# Software Controlled Clock Modulation for Energy Efficiency Optimization on Intel Processors

Robert Schöne, Thomas Ilsche, Mario Bielert, Daniel Molka, and Daniel Hackenberg
Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden, 01062 Dresden, Germany,
Email:{robert.schoene, thomas.ilsche, mario.bielert, daniel.molka, daniel.hackenberg}@tu-dresden.de

***Abstract*—Current Intel processors implement a variety of power saving features like frequency scaling and idle states. These mechanisms limit the power draw and thereby decrease the thermal dissipation of the processors. However, they also have an impact on the achievable performance. The various mechanisms significantly differ regarding the amount of power savings, the latency of mode changes, and the associated overhead. In this paper, we describe and closely examine the so-called software controlled clock modulation mechanism for different processor generations. We present results that imply that the available documentation is not always correct and describe when this feature can be used to improve energy efficiency. We additionally compare it against the more popular feature of dynamic voltage and frequency scaling and develop a model to decide which feature should be used to optimize inter-process synchronizations on Intel Haswell-EP processors.**

***Index Terms*—Microprocessors, Performance analysis, Systems modeling, Dynamic voltage scaling**

## I. Introduction

The limits of technology scaling due to the increasing power density [1] have resulted in various power optimization techniques that are now available in state-of-the-art processors. The ACPI standard [2] defines four different power saving states that are supported by current Intel processors: System sleeping states (S-states), processor power states (C-states), processor performance states (P-states), and throttling states (T-states). These states are implemented by using one of the following hardware power saving techniques: power gating (S- and deep C-states), clock gating (shallow C-states), dynamic voltage and frequency scaling (P-states, C1E-state), and clock modulation (T-states). S- and C-states stop the processing of instructions. They need an external signal, e.g., an interrupt, to return to a working state. Furthermore, the processor state has to be restored when resuming to normal operation, which creates considerable overhead [3]. P- and T-states are not subject of these restrictions. Thus, they can be used easily for energy efficiency optimization. While dynamic voltage and frequency scaling (DVFS) optimization is so common that it is part of operating systems [4], optimization efforts that use clock modulation are still rare. This paper describes details of Intel's clock modulation implementation for several processor generations. This will help researchers to assess whether they should use this feature for their optimization.

This paper is structured as follows: In Section II, we describe Intels clock modulation feature and discuss its usage for energy efficiency optimization in High Performance Computing (HPC). We define our measurement environment, hardware, and software in Section III. In Section IV, we describe how throttling states in various Intel processors perform in detail. The sustained behavior of power and performance for Haswell-EP processors is presented in Section V. In Section VI, we describe a model that determines whether P- or T-states should be used and exemplarily apply it on Intel Haswell-EP processors. We conclude this paper with a summary and outlook in Section VII.

## II. Background and Related Work

Weste and Harris describe the processor power consumption as follows [5, Section 5.1.3]:

$$P_{total} = \overbrace{\alpha C V_{DD}^2 f + I_{SC} V_{DD}}^{P_{dynamic}} + \overbrace{(I_{leak} + I_{cont}) \cdot V_{DD}}^{P_{static}} \quad (1)$$

The different power saving features target different parts of Equation 1. In this paper, we focus on the effects of software controlled clock modulation, which only influences the activity factor $\alpha$. Thus, it does not affect the static power consumption, which is dominated by leakage power. In contrast, DVFS reduces the frequency $f$ as well as the supply voltage $V_{DD}$, which reduces both, static power and dynamic power.

Clock modulation is related to clock gating: In clock gating (depicted in Figure 1), the clock is disabled whenever the stop-clock signal is active. The applied clock signal is then the result of the external clock signal AND the de-asserted stop-clock. Thus, the dynamic power consumption can be reduced significantly, as "short circuit current has become almost negligible"[5, Section 5.2.5] in nanometer processes.

Clock modulation uses a comparable mechanism. When a certain condition (clock modulation assertion) is set, the clock is disabled whenever a clock modulation signal indicates it.

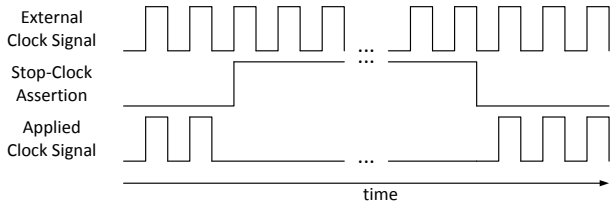



Fig. 1: Influence of clock gating on a processor clock signal

 **This paper has been published in the Proceedings of the 4th International Workshop on Energy Efficient Supercomputing (E2SC), 2016, DOI: 10.1109/E2SC.2016.015**

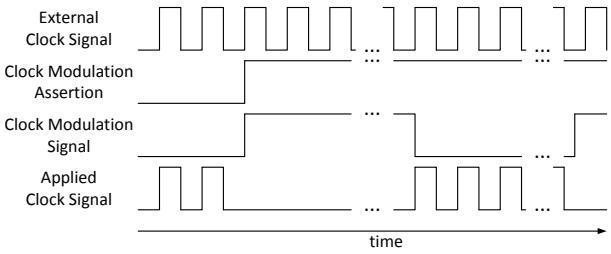



Fig. 2: Influence of clock modulation on a processor clock signal

The resulting signal that is applied to the processor is thus the result of ANDing the external clock signal, and the NEGated result of ANDing the condition, and the clock modulation signal. This is depicted in Figure 2.

The initial intent of support for clock modulation in Intel processors is to prevent them from overheating [6, Chapter 14.7.2]. The first implementation, which is called *Thermal Monitor 1* (TM1), has been introduced with Intel Pentium 4 processors. Another feature that enforces a low processor temperature is called *Thermal Monitor 2* (TM2), which uses DVFS to achieve a lower power dissipation. In [7], Rotem et al. describe how this technique is more effective than its clock modulation counterpart. Starting with Intel Core 2 processors, Intel introduced *Adaptive Thermal Monitor* where different transition targets can be used under overheating conditions. Processor manuals [6] describe this mechanism to dynamically select between TM2 and TM1. According to technical documents for desktop processors [8], [9], this mechanism applies DVFS when the target temperature is exceeded and clock modulation in addition, if the thermal effect of DVFS is not sufficient. Clock modulation can also be triggered by software via the MSR register `IA32_CLOCK_MODULATION`. This interface can be used to modify the percentage of skipped cycles in steps of 6.25 % (12.5 % for older architectures).

Clock modulation does not need any support by voltage regulators and can therefore be implemented with a reasonable amount of extra hardware. Thus, current Intel processors implement it per processor core in contrast to DVFS which typically has a coarser granularity.

Bhalachandra et al. use clock modulation to optimize the energy efficiency of imbalanced MPI programs [10]. They implement a model that changes the clock modulation setting at every collective MPI routine according to the time spent in communication phases. Cicotti et al. present the EfficientSpeed library [11]. This library determines the best DVFS/clock modulation setting for instrumented regions to optimize energy efficiency. Finally, Wang et al. also apply clock modulation to Sandy Bridge processors for energy efficiency purposes [12]. In this paper, we describe a peculiarity of Sandy Bridge and Ivy Bridge processors that indicates that the results in [10], [11], [12] will be significantly different on other architectures.

## III. Experimental Methodology

We investigate the short term effects of clock modulation as well as the behavior over longer time periods. In this Section, we describe the used workflows and the set of processors that we investigated.

### A. Analysis of short time scale effects

To determine how clock modulation is implemented by the processor, we use a measurement routine that is similar to the one used by Mazouz et al. for measuring P-state latencies [13]. Our implementation is shown in Listing 1. We run $2^{20}$ iterations of this measurement loop and record the respective run times in memory. In order to represent software that is already established in the operating system contexts, we skip the first 75 % of the results in the analysis, as they exhibit more noise than the latter 25 %. This results in a total number of $2^{18}$ samples. To control the clock modulation setting, we use the x86_adapt[1] library and kernel module [3].

The entire benchmark is repeated with all combinations of available frequencies and clock modulation settings. Additionally, the clock modulation is set either on one or on all cores of a system to check whether the implementation treats this differently. The gathered execution times are analyzed in a post-mortem step to answer the following questions:

- How long is the cycle of the clock modulation signal?
- How long is the assertion phase of the clock modulation signal depending on the clock modulation value?
- Is this mechanism influenced by the frequency of the processor?
- Does it depend on the number of cores using it?
- Is the signal synchronous on all cores of a processor?

To gather the initialization delay, we get the expected runtime of the measurement loop, activate clock modulation, and execute the loop until its runtime is significantly higher. Afterwards we ensure that the extended runtime is within the expected range. To measure the delay after deactivating clock modulation, we wait a random time after the last clock modulation cycle, deactivate clock modulation, and wait for 60 µs for an extended runtime. We register the random wait-time, the extended runtime and the time between deactivating clock modulation and the start time of the interrupted loop.

```
unsigned hi, lo; unsigned long long count;
for ( count = 0 ; count < STORE_SIZE ; count++ ) {
  asm volatile( // 150 adds
    "addl $1,%%eax;" //... repeat more adds
    :::"%eax");
  // get time in reference cycles
  asm volatile("mfence;rdtsc":"=a"(lo),"=d"(hi));
  // store results
  results[count] = ((unsigned long long) lo) |
    (((unsigned long long) hi) << 32);
}
```

Listing 1: measurement loop for short timescale analysis

[1] https://github.com/tud-zih-energy/x86_adapt

 **This paper has been published in the Proceedings of the 4th International Workshop on Energy Efficient Supercomputing (E2SC), 2016, DOI: 10.1109/E2SC.2016.015**

### B. Influence on Performance and Power Consumption

We use a second benchmark code to measure the influence of clock modulation on application performance and system power consumption. The benchmark consists of multiple kernels that exhibit diverse characteristics in terms of resource usage (e.g. memory bound vs. compute intense). Each measurement kernel is repeated for approximately 60 seconds, the number of repetitions is reported as *utility*. This metric represents an abstract way to compare the performance at different settings. The kernels are executed at varying clock modulation and frequency settings. In Section V, we present the results with one thread per core unless stated otherwise.

The execution is correlated with an external power measurement. Average power consumption is computed from the inner 60 % percentile to avoid influence of timer accuracy and synchronization issues. We pin the single threads of the measurement loop to processor cores via the environment variable `GOMP_CPU_AFFINITY`. We also disable all C-States, but `C0` and `C1` via writing to the corresponding pseudo-files in the `/sys` directory. C-State auto-demotion is also disabled.

### C. Test Systems and Setup

We perform our experiments on a selection of Intel desktop and server processors, listed in Table I. With this assortment of processors, we can illustrate the development of the clock modulation mechanism over multiple processor generations and describe how the implementations differ from one another. All systems use Ubuntu 16.04 with the Linux kernel in version 4.4.0-21.

TABLE I: Tested Processors

| | Architecture | Processor Model | Frequency Range [GHz] |
|---|---|---|---|
| Desktop | Sandy Bridge | i7-2600 | 1.6-3.4 |
| | Ivy Bridge | i5-3470 | 1.6-3.2 |
| | Haswell | i7-4770 | 0.8-3.4 |
| | Skylake | i7-6700K | 0.5-4.0 |
| Server | Sandy Bridge | 2x E5-2670 | 1.2-2.6 |
| | Haswell | 2x E5-2690 v3 | 1.2-2.6 |

## IV. Clock Modulation at Short Time Scale

In this Section, we describe low-level details of how different processors implement clock modulation. The findings enable us to understand the implemented mechanisms and to interpret results over longer time periods.

### A. Clock Modulation Parameters

Based on the measurements described in Section III-A, we determine the following parameters: $f$ defines the frequency that is applied to the processor core, $m$ defines the applied clock modulation setting. $t_{std}(f,m)$ represents the time spent for executing one benchmark loop. $t_{thr}(f,m)$ represents the execution time when the loop is interrupted by a throttling event. $\Delta t_{thr}(f,m)$ describes the difference between $t_{thr}(f,m)$ and $t_{std}(f)$ and represents the length of the assertion of the clock modulation signal. $T_{thr}(f,m)$ represents the time between two throttling events and the length of a clock modulation cycle. Two throttling cycles and the parameters and the measured times are depicted in Figure 3.

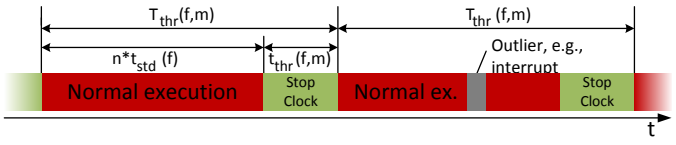



Fig. 3: Clock modulation, measured parameters.

From the captured runtimes $t^i(f,m)$, we first determine $t_{std}(f,m)$. To do so, we sort the runtimes in ascending order and create a cluster, starting with the minimum runtime $t^0(f,m)$. We add the next runtime $t^{i+1}(f,m)$ to this cluster as long as $t^{i+1}(f,m) - t^i(f,m) < \max(d_{abs}, t^i(f,m)\cdot(1+d_{rel}))$, where $d_{abs} = 30$ ref. cycles and $d_{rel} = 8\,\%$. We denote minimum and maximum of the cluster as $t^{min}_{std}(f,m)$, resp. $t^{max}_{std}(f,m)$. Afterwards, we attempt to find another cluster for $t_{thr}(f,m)$. We use the same algorithm as before and set the minimal time $t^{min}_{thr}(f,m)$ to the successor of $t^{max}_{std}(f,m)$. If the resulting cluster does not represent at least 5 % of the overall runtime, we increase $t^{min}_{thr}(f,m)$ and repeat the search until the 5 %-criterion is met. We classify measured runtimes that are not part of these clusters as outliers. To further remove outliers within the clusters, we use their median as $t_{std}(f,m)$, resp. $t_{thr}(f,m)$.

An example of these results is depicted in Figure 4. Even though the runtimes $t_{std}(f,m)$ vary, the major part of them clusters around 212 reference cycles. Please note that the diagram uses logarithmic axes.

To determine the distance between two throttling events $T_{thr}(f,m)$, we go through the initial unsorted list of measured runtimes and identify the time difference between loops with a runtime $t^i(f,m)$, where $t^i(f,m) >= t^{min}_{thr}(f,m)$ and $t^i(f,m) <= t^{max}_{thr}(f,m)$. Afterwards we use the median of these time differences as $T_{thr}(f,m)$.

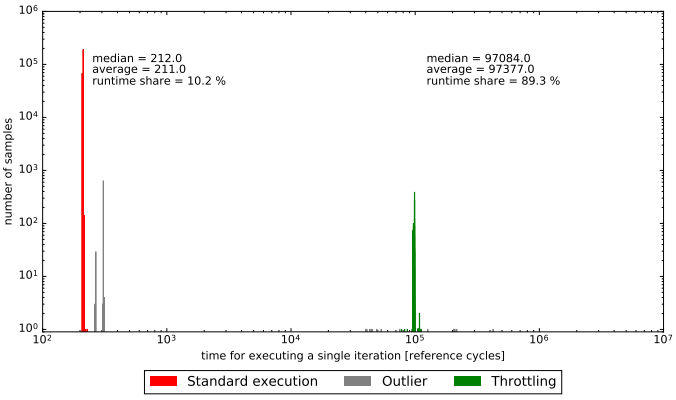
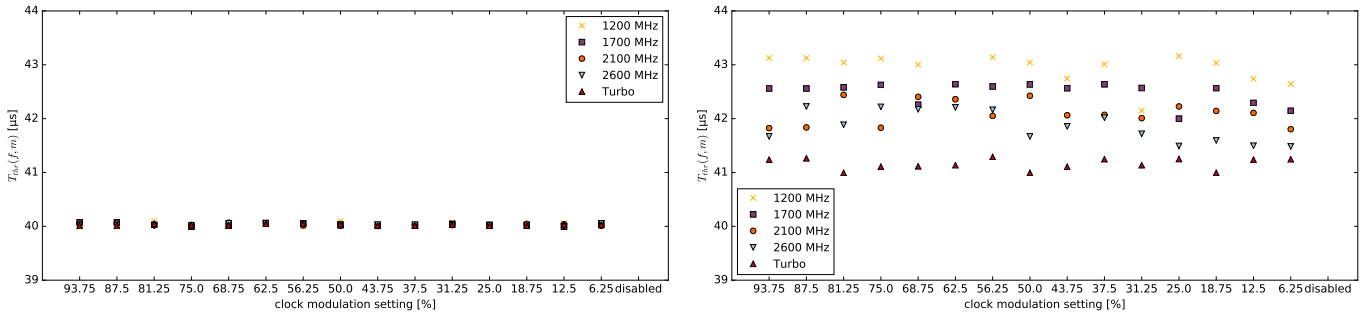



Fig. 4: Result distribution on Intel Haswell server processor ($f$=2600 MHz, $m$=93.75 %) for short time scale measurements.




(a) On Sandy Bridge and Ivy Bridge test systems, $T_{thr}(f, m)$ is not influenced by $f$ and $m$. The figure relates to Sandy Bridge EP sytem.

(b) On Haswell and Skylake processors, $T_{thr}(f, m)$ varies significantly and is influenced by the applied frequency $f$. The figure depicts results from a Haswell-EP test system.

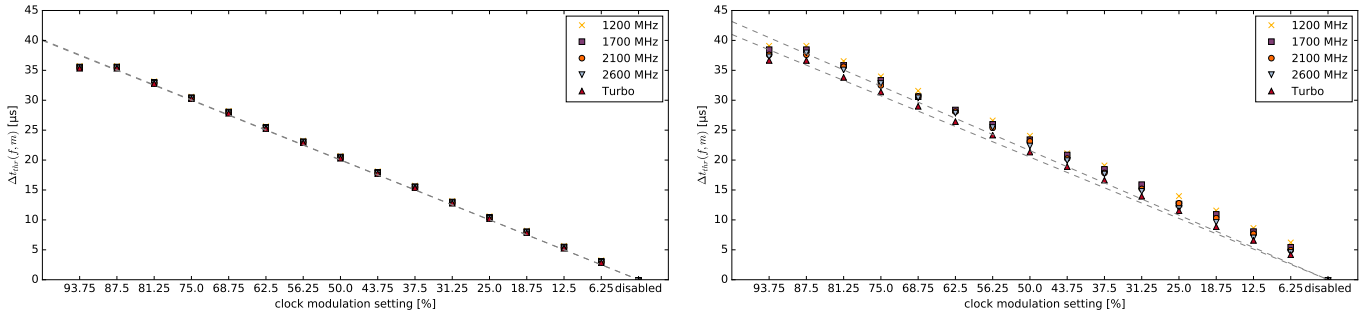
Fig. 5: The clock modulation window $T_{thr}(f, m)$, which includes a period of clock stop assertion and clock stop desertion is between 40 and 45 microseconds on all examined architectures.

### B. Results

The period of throttling events $T_{thr}(f, m)$ is independent from $f$ and $m$ on Sandy Bridge and Ivy Bridge processors and varies between 40.5 and 43.5 µs on Haswell and Skylake processors. This is depicted in Figure 5. On the newer architecures, $T_{thr}(f, m)$ increases with a lower core frequency.

In an ideal implementation, the clock modulation setting $m$ will be directly translated to the clock modulation assertion time $\Delta t_{thr}(f, m)$ so that the processor will not execute cycles for the respective share of $T_{thr}(f, m)$. For a theoretical clock modulation setting of 100 %, this share would result in $\Delta t_{thr}(f, m)$ being $T_{thr}(f, m)$, for disabled clock modulation $\Delta t_{thr}(f, m)$ would be 0. The remaining clock modulation settings should provide a $\Delta t_{thr}(f, m) = m * T_{thr}(f, m)$. As we show in Figure 6, the resulting throttling times do not follow the ideal, but $\Delta t_{thr}(f, m)$ is higher than expected for all clock modulation settings $<$ 93.75 %. On Haswell and Skylake architectures, the difference between the ideal and measured throttling time decreases with a higher clock modulation setting while it is almost constant for Sandy Bridge and Ivy Bridge processors. Furthermore, the highest clock modulation setting (93.75 %) provides the same results as the second highest (87.5 %). This can be seen for all architectures except the Skylake Desktop processor, where the final clock modulation step increases $\Delta t_{thr}(f, m)$ by approx. 2 % (depending on the frequency) compared to the previous setting of 87.5 % (not depicted).

Our results also explain the significant energy efficiency savings when using clock modulation on Sandy Bridge or Ivy Bridge processors as presented in [10], [11], and [12]. If the clock modulation setting is equal among *all cores* and can be represented as a frequency (i.e., if the original frequency is high enough and the clock modulation is not too high), the processor uses DVFS instead. For example, for $f$=2600 MHz, the runtime $t_{std}$ of the loop is 83.1 ns, which does not change when a single core applies clock modulation. This can be seen in Table II. When all cores use a common clock modulation value, e.g., 12.5 %, $t_{std}$ increases to 96.9 ns which would correspond to $f$=2230 MHz. No clock modulation can be observed. When the targeted performance of the common




(a) On Sandy Bridge processors, $T_{thr}(f, m)$ is constant for all $f$ and $m$. Thus the lines for the minimal and maximal expected $\Delta t_{thr}(f, m)$ overlap.

(b) $T_{thr}(f, m)$ varies significantly on Haswell processors, depending on $f$ and $m$. Still, most of the measured $\Delta t_{thr}(f, m)$ are above the expected range. $\Delta t_{thr}(f, m)$ is higher than expected, except for $m$=93.75 %.

Fig. 6: The clock modulation signal assertion time $\Delta t_{thr}(f, m)$ is higher than described in the processor manual with the exception of a 93.5 % clock modulation setting. The gray dashed lines depict the expected maximal and minimal $\Delta t_{thr}(f, m)$, based on the assumption that $\Delta t_{thr}(f, 100\%) = T_{thr}(f, m)$ and $\Delta t_{thr}(f, disabled) = 0$

TABLE II: Sandy Bridge EP loop runtimes. When all cores apply a common clock modulation setting, DVFS is used alternatively which increases $t_{std}$. This behavior applies only to Sandy Bridge and Ivy Bridge processors.

| Freq. [MHz] | cores | Result | (93.75) 87.5 | ex. 2 81.25 | 75 | 68.75 | 62.5 | 56.25 | **Clock modulation setting [%]** 50 | 43.75 | 37.5 | 31.25 | 25 | 18.75 | ex. 1 12.5 | 6.25 | dis-abled |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2600 | one | $t_{std}$[µs] | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 | 0.0831 |
| | all | | 0.18 | 0.18 | 0.18 | 0.18 | 0.18 | 0.18 | 0.166 | 0.153 | 0.135 | 0.127 | 0.112 | 0.105 | 0.0969 | 0.09 | 0.0831 |
| | one | $t_{thr}$[µs] | 35.5 | 32.9 | 30.4 | 28 | 25.4 | 23 | 20.4 | 17.9 | 15.5 | 12.9 | 10.4 | 7.97 | 5.41 | 3.01 | - |
| | all | | 28.4 | 23.4 | 18.3 | 13.3 | 8.38 | - | - | - | - | - | - | - | - | - | - |
| 2000 | one | $t_{std}$[µs] | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 | 0.108 |
| | all | | 0.18 | 0.18 | 0.18 | 0.18 | 0.177 | 0.18 | 0.18 | 0.18 | 0.18 | 0.166 | 0.144 | 0.135 | 0.127 | 0.12 | 0.108 |
| | one | $t_{thr}$[µs] | 335.6 | 33 | 30.5 | 28.1 | 25.5 | 23.1 | 20.6 | 18 | 15.6 | 13 | 10.5 | 8.07 | 5.51 | 3.11 | - |
| | all | | 30.8 | 28.4 | 23.4 | 18.3 | 15.9 | 10.8 | 5.82 | 3.42 | - | - | - | - | - | - | - |
| 1200 | one | $t_{std}$[µs] | 0.18 | 0.18 | 0.18 | 0.18 | 0.18 | 0.18 | 0.18 | 0.18 | 0.177 | 0.18 | 0.18 | 0.18 | 0.18 | 0.18 | 0.18 |
| | all | | 0.18 | 0.18 | 0.18 | 0.177 | 0.18 | 0.18 | 0.18 | 0.18 | 0.177 | 0.18 | 0.18 | 0.18 | 0.18 | 0.18 | 0.18 |
| | one | $t_{thr}$[µs] | 35.9 | 33.3 | 30.8 | 28.4 | 25.8 | 23.4 | 20.9 | 18.3 | 15.9 | 13.3 | 10.8 | 8.38 | 5.82 | 3.42 | - |
| | all | | 35.9 | 33.3 | 30.8 | 28.4 | 25.8 | 23.4 | 20.9 | 18.3 | 15.9 | 13.3 | 10.8 | 8.38 | 5.82 | 3.42 | - |

clock modulation setting is lower than the lowest supported frequency, clock modulation is used in addition to DVFS. For example, if a clock modulation setting of 81.25 % is applied at $f$=2600 MHz, the standard runtime $t_{std}$ increases to 180 ns, which indicates a processor frequency of 1200 MHz. Here, DVFS reduces the performance to 46.2 %. In addition, a clock modulation time $t_{thr}$ of 23.4 µs is introduced, which reduces the total average performance to 19.2 % relative to the baseline with no clock modulation. This is close to the 18.75 % performance target. Please note that this behavior only applies to Sandy Bridge and Ivy Bridge processors and could not be observed on other processors. Thus, the efficiency results in [10], [11], and [12] will significantly change on newer architectures.

On Sandy Bridge and Ivy Bridge architecture, the first clock modulation is executed 12.5 µs after its activation by software. On newer architectures, this initialization delay is $T(f,m)-\Delta t(f,m)$. Here, the activation trigger can be seen as falling edge of the clock modulation signal. On Sandy Bridge processors, the assertion is deactivated 17 µs after software triggers the register. Thus, clock modulation phase can be executed (partially) afterwards. On Haswell processors, we could not observe any residual clock modulation activity.

Our final observation targets the synchronicity of throttling events. Here we use an OpenMP parallel version of our measurement loop and store the runtimes of each thread. In Figure 7, we depict a measurements of the Haswell-EP system with $f$=2000 MHz and $m$=6.25 %. The results indicate that there is no synchronization between the clock modulation mechanisms of the single cores. A repeated experiment provided a completely different pattern.

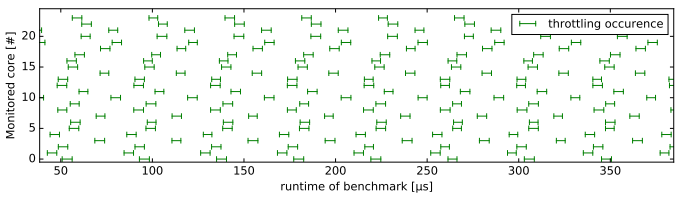



Fig. 7: Clock modulation pattern for all cores on a dual socket Haswell EP system.

## V. Sustained Performance and Power Characteristics in Comparison with DVFS

The effects that we described in the previous Section affect the performance and power consumption. In this Section we describe the sustained effect of applying clock modulation and compare it with DVFS.

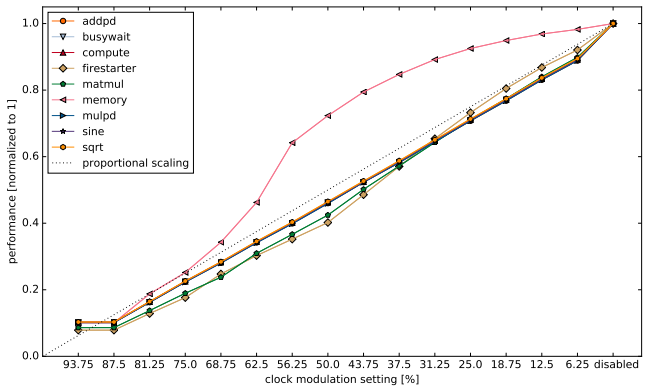
### A. Performance Measurements

The naïve assumption is that performance scales proportionally with the clock modulation setting. However, results from the benchmark described in Section III-B reveal, that this is not the case in practice. Figure 8a shows the normalized sustained performance for various workloads on a Haswell-EP system.

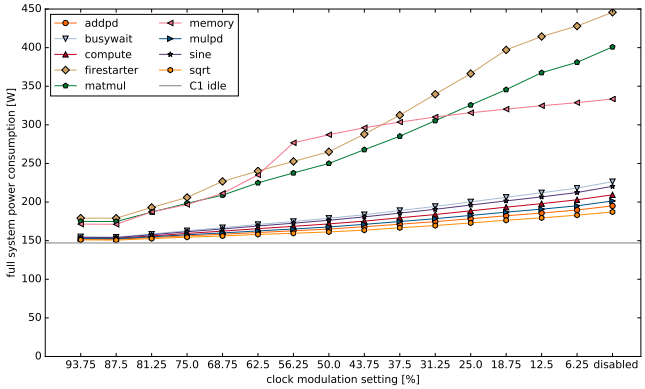
Memory-intense workloads deviate the most from the proportional scaling. Especially the workload that performs all operations in memory can retain more performance at medium clock modulation settings. In this scenario, multiple cores are still accessing the memory while others are inactive so that the memory subsystem as the main bottleneck is kept busy. For a reduced number of threads, the effect vanishes and the memory kernel behaves similar to the others. Compared to concurrency throttling where the number of active threads is reduced statically, the number of active threads changes dynamically depending on the phase shift between the clock modulation signals of different cores.

The kernels that do not access the main memory (`addpd`, `busywait`, `compute`, `mulpd`, `sine`, and `sqrt`) show identical performance scaling slightly below the ideal proportional expectation. One exception is that the 93.75 % clock modulation setting behaves exactly like the 87.5 % setting. These observations are in accordance with the results described in Section IV-B.

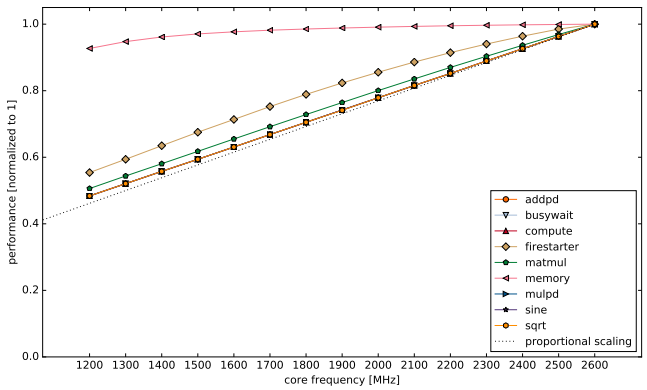
While the basic effect for non-memory workloads is consistent across architectures, frequency settings, and core counts, the exact quantity of the deviation from proportional scaling varies. For memory workloads, the patterns vary strongly between architectures, frequency settings, and core counts. Considering that most applications work in memory at least partially, it is best to measure the actual performance of the specific application workload at different settings. Assuming proportional performance can include significant errors.



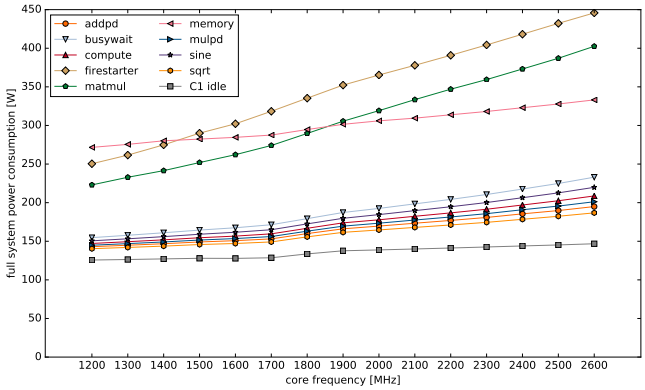

(a) Performance for clock modulation




(b) Power consumption for clock modulation




(c) Performance for frequency scaling




(d) Power consumption for frequency scaling

Fig. 8: Sustained power and performance characteristics of clock modulation at nominal frequency (2600 MHz) and frequency scaling with disabled clock gating for various workloads on a Haswell-EP system running at 24 threads.

### *B. Power Measurements*

The effect of clock modulation on the overall system power is shown in Figure 8b. Similarly to performance scaling, non-memory workloads exhibit a linear scaling of power consumption, while memory-intense workloads show a non-linear pattern. While the memory workload looses less performance from medium clock modulation, also less power is saved. Overall, the data indicates that the power consumption during a clock modulation signal assertion equals the idle power in C-State 1 (C1 idle). For non-memory kernels, a linear regression over all clock modulation settings except 93.75 % accurately predicts the power consumption of a theoretical 100 % clock modulation to be the C1 idle power.

### *C. Comparison with DVFS*

DVFS has the following key differences to clock modulation: First, it provides significant higher power savings as described in Section II. By changing the voltage in addition to the frequency, non-linear power savings can be achieved. Second, the granularity on which DVFS can be applied is coarser. Until the Ivy Bridge-EP processor generation, DVFS could only be set for each processor, while clock modulation is always available as per-core setting. Beginning with the Haswell-EP processor generation, per-core DVFS is possible, but the transition times have increased significantly [14]. Finally, reducing the frequency does not impact the latency to external events (e.g. interrupts), while clock modulation causes a delay if the interrupt arrives during a throttling phase.

Figure 8c shows the relative performance under frequency scaling as comparison to Figure 8a. Compared to clock modulation, the performance scaling of DVFS is slightly better. While the scaling is perfectly proportional for non-memory workloads, the memory workload shows almost no performance loss from reducing the frequency to a minimum. This is in accordance with measurements from [14].

The power reduction that can be achieved with DVFS is shown in Figure 8d. While generally similar, the non-memory workloads consume less power at minimal frequency than at a similarly performing clock modulation setting. As expected from the fundamental principles, the sustained energy efficiency is always better for DVFS than for clock modulation. Therefore the two advantages of clock modulation are that it can be changed almost instantaneously, while DVFS changes can take longer to become effective (up to 500 µs on Haswell-EP) and that it is always available as per-core setting even on older processor generations and Desktop processors.

## VI. Model for Optimizations

The data that we compiled in the previous Sections can be used to explain the effects of existing optimization approaches or to create a new optimization technique based on a deep understanding of the underlying architecture. In this example, we focus on the latter by providing a model for optimizing synchronization steps, e.g., `MPI_Barriers`, with the goal to reduce the total energy consumption of the application. We apply this model on the Haswell-EP architecture to determine when clock modulation or DVFS should be used.

In order to optimize energy consumption, we alter the system state during the synchronization step. The reference state is $f_{ref} = 2600\,\text{MHz}$ and $m_{ref} = 0\,\%$ (disabled). The used optimization setting is $f_{opt} = 1200\,\text{MHz}$ for DVFS, resp. $m_{opt} = 93.75\,\%$ for clock modulation. We used the generic model depicted in Figure 9. $s(\alpha, \beta)$ is the time during which the system is blocked in order to initiate the switch from state $\alpha$ to state $\beta$. Latency $l(\alpha, \beta)$ is the time, the system remains in state $\alpha$ after initiating a switch to $\beta$. $\mathcal{P}(\alpha, \beta)$ is the relative performance in state $\alpha$ compared to state $\beta$.

### A. Performance

Unlike computation regions, synchronization steps do not have to process a given amount of load, but the progress of the program is delayed until the synchronization signal is received. Therefore, the performance of an optimized execution of the synchronization step only decreases when the processor is not able to process the incoming signal when it arrives. In addition, the switch back to the reference state at the end of the synchronization implies a performance loss on the subsequent computation region. This loss is modeled with the switching delay $s(opt, ref)$ at the end of the synchronize step and the excess time to finish the work package of the compute region, i.e., $d = [1 - \mathcal{P}(opt, ref)] \cdot l(opt, ref)$. In general, the worst case performance loss is equal to $s(opt, ref) + d$. However, this delay can slow down the execution of the whole program while power is only saved locally in one thread of execution. Thus, these optimizations should not be executed on the critical path.

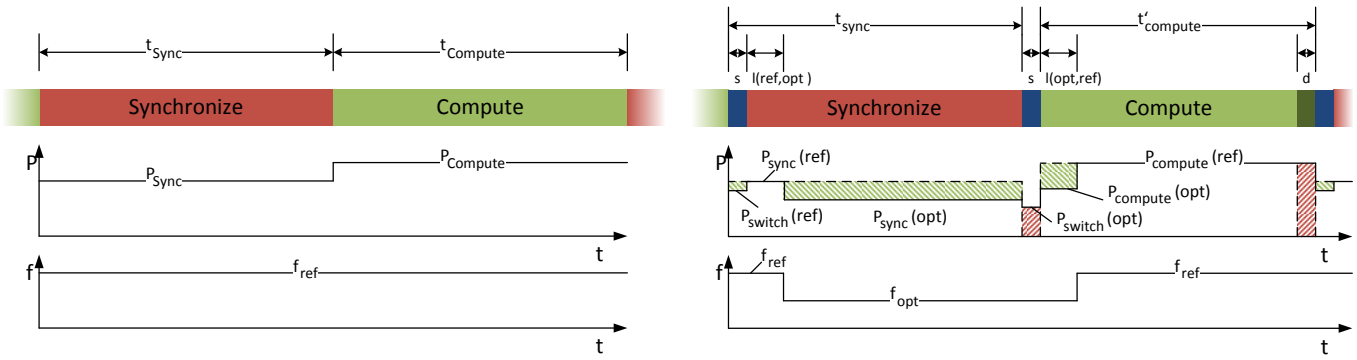
We assume, that the actual enabling and disabling of optimization setting in software can be done instantaneously. Thus, the DVFS delay $s(f_{opt}, f_{ref})$ is zero. According to [14], $l(f_{ref}, f_{opt})$ and $l(f_{opt}, f_{ref})$ are on average 272.5 µs. For clock gating, there is no single value for $s(m_{opt}, m_{ref})$, as it depends on the time, when the synchronization signal is received relative to the clock modulation assertion. We use a mean value of $\bar{s}(m_{opt}, m_{ref}) = \frac{1}{2} \cdot \Delta t_{thr}(f_{ref}, m_{opt}) = 16.67\,\mu\text{s}$, which equals the expectancy for an average clock modulation assertion. On Haswell processors, the clock modulation is applied without additional latency, so we assume $l(m_{ref}, m_{opt})$, $l(m_{opt}, m_{ref})$, and subsequently $d(m_{opt}, m_{ref})$ to be zero.

### B. Energy consumption

In the following, we discuss how clock modulation compares to DVFS in terms of energy efficiency. While a DVFS optimization can lead to a performance loss of the following region, clock modulation can lead to a prolonged execution of the optimized area itself. Therefore, we define the energy consumption that is needed for an optimized region (e.g. a barrier, that is slowed down for optimization purposes) as the energy used in this region plus the energy difference that might occur from echo effects of the optimization (e.g., a delayed resetting of the frequency).

Thus, the difference of the energy consumption can be calculated as the sum of the difference in energy consumption between the reference and the optimized region $\Delta E_{sync}$, and the difference in energy consumption implied on the following computation region $\Delta E_{compute}$. Additionally, the energy impact $\Delta E_{switch}$ of the state switch between both regions has to be considered. The model shown in Figure 9 is used to derive the formulae for these values given in Equations (2), (3), and (4).

$$\begin{aligned} \Delta E_{sync} &= [P_{switch}(ref) - P_{sync}(ref)] \cdot s(ref, opt) \\ &+ [P_{sync}(opt) - P_{sync}(ref)] \cdot \\ &\quad [t_{sync} - s(ref, opt) - l(ref, opt)] \quad (2) \\ \Delta E_{switch} &= P_{switch}(opt) \cdot s(opt, ref) \quad (3) \\ \Delta E_{compute} &= [P_{compute}(opt) - P_{compute}(ref)] \cdot l(opt, ref) \\ &+ d \cdot P_{compute}(ref) \quad (4) \end{aligned}$$




(a) default execution of an example application which waits for an external event in a region that might be optimized

(b) model of the optimization, with different performance losses in form of the delay $s$, the latency $l$ and the excess computation time $d$.

Fig. 9: Model of the default and optimized execution of a synchronization step in an example application.

TABLE III: Measured parameters of the energy model

| | Reference | DVFS | Clock Modulation |
|---|---|---|---|
| $\mathcal{P}$ | 1 | 0.53798 | 0.09876 |
| $P_{sync}$ [W] | 233.3 | 152.4 | 155.5 |
| $P_{compute}$ [W] | 320.1 | 176.6 | 165.1 |
| $P_{switch}$ [W] | - | - | 147.524 |

Based on the measured parameters from our test system (Table III) and the sum of the equations for $\Delta E_{sync}$, $\Delta E_{switch}$, and $\Delta E_{compute}$, we build a model for energy savings with DVFS and clock modulation. Combining these results leads to two linear functions describing the amount of energy saved for each optimization technique. Those two models are given in Equation (5) and (6).

$$\Delta E_{DFVS} = 21.4\,mJ - 80.9\,W \cdot t_{sync} \quad (5)$$

$$\Delta E_{CM} = 2.5\,mJ - 77.7\,W \cdot t_{sync} \quad (6)$$

With these models we can estimate the minimal duration for the synchronization step to apply an optimization to be 263.8 µs and 31.6 µs for DFVS and clock modulation, respectively. Hence, clock modulation should be preferred over DVFS for shorter synchronization steps. With an increased runtime of the synchronization, DVFS provides a higher effect on energy consumption. The break-even point is calculated to be at 5999.4 µs.

Thus, on our test system, synchronization steps with a duration of less than 31.6 µs should not be optimized at all. For a duration greater than 6 ms, DVFS should be used. Otherwise, clock modulation is the best choice.

## VII. Conclusion and Outlook

There are valid reasons to use clock modulation for energy efficiency optimization: (1) The granularity of this power saving technique is per-processor-core, which makes it superior to DVFS when the processor does not support per-core performance states. (2) The latency for enabling and disabling clock modulation is significantly lower compared to the usage of DVFS on some architectures. (3) It can be used in addition to DVFS to enable even more active power saving states, where a processor core does not need an external signal to switch back to a high performing performance state. However, the usage of clock modulation has to be considered with the implementation details in mind. In this paper we have shown that the implementation of software controlled clock modulation differs significantly between different processor architecture and from the description in the provided processor manuals. While on Sandy Bridge and Ivy Bridge architectures the processors use DVFS instead of clock modulation when all cores agree to a specific clock modulation setting, newer architectures like Haswell and Skylake deviate to a higher degree from the specified target performance. However, we have shown that clock modulation is a valid optimization technique for synchronization based optimization that should be used as an alternative to DVFS for short running synchronization steps. Future work will include a survey of the Xeon Phi processor, codenamed Knights Landing. This processor also has a low DVFS granularity but supports clock modulation per tile (a group of 2 processor cores) according to initial measurements.

## Acknowledgment

This work has been funded in a part by the German Research Foundation (DFG) in the Collaborative Research Center "Highly Adaptive Energy-Efficient Computing" (HAEC, SFB 912) and by the European Union's Horizon 2020 Programme in the READEX project under grant agreement number 671657. The authors would like to thank Sven Schiffner for his support.

## References

[1] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, 1999, DOI: 10.1109/40.782564.

[2] "Advanced configuration and power interface (acpi) specification, revision 6.1," 2016, Available online at uefi.org (2016-08-05).

[3] R. Schöne, D. Molka, and M. Werner, "Wake-up latencies for processor idle states on current x86 processors," *Computer Science - Research and Development*, 2014, DOI:10.1007/s00450-014-0270-z.

[4] V. Pallipadi and A. Starikovskiy, "The ondemand governor past, present, and future," in *Proceedings of the Linux Symposium*, 2006, Available online at kernel.org (2016-08-05).

[5] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design - A Circuits and Systems Perspective, 4th Edition*. Pearson, 2011.

[6] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A, 3B, and 3C: System Programming Guide*, Available online at Intel.com (2016-08-05).

[7] E. Rotem, A. Naveh, M. Moffie, and A. Mendelson, "Analysis of thermal monitor features of the intel pentium m processor," in *Workshop on Temperature-Aware Computer Systems*, 2004.

[8] *Intel Core i7-800 and i5-700 Desktop Processor Series and LGA1156 Socket Thermal/Mechanical Specifications and Design Guidelines*, Intel, 2009, Available online at Intel.com (2016-08-05).

[9] *Desktop 3rd Generation Intel Core Processor Family, Desktop Intel Pentium Processor Family, Desktop Intel Celeron Processor Family, and LGA1155 Socket Thermal Mechanical Specifications and Design Guidelines (TMSDG)*, Intel, 2013, Available online at Intel.com (2016-08-05).

[10] S. Bhalachandra, A. Porterfield, and J. Prins, "Using dynamic duty cycle modulation to improve energy efficiency in high performance computing," in *IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, 2015, DOI: 10.1109/IPDPSW.2015.144.

[11] P. Cicotti, A. Tiwari, and L. Carrington, "Efficient speed (es): Adaptive dvfs and clock modulation for energy efficiency," in *IEEE International Conference on Cluster Computing (CLUSTER)*, 2014, DOI: 10.1109/CLUSTER.2014.6968750.

[12] W. Wang, A. Porterfield, J. Cavazos, and S. Bhalachandra, "Using per-loop cpu clock modulation for energy efficiency in openmp applications," in *International Conference on Parallel Processing (ICPP)*, 2015, DOI: 10.1109/ICPP.2015.72.

[13] A. Mazouz, A. Laurent, B. Pradelle, and W. Jalby, "Evaluation of CPU frequency transition latency," *Computer Science - Research and Development*, 2013, DOI: 10.1007/s00450-013-0240-x.

[14] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," in *International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, 2015, DOI: 10.1109/IPDPSW.2015.70.