states[3]. In Intel CPUs, DDCM is used by the *HW power controller* to reduce the power consumption when the CPU identifies thermal hazards. Similarly to [14], we use DDCM to reduce the power consumption of the cores in MPI calls. We manually instrumented the target as we did in the *prologue* function of each MPI call to configuring DDCM to 12.5% of clock cycles, which means for each clock cycle we gate the next 7; while in the *epilogue* function, we restore the DDCM to 100% of clock cycles, we control it by writing to the DDCM configuration register, called *IA32_CLOCK_MODULATION*, through the MSR driver.

In figure 1.a the DDCM results are reported with *TS* bars. Surprisingly, the overheads induced by T-states are greater than the wait mode and equal to 34.78%. As a consequence, the energy saving is the worst, leading to an energy penalty of 14.94%. The load is significantly reduced owing to the throttling, at an average of 67.78%, while the frequency is constant to 2.6GHz.

In figure 1.b, we report T-state results for QE-CP-NEU. Even for this unbalanced workload case, the T-states are the worst. T-state transitions introduce an overhead of 15.82% consequent of the power reduction, with a very small energy saving, only of the 4.75%, and a power saving of 21.97%. The load of the system is reduced to 55.45%, similar to the idle mode, and the frequency remained unchanged as expected.

As a matter of fact, we show that phase agnostic fine-grain power management leads to significant application overheads which may nullify the overall saving. Though, we need to bring knowledge of the workload distribution and the communication granularity of the application in the fine-grain power management. In the next Sections, we introduce the COUNTDOWN approach which addresses this issue.

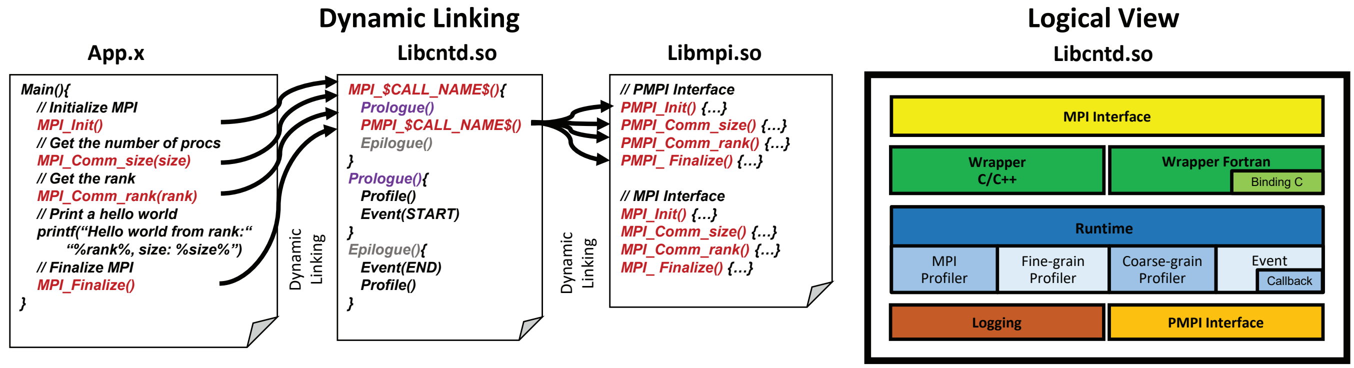
3 Framework

COUNTDOWN is a run-time library for profiling and fine-grain power management written in C language. COUNTDOWN is based on a *profiler* and on a *event* module to inspect and react to MPI primitives. The key idea in COUNTDOWN can be summarized as follows. Every time the application calls an MPI primitive, COUNTDOWN intercepts the call with minimal overhead and uses a timeout strategy [15] to avoid changing the power state of the cores during fast application and MPI context switches, where doing so may result only in state transition overhead without significant energy and power reduction.

In figure 3 the COUNTDOWN's components are depicted. COUNTDOWN exposes the same interface as a standard MPI library and intercepts all MPI calls from the application. COUNTDOWN implements two wrappers to intercept MPI calls: i) the first wrapper is used for C/C++ MPI libraries, ii) the second one is used for Fortran MPI libraries. This is mandatory since C/C++ and Fortran MPI libraries produce different assembly symbols. The Fortran wrapper implements (un)marshalling interfaces to bind MPI Fortran handlers into MPI C/C++ handlers.

When an application is instrumented with COUNTDOWN, every MPI call is enclosed in a corresponding wrapper function that implements the same signature. The wrapper function calls the equivalent PMPI call, but after and before a *prologue* and an *epilogue* routine. Both routines are used by the profile and by the event modules to support monitoring and power management, respectively. COUNTDOWN interacts with the *HW power manager* through a specific *Events* module in the library. The *Events* module can also be triggered by system signals registered as callbacks

[3] In this Section we also tried to use the Dynamic Duty Cycle Modulation (DDCM) (also known as throttling states or T-states) available in the Intel architectures which are characterized by lower overhead. DDCM has been supported in Intel processors since Pentium 4 and enables on-demand software-controlled clock modulation duty cycle.



```
Main(){
   // Initialize MPI
   MPI_Init()
   // Get the number of procs
   MPI_Comm_size(size)
   // Get the rank
   MPI_Comm_rank(rank)
   // Print a hello world
   printf("Hello world from rank:"
         "%rank%, size: %size%")
   // Finalize MPI
   MPI_Finalize()
}
```

```
MPI_$CALL_NAME$(){
   Prologue()
   PMPI_$CALL_NAME$()
   Epilogue()
}
Prologue(){
   Profile()
   Event(START)
}
Epilogue(){
   Event(END)
   Profile()
}
```

```
// PMPI Interface
PMPI_Init() {...}
PMPI_Comm_size() {...}
PMPI_Comm_rank() {...}
PMPI_ Finalize() {...}

// MPI Interface
MPI_Init() {...}
MPI_Comm_size() {...}
MPI_Comm_rank() {...}
MPI_ Finalize() {...}
```

Fig. 3. Dynamic linking events when COUNTDOWN is injected at loading time in the application and logical view of all the components.

for timing purposes. COUNTDOWN configurations can be done through environment variables, and it is possible to change the verbosity of logging and the type of HW performance counters to monitor.

The library targets the instrumentation of applications through dynamic linking, as depicted in figure 3, without user intervention. When dynamic linking is not possible COUNTDOWN has also a fallback, a static-linking library, which can be used while building the application, to add COUNTDOWN at compilation time. The advantage of using the dynamic linking is the possibility to instrument every MPI-based application without any modifications of the source code nor the toolchain, even without recompiling it. Linking COUNTDOWN to the application is straightforward: it is enough to configure the environment variable *LD_PRELOAD* with the path of the COUNTDOWN runtime library and launch the application as usual.

3.1 Profiler Module

COUNTDOWN allows extracting traces, which can be exploited to estimate application performance as [16]. COUNTDOWN uses three different profiling strategies targeting different monitoring granularity.

(i) The *MPI profiler* is responsible for collecting all information regarding the MPI activity. For each MPI process, it collects information on MPI communicators, MPI groups, and the coreId. In addition, the COUNTDOWN run-time library profiles each MPI call by collecting information on the type of the call, the entrance and exit times, and the data exchanged with the other MPI processes.

(ii) The *fine-grain micro-architectural profiler*, collects micro-architectural information at every MPI call along with the *MPI profiler*. This profiler uses the user-space RDPMC instruction to access the performance monitoring units implemented in Intel's processors. It monitors the average frequency, the time stamp counter (TSC), and the instructions retired for each MPI call and application phase. It can access up to 8 configurable performance counters that can be used to monitor user-specific micro-architectural metrics.

(iii) The *coarse-grain profiler* monitors a larger set of HW performance counters available in the Intel architectures. In Intel architectures, privileged permissions are required to access HW performance counters. Such a level of permissions cannot be granted to the final users in production machines. To overcome this limitation, we use the MSR-SAFE [17] driver, which can be configured to grant access to standard users on a subset of privileged architecture registers, while avoiding security issues. Moreover, we deploy in our target HPC system a plugin for the workload scheduler that restores the MSR registers of the compute nodes at the end of each power-aware job. At the core level, COUNTDOWN monitors TSC, instructions retired, average frequency, C-state residencies, and temperature. At the uncore level, it monitors CPU package energy consumption, C-state residencies, and temperature of the packages. This profiler uses Intel Running Average Power Limit (RAPL) to extract energy/power information from the CPU. The *coarse-grain profiler*, due to the high overhead needed by every single access to the set of HW performance counters monitored, uses a time-based sample rate. The *fine-grain micro-architectural profiler* at every MPI calls checks the timestamp of the previous sample of *coarse-grain profiler* and, if it is above Ts seconds, triggers it to get a new sample. These capabilities are added to the application through the *prologue* and *epilogue* functions as shown in figure 3.

COUNTDOWN also implements a logging module to store profile information in a text file that can be written in local or remote storage. While the log file of MPI profiler can grow with the number of MPI primitives and can become significant in long computation (thus the information is stored in binary files), the logging module also reports a summary of this information in an additional text file.

3.2 Event Module

COUNTDOWN interacts with the *HW power controller* of each core to reduce the power consumption. It uses MSR-SAFE to write the architectural register to change the current P-state independently per core. When COUNTDOWN is enabled, the *Events* module selects the performance level at which to execute a given phase.

COUNTDOWN implements a timeout strategy through the standard Linux timer APIs, which expose the system calls: *setitimer()* and *getitimer()* to manipulate user-space timers and register callback functions. This methodology is depicted in figure 4 in the top part. When COUNTDOWN encounters an MPI phase, which opportunistically can save energy by entering a low power state, *registers* a timer callback in the *prologue* function (Event(start)), after that the

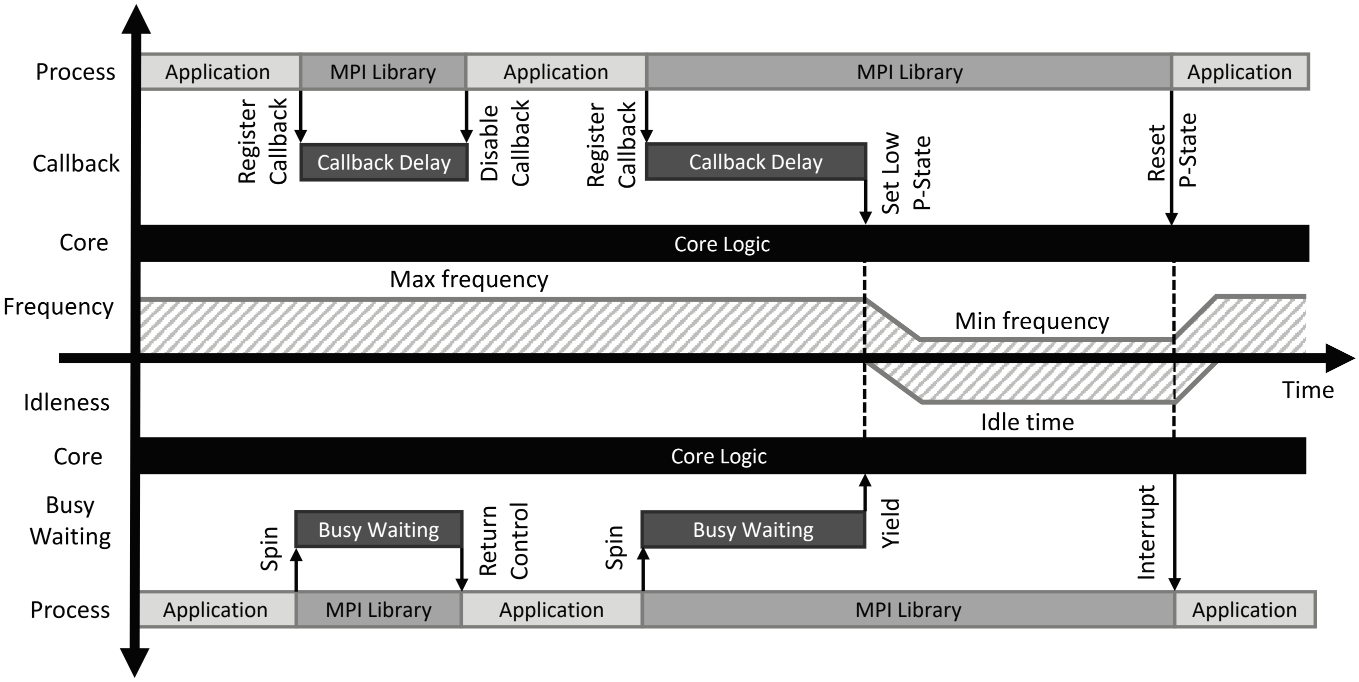


Fig. 4. On the upper side is depicted the timer strategy utilized in COUNTDOWN, while in the lower side is depicted the idle-wait mode with timer implemented in the Intel MPI library.

execution continues with the standard workflow of the MPI phase. When the timer expires, a system signal is raised, the "normal" execution of the MPI code is interrupted, the signal handler triggers the COUNTDOWN *callback*, and once the callback returns, execution of MPI code is resumed at the point it was interrupted. If the "normal" execution returns to COUNTDOWN (termination of the MPI phase) before the timer expiration, COUNTDOWN *disables* the timer in the *epilogue* function and the execution continues like nothing happened. The callback can be configured to enter the lower T-state (12.5% of load), later referred to as *COUNTDOWN THROTTLING*, or in the lower P-state (1.2GHz) later referred to as *COUNTDOWN DVFS*.

Intel MPI library implements a similar strategy, but it relies on the sleep power states of the cores. Its behavior is depicted in the bottom part of figure 4. If the environment variable *I_MPI_WAIT_MODE*, presented in Section 2.1, is combined with the environment variable *I_MPI_SPIN_COUNT*, it is possible to configure the spin count time for each MPI call. When the spin count becomes zero, the MPI library leaves the execution of the idle task of the CPU. This parameter does not contain a real-time value but includes a value that is decremented by the spinning procedure on the MPI library until it reaches zero. This allows the Intel MPI library to spin on a synchronization point for a while, and after that, enter in an idle low power state to reduce the power consumption of the core. The execution is restored when a system interrupt wakes up the MPI library signaling the end of the MPI call. Later, we will refer to this mode as *MPI SPIN WAIT*.

In the next Section, we will clarify though experiment on why the timeout logic introduced by COUNTDOWN is effective in making fine-grain power management possible and convenient in MPI parallel applications.

4 Experimental Results

In this Section, we present: (i) an overhead analysis of COUNTDOWN, (ii) the effect of timeout strategy using different timeout delays, and (iii) the evaluation on a single node and a production HPC system with real scientific applications.

4.1 Framework Overheads

We evaluate the overhead of running MPI applications instrumented with the profiler module of COUNTDOWN without changing the cores' frequency. We run QE-CP-EU on a single node, which is the worst case for COUNTDOWN in terms of number and granularity of MPI calls to profile because all network-related overheads in MPI calls are nullified and intra-chip communication and synchronization are orders-of-magnitude faster than the inter-chip or inter-node ones. Hence, MPI wait-times exploitable for power management are generally much shorter.

In this run, there are more than 1.1 million MPI primitives for each process in the diagonalization task: our runtime library needs to profile on average an MPI call every 200us for each process. We measured the overhead comparing the execution time with and without COUNTDOWN instrumentation. We repeated the test five times, and we report the median case. Our results show that even in this unfavorable setting, the COUNTDOWN profiler introduces an overhead in the execution time which is less than 1%. We repeated the same test changing the cores' frequency to assess the overhead of a fine-grain DVFS control. To measure only the overhead caused by the interaction with the DVFS knobs, we force COUNTDOWN to force always the highest P-state in the DVFS control registers. Thus, we avoid application slowdowns caused by frequency variation, and

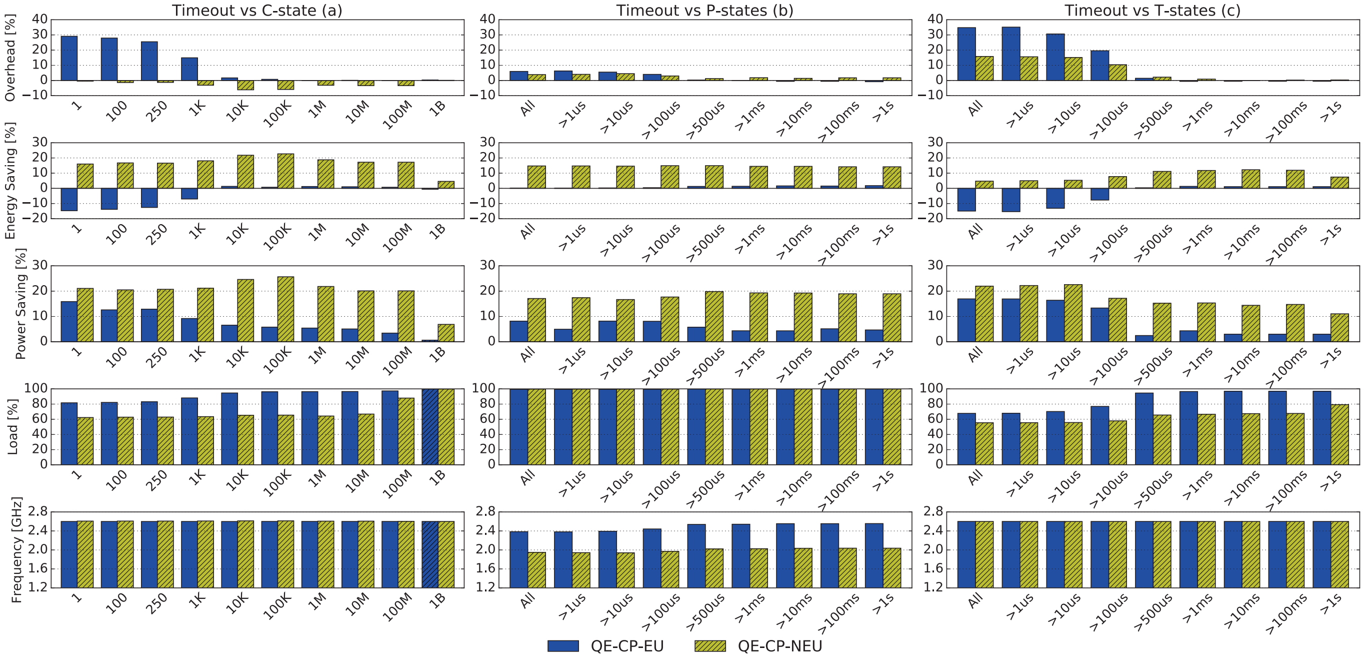


Fig. 5. Impact of MPI phases duration on the overheads, energy, and power savings of C/P/T-state for QE-EU and QE-NEU.

we obtained only the overhead caused by the register access. Our experimental results report of 1.04% of overhead to access the DVFS control register and for the profile routines.

These results prove that the source of the overheads of phase agnostic fine-grain power management is not related to issuing the low power state transition (DVFS in this case). Figure 5 focuses on understanding the source of this by replicating the tests of Section 2 for both QE-CP-EU and QE-CP-NEU, but now entering in the low power state only for MPI phases longer than a given time threshold. For the P-state and T-state (Figure 5.b and Figure 5.c) we obtained that by profiling in advance the duration of each MPI phase and instrumenting with the low power command only the phases which had a duration longer than the threshold. We report on the x-axes the time threshold value. For C-state (Figure 5.a) we leveraged the COUNTDOWN MPI logic, *I_MPI_SPIN_COUNT* parameter to filter out short phases. On the x-axis, we report the *I_MPI_SPIN_COUNT* parameter.

From the plot, we can recognize that there is a well-defined threshold of 500us for the T-state and P-state case and of 10K iteration steps for the C-state after which the overhead introduced by the fine-grain power management policy is reduced and the energy savings becomes positive for the QE-CP-EU. In the next Section, we will analyze why this happens by focusing on the P-state case.

The overhead in terms of memory is negligible since the memory required by COUNTDOWN is just a few megabytes for each MPI process.

4.2 DVFS Overheads and Time Region Analysis

To find the reason of the higher overhead when frequency reduction is applied in all the MPI phases as highlighted in the previous Section, we report two scatter plots in which we show on the x-axis of the left plot the time duration of each MPI phase and on the right plot the time duration of each application phase. For both plots, we report on the y-axes the measured average frequency in that phase. This test is conducted by instrumenting each MPI call through COUNTDOWN with a *prologue* routine to set the lowest frequency (1.2GHz) and with an *epilogue* routine to set the highest frequency (Turbo).

In theory, we would have expected that all MPI phases had executed at the minimum frequency and application phases had always run at maximum frequency. It is a matter of fact that MPI phases running at high frequencies may cause energy waste, while application phases running at low frequencies cause a performance penalty to the application. Our results show that for phases with a time duration between 0us and 500us, the average frequency varies in the interval between the high and low CPU's frequency values, while above it, it tends to the desired frequency for that phase. This can be explained by the response time of *HW power controller* in serving P-state transition of our Intel Haswell [10], we discover the same behavior on Intel Broadwell architecture. The *HW power controller* periodically reads the DVFS register to check if the OS has specified a new frequency, this interval has been reported to be 500us in a previous study [10] and matches our empirical threshold.

This means that every new setting for the core's frequency faster than 500us could be applied or completely ignored, depending on when the register was sampled the previous time. This can cause all sorts of average frequencies. Clearly, application phases that execute at a lower frequency than the maximum one may lead to a slowdown in the application, while MPI phases that execute at a higher frequency than the minimum one may lead to energy saving loss. It is nevertheless interesting to notice that phases with a duration from 0s to 500us are more likely to have the highest frequency for the MPI phases and the lowest frequency

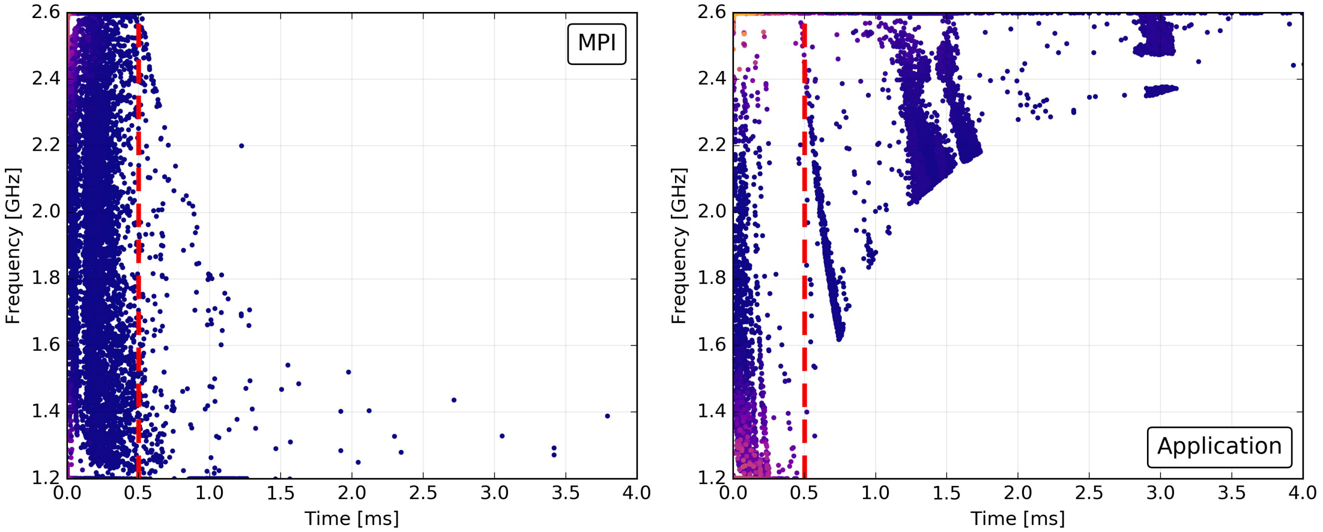


Fig. 6. Average frequency and time duration of Application/MPI phases for the single node benchmark of QE-CP-EU (16 MPI ranks).

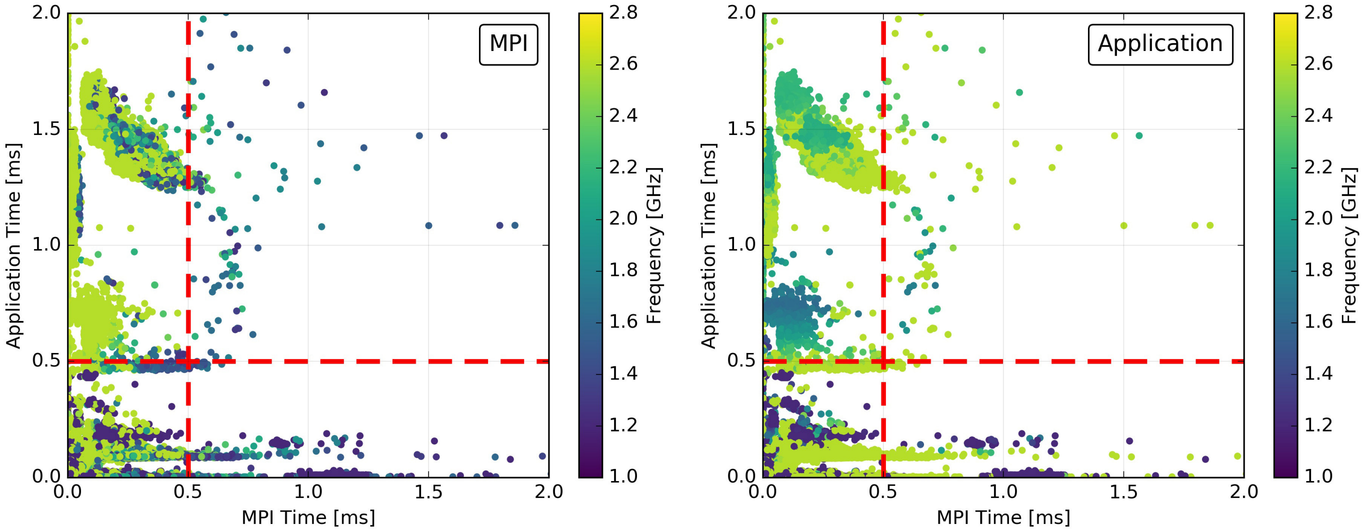


Fig. 7. Time and average frequency of Application/MPI phases for the single node benchmark for QE-CP-EU (16 MPI ranks).

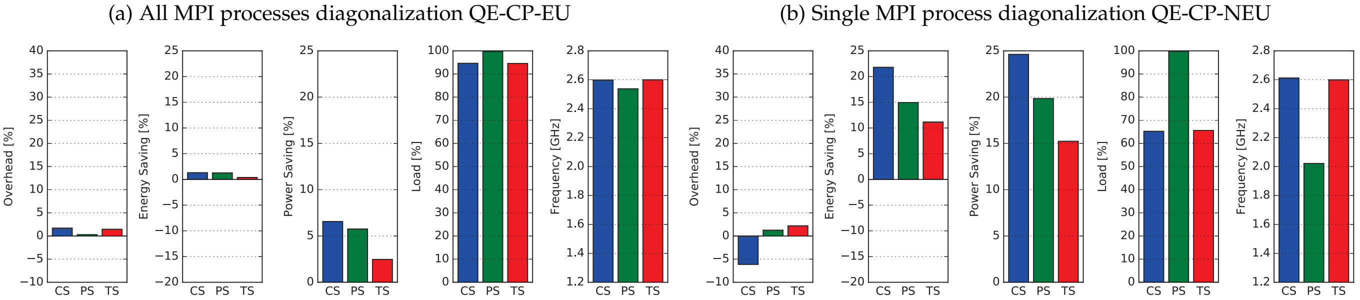
for the application phases, which is the opposite of what expected. We will explain it with the next analysis.

Thus, it is not possible to have effective control on the frequency selection for phases shorter than 500us, while for longer phases we have an asymptotic trend toward the requested frequency. We hypothesize that in phases shorter than 500us the average frequency depends more on the previous phase frequency than the requested one.

Following this intuition, in Figure 7, we correlate the time duration of each application phase with the time duration of the following MPI phase and its average frequency. We report in the y-axis the time duration of the application phase, in the x-axis the time duration of the subsequent MPI phase, and with the color code, we report the average frequency. In the left plot, we report the average frequency of the MPI phase, while in the right plot we report the average frequency for the application phase. For both plots, we can identify four regions/quadrants:

(i) **Application & MPI>500us**: this region contains long application phases followed by long MPI phases. Points in this region show low frequency in MPI phases and high frequency in application phases. This is the ideal behavior, where applying frequency scaling policy reduces energy waste in MPI but with no impact on the performance of the application. Phases in this region are perfect candidates for fine-grain DVFS policies.

(ii) **Application>500us & MPI<500us**: this region contains long application phases followed by short MPI phases. Points in this region show for both application and MPI phases high average frequency. This is explained by the short duration of the MPI phases, which does not give enough time to the *HW power controller* to serve the request to scale down the frequency (*prologue*) before this setting is overwritten by the request to operate at the highest frequency (*epilogue*). For this reason, fine-grain DVFS control in this region does not have an impact on the energy saving


(a) All MPI processes diagonalization QE-CP-EU

(b) Single MPI process diagonalization QE-CP-NEU

Fig. 8. Overhead, energy/power saving, average load, and frequency using COUNTDOWN for QE-CP-EU (a) and QE-CP-NEU (b). Legend: C-state (CS), P-state (PS) and T-state (TS) mode. The baseline is the busy-waiting mode of the MPI library.

as the frequency reduction in MPI phases is negligible, but it also does not deteriorate the performance as the application phases are executed at the maximum frequency. Phases in this region should not be considered for fine-grain DVFS policies, being preferable to leave frequencies unaltered at the highest level.

(iii) **Application<500us & MPI>500us**: this region contains short application phases followed with long MPI phases. This is the opposite case of *Application>500us & MPI<500us* region. Points in this region show for both application and MPI phases low average frequency. This is explained by the short duration of the application phases, which does not give enough time to the *HW power controller* to serve the request to raise up the frequency (requested at the exit of the previous MPI phase), before this setting gets overwritten by the request to operate at the lowest frequency (at the entrance of the following MPI phase). Applying fine-grain DVFS policies in this region can save power, but detriments the overall performance, as application phases are executed at low frequencies. Phases in this region should not be considered for fine-grain DVFS policies due to the high overheads in the application execution time.

(iv) **Application & MPI<500us**: This region shows the opposite behavior of *Application & MPI>500us* region. Both application and MPI phases execute randomly at high and low average frequencies due to the inability of the *HW power controller* to capture and service the requested frequency changes. The average frequency at which MPI and application phases execute are strictly related to the type of the previous long phase: if it was an application phase the following short phases will execute at high frequency in average; On the contrary, if it was an MPI phase the following short phases would execute at low frequency in average. Applying fine-grain DVFS policies in this region leads to unexpected behaviors that can detriment application performance. Fine-grain power managers should never consider all phases shorter than 500us.

4.3 Single-node Evaluation

We repeated the experiments of Section 2 using COUNTDOWN. We configure COUNTDOWN to scale down the P- and the T-states 500us after the prologues of MPI primitives.

To reproduce the same timeout strategy leveraging the C-states, we configure *MPI SPIN WAIT* as described in 3.2 with 10K as MPI spin counter parameter.

The *HW power controller* of Intel CPUs, has a different transition latency for sleep states w.r.t. DVFS scaling, as described in [10]. For this reason, we empirically determine the best spin counter setting to maximize energy efficiency and to minimize the overhead for the target application.

Figure 8 report the experimental results using *COUNTDOWN THROTTLING*, *COUNTDOWN DVFS* and *MPI SPIN WAIT*. We can see that in all cases the overhead, the energy saving, and the power saving are significantly improved w.r.t. the baseline (only MPI library).

Figure 8.a shows the experimental results for QE-CP-EU. For the C-state mode the overhead decrease from 25.85% to 1.70% by using *MPI SPIN WAIT*. Instead, for the P-state using *COUNTDOWN DVFS* the overhead decreases from 5.96% to a negligible overhead, and for the T-state using *COUNTDOWN THROTTLING* the overhead decreases from 5.96% to 0.29%. All evaluations report a non-negative energy saving, as it was for the MPI library without timeout strategy, but with better results. Energy saving shows 21.80%, 14.94%, and 11.16% improvements and power saving report 6.55%, 5.77%, and 2.47% respectively for C-state, P-state, and T-state. These experimental results confirm our exploration of the time duration of MPI phases reported in figure 6. Most of the MPI calls of this benchmark have been skipped due to their short duration to avoid overheads.

Figure 8.b show similar improvements for QE-CP-NEU. In this configuration, for C-State mode the speed-up increases from 1.08% to 6.14% using *MPI SPIN WAIT*. Instead of using COUNTDOWN, the overhead of P-state decreases from 3.88% to 1.25%, and for the T-state from 15.82% to 2.19%. As a result, the energy saving is 21.80%, 14.94%, and 11.16% while power saving corresponds to 24.61%, 19.84%, and 15.23% respectively for C-state, P-state, and T-state.

4.4 HPC Evaluation

After we have evaluated our methodology in a single compute node, we extend our exploration in a real HPC system. We use a Tier-1 HPC system based on an IBM NeXtScale cluster which is currently classified in the Top500 supercomputer list [18]. The compute nodes of the HPC system, are equipped with 2 Intel Broadwell E5-2697 v4 CPUs, with 18 cores at 2.3 GHz nominal clock speed and 145W TDP and interconnected with an Intel QDR (40Gb/s) Infiniband high-performance network.

To benchmark the parallel performances in our target HPC system we focused on three sets of applications. The

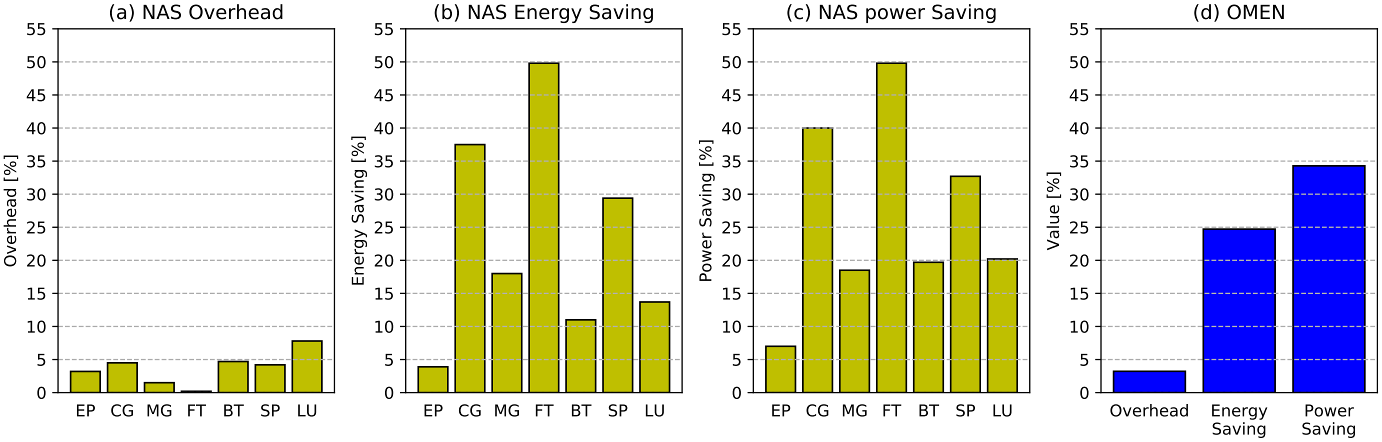


Fig. 9. Results on NAS parallel benchmarks suite and OMEN with COUNTDOWN on 1024 MPI rank/cores. The baseline is the busy-waiting mode.

first one is the NAS parallel benchmark suite [11] with the dataset E. We executed the NAS parallel benchmarks on 29 compute nodes with a total core count of 1024 cores. We use 1024 cores due the execution time of the application run using dataset E is on average ten minutes for each benchmark. The NAS parallel benchmarks are composed of 7 benchmarks which implement different mathematical workloads that are very representative of HPC scientific kernels, such as FFT, differential equations, ordering, etc. We select these benchmarks because they create a wide range of workloads. The second benchmark is OMEN, an atomistic quantum transport simulator that can compute the I-V characteristics of nano-devices ab-initio (from first principles) [19]. The code has been optimized to run on the largest supercomputers, winning the ACM Gordon Bell Prize in 2019 [19] and reaching two times the final in 2011 and 2015. In our runs, a transistor with a 2-D crystal as channel material serves as a benchmark. The last benchmark is QuantumESPRESSO PWscf configured for complex large-scale simulation. For this purpose, we performed ten iterative steps of the self-consistent loop algorithm that optimizes the electronic density starting from the superposition of atomic charge densities. To obtain a reasonable scaling up to the largest set of nodes, we chose an ad-hoc dataset.

During each iteration, the CPU time is mostly spent in linear algebra (matrix-matrix multiplication and matrix diagonalization) and FFT. Both these operations are distributed on multiple processors and operate on distributed data. As a consequence, FFT requires many AllToAll MPI communications while parallel diagonalization, performed with the PDSYEVD subroutine of SCALAPACK and requires mostly MPI broadcasting messages. We run QE on 96 compute nodes, using 3456 cores and 12 TB of DRAM due our target HPC machine allows application runs with at maximum 100 nodes. We use an input dataset capable of scaling on a such number of cores, and we configure QE using a set of parameters optimized to avoid network bottlenecks, which would limit the scalability. We name this configuration QuantumESPRESSO Expert User (QE-PWscf-EU), to differentiate it from the same problem but solved without optimizing the internal parameter as it was run by a user without domain-specific knowledge which we call QuantumESPRESSO Not Expert User (QE-PWscf-NEU).

In these tests, we exclude the T-state mode, because, in the single-node evaluation, it always reported the worst results that the P-state mode. We also excluded the C-state mode as when we started the configuration of the Intel MPI library for HPC experiments using idle mode. We discover that this feature is not supported in a distributed environment. The Intel MPI library overrides the request of idle mode with the busy-wait mode when the application runs on multiple nodes. For this reason, we only use the P-state mode (*COUNTDOWN DVFS*) in the HPC evaluation.

We run the benchmark with and without COUNTDOWN on the same nodes, and we compared the results.

Figure 9 shows the results for the NAS parallel benchmark suite and OMEN when executed on 1024 cores, while figure 10 shows the results for the QE-PWscf-* application when executed on 3456 cores. The different plots for Figure 9 reports the time-to-solution overhead, the energy, and power saving for the different large-scale benchmarks and application run. All the values are normalized against the default MPI busy waiting policy. From Figure 9.c, we can see that COUNTDOWN is capable of significantly cutting the energy consumption of the NAS benchmarks from 6% to 50%. From the overhead plot (Figure 9.a) we can see that all these energy savings happen with a very small time-to-solution overhead, on average below 5%. The results for OMEN, in Figure 9.d, confirms the one obtained for the NAS benchmarks with an overhead of 3.22% and an energy and power saving of 24.72% and 34.28% respectively. These results are very promising as they are virtually portable to any application, without the need to touch the application binary. When looking at the QuantumEspresso (QE-PWscf-*) case reported in figure 10, we see that COUNTDOWN attains similar results of NAS also with real production run optimized for scalability COUNTDOWN saves 22.36% of energy with an overhead of 2.88% in the QE-PWscf-EU case.

Figure 10.a shows the total time spent in the application and in MPI phases which are shorter and longer than 500us for the QE-PWscf-EU case. On the x-axis, the figure reports the Id of the MPI rank, while in the y-axis reports in the percentage of the total time spent in phases longer and shorter than 500us. We can immediately see that in this real and optimized run, the application spends a negligible time in phases shorter than 500us. In addition, the time spent

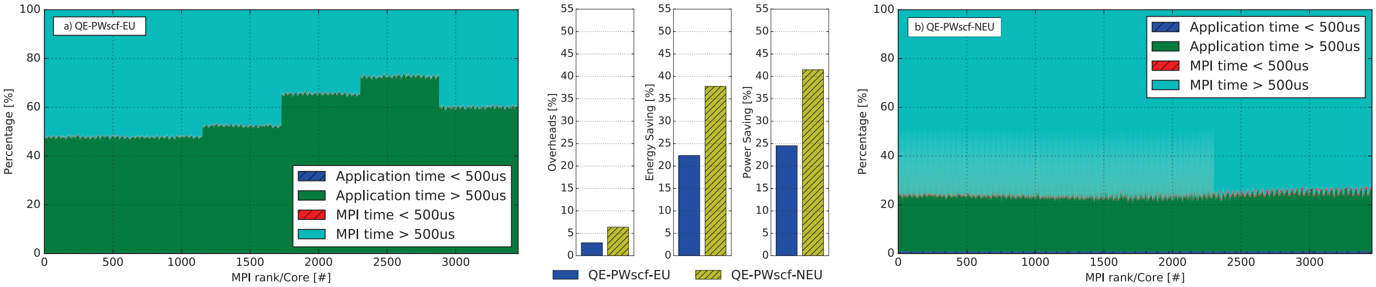


Fig. 10. (a,b) Sum of the time spent in phases longer and shorter than 500us for QE-PWscf-EU and QE-PWscf-NEU for 3,456 MPI rank/cores.

in the MPI library and the application is not homogeneous among the MPI processes. This is an effect of the workload parameters chosen to optimize the communications, which distribute the workload in subsets of MPI processes to minimize broadcast and All-to-All communications. Using this configuration, our experimental results report 2.88% of overhead with an energy saving of 22.36% and a power saving of 24.53% thanks to COUNTDOWN.

Figure 10.c shows that for the case QE-PWscf-NEU where the parameters are not optimized, all MPI processes have the same workload composition as they are part of the same workgroup and due the large overhead in the broadcast and All-to-All communications. Most of the processes spend almost 80% of the time in the MPI library. Even if it is suboptimal, this happens to HPC users running the application without being domain experts or before tuning the execution parameters. This is a rather typical scenario in scientific computing as only runs that are repeated multiple times are carefully optimized by domain experts.

In this situation, COUNTDOWN increases its benefits, reaching up to 37.74% of energy saving and a power saving of 41.47%. In this condition, we also notice that COUNTDOWN induces a small but relevant overhead of 6.38%. We suspect that some MPI primitives suffer more than others from the frequency scaling. We will analyze in depth this problem in our future works aiming to guarantee that the COUNTDOWN overhead always remains negligible. However, we remark that an overhead well below 10% is more than acceptable in many HPC facilities, especially when considering the massive energy savings.

In summary, we can conclude that results achieved by COUNTDOWN in production scale and application are very promising and if systematically adopted would dramatically reduce the TCO of today supercomputers.

Future works will analyze the source of the small overhead introduced by the COUNTDOWN algorithm aiming to zero it by challenging the assumption that all the MPI phases are composed only by slack time not dependent on the frequency at which they are executed.

5 Related Work

Several works focused on mechanisms and strategies to maximize energy savings at the expense of performance. These works focus on operating the processors at a reduced frequency for the entire duration of the application [2], [3], [4]. The main drawback of these approaches is the negative impact on the application performance which is detrimental to the data center cost efficiency and TCO.

Fraternali et al. [2], [8] analyzed the impact on frequency selection on a green HPC machine which can lead a significant global energy reduction in real-life applications but can also induce significant performance penalties. Auweter et al. [3] developed an energy-aware scheduler that relies on a predictive model to predict the wall-time and the power consumption at different frequency levels for each running applications in the system. The scheduler uses this information to select the frequency to apply to all the nodes executing the job to minimize the energy-to-solution allowing unbounded slowdown in the TTS.

The main drawback of the works mentioned so far is that they lead to a systematic increase of TTS, which may be acceptable for the user, but it is not easily acceptable by the facility manager since it reduces the data center cost efficiency and TCO [20]. For this reason, there is a trend in the literature towards HPC energy reduction methodologies with negligible or low impact on TTS of the applications.

Sundriyal et al. [21], [22], [23], [24] analyze the impact of fine-grain power management strategies in MVAPICH2 communication primitives, with a focus on send/receive [21], All-to-All [22], and AllGather communications [23]. In [21] the authors propose an algorithm to lower the P-state of the processor during send and receive primitives. The algorithm dynamically learns the best operating points for the different send and receive calls. In the [22], [23], [24] works, the authors propose to lower also the T-state during the send-receive, all-gather, and all-to-all primitives as this increases the power savings. These approaches show that power saving can be achieved by entering in a low power mode during specific communication primitives but they depend on a specific MPI implementation. Differently, we show that significant savings can be achieved without impacting the implementation of the MPI library. The work also proposed complex mathematical models on several MPI primitives taking into account the overheads of low power modes to predict the expected time of communication. This is in contrast with the philosophy of COUNTDOWN, which leverages on a reactive mechanism making it robust to miss-predictions on communication time of learning approaches [25].

Rountree et al. [26] analyze the energy savings which can be achieved on MPI parallel applications by slowing down the frequencies of processors that are not in the critical path. Authors of the paper define tasks as the region of code between two MPI communication calls, we will refer later in this paper to tasks as phases. The critical path is defined as the chain of the tasks which bounds the application execution time. Indeed, cores executing tasks in the critical path will be the latest ones to reach the MPI synchronization points, forcing the other cores to wait. In [26] authors propose a methodology for estimating offline the minimum frequency at which the waiting cores can execute without affecting the critical path and the TTS. In the same work, the authors suggest that the core's frequency cannot be changed too often without causing overheads. For this reason, the authors introduce a timer logic set at 10ms to avoid changing the core's frequency too often. This value is empirically found. With COUNTDOWN we demonstrate that in modern CPUs the best setting for this timer value corresponds to the built-in HW power controller latency.

A later work of the same authors [6], implements an online algorithm to identify the task and the minimum frequency at which it can be executed without worsening the critical path. This is done with a slack reclamation policy which is based on the measurement of the blocking time of the previous MPI primitive. If this was at least twice longer than an empirical time threshold (100ms) a timer is set to the empirical threshold when the MPI primitive is encounter again. If the timer expires, the core's frequency is set to the minimum available. This, in essence, implements a last-value prediction logic to determine if there will be enough blocking time which could be exploited to save energy. COUNTDOWN uses a timeout policy as well, but it applies it for each MPI phase without trying to predict its duration. This is a significant difference w.r.t to the [6] which makes it robust to miss-predictions [15]. Similarly, Kappiah et al. [9] developed Jitter, an online run-time library based on the identification of the critical path on the application among compute nodes involved in the application run. Liu et al. [27] use a similar methodology as Kappiah et al. [9] but they apply it to a multi-core CPU. Zhai et al. [16] propose a method for estimating the duration of an MPI application.

The authors of [28], as in [6], [26], focus on saving power by entering a low power state for processes which are not in the critical path. The authors propose an algorithm to save energy by reducing application unbalance. This is based on measuring the start and end time of each MPI_Barrier and MPI_Allreduce primitives to compute the duration of application and MPI code. Based on that the authors propose a feedback loop to lower the P-state and T-state if in previous compute and MPI region the overhead was below a given threshold. The algorithm is based on the assumption that the duration of the current application and MPI phases will be the same as the previous ones. In COUNTDOWN we target recent HW and larger production runs where we do not use any previous information on MPI and application phase duration, which may lead to costly performance overhead in case of misprediction in particular in irregular applications [25]. Instead, COUNTDOWN relies only on a pure-reactive timer-based logic. It is worth to notice that differently from [6], the COUNTDOWN logic does not use any pre-characterization of the message-transfer time of the MPI library to estimate the communication blocking time due to this can change depending to the network congestion of the high-performance interconnect.

To save energy during MPI phases, Lim et at. [29] propose to reduce core's frequency in "long" MPI phases. Subsequent short MPI phases are grouped and treated as a single long MPI phase. They use an algorithm to select the best P-state to be applied according to the micro-operation throughput in the MPI phase. Similarly to [6], [26], this approach is based on the assumption that the duration and instruction composition of current MPI phase will be the same as the previous ones. Moreover, by treating short MPI phases as a single long one, the application phases between them are executed at low frequency leading overheads.

Li et al. [30] use a similar approach to [29] to reduce power consumption in synchronization points. This work focuses on collective barriers for parallel applications in shared-memory multiprocessors. Differently, from the previous approaches, instead of using P-state, they use idle states (C-states) and specific hardware extensions to account for their transitioning (sleep and wake-up) times. As in the previously described approaches, this run-time library uses a history-based prediction model to identify the duration of the next barriers.

The authors of [31] show that the approaches in [6], [26] and the ones which estimate the duration of MPI and communication phases based on a last-value prediction [29], [30] can lead to significant misprediction errors. The authors propose to solve this issue by estimating the duration of the MPI phases with a combination of communication models and empirical observation specialized for the different groups of communication primitives. The authors of [31] focus on C-states as P-states are said to be not compatible with production software. Assumptions are then made to compute a time threshold for a C-state transition time based on an accepted application overhead. If the predicted execution time for the MPI phase is longer than this threshold, the core is transitioned into the deeper C-state. In COUNTDOWN, we focus on P-states for which we show that the transition time is negligible, but the short MPI phase should be filtered out as an effect of the polling time of the power controller logic. Besides, we do not rely on predicted execution time which can lead to miss-predictions. COUNTDOWN proves that implementing a DVFS strategy leveraging on PMPI interface and MSR-SAFE driver [17] is an efficient and safe solution.

Li et al. [7] analyzed hybrid MPI/OpenMP applications in terms of performance and energy saving and developed a power-aware run-time library that relies on dynamic concurrency throttling (DCT) and DVFS mechanisms. This runtime library uses a combination of a power model and a timing predictor for OpenMP phases to select the best cores' frequency when application manifests workload imbalance.

The works in the second group, namely [7], [9], [27], [29], [30], but also [6] in the slack reclamation policy, have in common the prediction of future workload imbalances or MPI phases obtained by analyzing previous communication patterns. However, this approach can lead to frequent mispredictions in irregular applications [25] which cause performance penalties. COUNTDOWN differs from the above approaches (and complements them) because it is purely reactive and does not rely on assumptions and estimation of the future workload unbalance.

The power management literature has analyzed in depth the issue of prediction inaccuracy and predictive model overfitting [15]. One of the key outcomes of COUNTDOWN is that timeout-based policies are effective if predictions are not available (e.g. when data is being collected for building a predictive model), and are also essential in mitigating miss-prediction overheads.

Eastep et al. propose GEOPM [5], an extensible and plug-in based framework for power management in large parallel systems. GEOPM is an open-source project and exposes a set of APIs that programmers can insert into applications to combine power management strategies and HPC workload. A plugin of the framework targets power constraint systems aiming to speed up the critical path migrating power to the CPU's executing the critical path tasks. In a similar manner, another plugin can selectively reduce the frequency of the processors in specific regions of codes flagged by the user by differentiating regions in CPU, memory, IO, or disk bound. Today, GEOPM is capable of identifying MPI regions and reducing the frequency based on MPI primitive type. However, it cannot differentiate between short and long MPI and thus cannot control the overhead caused by the frequency changes in short MPI primitives. COUNTDOWN addresses this limitation and can be integrated into future releases of GEOPM, as its design principles are entirely compatible with it (i.e. no application code modifications are required).

An earlier version of the COUNTDOWN run-time library was presented in [32]. This paper adds in COUNTDOWN the support for two additional low power state mechanisms (C-state and T-states) and their comparisons with P-state. Moreover, we extended [32] with a detailed analysis of the timeout configuration for the three different low power state mechanisms, and the implication of the timeout with the MPI and application phases duration. We finally extended [32] with a broader set of experimental results, including the NAS parallel benchmarks and an additional QE large-scale run with different network optimization, which is a common use-case in supercomputer environment.

6 Conclusion

In this paper, we presented COUNTDOWN, a methodology, and a tool for profiling HPC scientific applications and for adding DVFS capabilities into standard MPI libraries. COUNTDOWN implements a timeout strategy to avoid application slowdown and exploiting MPI communication slacks to reduce energy consumption drastically. COUNTDOWN has been demonstrated on real HPC systems and workloads and does not require any modification to application source code nor the compilation toolchain. The COUNTDOWN approach can leverage several low power state technologies — P/T/C states.

We compared COUNTDOWN with state-of-the-art power management approaches for MPI libraries, which can dynamically control idle and DVFS levels for MPI-based applications. Our experimental results show that using our tuned timeout strategy to make decisions on power control can drastically reduce overheads, maximizing the energy efficiency in small and large MPI communications. Our run-time library can lead up to 14.94% energy saving, and 19.84% of power saving with a less than 1.5% performance penalty on a single compute node. However, the benefits of COUNTDOWN increase with the scale of the application. In a 1K cores NAS run, COUNTDOWN always saves energy, with a saving which depends on the application and ranges from 6% to 50% at a negligible overhead (below 6%). In a full-scale production run of QE on more than 3.4K cores, COUNTDOWN saves 22.36% of energy with only 2.88% performance overhead. Energy reduction reaches 37.74% when the application is executed with a default conservative parallelization setting.

COUNTDOWN is an effective, non-intrusive, and low overhead approach to cut today's supercomputing center energy-consumption transparently to the user. In future work, we plan to integrate it within standard power management infrastructure, such as GEOPM [5], and to complement it with predictive and application-driven power management techniques.

Acknowledgments

Work supported by the EU FETHPC project ANTAREX (g.a. 671623) and EU H2020-INFRAEDI-2018-1 MaX ”Materials Design at the Exascale” (g.a. 824143).

References

[1] “Advanced Configuration and Power Interface (ACPI) Specification,” [Online]; http://www.acpi.info/spec.htm, 2019, accessed 29 March 2019.

[2] F. Fraternali, A. Bartolini, C. Cavazzoni, G. Tecchiolli, and L. Benini, “Quantifying the impact of variability on the energy efficiency for a next-generation ultra-green supercomputer,” in *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, ser. ISLPED '14. New York, NY, USA: ACM, 2014, pp. 295–298.

[3] A. Auweter, A. Bode, M. Brehm, L. Brochard, N. Hammer, H. Huber, R. Panda, F. Thomas, and T. Wilde, “A case study of energy aware scheduling on supermuc,” in *International Supercomputing conference*. Springer, 2014, pp. 394–409.

[4] C. Hsu and W. Feng, “A power-aware run-time system for high-performance computing,” in *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, Nov 2005, pp. 1–1.

[5] J. Eastep, S. Sylvester, C. Cantalupo, B. Geltz, F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, and S. Jana, “Global extensible open power manager: A vehicle for hpc community collaboration on co-designed energy management solutions,” in *High Performance Computing*. Springer International Publishing, 2017, pp. 394–412.

[6] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, “Adagio: Making dvs practical for complex hpc applications,” in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09. New York, NY, USA: ACM, 2009, pp. 460–469.

[7] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos, “Hybrid mpi/openmp power-aware computing,” in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, April 2010, pp. 1–12.

[8] F. Fraternali, A. Bartolini, C. Cavazzoni, and L. Benini, “Quantifying the impact of variability and heterogeneity on the energy efficiency for a next-generation ultra-green supercomputer,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 7, pp. 1575–1588, July 2018.

[9] N. Kappiah, V. W. Freeh, and D. K. Lowenthal, “Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs,” in *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, Nov 2005, pp. 33–33.

[10] D. Hackenberg, R. Schne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, “An energy efficiency feature survey of the intel haswell processor,” in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, May 2015, pp. 896–904.

[11] D. H. Bailey, *NAS Parallel Benchmarks*. Boston, MA: Springer US, 2011, pp. 1254–1259.

[12] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo *et al.*, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *Journal of physics: Condensed matter*, vol. 21, no. 39, p. 395502, 2009.

[13] G. Avvisati, S. Lisi, P. Gargiani, A. Della Pia, O. De Luca, D. Pacil, C. Cardoso, D. Varsano, D. Prezzi, A. Ferretti, and M. G. Betti, "Fepc adsorption on the moiré superstructure of graphene intercalated with a cobalt layer," *The Journal of Physical Chemistry C*, vol. 121, no. 3, pp. 1639–1647, 2017.

[14] S. Bhalachandra, A. Porterfield, and J. F. Prins, "Using dynamic duty cycle modulation to improve energy efficiency in high performance computing," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, May 2015, pp. 911–918.

[15] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299–316, June 2000.

[16] J. Zhai, W. Chen, W. Zheng, and K. Li, "Performance prediction for large-scale parallel applications using representative replay," *IEEE Transactions on Computers*, vol. 65, no. 7, pp. 2184–2198, July 2016.

[17] K. Hoga and B. Rountree, "Github scalability-llnl/msr-safe, 2014," [Online]; https://github.com/LLNL/msr-safe, 2019, accessed 29 March 2019.

[18] J. J. Dongarra, H. W. Meuer, E. Strohmaier *et al.*, "Top500 supercomputer sites," [Online]; https://www.top500.org/lists, 2019, accessed 29 March 2019.

[19] A. N. Ziogas, T. Ben-Nun, G. I. Fernández, T. Schneider, M. Luisier, and T. Hoefler, "A data-centric approach to extreme-scale ab initio dissipative quantum transport simulations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC 19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3357156

[20] A. Borghesi, A. Bartolini, M. Milano, and L. Benini, "Pricing schemes for energy-efficient hpc systems: Design and exploration," *The International Journal of High Performance Computing Applications*, vol. 32, pp. 1–19, 2018.

[21] V. Sundriyal, M. Sosonkina, and A. Gaenko, "Energy efficient communications in quantum chemistry applications," *Computer Science - Research and Development*, vol. 29, no. 2, pp. 149–158, May 2014.

[22] V. Sundriyal and M. Sosonkina, "Per-call energy saving strategies in all-to-all communications," in *European MPI Users' Group Meeting*. Springer, 2011, pp. 188–197.

[23] V. Sundriyal, M. Sosonkina, and Z. Zhang, "Achieving energy efficiency during collective communications," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 15, pp. 2140–2156, 2013.

[24] V. Sundriyal, M. Sosonkina, and A. Gaenko, "Runtime procedure for energy savings in applications with point-to-point communications," in *2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, Oct 2012, pp. 155–162.

[25] D. J. Kerbyson, A. Vishnu, and K. J. Barker, "Energy templates: Exploiting application information to save energy," in *2011 IEEE International Conference on Cluster Computing*. IEEE, 2011, pp. 225–233.

[26] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale mpi programs," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, ser. SC '07. New York, NY, USA: ACM, 2007, pp. 49:1–49:9.

[27] C. Liu, A. Sivasubramaniam, M. Kandemir, and M. J. Irwin, "Exploiting barriers to optimize power consumption of CMPs," in *19th IEEE International Parallel and Distributed Processing Symposium*, April 2005, p. 10.

[28] S. Bhalachandra, A. Porterfield, S. L. Olivier, and J. F. Prins, "An adaptive core-specific runtime for energy efficiency," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2017, pp. 947–956.

[29] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs," in *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, Nov 2006, pp. 14–14.

[30] J. Li, J. F. Martinez, and M. C. Huang, "The thrifty barrier: energy-aware synchronization in shared-memory multiprocessors," in *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, Feb 2004, pp. 14–23.

[31] A. Venkatesh, A. Vishnu, K. Hamidouche, N. Tallent, D. Panda, D. Kerbyson, and A. Hoisie, "A case for application-oblivious energy-efficient MPI runtime," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2015, pp. 1–12.

[32] D. Cesarini, A. Bartolini, and L. Benini, "Countdown - a run-time library for application agnostic energy saving in mpi communication primitives," in *Proceedings of the 2nd Workshop on AutotuniNg and aDaptivity AppRoaches for Energy Efficient HPC Systems*, ser. ANDARE '18, 2018, pp. 3:1–3:6.

Daniele Cesarini graduated in Computer Engineering from the University of Bologna (Italy) in 2014, where he also earned his Ph.D. in Electronics, Telecommunications, and Information Technologies Engineering in 2019. He is now an HPC Analyst at Cineca High Performance Computing department where he works in the area of performance optimization and evaluation of next-generation HPC architectures. His research interests concern the development of SW-HW co-design strategies as well as algorithms for parallel programming support for energy efficient HPC systems.

Andrea Bartolini received a Ph.D. degree in Electrical Engineering from the University of Bologna, Italy, in 2011. He is currently Assistant Professor in the Department of Electrical, Electronic and Information Engineering (DEI) at the University of Bologna. Before, he was Post-Doctoral researcher in the Integrated Systems Laboratory at ETH Zurich. Since 2007 Dr. Bartolini has published more than 80 papers in peer-reviewed international journals and conferences with focus on dynamic resource management for embedded and HPC systems.

Pietro Bonfà received his B.Sc. degree in Physical Engineering (2008) in Politecnico di Milano, the M.Sc. degree in Physics (2011) from University of Pavia and the Ph.D. in Physics (2015) from University of Parma. He is now a Research Associate at the Department of Mathematical, Physical and Computer Sciences of the University of Parma. His research activities concern solid-state physics and focus on experimental and computational spectroscopy methods for the characterization of the magnetic properties of materials.

Carlo Cavazzoni Carlo graduated in Physics from the University of Modena and earned his PhD Material Science at the International School for Advanced Studies of Trieste in 1998. He has authored or co-authored several papers published in prestigious international review including Science, Physical Review Letters, Nature Materials. He spend many years in CINECA HPC department as responsible for the R&D. Presently he is the head of computational R&D of Leonardo company and director of Leonardo HPC corporate Laboratory.

Luca Benini is professor of Digital Circuits and Systems at ETH Zurich, Switzerland, and is also professor at University of Bologna, Italy. His research interests are in system design of energy-efficient multicore SoC, smart sensors and sensor networks. He has published more than 800 papers in peer reviewed international journals and conferences, four books and several book chapters. He is a fellow of the ACM and Member of the Academia Europea. He is the recipient of the IEEE CAS Mac Van Valkenburg Award 2016.