Combining uncore frequency and dynamic power capping to improve power savings

Amina Guermouche

Combining Uncore Frequency and Dynamic Power Capping to Improve Power Savings

Amina Guermouche
University of Bordeaux, IMB, UMR 5251
INRIA

Talence, France
amina.guermouche@inria.fr

***Abstract*—The US Department of Energy sets a limit of 20 to 30 MW for future exascale machines. In order to control their power consumption, modern processors provide many features. Power capping and uncore frequency scaling are examples of such features which allow to limit the power consumed by a processor.**

In this paper, we propose to combine dynamic power capping to uncore frequency scaling. We propose DUFP, an extension of DUF, an existing tool which dynamically adapts uncore frequency. DUFP dynamically adapts the processor power cap to the application needs. Finally, just like DUF, DUFP can tolerate performance loss up to a user-defined limit. With a controlled impact on performance, DUFP is able to provide power savings with no energy loss.

The evaluation of DUFP shows that it manages to stay within the user-defined slowdown limits for most of the studied applications. Moreover, combining uncore frequency scaling to power capping: (i) improves power consumption by up to 13.98 % with additional energy savings for applications where uncore frequency scaling has a limited impact, (ii) improves power consumption by up to 7.90 % compared to using uncore frequency scaling by itself and (iii) leads to more than 5 % power savings at 5 % tolerated slowdown with no energy loss for most applications.

***Index Terms*—Power capping, Uncore frequency, Power consumption, Energy consumption**

I. Introduction

Reducing the power consumption of supercomputers has become one of the key challenges in high performance computing. As a matter of fact, Fugaku, the most powerful supercomputer consumes 20 MW [29] while the US Department of Energy sets a limit of 20 to 30 MW for future exascale machines [1].

There have been many efforts to reduce processors power consumption when an application is being executed. Some techniques rely on frequency scaling, targetting either core frequency [23], uncore frequency [4], [13] or both [24], [26], [27]. Computing the best frequency to apply can also be done through learning techniques [14], [19].

Another adopted solution within HPC systems is power capping. Power capping forces the power consumption to stay within the enforced power cap. By running applications under a reduced power budget, large power savings can be reached. However, this comes with an impact on performance which is not negligible. Moreover, studies showed that under power capping, the default frequency scaling fails to adapt to the application needs [4], [13].

In this work, we tackle the problem of power capping dynamically with a controlled impact on performance. We combine both power capping and uncore frequency scaling. This work tries to answer the following question: Using power capping, can we reduce the power consumed by an application with a limited impact on its energy consumption?

In previous work [4], we presented DUF, a tool which dynamically adapts the uncore frequency to the application needs. DUF outperforms the default uncore frequency scaling. In this work, we extend DUF in order to handle power capping dynamically. This new version is called DUFP (*Dynamic Uncore Frequency scaling and Power capping*) in the rest of this paper. Similarly to DUF, DUFP takes power capping decision according to a user-defined tolerated slowdown. Note that DUFP uses the same algorithm as DUF when it comes to uncore frequency. In other words, DUFP handles both uncore frequency scaling and power capping. Note that we did not study power capping by itself, since as already mentioned, uncore frequency scaling is better handled with DUF than with the default uncore frequency scaling [4].

The main goal of DUFP is to save power without degrading energy consumption. In other words, the goal of DUFP is not to save energy (since lowering the power cap will necessarily impact performance), but rather to save power without energy loss. We implemented and tested DUFP on 10 applications with 4 different slowdown-tolerance. We show that for all applications, combining dynamic power capping to uncore frequency scaling (i) improves the power consumption by up to 13.98 % with additional energy savings for applications where uncore frequency scaling has a limited impact and (ii) provides at most 7.90 % more savings than uncore frequency scaling by itself. Moreover, for most applications, a tolerated slowdown of 10 % allows for power savings with no energy loss.

The paper is organized as follows: Section II introduces a motivation to dynamic power capping before presenting DUF and some background on power capping. Section III presents DUFP. The experimental setup is described in Section IV while Section V presents DUFP evaluation.

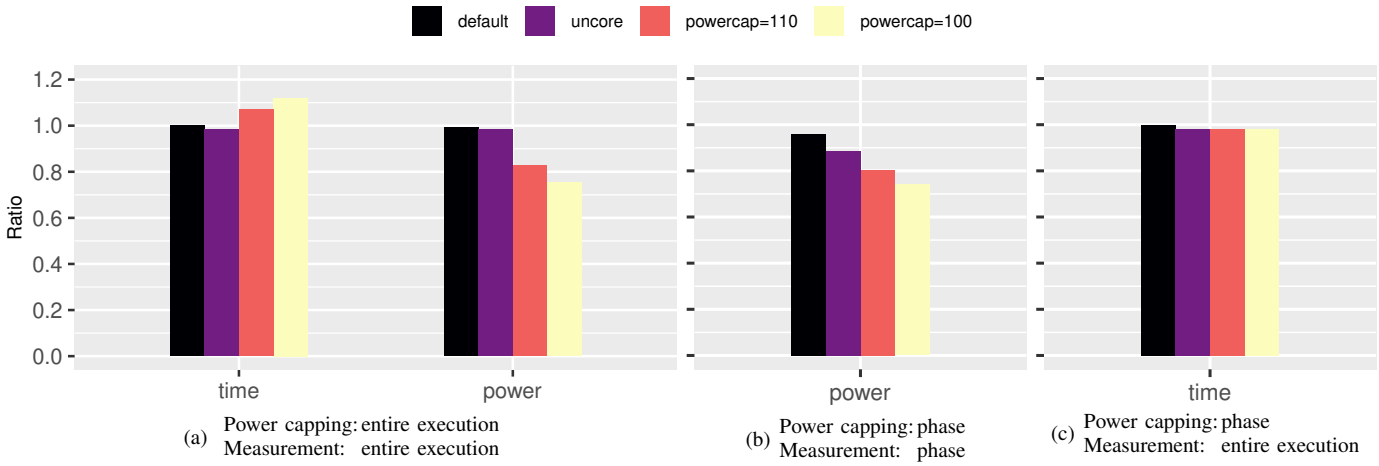

Fig. 1: Power capping on CG on four Intel Xeon Gold 6130 CPUs. Figure 1a represents the behavior throughout the execution. On Figure 1b, only one phase of the application is measured while Figure 1c shows the total execution time with partial power capping. The ratios are computed over the default values.

II. Background and motivation

This section presents a motivating example and introduces the basic principle behind power capping.

A. Motivating example

In order to understand the need for dynamic power capping, we conducted two set of experiments on the application CG from the NAS Parallel Benchmarks. More details on the architecture and the applications are provided in Sections IV-A and IV-B. In the first experiment, presented in Figure 1a, we ran CG in the default architecture configuration, with uncore frequency scaling and with uncore frequency scaling when applying power capping. The power cap was set before starting the application and was not modified until the execution completed. We studied the behavior of the application under two different power caps: 110 W and 100 W. For each run, we measured the execution time and the processor power consumption (as shown in Figure 1a). The default execution time represents the execution time of the application in the default architecture configuration while the power consumption is computed over the default power budget allocated to the architecture (125 W in this case). In other words, the power ratio is computed over the budget allocated to the processor rather than the power consumed by the application in the default configuration. The results show that, uncore frequency scaling provides limited power savings. Furthermore, combining uncore frequency scaling with a power cap of 110 W improves the power savings by 16 % while the savings reach 24 % with a power cap of 100 W. However, these savings come with a high impact on performance. As a matter of fact, the overhead reaches 7.15 % with a power cap of 110 W and 12 % with a power cap of 100 W. As a consequence, applying a power cap throughout the execution of an application while being oblivious to the application characteristics may lead to an uncontrolled overhead.

We then wanted to study if a better knowledge of the application may help the impact on performance. Therefore, we looked closely at CG. In the beginning, the application mainly performs memory accesses for several seconds, with very few computations. We applied the same power caps as described earlier (100 W and 110 W) but only on this part of the code. Uncore frequency scaling was also used under these power caps. After this phase completed, we just reset the power cap to the default value. Note that this phase accounts for 5 % of the total execution time of CG. Figure 1b shows the results obtained regarding the power consumption of the phase being modified. Recall that the ratios are computed over the budget allocated to the processor. This experiment shows that, on the one hand, applying a power cap for even a fraction of an application execution can have a large impact on power consumption. As a matter of fact, the power consumption of the studied phase is reduced by 16 and 19 % with a power cap of 110 W and 100 W respectively and is better than using uncore frequency scaling by itself. Moreover, Figure 1c shows the impact of the power capping on the first phase of the application, on the total execution time (not just the first phase). It shows that reducing the power budget on the first phase of CG does not impact at all its overall execution time. As a consequence, reducing the power budget for phases where mainly memory accesses are performed has no impact on performance. On the other hand, under the default configuration, the power consumption is almost at the maximum processor budget. This indicates that reducing the power budget according to the application needs has a positive impact on power consumption with no impact on performance. Note that we studied the impact of applying power capping on only 5 % of the application. As a consequence, the overall power savings are minimal. However the goal of this experiment was to show the benefits of dynamic power capping combined to uncore frequency scaling while considering the application characteristics.

B. Power capping

In this work, we focus on Intel architectures. By default, the processor is designed to respect TDP (Thermal Design Power). Thermal Design Power is defined as the maximum amount of power that can be dissipated by the processor cooling systems [2]. In other words, as long as the power consumption is under TDP, Intel guarantees that the chip will operate as intended. Note that TDP is not the maximum power a processor can consume [2].

A similar idea is provided by RAPL (Running Average Power Limit) regarding power capping: the user can define a long-term constraint and a short-term constraint [15]. This means that when setting a power cap, the processor has to respect the long-term constraint for most of the time, but can consume more for a short time while respecting the short-term constraint. From our observation, by default, the long-term period is equal to the Thermal Design Power of the processor. Note that in order to respect the power cap, RAPL uses Dynamic Voltage and Frequency Scaling [15]. RAPL provides power capping capabilities for different components like the processor, the memory, ...However, memory power capping is not available on the processor that we used. As a consequence, we will no further address memory power capping.

C. DUF

In [4], we presented DUF, a tool that dynamically adapts the uncore frequency to the application needs. DUF takes a user-defined slowdown. Periodically, DUF monitors the FLOPS/s and the operational intensity. Every time the operational intensity indicates a phase change, the uncore frequency is reset. We assume that if the operational intensity is lower than 1, then the current phase is memory intensive, otherwise it is assumed to be CPU-intensive. If the FLOPS/s dropped by more than the tolerated slowdown, then the uncore frequency is increased. Otherwise, DUF keeps decreasing the uncore frequency (until the minimum uncore frequency is reached). We will not further detail DUF. Readers can refer to [4] for the full algorithm.

Compared to the default uncore frequency scaling, DUF manages to better adapt to the application needs: it improves the power consumption by up to 18.7 %. Moreover, DUF is able to improve performance under power capping.

III. Dynamic Power Capping Algorithm

This section presents DUFP algorithm. More specifically, it only describes how power capping is handled. Because the decisions are taken separately, we do not provide any detail on uncore frequency management. Note that DUFP and does not require any application modification. In order to use DUFP, the user specifies the tolerated slowdown and the processors (sockets) where DUFP must be applied. Then one instance of DUFP is started on each user-specified socket.

Figure 2 presents an overview of DUFP behavior. The basic principles are similar to those of DUF: Periodically, DUFP monitors the FLOPS/s and the memory bandwidth, and the operational intensity is computed. Whenever a new phase is detected, the power cap is reset. We consider a phase change as any important variation in the behavior of the applications (from CPU to memory intensive phase or the opposite or if the FLOPS/s double within the same phase). Note that CPU or memory intensiveness depends on the architecture. But in this work, we only consider the ratio between FLOPS/s and memory and ignore architecture characteristics. This is however discussed in Section V-G.

If no new behavior is detected, DUFP compares the current FLOPS/s to the maximum FLOPS/s observed in the current phase. If the FLOPS/s are still within the tolerated slowdown (compared to the maximum FLOPS/s), then the power cap is decreased. Moreover, for applications which are highly memory-intensive (empirically defined as phases with an operational intensity lower than 0.02, like the phase studied in Section II-A), power capping can be decreased with no impact on performance. Note that at every iteration, if the FLOPS/s are equivalent to the slowdown (with respect to the considered measurement error), the power cap is kept steady.

Otherwise, *i.e.* the FLOPS/s are below the tolerated slowdown, then the power cap is increased. For highly CPU intensive phases (empirically defined as phases where the operational intensity is larger than 100), if the FLOPS/s drop below the tolerated slowdown, the power cap is reset. The reason for this is we want to avoid any negative impact on performance. Furthermore, for these applications, we also monitor the memory bandwidth. As a matter of fact, we have observed that such applications may suffer an overhead because of the memory bandwidth drop. As a consequence, we assume that the slowdown also applies to memory bandwidth. In other words, for these applications, if the memory bandwidth drops by more than the tolerated slowdown, the power cap is reset. However, a deeper study and modeling of the impact of power capping on memory bandwidth is required for more accurate decisions.

Note that DUFP and DUF algorithms are similar since they both monitor the performance and decide whether the slowdown is still respected or not, however, each metric is handled separately. There are only two situations where power capping and uncore frequency scaling decisions impact one another:

1) Unlike power capping, uncore frequency scaling decisions monitor memory bandwidth for all phases (and not only highly compute-intensive phases). As a consequence, uncore and power capping decision may diverge. In this case, if the decision to increase uncore frequency did not improve performance (even if the FLOPS/s are still within the tolerated slowdown), the power cap is increased (to avoid any negative impact on performance).
2) When resetting both the uncore frequency and the power cap, the applied uncore frequency may be different from the maximum (because the power cap impact is still observed). As a consequence, whenever we reset both values, DUFP checks if the uncore frequency is at the maximum. If not, it tries to reset it once again.

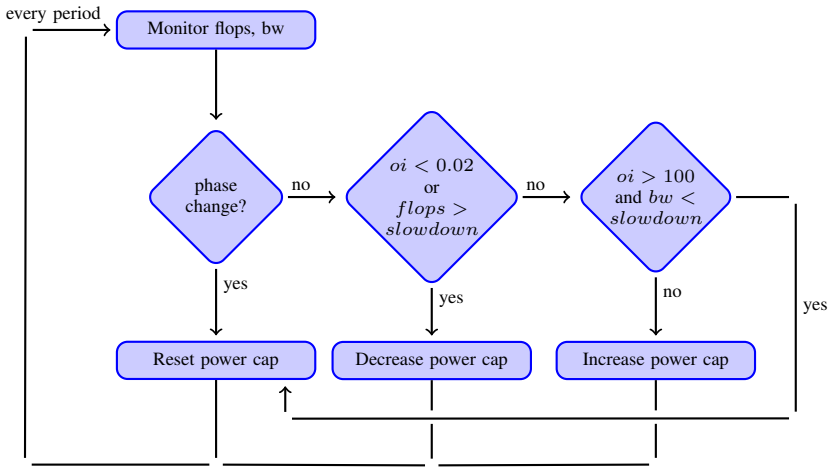

Fig. 2: DUFP basic algorithm: bw stands for memory bandwidth and oi stands for operational intensity (FLOPS/s/bw)

Finally, note that DUFP does not handle the long term and the short term constraints the same way. As soon as the power cap is decreased, both constraints are set to the same value. When the power cap is increased, if the value reached by the long term constraint is equal to its default value, the power cap is reset. Resetting the power cap will reset each constraint to its default value. Note that in the next iteration after resetting the power cap, DUFP checks the current power consumption. If it is lower than the power cap, we set the short term constraint to the same value as the long term constraint.

IV. Experimental setup

This section first presents the target platform and applications and the measurement framework before presenting DUFP configuration in Section IV-D.

A. Target platform

All experiments were performed on the Yeti cluster of Grid'5000 [7]. More specifically, all experiments were run on yeti-2. It is equipped with four Intel Xeon Gold 6130 CPUs (Skylake microarchitecture) with 16 cores per CPU. Each NUMA node has 64 GiB of memory. The uncore frequency ranges from 1.2 GHz to 2.4 GHz. The power cap long term constraint is 125 W while the short term constraint is at 150 W. It runs under Intel Pstate with performance governor. Note that we use power capping on the whole processor in this work. The architecture characteristics are summarized in Table I.

The uncore frequency step is set to 100 MHz while the power cap step is set to 5 W.

In our study, we use 65 W as a minimum value for power capping. This is because from our observations, only highly memory intensive applications can sustain low power caps (because they have few flops), but we have noticed that lower power cap values have an impact on memory bandwidth, which impacts memory-intensive applications.

B. Target applications

We conducted the experiments using several applications.

cores	uncore frequency (GHz)	long term (W)	short term (W)
64	[1.2-2.4]	125	150

TABLE I: Target architecture characteristics

- The NAS Parallel Benchmarks [6] provide a set of small applications. We use: BT, CG, EP, FT, LU, MG, SP, UA from NPB-3.3.1 OpenMP version. We choose the problem size so that each application execution time is in the [20s-400s] range. All benchmarks run using class D except SP for which we use class C. The OpenMP threads are bound to cores in a round-robin fashion.
- High-Performance Linpack (HPL) [20] is a software package that solves dense linear algebra systems. We use HPL version 2.3 compiled with Math Kernel Library (MKL) version 2019.1.144. HPL uses a configuration file where we set the problem size (N) to 91840, the (NB) to 224 and (PxQ) to (8x8).
- LAMMPS [21] [1] performs molecular dynamics simulation. We use input file `in.lj` provided for the `accelerate` suite where we set the run value to 100000.

On all platforms, the applications were compiled with gcc 6.3.0 with -O3 flag. The machines were running Linux version 4.9.0-9. HPL and LAMMPS were compiled against Open MPI 3.1.4. Finally, all 64 cores of yeti-2 were used during all the experiments while hyperthreading was disabled.

C. Measurement framework

Just like DUF, DUFP relies on PAPI [28] for power, FLOPS/s and bandwidth measurements. Uncore frequency is directly accessed and modified through the MSR registers. Finally, power capping is performed by using the power cap library[2]. Note that we are working on using PAPI for power capping as well, but this is still work in progress.

[1] commit `aa2b88578`

[2] https://github.com/powercap/powercap.git

D. DUF *and* DUFP *configurations*

Just like with DUF, we consider a 200 ms measurement interval. Shorter intervals lead to an overhead, while longer intervals lead to applying a power cap for longer intervals and having a larger impact on performance. As a consequence, we used 200 ms as a good trade off between overhead and accuracy. Note that we have observed that in some cases, some time is needed to apply a new power cap. As a consequence, the consumed power may be larger than the power cap. This happens on the iteration right after a decrease for instance (since we decrease both constraints at the same time). As a consequence, whenever this situation occurs, the power cap is reset.

V. Experiments

This section presents the evaluation of DUFP and its impact on performance (Section V-A), on processor power consumption (Section V-B), on DRAM power consumption (Section V-C) and finally on processor + memory energy consumption (Section V-D).

We performed 10 runs of each experiment. To mitigate outliers, we removed the lowest and highest execution times and returned the average over the remaining 8 executions. For each application, all the results are presented as a percentage over its default execution time and power and energy consumption. The default values are obtained when running the application under the default architecture configuration described in Section IV-A.

To quantify the stability of our measurements (over the 8 remaining runs), we present error bars showing the minimum and maximum observed values. The measurement difference is lower than 2 % for most of the configurations, while very few applications see a variation over 3 %. This indicates that our measurements are stable and accurate.

A. Impact on performance

Figure 3a shows the impact of DUFP on the execution time of the applications and how DUFP respects the tolerated slowdown. The results show that for most applications, DUFP manages to respect the tolerated slowdown. More specifically, DUFP manages to respect the tolerated slowdown for 34 out of the 40 configurations. For the 6 remaining configurations, the maximum slowdown is 3.17 %.

Compared to DUF, there are 5 configurations where DUFP does not respect the tolerated slowdown, naming: LAMMPS for all configurations except at 10 % slowdown, CG at 20 % slowdown and UA at 0 % slowdown. For LAMMPS, the observed overhead is 1.67 %, 0.14 % and 3.17 % at 0, 5 and 20 % tolerated slowdown respectively, while it reaches 1.17 % for UA and 0.4 % for CG. CG overhead is small and we could not explain why it occurs. The reason behind UA overhead lies in the behavior of the application itself. UA alternates between 1 compute bound iteration followed by several memory bound iterations. At every phase change, the power cap should be reset. However, since the memory-intensive phase lasts few iterations, the power cap is decreased. As a consequence, not all CPU-intensive iterations are detected because the low power cap has an impact on performance and does not allow the FLOPS/s to sufficiently increase to detect a phase change. As a consequence, to solve UA overhead, we should have a smaller monitoring period (which will however introduce an overhead due to both monitoring and uncore frequency and power capping changes). We are still investigating LAMMPS overhead. From our observations, at 20 % tolerated slowdown for instance, we do not observe an impact on performance (with respect to the tolerated slowdown) before DUFP sets low power caps. In order to understand why there is no visible impact on performance while the global performance are impacted, we decreased the measurement interval to 50 ms (only measurements were performed, no power capping). We observed that the power consumption have some bursts sometimes that are missed with a 200 ms interval. As a consequence, just like UA, it would be preferable to reduce the measurement period to better catch the variations. Note however that applying DUF or DUFP with 50 ms interval induces a larger overhead (as already discussed in Section IV-D).

Note that both DUFP and DUF introduce an equivalent overhead for LU. This is due to uncore frequency rather than power capping because the applied uncore frequency is similar throughout both executions.

Finally, DUFP manages to slow down some applications where DUF could not (while still respecting the tolerated slowdown). This is the case for BT, EP and UA. In the next section, we will show that this additional slowdown comes with additional power savings.

Overall, DUFP manages to respect the tolerated slowdown for 85 % of the configurations while staying within 3 % extra overhead in the 5 % remaining situations.

B. Impact on processor power consumption

Figure 3b shows the impact of combining power capping and uncore frequency scaling on the processor power consumption. In all configurations, DUFP manages to provide power savings. The best savings are reached for EP with 24.27 %. Note that for EP, uncore frequency scaling has the larger impact on power consumption compared to power capping.

Comparing the impact of power capping (DUFP compared to DUF) one can see that power capping provides additional power savings. The maximum improvement is observed with CG at 20 % tolerated slowdown where DUFP improves the power consumption by 7.90 % (9.66 % power savings with DUF and 17.57 % for DUFP). Moreover, with a 10 % tolerated slowdown, the power savings with FT almost double with DUFP compared to DUF.

Finally, because DUFP is able to introduce additional slowdown (still within the tolerated slowdown), it manages to provide power savings when DUF could not. This is the case for BT where DUFP provides 5.14 % power savings for 20 % slowdown while DUF manages only to save 0.64 % of power.

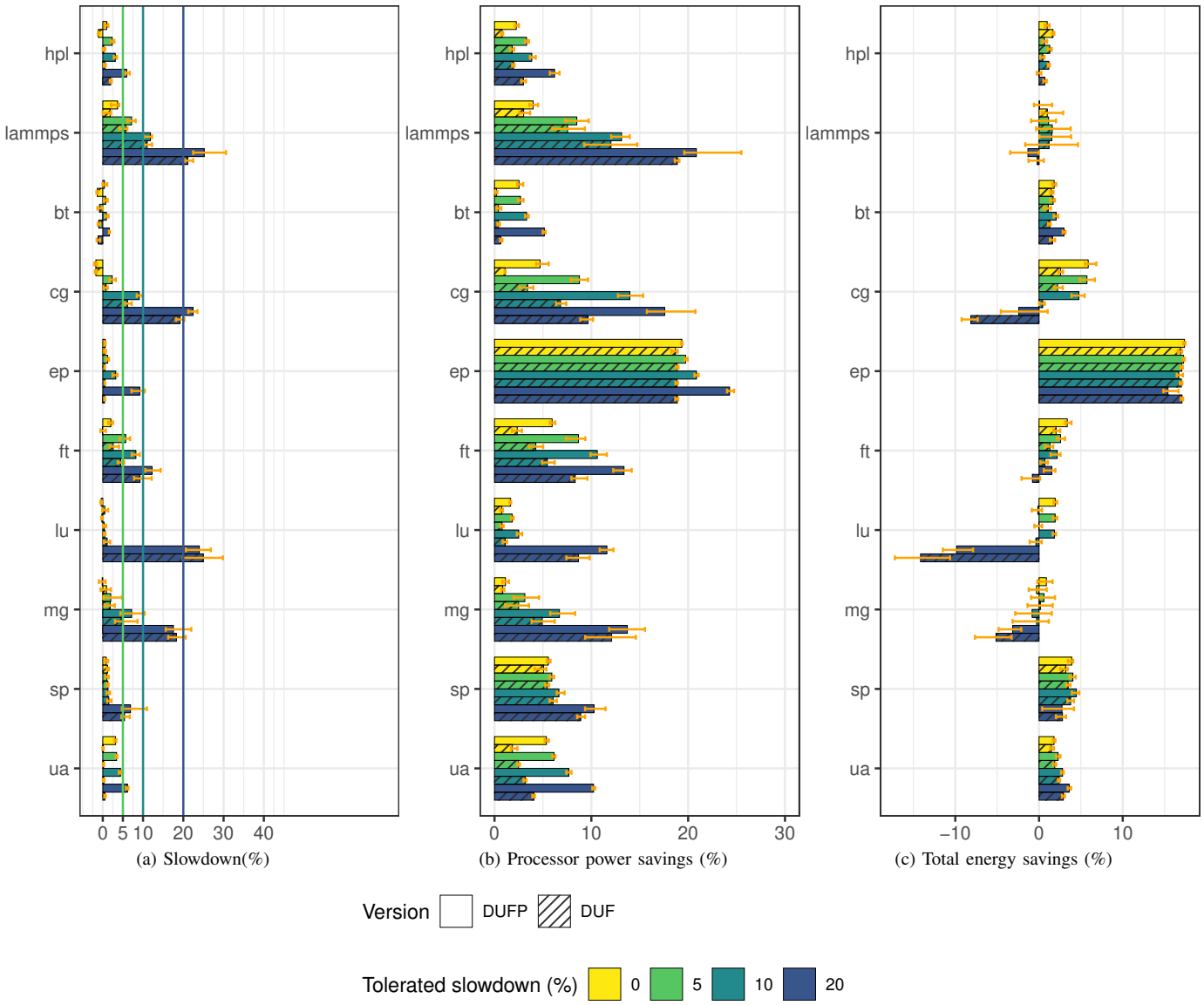

Fig. 3: DUF impact on performance, power and energy consumption on YETI

As a conclusion, setting aside EP, with less than 2 % slowdown, DUFP manages to save up to 6.65 % power. With less than 5 % slowdown, it manages to reach 8.76 % power savings.

C. Impact on memory power consumption

Figure 4 shows the impact of DUFP on memory power consumption. Just like for processor power consumption, DUFP manages to reach power savings for most configurations. Considering only applications where DUFP respects the tolerated slowdown, the best savings are reached for CG with 8.83 % savings with 20 % slowdown. Power loss is only observed with MG with 0 % tolerated slowdown (with 0.81 % power loss).

Compared to DUF, DUFP manages to outperform DUF for most configurations except for MG at 20 % tolerated slowdown (where DUF outperforms DUFP with 0.78 % improvement). Moreover, DUFP is also able to provide memory power savings (or at least no power consumption increase) when DUF fails to. This is the case for BT and UA. In this case, the best savings are for UA with 20 % slowdown (3.23 %).

D. Impact on total energy consumption

As previously stated, the goal of this work is not to improve energy savings, but rather to reduce the power consumption without impact on energy consumption. Figure 3c shows the impact of DRAM + processor energy consumption. The results show that overall, DUFP (i) avoids energy loss on almost all applications, (ii) provides energy savings for some applications and (iii) provides better savings than uncore frequency scaling by itself.

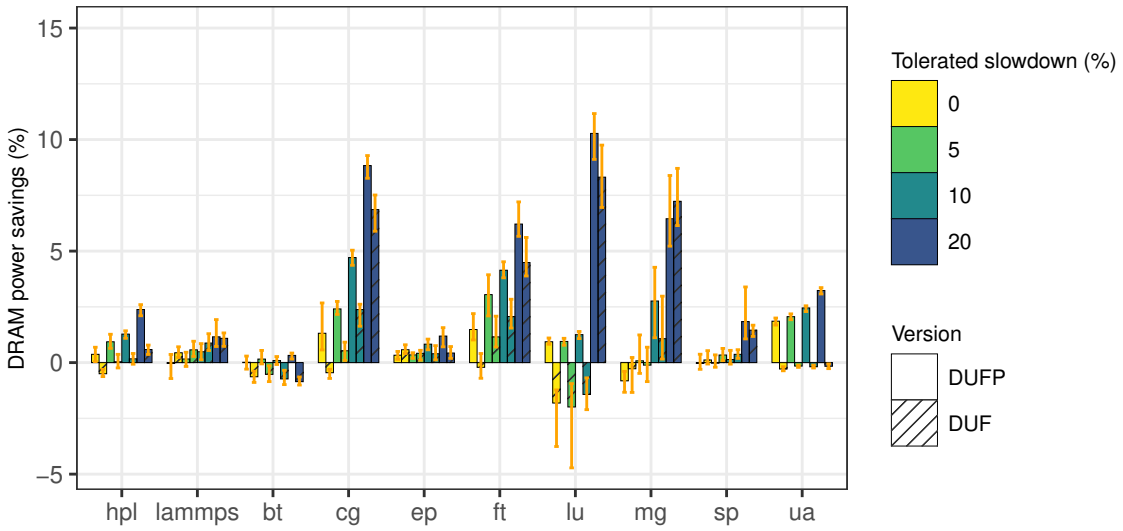

Fig. 4: Impact of DUFP on DRAM power consumption

Energy loss occurs at 20 % tolerated slowdown. This is the case for LAMMPS, CG, LU and MG and MG at 10 %. For these configurations, the power savings are not enough to cover the performance loss. Moreover, for all applications except UA and BT, it is not beneficial to tolerate 20 % slowdown. And even for BT and UA, the energy savings at 20 % tolerated slowdown are close to those at 10 % slowdown (0.98 % improvement for BT and 0.78 % for UA). Besides, for all applications except MG, a 10 % tolerated slowdown leads to no energy loss with the best power savings. For instance, for CG, with a 10 % tolerated slowdown, DUFP manages to save both power and energy (13.98 % processor power savings and 4.7 % total energy savings).

Finally, DUFP leads to more energy consumption than DUF for EP with 20 % tolerated slowdown and for HPL with all configurations. However, even in the worst case, DUFP still provides no or small energy savings, but no energy loss.

Overall, DUFP manages to provide equivalent to better energy consumption compared to DUF for most applications.

E. Impact of power capping on CPU frequency

In order to understand the power savings provided by DUFP, Figure 5 shows the measured frequency while using DUF and DUFP for CG with a 10 % tolerated slowdown. Note that only the frequency of the core 0 is presented, but all cores have equivalent behaviors in this case.

The comparison shows that, using uncore frequency by itself, the CPU frequency is at the maximum for the majority of the execution, whereas power capping enables frequency reduction which provides power savings. As a matter of fact, the average observed frequency with DUFP is 2.5 GHz while it is 2.8 GHz with DUF.

F. Discussion on the impact of applications characteristics

Characterizing how much an application will be impacted by power capping (both in terms of performance and power consumption) is not straightforward.

On the one hand, although CPU-intensive applications (like HPL or BT) show processor power savings, they remain below 7 % while the memory power savings are negligible. This is because CPU-intensive applications are very sensitive to CPU frequency, and power capping impacts frequency.

On the other hand, highly-memory intensive applications (or at least phases of the application) are not impacted by power capping. This was already stated in Section II-A. From our observations, the power cap can be set to low values with no impact. As a matter of fact, for such phases, the power cap was set to 65 W (which was the minimal value that we defined). This behavior was observed on both CG and FT.

For the remaining applications, it is not easy to draw any characteristic without further studying the application behavior. This is due to the application behavior in terms of FLOPS/s which drives DUFP decisions.

G. Limitations and possible improvements

Although DUFP manages to achieve its goal of saving power with no performance loss, we have identified some limitations that can be improved.

Just like DUF, DUFP assumes that the slowdown can be used to control the memory bandwidth drop the same way as the FLOPS/s (as shown in Figure 2). This is an approximation and modeling the impact of power capping on memory bandwidth would improve the algorithm behavior. We could also match the operational intensity to the architecture and prevent DUFP from detecting a new behavior while it is actually not.

The second improvement is related to applications where the FLOPS/s at each iteration are within the tolerated slowdown, but the overall performance are over the tolerated slowdown (as shows with LAMMPS in Section V-A). We should investigate if other performance counters provide more insight on the application behavior and could help detect any slowdown.

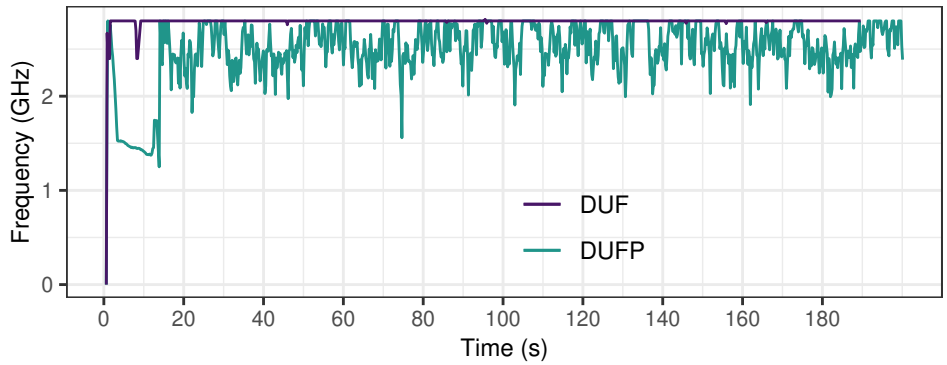

Fig. 5: DUFP vs DUF behavior regarding CPU frequency for CG at 10 % tolerated slowdown. When using all 16 cores of the processor, the maximum achieved frequency is 2.8 GHz for this processor.

Finally, as shown in Figure 5, power capping impacts CPU frequency. Therefore, better handling CPU frequency under power capping, instead of relying on power capping to change the CPU frequency may improve even more both performance and power consumption.

H. Conclusions

The goal of DUFP was to provide power savings without energy loss. The experiments show that for all applications, in the worst case scenario, DUFP manages to improve power consumption without increasing energy consumption, while in the best case, energy consumption is decreased as well. In other words, it is possible to find a tolerated slowdown configuration which reaches power savings with no energy loss: At 5 % tolerated slowdown, DUFP improves the power consumed of all applications while improving the energy consumption as well. For most applications, tolerating 10 % slowdown also allows for power savings with no increase on energy consumption.

The following conclusions can be drawn:

- DUFP manages to respect the tolerated slowdown for 85 % of the studied configurations.
- DUFP manages to reduce the power consumption of all applications.
- DUFP manages to save up to 6.65 % power with less than 2 % slowdown. With less than 5 % slowdown, it manages to reach 8.76 % power savings.
- For most applications, 0 % tolerated slowdown offers the best energy savings, while a 10 % performance slowdown leads to the best power savings with no energy loss.
- DUFP manages to provide power savings when DUF failed to.

VI. Related work

There are many studies which rely on power capping, however very few studies rely on dynamic power capping while fewer studies combine uncore frequency scaling to dynamic power capping. In [32] the authors propose to rely on reinforcement learning to get the best energy consumption with uncore frequency and power capping. Instruction Per Cycles (IPC) are used to control performance loss. This approach is complementary to DUFP which makes all decisions at runtime without any prior knowledge. Moreover, DUFP takes performance loss as a parameter (and thus can be configured according to the user needs).

Focusing on dynamic power capping with no uncore frequency scaling, DNPC is the closest to our work [25]. It is a library which dynamically adapts power capping to the application and uses a user-defined performance degradation limit. Based on measurements of the current period, DNPC estimates the performance degradation for the next period and decides to increase or decrease the power cap accordingly. One of the main differences between DUFP and DNPC is that the performance degradation model described in DNPC relies on frequency. In other words, the authors assume that there is a linear relation between CPU frequency and performance. This is not the case especially when targeting memory-intensive or vectorized applications. DUFP reads the flops to detect if there was a performance change. Note that we tried using DNPC, but the architecture that we used for our experiments is not yet supported.

CoPPer [16] is a tool that automatically adapts the power cap to reach user-specified performance: based on a model, CoPPer computes the power cap to apply to meet the performance. The performance are provided per application, by the user, as absolute performance (not percentage) depending on the architecture and the execution configuration (like the number of cores). One of the main differences between CoPPer and DUFP (in addition to uncore frequency scaling) is that CoPPer relies on application instrumentation to measure the performance and call CoPPer for decisions. Moreover, with DUFP, the user specifies how much slowdown he can accept, rather than performance the application must reach. As a consequence no knowledge of the application is needed by DUFP.

In [8], the authors use control theory to dynamically adapt the power cap to the application needs. They accurately model the impact of power capping on the performance of the STREAM benchmark. The authors noted that the model extends similarly to memory-intensive applications (or phases of applications) but it would need to be adapted for CPU-

intensive applications. Instead, DUFP is able to adapt to the application phases. Note that the model presented in [8] can be used by DUFP to adapt the power cap for memory-intensive phases.

Many studies rely on adapting the frequency or number of cores under a user-specified power cap. In [30] the authors propose OPAM, an operation-aware management strategy. The idea is to apply the best core and uncore frequency to limit performance loss under a user-specified power cap. Similar studies like [17] use a defined power capping and rely on frequency and voltage scaling to limit performance loss. As a consequence, these studies does not rely on dynamic power capping but rather on a static user-defined power cap. Similarly, in [10] the authors adapted the number of cores and the frequency to match a defined power cap. In [33], the authors study the resources configuration to set in order to respect a power cap (hyperthreading, number of cores, ...). As a consequence the power cap is not set: the resource configuration must respect the power cap. Finally, in [22] the authors propose a performance prediction model under power capping constraints. The power cap is set at different values during the execution and the model is evaluated. They evaluated how much a metric is impacted by power capping. As already mentioned, unlike DUFP, these studies do not target dynamic power capping.

Many studies focus on power budget distribution on several nodes through models [3], [5], [9], [18]. In these studies power capping is used to reduce the power consumed by a node and allocate it to another node. In [12] the authors provide heuristic on the power budget to allocate to different domains (CPU, memory, GPU). In [31], the authors describe DAPS, a strategy to allocate power among nodes on a cluster. Finally, GEOPM [11] is a job-level energy manager which assigns a power cap for the jobs. It is also able to adjust CPU frequency to the application phases. These studies are complementary to DUFP since they propose power budget allocation strategies across nodes while DUFP provides node-level dynamic power-capping.

VII. Conclusion and future work

This work tries to answer the following question: Using power capping, can we reduce the power consumed by an application with a limited impact on its energy consumption?

For that purpose, we extended a tool which uses dynamic uncore frequency scaling to also apply dynamic power capping. The results show that combining both techniques improves power consumption with no impact on energy consumption up to 10 % tolerated slowdown. As a conclusion, while uncore frequency scaling by itself shows power and energy savings with respect to the tolerated slowdown, power capping provides an additional leverage and manages to provide additional savings.

As a future work, as already mentioned, we plan to study if CPU frequency is properly managed under power capping and manage it with DUFP if not. Moreover, we could rely on learning techniques to get the best configuration depending on the applications and the target architectures.

Finally, we plan to target heterogeneous architectures: With a specified shared power budget to distribute over a CPU and a GPU, can we benefit from dynamic power capping to reduce the budget of the CPU when in does not need it and increase the GPU power budget? In that context, we could consider two different applications, one running on the CPU and another one on the GPU and try to match both applications needs in terms of performance and power needs.

Acknowledgment

This work was supported by the European High-Performance Computing Joint Undertaking EuroHPC under grant agreement No 955495 (MICROCARD) co-funded by the Horizon 2020 program of the European Union (EU), the French National Research Agency ANR, the German Federal Ministry of Education and Research, the Italian ministry of economic development, the Swiss State Secretariat for Education, Research and Innovation, the Austrian Research Promotion Agency FFG, and the Research Council of Norway.

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] Exascale computing project. https://exascale.llnl.gov/.

[2] Intel® xeon® processor e5 v4 product family. thermal mechanical specification and design guide, 2018.

[3] Kishwar Ahmed, Samia Tasnim, and Kazutomo Yoshii. Energy-efficient heterogeneous computing of parallel applications via power capping. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1237–1242, 2020.

[4] Étienne André, Rémi Dulong, Amina Guermouche, and François Trahay. duf: Dynamic uncore frequency scaling to reduce power consumption. *Concurrency and Computation: Practice and Experience*, n/a(n/a):e6580.

[5] Reza Azimi, Chao Jing, and Sherief Reda. Powercoord: Power capping coordination for multi-cpu/gpu servers using reinforcement learning. *Sustainable Computing: Informatics and Systems*, 28:100412, 2020.

[6] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga. The nas parallel benchmarks. *International Journal of Supercomputing Applications*, 5(3):63–73, September 1991.

[7] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, et al. Adding virtualization capabilities to the Grid'5000 testbed. *International Conference on Cloud Computing and Services Science*, pages 3–20, 2012.

[8] Sophie Cerf, Raphaël Bleuse, Valentin Reis, Swann Perarnau, and Eric Rutten. Sustaining Performance While Reducing Energy Consumption: A Control Theory Approach. In *EURO-PAR 2021 - 27th International European Conference on Parallel and Distributed Computing*, volume 12820 of *Euro-Par*, pages 334–349, Lisbon, Portugal, August 2021. Springer. The datasets and code generated and analyzed during the current study are available in the Figshare repository.v2 fix erroneous metadata.

[9] Tomasz Ciesielczyk, Alberto Cabrera, Ariel Oleksiak, Wojciech Piatek, Grzegorz Waligóra, Francisco Almeida, and Vicente Blanco. An approach to reduce energy consumption and performance losses on heterogeneous servers using power capping. *Journal of Scheduling*, 05 2020.

[10] Stefano Conoci, Pierangelo Di Sanzo, Alessandro Pellegrini, Bruno Ciciani, and Francesco Quaglia. On power capping and performance optimization of multithreaded applications. *Concurrency and Computation: Practice and Experience*, 33(13):e6205, 2021.

[11] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Brad Geltz, Federico Ardanaz, Asma Al-Rawi, Kelly Livingston, Fuat Keceli, Matthias Maiterth, and Siddhartha Jana. Global extensible open power manager: A vehicle for hpc community collaboration on co-designed energy management solutions. In Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes, editors, *High Performance Computing*, pages 394–412, Cham, 2017. Springer International Publishing.

[12] Rong Ge, Xizhou Feng, Yangyang He, and Pengfei Zou. The case for cross-component power coordination on power bounded systems. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 516–525, 2016.

[13] Neha Gholkar, Frank Mueller, and Barry Rountree. Uncore power scavenger: a runtime for uncore power conservation on HPC systems. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 27:1–27:23, 2019.

[14] A. Gocht, R. Schöne, and M. Bielert. Q-learning inspired self-tuning for energy efficiency in hpc. *International Conference on High Performance Computing Simulation (HPCS)*, pages 344–347, 2019.

[15] Azzam Haidar, Heike Jagode, Phil Vaccaro, Asim YarKhan, Stanimire Tomov, and Jack Dongarra. Investigating power capping toward energy-efficient scientific applications. *Concurrency and Computation: Practice and Experience*, 31(6):e4485, 2019. e4485 cpe.4485.

[16] Connor Imes, Huazhe Zhang, Kevin Zhao, and Henry Hoffmann. Copper: Soft real-time application performance using hardware power capping. In *2019 IEEE International Conference on Autonomic Computing (ICAC)*, pages 31–41, 2019.

[17] Panos Koutsovasilis, Christos Antonopoulos, Nikolaos Bellas, Spyros Lalis, George Papadimitriou, Athanasios Chatzidimitriou, and Dimitris Gizopoulos. The impact of cpu voltage margins on power-constrained execution. *IEEE Transactions on Sustainable Computing*, pages 1–1, 2020.

[18] Nirmal Kumbhare, Ali Akoglu, Aniruddha Marathe, Salim Hariri, and Ghaleb Abdulla. Dynamic power management for value-oriented schedulers in power-constrained hpc system. *Parallel Comput.*, 99:102686, 2020.

[19] Santiago Pagani, P. D. Sai Manoj, Axel Jantsch, and Jörg Henkel. Machine learning for power, energy, and thermal management on multicore processors: A survey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(1):101–116, 2020.

[20] Antoine Petitet, R C. Whaley, Jack Dongarra, and A Cleary. Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. *http://www.netlib.org/benchmark/hpl*, September 2000.

[21] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.

[22] Srinivasan Ramesh, Swann Perarnau, Sridutt Bhalachandra, Allen D. Malony, and Pete Beckman. Understanding the impact of dynamic power capping on application progress. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 793–804, 2019.

[23] Barry Rountree, David K. Lowenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler K. Bletsch. Adagio: making DVS practical for complex HPC applications. In *Proceedings of the 23rd international conference on Supercomputing, 2009, Yorktown Heights, NY, USA, June 8-12, 2009*, pages 460–469, 2009.

[24] J. Schuchart, M. Gerndt, P. G. Kjeldsberg, M. Lysaght, D. Horák, L. Říha, A. Gocht, M. Sourouri, M. Kumaraswamy, A. Chowdhury, M. Jahre, K. Diethelm, O. Bouizi, U. S. Mian, J. Kružík, R. Sojka, M. Beseda, V. Kannan, Z. Bendifallah, D. Hackenberg, and W. E. Nagel. The readex formalism for automatic tuning for energy efficiency. *Computing*, 2017. doi: 10.1007/s00607-016-0532-7.

[25] Sahil Sharma, Zhiling Lan, Xingfu Wu, and Valerie Taylor. A dynamic power capping library for hpc applications. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 797–798, 2021.

[26] Vaibhav Sundriyal, Masha Sosonkina, Bryce Westheimer, and Mark Gordon. Core and uncore joint frequency scaling strategy. *Journal of Computer and Communications*, 06:184–201, 01 2018.

[27] Vaibhav Sundriyal, Masha Sosonkina, Bryce M. Westheimer, and Mark Gordon. Comparisons of core and uncore frequency scaling modes in quantum chemistry application gamess. *High Performance Computing Symposium (HPC)*, pages 13:1–13:11, 2018.

[28] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with papi-c. In Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 157–173, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[29] TOP500 Supercomputer Site. http://www.top500.org.

[30] Bo Wang, Julian Miller, Christian Terboven, and Matthias Müller. Operation-aware power capping. In Maciej Malawski and Krzysztof Rzadca, editors, *Euro-Par 2020: Parallel Processing*, pages 68–82, Cham, 2020. Springer International Publishing.

[31] Bo Wang, Dirk Schmidl, Christian Terboven, and Matthias S. Müller. Dynamic application-aware power capping. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, E2SC'17, New York, NY, USA, 2017. Association for Computing Machinery.

[32] Yiming Wang, Weizhe Zhang, Meng Hao, and Zheng Wang. Online power management for multi-cores: A reinforcement learning based approach. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–1, 2021.

[33] Huazhe Zhang and Henry Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *SIGPLAN Not.*, 51(4):545–559, mar 2016.