Reactive DVFS Control for Multicore Processors

Jean-Philippe Halimi*, Benoît Pradelle*, Amina Guermouche*, Nicolas Triquenaux*,
Alexandre Laurent*, Jean Christophe Beyler†, William Jalby*

*Université de Versailles Saint-Quentin-en-Yvelines.
first.last@uvsq.fr
†Intel Corporation.
jean.christophe.beyler@intel.com

***Abstract*—Several solutions are considered to reduce energy consumption of computers. Among them, Dynamic Voltage and Frequency Scaling (DVFS) emerged as an effective way to enhance energy efficiency by adapting processor frequency to workloads.**

We propose FoREST, a new DVFS controller designed to efficiently exploit the recent technologies introduced in processors. FoREST is a runtime DVFS controller able to estimate the energy savings it can achieve from power gains, evaluated offline using power probes embedded in modern CPUs, and speedups measured at runtime for the current workload. It does not use any performance model but rather directly measures the effect of frequency transitions on energy. Using such methodology, FoREST can achieve energy savings on the whole system under user-defined slowdown constraints.

In our experiments, FoREST is able to achieve more than 39% CPU energy savings compared to the default Linux DVFS controller, with a slowdown below 5%, as requested by the user.

I. Introduction

Energy is now considered as one of the major research topics for computer science. Indeed, several concerns regarding energy consumption have recently been raised and, among other ecological or technical issues, the growing cost of energy is now a strong motivation for reducing energy consumption in computer science.

In computers, energy is consumed by several distinct hardware components such as processors, memory, and fans among others. However, one can often observe that processors consume a major part of the total power [1]. Then, if a target must be designated for energy optimization, the CPU is probably one of the most interesting component.

Recent processors are well equipped regarding energy saving as they integrate several energy-friendly technologies. One of them, Dynamic Voltage and Frequency Scaling (*DVFS*) allows the user to control the chip frequency and input voltage in order to reduce power consumption. DVFS is integrated as SpeedStep on Intel processors [2], or as Cool'n'Quiet on AMD processors [3]. Moreover, software support for DVFS is common among all major operating systems. As an example, Linux comes with `cpufreq` that allows the user to set the desired frequency at any time. As a consequence, several automatic DVFS controllers emerged such as the `ondemand` policy on Linux. DVFS is then a common technology that can be exploited to reduce the power consumption of processors.

DVFS control may seem easy at first sight but is, in fact, a complex operation. Indeed, reducing CPU frequency has a strong impact on performance that users may not tolerate. Moreover, even if reducing CPU frequency decreases power consumption, the resulting slowdown may in fact lead to an increased energy consumption since energy depends on both power and execution time. Such situation can often be observed in practice although several existing DVFS controllers ignore it. Thus, DVFS control is hard and requires precise strategies to achieve significant energy savings.

Several DVFS controllers were proposed in the past, demonstrating significant energy gains [4], [5], [6], [7], [8]. However processors recently evolved and several important changes were performed by manufacturers such as the introduction of multicore processors, also known as chip multi-processors (*CMP*), or the addition of embedded energy probes, providing new opportunities for enhancing DVFS control. We propose in this paper a new DVFS controller, called FoREST (FREquency Scaling Tool), specifically designed for recent processors. In order to determine the frequency to apply, FoREST directly evaluates the impact of a frequency transition on energy, decomposed in two distinct aspects: execution time and power consumption. Although execution time can be measured with an extremely high precision, power probes often suffer from low sampling rates. Thus, based on the assumption that ratios of power consumption for different frequencies are program-independent, FoREST estimates power gains from a short offline profiling and measures at runtime speedups or slowdowns achieved when transitioning frequency. On top of that, FoREST was designed with multicore processors in mind and, unlike many existing DVFS systems, it does not use any performance model that could be soon outdated. Thus, FoREST represents a major evolution of DVFS controllers, adapting advanced DVFS control to the real world.

FoREST has been implemented for recent Intel x86_64 processors on the Linux operating system. It is distributed as open-source software at http://code.google.com/p/forest-dvfs.

The main contributions are:

- We introduce FoREST, a new DVFS runtime controller independent from any performance model, suited to multicore processors, and whose maximal slowdown is chosen by the user. Unlike many existing DVFS systems, FoREST does not consider the slowdown provided by the user as a target slowdown but as a constraint to enforce.

- We present a novel approach to enhance DVFS control thanks to energy probes recently introduced in processors.
- The energy savings achieved by FoREST are compared to those of the default Linux controller, Granola [9], a commercial DVFS controller, and beta-adaptive [4].

The paper is organized as follows. Section II presents some existing DVFS tools. Section III describes the power estimation used in FoREST. Section IV presents the general method that can be decomposed in an offline phase, presented in Section V, a runtime evaluation step, described in Section VI, and a runtime execution step, as detailed in Section VII. FoREST is compared to other existing DVFS controllers in Section VIII before concluding in Section IX.

II. Related Works

Many DVFS controllers were proposed in the past. Some of them focus on reducing the energy consumption of a specific program during its execution [10], [11], while others consider the processor workload and do not require any program-specific knowledge [5], [6], [10], [11], [12]. The latter predict the impact of frequency transition, usually on performance, to decide on the frequency to apply. They exploit models correlating hardware counters to CPU intensity and then CPU intensity to the sensitivity of the workload to frequency transitions.

Closer to our work, Semeraro et al. proposed to periodically reduce the CPU frequency until an impact on execution time is suspected from hardware observation [13]. The proposed mechanism is not able to control its impact on slowdown as opposed to more recent solutions. Hsu et al. proposed *beta-adaptive* [4], a runtime DVFS controller that periodically evaluates the impact of frequencies on performance to deduce the best frequency to use under performance constraints. It shares several features with FoREST as it directly evaluates the impact of a frequency transition on Instruction Per Second rates (*IPS*) and reacts accordingly.

In general, the existing dynamic DVFS controllers suffer from several limitations. First, several existing controllers exploit a complex model to estimate the impact of a frequency transition on energy. Such models heavily depend on the targeted hardware and may be quickly outdated. For instance, a recent study shows that memory bandwidth is impacted by frequency transitions since the SandyBridge generation of Intel x86 CPUs [7]. Such subtle evolution, even within a micro-architecture, leads most of the existing models to fail. Moreover, all the presented systems are ignoring energy when selecting the frequency to apply. Indeed, most of them assume energy gains when reducing frequency while ensuring a relatively small slowdown. Such hypothesis is wrong on modern processors where energy consumption may increase when decreasing the frequency, depending on programs CPU usage [6]. Finally, multicore support is unclear for several systems and, in some cases, the method cannot fit current multicore processors where frequency has to be applied simultaneously to several cores. The following sections describe how FoREST addresses the presented issues by considering the impact of both power and execution time when choosing a frequency.

III. Power Ratios

Power consumption of current processors can be decomposed as the sum of static and dynamic power [14]. The first one is mainly architecture-dependent whereas the second one depends on the workload. Based on such decomposition and using a common hypothesis, we demonstrate that power ratios are approximately program independent, allowing FoREST to estimate power consumption of any workload from a simple offline profiling.

A. Power ratios computation

The dynamic power can be expressed as follows [15]:

$$P_{dynamic} \simeq A \times C \times V^2 \times f \tag{1}$$

where A is the percentage of active gates, C is the total capacitance load, V is the supply voltage, and f is the processor frequency. Note that the power depends on the machine characteristics (V, f and C) and the program (A).

As in other previous work [14], [16], [17], we assume that $P_{dynamic}$ is proportional to P_{static}, in other words, $P_{static} = k \times P_{dynamic}$ and power consumption can be expressed as $P = (k+1) \times P_{dynamic}$.

Let P_1 and P_2 be the power consumption of a given program at two different frequencies f_1 and f_2. It is possible to compute the power ratio between P_1 and P_2 as shown in formula 2:

$$\begin{aligned} P_1 &= (k_1+1) \times (A \times C_1 \times V_1^2 \times f_1) \\ P_2 &= (k_2+1) \times (A \times C_2 \times V_2^2 \times f_2) \\ \frac{P_1}{P_2} &= \frac{k_1+1}{k_2+1} \times \frac{C_1 \times V_1^2 \times f_1}{C_2 \times V_2^2 \times f_2} \end{aligned} \tag{2}$$

The activity (A) is assumed to not be affected by frequency transitions and then, for the same program, A remains unchanged for all frequencies. The assumption is discussed in paragraph III-B. Moreover, Piguet et al. showed that k is in fact program independent [16]. Thus, ratios of power consumption induced by a single program running at two frequencies are program independent. It is then possible to evaluate such power ratio for a given program and reuse the information for any program.

FoREST exploits such hypothesis to estimate the power gains achieved by any frequency compared to another one from power measurements performed offline. As the ratios are program-independent, the offline measurements can be transposed to any program.

B. Discussion

1) Frequency-independence of A: In the previous paragraph, A is assumed to be independent from the frequency. It is in fact an approximation as subtle variations can appear depending on the frequency. For instance, a memory-intensive program can saturate some resources such as store queues at high frequencies only, leading different activities to occur on the processor depending on the frequency. We assume such variations to be negligible and consider the average number of active gates to be stable for a given program, independently from the frequency.

2) Program-independence: In order to verify the program independence of power consumption ratios, the *NAS Parallel Benchmarks 3.0* suite and *SPEC CPU 2006* were run using every processor frequency while measuring power consumption, resulting in 688 different runs. Then, for any pair of frequencies, we computed the power ratio induced by each program. Finally, we computed the standard deviation of the power ratios involving the same frequencies but while different programs were running on the same number of cores. The standard deviation expresses the average error of the hypothesis for the evaluated programs.

Results show a maximal standard deviation of $0.02\,\%$ for power ratios whereas power itself has a standard deviation of more than $4.5\,\mathrm{W}$ for different programs using the same frequency. Thus, even if power consumption obviously depends on the program itself, considering ratios of power consumption for different frequencies as being program-independent is a realistic hypothesis.

IV. Overview

FoREST aims at applying the frequencies minimizing the energy consumption with respect to a slowdown constraint S. To do so, it performs three main operations: an offline analysis, an evaluation phase, and an execution phase. Its algorithm is made of the following steps:

1) Compute the power ratios, denoted pr_i, of every frequency f_i over the power consumption of the maximum frequency f_{max} during an offline analysis.
2) Evaluate the speedup s_i of each frequency f_i compared to f_{max}.
3) Repeat steps 4 to 8 for all $0 \leq S^* \leq S$.
4) Build on each core the frequency couples (f_1, f_2) such that $s_1 < S^* < s_2$.
5) Keep only the couples built on all the processor cores.
6) Compute on each core the times (t_1, t_2) for each frequency in (f_1, f_2) such that: $t_1 \times s_1 + t_2 \times s_2 = S^*$.
7) Select a couple $((f_1, t_1), (f_2, t_2))$ for the whole processor.
8) Compute the energy gain induced by the chosen couple using s_i, pr_i, and t_i.
9) Pick the frequency couple and its associated slowdown S^* providing the greatest energy saving.
10) Apply the frequency couple.

Step 1 is performed offline while steps 2-9 are part of the evaluation phase, and step 10 is the main task of the execution phase. The details for each step are presented in the next sections.

V. Offline Power Measurement

The offline analysis aims at determining the impact of frequencies on power consumption. As explained in Section III, power ratios are considered as being program-independent. Thus, FoREST runs a small benchmark program made of a sequence of additions while measuring power consumption using the probes embedded within the processor. Then, considering the maximal frequency as a reference, FoREST divides the power consumption measured at any frequency by that of the reference frequency. The same operation is repeated for any number of active cores, in order to take into account variations of P_{static} for different number of active cores. The offline profiling lasts for a couple of minutes and only needs to be performed once, typically during program installation. Such offline power measurement, combined with runtime performance measurements, allows FoREST to select a frequency considering not only the induced execution time but also the energy savings.

Frequency	f_1	f_2	f_3	f_4
IPS ($\times 10^9$)	1.7	2.0	2.5	3
Speedup of f_4 (t_i/t_4)	1.8	1.5	1.2	1
Power gain vs. f_4 (P_i/P_4)	0.4	0.6	0.7	1
Energy gain vs. f_4 (e_i/e_4)	0.72	0.9	0.84	1

Fig. 1. Sample measurement results from offline profiling and online evaluation for one processor core. t_i, P_i, and e_i represent respectively the execution time, power consumption, and energy consumption at frequency f_i

VI. Frequency Evaluation

In order to determine which frequency to use, FoREST combines power and execution time gains expected from every frequency. Power gains are computed for any possible workload during the offline profiling but the impact of a frequency transition on execution time remains to be determined. As opposed to power, execution time gains heavily depend on the workload, and more specifically on its CPU boundness [18]. FoREST must then evaluate the speedups induced by frequency transitions at runtime.

A. Runtime IPS profiling

To measure the speedup achieved for the current workload depending on the frequency, FoREST applies each frequency during short periods of time while measuring the number of instructions executed per second (IPS). Although it is not perfectly representative of execution time, IPS can be considered as a precise-enough metric for evaluating speedups: doubling IPS for instance generally translates into a $2\times$ speedup. Thus, FoREST measures the IPS of the running workload during periods of $100\,\mu\mathrm{s}$ for several frequencies. The measurement is performed synchronously on all the cores sharing the same frequency setting, allowing FoREST to work on recent multicore processors. Then, every measured IPS is divided with the one achieved by the maximal frequency in order to deduce speedups of every frequency compared to the highest frequency. As detailed later, the evaluated frequencies include those close to the one previously chosen plus the highest frequency. Sample IPS measurements for one core are presented in Figure 1, associated to the corresponding speedup over the highest frequency. Thanks to the runtime

IPS evaluation, FoREST is then aware of the speedup induced by any frequency.

FoREST assumes the IPS to remain constant during the whole evaluation, which may not be correct, leading to incorrect frequency selection. As for many dynamic systems, such mispredictions are tolerated, considering that a new evaluation will be performed soon after the incorrect one.

Once IPS evaluation is performed, power gains and speedups of various frequencies compared to the highest one are known. It is then possible to predict the potential energy gains induced by the different frequencies.

B. Energy Gains

Power gains and speedups can be multiplied to obtain energy gains compared to an arbitrarily chosen frequency, the maximal one in our case as illustrated in Figure 1. Using such energy gains, it is then immediately possible to estimate the energy savings that can be expected from the evaluated frequencies.

C. Best Frequency Pair

Applying a unique frequency may not provide enough flexibility. Indeed, in some cases, all the frequencies lead to a slowdown greater than what the user tolerates. Thus, even if it is profitable to reduce frequency, it may be possible that none suits the slowdown constraint. Therefore, in many cases, there is no unique frequency providing maximal energy gains under the slowdown constraint.

When two frequencies are applied, the achieved IPS is proportional to the one achieved by every frequency. It is then possible to emulate any frequency by combining two frequencies during variable amounts of time [4], [5]. For example, it is possible to achieve 0.46 IPS using two frequencies able to perform respectively 0.4 IPS and 0.5 IPS. FoREST exploits this property and actually selects a couple of frequencies for every core to achieve any desired IPS within the limits imposed by the user.

For a specified slowdown, FoREST determines the best frequency couple to use in three successive steps, as detailed below.

First, the cores may run various workloads that react differently to frequency transition. Every core has then a different *target* IPS, computed as the maximal IPS observed on the core among the evaluated frequencies, minus the desired slowdown. Notice that the maximal frequency is always evaluated. The target IPS represents the objective IPS for a given processor core. To achieve such IPS, all the couples made of a frequency resulting in lower IPS and another one leading to higher IPS can be used. However, some frequency pairs may surround the target IPS on some cores but not on others and must be ignored. The goal of the first step is then to eliminate frequency couples that cannot be used on all the cores sharing the same frequency setting.

Second, FoREST computes the execution times associated to each frequency in couples obtained at the first step. The execution times depends on the target IPS to achieve on each core. Thus, the cores associate different durations to the frequencies in pairs but, as the frequency is shared among different cores, FoREST has to chose only one duration for each frequency in every couple used on all the cores. We assume that the IPS cannot decrease when the frequency increases. Then, for every couple, FoREST selects among the cores the pair of durations that maximizes the execution time of the highest frequency, enforcing the slowdown constraint. At the end of the second step, all the possible frequency pairs are then associated to one unique pair of execution times enforcing the slowdown requirements on all the cores.

Finally, FoREST has to choose one frequency pair. To do so, it computes the energy gain achieved by every couple and only selects the one providing maximal energy savings. However, in recent multicore processors, a frequency is necessarily set simultaneously on all the cores. FoREST considers this limitation and computes the energy gains achieved by the couples on each individual processor core. Then, the overall processor energy gain for a given couple is computed as the average gain over all the cores sharing the same frequency setting. By doing so, FoREST assumes that all the cores equally participate to the total energy consumption. Then, once energy gains are known for all the considered frequency couples, FoREST picks the one achieving maximal energy savings. All the couples were forged to achieve a particular target slowdown, the chosen couple is then guaranteed to respect the user-defined slowdown constraint.

Once the best couple is found, each frequency can be set for the associated duration, computed in the second step, in order to achieve the desired slowdown.

D. Ideal Slowdown

The last remaining element FoREST must compute is the best slowdown. Indeed, if the user specifies a tolerated slowdown, it is not necessarily the slowdown providing maximal energy gains. FoREST has the ability to generate a frequency pair ensuring the chosen slowdown, and can estimate the associated energy savings. Thus, it evaluates several slowdowns from 0 % to the maximum tolerated, by steps of 1 %. As a result, FoREST computes for several slowdowns a frequency pair and an associated energy gain to finally pick the frequency couple maximizing energy savings.

The ability to detect the ideal slowdown is a major improvement over existing work. Indeed, existing runtime controllers usually do not consider power when predicting the frequency to use. In such conditions, it is impossible to determine if a slowdown is profitable for energy.

E. Scope Matters

Although DVFS has a scope limited to the CPU, total system consumption cannot be ignored. In fact, saving energy at the processor level is not of any help if the system energy increases. Such case can happen if the energy saved on the processor is much lower than what the rest of the system consumes because of the slowdown induced by a reduced frequency. Thus, in order to perform efficiently, FoREST has to consider system power consumption.

Currently, it is often impossible to obtain reliable system power consumption. Thus, FoREST approximates it to an arbitrary constant value of $50\,\mathrm{W}$ for desktop computers. It does not lead to major changes in the algorithm: the approximate system power consumption is simply added to the measured CPU power when computing power gains. Such approximation can be replaced by the actual power consumption when it is available, increasing the efficiency of the decisions taken by FoREST. As a result, FoREST considers energy gains at the system level rather than at the processor level, trying to optimize the system energy. It is an additional improvement of FoREST over existing controllers that often focus on processor energy consumption.

F. Performance and Energy Modes

Depending on the context and the computer, users have different requirements regarding energy and performance. For instance, users playing games or encoding videos may desire the highest achievable performance, while users working on battery-powered laptops are ready to huge performance sacrifices to save energy. Thus, different user profiles exist, requiring different approaches depending on the user needs.

The main issue with existing implementations in operating systems is the usual confusion between power and energy, as energy-saving modes often consist in systematically applying the lowest frequency. Such approach fails with many programs for which frequency reduction induces such high slowdowns that the power gain is insufficient to achieve energy savings. On the other hand, FoREST automatically detects the optimal slowdown to induce under user-defined constraints. Thus, users requiring high performance can specify a small tolerated slowdown such as $5\,\%$, while those concerned by energy can set a higher tolerated slowdown such as $50\,\%$ or $100\,\%$. Thanks to such parameterizable maximal slowdown, FoREST can then adapt to several scenarios and ensure energy savings at the system scale according to the user needs.

G. Discussion

1) Multicore Processors: FoREST considers shared frequency domains of multicore processors at every stage. First, it is only replicated once per group of cores sharing the same frequency transition. Then, during the frequency evaluation, IPS is measured synchronously on all the cores of a group. Finally, the selected frequency couple is specifically forged to enforce the desired slowdown on every core. FoREST is then specific in its ability to work on the recent multicore processors. Moreover, no major operation in FoREST depends on the number of cores on the processor, thus, FoREST is expected to scale well with the number of cores.

2) Frequency Transition Overhead: As any other DVFS controller, FoREST has to take care of the frequency transition latency. Indeed, it can take a significant amount of time to switch the frequency on exiting processors which may lead to incorrect measurements when the frequency transition latency is close to the duration of the IPS evaluation. Thus, in order to increase the measurement precision, FoREST starts measuring the IPS only after having waited for the frequency to change. FoREST is then aware of the frequency transition overhead and takes it into account when running.

VII. Sequence Execution

When the evaluation step finishes, a frequency couple is built and each frequency in the couple is associated to a duration. The actual frequency transitions are performed by FoREST in no specific order. Although executing the frequency sequence induces no measurable overhead, evaluating a frequency has a cost. In fact, most of the overhead of FoREST comes from the time spent in evaluating inefficient frequencies. In order to limit this inconvenience, FoREST employs two main techniques.

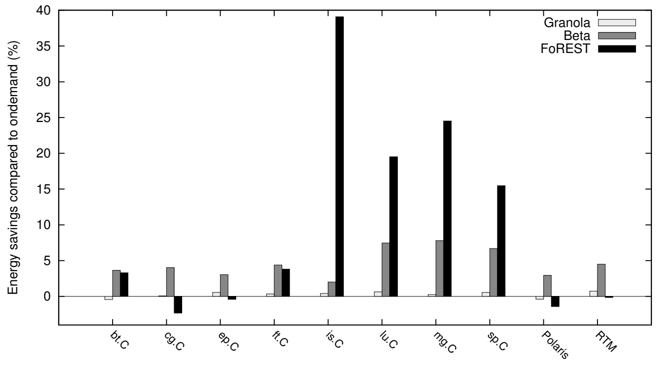
First, FoREST modulates the duration of the whole frequency couple depending on workload stability. FoREST defines the main frequency $mainFreq$ of a couple c as the one executed for the longest duration. If $mainFreq$ remains the same over two consecutive sequence executions, the system workload is assumed as stable and FoREST doubles the duration of c until it reaches a maximum of $100\,\mathrm{ms}$. As soon as $mainFreq$ changes, the duration of c is reset to $1\,\mathrm{ms}$. Thus, when the workload is unstable, FoREST triggers evaluations more frequently, quickly reacting to workload instabilities. However, when the workload is stable, the evaluation rate is reduced in order to limit the induced overhead. Couple durations are chosen to be greater than the average frequency transition latency on current processors as well as small enough to reach negligible overhead on our experimental platform. Determining more precise events to achieve better workload phase detection is possible thanks to hardware counters for instance, but is left for future work.

Second, FoREST limits the duration of evaluation steps by restricting the set of evaluated frequencies to those likely to be chosen in the near future. To do so, FoREST only evaluates the frequencies in the neighborhood of $mainFreq$. In our implementation, FoREST evaluates the frequency immediately higher and immediately lower than $mainFreq$. Additionally, the highest frequency is always evaluated as it serves as a reference for computing slowdowns and energy gains. Evaluating only a subset of the frequencies can greatly reduce the overhead of the evaluation phase as it prevents inefficient frequencies to be set, even for a short profiling.

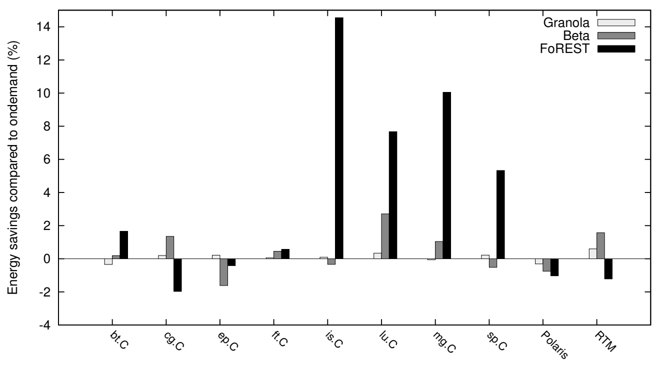
As any dynamic system, runtime DVFS could provoke massive overheads on the system if it is applied without considering performance. In our experiments, we measured an execution time overhead of $0.51\,\%$ and a $0.98\,\%$ energy overhead when FoREST is working. Such low overhead is only reached because of the two described adaptive techniques implemented in FoREST.

VIII. Experiments

FoREST is implemented for x86_64 processors on Linux, allowing us to evaluate its main features on a real environment.



(a) CPU



(b) Whole system

Fig. 2. Energy savings over what `ondemand` achieves. 5 % slowdown allowed for FoREST and beta-adaptive.

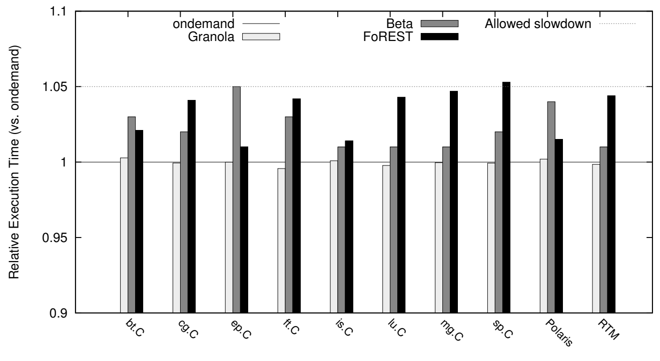


Fig. 4. Execution time normalized to that achieved by `ondemand`. 5 % slowdown allowed for FoREST and beta-adaptive.

The energy savings achieved by FoREST and the associated slowdowns are measured on our experimental platform in order to check its ability to reduce energy consumption and guarantee the requested maximal slowdown.

The experiments were run on an Intel Core i5 2380P quad-core processor, running Linux 3.5.3. The sixteen processor frequencies range between 1.6 GHz and 3.1 GHz, plus a turbo mode. The benchmark programs consist in the NAS OpenMP parallel programs 3.0 running the C class datasets. Additionally, we considered two industrial programs: `RTM`, the main kernels extracted out of a proprietary program from TOTAL[1] used to perform reverse time migration, and `Polaris`, a molecular dynamics program from CEA[2]. Measurements were performed using energy probes embedded in the processor and using a Yokogawa WT210 power meter plugged to the computer's electrical socket in order to measure both processor and overall system energy consumption. Results are the median value of 5 executions, normalized relatively to `ondemand`, the default Linux DVFS control policy.

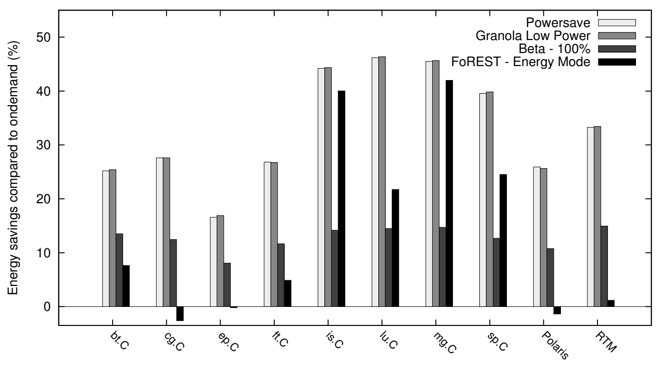
A. Energy gains

The benchmark programs were first run on the experimental platform using different DVFS controllers. The first one is Granola, a commercial DVFS controller designed by MiserWare Inc.; the second one is beta-adaptive [4]; and the last one is FoREST. Granola uses its default configuration while beta-adaptive and FoREST are allowed to provoke at most 5 % slowdown. Notice beta-adaptive suffers from a major limitation here: as many other DVFS controllers, it is not designed to support multicore processors where all the cores have to use the same frequency, which is the case for all recent Intel x86 multicore processors. The energy savings achieved both at processor and system levels for every DVFS controller are presented in Figure 2. In the figure, values higher than 0 are energy savings compared to an execution when using `ondemand`. Conversely, values below 0 represent additional energy consumption.

Among all the evaluated programs, some contain large memory intense phases. In such case, it is possible to decrease CPU frequency without impacting the execution time. Such programs are then perfect targets for DVFS controllers, that can achieve major energy savings. Conversely, some programs are CPU intense and reducing CPU frequency often increases their energy consumption. Therefore, no significant energy savings can be expected for such programs.

Granola was able to achieve light energy savings in many cases but did not significantly outperforms `ondemand`. In fact, from the words of Granola's authors, Granola is not designed to outperform `ondemand` but rather to save as much energy as possible without harming performance. On the other hand, beta adaptive is able to save more energy in general, at the cost of an increased execution time. However, FoREST clearly outperforms all the DVFS controllers with memory-bound programs while maintaining a decent consumption with other programs. Indeed, more than 15 % energy is saved for `is`, `lu`, `mg`, and `sp` at the processor level. It illustrates the

[1]TOTAL is a France-based oil and gas company
[2]CEA is a French government-funded technological research organization


(a) CPU

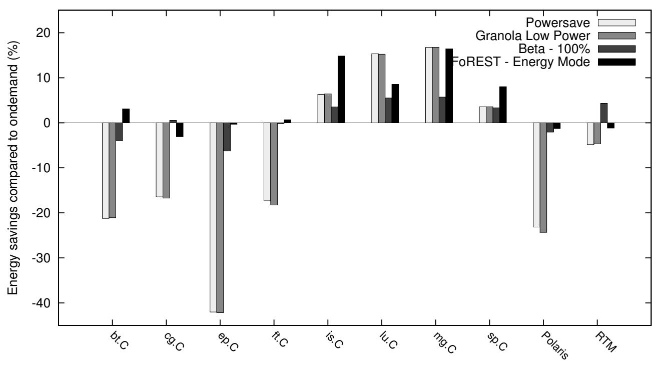

(b) Whole system

Fig. 3. Energy savings over what `ondemand` achieves. 100 % slowdown allowed for FoREST and beta-adaptive.

ability of FoREST to detect even short memory phase in programs and to exploit them to save energy.

For CPU-intense programs, FoREST sometimes achieves slight energy over consumption. One can notice a similar behavior for the beta-adaptive method. In fact, it is mostly due to the adaptive method chosen by FoREST and beta-adaptive. Both systems evaluate the impact of a frequency transition on performance and, in the case of FoREST, on energy consumption. It implies periodic evaluation of frequencies, including inefficient ones. Such evaluation on CPU intense program immediately leads to an increased energy consumption, whose importance depends on how frequently and for how long the evaluations are performed. Additionally, dynamic systems may pick incorrect frequencies for short durations before correcting their mistake at the next evaluation, increasing in some rare cases their overhead.

B. Performance degradation

FoREST directly measures the impact of frequency transitions on IPS to guarantee a maximal slowdown afterwards. In order to determine if it actually enforces the requested maximal slowdown, we measured the execution time of all the benchmark programs when using `ondemand`, Granola, beta-adaptive, and FoREST. The execution times are normalized over that of `ondemand` in Figure 4.

FoREST is able to enforce the maximal requested slowdown as it never provokes slowdowns significantly above the 5 % limit. Granola leads to execution times similar to what `ondemand` achieves. Compared to `ondemand`, beta-adaptive and FoREST increase programs execution time but the resulting slowdown is within the range tolerated by the user. When considering both slowdowns and energy savings, the presented results indicate that FoREST takes relevant decisions as it can trade slowdown for energy all the while not exceeding the requested slowdown threshold.

FoREST has the ability to automatically determine what slowdown must be applied at anytime in order to save as much energy as possible. As opposed to many other mechanisms, it does not systematically choose the maximal tolerated slowdown. This ability is reflected in Figure 4 as the measured slowdown is often lower than what the user tolerates. Interestingly, beta-adaptive is designed to always provoke the maximal slowdown configured by the user but its poor sensitivity to program phases and its inability to efficiently support existing multi-core processors allow it to provoke slowdowns below the maximum, avoiding large energy over consumption in some cases.

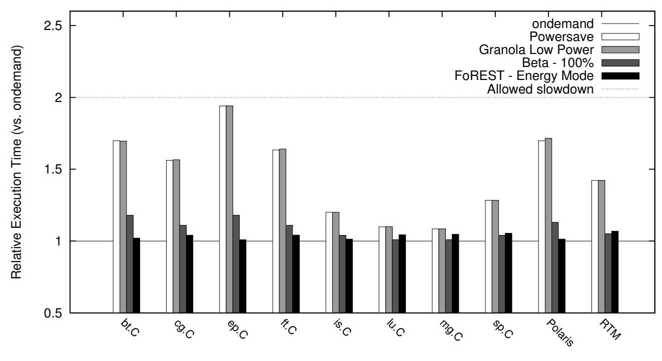


Fig. 5. Execution time normalized to that achieved by `ondemand`. 100 % slowdown allowed for FoREST and beta-adaptive.

C. Energy saving mode

For some users, energy consumption is a major concern and execution time does not matter. For instance, when working on battery-powered devices, autonomy becomes a critical criterion for the user. For that purpose, we configured FoREST to ensure a maximal slowdown of 100 % and run again the experiments. The resulting energy savings and execution times are presented respectively in Figure 3 and Figure 5. During the experiments, we compared FoREST to `ondemand`, the `powersave` Linux DVFS policy that systematically sets the lowest frequency, Granola using its "low power" mode, and beta-adaptive, also targeting a 100 % slowdown.

When considering such extreme allowed slowdowns, both processor-level and system-level energy consumption matters. It is illustrated by the large savings achieved by `powersave` and Granola at the processor scale and the corresponding major energy losses at the system scale for many programs. Thus, FoREST does not necessarily induce the best energy savings at the processor level but it often performs best on the overall system. In fact, FoREST benefits from its ability to target system-wide energy savings rather than focusing on the processor scale. It is then able to perform significant system energy savings on memory intense programs while avoiding major energy losses on CPU intense programs. The few cases where FoREST is not able to reach the maximal savings are generally due to short program phases for which FoREST may react too late. We plan to enhance further FoREST for such programs by considering hardware counter hints about phase transitions but this is left for future work.

IX. Conclusion

FoREST is a new runtime DVFS controller suited to recent technologies. It determines potential energy gains from two phases: an offline phase exploiting energy probes embedded in processors, and runtime speedup measurements for the most interesting frequencies. FoREST is then able to reduce energy consumption for the whole system under a maximum slowdown constraint configured by the user. Moreover, FoREST inherently supports multicore processors, extending the previously proposed DVFS controllers to allow efficient and controlled energy savings on recent processor technologies.

A major extension of this work is related to distributed systems. Indeed, a slowdown on one node may degrade the energy consumption on all the other nodes when they are synchronized. With that issue in mind, we plan to extend FoREST to support distributed computers. More generally, we show in this paper that major energy savings can be achieved for the processor and system, extending this work to a larger scale could ensure even greater energy savings for a whole cluster.

References

[1] S. Song, R. Ge, X. Feng, and K. W. Cameron, "Energy profiling and analysis of the hpc challenge benchmarks," *International Journal of High Performance Computing Applications*, 2009. [Online]. Available: http://hpc.sagepub.com/content/early/2009/06/05/1094342009106193.abstract

[2] Intel Corporation, "Intel 64 and IA-32 Architectures Software Developers Manual." [Online]. Available: http://download.intel.com/design/processor/manuals/253668.pdf

[3] AMD, "AMD Cool'n'Quiet." [Online]. Available: http://www.amd.com/us/products/techno-logies/cool-n-quiet/Pages/cool-n-quiet.aspx

[4] C.-h. Hsu and W.-c. Feng, "A power-aware run-time system for high-performance computing," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 1–. [Online]. Available: http://dx.doi.org/10.1109/SC.2005.3

[5] R. Ge, X. Feng, W. chun Feng, and K. Cameron, "CPU MISER: A performance-directed, run-time system for power-aware clusters," in *Parallel Processing, 2007. ICPP 2007. International Conference on*, sept. 2007, p. 18.

[6] K. Livingston, N. Triquenaux, T. Fighiera, J. Beyler, and W. Jalby, "Computer using too much power? give it a REST (Runtime Energy Saving Technology)," *Computer Science - Research and Development*, pp. 1–8, 2012. [Online]. Available: http://dx.doi.org/10.1007/s00450-012-0226-0

[7] R. Schöne, D. Hackenberg, and D. Molka, "Memory performance at reduced cpu clock speeds: an analysis of current x86_64 processors," in *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, ser. HotPower'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387869.2387878

[8] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks, "A dynamic compilation framework for controlling microprocessor energy and performance," in *MICRO*, 2005, pp. 271–282.

[9] MiserWare Inc., "Granola energy saving tool." [Online]. Available: http://grano.la

[10] V. W. Freeh and D. K. Lowenthal, "Using multiple energy gears in mpi programs on a power-scalable cluster," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, ser. PPoPP '05. New York, NY, USA: ACM, 2005, pp. 164–173. [Online]. Available: http://doi.acm.org/10.1145/1065944.1065967

[11] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, ser. PLDI '03. New York, NY, USA: ACM, 2003, pp. 38–48. [Online]. Available: http://doi.acm.org/10.1145/781131.781137

[12] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *Proceedings of the 2004 international symposium on Low power electronics and design*, ser. ISLPED '04. New York, NY, USA: ACM, 2004, pp. 174–179. [Online]. Available: http://doi.acm.org/10.1145/1013235.1013282

[13] G. Semeraro, D. H. Albonesi, S. G. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture," in *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, ser. MICRO 35. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 356–367. [Online]. Available: http://dl.acm.org/citation.cfm?id=774861.774899

[14] L. Wang, G. von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 368–377. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2010.19

[15] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 34–. [Online]. Available: http://dx.doi.org/10.1109/SC.2005.57

[16] C. Piguet, C. Schuster, and J.-L. Nagel, "Optimizing architecture activity and logic depth for static and dynamic power reduction," in *Circuits and Systems, 2004. NEWCAS 2004. The 2nd Annual IEEE Northeast Workshop on*, june 2004, pp. 41 – 44.

[17] J. Li and J. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," in *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, feb. 2006, pp. 77 – 87.

[18] J. Noudohouenou, V. Palomares, W. Jalby, D. C. Wong, D. J. Kuck, and J. C. Beyler, "Simsys: a performance simulation framework," in *Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, ser. RAPIDO '13. New York, NY, USA: ACM, 2013, pp. 1:1–1:8. [Online]. Available: http://doi.acm.org/10.1145/2432516.2432517