DUF : Dynamic Uncore Frequency scaling to reduce power consumption

Etienne André, Remi Dulong, Amina Guermouche, François Trahay

DUF : Dynamic Uncore Frequency scaling to reduce power consumption

Étienne André*, Rémi Dulong*†, Amina Guermouche*, François Trahay*
* Télécom SudParis
Institut Polytechnique de Paris
Evry, France
† University of Neuchâtel
Neuchâtel, Switzerland
Email : first.last@telecom-sudparis.eu

***Abstract*—Reducing the power consumption of applications has become is one of the key challenges in high performance computing. Recent processor architectures differentiate the processor core frequency (that affects the computation units and the L1 and L2 caches) from its uncore frequency (that affects the last level cache and the memory controller). As a consequence, in addition to tuning processor core frequency with DVFS, power consumption can also be controlled through Uncore Frequency Scaling (UFS).**

In this paper, we study how the uncore frequency impacts parallel applications performance and power consumption. We also propose DUF, a runtime system that dynamically adapts the uncore frequency in order to reduce an application power consumption with a user-defined limit on performance degradation.

We evaluate DUF on 3 different architectures. The evaluation shows that when allowing a 10 % performance degradation, DUF can reduce the power consumption of applications by up to 21 %. We also show that DUF can reduce the total energy consumption of by up to 16.56%.

***Index Terms*—Green computing, Power consumption, Uncore frequency, High Performance Computing**

I. Introduction

Reducing the power consumption of supercomputers has become one of the key challenges in high performance computing. As a matter of fact, Summit, the most powerful supercomputer consumes 10.90 MW [14] while the US Department of Energy sets a limit of 20 MW for future exascale machines [1].

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.

One major change that was observed since Intel Haswell processors is the definition of core and uncore frequency domains. In other words, recent processor architectures differentiate the processor core frequency (that affects the computation units and the L1 and L2 caches) from its uncore frequency (that affects the last level cache and the memory controller) [7]. Uncore frequency existed in older processors but was either fixed or sharing the CPU domain [6]. Since Haswell processors, it is possible to set the uncore frequency independently. As a consequence, processors now have Uncore Frequency Scaling (UFS) that handles this uncore frequency. UFS sets the uncore frequency according to the CPU frequency, the energy and performance bias hints and cores stall cycles [6].

In this paper, we study how the uncore frequency impacts parallel applications performance and power consumption on three different architectures. We also propose DUF, a daemon that dynamically adapts the uncore frequency in order to reduce 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.

This paper makes the following contributions:

- we study the impact of uncore frequency on the memory hierarchy and applications power and energy consumption
- we propose, DUF, a daemon that automatically adapts the uncore frequency in order to reduce a running application energy consumption with a limited performance degradation
- we compare DUF to the default behavior on three architectures and 11 applications and benchmarks. The experiments show that, (i) DUF reduces the power consumption by up to 19.21% for HPL (ii) 6% slowdown on MG allows for more than 24% power savings (iii) DUF has a small impact on DRAM power consumption showing more savings than the default behavior
- we show how uncore frequency can be used as a leverage to improve performance by combining DUF to powercapping

The remainder of this paper is organized as follows: Section II present the measurement methodology we used in our experiments. In Section III, we describe how uncore frequency impacts the performance and power consumption of applications. In Section IV, we propose DUF, a daemon that adapts the uncore frequency to the application behavior in order to reduce its power consumption. We evaluate DUF in

Section V, and we compare it to the related work in Section VI. Finally, Section VII concludes the paper.

II. Measurement methodology

This section describes the hardware and software testbed used in our experiments.

A. Target platform

For the experiments, we used three servers from the Grid'5000 [3] platform. The nodes are described below and their characteristics in terms of TDP and frequencies are summarized in Table I. One should note that while TDP constitutes a physical hard limit, a processor can briefly go beyond it due to thermal inertia:

- NOVA is a 23-nodes cluster, where each node is equipped with 2 Intel Xeon CPU E5-2620 v4 CPUs (8 cores per CPU) and 64 GiB of memory. We run our experiments nova-1.
- CHIFFLET is an 8-nodes cluster, where each node is equipped with 2 Intel Xeon E5-2680 v4 CPUs (14 cores per CPU) and 768 GiB of memory. We used chifflet-1 for our experiments.
- YETI is a 4-nodes cluster. Each node is equipped with four Intel Xeon Gold 6130 CPUs (16 cores per CPU). Each NUMA node has 64 GiB of memory. Thus, YETI has 64 cores and 256 GiB of memory in total. In all experiments, we used yeti-2.

We chose these nodes because they provide various characteristics: NOVA sets the uncore frequency to the maximum when it detects activity on the CPU. CHIFFLET and YETI use more complex uncore frequency scaling algorithms where the uncore frequency varies depending on the CPU load. All platforms run under Intel Pstate with performance governor. Note that, for all our experiments, hyperthreading was disabled.

	NOVA	CHIFFLET	YETI
number of cores	16	28	64
microarchitecture	Broadwell	Broadwell	Skylake
base frequency (GHz)	2.1	2.4	2.1
TDP (W) per socket	85	120	125
uncore frequency (GHz)	[1.2-2.7]	[1.2-2.7]	[1.2-2.4]

TABLE I: Target platforms characteristics extracted from processors documentation

B. Target applications

Throughout this study, we used several applications. In this section, we present the applications that we used for all our experiments.

On all platforms, applications were compiled with gcc 6.3.0 with -O3 flag and 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.

1) NAS Parallel Benchmarks: The NAS Parallel Benchmarks [2] provide a set of small applications used to evaluate the performance of supercomputers. We used the following benchmarks: BT, CG, EP, FT, LU, MG, SP, UA from NPB-3.3.1 OpenMP version. We chose the problem size so that each application execution time is in the [30s-200s] range. On NOVA EP, and MG were run using class D while on CHIFFLET, FT was also run using the class D problem size. The remaining benchmarks were run using class C. On YETI, all benchmarks were run using class D except SP for which we used class C. The OpenMP threads were bound to cores in a round-robin fashion.

2) High Performance Linpack: High Performance Linpack (HPL) [9] is a software package that solves dense linear algebra systems. It is used as a reference benchmark to compute the performance of the supercomputers in the TOP500 [14].

We used HPL version 2.3 compiled using the Math Kernel Library (MKL) version 2019.1.144. HPL takes a configuration file where we only modified the problem size (N), the block size (NB) and the processes grid (PxQ). We set NB to 224 on all platforms. N was set to 58912 on NOVA, 62720 on CHIFFLET and 91840 on YETI. (PxQ) was set to (4x4), (4x7) and (8x8) on NOVA, CHIFFLET and YETI respectively.

3) LAMMPS: LAMMPS [10] is an MPI application that performs molecular dynamics simulation. We used the input file `in.lj` provided for the accelerate suite where we set the run value to 50000. We used commit version `aa2b88578`.

4) nwchem: Nwchem [17] is a computational chemistry application. We used the input data set `3carbo_dft.nw` from the qdm provided files. We used commit version `67f5237ab`. Note that nwchem problem size depends on the number of processes being used.

C. Measurement framework

As stated before, our work is divided in two part : (i) understanding the behavior and impact of uncore frequency (section III) and (ii) providing DUF, an uncore frequency scaling daemon.

In the first part of the paper (section III), we use LIKWID [16] to set the uncore frequency and to measure the power consumption of the applications. We use `likwid-perfctr` command from LIKWID version 4.3.0. This command reads hardware counters provided by the Intel RAPL interface. These counters provide energy consumption information for components such as the processor and the DRAM. On processors like Intel Sandy Bridge, RAPL is based on a modeling approach. Since the Intel Haswell generation, processors have fully integrated voltage regulators (FIVR) that provide actual measurements and allow for more accuracy [6].

All the measurements performed with LIKWID were done every second, with no overhead for all the applications. Regarding measurements errors, all configurations show a small standard deviation.

In the section V, we describe DUF which relies on flops and memory bandwidth measurements. These measurements were collected using the PAPI library [13] with git commit version

	application	Power (Watt)	ufreq (GHz)
NOVA	HPL	65.39	2.7
	NPB CG	38.89	2.7
	NPB EP	41.72	2.7
	NPB MG	44.55	2.7
CHIFFLET	HPL	120.48	2.4
	NPB CG	79.2	2.7
	NPB EP	100.83	2.7
	NPB MG	82.69	2.7
YETI	HPL	124.61	[1.6-1.7]
	NPB CG	123.37	[2.3-2.4]
	NPB EP	116.08	2.4
	NPB MG	123.32	[2.2-2.3]

TABLE II: Average observed uncore frequency on NOVA, CHIFFLET and YETI.

`ceb64276`. We did not use LIKWID because we observed an overhead when reducing the measurement period. LIKWID was only used to set the uncore frequency. Note that PAPI is also used to measure the power consumption in section V.

D. Uncore frequency scaling default behavior

In order to understand the Uncore Frequency Scaling (UFS) default behavior on our experimental testbed, we measure the average uncore frequency when applications with different profiles run. For that purpose, we use CG, EP, and MG from the NAS Parallel Benchmarks [2] and HPL [9]. Section III provides a detailed characterization of the applications.

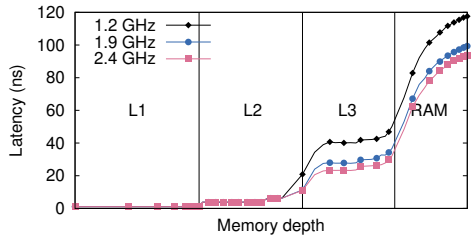
Table II shows the average uncore frequency on the different applications and the different platforms. It also provides the average power consumed by the applications. For each platform, the uncore frequency is measured for each socket and we report the average observed uncore frequency.

On NOVA, UFS sets the uncore frequency to its maximum (2.7 GHz) for both sockets. On CHIFFLET, CG, EP and MG run at the maximum uncore frequency (2.7 GHz) on both sockets. The uncore frequency for HPL is lower (2.4 GHz), but it remains higher than the minimum uncore frequency on this machine (1.2 GHz). We also observe that HPL reaches the thermal design power (TDP) of the machine (120 W). This behavior suggests that on CHIFFLET, the Uncore Frequency Scaling policy first sets the uncore frequency to its maximum, and then reduces its when the TDP is reached.

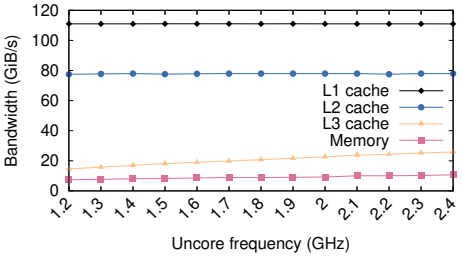
A similar behavior is observed on YETI: EP has a limited power consumption. Thus, the uncore frequency is set to the maximum (2.4 GHz). Meanwhile, since CG, MG, and HPL reach the TDP, their uncore frequency is reduced. We also observe that while the uncore frequency is set for each socket, the four sockets on YETI have similar behavior: for instance, for HPL, two sockets have an uncore frequency of 1.6 GHz while the others have an uncore frequency of 1.7 GHz.

III. IMPACT OF UNCORE FREQUENCY ON PERFORMANCE AND POWER CONSUMPTION

In this section, we evaluate how the uncore frequency impacts the performance and the power consumption of parallel applications. For that purpose, we run experiments on YETI.



(a) latency



(b) bandwidth

Fig. 1: Uncore frequency impact on memory performance

Its L1 cache is divided in 2x32 KiB (instruction and data). The L2 cache size is 1 MiB while the L3 cache size is 22 MiB.

A. Impact on the memory hierarchy

In order to assess the impact of uncore frequency on the performance of memory, we measure the memory latency and bandwidth using the `lmbench` benchmark [8]. Figure 1 shows the impact of uncore frequency on the different cache levels and the main memory regarding both latency (Figure 1a) and bandwidth (Figure 1b) on YETI.

Figure 1a presents the impact of three uncore frequencies on cache and memory latency. The results show that the uncore frequency does not impact L1 and L2 latencies. As described in [7], the uncore frequency affects memory access latency to the L3 cache and to the main memory. More specifically, the latency when accessing the L3 cache is 20 ns with an uncore frequency of 2.7 GHz, while it takes 35 ns when the uncore frequency is 1.2 GHz. Similarly, accessing the main memory costs up to 93.5 ns with an uncore frequency of 2.7 GHz, and up to 117.6 ns when the uncore frequency is 1.2 GHz.

Figure 1b shows that uncore frequency has no impact on L1 and L2 cache bandwidth. For L3 cache and main memory, the higher the frequency, the larger the bandwidth.

As a conclusion, uncore frequency has no impact on L1 and L2 cache, but impacts L3 cache and main memory. As a matter of fact, decreasing uncore frequency has a negative impact on L3 cache and main memory latency and bandwidth.

B. Impact on applications performance

Figure 2 shows the performance of the NAS Parallel benchmarks (NPB) as the uncore frequency varies. The results are presented as a percentage over the default execution time for each benchmark.

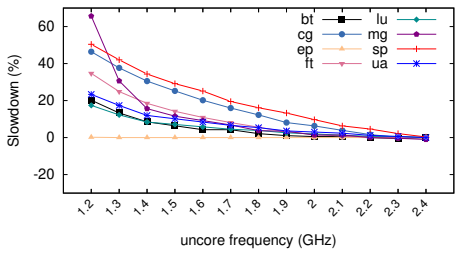


Fig. 2: Impact of uncore frequency on the performance of NAS Parallel Benchmarks on YETI.

Kernel	perf. degradation (%)	uncore cost (%)	comp. intensity
BT	20.2	29.1	1.75
CG	46.4	93.3	0.17
EP	0.2	0.9	60.42
FT	34.7	68.2	0.51
LU	17.4	33.3	1.45
MG	65.6	50.0	0.33
SP	50.4	82.6	0.19
UA	23.3	33.2	0.0052

TABLE III: Characterization of the NPB applications on YETI

For all the tested applications, the best performance is obtained with the highest uncore frequency. Lowering the uncore frequency significantly degrades the performance of most applications such as MG (65 % performance degradation). However, the performance of EP is not affected by the uncore frequency (less than 0.2 % performance variation).

In order to characterize the NPB applications, we analyze them using the NumaMMA memory profiler [15] with an uncore frequency of 2.4 GHz. NumaMMA uses Intel Precise Event-Based Sampling (PEBS) to collect information on an application memory access and it reports, for each sample, which level of the memory hierarchy was accessed (L1 cache, L2 cache, etc.) as well as the memory access latency. We use this information to compute, for each application, the *uncore cost*, which corresponds to the time spent accessing data in the uncore part of the memory hierarchy (ie. L3 cache, and RAM). The *uncore cost* is expressed as a percentage of the total time spent accessing data from any part of the memory hierarchy. Table III reports the measured performance degradation, the *uncore cost*, and the computational intensity of the applications. We observe that the applications with a high uncore cost (such as CG, FT, MG, or SP) are the most impacted by the uncore frequency. On the contrary, the applications whose memory accesses are mostly in the L1 or L2 cache (such as BT, EP, LU, or UA) are less affected by the uncore frequency.

Table III also shows the computational intensity of the applications over their entire execution. It indicates that BT, EP, LU are CPU-intensive while CG, FT, MG, SP and UA are memory intensive.

C. Impact on application power consumption

Figure 3 shows the impact of uncore frequency on the power consumption of the processors (Figure 3a) and the memory (Figure 3b), and the total energy consumption of applications (Figure 3c).

Figures 3a and 3b show that the higher the frequency the larger the power consumption of both the processor and the DRAM. For instance, at 1.2 GHz, CG consumes roughly 74% of the default power consumption and 91.86% of the default DRAM power consumption. Thus, the uncore frequency significantly impacts the power consumption of all the tested applications. As expected, memory-intensive applications have the most package power consumption difference between the minimum and maximum frequency. Regarding DRAM power consumption, the uncore cost provided in Table III seems to be a good indicator of larger DRAM power consumption. For instance, UA has a very low computational intensity, indicating that it is memory-intensive. However, as its uncore cost is 33.2%, this indicates that most of its accesses are in L1 and L2 cache which explains why its DRAM power savings are low compared to the other memory-intensive applications such as CG.

As described in section III-B, reducing the uncore frequency also increases the completion time of most applications. Thus, the total energy consumption may be reduced only for some applications, as shown in Figure 3c. For instance, since EP performance is not affected by a lower uncore frequencies while its power consumption is reduced, its total energy consumption at 1.2 GHz is 16.6 % lower than when running at the default frequency. On the contrary, although using a low uncore frequency for SP reduces its power consumption by up to 25%, its performance it largely degraded (up to 50.4 %), and 2.0 GHz is the most energy efficient uncore frequency.

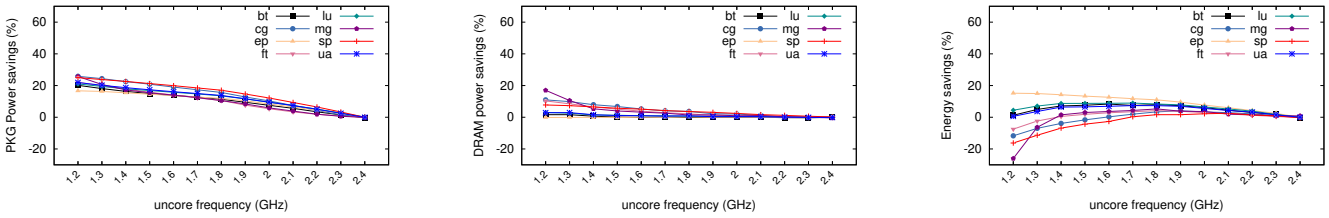
D. Conclusion

Carefully selecting the uncore frequency for an application allows to significantly reduce its power consumption. However, the performance of some applications are greatly degraded. As a result, while the best performance is usually achieved with the highest uncore frequency, significant power savings can be achieved in return of a performance deterioration for some applications.

IV. NEW UNCORE FREQUENCY SCALING TOOL

In this section, we describe DUF [1] (that stands for Dynamic Uncore Frequency scaling), a daemon that dynamically adapts the uncore frequency while a parallel application runs in order to trade a limited performance degradation for power savings. The aim of DUF is twofold: to reduce the power consumption of an application, and to limit the performance degradation to a user-provided upper-bound.

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



(a) PKG power (b) DRAM power (c) Total energy

Fig. 3: Uncore frequency impact on NAS Parallel Benchmarks processor and memory power consumption and on the total energy consumption. The results are presented as savings over the default power consumption on YETI.

A. *Overview of* DUF

DUF runs as a daemon that aims at lowering the power consumption while limiting the performance degradation to an upper threshold. To do so, when starting the daemon, the user specifies a set of sockets to monitor, and a maximum performance degradation tolerance. DUF then periodically invoke its *measurement module* that collects the CPUs performance counters. Based on the collected data, the *regulator module* decides whether the uncore frequency should be changed or not. The decision algorithm described in section IV-C applies for each user-specified socket. It can be summarized as follows: DUF detects the applications phases (eg. memory intensive vs compute intensive) 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.

B. *Measurement module*

DUF *measurement module* collects the CPUs hardware counter in order to guide the *regulator module* in its decisions. DUF uses PAPI to read the performance counters related to the FLOPs and to the memory bandwidth. It then computes the arithmetic intensity as the ratio between the FLOPs and the memory bandwidth. An arithmetic intensity greater than 1 indicates that the application mostly performs computation. An arithmetic intensity lower than 1 indicates that the application is currently memory intensive. Note that, at every time step, DUF prints the different measurements in addition to the power consumption collected using PAPI.

C. *Regulator module*

In order to decide which uncore frequency to apply to a socket, DUF *regulator module* runs Algorithm 1 after every measurement period. If a new application phase is detected (lines 6 to 11), DUF sets the maximum uncore frequency and measures the performance counters max_flops and max_bw (the achieved memory bandwidth). Note that we also assume that if, during the execution of the application, the flops and the memory bandwidth increase by 100%, without changing the phase, DUF resets the uncore frequency in order have accurate measurements (lines 15-18).

Otherwise, DUF checks how the previous decision impacted flops and memory bandwidth. DUF considers that if the flops drop compared to the previous iteration, but the memory bandwidth remains stable, then the drop was the behavior of the application itself rather than the impact of uncore frequency. Based on this assumption, DUF decreases uncore frequency (lines 12-14) in this case. 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%.

Finally, if the current uncore frequency does not degrade the maximum performance for the current phase more than the user-specified threshold, DUF decreases the uncore frequency (lines 19-21). If the minimum uncore frequency is reached, DUF increases the measurement period as we reach a stable phase (line 23). Note that we limit the measurement period to 10 times the initial period. Note also that every time DUF changes the uncore frequency, it resets the period to the initial period.

In case the performance degradation exceeds the tolerance, DUF increases the uncore frequency in order to remain within the tolerated performance drop (lines 24-25). If the maximum uncore frequency is reached, we consider that the application behavior changed and update the maximum flops to the current flops and decrease the uncore frequency (lines 26-27).

V. EXPERIMENTS

In this section, we evaluate whether DUF meets its two objectives as stated in section IV: saving power, while limiting the performance degradation to a user-defined limit.

In order to study the performance of DUF in terms of execution time and power consumption, we run DUF with the aforementioned applications on NOVA, CHIFFLET and YETI. For each application, we use DUF with four different slowdown tolerances: DUF_0 (0 % tolerance), DUF_5 (5 % tolerance), DUF_{10} (10 % tolerance), and DUF_{20} (20 % tolerance). The results presented in this section are a mean over 10 runs.

On each platform, we measured the slowdown (Figures 4a, 4c and 4e), the socket power savings (Figures 4b, 4d

Algorithm 1 Uncore Frequency Scaling algorithm

```
period                                   ▷ interval between measures
phase ← unknown
loop                                     ▷ Every period
    flops ← measure_flops()
    oi ← measure_operational_intensity
    if oi > 1 and phase! = CPU then
        phase ← CPU
        FREQ=MAX_UFREQ
    else if oi < 1 and phase! = memory then
        phase ← memory
        FREQ=MAX_UFREQ
    if flops − flops < old_flops then
        if bw/max_bw > 1 − perf_loss then
            DECREASE_FREQUENCY
    if flops > 2 ∗ old_flops then
        if bw > 2 ∗ old_bw then
            FREQ=MAX_UFREQ
        else DECREASE_FREQUENCY
    if flops > perf_loss ∗ max_flops then
        if freq > min_freq then
            DECREASE_FREQUENCY
        else if period < 10 ∗ period then
            period = period ∗ 2
    else if freq < max_freq then
        INCREASE_FREQUENCY
    else max_flops ← flops
        DECREASE_FREQUENCY
```

and 4f), and the total energy savings (Figures 4g, 4h and 4i). All the results are presented as a percentage of decrease or increase over the default values on each platform. In addition to DUF results, the figures also present the best and worst possible values obtained by manually setting the uncore frequency. Note that we use the terms socket, processor and package interchangeably.

All the comparisons that DUF makes consider that a 2 % difference is considered as a measurement error. Finally, we set DUF uncore frequency step to 100 MHz and the measurement period to 200 ms. Note that we varied the measurement period from 10 ms to 500 ms and did not observe any overhead. However, at 10 ms, some applications keep changing phases (from CPU to memory and *vice versa*). As a consequence, for those applications, DUF keeps resetting the uncore frequency (especially when the tolerated slowdown is low) and the behavior is equivalent to the default behavior. 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 handle some of these limitations in paragraph V-F.

Note that on YETI, we do not present the results on LAMMPS and MG. This is because, unlike the other applications, these two applications show an overhead when we do not use the regulator of DUF. In other words, the measurement itself impacts these two applications. We are still investigating the reasons behind this behavior. This is also the case for BT and nwchem but the measurement overhead is small (1.6%). As a consequence, BT and nwchem results are presented for YETI. Finally, the class D problem for FT could not run on NOVA while the class C execution time is too short. Thus, we do not present FT results on NOVA.

A. DUF *impact on execution time*

This section describes the impact of DUF on the applications execution time as reported in Figures 4a, 4c, and 4e.

The results show that, on all platforms, DUF remains within the tolerated slowdown for most of the applications. For some applications, an overhead is observed but it is still within the 2% measurement error considered by DUF.

In the remaining of this section, we discuss the three behaviors we observed: applications which slowdown exceed the user-defined limit, applications which performance are not degraded, and applications which slowdown reach the user-defined limit,

1) Slowdowns that exceed the tolerance threshold: Two applications exceed the user-provided limit: SP on NOVA when running with DUF_{20}, and CG on YETI when running with DUF_5 and DUF_{10}. We could not find the reason behind the small overhead on CG, and we are still investigating.

When running SP with DUF_{20} on NOVA, the performance degradation reaches 24 %. This overhead is due to SP memory bandwidth usage. As stated in section IV-C, with DUF_{20}, the tolerated bandwidth drop is set to 80 %. However, for SP, 80 % bandwidth decrease causes a large performance degradation. Note that we tried a 85% lower bound on the bandwidth and observed that in that case, SP slowdown was within the 20 % limit. This problem is further discussed in section V-F.

2) Applications which performance do not degrade: Although a performance degradation limit is allowed to applications running with DUF, some of them do not slow down up to the limit. For most of the applications, this is due to their behaviors. For instance, as EP is not impacted by uncore frequency as discussed in III-B, it shows no overhead. The other applications either switch phases rather often (nwchem) or see a sudden increase in their flops and memory bandwidth (MG). Both of these behaviors lead DUF to reset the uncore frequency. Note that DUF still manages to reduce the uncore frequency as discussed in section V-B. BT, on the other hand, exhibits a different behavior where the flops actually drop by more than the tolerated slowdown at some point, which lead DUF to increase the uncore frequency. This is also the case for UA, which in addition to that, needs to reach very low frequencies to cause any slowdown (as shown in Figure 2).

The other reason behind the absence of overhead is the condition on the memory bandwidth drop as stated in section V-A1. With DUF_5 on NOVA, SP shows no slowdown. This is because, at some point, the memory bandwidth drop by more than 95% which leads DUF to increase the uncore frequency.

3) Slowdowns that reach the tolerance threshold: For most of applications, the observed slowdown is close to the tolerated slowdown or to the maximum possible slowdown (for the situations where the maximum slowdown is lower than the tolerated slowdown).

This indicates that DUF manages to (i) detect phase changes, and (ii) properly regulates the uncore frequency. For instance, on NOVA, one can see that HPL, LAMMPS, CG and LU are very close to the tolerated slowdown while on CHIFFLET only nwchem shows no change at all while hpl shows an effect with DUF_{10} and DUF_{20}. On YETI,

4) Conclusions: The experiments using DUF on different applications and different platforms allowing 4 different slowdowns show that : (i) DUF manages to respect the maximum tolerated slowdown for all applications but one, and (ii) DUF manages to get as close as possible to the tolerated slowdown.

B. DUF *impact on socket power consumption*

Figures 4b, 4d and 4f show the package power saving when using DUF on NOVA, CHIFFLET and YETI respectively. The figures show that most applications show power savings on all platforms.

However, running nwchem with DUF on NOVA does not reduce the application power consumption. DUF only manages to slightly decrease the uncore frequency for a short moment before increasing it, as explained in section V-A2.

For the other applications, DUF manages to provide power saving reaching maximum savings with DUF_{20} except for EP. For instance, HPL provides the best saving at 19.21 % on NOVA while CG reaches 22.55 % on CHIFFLET and 16.34% on YETI.

EP reaches 12.5 % power saving on YETI regardless of the tolerated slowdown. This is because EP is not impacted by uncore frequency which is set to the minimum on all platform. Note that the power savings observed on Figure 3a for EP are 16.67 %. The difference in power savings lies in the time DUF requires to reach the minimum frequency as it decreases it by 100MHz at each time step.

The figures also show that in some cases the power consumption of an application is reduced without impacting the performance. On NOVA, this is the case for BT, MG and UA. For instance, when running BT on NOVA, DUF_{10} reduces the performance by only 0.67 %, but the power savings reach 4.89 %. A similar behavior is observed with CG, MG, and SP on CHIFFLET. The reason behind this behavior is that DUF manages to reduce uncore frequency but the application behavior does not allow DUF to keep a low frequency. Thus, during the steps when the uncore frequency is reset, DUF manages to reduce the uncore frequency.

With DUF_{20}, HPL power savings are 4.3 % on CHIFFLET while we observe a high slowdown (21.84 %). This is because HPL power consumption is roughly the same until 1.6 GHz (when manually setting the uncore frequency). Moreover, when manually setting the uncore frequency to 1.4 GHz in order to reach a 20% slowdown, the observed that the power consumption is reduced by 5.4 %. The uncore frequency when using DUF reaches 1.4GHz, but the application does not run all the time at this frequency, which explains the difference.

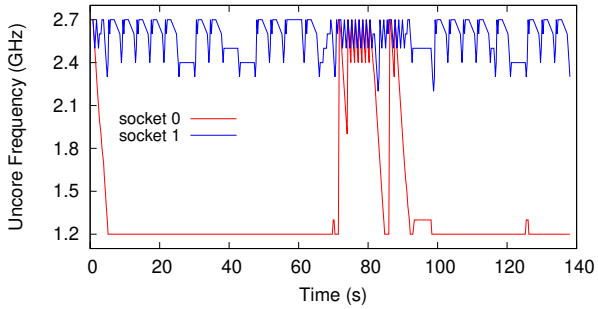


Fig. 5: Uncore frequency behavior during SP execution time when using DUF on CHIFFLET with 20% tolerated slowdown

Moreover, as DUF handles each socket separately, the uncore frequency may not be the same on each socket. Figure 5 shows, for SP on CHIFFLET, how the uncore frequency varies on each socket when 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 the maximum (2.7 GHz) on socket 1. This is due to the fact that on socket 1, both the flops and the memory bandwidth follow a pattern where they both increase, then both decrease then both increase ...etc. Thus, DUF decreases the uncore frequency, then increases it and so on and so forth. On the other hand, on socket 0, DUF manages to reduce the uncore frequency since the pattern is different as the memory bandwidth is stable.

Finally, Figure 5 also shows that DUF manages to reach a stable phase (on socket 0) where the uncore frequency does not vary until a variation is detected. It does so for all the applications which behavior remain stable for some time.

C. DUF *impact on DRAM power consumption*

Although the power consumption of DRAM is small relative to the socket power consumption, it may still significantly affect the total power consumption. In our experiments, the impact of DUF on the DRAM power consumption is roughly the same on all platforms. Figure 6 reports the DRAM power savings on CHIFFLET.

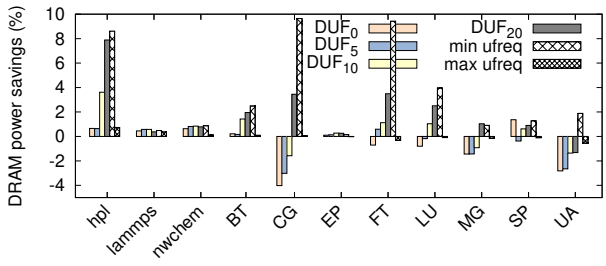


Fig. 6: DUF impact on DRAM power consumption on CHIFFLET for a 200ms period measurement

The figure shows that for most applications, the DRAM power consumption corresponds to the default DRAM power

consumption $\pm$ 2 %. The only exception is HPL whose DRAM power consumption is reduced by up to 7.89 %.

D. DUF *impact on energy consumption*

Figures 4g, 4h and 4i show the impact of DUF on application total energy consumption. In other words, the figures show the sockets + memory energy consumption. For most applications, using DUF allows to save energy. However, as the energy consumption is computed as the product of the execution time and the power consumption, if the application slowdown is higher than the power consumption, then DUF actually wastes energy instead of saving it.

As a consequence, limiting the slowdown to up to 10 % allows to greatly reduces the power consumption, which reduce the overall energy consumption. For instance, a 6 % performance degradation for MG on CHIFFLET leads to 20.5 % power savings with DUF_{20}. However, using DUF_{20} greatly reduces the performance of some applications while only slightly reducing their power consumption. As a result, the overall energy consumption is not always reduced with DUF_{20}. This is the case for SP (2.78 %) on NOVA, HPL (5.03 % with DUF_{10}, and 15.84% with DUF_{20}) on CHIFFLET, and LU (4.07 %) on YETI.

On all the platforms, EP is the application with the best energy savings (up to 16.54 % on NOVA). This is due to the fact that lowering the uncore frequency does not affect EP performance, but does reduce its power consumption. As a result EP energy consumption is significantly reduced.

E. Going further with DUF *: the impact of powercapping*

As the uncore frequency has a positive impact on power consumption, it can be used as a leverage to improve performance. As a matter of fact, when reaching thermal design power, the CPU frequency may be reduced to respect the power limit, which degrades the performance.

Many of the applications that we studied merely reach the thermal design power on the tested platforms. In order to put a stronger constraint, we use a powercapping technique on YETI on BT, EP, and FT in a similar fashion as described in [5]. We chose these three applications as they exhibit different behavior as shown in Table III. Note that powercapping may become a solution to limit power consumption as DoE plans to limit the power budget for the exascale machines to 20 MW [1].

Figure 7 shows the performance increase when using DUF compared to YETI default frequency scaling. We set the powercap to 100W for BT and FT, and 98W for EP. EP powercap is lower as its power consumption in the default behavior is not as high as BT and FT.

The results show that for all applications, using DUF improves the performance, with a maximum of 14.59% for EP. The reason behind this behavior lies in the core frequency. For instance for FT, when the default behavior is used under powercapping, the average CPU frequency on all four sockets is 1.97 GHz while the uncore frequency does not go below 2.23 GHz. When using DUF, the CPU frequency is 2.27 GHz and the uncore frequency go as low as 1.7GHz. This shows that even if YETI uncore frequency scaling algorithm is more reactive to the behavior of the applications, DUF manages to improve performance. Note that we also studied the impact of power consumption, and as CPU frequency is increased, all the applications reach the powercap, thus no power is saved.

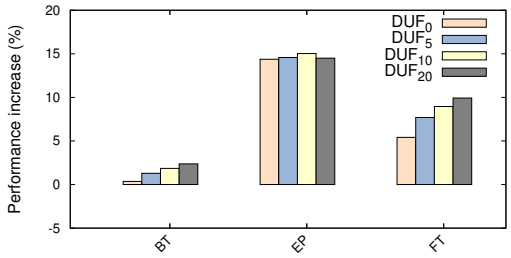


Fig. 7: Performance increase when using DUF under powercapping constraints on YETI

F. 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 IV-C, DUF assumes that the bandwidth drop is correlated to the performance drop. Although this assumption did not impact the performance described in section V-A (except for SP), it does not reflect the real impact of the memory bandwidth. The best way to handle this issue would be to model the impact of uncore frequency on memory bandwidth and integrate the model to DUF.

The other limitation that DUF has is that it considers floating point operations to control the overhead and detect a phase change. This eliminates the applications that mainly performs integer operations (IOPS). One solution to the problem would be to consider the entire operations being performed such as the UOPS. However UOPS include all operations, which may impair the results.

Finally, depending on the application, DUF period should be larger because the application behavior varies too frequently. For now, this is done manually. Working on an automatic way to adjust the period, by studying how often the phase changes, is part of our future work. Note that this solution will not help for applications which behavior varies within the same phase.

G. Conclusions

This sections presents the results of DUF regarding execution time, socket and DRAM power consumption and energy consumption. It also show the potential of uncore frequency as a leverage to improve performance. The overall conclusions of DUF are :

- DUF manages to stay within the tolerated slowdown

- DUF manages to reduce the socket and memory power consumption of the applications
- DUF manages to reduce energy consumption when the slowdown it not higher than the energy savings
- DUF manages to improve applications performance under power capping constraints by allowing the cores frequency to be increased

VI. Related work

One of the main differences between our work and the following works is that our tool was used on different architectures using different UFS algorithms and providing different characteristics (vectorization, different TDP, different CPU frequencies, ...).

Very few studies exist on uncore frequency. In [4] 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 maximum energy saving that can be reached. Note that in this study, the authors studied the impact of different performance loss policies.

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 [18]. In this study, the authors emulate a new hardware mechanism that would implement their approach.

In [12] the authors present a study of the potential energy savings using DVFS and UFS for GAMESS. The work was later extended in [11] where a performance model, a power model and a runtime to adjust both core and uncore frequencies were provided. The runtime also takes a maximum performance degradation limit. The results show great energy savings with, in some cases, very low overhead.

The work presented in [5] is the closest to our work. The authors describe UPSCavenger, an uncore frequency scaling library which adapts to application phases and manages to reduce power consumption. Just like DUF and unlike the default UFS, UPSCavenger tend to reduce uncore frequency instead of core frequency which explains the performance improvement under power capping constraints. However, the provided tool does not consider a performance loss limit. As a consequence, DUF can be seen as a generalization of UPSCavenger, as tolerating a 0 % slowdown in DUF is equivalent to UPSCavenger. Note that we could not compare DUF to UPSCavenger as its source code is not available.

VII. Conclusion and future work

This paper presents a study on uncore frequency. We start by studying the impact of uncore frequency on application performance and power consumption. Then we present DUF, a daemon 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. The evaluation show that DUF significantly reduce the power and energy consumption while respecting the slowdown limit.

As a future work, we plan to further study how uncore frequency and powercapping can be combined, before also considering CPU frequency.

References

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

[2] 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. *Int. J. High Perform. Comput. Appl.*, 5(3):63–73, September 1991.

[3] D. Balouek et al. Adding virtualization capabilities to the Grid'5000 testbed. In I. Ivanov, Marten van Sinderen, Frank Leymann, and Tony Shan, editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer, 2013.

[4] S. A. Bekele, M. Balakrishnan, and A. Kumar. Ml guided energy-performance trade-off estimation for uncore frequency scaling. In *2019 Spring Simulation Conference (SpringSim)*, pages 1–12, April 2019.

[5] Neha Gholkar, Frank Mueller, and Barry Rountree. Uncore power scavenger: A runtime for uncore power conservation on hpc systems. In *Proceedings of the Conference for High Performance Computing, Networking, Storage and Analysis, SC'19*, 2019.

[6] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer. An Energy Efficiency Feature Survey of the Intel Haswell Processor. In *IEEE Int. Parallel and Distributed Processing Symposium (IPDPS Workshops)*, pages 896–904, 2015.

[7] David L Hill, Derek Bachand, Selim Bilgin, Robert Greiner, Per Hammarlund, Thomas Huff, Steve Kulick, and Robert Safranek. The uncore: A modular approach to feeding the high-performance cores. *Intel Technology Journal*, 14(3), 2010.

[8] Larry W McVoy and Carl Staelin. lmbench: Portable tools for performance analysis. In *Proceedings of the Usenix Annual Technical Conference, USENIX ATC'96*, pages 279–294, 1996.

[9] Antoine Petitet, R C. Whaley, Jack Dongarra, and A Cleary. Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. *Innovative Computing Laboratory*, page Available at http://icl.utk.edu/hpl/, September 2000.

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

[11] 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.

[12] Vaibhav Sundriyal, Masha Sosonkina, Bryce M. Westheimer, and Mark Gordon. Comparisons of core and uncore frequency scaling modes in quantum chemistry application gamess. In *Proceedings of the High Performance Computing Symposium*, HPC '18, pages 13:1–13:11, San Diego, CA, USA, 2018. Society for Computer Simulation International.

[13] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with papi-c. In *Tools for High Performance Computing 2009*, pages 157–173. 2010.

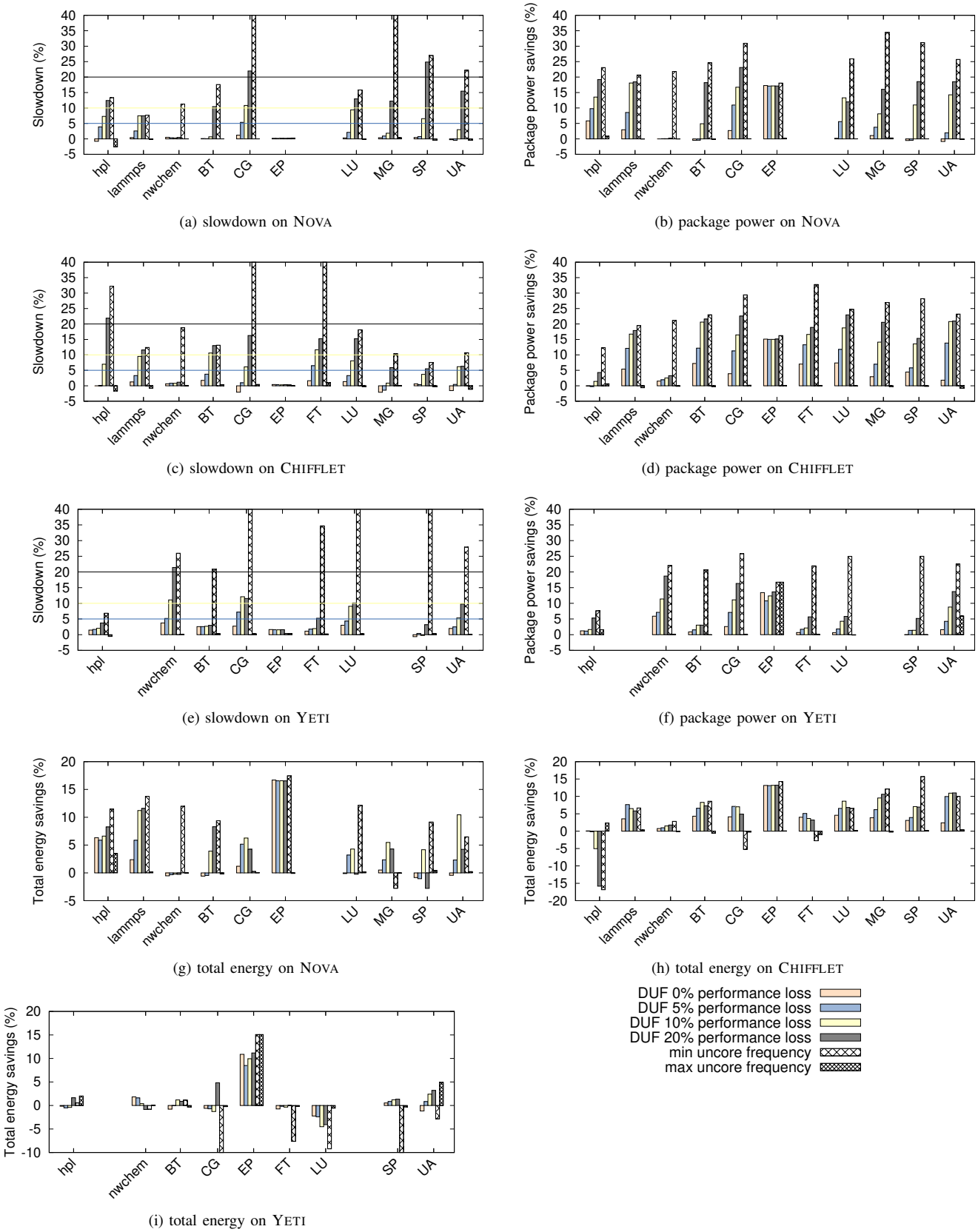
[14] TOP500 Supercomputer Site. http://www.top500.org.

[15] François Trahay, Manuel Selva, Lionel Morel, and Kevin Marquet. NumaMMA: Numa MeMory Analyzer. In *Proceedings of the International Conference on Parallel Processing, ICPP'18*, 2018.

[16] Jan Treibig, Georg Hager, and Gerhard Wellein. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In *Int. Conf. on Parallel Processing (ICPP Workshops)*, pages 207–216, 2010.

[17] Marat Valiev, Eric J Bylaska, Niranjan Govind, Karol Kowalski, Tjerk P Straatsma, Hubertus JJ Van Dam, Dunyou Wang, Jarek Nieplocha, Edoardo Apra, Theresa L Windus, et al. Nwchem: A comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*, 181(9):1477–1489, 2010.

[18] Jae-Yeon Won, Xi Chen, Paul Gratz, Jiang Hu, and Vassos Soteriou. Up by their bootstraps: Online learning in artificial neural networks for CMP uncore power management. In *Proceedings of the symposium on High Performance Computer Architecture, HPCA'14*, pages 308–319, 2014.



(a) slowdown on NOVA

(b) package power on NOVA

(c) slowdown on CHIFFLET

(d) package power on CHIFFLET

(e) slowdown on YETI

(f) package power on YETI

(g) total energy on NOVA

(h) total energy on CHIFFLET

(i) total energy on YETI

Fig. 4: DUF impact on performance, power and energy consumption for a 200ms period measurement