


Benefits in relaxing the power capping constraint

Conference Paper**Author(s):**

Cesarini, Daniele; Bartolini, Andrea; [Benini, Luca](#) 

Publication date:

2017-09-09

Permanent link:

<https://doi.org/10.3929/ethz-b-000224609>

Rights / license:

[In Copyright - Non-Commercial Use Permitted](#)

Originally published in:

<https://doi.org/10.1145/3152821.3152878>

This is the post peer-review accepted manuscript of:

Daniele Cesarini, Andrea Bartolini and Luca Benini, "Benefits in Relaxing the Power Capping Constraint", 1st Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems (ANDARE'17), Portland (OR), USA, 2017. doi: 10.1145/3152821.3152878

The published version is available online at: <https://ieeexplore.ieee.org/abstract/document/8203471>

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Benefits in Relaxing the Power Capping Constraint

ABSTRACT

In this manuscript we evaluate the impact of HW power capping mechanisms on a real scientific application composed by parallel execution. By comparing HW capping mechanism against static frequency allocation schemes we proved that application speed up can be achieved by relaxing the time granularity at which the power cap is maintained. Enforcing RAPL on a few ms time scale fails on sharing power budget between more demanding and less demanding application phases.

KEYWORDS

power capping, DVFS, HPC, RAPL, monitoring, P-states, power management, hardware performance counters

1 INTRODUCTION

The pace dictated by the Moore's law on technological scaling has provided a constant increase in the performance of computing devices. However with the breakdown of Dennard scaling, this has come with an increase in the power consumption.

Supercomputers are the cutting edge of computing performance. Supercomputers integrate hundred thousands of the most powerful processors and accelerators and they are periodically ranked based on their peak performance [1]. Until June 2016, every new most powerful supercomputer in the world (1st in the Top500 list) has marked an increase in its power consumption. Reaching in 2013, with Tianhe-2, 17.8 MWatts of IT peak power consumption, which increases to 24MWatt when considering also the cooling power [8]. This has set the record for the power consumption of a single supercomputer installation, reaching the practical limit in power provisioning which is 20MWatt [3]. The today's most powerful supercomputer (Taihulight) consumes 15.4 MWatt underlining the fact that performance increase are nowadays possible only at a fix power budget: as a matter of fact supercomputers are power limited!

To ensure this power budget during design time, it requires considering the worst-case power consumption of the computing resources. However supercomputer workloads rarely causes worst-case power consumption during their life making worst-case design approaches a bad choice which decreases the average supercomputing performance.

Power capping approaches support the design for "the average power consumption case" by packing more computing capacity, with a feasible power budget under average workload and dynamically reducing the performance during the execution of workloads with peak power consumption.

At the basis of these approaches there is the capability of computing elements in trading off performance to power consumption. This is done by mean of dynamic power management states. Shutdown states (ACPI C-states [12]) allow to reduce power consumption of idle resources. Dynamic and Voltage Frequency states (DVFS) (ACPI P-states [12]) allow to reduce the power consumption during active computation. Dynamic power management policies take advantages of these HW states to create feedback loops adapting the performance to workload phases aiming to reduce the energy consumption or ensure a specific power and thermal budget. Pure software implementations of these policies have clear software advantages but need to be executed on the same computational resources, interrupting the application flow and causing overheads. Recently vendors have added HW components to implement in HW these policies allowing a more fine-grain control. Intel Run Average Power Limits (RAPL) technology can enforce in hardware a given power limit. This is done exploiting a power sensors and power management knobs. Current RAPL implementations aim to enforce a constant power consumption within a 1-46ms time windows.

However this is far below the timescale of supercomputing centres where the power budget must be respected in large time windows coming from the power grid and supplier [6, 7]. Differently, RAPL enforces the power cap for every application phase without taking into account the real power efficiency of these phases. RAPL mechanism is quite strong to maintain the assigned power budget and can produce inefficiency respect to weak strategies targeting the control of the average system power consumed on longer horizons. In this work we compare RAPL with a policy that maintains the same power consumption budget for the entire duration of the application using as simple static assignment of frequencies. This comparison serves to highlight opportunities for improving RAPL performance by relaxing the today practice of constraining the power consumption on a fine-time granularity.

The paper is organized as follows. Section 2, presents related works, Section 3 characterizes HPC architectures and their power manager used to power constraint the system. Section 4 defines our architecture and application targets presenting the monitoring infrastructure. While section 5 shows the methodology used for the exploration and our experimental results.

2 RELATED WORK

Several approaches in the literature have proposed mechanisms to constraint the power consumption of large scale computing infrastructures. These can be classified in two main families. Approaches in the first class use predictive models to estimate the power consumed by a job before its execution. At job schedule time this information is used to let enter jobs in the system that satisfy the total power consumption budget. Hardware power capping mechanism like RAPL are used to ensure that the predicted budget is respected during all the application phases and to tolerate prediction errors in the job average power consumption estimation [4, 5, 16]. Approaches in the second class distribute a slice of the total system power budget to each computing element which is active. The per-compute element power budget is ensured by mean of hardware power capping mechanism like RAPL. The allocation of the power consumption budget to each compute nodes can be done statically or dynamically [9, 13, 15]. It is goal of the run-time to trade off power reduction with application performance loss. The

GEOPM [9] runtime developed by Intel is an open source, plugin extensible runtime for power management. GEOPM implement a plugin for power balancing to improve performance in power constraint systems reallocating power on sockets involved in the critical path of the application. Authors in [17] quantitatively evaluated RAPL as a control system in term of stability, accuracy, settling time, overshoot, and efficiency. In this work, authors evaluate only the proprieties of RAPL mechanism without considering other power capping strategies and how can vary application workload.

3 HPC ARCHITECTURES AND POWER MANAGEMENT SYSTEMS

3.1 HPC Architectures

HPC systems are composed of tens to thousands computational nodes interconnected with a low-latency high-bandwidth network. Nodes are usually organized in sub-clusters allocated at execution time from the system scheduler according to the user request. Sub-clusters have a limited lifetime, after which resources are released to the system scheduler. Users request resources through a batch queue system, where they submit applications to be executed. Even a single node can be split in multiple resources shared among users. The single indivisible units in a HPC machine are CPU, memory and possibly accelerators (GPGPU, FPGA, Many-core accelerator, etc.).

HPC software are Single Program Multiple Data (SPMD) application, the same application executable is instanced multiple time on different nodes of the cluster, each instance works on a partition of the global workload and communicates with other instances to orchestrate next computational steps. For this reason, a HPC application can be seen as the composition of several tasks executed in a distributed environment which exchanges messages among all the instances. Achieving high-performance communication on distributed applications in large clusters is not an easy task. The Message-Passing Interface (MPI) runtime responds to these demands by abstracting the level of network infrastructure using a simple but high-performance interface for communication that can scale up on thousands of nodes.

HPC machines are extremely energy consumers, moreover server rooms require a proportioned cooling system to avoid overheating situations. The extreme working conditions of these kind of machines brings a lot of inefficiencies in terms of energy and thermal control, that turn in computational performance degradation. Hardware power manager are becoming a fundamental component to control power utilization using different strategies in order to reduce energy waste and, at the same time, to assure a safe-thermal environment.

3.2 Power Management in HPC Systems

Nowadays, operating systems can communicate with different hardware power managers through an open standard interface called Advanced Configuration and Power Interface (ACPI) [12]. In this work, we focus on ACPI implementation of Intel architecture, moreover most of HPC machines (more than 86% in [1]) are based on Intel CPUs. Intel implements ACPI specification defining different component states which a CPU can use to reduce power consumption. Today's CPU architectures are composed of multiple processing elements (PE) which communicate through a network subsystem that interconnect PEs, Last Level Cache (LLC), Integrated Memory Controllers (IMC) and other uncore components. Intel architecture optimizes ACPI using different power saving levels for cores and uncore components. The ACPI standard defines P-states to select DVFS operating points targeting the reduction of active

power, while defines C-States the idle power levels. In our work, we take into account only P-states to manage DVFS control knob, this because HPC applications do not manifest idle time during the execution.

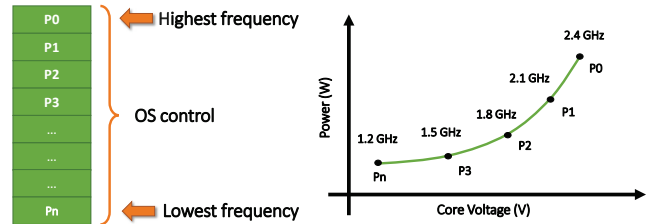


Figure 1: DVFS levels and Intel P-states

Intel P-States show in figure 1, define a number of levels which are numbered from 0 to n where n is the lowest frequency and 0 is the highest frequency with the possibility to take advantage of Turbo Boost technology. Turbo Boost is an Intel technology that enables processors to increase their frequency beyond the nominal via dynamic control of clock rate. The maximum turbo frequency is limited by the power consumption, thermal limits and the number of cores that are currently using turbo frequency. Since Haswell, Intel cores allow independent per-core P-State.

3.2.1 Intel Power Management Driver. Intel P-States are managed by a power governor implemented as a Linux kernel driver. By default on Linux system, Intel architectures are managed by a kernel module called *intel_pstate*.

This driver implements a Proportional-Integrated-Derived (PID) feedback controller. The PID controller calculates an error value every 10ms as the difference between a desired setpoint and the measured CPU load in that period. The PID controller acts to compensate this error by adapting the P-State value. The PID internal parameters are defined with default values by the Intel driver but can be customized by system administrator.

Inside the *intel_pstate* driver only two governors are implemented: *powersave* (default) and *performance*. We will not describe in details the operations of these governors because it is outside the scope of this work, but from a practical point of view, *performance* always maintains the CPU at maximum frequency while *powersave* can choose a different level depending of the machine workload. Hence, *powersave* tries to achieve better energy efficiency while *performance* tries to achieve the best performance at the expense of higher energy consumption.

3.2.2 Linux Power Management Driver. The *intel_pstate* driver does not support a governor that allows users to select per-core fixed frequency. Differently, the default power management driver of Linux *acpi-cpufreq* does it. *acpi-cpufreq* is similar to Intel driver but implement a large set of governors which implement different algorithms. The available governors are:

- (1) *powersave*: this governor differently from Intel driver, runs the CPU always at the minimum frequency.
- (2) *performance*: runs the CPU always at the maximum frequency.
- (3) *userspace*: runs the CPU at user specified frequencies.
- (4) *ondemand*: scales the frequency dynamically according to current load. It is equivalent to the *powersave* governor of Intel driver [14].
- (5) *conservative*: similar to *ondemand* but scales the frequency more gradually.

In our work, we use *userspace* governor to select fixed frequencies for all the duration of our benchmarks.

3.3 Hardware Power Controller

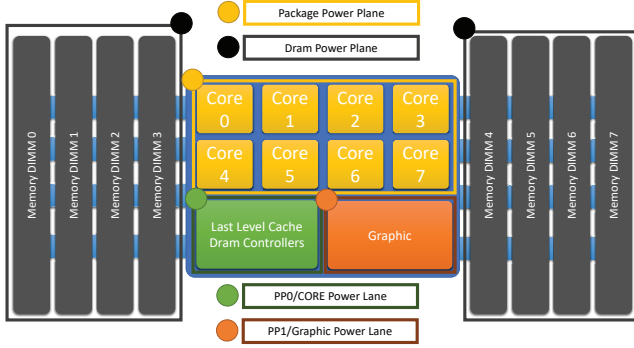


Figure 2: Intel RAPL design with the identification of power domains

Today’s CPU architectures implement reactive hardware controller to maintain the processor always under an assigned power budget. The HW controller tries to maximize the overall performance while constraining the power consumption and maintaining a safe silicon temperature. Intel architectures implement in its CPU a hardware power controller called Running Average Power Limit (RAPL) depicted in figure 2. RAPL is a control system, which receives as input a power limit and a time window. As consequent, RAPL continuously tunes the P-states to ensure that the limit is respected in the specified time window. RAPL can scale down and up core’s frequencies when the power constraint is not respected overriding the selected P-states. RAPL power budget and time window can be configured writing a Machine Specific Register (MSR) on the CPU. Maximum and minimal values for both power budget and time window are specified in a read-only architectural register. Values for both power and time used in RAPL are represented as multiple of a reference unit contained in a specific architectural register. At the machine startup, RAPL is configured using thermal design power (TDP) as power budget with a 10ms time window. RAPL also provides 32bit performance counters for each power domain to monitor the energy consumption and the total throttled time. RAPL implements four power domains which can be independently configured:

- (1) *Package Domain*: this power domain limits the power consumption for the entire package of the CPU, this includes cores and uncore components.
- (2) *DRAM Domain*: this power domain is used to power cap the DRAM memory. It is available only for server architectures.
- (3) *PP0/Core Domain*: is used to restrict the power limit only to the cores of the CPU.
- (4) *PP1/Graphic Domain*: is used to power limit only the graphic component of the CPU. It is available only for client architectures due Intel server architectures do not implement graphic component into the package.

In the experimental result section we focus our exploration on the package domain of RAPL controller because core and graphic domains are not available on our Intel architecture. DRAM domain is left for future exploration works. We also tried to modify the time windows of package domain (which can be set in a range of 1ms to 46ms in our target system) to see its impact on application

performance. Our results show that this parameter does not lead to noticeable changes in the results obtained. For this reason we report results only for the default 10ms time window configuration.

4 BENCHMARKING SCENARIO

4.1 Architecture Target

In this work, we take as architecture target a high performance computing infrastructure, which is a Tier-1 HPC system based on an IBM NeXtScale cluster. Each node of the system is equipped with 2 Intel Haswell E5-2630 v3 CPUs, with 8 cores with 2.4 GHz nominal clock speed and 85W Thermal Design Power (TDP, [11]). As regards the software infrastructure, SMP CentOS Linux distribution version 7.0 with kernel 3.10, runs on each node of the system. We use the complete software stack of Intel systems for HPC production environment. In particular, we use *Intel MPI Library 5.1* as the runtime for communication and *Intel ICC/IFORT 16.0* in our toolchain. This Tier-1 supercomputer is currently classified in the Top500 supercomputer list [1]. We focus our analysis on a single node of the cluster.

4.2 Application Target

Quantum ESPRESSO (QE) [10] is an integrated suite of computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is an open source package for research in molecule dynamics simulations and it is freely available to researchers around the world under the terms of the GNU General Public License. Quantum ESPRESSO is commonly used in high-end supercomputers. QE main computational kernels include dense parallel Linear Algebra (LA) and 3D parallel Fast Fourier Transform (FFT). Moreover, most of application workload is based on LA and FFT mathematical kernels which makes our exploration work relevant for many HPC codes. In our tests we use a Car-Parrinello (CP) simulation, which prepares an initial configuration of a thermally disordered crystal of chemical element by randomly displacing the atoms from their ideal crystalline positions. This simulation consists of a number of test that have to be executed in the correct order.

4.3 Monitoring Framework

In this section, we describe in details the monitoring framework used to profile the application and the system. Our monitoring framework is composed to two monitoring tools. The first one can monitor several hardware performance counters with a regular time stamping. The second monitoring framework is synchronized with parallel phases allowing to isolate performance and architectural metrics for each program phase. However, due to the higher number of monitoring points per time unit it can access to only on a sub-set of performance counters of the one monitored by system monitoring tool. Our monitoring frameworks have a minimal overhead, less than 1% w.r.t. application execution time.

4.3.1 System-aware Monitoring Tool. We use as system-aware monitoring tool Examon [2]. This monitoring tool can be used to read periodically per-core frequency, CPI and scalar/vector instructions retired. In addition, it can monitor for each socket the DRAM memory bandwidth and package power consumption using RAPL performance counters. This monitor is a simple daemon process that access to the performance counters of the CPU using MSR read/write operations. The daemon starts at a given T_{samp} rate, in our benchmarks we use a T_{samp} of 1 second. Its overhead is negligible.

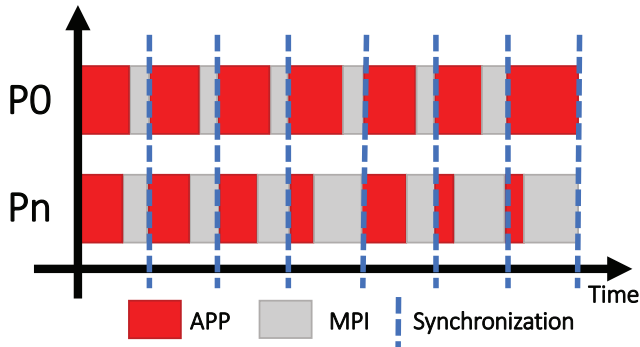


Figure 3: Phases of computation and communication identified by application-aware Monitoring Runtime

4.3.2 Application-aware Monitoring Runtime. We developed a monitor runtime to extract system information synchronously with the application flow. The runtime is a simple wrapper of the MPI library where every MPI function of each process has been enclosed by an epilogue and a prologue function. We used the MPI standard profiling interface (PMPI), which allow us to intercept all the MPI library functions without modify the application source code. The runtime is integrated in the application at linked time. Hence, Application-aware Monitoring Runtime is able to extract information distinguishing application and MPI phases as shows in figure 3. The monitor runtime uses *RDPMC* and *RDTSC* assembly instructions to access respectively the Time Stamp Counter (TSC) and Intel Performance Monitoring Unit (PMU) counters with an overhead of few hundreds of cycles for each counter access. PMU counters are programmable through standard MSR operations which require administrative permission and are costly in terms of access overhead. However, the counter value can be read using the *RDPMC* instruction directly from user space and without involving syscalls. We programmed per-core PMU registers to monitor frequency, CPI, and scalar/vector instructions retired. The monitor runtime can intercept a very high number of MPI calls of the application, for this reason is not possible to use MSR operations to access low level performance counters through syscalls, which cause high-performance penalty.

5 EXPERIMENTAL RESULTS

5.1 Methodology

We run QE-CP with a configuration of 16 MPI processes with a one-to-one bind to each core of our HPC node. We start by comparing different configurations of power capping in our test environment. Initially, we split the power budget in an equal manner on both sockets, we set 48W on each socket for a global power envelop of 96W. This test shows that core’s frequencies on different sockets are heterogeneous, suggesting that the two sockets have different inherit power efficiency. To have the same frequency among all the cores, the tested computing node needs of 11.31% higher power on socket 0. As consequent of this result, we run a set of benchmarks fixing the same frequency for all the cores while monitoring the power consumption of each socket. We use this per-socket power budget as power constraint to obtain the same frequency among all the cores. We execute again the tests using RAPL to impose these per-socket power caps and leave RAPL decides the actual frequency.

Table 1 shows the results of our set of experiments using different levels of power caps. In the first column there are reported the

target frequencies used to extract the power limits specified in the second column. Second and third columns show the sum of power consumption of both sockets using DVFS and RAPL mechanisms for power capping. We can see that the power consumption are the same, so the power cap is respected and the tests are comparable. In the frequency columns are reported the average frequencies for the entire application and among all the cores. These columns show that RAPL has an average frequency of 11.14% higher than DVFS but, if we take a look at the execution time (reported in next columns), DVFS has a lower execution time, in average 2.87% faster than RAPL. In the next sections we will explore why DVFS power cap has a lower execution time respect to RAPL which, in contrast, has a higher average frequency.

5.2 System Analysis

Figure 4 shows a time window of the system-aware monitoring tool for both the power capping mechanisms while QE-CP iterates on the same computational kernel. The test reports the case of a power constraint relative to 1.5 GHz for DVFS and RAPL power cappers. So, the results are comparable directly.

First of all, we can check the correct behaviors of power capping logic by looking at the core’s frequencies and package power consumption (first two top plots). In the DVFS plot on the left part of figure 4, core’s frequencies are fixed at 1.5 GHz while package power consumption floats around the average value as effect of the different application phases. In contrast, RAPL (on the right) maintains constant the power consumption for both the sockets while core’s frequencies changes following the current application phase. Table 1 reports a similar average power consumption for both the two cases, thus the power cappers are working as expected. Both benchmarks show a lower CPI when the memory bandwidth is low and SIMD instructions retired are high. In these phases, RAPL has lower frequency than the DVFS case as effect of the higher power demand of SIMD instructions. On the other hand, RAPL assigns higher frequencies than DVFS when CPI is high and this happens when the application is moving data from/to memory as proved by the high memory traffic/bandwidth reported by the Mem Ch[GB/s] plot. In these phases, the number of SIMD instructions retired are lower and, as already pointed out and shown in the RAPL plot, the core’s frequencies selected by RAPL increases above average due the higher power budget. However, increasing core’s frequencies when the application is memory bound does not reflect in a consequent performance gain due the higher CPI and sub-linear dependency of application speed-up with frequency in these phases.

Hence, DVFS power capper is more efficient of RAPL (shorter execution time) for two reasons: i) DVFS executes with higher instruction per seconds when application has high SIMD instructions density. ii) RAPL instead reduces the core’s frequency in the same phase to avoid excessive power consumption. On the contrary, RAPL increases the frequency during memory bound phases obtaining a similar average power as the DVFS case.

From the RAPL case (plots on the right), we can also notice that the time spent to move data from memory to CPUs is higher than the time spent in computation, because the duration of high SIMD phases are shorter than the duration of memory traffic and CPI regions. This behavior entails a higher average frequency for RAPL power capping respect to DVFS, which is not translated in a shorter application executing time.

5.3 Application Analysis

In this section, we monitor the system using the application-aware monitoring runtime. This runtime is able to recognize application

Table 1: Quantum ESPRESSO - Power Capping

	Power		Frequency		Execution Time			
	DVFS	RAPL	DVFS	RAPL	DVFS vs RAPL	DVFS	RAPL	DVFS vs RAPL
1.5 GHz	95.56W	94.81W	1499MHz	1766MHz	-15.11%	311.43sec	328.16sec	5.10%
1.8 GHz	111.86W	110.63W	1797MHz	2144MHz	-16.22%	274.11sec	274.42sec	0.11%
2.1 GHz	122.87W	120.71W	2094MHz	2323MHz	-9.86%	247.60sec	254.59sec	2.75%
2.4 GHz	134.44W	131.32W	2392MHz	2476MHz	-3.37%	231.19sec	239.65sec	3.53%

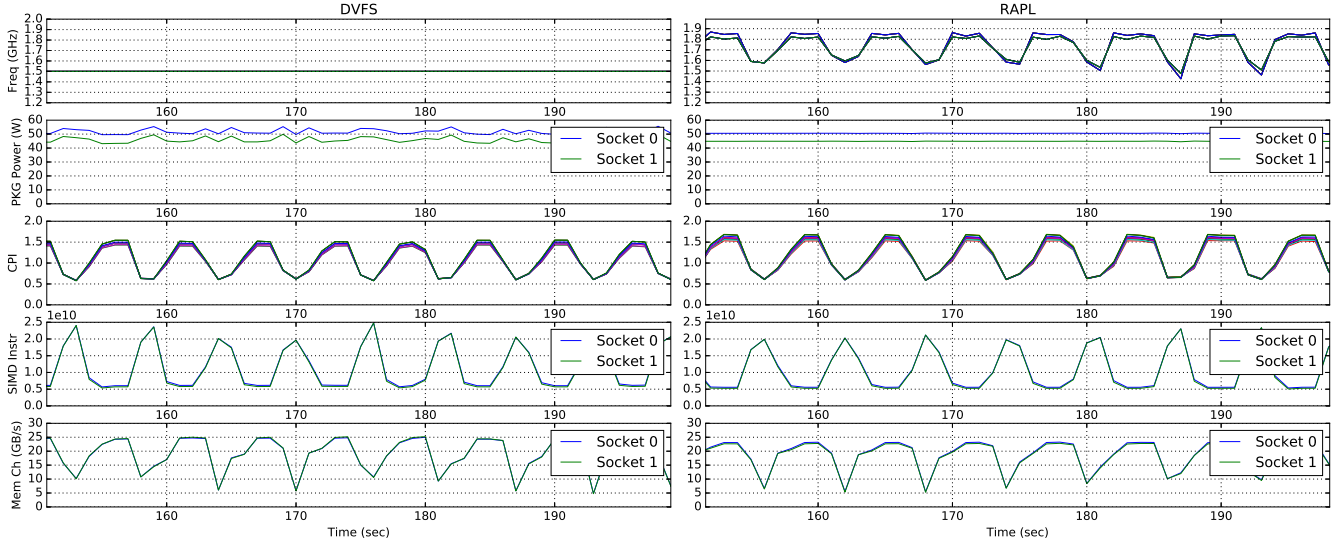


Figure 4: Time window of 50 second of the system monitor

phases marked by global synchronization points as depicted in figure 4. In the figures 5, 6, and 7 are reported the average values of performance counters of RAPL power capper divided into frequency operational intervals. In figure 5 is depicted the amount of time for computation (APP) and for communication (MPI) phases. Figure 6 shows the time gain for the DVFS power capper respect to RAPL power capper integrated on a given RAPL frequency range. Values are in seconds. Negative values mean seconds of application execution time reduction saved by DVFS with respect to RAPL power capper. Figure 7 shows the values for CPI and SIMD instructions retired for the same phases that RAPL executes at a given frequency.

To explain the behaviours that characterizes DVFS and RAPL power cappers, we need to look at all the three plots together. Starting from figure 6, we can recognize that in the frequency interval 1.3 - 1.5 GHz and 1.7 - 1.9 GHz, DVFS power capper obtains its most speed up. The first gaining interval is justified by the high number of SIMD instructions retired and by the lower CPI with respect to other application phases. Indeed from figure 7, we can notice that most of the SIMD instructions are executed with these lower frequencies by RAPL.

In the interval 1.7 - 1.9 GHz, the CPI is higher and the SIMD instructions retired are not negligible. From figure 5, we can recognize that most the application time is spent in this frequency range. With a lower SIMD instructions density respect to the 1.3 - 1.5 GHz interval. Hence, high CPI, low density of SIMD instructions and high frequency suggest memory bound phases as shown by previous section. Interesting these phases runs at higher frequency

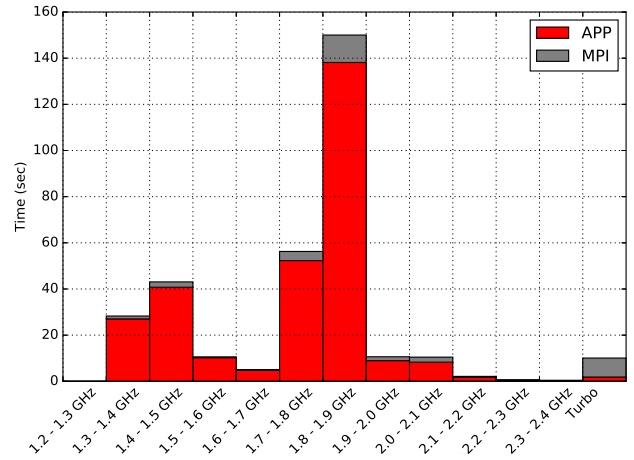


Figure 5: Sum of MPI and application time grouped by interval frequencies

than the DVFS but leads to a performance penalty. This suggests side effects of high frequency in terms of memory contentions.

In the interval 2.0 - 2.1 GHz, RAPL has a performance gain with respect to DVFS. This behavior is explained by the CPI and the number of SIMD instructions retired during this phase. In this interval, RAPL has a low CPI and does not perform SIMD instructions,

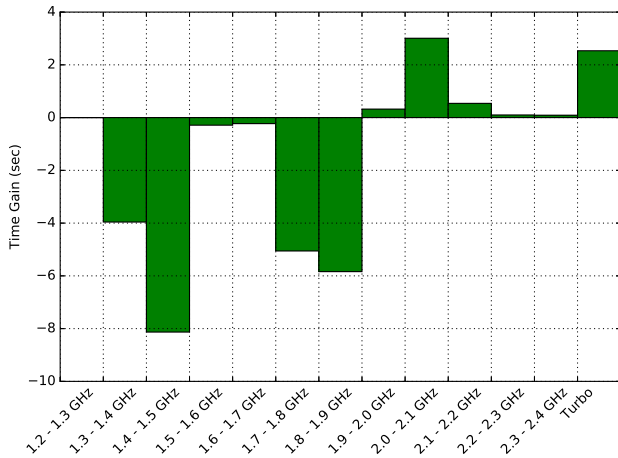


Figure 6: Time gain of DVFS w.r.t RAPL grouped by interval frequencies

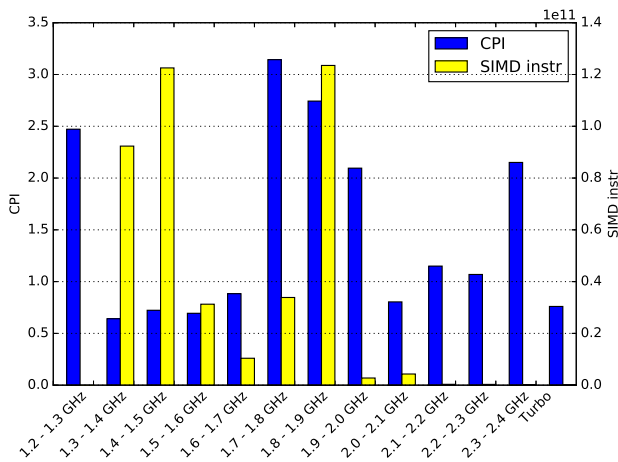


Figure 7: Average CPI and number of AVX instructions required on different interval frequencies

so this phase scales its execution with the frequency. RAPL can dynamically manage the available power budget made available by the low SIMD instructions and can increase the frequency. This leads to a consequent performance increment.

In the turbo frequency interval, RAPL performs better than DVFS as it is a MPI reduction phase where only the root process is active. During the reduction, all the processes except the root MPI remain in a barrier to wait the termination of the root. This is explained by the high MPI runtime time (figure 5) present at this frequency interval. Hence, RAPL can use the power budget released by the processes in barriers to speed up the root process leading to a performance gain.

6 CONCLUSION

In this paper we presented a novel exploration of power capping mechanisms for power constrained HPC nodes. Differently from

state-of-the-art explorations, we focused our analysis on an exploration of power capping strategies used in real HPC system node. In details, we explored the characteristics of Intel RAPL and DVFS power manager during the execution of a real scientific HPC application which is performance constrained by the power budget assigned to the node.

Our exploration explains why RAPL mechanism has an average performance penalty of 2.87% and up to 5.10% w.r.t. DVFS power manager, even if RAPL shows a higher average frequency for the entire application time. This proved that dynamically manage the available power budget without be aware of the application phases is not always beneficial from performance point of view. Furthermore, RAPL mechanism always increases the core's frequencies during less demanding phases to fill in the available power budget, leading to unnecessary power consumption in memory bound phases that can cause interference and as consequence performance loss.

REFERENCES

- [1] 2017. TOP500.Org. Top 500 Supercomputer Sites. <http://www.top500.org>. (2017).
- [2] Francesco Beneventi, Andrea Bartolini, Carlo Cavazzoni, and Luca Benini. 2017. Continuous learning of HPC infrastructure models using big data analytics and in-memory processing tools. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1038–1043.
- [3] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. 2008. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep 15* (2008).
- [4] Andrea Borghesi, Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini. 2016. Predictive Modeling for Job Power Consumption in HPC Systems. In *International Conference on High Performance Computing*. Springer, 181–199.
- [5] Andrea Borghesi, Christian Conconi, Michele Lombardi, and Andrea Bartolini. 2015. MS3: a Mediterranean-Style Job Scheduler for Supercomputers-do less when it's too hot!. In *High Performance Computing & Simulation (HPCS), 2015 International Conference on*. IEEE, 88–95.
- [6] Hao Chen, Michael C Caramanis, and Ayse K Coskun. 2014. The data center as a grid load stabilizer. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE, 105–112.
- [7] Hao Chen, Michael C Caramanis, and Ayse K Coskun. 2014. Reducing the data center electricity costs through participation in smart grid programs. In *Green Computing Conference (IGCC), 2014 International*. IEEE, 1–10.
- [8] Jack Dongarra. 2013. Visit to the national university for defense technology changsha, china. *Oak Ridge National Laboratory, Tech. Rep., June* (2013).
- [9] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Federico Ardanaz, Brad Geltz, Asma Al-Rawi, Fuat Keceli, and Kelly Livingston. 2016. Global extensible open power manager: a vehicle for HPC community collaboration toward co-designed energy management solutions. *Supercomputing PMBS* (2016).
- [10] Paolo Giannozzi, Stefano Baroni, Nicola Bonini, Matteo Calandra, Roberto Car, Carlo Cavazzoni, Davide Ceresoli, Guido L Chiarotti, Matteo Cococcioni, Ismaila Dabo, et al. 2009. QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. *Journal of physics: Condensed matter* 21, 39 (2009), 395502.
- [11] Per Hammarlund, Rajesh Kumar, Randy B Osborne, Ravi Rajwar, Ronak Singhal, Reynold D'Sa, Robert Chappell, Shiv Kaushik, Srinivas Chennupaty, Stephan Jourdan, et al. 2014. Haswell: The fourth-generation intel core processor. *IEEE Micro* 34, 2 (2014), 6–20.
- [12] Emma Jane Hogbin. 2015. ACPI: Advanced Configuration and Power Interface. (2015).
- [13] Aniruddha Marathe, Peter E Bailey, David K Lowenthal, Barry Rountree, Martin Schulz, and Bronis R de Supinski. 2015. A run-time system for power-constrained HPC applications. In *International Conference on High Performance Computing*. Springer, 394–408.
- [14] Venkatesh Pallipadi and Alexey Starikovskiy. 2006. The ondemand governor. In *Proceedings of the Linux Symposium*, Vol. 2. sn, 215–230.
- [15] Osman Sarood, Akhil Langer, Abhishek Gupta, and Laxmikant Kale. 2014. Maximizing throughput of overprovisioned hpc data centers under a strict power budget. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 807–818.
- [16] Alina Sirbu and Ozalp Babaoglu. 2016. Predicting system-level power for a hybrid supercomputer. In *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 826–833.
- [17] Huazhe Zhang and H Hoffman. 2015. A Quantitative Evaluation of the RAPL Power Control System. *Feedback Computing* (2015).