COUNTDOWN - A Run-time Library for Application-agnostic Energy Saving in MPI Communication Primitives

Author(s):
Cesarini, Daniele; Bartolini, Andrea; Bonfà, Piero; Cavazzoni, Carlo; Benini, Luca

Publication date:
2018-11

Permanent link:
https://doi.org/10.3929/ethz-b-000313834

Originally published in:
https://doi.org/10.1145/3295816.3295818

COUNTDOWN - A Run-time Library for Application-agnostic Energy Saving in MPI Communication Primitives

Daniele Cesarini
DEI, University of Bologna
Bologna, Italy
daniele.cesarini@unibo.it

Andrea Bartolini
DEI, University of Bologna
Bologna, Italy
a.bartolini@unibo.it

Piero Bonfà
SCAI, CINECA
Casalecchio di Reno (BO), Italy
p.bonfa@cineca.it

Carlo Cavazzoni
SCAI, CINECA
Casalecchio di Reno (BO), Italy
c.cavazzoni@cineca.it

Luca Benini
IIS, Swiss Federal Institute of Technology
Zürich, Switzerland
lbenini@iis.ee.ethz.ch

ABSTRACT

Energy and power consumption are prominent issues in today's supercomputers and are foreseen as a limiting factor of future installations. In scientific computing, a significant amount of power is spent in the communication and synchronization-related idle times among distributed processes participating to the same application. However, due to the time scale at which communication happens, taking advantage of low-power states to reduce power in idle times in the computing resources, may introduce significant overheads.

In this paper we present COUNTDOWN, a methodology and a tool for identifying and automatically reducing the frequency of the computing elements in order to save energy during communication and synchronization primitives. COUNTDOWN is able to filter out phases which would detriment the time to solution of the application transparently to the user, without touching the application code nor requiring recompilation of the application. We tested our methodology in a production Tier-0 system, a production application - Quantum ESPRESSO (QE) - with production datasets which can scale up to 3.5K cores. Experimental results show that our methodology saves 22.36% of energy consumption with a performance penalty of 2.88% in real production MPI-based application.

KEYWORDS

HPC, MPI, profiling, power management, idleness, DVFS, DDCM, C-states, P-states, T-states, hardware performance counters, timer, energy saving, power saving

1 INTRODUCTION

While Moore's law is approaching its end, Dennard's scaling has already run out of steam. This has caused a constant increase of the power density required to operate each new processor generation at its maximum performance: causing de facto, the total power consumption of each device to limit the practical achievable performance. In addition of the detrimental effect of power density on the final performance, total power consumption needs to be delivered and removed through cooling consuming additional power. All these three issues impact the total costs of ownership (TCOs) and operational costs, which limits the budget for the supercomputer capacity. As a matter of fact, thermal limit and power wall are the key challenges to be faced if we wish to deliver the planned performance growth in future.

Computing elements are built with low power design principles and they allow to trade off performance vs. power consumption by mean of Dynamic and Voltage Frequency Scaling (DVFS) (also known as performance states or P-states [9]) and low power states which switch off unused resources (C-states [9]). Operating systems can change P-states and C-states at execution time adapting the performance of the current workload to reduce the power consumption. Transition time and execution time dependency can impact the application execution time leading or not to an energy saving.

A typical HPC application is composed by several processes executed in a cluster of nodes which exchange messages through a low-latency high-bandwidth network. These processes can access the network sub-system through a software interface that abstract the network level. The Message-Passing Interface (MPI) is a simple but high-performance standard interface for communication that allows these application processes to exchange explicit messages. Usually in large application runs, the time spent by the application in the MPI library is not negligible and impacts the power consumption of the system. By default, MPI libraries use a busy-waiting mechanism when MPI processes are waiting in a synchronization primitive. However, running an application in a low power mode during MPI primitives, may result in lower CPU power consumption with limited impact on the execution time due the wait time and IO/memory intensity of MPI primitives. MPI libraries implements idle-waiting mechanisms, but these are not used in practice to avoid performance penalties caused by the low power states transition time [8].

In this paper, we preset COUNTDOWN, a methodology and a tool to leverage the communication slack to save energy in scientific applications. It consists of a system runtime able to automatically inspect at fine granularity MPI and application phases and to inject power management policies opportunistically during MPI calls.

The paper focuses on understanding the implications of fine-grain power management in today's supercomputing systems targeting MPI library and providing a methodology for selecting at execution time when to enter in a low-power state to limit the transition time overheads. Indeed, COUNTDOWN is able to identify MPI calls with energy-saving potential for which it is worth to enter in a low power state, leaving fast MPI calls unmodified to prevent overheads in low-power state transitions. COUNTDOWN works at execution time without requiring any previous knowledge of the application, it is completely plug-and-play ready, this means that it does not require any modification of the source code and compilation toolchain. COUNTDOWN can be dynamically injected in the application at loading time, this means that it can intercept dynamic linking to the MPI library by instrumenting all the application calls to MPI functions before that the execution workflow pass to the MPI library. COUTDOWN supports C/C++ and Fortran HPC applications and most of the open-source and commercial MPI libraries.

The paper is organized as follows. Section 2, presents the state-of-the-art of power and energy management approaches for scientific computing systems. Section 3 introduces the key concepts on power-saving in MPI primitives of the application. Section 4 explains the components of our COUNTDOWN runtime. Section 5 reports experimental results in power saving of production runs of applications in a tier0 supercomputer.

2 RELATED WORK

In scientific computing, two main families of energy saving techniques have emerged. The first is based on the assumption that a performance penalty can be tolerated to reduce the overall energy consumption [1, 5, 6, 10]. The second is based on the assumption that it is possible to slow down the speed of a CPU only when it does not execute critical tasks to save energy without penalizing the application performance [7, 12, 16]. Both approaches are based on the concept of application slack/bottleneck (memory, IO, and communication) that can be opportunistically used to reduce performance and to save energy. However, there are drawbacks which de facto limits their usage in production environment. The first approach causes overheads in the application execution time 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.

Similar to COUNTDOWN, Lim et at. [13] adapt the core's frequency in "long" MPI phases. Instead, "short" MPI phases are grouped in "long" phases using a horizon value. MPI phases repetitions are identified using the program counter of the CPU while the P-state is selected through a prediction model based on the last value prediction on the number of micro-operations retired.

These works [7, 12–14] have in common the prediction of future workload imbalances or the time duration of MPI phases obtained by analyzing previous communication patterns. However, this approach can frequently lead to mispredictions in irregular applications [11] 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.

Eastep et al. propose GEOPM [4], 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 target 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 to identify MPI regions and to reduce the frequency based on MPI primitive type. However, while this solution is an interesting first step it cannot differentiate between short and long MPI primitives and thus cannot control the overhead caused by frequency change and runtime in short MPI primitives. In this manuscript we present an approach which solve this issue opening new opportunities for MPI-aware power reduction.

Marathe et at. [15] developed Libpowermon, a profiling framework for HPC used to correlate application metrics with system level metrics and thermal measurements. Differently from COUNTDOWN, Libpowermon implements only profiling capability without implementing any power control policy.

Benini et al. [2] presented a survey on dynamic power management policies and systems to minimize power consumption under performance constraints. In particular, they show that timeout-based shutdown policies are the most effective ones in mitigating the overheads of power states transitions which detriment the savings achievable with low power states. In this paper we leverage this property in the HPC power management context. Indeed previous works showed that unbalance in MPI workload can be exploited by power management solutions, however a overhead-free solution which can take advantage of this slack is still missing.

3 BACKGROUND AND MOTIVATION

As previously highlighted today's power management approaches for HPC systems lack the support for taking advantages of the slack induced by synchronization and communication primitives. Indeed, as described in previous section, today's CPUs have the capability of changing the performance and power consumption trade off dynamically by entering in idle (shutdown) and active (DVFS) low power states, thus in practice there are no limitation for taking advantage of them during MPI communication and synchronization phases. In this section we give two practical examples on the drawbacks and limitation of doing it. We can recognize two families of approaches: (i) bringing the core in idle low power mode or (ii) in an active low power mode each time the execution encounters an MPI call.

3.1 Target Architecture and Benchmark

We focus our analysis on 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). We use the complete software stack of Intel systems for real production environment. We use Intel *MPI Library 5.1* as the runtime for communication and Intel *ICC/IFORT 18.0* in our toolchain. We choose Intel software stack because it is currently used in our target systems as well as well supported in most of HPC machines based on Intel architectures.

QuantumESPRESSO 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*. One of its mostly used codes is PWscf (Plane-Wave Self-Consistent Field) used here to solve the self-consistent Kohn and Sham (KS) equations and obtain the ground state electronic density for a representative case study. The code uses a pseudo-potential and plane-wave approach and implements multiple hierarchical levels of parallelism implemented with a hybrid MPI+OpenMP approach. As of today, OpenMP is generally used when MPI parallelism saturates and it can improve the scalability in the highly parallel regime. Nonetheless in the following we

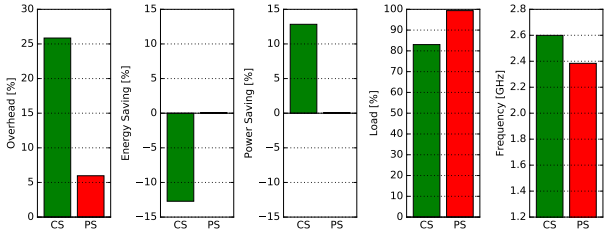

Figure 1: Overhead, energy/power saving, average load and frequency. Legend: C-state (*CS*) and P-state (*PS*) mode. Baseline is busy-waiting mode (default mode) of MPI library.

will only refer to data obtained with pure MPI parallelism since this is the main focus of this article and this choice does not significantly impair the conclusions reported later.

The following experimental results are compared with the busy-waiting mode (default mode) of MPI library, where processes continuously polling the CPU for the whole waiting time in MPI synchronization points.

3.2 Wait-mode/C-state MPI library

MPI libraries use a polling mechanism in collective synchronizations to avoid entering in C-states which can induce performance penalties. This is also the default behavior of Intel MPI library. This library can also be configured to release the control to the idle task of the operating system (OS) during waiting time to leverage on the power 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 Intel MPI library, it is possible to configure 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. Clearly, the transitions into and out of the sleep mode induce overheads in the application execution time.

In figure 1 are reported the experimental results, the wait-mode strategy is identify with *CS*. We can see the overhead induced by the wait mode w.r.t. the polling strategy, which worsen by the 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 consequently huge overheads. From the same figure, we can also see that the energy saving is negative, which is -12.72%. This means that the power savings obtained in the MPI primitives does not compensate the overhead induced by the sleep/wake-up transitions. Indeed, the power reduction is 12.83%, which is confirmed by the average load of the system, as effect of the entering in C-states during wait periods in MPI primitives. While the average frequency has not changed, and it correspond to 2.6GHz, which is the turbo frequency of our target system.

As a conclusion of this first exploration, we confirm that is not possible to leverage on the wait mode of the MPI library to save power in HPC applications without increasing the execution time. This is also the reason why MPI libraries by default work in busy-waiting mode.

3.3 DVFS/P-state MPI library

To overcome the overheads of C-state transitions, we focus our initial exploration on the DVFS states (P-state). Intel MPI library does not implement such a feature, so we manually instrumented all the MPI calls of the application with an *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 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, we can see that the overhead is significantly reduced w.r.t C-state mode, reducing the 25.85% overhead obtained previously to 5.96%. This means that the overhead of scaling the frequency is less respect to the sleep/wake-up transitions cost. However, the energy and power savings are almost zero. This is due to the fact that the application on a single node has a high number of MPI calls with very short duration. This is also confirmed by the average frequency, which does not show significant variations w.r.t. the baseline (busy waiting), with a measured average frequency of 2.4GHz. The load bar reports 100% of activity, which means that there is no idle time as expected.

In conclusion using DVFS for fine-grain power management instead of the idle mode allows to better control the overhead, however the overhead is still significant and in HPC domains performance are the prime goal.

3.4 Overhead Considerations

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. To explore if the overhead was caused by the system call that interacts with the DVFS control knobs, we force COUNTDOWN to write always the highest P-state in the DVFS control registers. Thus, we avoid application slowdown caused by frequency variation and we obtained only the overhead caused by the system call on the register access. Our experimental results report of 1.04% of overhead of the system call and for the profile sampling, so we have rule out this possibility. Though, we think that the overhead can be explained by the response time of *HW power controller* in serving P-state transition of our Intel Haswell[1] Authors in [8], show how works the Intel *HW power controller* through a reverse engineering exploration. In Intel CPUs, the *HW power controller* periodically reads the DVFS control register to check if the O.S. has specified a new frequency level, this interval has been reported to be 500us. This means that every new setting for the core's frequency faster than 500us could completely ignored or applied, depending on when the register was sampled the previous time. In our scenario, this means that it is not possible to have an effective control on the frequency selection for MPI phases shorter than 500us. For this reason, in the following Sections we present a methodology and a tool based on a timeout strategy that takes in account this technological wall to skip "short" MPI phases.

4 FRAMEWORK

COUNTDOWN is a simple run-time library for profiling and fine-grain power management written in C language. COUNTDOWN is based on a *profiler* and on an *event* module to inspect and react to the MPI primitives. Every time an application calls an MPI primitive, COUNTDOWN profiles the call and uses a timeout strategy [2] to avoid changing the power state of the cores during extremely fast application and MPI context switches, where doing so may result only in an increment of the overhead without a significant energy and power reduction. As we will see later in this Section, each time

[1]The same is true for Intel Broadwell's *HW power controller*.

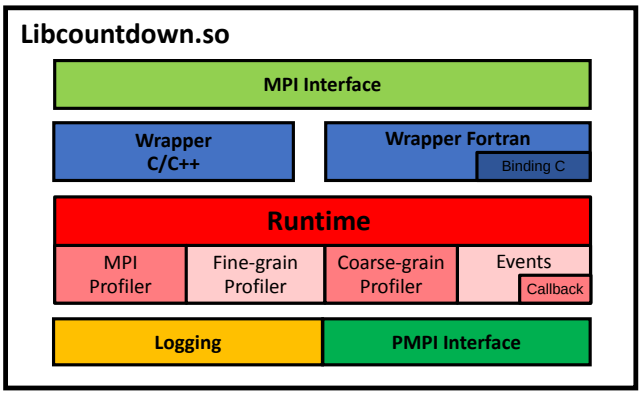

Figure 2: Logical view of COUNTDOWN components.

the MPI library asks to enter in low power mode, COUNTDOWN defers the decision for a defined amount of time. If the MPI phase terminates within this amount of time COUNTDOWN does not enter in the low power states, filtering out too short MPI phases to save energy, but costly in terms of overheads.

In figure 2 the components of the COUNTDOWN are depicted. COUNTDOWN exposes the same interface of a standard MPI library and it can intercept 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 due C/C++ and FORTRAN MPI libraries produce assembly symbols which are not application binary (ABI) compatible. The FORTRAN wrapper implements a marshalling and unmarshalling interface to bind MPI FORTRAN handlers in compatible MPI C/C++ handlers. This allows COUNTDOWN to interact with MPI libraries in FORTRAN applications. When COUNTDOWN is injected in the application, every MPI call is enclosed in a corresponding wrapper routine that implements the same signature. In the wrapper routine is called the equivalent PMPI call, but after a *prologue* routine and before an *epilogue* routine. Both routines are used from the profile and from event module to inject profiling capabilities and power management strategies in the application. COUNTDOWN interacts with the *HW power manager* through a specific *Events* module of the library. The *Events* module can also be triggered from system signals registered as callbacks for timing purposes. COUNTDOWN configurations can be done through environment variables, 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 without user intervention. When dynamic linking is not possible COUNTDOWN has also a fallback, a static-linking library, which can be used in the toolchain of the application to inject 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. Linking COUNTDOWN to the application is straightforward: it is enough to configure the environment variable *LD_PRELOAD* with the path of COUNTDOWN library and start the application as usual.

4.1 Profiler Module

COUNTDOWN uses three different profile logics targeting three different monitoring granularities.

(i) The *MPI profiler*, is responsible to collect all information regarding the MPI activity. For each MPI process, it collects information on MPI communicators, MPI groups and the coreId. In addition, it profiles each MPI call by collecting information on the type of the call, the enter and exit time and the data exchanged with the others MPI processes. The *MPI profiler* is called in the *prologue* and *epilogue* routines on the wrapper interface.

(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 assembly instruction to access to the performance monitoring units (PMU) implemented in Intel's processors. It monitors the average frequency, the time stamp counter (TSC) and the instruction retired for each MPI application phase. Moreover, it is able to access to 8 configurable performance counters in the PMU that can be used to monitor user-specific micro-architectural metrics. The *fine-grain micro-architectural profiler* is called in the *prologue* and *epilogue* routines on the wrapper interface.

(iii) The *coarse-grain profiler*, monitors a larger set of HW performance counters available in the Intel architectures. In Intel architectures to access on HW performance counters, is required a privileged permission, which cannot be granted to the final users in production machines. To overcome this limitation, we use the MSR_SAFE [17] driver to access to the model-specific registers of the system (MSR), which can be configured to grant the access of standard users to a subset of privileged architecture registers avoiding security issues. At the core level, COUNTDOWN monitors TSC, instruction retired, average frequency, C-state residencies and temperature. While at 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 information to the CPU. The *coarse-grain profiler*, due the high overhead needed by each single access to the set of HW performance counters monitored, uses a time-base sample rate. The data are collected at least Ts second delay from the previous sample. The *fine-grain micro-architectural profiler* at every MPI calls checks the time stamp of the previous sample of *coarse-grain profiler* and, if it is above Ts seconds, triggers it to get a new sample. Currently Ts is configured to 1s based on empirical studies but it is possible to configure for fast or slow sampling.

COUNTDOWN also implement a logging module to store profile information in a text file which can be written in a local or remote storage. While the log file of MPI profiler can grows with the number of MPI primitives and can become significant in long computation, the information is stored in binary files, but the logging component also summarized this information in compact text file.

4.2 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 decides the performance at which to execute a given phase.

COUNTDOWN implements the timeout strategy through the standard Linux timer APIs, which expose the system calls: *setitimer()* and *getitimer()* to manipulate user's space timers and register callback routines. This methodology is depicted in figure 3. When COUNTDOWN encounters an MPI phase, in which opportunistically can save energy by entering in a low power state, COUNTDOWN *registers* a timer callback in the *prologue* routine (Event(start)), after that the execution continues with the standard workflow of the MPI phase. When the timer expires, a system signal is raised,

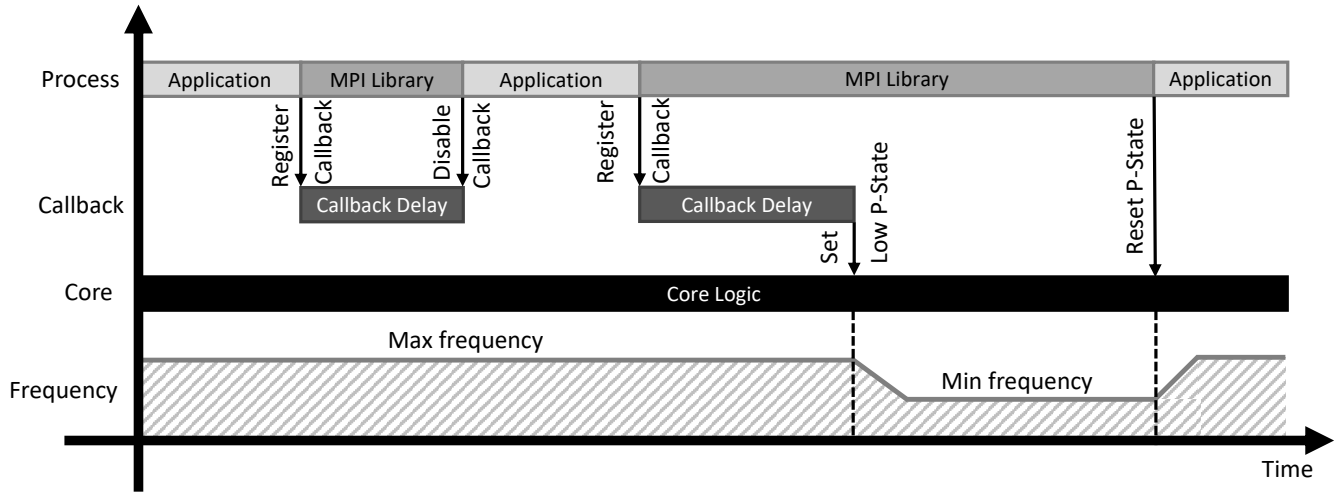

Figure 3: Timer strategy implemented in COUNTDOWN.

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* routine and the execution continues like nothing happened.

In next section we will explain why the timeout logic introduced by COUNTDOWN is effective in making fine-grain power management possible and convenient in MPI parallel application.

5 EXPERIMENTAL RESULTS

5.1 Framework Overheads

We evaluate the overhead of running MPI applications instrumented with the profiler module of COUNTDOWN without changing the cores' frequency on a single benchmark. In this evaluation, we count more than 1.1 million of MPI primitives for each process: our run-time library needs to profile in 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 COUNTDOWN profiler introduces an overhead in the execution time which is less than 1%. This result is also supported by the overhead measurements of the *prologue* and the *epilogue* routines that COUNTDOWN injects in the application for each MPI call, which costs between 1us and 2us.

5.2 Single-node Evaluation

We repeat the experiments for the P-state of Section 3 using COUNTDOWN. We configure COUNTDOWN to scale down P-state after 500us in *prologues* of MPI primitives.

Figure 4 report the experimental results. We can see that the overhead is greatly improved w.r.t. the baseline (default MPI library, without COUNTDOWN).

Using COUNTDOWN, the overhead decreases from 5.96% to a negligible overhead. Energy saving is of 1.27% and the power saving is of 5.75%. Thanks to COUNTDOWN, we make the overhead of fine-grain power management strategy negligible and we achieve an energy and power saving. In this single node example, the savings are limited but this is due to the application composition for which most of the MPI calls are very short and there is no profit margin to reduce cores' frequency. We will show that these savings become prominent in the multi-node case which a more relevant case in large scale HPC environments. These experimental results prove that COUNTDOWN is able to skip short MPI calls to avoid overheads even in application with a high density of short MPI phases.

5.3 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-0 HPC system based on an IBM NeXtScale cluster which is currently classified in the Top500 supercomputer list [3]. 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 135W TDP and interconnected with an Intel QDR (40Gb/s) Infiniband high-performance network.

We run QE on 96 compute nodes, using 3456 cores and 12 TB of DRAM. We use an input dataset capable to scale on such number of cores and we configure QE using a set of parameters optimized to avoid network bottlenecks, which would limit the scalability. We run an instance of the application with and without COUNTDOWN on the same nodes and we compared the results.

Figure 5 shows the total time spent in application and in MPI phases which are shorter and longer than 500us. On the x-axis, the figure reports the id of the MPI rank, while in the y-axis reports in percentage of the total time the accumulated 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 in the MPI library and in 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. Differently from

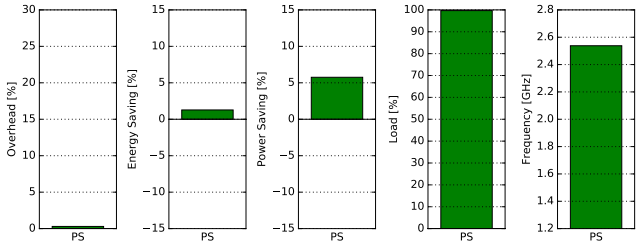

Figure 4: Overhead, energy/power saving, average load and frequency using COUNTDOWN with P-state (*PS*). Baseline is busy-waiting mode (default mode) of MPI library.

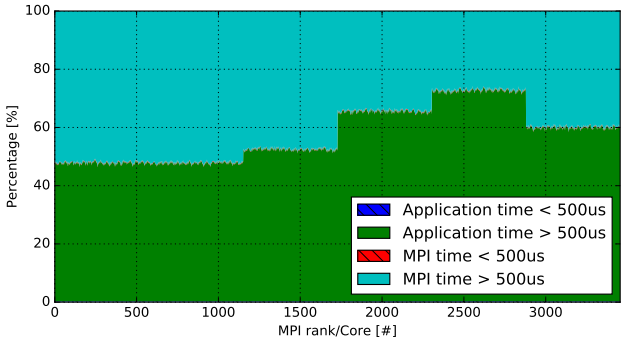

Figure 5: Sum of the time spent in phases longer and shorter than 500us for the benchmark with the optimization for the communications.

the single node evaluation we have a not negligible overhead which are not caused by short MPI phases. We suspect that some MPI primitives suffer more than others from the frequency scaling. We will analyze in dept this problem in our future works aiming to keep the COUNTDOWN overhead negligible. We believe this condition will make the adoption of COUNTDOWN wider.

The result achieved by COUNTDOWN in production scale application are very promising and if it systematically adopted would dramatically reduce the TCO of today supercomputers.

6 CONCLUSION

In this paper, we present COUNTDOWN a methodology and a tool for profiling HPC scientific applications and injecting DVFS capabilities into standard MPI libraries. COUNTDOWN implements a timeout strategy to avoid costly performance overheads and leveraging on communication slacks to drastically reduce energy consumption. Our work targets real HPC systems and workloads and does not require any kind of modification to the source code nor to the compilation toolchain of the application.

In the experimental section, we compared our system with state-of-the-art power management for MPI libraries, which can dynamically control idle and DVFS levels for MPI-base application. Our experimental results show that using timeout strategy to take decisions on power control can drastically reduce overheads maximizing the energy efficient in small and large MPI communications. Our run-time library avoids high overheads induced by short MPI calls where the *HW power manager* is not fast enough to react to fast low to high frequencies requests. Indeed, we show that thanks to the proposed COUNTDOWN approach we can reduce the energy consumption of the 22.36% and the power consumption of the 24.53% with only 2.88% of performance overhead in large-scale HPC for production application run. In future works we will extend the evaluation of COUNTDOWN with a wider set of applications and working conditions.

ACKNOWLEDGMENTS

Work supported by the EU FETHPC project ANTAREX (g.a. 671623), EU project ExaNoDe (g.a. 671578), and EU ERC Project MULTITHERMAN (g.a. 291125).

REFERENCES

[1] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. 2014. A Case Study of Energy Aware Scheduling on SuperMUC. In *Lecture Notes in Computer Science*. Springer International Publishing, 394–409. https://doi.org/10.1007/978-3-319-07518-1_25

[2] L. Benini, A. Bogliolo, and G. De Micheli. 2000. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8, 3 (June 2000), 299–316. https://doi.org/10.1109/92.845896

[3] Jack J Dongarra, Hans W Meuer, Erich Strohmaier, et al. 1995. TOP500 supercomputer sites. *Supercomputer* 11 (1995), 133–133.

[4] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Brad Geltz, Federico Ardanaz, Asma Al-Rawi, Kelly Livingston, Fuat Keceli, Matthias Maiterth, and Siddhartha Jana. 2017. Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions. In *Lecture Notes in Computer Science*. Springer International Publishing, 394–412. https://doi.org/10.1007/978-3-319-58667-0_21

[5] Francesco Fraternali, Andrea Bartolini, Carlo Cavazzoni, and Luca Benini. 2017. 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* (2017), 1–1. https://doi.org/10.1109/tpds.2017.2766151

[6] Francesco Fraternali, Andrea Bartolini, Carlo Cavazzoni, Giampietro Tecchiolli, and Luca Benini. 2014. 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 - ISLPED '14*. ACM Press. https://doi.org/10.1145/2627369.2627659

[7] Vincent W Freeh, Nandini Kappiah, David K Lowenthal, and Tyler K Bletsch. 2008. Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. *J. Parallel and Distrib. Comput.* 68, 9 (2008), 1175–1185.

[8] Daniel Hackenberg, Robert Schone, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. 2015. An Energy Efficiency Feature Survey of the Intel Haswell Processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE. https://doi.org/10.1109/ipdpsw.2015.70

[9] Emma Jane Hogbin. 2015. ACPI: Advanced Configuration and Power Interface. (2015).

[10] Chung hsing Hsu and Wu chun Feng. [n. d.]. A Power-Aware Run-Time System for High-Performance Computing. In *ACM/IEEE SC 2005 Conference (SC'05)*. IEEE. https://doi.org/10.1109/sc.2005.3

[11] Darren J Kerbyson, Abhinav Vishnu, and Kevin J Barker. 2011. Energy templates: Exploiting application information to save energy. In *2011 IEEE International Conference on Cluster Computing*. IEEE, 225–233.

[12] Dong Li, Bronis R de Supinski, Martin Schulz, Kirk Cameron, and Dimitrios S Nikolopoulos. 2010. Hybrid MPI/OpenMP power-aware computing. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 1–12.

[13] Min Yeol Lim, Vincent W Freeh, and David K Lowenthal. 2006. Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In *SC 2006 conference, proceedings of the ACM/IEEE*. IEEE, 14–14.

[14] C. Liu, A. Sivasubramaniam, M. Kandemir, and M. J. Irwin. 2005. Exploiting Barriers to Optimize Power Consumption of CMPs. In *19th IEEE International Parallel and Distributed Processing Symposium*. 5a–5a. https://doi.org/10.1109/IPDPS.2005.211

[15] Aniruddha Marathe, Hormozd Gahvari, Jae-Seung Yeom, and Abhinav Bhatele. 2016. Libpowermon: A lightweight profiling framework to profile program context and system-level metrics. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE, 1132–1141.

[16] Barry Rountree, David K Lownenthal, Bronis R De Supinski, Martin Schulz, Vincent W Freeh, and Tyler Bletsch. 2009. Adagio: making DVS practical for complex HPC applications. In *Proceedings of the 23rd international conference on Supercomputing*. ACM, 460–469.

[17] Kathleen Shoga, Barry Rountree, Martin Schulz, and Jeff Shafer. 2014. Whitelisting MSRs with msr-safe. In *3rd Workshop on Exascale Systems Programming Tools, in conjunction with SC14*.