Noname manuscript No.
(will be inserted by the editor)

Evaluation of CPU Frequency Transition Latency

Abdelhafid Mazouz · Alexandre Laurent · Benoît Pradelle · William Jalby

Abstract Dynamic Voltage and Frequency Scaling (DVFS) is one of the most employed techniques to reduce energy consumption in computers. The main idea exploited by DVFS controllers is to reduce the CPU's frequency whenever it is less intensively used to save energy. For example, memory intense program phases are good targets for frequency reduction. However, depending on the CPU model, switching the frequency can be performed in varying delays. Such delays are often optimistically ignored in DVFS controllers, whereas their knowledge could enhance the quality of frequency setting decisions.

The following document presents FTaLaT (Frequency Transition Latency measurement Tool), a tool able to measure the frequency transition latency directly on the computer. The measurement methodology is also presented, accompanied by evaluations on three recent Intel processors.

Keywords DVFS · Statistical performance evaluation · Frequency transition latency

1 Introduction

Energy is now considered as a determinant topic for computer science. Driven by autonomy constraints, embedded platforms pioneered, but more recently, major concerns have been raised in unexpected domains. It is the case for instance in high-performance computing where the supercomputers power consumption has reached levels preventing the current technologies to be used in the next generation of supercomputers.

Among the different optimizations applied to reduce energy consumption, Dynamic Voltage and Frequency Scaling (DVFS) has proven to be one of the most successful techniques [Ge et al, 2007; Hsu and Feng, 2005; Wu et al, 2005]. The core idea of DVFS is to dynamically adapt processors frequency and input voltage in order to reduce power consumption. Common DVFS controllers exploit the slack time of programs provoked by memory accesses or I/O and decrease the processor speed during such phases, effectively reducing the total energy consumption without harming the program execution time.

In most of the existing DVFS controllers, the latency required to transition the frequency is considered as a negligible quantity. However, several DVFS controllers are based on frequent periodic polling of resource usage. They must correctly estimate the CPU frequency transition latency in order to determine an efficient polling period. Additionally, some other approaches consider theoretical modeling of hardware and can benefit from a precise estimation of the transition latency [Snowdon, 2010]. The frequency transition latency is then an important information for many DVFS controllers, among other potential usages.

We propose FTaLaT, a new frequency transition latency estimator. Using precise measurements under different frequency settings, FTaLaT can quickly provide a precise estimation of all the possible frequency transition latencies. FTaLaT is freely distributed as open source software at: `http://code.google.com/p/ftalat`.

2 How does it works?

The goal of FTaLaT is to measure the delay between the request for a new frequency and the actual frequency transition. To do so, it executes a specifically designed short program, called *micro benchmark*, or *kernel*. The benchmark is run several times to determine its execution time with the initial and target frequencies. Then, starting from the initial frequency, the transition towards the target frequency is trig-

Abdelhafid Mazouz · Alexandre Laurent · Benoît Pradelle · William Jalby
Univ. Versailles St-Quentin en Yvelines E-mail: first.last@uvsq.fr

Listing 1 Micro benchmark assembly instructions

```
start_time_counter
    addl $1,%%eax
    addl $1,%%eax
    addl $1,%%eax
    addl $1,%%eax
    ...
stop_time_counter
```

gered and the benchmark is repeatedly run while measuring its execution time. Once the benchmark execution time is close to what is expected for the target frequency, FTaLaT detects a frequency transition and deduces the transition latency.

The benchmark used for the measurement is a simple list of `add` assembly operations, as presented in Listing 1. With such structure, the benchmark exhibits a repeatable behavior and is sensitive to frequency transitions as it is only made of arithmetic operations. The benchmark is then a perfect target program to detect actual CPU transitions from execution time variations.

The main issue with such methodology is to correctly detect a frequency transition from execution times. Indeed, background tasks may wakeup at anytime on the computer for instance, perturbing the execution of the micro benchmark [Mazouz et al, 2010; Todd Mytkowicz and Amer Diwan and Peter F. Sweeney and Mathias Hauswirth, 2009]. As the benchmark runs are extremely short, it may then be difficult to distinguish between measurement noise and actual frequency transitions. To solve that issue, FTaLaT relies on statistical tests.

FTaLaT uses confidence intervals [Raj Jain, 1991] to determine if an execution time can be considered as close to what was observed previously at a given frequency. More precisely, confidence intervals define a range of values one can expect to be a good estimate of a population. They are computed from the mean of several measurements, their standard deviation, and a constant called the t distribution value that depends on the desired confidence level. The resulting confidence intervals are used for two main purposes. First, an execution time can be considered as close to others if it belongs to their confidence interval. Second, a test called the two samples t-test can determine if two sets of values are distinct or not. Such usages are perfectly suited to the needs of FTaLaT.

Thanks to the statistical tools at its disposal, FTaLaT can determine if the execution times of the benchmark with the initial and target frequency are distinct or not. It can then ignore the cases where the benchmark execution times with both frequencies are too close to conclude. It also determines if an execution time is similar to those achieved with the target frequency in order to detect a frequency transition. Finally, FTaLaT uses the t-test to determine if a detected transition is a measurement noise or if it is confirmed over a long period. In our implementation, confidence intervals are built for a 95 % confidence level. A more detailed methodology is presented afterwards.

The general FTaLaT algorithm is as follows. First of all, the micro benchmark is run 10,000 times while the initial frequency is set, and again 10,000 times with the target frequency. Then, both measurement sets are compared using the t-test: if they are not found distinct, the procedure ends as FTaLaT is not able to distinguish between execution times. In practice, this never happened during our experiments. Besides, the confidence interval of the mean of the target frequency is built. Then, the initial frequency is set, the target frequency is requested, and the micro benchmark is repeatedly executed while measuring its execution time. After every execution, if the resulting execution time lies in the confidence interval for the target frequency, a frequency transition is assumed. At this point, the transition is not certain as the execution time may have changed because of various external events. In order to ensure that the transition actually occurred, the micro benchmark is run again 100 times while measuring its execution time. Using the t-test, the set consisting of the 100 final execution times is compared to that of the target frequency. If they are determined similar, FTaLaT considers that a frequency transition actually happened and records the transition latency. Otherwise, the measurement is started over as it may have been disturbed by external noise. As a result, either FTaLaT aborts when the stability conditions are not met, or a frequency transition delay is determined.

For more details about the statistical methods employed, or the detailed algorithm, the reader is referred to the scientific paper presenting the system [Mazouz et al, 2013].

3 Experiments

3.1 Experimental setup

Our experiments have been conducted on three distinct machines, presented in Table 1, running a recent Linux system. On each machine, FTaLaT measured the frequency transition latency between every available frequency pair. For statistical significance, each measurement was repeated 31 times while an effort was made to reduce the sources of performance instability on our measurements: to achieve high precision in our measurements, we use thread affinity for better performance stability and the time stamp counter, via the `RDTSC` instruction, for precise frequency-independent time measurements.

Processor	Xeon X5650	Xeon E3-1240	Core i7-3770
CPU type	Intel Core Westmere	Intel Core SandyBridge	Intel Core IvyBridge
Micro-architecture	Nehalem	SandyBridge	IvyBridge
Cores	2x 6	1x4	1x 4
Hardware threads	2x 6	1x4	1x 8
Min CPU Frequency	1.59 GHz	1.6 GHz	1.6 GHz
Max CPU Frequency	2.66 GHz	3.3 GHz	3.4 GHz

Table 1 Test machines description

3.2 Experimental results

Figures 1, 2, and 3 present the frequency transition latency in micro-seconds on the vertical axis for the three test machines. The horizontal axis in each figure shows the different target frequencies while the initial frequencies are represented using distinct plotting colors and symbols. For each pair of initial and target frequency, we report the delay required to achieve an effective frequency transition, as measured by FTaLaT.

On the three machines, we observe that the transition delay is not constant but rather depends on the initial and target frequencies. The transition latency usually increases when the target frequency is higher than the initial frequency. On the other hand, when the frequency decreases, the transition latency stays in a very tight range of small values. One can also notice that the transition latency does not follow a similar trend on all machines: while transition latency seems to increase linearly when CPU frequency increases on the `SandyBridge` machine, at least three steps appear on the `IvyBridge` and the `Westmere` machines. Finally, the range of possible latencies tends to tighten for more recent processor models.

The results obtained by FTaLaT are consistent with the short description found in manufacturer documentation for similar processors [Intel Corporation, 2011, 2012] where it is stated that the voltage increase induced by a frequency increase is performed in several steps while a frequency and voltage reduction is described as a one-step operation. Our tool is able to confirm that a similar behavior can be observed on our experimental platforms.

In order to have a better view of the transition latency, we have represented in Figure 4 the measured benchmark execution times on the `IvyBridge` machine when switching the CPU frequency from 1.6 GHz to 3.4 GHz. While the vertical axis reports the execution time of the successive runs of the kernel, the horizontal axis represents the different iterations until the transition is detected and confirmed. Notice our system immediately detects the transition but runs additional measurements afterwards to confirm it.

We can observe two distinct steps in execution times: 1) from iteration 1 to iteration 48 and 2) from iteration 50 to 149. The first step represents the executions at the 1.6 GHz frequency, while the second step occurs when the benchmark runs at 3.4 GHz. Thus, the CPU does not change its operating frequency until iteration 49. The transition latency computed by FTaLaT is then the elapsed time between the request for a new frequency (at iteration 1) and the kernel's execution time change detection (iteration 50).

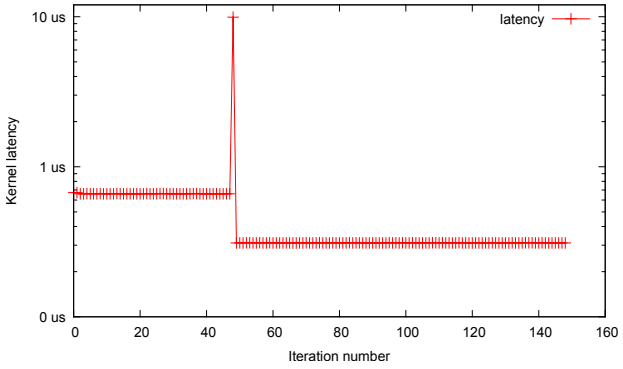

Fig. 4 Observed execution times of the assembly kernel for the pair (1.6 GHz, 3.4 GHz) of CPU frequencies on the `IvyBridge` machine.

Additionally, when executing the kernel at iteration 49, we can observe that the frequency transition provokes a short pause, leading the kernel's execution time to rise in a significant extent. Thus, aside frequency latency, there is also an overhead to transition frequency. The overhead directly impacts the execution time of the running programs as the processor can be considered as paused during the actual frequency transition. In the presented measurement, the overhead from transitioning frequencies can be deduced from the extra-time spent in the 49th iteration and represents about 9.5 µs. Ideally, such overhead should also be taken into account when performing DVFS, for instance by avoiding non-necessary frequency transitions.

4 Conclusion

FTaLaT is able to determine experimentally the frequency transition latency on modern x86_64 platforms for every couple of available frequencies. It uses a rigorous statistical approach to distinguish between measurement noise and actual information. The tool is distributed as open source software for recent Linux systems.

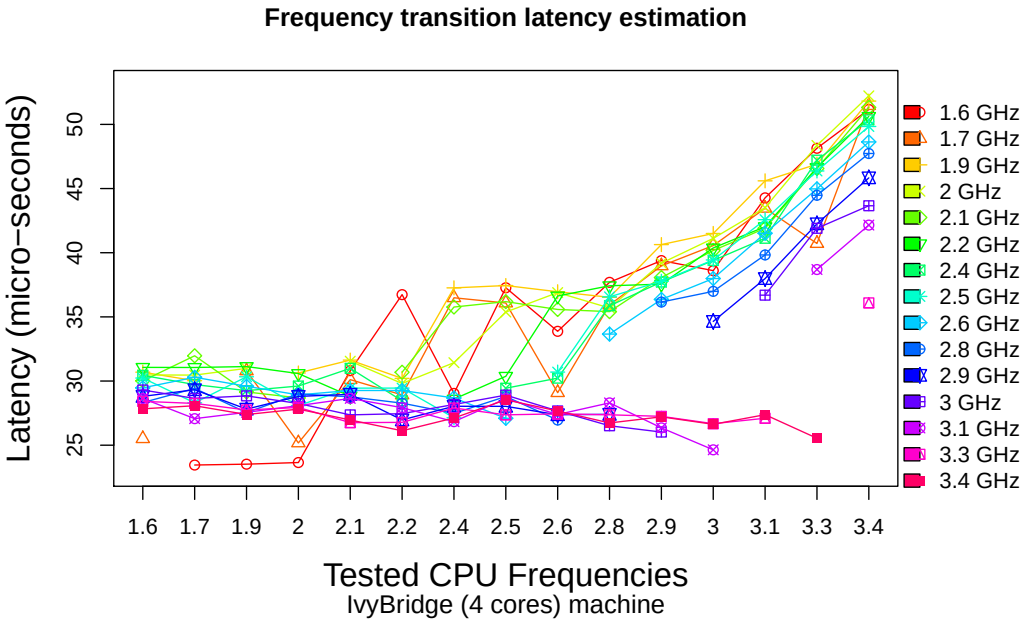

Fig. 1 Observed frequency transition latency on the `IvyBridge` machine.

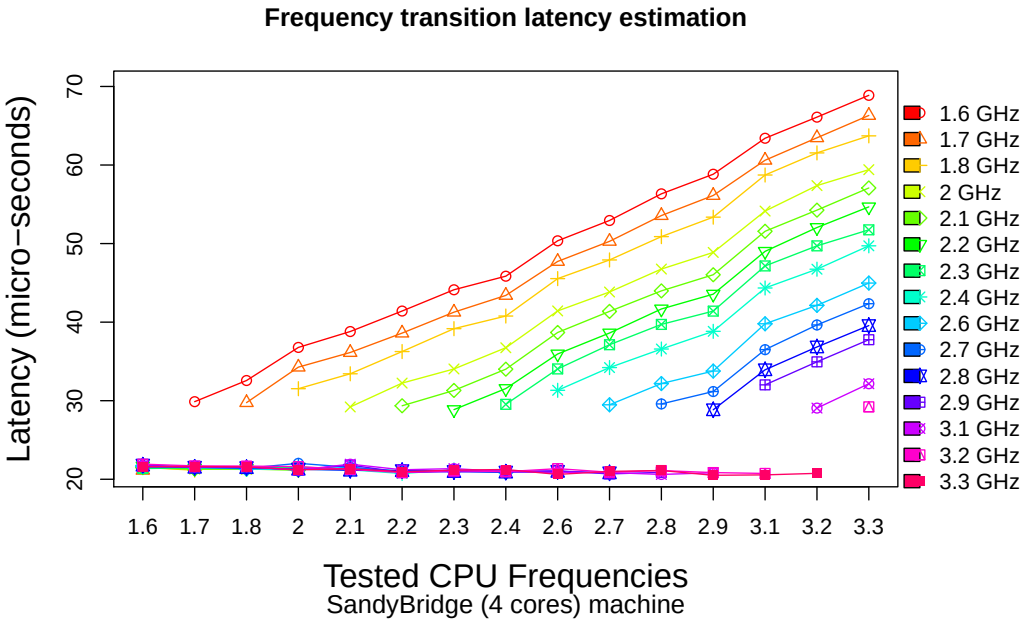

Fig. 2 Observed frequency transition latency on the `SandyBridge` machine.

From the experiments, we observed various interesting phenomena such as the variable cost of a frequency increase compared to the nearly fixed cost of a frequency decrease. FTaLaT is then of great help to measure frequency transition latency and better understand the processors. Thus, it can also be used, aside from its uses in DVFS control, to track the advances in frequency transition delays or highlight conception issues in processors.

References

Ge R, Feng X, chun Feng W, Cameron K (2007) CPU MISER: A performance-directed, run-time system for

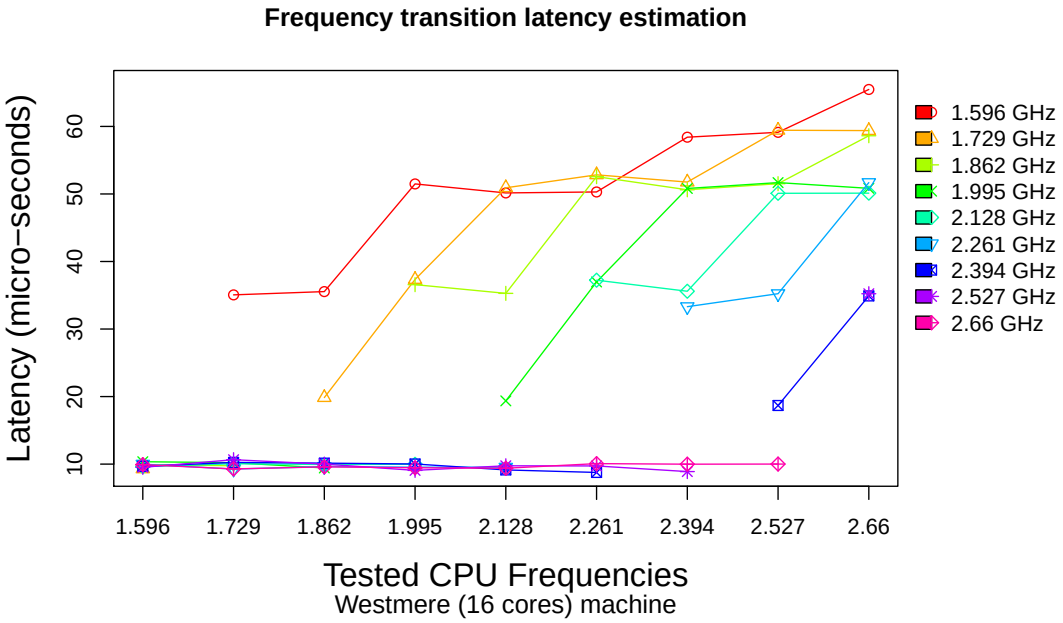

Fig. 3 Observed frequency transition latency on the `Westmere` machine.

power-aware clusters. In: ICPP 2007. International Conference on Parallel Processing, p 18, DOI 10.1109/ICPP.2007.29

Hsu Ch, Feng Wc (2005) A power-aware run-time system for high-performance computing. In: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, IEEE Computer Society, Washington, DC, USA, SC '05, pp 1–, DOI 10.1109/SC.2005.3, URL `http://dx.doi.org/10.1109/SC.2005.3`

Intel Corporation (2011) Intel xeon processor E3-1200 family datasheet

Intel Corporation (2012) Intel Xeon processor E5-1600/E5-2600/E5-4600 product families

Mazouz A, Touati SAA, Barthou D (2010) Study of variations of native program execution times on multi-core architectures. In: CISIS '10: Proc. of the International Conference on Complex, Intelligent and Software Intensive Systems, IEEE Computer Society, Washington, DC, USA, pp 919–924, DOI http://dx.doi.org/10.1109/CISIS.2010.96, MuCoCos workshop

Mazouz A, Laurent A, Pradelle B, Jalby W (2013) Evaluation of cpu frequency transition latency. Computer Science - Research and Development pp 1–9, DOI 10.1007/s00450-013-0240-x, URL `http://dx.doi.org/10.1007/s00450-013-0240-x`

Raj Jain (1991) The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modelling. John Wiley and Sons

Snowdon D (2010) Operating system directed power management. PhD thesis, University of New South Wales

Todd Mytkowicz and Amer Diwan and Peter F Sweeney and Mathias Hauswirth (2009) Producing wrong data without doing anything obviously wrong! In: Architectural Support for Programming Languages and Operating Systems (ASPLOS)

Wu Q, Martonosi M, Clark DW, Reddi VJ, Connors D, Wu Y, Lee J, Brooks D (2005) A dynamic compilation framework for controlling microprocessor energy and performance. In: MICRO, pp 271–282