Evaluation of CPU Frequency Transition Latency

Abdelhafid MAZOUZ ¹  Alexandre LAURENT ¹  
Benoît PRADELLE ¹  William JALBY ¹

¹University of Versailles Saint-Quentin-en-Yvelines, France

ENA-HPC 2013, Dresden
September 02, 2013
Outline

1 Introduction
2 Evaluation methodology
3 Experimental results
4 Conclusion
Introduction

- Power consumption is now a major concern in computing systems
- DVFS is an important technique to reduce energy consumption:
  - Dynamically adapt CPU frequency and voltage
  - Reduce CPU frequency for memory-bound programs
  - Increase CPU frequency for CPU-bound programs
Introduction

- CPU frequency switching may imply varying delays

- What about multi-phased programs?
  - Switching frequency between short phases incurs overhead
  - Need for precise estimation of transition latency

- We propose a statistical approach to measure these delays:
  - We implemented a tool called FTaLaT.
  - Is freely distributed as open source software at http://code.google.com/p/ftalat
Why CPU frequency transition latency estimation?

Each region has distinct performance/power behavior.

Two frequency sequences are used.

Up to **30% in energy savings** with effective frequency settings.
FTaLaT’s Measurement methodology

- FTaLaT automatically measures the transition latency for each pair of start and target CPU frequency:
  - Time between the request for target and start frequency

- FTaLaT measures the performance of an assembly kernel:
  - CPU-bound kernel: a set of `add` instructions
  - Sufficiently sensitive to detect frequency change
Measurement through two main steps:

1. Initialization:
   1. Measure time of the kernel when start frequency is set
   2. Measure time of the kernel when target frequency is set

2. Frequency transition latency measurement:
   1. Set CPU frequency to target
   2. Iteratively measure execution time of the kernel
   3. Stop measurement when kernel’s time change is detected
FTaLaT’s Measurement methodology

Effective evaluation methodology:

1. Precise estimation of execution time of the kernel for a given CPU frequency

2. Comparing the kernel’s performance of two samples of execution times
Estimating the execution time

- Running a program/kernel \( N \) times may lead to \( N \) distinct execution time

- Separate true performance from measurement noise

- Average or median are not sufficient: outliers

- For a fixed confidence level, building a confidence interval (CI) of the average

- Lower and upper bounds on the performance of the assembly kernel for a tested CPU frequency
Comparing the performance of two CPU frequencies

- How to decide if two samples/sets are similar/different
- A best practice: rely on a statistical test

The Student $t$-test: compares between the average execution times of two samples:
- Builds a confidence interval of the mean difference
- Samples are not different if CI includes zero
- Samples are different if CI does not include zero
Initialization phase

Measure time with the \textit{start} CPU frequency (10000 times)

Measure time with the \textit{target} CPU frequency (10000 times)

\textit{compare the average of start and target}

\textit{Student's t-test}

\texttt{yes} \hspace{1cm} \texttt{no}

average of \textit{start} and average of \textit{target} are not different?

Stop measurement

Build the CI (LB and UP) of the mean for the \textit{target} frequency
Latency estimation

Set CPU frequency to \textit{target};
Start time measurement

Repeat kernel execution

\textit{no}

Kernel's execution time in CI of the mean of \textit{target}?  

\textit{yes}

Stop time measurement;
Trigger additional measurements

Perform Student's t-test:
(Initial runs of \textit{target} against new ones)

\textit{yes}

Confidence interval of mean difference includes zero?

\textit{no}

Frequency transition not detected

Frequency transition detected;
Report transition delay

\textit{try again}
Experimental setup

<table>
<thead>
<tr>
<th>Hardware setup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
</tr>
<tr>
<td><strong>CPU type</strong></td>
</tr>
<tr>
<td><strong>Micro-architecture</strong></td>
</tr>
<tr>
<td><strong>Cores</strong></td>
</tr>
<tr>
<td><strong>Hardware threads</strong></td>
</tr>
<tr>
<td><strong>Min CPU Frequency</strong></td>
</tr>
<tr>
<td><strong>Max CPU Frequency</strong></td>
</tr>
</tbody>
</table>

**Software setup**
- FTaLaT execution is repeated 31 times for each tested start and target CPU frequency pair
- FTaLaT relies on the TSC (RDTSC instruction) for time measurement:
  - TSC is unaffected by frequency change on our test machines.
- FTaLaT uses the *userspace* Linux governor to select a given CPU frequency.
Experimental results and analysis

Frequency transition latency estimation

- Transition delay is **not constant** across our test platforms.
- Transition latency **increases** when target frequency is **higher** than the start one.
- Voltage and frequency increase performed in **multiple steps**.
**Experimental results and analysis**

### Frequency transition latency estimation

![Graph showing latency against CPU frequencies](image)

<table>
<thead>
<tr>
<th>Tested CPU Frequencies</th>
<th>Latency (micro-seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.596 GHz</td>
<td>10</td>
</tr>
<tr>
<td>1.729 GHz</td>
<td>20</td>
</tr>
<tr>
<td>1.862 GHz</td>
<td>30</td>
</tr>
<tr>
<td>1.995 GHz</td>
<td>40</td>
</tr>
<tr>
<td>2.128 GHz</td>
<td>50</td>
</tr>
<tr>
<td>2.261 GHz</td>
<td>60</td>
</tr>
<tr>
<td>2.394 GHz</td>
<td>70</td>
</tr>
<tr>
<td>2.527 GHz</td>
<td>80</td>
</tr>
<tr>
<td>2.66 GHz</td>
<td>90</td>
</tr>
</tbody>
</table>

- Transition latency is almost **similar** when target frequency is **smaller** than the start one.
- Voltage and frequency decreased in **one step**.
Experimental results and analysis

Frequency transition latency estimation

IvyBridge (4 cores) machine

- Transition latency does not increase **linearly** on IvyBridge
Experimental results and analysis

- Case study: switching frequency from 1.6 GHz to 3.4 GHz on IvyBridge
- Kernel execution times breakdown:
  1. Iterations 1 to 48: execution times at 1.6 GHz
  2. Iteration 49: transition point
  3. Iterations 50 to 150: effective frequency change

- Frequency transition latency represents the total elapsed time from iteration 1 to 50.
- Frequency overhead (iteration 49) represents the effective switching delay of frequency.
Conclusion

- **FTaLaT:**
  - Statistical estimation of CPU frequency transition latency
  - Use of CIs to determine when a CPU frequency is enforced
  - Can be downloaded at [http://code.google.com/p/ftalat](http://code.google.com/p/ftalat)

- **Observations:**
  - We observe that changing CPU frequency
    - upward leads to higher transition delays
    - downward leads to smaller/constant transition delays
  - Oldest processors generations has larger CPU frequency transition latencies compared to newest ones
Thank you for your attention.