Explicit uncore frequency scaling for energy optimisation policies with EAR in Intel architectures

Julita Corbalan
Comp. Science Dpt. (BSC)
Comp. Science Dpt. (UPC)
Barcelona, Spain
Email: julita.corbalan@bsc.es

Oriol Vidal
Comp. Science Dpt. (BSC)
Barcelona, Spain
Email: oriol.vidal@bsc.es

Lluis Alonso
Comp. Science Dpt. (BSC)
Barcelona, Spain
Email: lluis.alonso@bsc.es

Jordi Aneas
Comp. Science Dpt. (BSC)
Barcelona, Spain
Email: jordi.aneas@bsc.es

Abstract—**EAR is an energy management framework which offers three main services: energy accounting, energy control and energy optimisation. The latter is done through the EAR runtime library (EARL). EARL is a dynamic, transparent, and lightweight runtime library that provides energy optimisation and control. It implements energy optimisation policies that selects the optimal CPU frequency based on runtime application characteristics and policy settings. Given that EARL defines a policy API and a plugin mechanism, different policies can be easily evaluated.**

In this paper we propose and evaluate the utilisation of explicit Uncore Frequency Scaling (explicit UFS) in Intel architectures to increase the energy savings opportunities in the cases where the hardware cannot select the optimal frequency for the Integrated Memory Controller (IMC). We extended the min_energy_to_solution policy to select the CPU and IMC frequencies and we executed and evaluated it with some kernels and six real applications. Results showed an average energy saving of 9% with an average time penalty of 3%. On some use cases, the impact of explicit UFS compared with HW UFS was up to 8% of extra energy savings.

Index Terms—**Energy management, Data centers, energy optimisation, energy models, DVFS, uncore frequency, Intel**

I. Introduction

The increase of new generation processors performance and the race for achieving exascale machines have put on the spotlight power/energy consumption as a factor that high performance computing needs to consider very seriously. The key challenge for green computing is to control/save power and/or energy consumption without reducing performance.

The most employed technique to achieve this goal is through Dynamic Voltage and Frequency Scaling (DVFS) which leads to tuning CPU frequency. Since Haswell micro-architecture [1], processors operate at different frequencies for core components and uncore components (e.g. LLC and DRAM). Intel architectures include a dynamic uncore frequency scaling that automatically adapts the uncore frequency. However, even though the hardware is doing a good job when selecting the uncore frequency, there are use cases where a fine grain selection taking into account the impact on application performance can result in a more energy efficient execution.

The fact Intel architectures expose the tools to explicitly manage the uncore frequency brings an opportunity to save power for some applications which not depend, or are not bound, on uncore performance to get good throughput and also for those applications for which hardware selects a suboptimal uncore frequency in terms of efficiency.

This work has been funded by the BSC-Lenovo collaboration agreement.

EAR is a holistic system software for energy management for data centers. One of the EAR services, among others, is energy optimisation though energy policies executed in the context of parallel applications. Energy policies are dynamically loaded by the EAR Library, and they have been implemented as plugins. All the policies implements the same policy API, being relatively easy to compare energy and performance results with EAR and different policies. In the previous work presented in [2] DVFS applied to the CPU was the technique used by energy policies for energy optimisation, where UFS was enabled by hardware.

This paper shows how we integrate the explicit Uncore Frequency Scaling (we will refer also as eUFS in this paper.) in EAR energy policies in Intel architectures (Skylake in particular). The EAR API for energy policies has been extended to select frequencies for the CPU and Integrated Memory Controller (IMC) scopes. We have incorporated and evaluated the utilisation of eUFS applied to one of the EAR default policies: min_energy_to_solution. The second one, min_time_to_solution, is still under evaluation.

We have extended min_energy_to_solution to apply a state diagram where the CPU frequency is selected in a first stage and then the uncore frequency is selected. Section V presents the different approaches evaluated for uncore frequency selection. This paper presents the min_energy_to_solution algorithm and the extensions to incorporate the eUFS, the new energy models used in this evaluation and the evaluation compared with using no energy policy and with min_energy_to_solution with hardware UFS. Results are presented using DC node power which includes all the sources of power in a node and not only the package (PCK) power, which are the components mostly affected by the uncore frequency.

For the evaluation we executed some kernels and real applications with the uncore frequency enabled and disabled compared with the execution at nominal CPU frequency and UFS automatically done by the HW. We will present the impact on time, average DC node power and energy.

Results show promising energy efficiency benefits when explicitly dealing with the uncore frequency with a limited performance penalty, where more computational intensive applications become ideal cases as they don't get penalized when reducing uncore frequency while we get package and DC node power savings. For memory intensive applications we show that UFS default management done by HW can be improved to get a better ratio between energy savings and time penalty. Results show an average energy saves up to 13.77% with a time penalty up to 2.47% for cpu bound applications and an average energy save up to 11.64% with a time penalty up to 4.95% for the worst case evaluated.

The rest of this work is as follows: in section II we present the starting point by giving a first study on how UFS affects power consumption and performance of some applications; section III introduces EAR and how it works; in IV we collect information of how Intel(R) processors manage IMC frequency; section V contains our approach to join CPU frequency selection with our eUFS for min_energy_to_solution policy followed by the evaluation in VI; finally, we show related work in VII and conclusions and future work in VIII.

II. Motivation

In order to look for opportunities in energy savings, we ran some applications with fixed core and uncore frequencies combinations to see the impact of these parameters on the application performance. We ran tests with different pairs of core and uncore frequency to mimic the case where EAR selects the optimal core frequency based on the energy policy. For details on CPU frequency selection see section III. The goal of this experiment was to check if we could improve the IMC frequency selected by hardware (see section IV) once EAR sets the optimal CPU frequency.

Tests were made using a two socket Intel(R) Xeon(R) Gold 6148 20 core package at 2.4GHz. DRAM memory is DDR4 SDRAM with a maximum frequency of 2.4GHz. Table I shows results for two kernels with the EAR policy *min_energy_to_solution* (see V-B). It shows the average cycles per instructions (CPI) , memory bandwidth (GB/s) , average CPU frequency and also the average IMC frequency selected by hardware across three runs of each application using the policy. We used 160 processes for class C multi-zone Block Tri-diagonal solver (BT-MZ) executed across four compute nodes (40 processes per node, OMP_THREADS = 1) and 2 processes across two compute nodes (one process per node, OMP_THREADS = 40) for class D Lower-Upper Gauss-Seidel solver (LU) , both taken from [3]. In the case of BT-MZ, application signature corresponds with a CPU intensive use case where the policy did not reduce the CPU frequency and therefore the HW selected the maximum IMC frequency. Second case, LU, is more memory intensive, so even though the CPU frequency had been reduced one pstate in average, the HW left the IMC up to the maximum. We can see how, even having clearly different performance profiles, the uncore frequency selected by the hardware has been the same.

This first test gave us the CPU frequency reference and the IMC frequency selected by the HW. Given this reference, we

TABLE I
KERNELS'S METRICS APPLYING EAR'S MIN_ENERGY_TO_SOLUTION POLICY WITH HARDWARE IMC SELECTION

kernel	CPI	GB/s	CPU freq. (GHz)	IMC freq. (GHz)
BT-MZ.C	0.38	10.19	2.38	2.39
LU.D	1.04	75.93	2.31	2.39

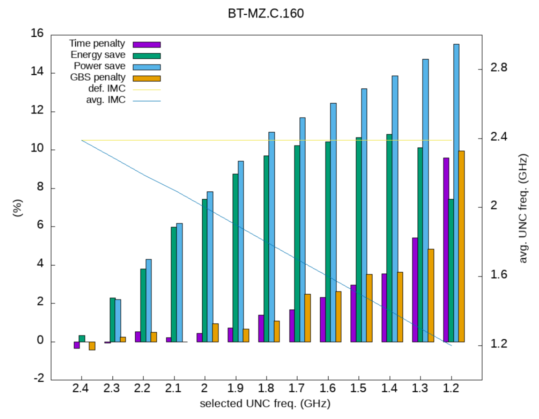
ran again applications with the CPU frequency selected by the policy fixed since the beginning and the IMC frequency set to its default values (2.4GHz - 1.2GHz) , allowing the HW to dynamically set it. This second set of executions gave us the performance and power reference metrics when using a hardware uncore selection. The goal of these graphs was to compare the uncore HW selection vs. an explicit software selection.

To do this comparison, applications were executed again with the same CPU frequency but fixing the uncore frequency by setting values from 2.4GHz to 1.2GHz with 100MHz steps. For each configuration, we also made three runs and computed the average. Figures 1(a) and 1(b) show results for BT-MZ and LU. Each graph contains, respectively, the average DC node power and energy savings and time and GB/s penalties with respect to the average of these metrics across the three runs where we enabled the hardware UFS. The second y-axis shows a constant line which represents the average IMC frequency selected by hardware across these first three runs mentioned for each application, respectively, and also the average IMC for each fixed uncore frequency configuration.

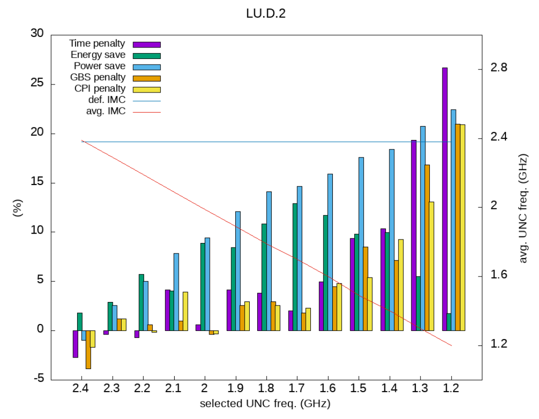
We noted three facts when analyzing the above graphs. The first one is that reducing the uncore frequency step by step brings to more power saving than time penalty. This assertion looks more clear on the first figure which shows results for a less memory dependent kernel than the others, but it's worth mentioning for all figures that at lowest uncore frequencies the time penalty outweighs energy saving. The second fact we noted is that time and memory bandwidth penalties have very closed results for the uncore frequency variations in figure 1(a) while for figure 1(b) we noted that uncore variation also affects CPI performance. This leads us to think about GB/s and CPI penalties as intuitive metrics for selecting a local optimal uncore frequency.

III. EAR: Energy management framework

EAR is a system software for energy management in data centres [2]. EAR offers four main services concerning energy management: Monitoring, Accounting, Control and Optimisation. EAR Library (EARL) is a dynamic, transparent, and lightweight runtime library that provides energy optimisation and control. EARL identifies on the fly the application iterative structure existing in many parallel codes. It does it without any user intervention (without hints, code marks, tags, etc). When the application is MPI, we use our Dynais technology to detect outer loops of the application based on repetitive invocations of MPI calls. When the application is not MPI, EARL is time guided. Anyway, every 10 or more seconds depending



(a) Tests results for BT-MZ kernel. CPU frequency set to 2.4GHz.



(b) Tests results for LU kernel. CPU frequency set to 2.3GHz.

Fig. 1. Average time penalty, average DC node power and energy save, memory bandwidth penalty for different fixed values of uncore frequency with respect to execution with default uncore frequency range for BT-MZ and Lu kernels.

on the application and the architecture, EARL computes the application loop signature. The duration of the signature computation depends on the application and HW facilities to measure power. As we use average DC node power[1], typically measured with IPMI commands, it cannot be measured at high frequencies. Energy readings to compute power have been done every 10 seconds [2]. The loop (or application) signature is a set of performance and power metrics characterising application computational behaviour. This signature is then used by the energy policy applied by EARL to select the optimal frequency. Up to this paper, EARL have been only dealing with CPU frequency. In this paper we extended one of the energy policies included with EAR by default to select the CPU and IMC frequencies. The policy selected for this paper has been min_energy_to_solution. The idea of this policy is to minimise the energy by applying DVFS with a limit in the performance degradation suffered by the application. Section V describes the proposed min_energy_to_solution extension and it also includes the CPU frequency selection algorithm used till now.

IV. Uncore frequency management in Intel architectures

Haswell-EP was the first release which incorporated UFS, an independent frequency management of uncore components. Previous Intel(R) processor generations such as Nehalem-EP and Westmere-EP have worked at fixed uncore frequency, while Sandy Bridge-EP and Ivy Bridge-EP have used a common frequency for cores and uncore [4]. The IMC frequency can be guided through writing on the Model Specific Register (MSR) 0x620H called UNCORE_RATIO_LIMIT provided by [5]. This register lets the user specify the minimum and maximum values for uncore frequency through bits 14-8 bits and 6-0 respectively, so hardware can then apply a multiplier to get the operational frequency for each socket. In order disable the default UFS mechanism offered by hardware the user may put specific values for the two ranges of bits mentioned and therefore the same value for both parts if one wants to set a fixed IMC frequency. The available uncore frequency range that a specific hardware offers can be read from this MSR register after the boot.

By default, the hardware implements a control loop which controls the uncore frequency based on the workload on the current socket and the information given by the UNCORE_RATIO_LIMIT register. Authors of [4] state that another factor to take in consideration is the *Performance and Energy Bias Hint* (EPB) , which serves as a hardware heuristic for power management features [5]. According to Intel's patent [6] and results showed in [4], uncore frequency selection depends on the fastest active core frequency, and in [7] is stated that it takes an average of 10 ms to detect workload patterns changes and therefore adapt uncore frequency.

V. EAR energy policies with explicit UFS

CPU frequency selection is based on the application signature, the time and power predictions are made using the energy model and the policy settings (specific for each policy). Each policy predicts the time and power for each CPU frequency using the models and selects the *optimal* CPU frequency, were optimal depends on the policy. The utilisation of energy models and signatures computed at runtime makes possible EARL selects CPU frequency after few seconds of execution and can be re-applied dynamically each time the signature changes.

The application signature includes performance and power metrics. The ones used by the energy models are: DC node power, Iteration time, CPI, TPI (main memory Transactions per Instruction) and the percentage of AVX512 instructions compared with the total number of instructions (VPI).

[1]Computed using the accumulated energy.

[2]In this paper we have used Intel Node Manager for energy readings. INM offers an energy counter updated every 1s.

A. *EAR AVX512 energy model*

The energy model is a new one based on the default model used in [2]. The default model was proposed in [8] and [9]. It has been extended to take into consideration the new types of AVX instructions and their relationship with CPU frequency. In this paper we have only considered AVX512 (and not AVX2) instructions because AVX2 maximum frequency was higher than the maximum frequency for non-AVX instructions (when turbo was not used).

The new functions for predicting time and power receives three inputs: the signature, and the source (from) and target (to) pstates. Source is the CPU pstate at which the signature has been computed and target is the CPU pstate we want to predict time and power. The new model combines two predictions each: (1) the one generated with the requested `to` CPU pstate (`default_pred`) and (2) one were the target pstate has been limited based on the maximum pstate supported when all cores are running AVX512 instructions (`avx512_pred`). Finally, time and power predictions are generated by combining the two predictions (default_pred and avx512_pred) weighted by VPI. For example, in the Intel(R) Xeon(R) 6148 used in the evaluation, the nominal CPU frequency is 2.4GHz and the maximum CPU frequency for AVX512 when all the cores are running is 2.2GHz, corresponding with pstate 3. This model captures the fact that AVX512 instructions will not take benefit of higher CPU frequencies.

B. *Min energy to solution: CPU and explicit uncore frequency selection*

The min_energy_to_solution basic algorithm is a linear search. The algorithm selects the CPU pstate with the minimum energy where the predicted time penalty is below the limit (`limit = time *(1 + cpu_policy_th)`). The cpu_policy_th is a policy argument and it can be either specified at runtime or by default by the sysadmin. In the evaluation of this paper, we have used 3% and 5% as cpu_policy_th values.

EARL implements a state diagram to deal with application phases, signature computation, policy validation, etc. Code 1 shows a simplified version of the function executed each time a signature is computed. EARL re-applies the energy policy until the policy state returned is READY. This allows the implementation of iterative policies as it is this case. Policy symbols are dynamically loaded in a policy_operations data structure with function pointers. The policy API is not included in this paper but several application lifetime events are captured to invoke policy functions. These events include, among others, start/end of the application, loop, mpi call and the signature computation.

```
state state_new_signature(sig_t *sig)
{
  node_freqs_t nextf;
  state next = READY;
  switch (ear_state){
  case NODE_POLICY:
    next = policy_ops.node_policy(sig,nextf);
    if (next == READY){
      set_freqs(nextf);
      ear_state = VALIDATE_POLICY;
    }
    break;
  case VALIDATE_POLICY:
    ok = policy_ops.validate(sig);
    if (!ok){
      ear_state = NODE_POLICY;
      policy_ops.set_def(nextf);
      set_freqs(nextf);
    }
  }
  return ear_state;
}
```

Listing 1. EAR state diagram

The basic min_energy_to_solution basic algorithm has been extended and it is now the first stage of a more complex policy. The policy implements the state diagram shown in figure 2. The first time the application is executed, it enters in the CPU_FREQ_SEL state and selects the CPU frequency applying the basic min_energy_to_solution algorithm. Depending on the selected CPU frequency, it goes to the COMP_REF state, which is basically an intermediate state to compute the reference metrics to the next state. In case the CPU frequency selected is the default frequency for min_energy_to_solution (the maximum frequency), the policy goes to the IMC_FREQ_SEL state directly. The policy enters N times into this state until it reaches the optimal IMC frequency.

We evaluated two approaches in this search: a linear search and HW guided search. The first one sets the IMC frequency to the maximum value and starts the search from this. The second one lets the initial IMC settings to be decided by the HW and then uses this value as reference. This second strategy is faster than the first one because even though the HW is applying a conservative strategy, it is faster than starting from the maximum. In the evaluation section we will include only results for the second strategy except in one of the cases to show the impact of using HW guide.

The uncore configuration supports to specify the maximum and the minimum frequency, so different alternatives could be applied such as setting max and min to the same values, defining a given range (0.1 GHz for example) between max and min, or reducing only the maximum, being possible to be reduced by the HW in case the application enters in a different phase. We have done a pre-evaluation of the proposal (not included in the paper) and decided to just move the maximum uncore frequency and reduce it by 0.1GHz each try.

In order to control the impact of the uncore frequency, the policy settings have been extended with a second threshold: the unc_policy_th. This value is the percentage of extra performance penalty supported because of the uncore frequency scaling. This percentage is a limit on the supported CPI and GB/s variation. If current CPI is greater than the reference CPI + extra_CPI (where extra_CPI is CPI*unc_policy_th) or current GB/s is bellow reference GB/s - extra_GB/s (where

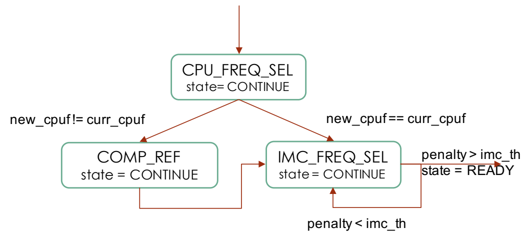


Fig. 2. Min_energy_to_solution state diagram

extra_GB/s is GB/s*unc_policy_th), the last uncore frequency selected is reverted and the policy returns a READY state to the EAR Library, otherwise the uncore frequency is reduced by 0.1GHz and it returns a CONTINUE state.

When a policy returns with a READY state, the EAR Library changes its internal to become stable (applies the same CPU and IMC frequencies) until a significant change is detected in the signature. In this work, we accept up to 15% of variations in the signature before re-applying the energy policy again. When a policy returns CONTINUE, it is executed again once the signature is recomputed. To summarize the min_energy_to_solution extensions:

1) It uses a new energy model which takes AVX512 instruction characteristics into account.
2) IMC frequency selection starts on HW selection.
3) We change the maximum but not the minimum IMC frequency.
4) An additional threshold is used to limit the performance penalty.
5) CPI and GB/s are used to determine whether the IMC is the optimal.
6) CPI and GB/s are used to determine signature changes.

As min_energy_to_solution has become an iterative policy, we have introduced an additional check to detect cases where the signature changes during the IMC frequency selection. If that happens, the policy state is set again to CPU_FREQ_SEL. This situation is not usual, but it could happen.

VI. Evaluation

We have evaluated two types of applications: single node kernels and MPI real applications using multiple nodes.

A. Kernels

Table II shows the list of kernels executed. BT-MZ and SP-MZ are the OpenMP implementation of the NAS parallel benchmarks [3]. CUDA versions have been downloaded from [10]. DGEMM is from the [11]. For OpenMP and MKL kernels 40 threads are used whereas for CUDA kernels one thread and one GPU are used. Columns show, respectivelly, the application name, the programming model, the execution time in seconds, the CPI and GB/s and the average DC node power.

Kernels and applications have been executed in a cluster of Lenovo ThinkSystem SD530 nodes where each node includes two Intel(R) Xeon(R) Gold 6148 CPU @ 2.40 GHz (20c)

TABLE II
Single node kernels

kernel	Prog. Model	Time (s)	CPI	GB/s	Avg.DC Power(W)
BT-MZ.C	OpenMP	145	0.39	28	332
SP-MZ.C	OpenMP	264	0.53	78	358
BT.D	CUDA	465	0.49	0.09	305
LU.D	CUDA	256	0.54	0.19	290
DGEMM	MKL	160	0.45	98	369

per node (Hyper-threading is activated, but we are not using it.) , 40 cores in total and 12 * 8GB dual rank DIMMs per node. Maximum/minimum uncore frequencies for this system are 2.4 GHz - 1.2 GHz respectively. For all the experiments, three runs have been executed, and we are using the average of all three. For a fair comparison, all the executions for each application have been done using the same set of nodes.

For the particular case of CUDA kernels, a Intel(R) Xeon(R) Gold 6142M CPU @ 2.60 GHz (16C) with two NVIDIA Tesla V100 per node at 1.38GHz. CentOS Linux release 8.2.2004 is installed and intel OpenMp version 2020.4.304 was used for OpenMP, CUDA 10.1 and intel mkl version 2020.4.304 for dgemm. Uncore frequency limits are the same (2.4 GHz - 1.2 GHz). However, even though the nodes have two GPUS, the kernels use only one and one single core that is mostly running a busy waiting code. The power of the second GPU is automatically reduced by the NVIDIA driver both in the execution at nominal frequency and with min_energy_to_solution.

Table III shows the performance penalty, power saving and energy saving when running the kernels with min_energy_to_solution policy with the uncore frequency selection done by the hardware (ME) and min_energy_to_solution with explicit uncore frequency selection done by EAR (ME+eU). We show performance metrics compared with the default values when the applications are executed at the nominal frequency (2.4 GHz for non-CUDA applications and 2.6GHz for the CUDA ones). In the case of CUDA apps, the savings comes from the fact the CPU is doing a busy waiting while the GPU is computing. EAR detects the uncore frequency can be reduced without affecting the application performance given the main memory activity is very low. All these experiments have been done with cpu_policy_th set to 5% and unc_policy_th set to 2%.

Results show there is window for improvement in energy savings by doing a fine grain uncore frequency selection additional to the HW selection. We can see how EAR is able to get additional energy savings by reducing the uncore frequency without penalising execution time.

To fully understand results, table IV shows the average node CPU frequency[3] and the average node IMC frequency in the three use cases executed: No policy (nominal frequency used in both CPU and GPU) , min_energy_to_solution with UFS done by the hardware and min_energy_to_solution with

[3]Remember CUDA kernels use only one core.

TABLE III
SINGLE NODE KERNELS EVALUATION

Application	Time penalty		Power saving		Energy Saving	
	ME	ME+eU	ME	ME+eU	ME	ME+eU
BT-OMP.D	0%	1%	0%	8%	0%	7%
SP-OMP.D	1%	0%	0%	8%	-1%	8%
BT.CUDA.D	0%	0%	10%	11%	10%	11%
LU.CUDA.D	0%	0%	0%	5%	0%	5%
DGEMM	0%	0%	0%	2%	0%	1%

TABLE IV
AVG CPU AND IMC FREQUENCY DOMAINS

Kernel	Dom	No policy	ME	ME+eU
BT-MZ.C	CPU	2.38	2.38	2.38
	IMC	2.39	2.39	1.98
SP-MZ.C	CPU	2.38	2.38	2.38
	IMC	2.39	2.39	2.08
BT.CUDA	CPU	2.44	2.28	2.13
	IMC	2.39	1.51	1.30
LU.CUDA	CPU	2.02	2.01	2.05
	IMC	2.39	2.39	1.60
DGEMM	CPU	2.18	2.19	2.19
	IMC	1.98	1.95	1.87

eUFS (done by EAR). The average is computed using all the cores and the whole kernel execution. In this case the table shows absolute values. We can see how in the OpenMP cases the CPU frequency selected by the min_energy_to_solution policy is not reduced given the limit on the cpu_policy_th, however, the new stage of the policy is able to reduce the uncore frequency without penalty in all the cases. We can see how the CPU frequency is adapted in all the cases (with min_energy) to CPU requirements and how the eUFS adapts the uncore frequency to memory requirements. The case of DGEMM is a bit different because this kernel has a VPI of 100% therefore the CPU frequency is automatically reduced by the hardware (and the uncore frequency), so the eUFS done by EAR is only applying a minor adjustment (from 1.98 to 1.87 in average).

B. Applications

For this second part of the evaluation we have used the following applications and use cases.

- BT-MZ class D. Block Tri-diagonal solver from the NAS-PB [3]. We used 160 MPI processes, four nodes.
- BQCD. Berlin quantum chromodynamics program [12] is a Hybrid Monte-Carlo program for simulating lattice QCD with dynamical Wilson fermions. We used 40 MPI processes, four threads per task in four nodes.
- GROMACS. GROningen MAchine for Chemical Simulations [13] is a molecular dynamics package mainly designed for simulations of proteins, lipids and nucleic acids. In this work we present two different inputs, (I) *ion_channel* with 160 MPI processes equally distributed within four nodes, and (II) *lignocellulose-rf* with 640 MPI processes within 16 nodes.
- HPCG [14]. The High Performance Conjugate Gradients (HPCG) Benchmark project is an effort to create a new metric for ranking HPC systems. HPCG is designed to exercise computational and data access patterns that more closely match a different and broad set of important applications, and to give incentive to computer system designers to invest in capabilities that will have impact on the collective performance of these applications.
- POP [15] is the open source Parallel Ocean Model version 2 developed by Los Alamos National Lab. We used 384 MPI processes, 10 nodes.
- DUMSES [16] is a 3D MPI+OpenMP & MPI/OpenACC Eulerian second-order Godunov (magneto)hydrodynamic simulation code in cartesian, spherical and cylindrical coordinates. We used 512 MPI processes, 13 nodes.
- AFiD. AFiD is a highly parallel application for Rayleigh-Benard and Taylor-Couette flows. It is developed by Twente University, SURFsara and University of Rome ”Tor Vergata” [17]. We used 576 MPI processes, 15 nodes.

In this section we present different use cases for applying EAR's eUFS in conjunction with min_energy_to_solution policy: Eight applications which can be divided into two classes: cpu bound (e.g. BQCD, the two GROMACS configurations and BT-MZ) and memory bound (e.g. HPCG, POP, DUMSES and AFiD) applications.

As stated for kernels evaluation, we present per-application characteristics and average nodes CPU and IMC frequencies in tables V and VI, respectively. In all the cases, we show performance penalty (the lower the better) and energy and power savings (the higher the better). Results compare the execution with min_energy_to_solution with the different cpu_policy_th and unc_policy_th compared with the metrics when running at the nominal CPU frequency and the IMC selected by the hardware. Power and energy are evaluated with the DC node power measured with the Intel Node Manager.

All the applications have been executed with a cpu_policy_th of 5% except BQCD, where a cpu_policy_th of 3% was used because this application gets more penalized in terms of energy save.

By default, an unc_policy_th of 2% was used except in BQCD and BT-MZ where different threshold have been shown to demonstrate the effect of this parameter. Cases evaluated are (except the specific thresholds):

1) ME: Min_energy_to_solution where the uncore frequency is selected by the hardware.
2) ME+eU: Min_energy_to_solution where the uncore frequency is selected by EAR.

Figure 3 shows graphically BQCD's savings and penalties. These configurations are the policy with HW UFS (e.g. ME) and the policy with eUFS with different unc_policy_th (e.g. ME+eU 1%, ME+eU 2% and ME+eU 3%). As the policy doesn't reduce core frequency, results for ME configuration don't show any saving, so the rest of configurations show the effect of reducing uncore frequency in terms of DC node power save and time penalty and the resulting energy save. Note that power saving scales better than time penalty. Figure

TABLE V
MPI APPLICATIONS

kernel	Time (sec)	CPI	GB/s	Avg.DC Power(Watts)
BQCD	130.54	0.68	10.98	302.15
BT-MZ	465.01	0.38	6.60	320.74
GROMACS (I)	313.92	0.48	10.39	319.35
GROMACS (II)	390.60	0.63	13.34	315.48
HPCG	169.61	3.13	177.45	339.88
POP	1533.03	0.72	100.66	347.18
DUMSES	813.21	1.08	119.07	333.69
AFiD	268.22	0.77	115.20	333.65

TABLE VI
AVG CPU AND IMC FREQUENCY DOMAINS

Application	Dom	No policy	ME	ME+eU
BQCD	CPU	2.38	2.37	2.38
	IMC	2.39	2.39	2.19
BT-MZ	CPU	2.38	2.38	2.38
	IMC	2.39	2.39	1.79
GROMACS (I)	CPU	2.28	2.27	2.27
	IMC	2.39	2.04	1.91
GROMACS (II)	CPU	2.29	2.27	2.27
	IMC	2.39	1.45	1.41
HPCG	CPU	2.38	1.75	1.73
	IMC	2.39	2.39	2.29
POP	CPU	2.38	2.23	2.23
	IMC	2.39	2.35	2.06
DUMSES	CPU	2.38	2.12	2.12
	IMC	2.39	2.39	2.13
AFiD	CPU	2.38	2.2	2.22
	IMC	2.39	2.35	2.17

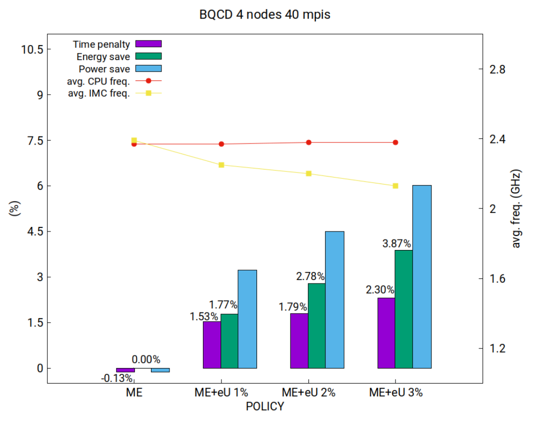


Fig. 3. Average time penalty, DC node power and energy save for BQCD applying different *min_energy_to_solution* (ME) configurations.

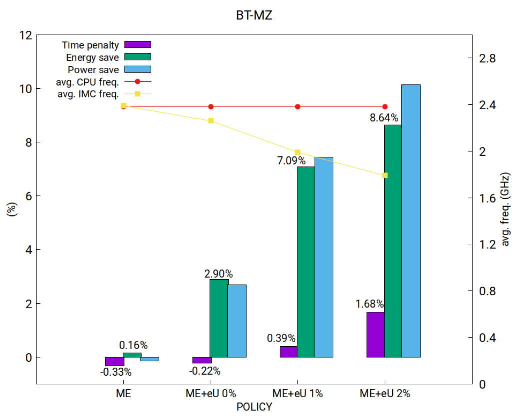


Fig. 4. Results comparing average time penalty, DC node power and energy save for BT-MZ applying *min_energy_to_solution* policy (ME) and EAR's UFS (ME+eU) configurations with a cpu_policy_th value of 3%.

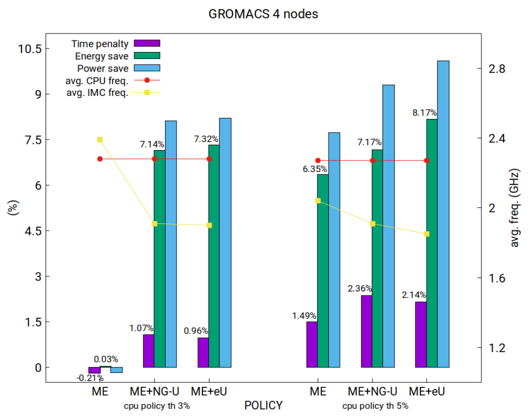


Fig. 5. Average time penalty, DC node power and energy save for GROMACS(I) applying different *min_energy_to_solution* (ME) configurations with a cpu_policy_th value of 3% and 5%.

4 shows same behaviour for BT-MZ, but we show results for applying unc_policy_th values from 0% to 2%. First case (e.g. unc_policy_th to 0%) shows that we can reduce uncore frequency without reducing per-iteration execution time (See section III) and we are still saving power.

Figure 5 shows results for GROMACS(I) when executed with cpu_policy_th 3% and 5% respectively (both cases with unc_policy_th set to 2%). In this case, we show the impact of selecting the uncore frequency without the hardware *guide* (e.g. Not-Guided Uncore, ME+NG-U) or using it as the starting point for the algorithm (ME+eU). This second strategy is the one used by default but we have included this case here to show the benefits of this solution. Note that adding EAR's UFS policy (e.g. ME+NGU and ME+eU) leads to improve results of applying a HW UFS policy (e.g. ME) giving an energy save up to 7.32% for a cpu_policy_th of 3% and 8.17% for a cpu_policy_th of 5%, which leads to get savings up to 7 and 3 times greater than time penalties, respectively. This figure also shows that a less restrictive DVFS policy (e.g. cpu_policy_th value of 5%) brings more energy savings than a more restrictive one at the cost of getting more time penalty due to reducing both CPU and IMC frequencies.

Figure 6 shows a simplified graph where we put results for applying ME policy with a cpu_policy_th and unc_policy_th values of 5% and 2%, respectively, on GROMACS(II). In this case, EAR's uncore frequency selection on ME+eU has been the same than the hardware's UFS applied on ME test, but we could improve energy savings as EAR's UFS mantains the selected uncore frecuency fixed after HW's selection.

In a similar fashion we present results for HPCG and POP in figures 7(a) and 7(b), respectively. On the former

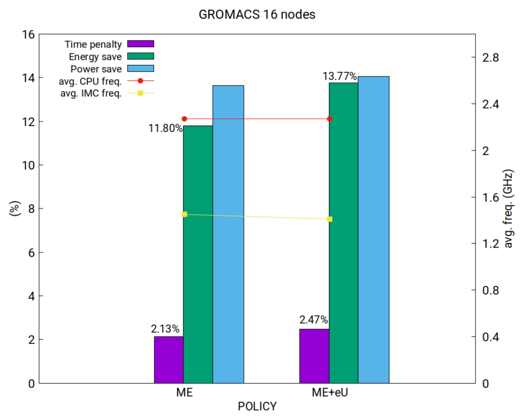


Fig. 6. Average time penalty, DC node power and energy save for GROMACS(II) applying different *min_energy_to_solution* (ME) configurations with a policy_cpu_th value of 5%.

ME+eU doesn't improve ME's efficiency ratio (e.g. 3.5 vs. 4.76, respectively) as this application is the most memory bound we present in this work, but leads to improve energy saving if a time penalty up to 3.33% is tolerated. The latter shows a ratio improvement up to 2.31.

Finally, we show a comparison of applying EAR's eUFS using different cpu_policy_th values for DUMSES and AFiD applications in figures 8(a) and 8(b), respectively. Figures show how EAR offers flexibility by giving to the user two threshold values which will be related to the final execution time of the application. In both cases it can be seen that different combinations can be chosen depending on whether the user wants to optimise the efficiency ratio between energy save and time penalty or wants to improve the energy savings get by only applying DVFS policy. For example, in figure 8a as the average core frequency remains the same for both ME and ME+eU tests, the ratio gets improved in the two cpu_policy_th cases. On the other hand, figure 8b shows that AFiD lost CPI when applying ME+eU tests, but note that the average core frequency for ME+eU tests with policy_cpu_th of 3% was very closed to ME tests applying a policy_cpu_th value of 5%, but in the first case, as EAR had applied its UFS, the resulting energy savings were better than the latter.

In this section we presented different use cases for applying EAR's UFS in conjunction with min_energy_to_solution policy. We showed eight applications which can be divided into two classes: cpu bound (e.g. BQCD, the two GROMACS configurations and BT-MZ) and memory bound (e.g. HPCG, POP, DUMSES and AFiD) applications. For those in the first class, we have seen that reducing uncore frequency leads to save power and energy without significantly affecting the execution time and we have even managed to obtain savings opportunities for those cases where the DVFS policy does not reduce the core frequency to prevent the application from being affected in terms of performance. For the second one class, we showed how we can improve energy savings offered by DVFS policy applying EAR's eUFS where in most cases the average uncore frequency remained one or two MHz below default hardware selected frequency.

TABLE VII
DC NODE POWER SAVINGS VS. RAPL PCK POWER SAVINGS

Application	DC Node Power	RAPL PCK power
BQCD	4.69%	10.56%
BT-MZ	10.15%	15.03%
GROMACS (II)	14.06%	15.65%
HPCG	14.49%	16.88%
POP	10.25%	13.37%
DUMSES	13.13%	15.43%
AFiD	12.02%	13.37%

Finally, we have compared the average DC node power variation and the average RAPL PCK (package) power when using the ME+eU policy. All the papers in the related work (see VII) use the RAPL PCK power as reference to evaluate energy and power savings as well as for developing their controller-based policy like [18], [19]. The PCK power is a non constant percentage of the DC node power, this is the reason why we think we must evaluate energy policies with DC node power. Table VII shows the DC and PCK power savings when running with ME+eU with the cpu_policy_th 5% and unc_policy_th 2% compared with the nominal CPU frequency and hardware UFS. As we can observe, the difference is not constant. Moreover, these values are percentages but the same behaviour, even more, can be observed in absolute values. Using PCK rather than DC node power could lead us to incorrect conclusions in terms of savings.

VII. Related Work

There exist several studies about UFS in order to get energy savings on the execution of HPC applications without considerable losses in execution time. [1] presents a survey of architectural features among different generations of Intel processors and show how Uncore clock speed has an impact on the power consumption of memory (and L3 cache) bandwidth applications such as LINPACK and HPCG benchmarks. [7] is a study about energy efficiency and performance impact features of new Intel(R) Skylake-SP architectures. The most relevant to what we are dealing with is that it shows hardware latencies on UFS changes and AVX frequencies to adapt itself to application behaviour changes. Authors previously published a similar study with Haswell-EP micro-architecture in [4]. Remain works that play with UFS can be divided in two main categories: model-based and controller-based strategies.

First ones create models based on the impact of Uncore frequency on performance and power consumption. In [20] search trees are used to find the optimal Uncore frequency values for SPEC CPU2017 benchmark suite on Intel Xeon Processor E5-2620 v4 server. Authors build a model based on apps' cache characteristics taking into account that multiple apps are running at the same time on the same core, and make experiments executing programs for 30 seconds. This approach predicts optimal Uncore frequency but not Core frequency, and the predictor is executed once before the application execution, while our approach is based on changing CPU and IMC frequencies dynamically based on execution time

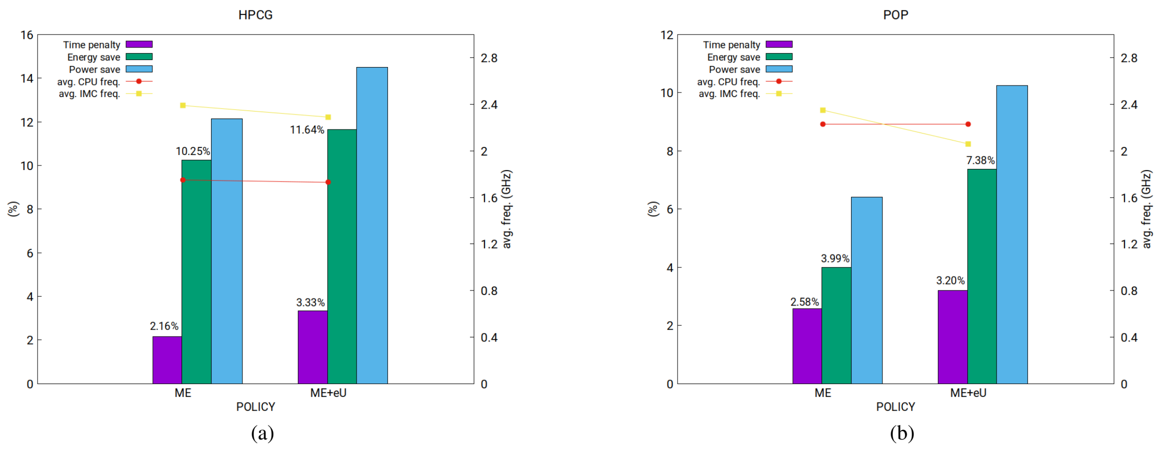


Fig. 7. Results comparing average time penalty, DC node power and energy save for HPCG (a) and POP (b) applying *min_energy_to_solution* policy (ME) and EAR's UFS (ME+eU) configurations with a policy_cpu_th value of 5%.

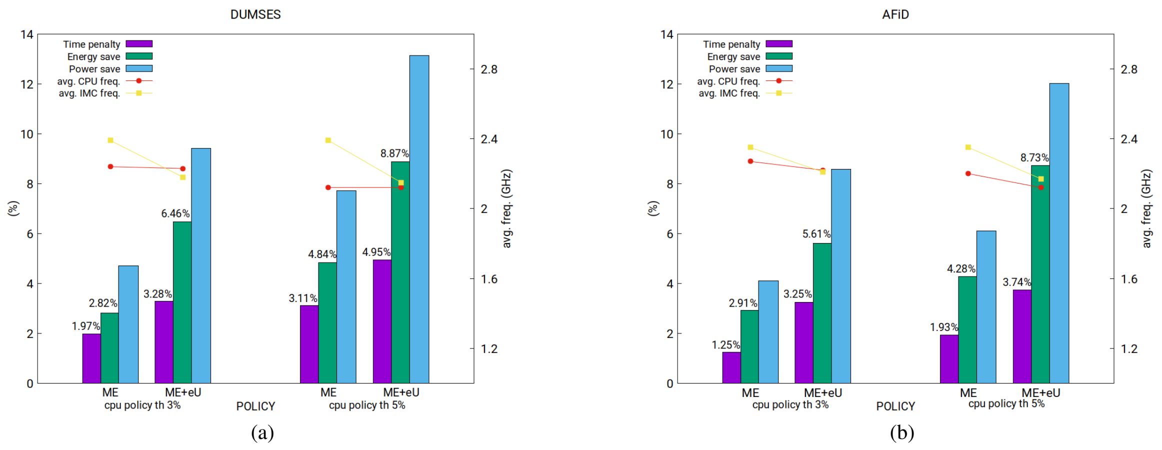


Fig. 8. Results comparing average time penalty, DC node power and energy save for DUMSES (a) and AFiD (b) applying *min_energy_to_solution* policy (ME) and EAR's UFS (ME+eU) configurations with policy_cpu_th values of 3% and 5% and policy_imc_th value of 2%.

metrics. Authors of [21] develop a tuning plugin for Periscope Tuning Framework that selects optimal OpenMP threads and build a neural network to select the optimal CPU and IMC frecuencies. They also make comparisons for static and dynamic tuned applications from NPB, CORAL, Mantevo and LLCBench benchmark suites and the real world application BEM4I on two Intel Xeon E5-2680v3 processors with 12 cores each. Finally, [22] build a model to evaluate the impact of Core and Uncore frequency on power consumption and performance loss on a 20 core Haswell-EP platform using class C NAS benchmarks and the real-world application GAMESS for evaluating the proposed work. Authors also compare in [23] the quality of their strategies based on only controlling DVFS and joining it with UFS in GAMESS.

Those in the second category aim to find local optimal Uncore frequency by changing it during execution time taking as reference the impact of Uncore frequency changes on some metrics. Controllers consist of two principles. First, based on the fact that a reduction of the uncore brings power savings without significantly decreasing performance, they try to lower the uncore. Then with metrics obtained from this uncore change they decide whether this change has achieved the expected effect and decide whether to keep lowering it, keep it, or raise it. In [18] authors monitor DRAM power to detect phase changes and IPC to detect performance degradation, as they relate these metrics to memory intensive and cpu intensive phases, respectively. They show results for some class C NPB and class D ECP proxy applications suites, running up to 16 Broadwell with two Intel Xeon CPU E5-2620 v4 @ 2.10GHz with eight cores each. Authors of [19] find a similar approach but this time they use memory bandwidth and FLOPS as metrics for their controller, but also checks for L3 cache bandwidth changes to detect phase changes. They also compare their result with the last mentioned [18], getting similar results on power savings but improving performance penalty on Broadwell and Skylake microarchitectures, using class C for most of NPB on first testbed and class D for the second one. They also make tests with High Performance

Linpack and LAMMPS and Nwchem.

All works presented use RAPL for power performance accounting. This paper shows how we integrate a controller-based approach to manage UFS in EAR framework. We present an algorithm here which coexists with EAR's min_energy_to_solution policies, trying to be a counterpart of it to improve energy savings in terms of DC node power and per-application execution time. For MPI applications where EARL is guided by Dynais our method can manage uncore frequency with direct knowledge of time penalty. On the other hand, when EARL policy is called periodically (e.g. no MPI application) , our UFS policy lets to be parametrised by the metric the user wants to limit its penalisation, which will be correlated with the performance. We also take into account the limitations imposed by hardware when selecting the uncore frequency.

VIII. Conclusions and future work

This paper has presented an extension on the EAR Library to include the uncore frequency selection on energy policies and the integration of this new option in the min_energy_to_solution policy. This feature has been evaluated on Skylake architectures with different types of applications. Results have demonstrated there are three main sources of energy savings: CPU bound applications where the CPU frequency is the nominal (or turbo) , memory bound applications and CPU bound where the average CPU frequency is lower than nominal.

After considering different strategies, we have selected a HW-guided strategy as the best one, using HW uncore frequency selection as the starting point for our search algorithm. CPI and GB/s metrics variation have been selected as the reference to identify the additional penalty introduced by the uncore frequency reduction. Results have shown a maximum benefit of 13.77% of energy savings when using a software uncore frequency selection. The configuration used by default in this paper have been done with a maximum penalty of 5% for the CPU frequency selection and an additional 2% for the uncore. However, the maximum performance penalty measured has been 4.95%. The average energy savings and time penalty have been 8.75% (with seven out of eight applications having more than 10%) and 2.91%, respectively.

As a future work, we are integrating the same strategy in the other policy included with EAR, min_time_to_solution. In this case, we are also considering additional strategies such as increasing the uncore frequency. We are also evaluating the potential impact on high communication intensive applications.

Acknowledgment

Authors want to thank the SURF Open Innovation Lab, in particular Dr Axel Berg and Sagar Dolas with their help in evaluating POP and AFiD and also to Marc Joos for his help with DUMSES application.

References

[1] J. Hofmann, G. Hager, G. Wellein, and D. Fey, "An analysis of core- and chip-level architectural features in four generations of intel server processors," *High Performance Computing*, p. 294–314, 2017. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-58667-0_16

[2] J. Corbalan, L. Alonso, J. Aneas, and L. Brochard, "Energy optimization and analysis with ear," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2020, pp. 464–472.

[3] D. Bailey, T. Harris, W. Saphir, R. Van Der Wijngaart, A. Woo, and M. Yarrow, "The nas parallel benchmarks 2.0," Technical Report NAS-95-020, NASA Ames Research Center, Tech. Rep., 1995.

[4] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," in *2015 IEEE international parallel and distributed processing symposium workshop*. IEEE, 2015, pp. 896–904.

[5] *Intel(R) 64 and IA-32 Architectures Software Developer's Manual*, Intel Corporation, Apr 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/download/

[6] "Dynamically controlling interconnect frequency in a processor," U.S. Patent US9 323 316B2, 2016.

[7] R. Schöne, T. Ilsche, M. Bielert, A. Gocht, and D. Hackenberg, "Energy efficiency features of the intel skylake-sp processor and their impact on performance," in *2019 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2019, pp. 399–406.

[8] R.Bell, L. Brochard, D. DeSotta, R. Panda, and F.Thomas, "Energy-aware job scheduling for cluster environments," U.S. Patent US8 527 997B2, 2011.

[9] L. Brochard, R. Panda, D. DeSota, F. Thomas, and R. H. Bell Jr, "Power and energy-aware processor scheduling," in *Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering*, 2011, pp. 227–234.

[10] Npb-cuda web page. [Online]. Available: https://www.tu-chemnitz.de/informatik/PI/sonstiges/downloads/npb-gpu/index.php.en

[11] Intel® math kernel libray. [Online]. Available: https://software.intel.com/en-us/mkl/documentation/code-samples

[12] The BQCD website. [Online]. Available: https://www.rrz.uni-hamburg.de/services/hpc/bqcd/

[13] The GROMACS website. [Online]. Available: http://www.gromacs.org

[14] Hpcg benchmark. [Online]. Available: https://www.hpcg-benchmark.org/

[15] Parallel ocean program. [Online]. Available: http://www.cesm.ucar.edu/models/ccsm4.0/pop

[16] The DUMSES website. [Online]. Available: https://github.com/marcjoos-phd/dumses-hybrid

[17] E. P. Van Der Poel, R. Ostilla-Mónico, J. Donners, and R. Verzicco, "A pencil distributed finite difference code for strongly turbulent wall-bounded flows," *Computers & Fluids*, vol. 116, pp. 10–16, 2015.

[18] N. Gholkar, F. Mueller, and B. Rountree, "Uncore power scavenger: A runtime for uncore power conservation on hpc systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, vol. 27. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–23. [Online]. Available: https://doi.org/10.1145/3295500.3356150

[19] E. André, R. Dulong, A. Guermouche, and F. Trahay, "DUF : Dynamic Uncore Frequency scaling to reduce power consumption," Feb 2020, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02401796

[20] 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)*. IEEE, 2019, pp. 1–12.

[21] M. Chadha and M. Gerndt, "Modelling dvfs and ufs for region-based energy aware tuning of hpc applications," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 805–814.

[22] V. Sundriyal, M. Sosonkina, B. Westheimer, and M. Gordon, "(2018) core and uncore joint frequency scaling strategy," *Journal of Computer and Communications*, vol. 6, pp. 184–201, 2018.

[23] V. Sundriyal, M. Sosonkina, B. M. Westheimer, and M. Gordon, "Comparisons of core and uncore frequency scaling modes in quantum chemistry application gamess," in *Proceedings of the High Performance Computing Symposium*, ser. HPC '18, vol. 13. San Diego, CA, USA: Society for Computer Simulation International, 2018, p. 11.