DUF : Dynamic Uncore Frequency scaling to reduce power consumption

Etienne André, Rémi Dulong, Amina Guermouche, François Trahay

ARTICLE TYPE

DUF : Dynamic Uncore Frequency scaling to reduce power consumption

Étienne André[1] | Rémi Dulong[1,2] | Amina Guermouche[1] | François Trahay[1]

[1]Télécom SudParis, Institut Polytechnique de Paris, Evry, France
[2]University of Neuchâtel, Neuchâtel, Switzerland

Reducing the power consumption of applications has become one of the key challenges in high-performance computing. Recent processor architectures differentiate processor core frequency from its uncore frequency. As a consequence, in addition to tuning processor core frequency with Dynamic Voltage and Frequency Scaling (DVFS), power consumption can also be controlled through Uncore Frequency Scaling (UFS).

This paper studies how the uncore frequency can be used as a leverage to improve power consumption. We propose DUF, a daemon process that dynamically adapts the uncore frequency to reduce an application power consumption with a user-defined limit on performance degradation.

The evaluation of DUF on three different architectures shows that with no performance degradation (less than 0.6 %), DUF can reduce socket power consumption by 7.94 %. We also show that DUF is able to reduce the total energy consumption by up to 18.20 %.

KEYWORDS:
Green computing, Power consumption, Uncore frequency, High-Performance Computing, Powercapping

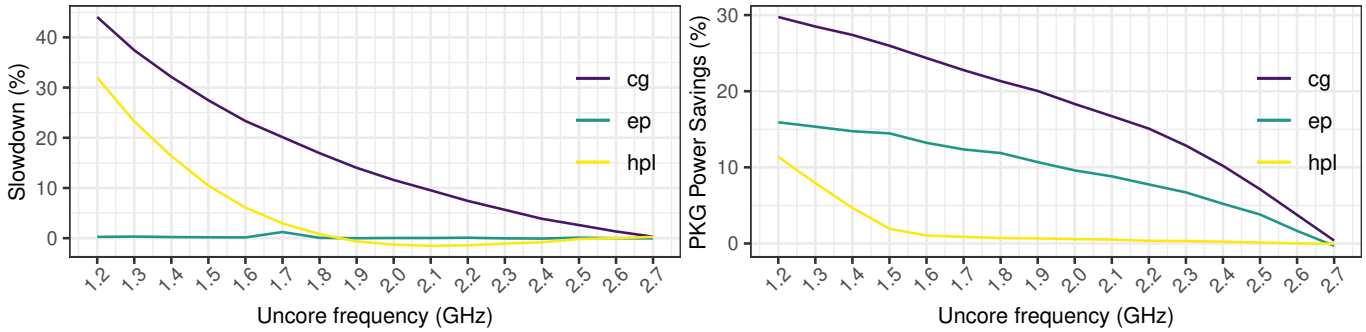
1 | INTRODUCTION

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 29.89 MW [1] while the US Department of Energy sets a limit of 20 MW for future exascale machines [2].

Dynamically adapting the processor frequency according to the application workload is a common technique to control power consumption. It is widely used in recent architectures where limiting the power consumption and respecting the thermal design power (TDP), while using the processor to its maximum capacity (number of cores, vectorized instructions, ...) requires to lower the CPU frequency, which may negatively impact performance.

Recent processor architectures differentiate the processor core frequency (that affects the computation units and the L1/L2 caches) from its uncore frequency (which affects the last level cache and the memory controller)[1]. The Uncore Frequency Scaling (UFS) automatically selects the uncore frequency according to the CPU frequency, the energy and performance bias hints and cores stall cycles[2]. However, it does not fully benefit from the leverage provided by the uncore frequency: Figure 1 shows the effects of varying the uncore frequency in terms of slowdown (figure 1a) and power savings (figure 1b) for NAS

[1]http://www.top500.org
[2]https://exascale.llnl.gov/



(a) Slowdown

(b) PKG power

FIGURE 1 Uncore frequency impact on execution time and package power consumption on a machine equipped with two Intel Xeon E5-2680 v4 CPUs.

Parallel Benchmarks EP and CG[3], and HPL[4]. These figures report the relative slowdown and power saving over the default values (obtained with UFS) on the CHIFFLET platform. Both the applications and the platform are described in Section 3.

Figure 1a shows that uncore frequency does not impact EP performance, while Figure 1b shows that setting the uncore frequency to 1.2 GHz reduces power consumption by more than 16 % compared to UFS. Note that the uncore frequency has no impact on EP because it has a very low memory and L3 bandwidth. Hence, it is possible to achieve the same performance as UFS with a lower power consumption.

Interestingly, regarding HPL, reducing the uncore frequency actually slightly improves its performance by 1.47%. Since HPL power consumption reaches TDP, the core frequency is automatically lowered which degrades the performance. Manually lowering the uncore frequency decreases the power consumption. As a consequence, core frequency increases leading to better performance.

For CG, if a small performance degradation is tolerated, reducing the uncore frequency significantly lowers the power consumption. For instance, if a 5% slowdown is tolerated for CG, the uncore frequency can be lowered to 2.3 GHz, which reduces the power consumption by 13 %.

As a consequence, one can conclude that the default UFS does not always adapt to the application characteristics (like EP and HPL). Moreover, by allowing more flexibility, it is possible to provide power savings with a controlled impact on performance.

Based on these observations, we propose DUF (that stands for Dynamic Uncore Frequency scaling), a daemon process that dynamically adapts the uncore frequency to the application needs. DUF aims at reducing an application power consumption with a user-defined limit on performance degradation. DUF can be seen as providing different uncore frequency governors (performance at 0% slowdown, powersave at 100% slowdown), in a similar fashion to what is done for DVFS. DUF aims at overcoming the default UFS drawbacks: it adapts the uncore frequency to the application needs and is able to tolarate a user-defined slowdown to improve the power consumption.

We tested DUF on 4 different user-defined slowdown and 11 applications and benchmarks on three different architectures. The results show that:

- On all platforms, DUF is able to respect the user-defined slowdown for 97.7 % of the tested configurations;
- DUF is able to reduce the power consumption of applications: (i) EP power consumption is reduced by up to 18.76% without altering its performance (ii) a 5.5 % slowdown on CG allows for 9.77 % power savings;
- When running under power capping constraints, DUF is able to improve the performance with a maximum of 10.53 % for CG;

The contributions of DUF over state of the art techniques such as UPSCAVENGER[5] are the following:

- we show that tolerating a limited performance degradation can yield significant energy savings ;
- our proposed Uncore Frequency Scaling algorithm respects the tolerated slowdown provided by user.

The remainder of this paper is organized as follows: We describe DUF in Section 2. Section 3 presents the measurement methodology we use in our experiments and the evaluation of DUF. Finally, we compare it to the related work in Section 4 before concluding in Section 5.

2 | DYNAMIC UNCORE FREQUENCY (DUF)

In this section, we describe DUF [3], a daemon process that dynamically adapts the uncore frequency in order to trade a limited performance degradation for power savings. The aim of DUF is twofold: reducing the power consumption of an application, and limiting the performance degradation to a user-provided upper-bound.

2.1 | Overview of DUF

The user of DUF specifies the sockets to monitor and a maximum performance degradation to tolerate. One instance of DUF is then run on each socket specified by the user. DUF periodically invokes its *measurement module* that collects the CPUs performance counters. Using collected data, the *regulator module* decides whether the uncore frequency should be changed. The decision algorithm described in section 2.3 applies for each user-specified socket. It can be summarized as follows: DUF detects the application phases (memory intensive vs compute intensive) based on the current operational intensity of the application and, for each phase, measures the performance obtained with the maximum uncore frequency. It then decreases the uncore frequency until the performance degradation reaches the user-specified limit.

2.2 | Measurement module

DUF *measurement module* collects various CPU hardware counters corresponding to the application FLOPS/s and the memory bandwidth in order to guide the *regulator module*. Then it computes the arithmetic intensity as the ratio between the FLOPS/s and the memory bandwidth. An arithmetic intensity greater than 1 indicates that the application is in a CPU-intensive phase. Otherwise, we assume that the application has entered a memory-intensive phase.

2.3 | Regulator module

In order to select the uncore frequency of a socket, DUF *regulator module* runs Algorithm 1 after every measurement period.

If a new application phase is detected, DUF sets the maximum uncore frequency and measures max_flops and max_bw (the achieved memory bandwidth) during the next measurement period. DUF assumes a phase change happened either because the application arithmetic intensity changed from CPU-intensive to memory intensive or the opposite (lines 4 to 9), or because the FLOPS/s and the memory or the L3 cache bandwidth increased significantly (lines 25-28).

Otherwise, DUF first checks how the uncore frequency could impact the performance. As a matter of fact, if the arithmetic intensity is too high, then the uncore frequency will most likely not impact the performance. As a consequence, it is decreased (lines 10-11). In the opposite way, if the arithmetic intensity is too low, then the uncore frequency should not be changed since it may have a high impact on memory bandwidth (lines 12-13). After that, as soon as the arithmetic intensity increases, the measurement period is increased in order to leave enough time for the application to reach a steady state before changing the uncore frequency (lines 14-15).

Otherwise, DUF checks how the previous decision impacted the FLOPS/s and the memory bandwidth. DUF considers that if the FLOPS/s dropped compared to the previous measurement, then three different situations may have happened:

- If the flops dropped despite increasing the frequency or keeping it steady, then either the drop is less than 50 % and the uncore frequency is kept steady. Or the drop is higher than 50 % then the frequency is decreased because we assume that this large drop is due to the application behavior itself (lines 17-20);
- If the flops and the memory dropped by the same ratio, then the uncore frequency is increased to make sure that the impact on memory bandwidth does not impact performance (lines 21-21);

[3] available as open-source at:
https://gitlab.com/parallel-and-distributed-systems/DUF

- If the memory bandwidth remained stable, then the drop comes from the behavior of the application itself rather than the impact of the uncore frequency. Based on this assumption, DUF decreases the uncore frequency (lines 16-24). Note that DUF considers the memory bandwidth as stable if it decreased by less than the tolerated slowdown. In other words, if the tolerated performance loss is 20 % then the bandwidth is considered as stable if it dropped by less than 80 %. This assumption is only based on our observations. Possible improvements of this assumption are further discussed in Section 3.7.

Finally, DUF decreases the uncore frequency as long as the performance remains within the user-specified threshold (lines 29-31) and increases it otherwise (lines 34-35). If a decrease is requested while the uncore frequency is at the minimum, DUF increases the measurement period as we reach a stable phase (line 33). The period is reset every time DUF changes the uncore frequency. DUF also increases the period if the requested uncore frequency is stable across iterations, indicating that a stable phase was reached. From our observations, doubling the period duration still allows to detect changes quickly enough to avoid an impact on performance. In all cases, we limit the measurement period to 10 times the initial period in order to avoid too long periods. However, for our experiments, since power measurements are reported by DUF, increasing the measurement period may impair the overall average power consumption. As a consequence, for all the experiments, except in section 3.6; the measurement period is never increased except in the case of lines 14-15. Note that when the application behavior changes from almost no computation ($oi < 0.02$) to more computations, the period is increased by x3 (line 15). This is because the application may need some additional time to reach the maximum FLOPS/s and bandwidth for the current phase. Otherwise, we may consider wrong maximum FLOPS/s and memory bandwidth values and take wrong decisions after that. From our observations, a x3 factor is enough.

3 | EXPERIMENTS

In this section, we evaluate if DUF meets its two objectives: saving power while limiting the performance degradation to a user-defined limit. We first provide the hardware settings of the experiment testbed. Then, we describe the different regulators that we study. We finally present the results of our experiments.

3.1 | Experiments testbeds

This section describes the architectures and applications that we used.

3.1.1 | Hardware settings

We used three servers from the Grid'5000[6] platform. All platforms run under Intel Pstate with performance governor. All platforms characteristics are summarized in Table 1.

- NOVA is a 23-nodes cluster, where each node is equipped with 2 Intel Xeon CPU E5-2620 v4 CPUs (Broadwell microarchitecture) with 8 cores per CPU and 64 GiB of memory. The uncore frequency ranges from 1.2 GHz to 2.7 GHz. We run our experiments nova-1.
- CHIFFLET is equipped with two Intel Xeon E5-2680 v4 CPUs (Broadwell microarchitecture) with 14 cores per CPU, and 768 GiB of memory. The uncore frequency ranges from 1.2 GHz to 2.7 GHz. We used chifflet-1 for our experiments.
- YETI 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. In all experiments, we used yeti-2.

3.1.2 | Software testbed

We conducted the experiments using several applications.

- The NAS Parallel Benchmarks[3] 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. On

Algorithm 1 Uncore Frequency Scaling algorithm

```
loop                                                                    ▷ Every period
    flops ← measure_flops()
    oi ← measure_operational_intensity
    if oi > 1 and phase! = CPU then
        phase ← CPU
        FREQ=RESET_UFREQ
    else if oi < 1 and phase! = memory then
        phase ← memory
        FREQ=RESET_UFREQ
    if oi > 100 then
        DECREASE_FREQUENCY
    else if oi < 0.02 then
        DO_NOTHING
    else if old_oi < 0.02 and oi >= 0.02 then
        period = 3 * default_period
    if flops < old_flops then
        if old_decision == INCREASE_FREQUENCY or old_decision == DO_NOTHING then
            if flops < 0.5 * old_flops then
                DECREASE_FREQUENCY
            else DO_NOTHING
        if bw/old_bw == flops/old_flops then
            INCREASE_FREQUENCY
        if bw/max_bw > 1 − perf_loss then
            DECREASE_FREQUENCY
    if flops > 2 * old_flops then
        if bw > 2 * old_bw or l3_bw > 2 * old_l3_bw then
            FREQ=RESET_UFREQ
        else DECREASE_FREQUENCY
    if flops > perf_loss * max_flops then
        if freq > min_freq then
            DECREASE_FREQUENCY
        else if period < 10 * default_period then
            period = period * 2
    else if freq < max_freq then
        INCREASE_FREQUENCY
```

NOVA, EP and MG were run using class D while on CHIFFLET, EP, MG, and FT run using the class D problem size. The remaining benchmarks run using class C. On YETI, 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)[4] 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 NB to 224 on all platforms. N is set to 58912 on NOVA, 62720 on CHIFFLET and 91840 on YETI. (PxQ) is set to (4x7) on CHIFFLET and (8x8) on YETI.
- LAMMPS[7] [4] performs molecular dynamics simulation. We use input file `in.lj` provided for the accelerate suite where we set the `run` value to 100000.

[4] commit `aa2b88578`

	NOVA	CHIFFLET	YETI
number of cores	16	28	64
microarchitecture	Broadwell	Broadwell	Skylake
TDP (W) per socket	85	120	125
uncore frequency (GHz)	[1.2-2.7]	[1.2-2.7]	[1.2-2.4]

TABLE 1 Platforms characteristics extracted from processors documentation

- Nwchem[8] [5] is a computational chemistry application. We use the input data set `3carbo.nw` from the qdm provided files.

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, LAMMPS and nwchem were compiled against Open MPI 3.1.4. Finally, all platforms cores were used during all the experiments (16 on NOVA, 28 on CHIFFLET and 64 on YETI) while hyperthreading was disabled.

3.1.3 | Measurement framework

In Section 1 and 3.2.1, we use LIKWID[9] [6] to set the uncore frequency and DUF measurement module to measure the power consumption of the applications. All the measurements are performed every 200 *ms*. Lower measurement periods lead to an overhead on some applications. On the other hand, periods such as 500 *ms* are too large for short running applications such as LAMMPS or CG on CHIFFLET. From our observations, 200 *ms* offers a good trade off for all the applications. Note that we discuss how DUF could automatically change its measurement period in Section 3.7.

In Section 3.3, all measurements (DUF and UPSCAVENGER require collecting hardware counters in addition to power) are performed using the PAPI library [10] [7]. Uncore frequency is modified and read by directly accessing the appropriate MSR registers.

3.2 | Description and configurations of UFS, UPSCAVENGER and DUF

This section briefly describes the default UFS behavior. It also describes UPSCAVENGER algorithm and the configurations used for DUF.

3.2.1 | Default behavior of Uncore Frequency Scaling

In order to understand the Uncore Frequency Scaling (UFS) default behavior on our experimental testbed, we measure the average uncore frequency when running applications with different profiles. For that purpose, we use two memory-intensive applications (CG and MG), and two CPU-intensive applications (EP and HPL).

	NOVA		CHIFFLET		YETI	
application	Power (W)	ufreq (GHz)	Power (W)	ufreq (GHz)	Power (W)	ufreq (GHz)
HPL	64	2.7	119.42	2.4	123.01	[1.6-1.7]
NPB CG	39.04	2.7	78.60	2.7	123.69	[2.2-2.4]
NPB EP	41.89	2.7	100.34	2.7	114.64	2.4
NPB MG	44.15	2.7	82.64	2.7	121.89	[2.1-2.3]

TABLE 2 Average observed power and uncore frequency on NOVA, CHIFFLET and YETI over all sockets with UFS.

Table 2 depicts the average uncore frequency range observed over the sockets. It also provides the average power consumed by the applications over all sockets.

[5] commit `67f5237ab`
[6] commit 267d
[7] git commit version `ceb64276`

On NOVA, all applications run at the maximum uncore frequency. This indicates that on NOVA, the uncore frequency is always set to the maximum. On CHIFFLET, CG, EP and MG run at the maximum uncore frequency (2.7 GHz). The uncore frequency for HPL is lower (2.4 GHz).We also observe that HPL reaches the thermal design power (TDP) of the machine (120 W). This behavior suggests that on CHIFFLET, the UFS policy first sets the uncore frequency to its maximum, and reduces it only when TDP is reached.

A similar behavior is observed on YETI: EP has a limited power consumption. Thus, the uncore frequency is set to the maximum (2.4 GHz). Meanwhile, since CG, MG, and HPL power consumption is closer to TDP, their uncore frequency is reduced.

3.2.2 | UPSCAVENGER

We compare DUF with UPSCAVENGER, a tool that regulates the uncore frequency. Since UPSCAVENGER source code is not available, we implemented our own version of UPSCAVENGER [8] based on the description in[5].

At every phase change, UPSCAVENGER updates the maximum DRAM power consumption to the one observed. Periodically, it: (i) decreases the uncore frequency if the DRAM power consumption is steady (ii) detects a phase change and resets the uncore frequency if the power consumption increases (iii) increases the uncore frequency if both the DRAM power consumption and the IPC decrease (iv) otherwise it detects a phase change and resets the uncore frequency.

Unlike DUF, UPSCAVENGER does not consider a tolerated slowdown. It aims at reducing applications power consumption without degrading their performance. It considers a 5% measurement error. As a consequence, DUF can be seen a generalization of UPSCAVENGER, where UPSCAVENGER should approximately stand between DUF with a 0 % and a 5 % slowdown tolerance.

3.2.3 | DUF configuration

In order to evaluate DUF, we use four different slowdown tolerances: DUF_0 (0 % tolerance), DUF_5 (5 % tolerance), DUF_{10} (10 % tolerance), and DUF_{20} (20 % tolerance).

DUF considers an error margin of 2 % regarding accuracy of measurements. Finally, we set DUF measurement period to 200 *ms* and the uncore frequency step to 100 MHz. Note that we use a 100 MHz step because we noticed that if we set the uncore frequency to 2.55 GHz for instance using LIKWID, the value that was read was 2.5 GHz. Moreover, as the experiments show, a 100 MHz allows for power savings without impact on performance (which would have indicated that a special behavior was missed). As a consequence, we did not further investigate LIKWID behavior.

Finally, as stated in Section 2, line 33 from Algorithm 1 is disabled for all experiments except in Section 3.6.

3.2.4 | Coping with UFS

The default UFS can only be disabled in the BIOS which we cannot access on Grid'5000. As a consequence, when running DUF, the default UFS runs as well. Therefore, a decision taken by DUF can be overwritten by the default UFS. Section 3.2.1 concluded that both CHIFFLET and YETI default UFS decreases the uncore frequency when TDP is reached. Thus, the frequency set can be lower than the one requested by DUF. From our observations, this behavior occurs only for applications that reach TDP. In order to handle this situation, at every iteration, both DUF and UPSCAVENGER use the uncore frequency that was set by UFS to compute the next frequency.

3.2.5 | Summary of experiments configurations

Each experiment shows the following configurations:

- DUF_0: DUF when considering no tolerated slowdown.
- DUF_5: DUF when considering 5 % tolerated slowdown.
- DUF_{10}: DUF when considering 10 % tolerated slowdown.

[8] our implementation is available as open source at https://gitlab.com/parallel-and-distributed-systems/DUF/tree/master/PowerScavenger

- DUF_{20}: DUF when considering 20 % tolerated slowdown.
- UPSCAVENGER.
- worstUfreq: The worst value obtained when manually setting the uncore frequency. In this case, there is no uncore frequency scaling (the uncore frequency does not vary).
- bestUfreq: The best value obtained when manually setting the uncore frequency. Just like worstUfreq, there is no uncore frequency scaling in this case as well.

3.3 | Experiments results

We run the applications described in Section 3.1.2 on NOVA, CHIFFLET and YETI while running the regulators. Figures 2a, 3a and 4a report the measured slowdown, Figures 2b, 3b and 4b show the socket power savings, and Figures 3c, 3c and 4c depict the socket + DRAM energy savings. Finally, Figure 6 reports the DRAM power savings on NOVA (Figure 6a), CHIFFLET (Figure 6b) and YETI (Figure 6c). Each experiment was run 10 times and we keep the average over the 8 runs between the minimum and maximum execution times. All the results are presented as a percentage over the default values (obtained with UFS) on each platform. All the data collected from the experiments of these study and presented hereafter are publicly available at: https://gitlab.inria.fr/aguermou1/uncore-duf.

On each figure, error bars are also shown. They show the minimum and maximum observed values. The measurement difference is lower than 1 % for most of the configurations, while very few applications see a variation over 2 %. This indicates the accuracy of the measurements. Note however that SP shows more variation on NOVA and YETI but also when applying powercapping. But a similar behavior is also observed on the default behavior or on the bestUfreq and worstUfreq plots.

3.4 | Impact on execution time

This section evaluates how DUF and UPSCAVENGER affect the application execution time, and if the slowdown respects: a user-defined limit for DUF with a 2 % measurement error, and a 5% measurement error for UPSCAVENGER.

Figures 2a, 3a and 4a show that, on all platforms, DUF remains within the tolerated slowdown for the majority of applications. Overall, DUF respects the user-defined limit for 128 of the 131 tested settings. Only three applications exceed the limit. CG on NOVA (2.37 %) and CHIFFLET (2.15 %) with DUF_0 and LU on YETI (22.11 %) with DUF_{20}. These overheads are however very small and we could not explain why they occur.

The figures also show that the behavior of some applications does not allow DUF to slow them down. For instance, BT on NOVA and YETI and nwchem on NOVA and CHIFFLET keep switching phases, while HPL on CHIFFLET and LU, SP and UA on YETI naturally see their FLOPS/s drop by more than the tolerated slowdown which leads DUF to increase the uncore frequency. EP also shows no slowdown at all. This is because uncore frequency has no impact on EP as stated in Figure 1.

The slowdown caused by UPSCAVENGER remains below the 5% measurement error for 20 out of the 32 tested settings. For the remaining applications (CG and MG on NOVA, CG, FT, MG and SP on CHIFFLET, and LAMMPS, nwchem, CG and MG on YETI), the slowdown is between 5 and 29.67%. Note that the slowdown of MG on NOVA, SP on CHIFFLET and MG on YETI is very close to 5 % (5.64, 5.81 and 5.32 respectively). LAMMPS and CG have the highest slowdown on YETI (29.67% for CG and 20.40% for LAMMPS). This is because UPSCAVENGER assumes that as long as the DRAM power consumption remains constant, the frequency can be decreased, regardless of the IPC and the L3 bandwidth. However, for some applications, (e.g. LAMMPS on YETI), both the FLOPS/s and the L3 bandwidth drop while the memory power consumption is slightly impacted. As a consequence, UPSCAVENGER keeps decreasing uncore frequency while DUF increases it. Regarding CG, the power memory consumption remains steady for several iterations. As a consequence, UPSCAVENGER decreases the uncore frequency during the steady phases. On the other hand, decreasing the uncore frequency has a direct impact on CG performance which allows DUF to stop decreasing.

As a conclusion, DUF manages to better respect the tolerated slowdown compared to UPSCAVENGER. As a matter of fact, in 97.7 % of the studied cases, DUF remains within the tolerated slowdown while the percentage drops to 62.5 % for UPSCAVENGER. Moreover, the slowdown goes as high as 29.67 % for CG on YETI. This indicates that memory power consumption is not the best indicator to lead uncore frequency decisions for some applications. From our observations, this is because DRAM power consumption may not be as sensitive to uncore frequency compared to memory and L3 bandwidth. As a consequence, memory and L3 bandwidth may be better indicator than DRAM power consumption.

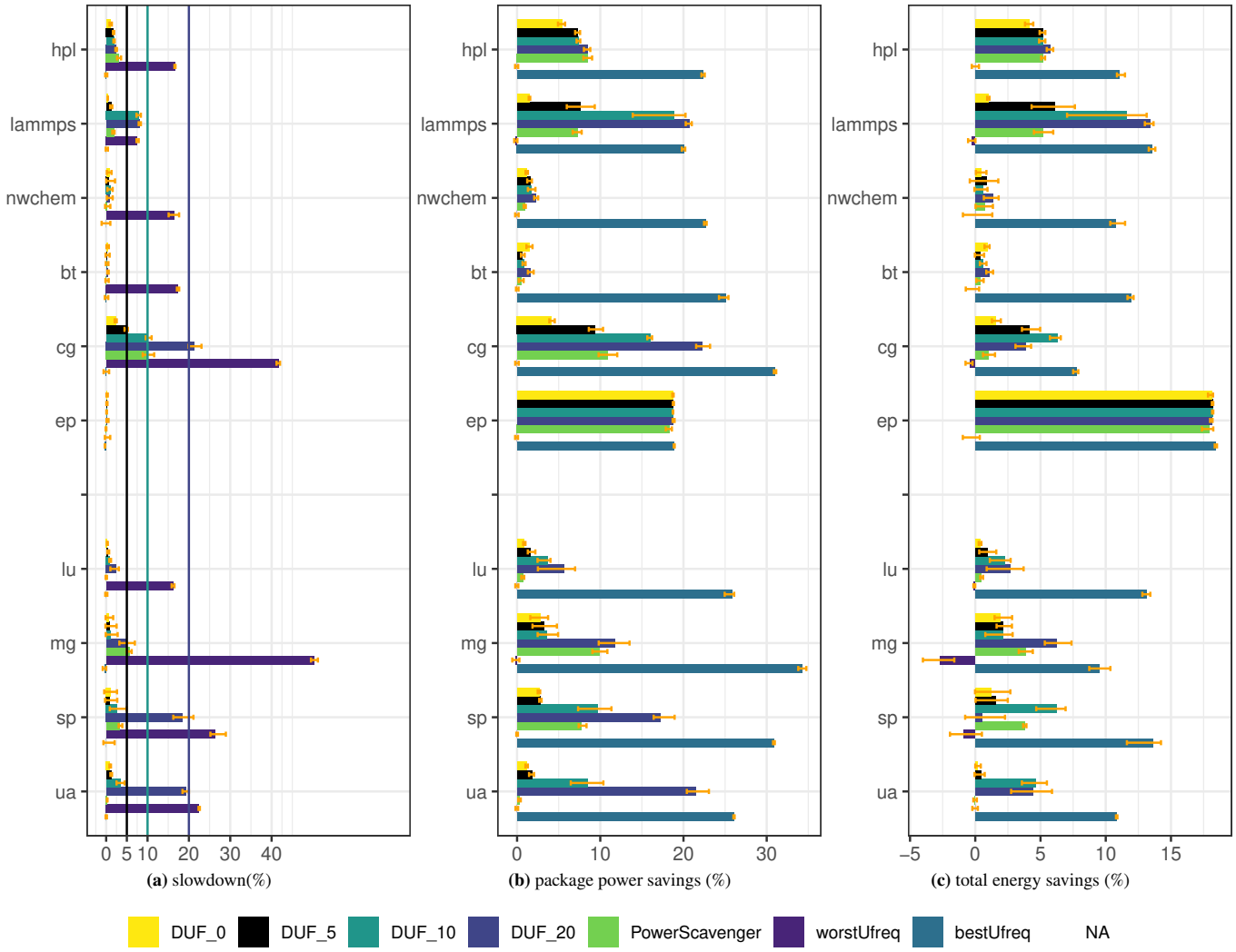


FIGURE 2 DUF impact on performance, power and energy consumption on NOVA

3.5 | Impact on power and energy consumption

This section evaluates how DUF and UPSCAVENGER reduce the power and energy consumption of the applications.

3.5.1 | Impact on socket power consumption

Figures 2b, 3b and 4b show the package power saving when using DUF and UPSCAVENGER on NOVA, CHIFFLET and YETI.

The figures show that for most applications, both DUF and UPSCAVENGER manage to provide power savings reaching up to 23.54 % for DUF. As expected, among DUF four configurations, DUF_{20} reaches the maximum power savings for most applications. For instance, DUF_{20} provides the best savings at 22.19 % with CG on NOVA.

EP has the exact same behavior for all regulators. As reported in figure 1, EP is not impacted by uncore frequency, it reaches 16.55 % power saving on CHIFFLET regardless of the regulator. For other applications, a small slowdown allows for power savings. For instance, with a slowdown of 0.58 %, DUF_5 manages to reach 7.94 % of power savings for BT on CHIFFLET. A similar behavior is observed with HPL where DUF_{10} manages to save 7.39 % of power while the slowdown reaches 1.88 % on NOVA. With a slightly higher slowdown, for UA, DUF_{10} shows 3.46 % performance degradation while providing 8.49 % power savings on NOVA. A similar behavior with MG on CHIFFLET is observed: with a slowdown of 3.51 %, DUF_{20} manages to reach 8.22 % power savings. On YETI, DUF_5 manages to provide 7.31 % power savings with 2.98 % slowdown for nwchem.

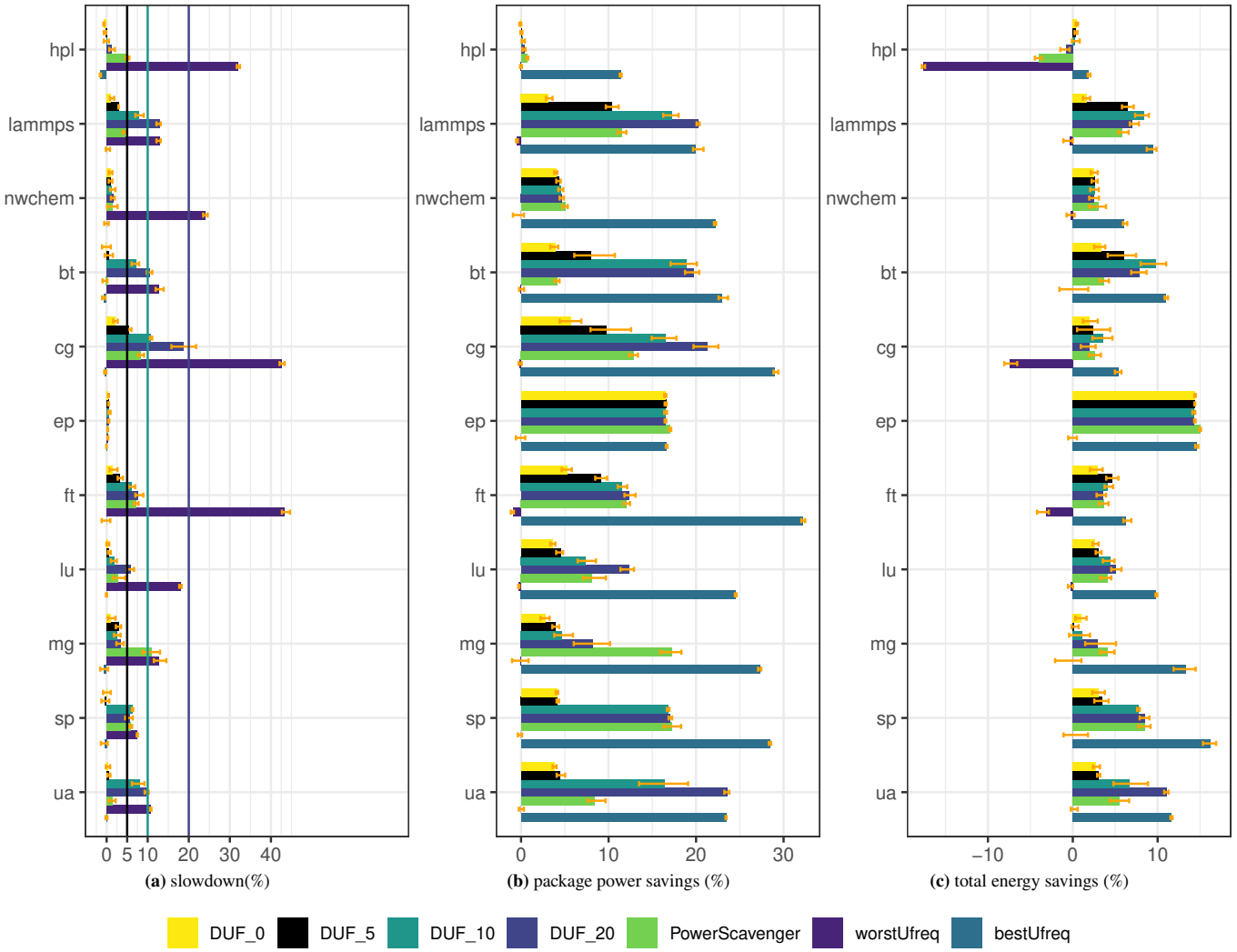


FIGURE 3 DUF impact on performance, power and energy consumption on CHIFFLET

Overall, in addition to EP, with less than 4 % slowdown, DUF manages to improve the power consumption of many applications (hpl, LAMMPS, SP and UA on NOVA, LAMMPS, BT, FT, LU and MG on CHIFFLET, nwchem on YETI) by 7.3 to 10.36 % for all three platforms.

For hpl, DUF_{20} causes a 1.17 % slowdown while only 0.36 % power savings are observed on CHIFFLET. This is because HPL power consumption and performance are roughly the same from 2.4GHz to 1.7 GHz as shown in Figure 1. However, as stated in Section 3.4, in HPL, the FLOPS/s decrease below the tolerated slowdown. Thus DUF does not manage to reach 1.6 GHz. UPSCAVENGER manages to reach lower frequencies but for very few iterations which is not enough to show an impact on power consumption. In addition to HPL, BT on NOVA and YETI show less than 2 % power savings for both tools. This is because of the behavior of the applications which keep switching phases. As a consequence neither DUF nor UPSCAVENGER can reduce the uncore frequency.

For the majority of the applications, the power savings obtained with UPSCAVENGER are similar to those of DUF with equivalent slowdown. For instance, on NOVA, HPL power savings with UPSCAVENGER and DUF_{20} are equivalent and both tools have the same slowdown. This is also the case for CG on CHIFFLET where UPSCAVENGER performance and power savings are between DUF_5 and DUF_{10}. This indicates that, for the same slowdown, DUF and UPSCAVENGER reach the same uncore frequency.

However, for some applications DUF provides better power savings than UPSCAVENGER while performing better. This is the case for LAMMPS, CG, MG and SP on NOVA. For instance, CG performance with UPSCAVENGER stand between DUF_{10}

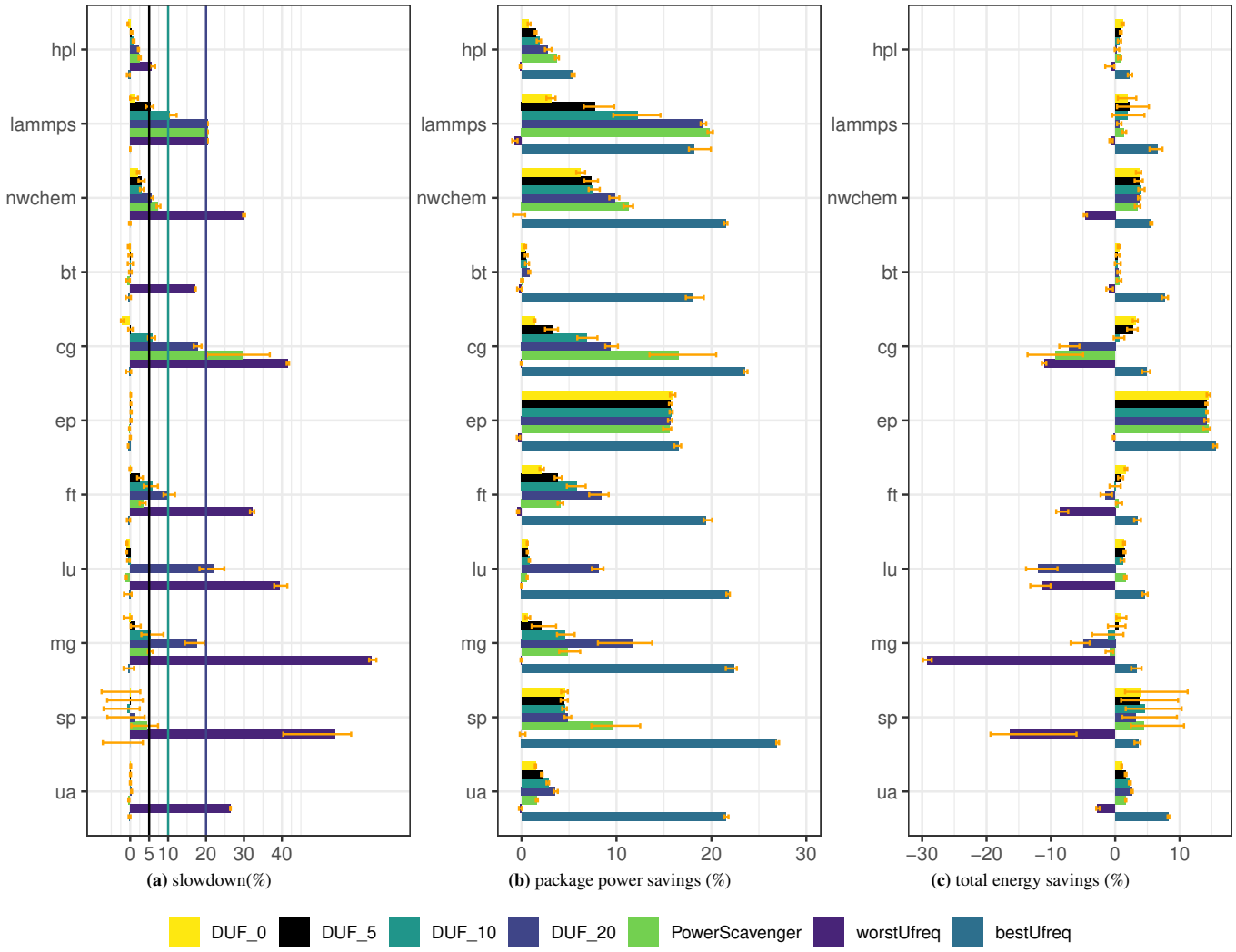


FIGURE 4 DUF impact on performance, power and energy consumption on YETI

and DUF_{20} while DUF_{10} provides better savings (16.01 % savings for DUF_{10} and 10.84 % savings with UPSCAVENGER). On the other hand, UPSCAVENGER provides better savings for nwchem and SP on CHIFFLET and LAMMPS and UA on YETI. For instance, with UPSCAVENGER and DUF_{20}, LAMMPS reaches equivalent slowdown but the power savings with UPSCAVENGER reach 19.83 % while they reach 19.13 % with DUF_{20}.

3.5.2 | Per-socket uncore frequency regulation

Since DUF handles each socket separately, the uncore frequency may not be the same on each socket. Figure 5 shows how the uncore frequency varies on each socket when running SP on CHIFFLET using DUF_{20}. It shows that, on socket 0, the frequency is most of the time at the minimum while it varies between 2.4 GHz and 2.7 GHz on socket 1. This is due to the fact that on socket 1, both the FLOPS/s and the memory bandwidth keep increasing then decreasing. Thus, DUF keeps decreasing and increasing the uncore frequency. On the other hand, on socket 0, the memory bandwidth is stable and DUF manages to reduce the uncore frequency. Note that as we decrease the frequency by 100 MHz at each iteration, DUF reaches the minimal frequency after 3.2 s (which represents 2.3 % of the total execution time). After that, the uncore frequency does not change anymore and the measurement period is increased.

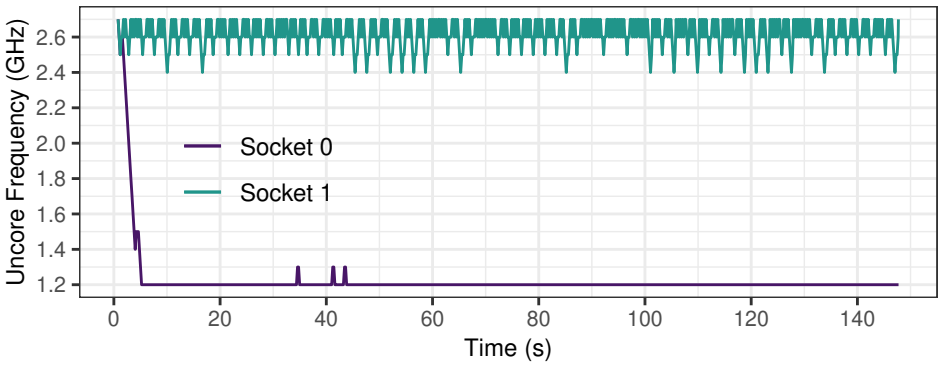


FIGURE 5 Uncore frequency during SP execution with DUF_{20} on CHIFFLET

3.5.3 | Impact on DRAM power consumption

Figures 6a, 6b and 6c show the impact of DUF and UPSCAVENGER on memory power consumption.

The results show that on most applications, both DUF and UPSCAVENGER manage to provide power savings. Just as the socket power consumption, the best savings are reached with DUF_{20} for most applications. For instance, CG on NOVA reaches 7.02 % power savings while LU reaches 9.09 % on YETI. On CHIFFLET, for most applications, the DRAM power consumption corresponds to the default DRAM power consumption ± 2.5 %. CG and UA show larger savings reaching 4.18 % and 3.21 % respectively.

Regarding UPSCAVENGER, for most applications, the behavior is similar to package power savings where DRAM power savings are equivalent to DUF for the same slowdown on NOVA, CHIFFLET and YETI. However, HPL and SP on NOVA and nwchem on YETI provide better savings with DUF compared to UPSCAVENGER. For instance, UPSCAVENGER and DUF_0 show equivalent DRAM power savings for HPL on NOVA while its slowdown reaches 0.97 % with DUF_0 and 3.08 % with UPSCAVENGER. On the other hand, UPSCAVENGER shows better savings than DUF for nwchem on NOVA and SP on CHIFFLET. As a matter of fact, nwchem slowdown reaches 0.29 % with UPSCAVENGER and 0.69 % with DUF_5 while the power savings with UPSCAVENGER reach 2.82 % and 1.19 % with DUF_5.

3.5.4 | DUF impact on energy consumption

Figures 2c, 3c and 4c show how DUF and UPSCAVENGER impact applications energy consumption. We consider both socket and DRAM power consumption when measuring the energy consumption.

On all platforms, both UPSCAVENGER and DUF allow for energy savings for most applications. DUF provides energy savings for all applications except for HPL on CHIFFLET and BT and MG YETI. Note that nwchem and BT on NOVA and HPL, FT and LU show very low energy savings (between 1 and 2 %) on YETI. The maximum savings reach 18.20 % on NOVA, 14.41 % on CHIFFLET and YETI for EP with all configurations.

On CHIFFLET, LU, MG, SP and UA show the best energy savings with DUF_{20}. For the other applications, limiting the overhead provides better energy savings. For instance, with a slowdown of 7.14 %, energy savings reach 9.76 % for BT. In addition to BT and EP, DUF manages to reach over 5 % energy savings for LAMMPS, LU, SP and UA.

On NOVA, HPL, LAMMPS, nwchem, BT, LU and MG reach their best energy savings with DUF_{20}, while CG, SP and UA provide better energy savings with DUF_{10}. Overall, in addition to EP, DUF manages to save more than 5 % energy for HPL, LAMMPS, CG, MG and SP.

On YETI, due to its significant slowdown, DUF_{20} leads to more energy consumption for CG, FT, LU and MG. For the other applications, the best savings are reached with DUF_0 or DUF_5 except for SP and UA where DUF_{10} and DUF_{20} reach the best savings. However, unlike on NOVA and CHIFFLET, DUF manages to save more than 5 % energy only for EP on YETI.

Overall, the power savings with DUF_0 and DUF_5 reach a maximum of 18.76 %. Regarding DUF_{10}, the maximum power savings reach 19.18 %. Finally with DUF_{20}, the maximum power savings reach 23.54 %. For all configurations, the best energy savings are reached with EP with a maximum of 18.20 %.

UPSCAVENGER provides better savings than DUF for nwchem and MG on CHIFFLET (3 % and 4.12 % respectively) and LU on YETI (1.62 %). For HPL, LAMMPS, CG, MG and SP on NOVA, LAMMPS and FT on CHIFFLET and nwchem and LU on YETI, DUF provides equivalent to better energy consumption compared to UPSCAVENGER despite similar or better performance.

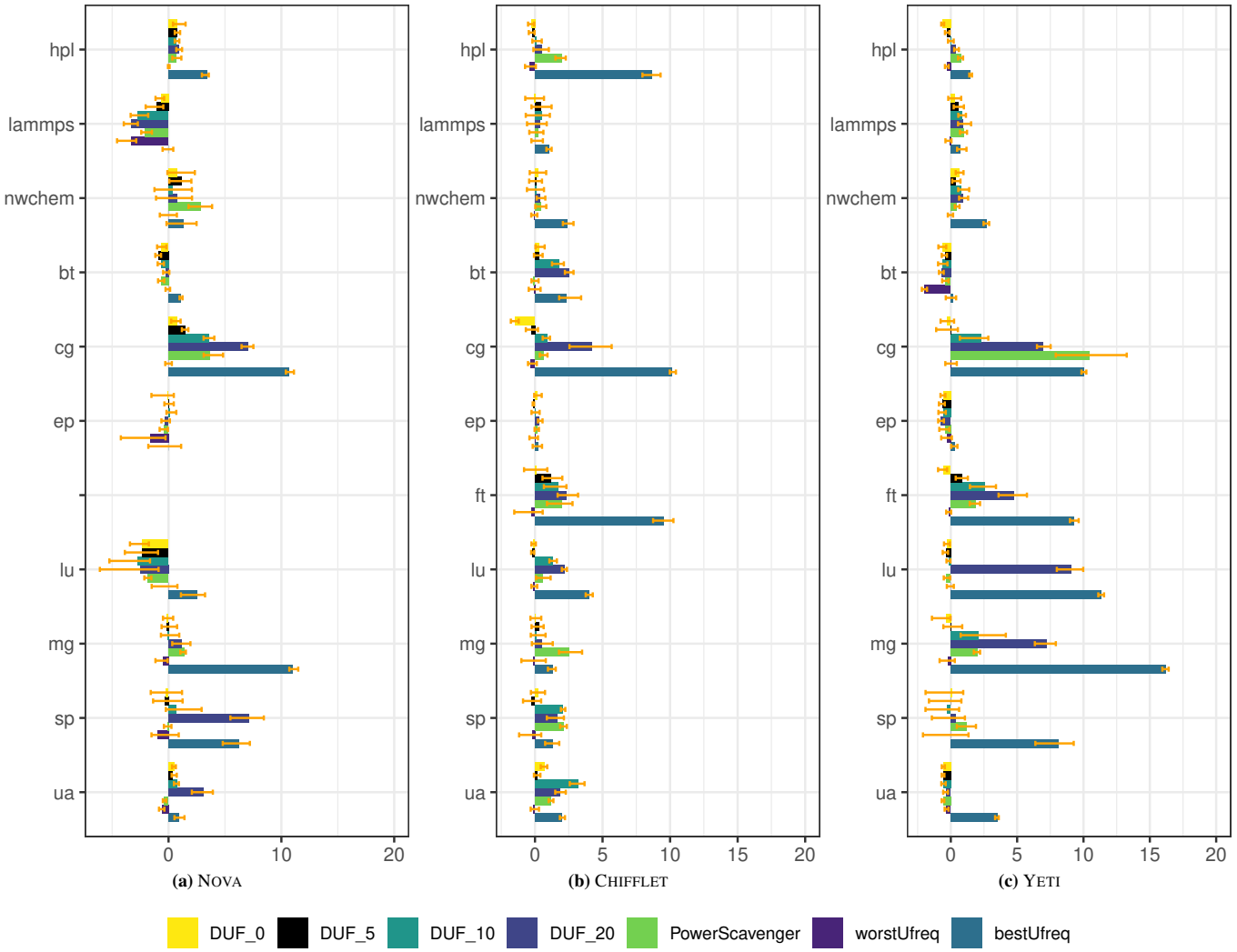


FIGURE 6 DRAM power saving (%)

Finally, DUF provides better savings than UPSCAVENGER for all applications on NOVA (except EP where they provide equivalent savings), all applications except nwchem and MG on CHIFFLET (note that DUF and UPSCAVENGER show equivalent slowdown for SP and EP) and HPL, LAMMPS, CG, FT, MG and UA on YETI regardless of the slowdown.

3.6 | Improving performance with uncore frequency

As stated in Section 1, in addition to improving power consumption, uncore frequency can be used as a leverage to improve the performance of applications that reach TDP. However, the observed performance improvements were rather small. In order to better observe this behavior, we use powercapping, to put a stronger constraint on the default UFS.

Figure 7 shows the performance increase when using DUF_0 and UPSCAVENGER on YETI. We set the powercap to 100W for all applications except EP (98 W), nwchem (90 W) and SP (80 W) because these applications have a lower power consumption under normal configuration. The results are compared to those with default UFS under the same constraints. Note that we only run these experiments on YETI because powercapping is not enabled on NOVA and CHIFFLET. Recall that for these experiments, we use Algorithm 1 while considering longer measurement periods if the minimum uncore frequency is reached. This is because, in this section, we only focus on performance. As a consequence, DUF does not need to report as frequently power measurements.

The results show that for all the applications, using DUF improves performance, with a maximum of 10.53% for CG. UPSCAVENGER shows performance loss for 5 out of the 11 applications (namely LAMMPS, nwchem, CG, MG, and SP). Moreover, DUF outperforms UPSCAVENGER in all cases except for EP where both tools show the same performance. For instance, UPSCAVENGER degrades the performance of CG by 11.90 %, while DUF improves the run time by 10.53 %. Regarding EP performance, they reach 6.58 % with DUF and 6.33 % with UPSCAVENGER. There are two different behaviors which lead to UPSCAVENGER performance loss. For LAMMPS, the DRAM power consumption sees almost no variation. As a consequence the uncore frequency is set to the minimum. For nwchem, a similar behavior is observed within each newly detected phase. Regarding CG, MG and SP, UPSCAVENGER detects many phases for each application. As a consequence, the uncore frequency is reset to the maximum. However, the processor cannot sustain the maximum uncore frequency (because of its power consumption) and the uncore frequency is automatically reduced. Therefore, any decision to reduce the uncore frequency will use the reduced value as current one. For instance, when requesting a reset, the actual uncore frequency that is set may be 1.7 GHz (instead of 2.4 GHz). When UPSCAVENGER request a decrease after that, the set frequency 1.6 GHz. This may lead the uncore frequency to reach lower values. We would like to once again stress that the behavior is based on our own implementation of UPSCAVENGER.

The reason behind the performance increase lies in the core frequency. For instance for FT, when DUF is not used, the core frequency varies between 1.6 and 1.8 GHz for the four sockets, and UFS sets the uncore frequency to 2.21 GHz or higher. When using DUF, the average core frequency is 2.02 GHz while the average uncore frequency is 1.79 GHz. Thus, by limiting the power consumption with the uncore frequency, DUF allows to increase the core frequency, which improves the application performance. This shows that even if YETI uncore frequency scaling algorithm is more reactive to the behavior of the applications, DUF is actually able to better match the needs of the application being executed.

Finally, because of their behavior, nwchem and UA show a performance difference of less than 1.81 %. UA behavior for instance leads to frequently resetting the uncore frequency. As a consequence, the uncore frequency cannot reach low values to allow increasing the core frequency.

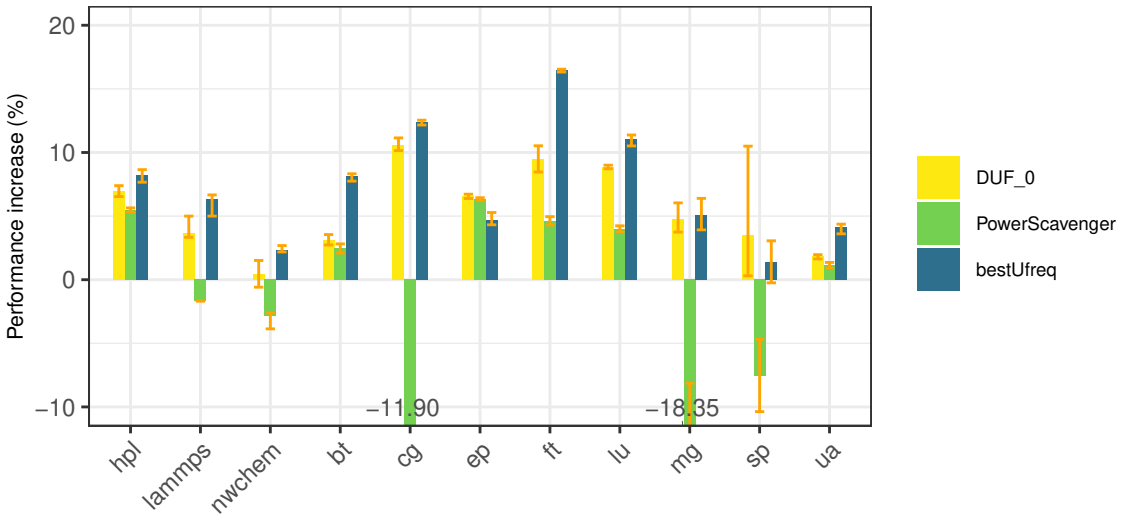


FIGURE 7 Performance increase when using DUF under powercapping on YETI

3.7 | Limitations and possible improvements

DUF evaluation shows how it can improve power consumption while respecting the tolerated slowdown. However, we identified some limitations which are discussed in this section.

As stated in Section 2.3, DUF assumes that the bandwidth drop is correlated to the performance drop. Although we did not observe a situation where this assumption affects the performance, it does not reflect the real impact of memory bandwidth. Moreover, DUF assumes that an increase in the FLOPS/s can come with an increase in the L3 or memory bandwidth with

same factor. Modeling the impact of uncore frequency on L3 cache and memory bandwidth is required to better adapt to the application.

Another improvement could be to adapt the phase change to the platform. As a matter of fact, DUF assumes that if the arithmetic intensity is higher than 1, then the current phase is CPU intensive, otherwise it is memory intensive. However, memory or CPU intensiveness is also related to the processor. As a consequence, we should adapt the condition of the detection phase to the target processor.

Finally, depending on the application, DUF period should adapt if the application behavior varies too frequently by studying how often the phase changes.

3.8 | Conclusions

The evaluation of DUF exhibits how uncore frequency can improve power consumption. It also showed the potential of uncore frequency as a leverage to improve performance. The overall conclusions of the experiments are:

- DUF can adapt to different architectures and different applications;
- DUF manages to stay within the tolerated slowdown for 97.7 % of the tested configurations;
- DUF manages to reduce socket and memory power consumption. For some applications (such as BT), DUF reaches significant power savings (7.94%) without degrading the performance;
- DUF manages to reduce the energy consumption of most applications. Some applications show significant energy savings (such as BT with 9.76 % energy savings).
- By slightly degrading the performance of applications, DUF significantly reduces their power consumption;
- DUF manages to improve applications performance under power capping constraints by allowing the cores frequency to be increased;
- Compared to UPSCAVENGER, DUF manages to better respect the applications slowdown, and to provide better performance under power capping.

4 | RELATED WORK

Adapting uncore frequency is a recent research topic. In [11] the authors provide a machine learning technique to predict the optimal uncore frequency to be used and showed that the nature of the application impacts the energy saving that can be reached. The authors also study the impact of different performance loss policies. However, the proposed tool is static and needs a training phase on all possible frequencies before deciding the best frequency to run the applications whereas DUF is dynamic and is able to adapt to the application behavior.

Won et al. use a similar approach: they design an artificial neural network to characterize applications and to apply the best uncore power management policy to a network of chips [12]. In this study, the authors emulate a new hardware mechanism that would implement their approach.

In [13,14] the authors present a study of the potential energy savings using DVFS and UFS for the application GAMESS. They proposed a performance and a power model, and a runtime to adjust both core and uncore frequencies. The runtime also takes a maximum performance degradation limit. The results show great energy savings with, in some cases, very low overhead. However, this work targets only GAMESS. The models were later used to design a tool that distributes a power budget over socket and memory [15]. However, the tool computes the performance obtained with all core and uncore frequencies, rather than adapting to the behavior of the applications like DUF.

The READEX project [16] aims at providing a tool suite to improve the energy-efficiency of HPC applications by providing the best combination of tuned parameters (like core and uncore frequency). Using READEX, an application is first instrumented using an automatic instrumentation tool. Then the program is analyzed and optimal configurations are stored in a configuration file. Later, learning techniques to improve the decisions were introduced [17]. This approach is complementary to DUF which makes all its decisions at runtime without knowing the application global behavior.

Regarding powercapping, studies like[18] show how dynamically setting core and uncore frequency can improve performance under powercapping. However, they consider iterative applications where decisions made in a loop are used in the next loop.

The work presented in[5] is the closest to DUF. In Section 3.3, we compared DUF to our own implementation of UPSCAVENGER. We showed that for most applications, UPSCAVENGER and DUF provide equivalent power and energy savings for equivalent slowdown. One of the major differences between UPSCAVENGER and DUF is that UPSCAVENGER relies on memory power consumption to avoid any slowdown while saving power. On the other hand, DUF considers that the user can decide the value of the tolerated slowdown. As a consequence, DUF is better able to trade performance for energy whenever possible.

5 | CONCLUSION AND FUTURE WORK

This paper presents DUF, a daemon process that dynamically adapts the uncore frequency in order to reduce the power consumption of applications while limiting the performance degradation to a user-defined limit. Our goal was to provide a tool which better adapts to the behavior of the application and sets the uncore frequency accordingly, unlike the default UFS. The evaluation shows that DUF significantly reduces the power and energy consumption while respecting the slowdown limit. Moreover, DUF manages to reach power savings without impact on performance for some applications. It also shows how uncore frequency can be used as a leverage to improve performance by up to 11.90 % under power capping constraints.

As a future work, we plan to further study how to dynamically set the powercap to the application need and combine it to uncore frequency management. As a matter of fact, if there was a total power budget, instead of applying the same powercap for the hole execution of an application, we could adapt it to the application phases. Using power capping combined to DUF to manage the performance could use the right power budget for the whole execution. We also plan to target DVFS in DUF. In this paper, we rely on automatic frequency scaling (for instance under power capping). However, we plan study if a better frequency could be used and integrate the decision to DUF. Finally, since DUF handles each socket separately, we will study how applications with different behaviors can be run on each socket in order to maximize power savings.

ACKNOWLEDGMENT

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. Hill DL, Bachand D, Bilgin S, et al. THE UNCORE: A MODULAR APPROACH TO FEEDING THE HIGH-PERFORMANCE CORES.. *Intel Technology Journal* 2010; 14(3).

2. Hackenberg D, Schöne R, Ilsche T, Molka D, Schuchart J, Geyer R. An Energy Efficiency Feature Survey of the Intel Haswell Processor. *IEEE International Parallel and Distributed Processing Symposium Workshop, IPDPS* 2015: 896–904. doi: 10.1109/IPDPSW.2015.70

3. Bailey D, Barszcz E, Barton J, et al. The Nas Parallel Benchmarks. *International Journal of Supercomputing Applications* 1991; 5(3): 63–73.

4. Petitet A, C. Whaley R, Dongarra J, Cleary A. HPL - a Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. *http://www.netlib.org/benchmark/hpl* 2000.

5. Gholkar N, Mueller F, Rountree B. Uncore power scavenger: a runtime for uncore power conservation on HPC systems. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* 2019: 27:1–27:23. doi: 10.1145/3295500.3356150

6. Balouek D, Amarie AC, Charrier G, et al. Adding virtualization capabilities to the Grid'5000 testbed. *International Conference on Cloud Computing and Services Science* 2012: 3–20.

7. Plimpton S. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics* 1995; 117(1): 1–19.

8. Valiev M, Bylaska EJ, Govind N, et al. NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications* 2010; 181(9): 1477–1489.

9. Treibig J, Hager G, Wellein G. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. *International Conference on Parallel Processing (ICPP)* 2010: 207–216.

10. Terpstra D, Jagode H, You H, Dongarra J. Collecting performance data with PAPI-C. *Tools for High Performance Computing* 2010: 157–173.

11. Bekele SA, Balakrishnan M, Kumar A. ML Guided Energy-Performance Trade-Off Estimation For Uncore Frequency Scaling. *Spring Simulation Conference (SpringSim)* 2019: 1-12.

12. Won J, Chen X, Gratz P, Hu J, Soteriou V. Up by their bootstraps: Online learning in Artificial Neural Networks for CMP uncore power management. *International Symposium on High Performance Computer Architecture (HPCA)* 2014: 308-319.

13. Sundriyal V, Sosonkina M, Westheimer BM, Gordon M. Comparisons of Core and Uncore Frequency Scaling Modes in Quantum Chemistry Application GAMESS. *High Performance Computing Symposium (HPC)* 2018: 13:1–13:11.

14. Sundriyal V, Sosonkina M, Westheimer B, Gordon M. Core and Uncore Joint Frequency Scaling Strategy. *Journal of Computer and Communications* 2018; 06: 184-201.

15. Sundriyal V, Sosonkina M, Westheimer B, Gordon M. Maximizing Performance under a Power Constraint on Modern Multicore Systems. *Journal of Computer and Communications* 2019; 07: 252-266.

16. Schuchart J, Gerndt M, Kjeldsberg PG, et al. The READEX formalism for automatic tuning for energy efficiency. *Computing* 2017. doi: 10.1007/s00607-016-0532-7.

17. Gocht A, Schöne R, Bielert M. Q-Learning Inspired Self-Tuning for Energy Efficiency in HPC. *International Conference on High Performance Computing Simulation (HPCS)* 2019: 344-347. doi: 10.1109/HPCS48598.2019.9188112

18. Wang B, Miller J, Terboven C, Müller M. Operation-Aware Power Capping. *Euro-Par 2020: Parallel Processing* 2020: 68–82.