Evaluation of CPU Frequency Transition Latency

Abdelhafid Mazouz [1] Alexandre Laurent [1]
Benoît Pradelle [1] William Jalby [1]

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

ENA-HPC 2013, Dresden
September 02, 2013

Outline

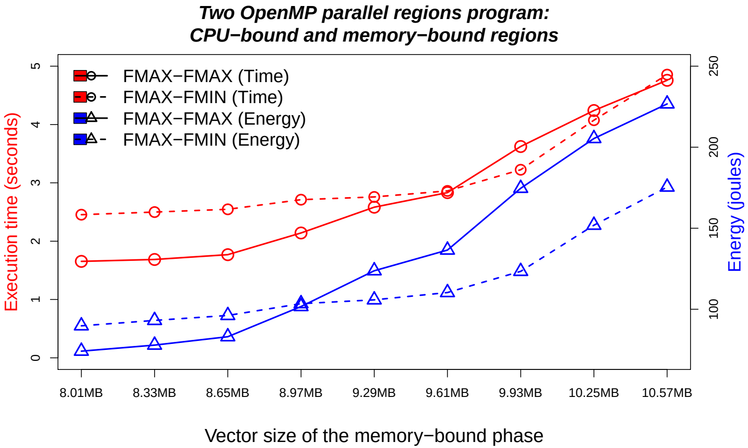
Introduction

- 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

FTaLaT's Measurement methodology

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

FTaLaT's Measurement methodology

Estimating the execution time

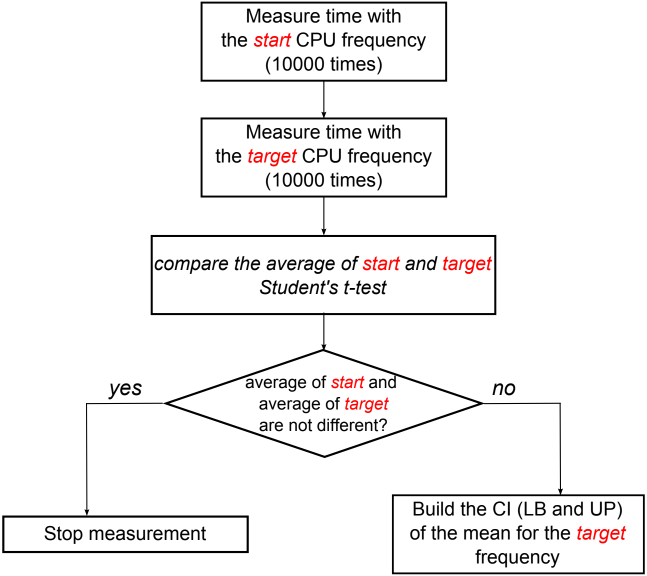
- Running a program/kernel N times may lead to N distinct execution time
- Separate true performance from measurement noise
- Average or median are not sufficient: outliers
- For a fixed confidence level, building a confidence interval (CI) of the average
- Lower and upper bounds on the performance of the assembly kernel for a tested CPU frequency

FTaLaT's Measurement methodology

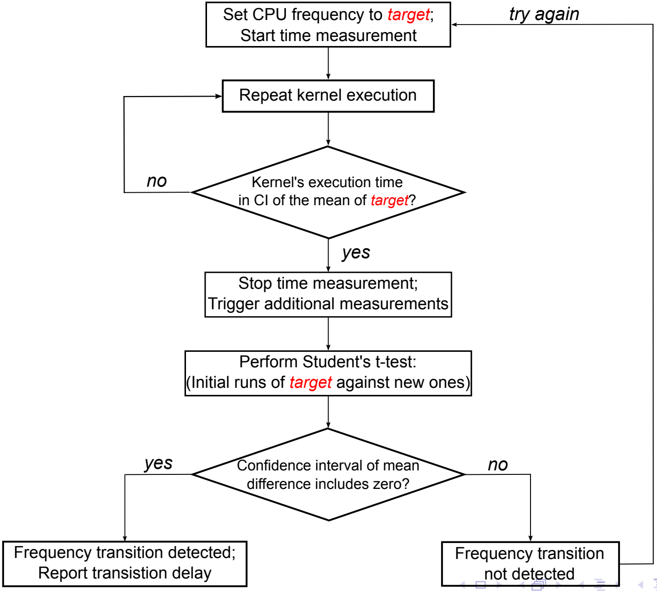
Comparing the performance of two CPU frequencies

- 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



Latency estimation



Experimental setup

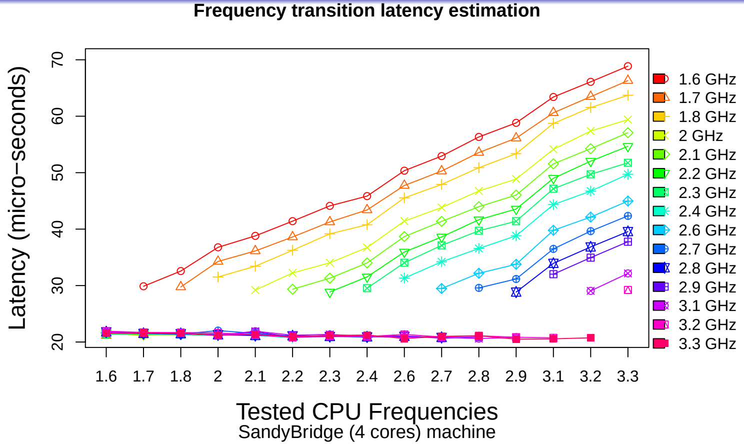
Hardware setup

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

Software setup

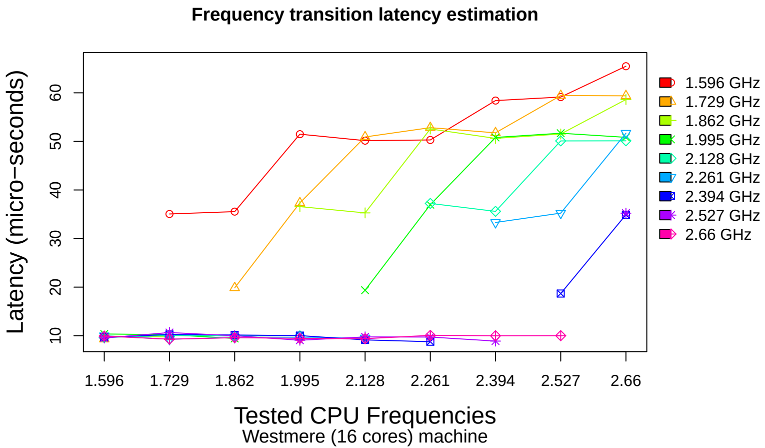
- FTaLaT execution is repeated 31 times for each tested start and target CPU frequency pair
- FTaLaT relies on the TSC (RDTSC instruction) for time measurement:
  - TSC is unaffected by frequency change on our test machines.
- FTaLaT uses the `userspace` Linux governor to select a given CPU frequency.

Experimental results and analysis



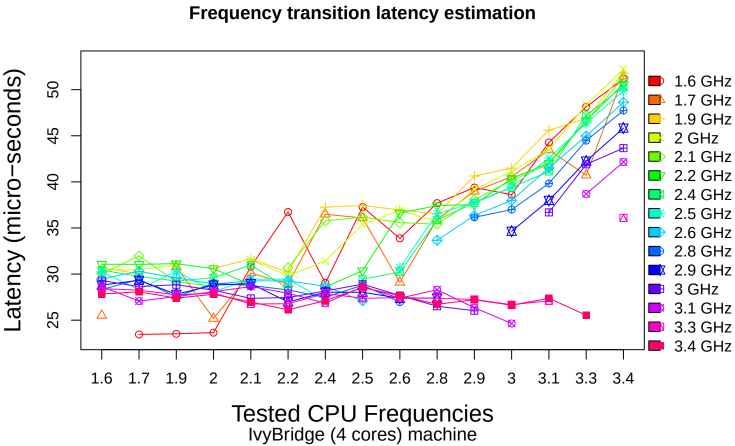
- Transition delay is **not constant** across our test platforms
- Transition latency **increases** when *target* frequency is **higher** than the *start* one
- Voltage and frequency increase performed in **multiple steps**

Experimental results and analysis



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

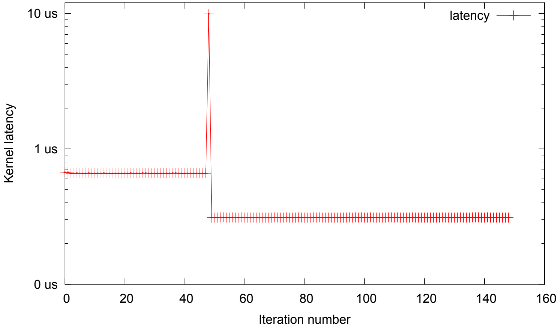
Experimental results and analysis



- 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`
- **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.