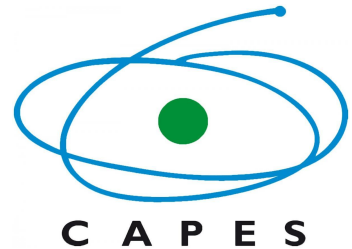
Three-Level Screening Designs for Fast Energy Savings

Lucas Mello Schnorr
schnorr@inf.ufrgs.br



First EnergySFE Intl. Workshop
Grenoble, September 1st, 2016

Introduction & Context

DVFS (Dynamic Voltage and Frequency Scaling) hardware support

- Per-processor (more common) or per-core DVFS
- Very common sense: "OS governor is already here"
- Things might be more complex in HPC $\rightarrow$ Fine-grain DVFS

Introduction & Context

DVFS (Dynamic Voltage and Frequency Scaling) hardware support

- Per-processor (more common) or per-core DVFS
- Very common sense: "OS governor is already here"
- Things might be more complex in HPC $\rightarrow$ Fine-grain DVFS

Fine-grain DVFS for HPC applications?

- Subtle phase changes (microsecs. range) during runtime
  - CPU-bound, Memory-bound
- Frequency scaling in per-phase fashion
  - If phase is memory-bound $\rightarrow$ minimal (or no) performance losses
- What is a phase: a OpenMP parallel loop

Per-Phase Frequency Scaling

There are many strategies. Here's two widely different

Freeh et al. (2005): per-phase frequency scaling

- Test all possible frequencies one by one for each phase
  - Sequentially and in order (f $\times$ p plus replications)

Per-Phase Frequency Scaling

There are many strategies. Here's two widely different

Freeh et al. (2005): per-phase frequency scaling

- Test all possible frequencies one by one for each phase
  - Sequentially and in order (f $\times$ p plus replications)

Laurenzano et al. (2011, 2013): loop-based strategy

- Series of loops (with different CPU and memory behavior)

Per-Phase Frequency Scaling

There are many strategies. Here's two widely different

Freeh et al. (2005): per-phase frequency scaling

- Test all possible frequencies one by one for each phase
  - Sequentially and in order (f $\times$ p plus replications)

Laurenzano et al. (2011, 2013): loop-based strategy

- Series of loops (with different CPU and memory behavior)

1. System characterization
   - Several loop versions are executed with all available frequencies
   - Defines which frequency is the best for each loop configuration
2. Real application loops are profiled
   - Signature (cache hits, flops, memory accesses, etc)
   - Search the closest point in system characterization data

Problem Statement (with example)

Situation

1. Many freq. levels available on current processors architectures
2. Many code region in complex HPC applications

Too much time to discover best frequency for each code region

- Results depend on the input size + other factors

Problem Statement (with example)

Situation

1. Many freq. levels available on current processors architectures
2. Many code region in complex HPC applications

Too much time to discover best frequency for each code region

- Results depend on the input size + other factors

An example: applying Freeh's methodology: f × p × replications

- Graph500 benchmark: 17 code regions (manually marked)
  - Input size 25, it takes roughly six minutes
- Intel Xeon E5-2630: 13 frequency levels (including turbo boost)
- 221 experiments × 10 replications × 6 minutes = 13260 minutes

Problem Statement (with example)

Situation

1. Many freq. levels available on current processors architectures
2. Many code region in complex HPC applications

Too much time to discover best frequency for each code region

- Results depend on the input size + other factors

An example: applying Freeh's methodology: f × p × replications

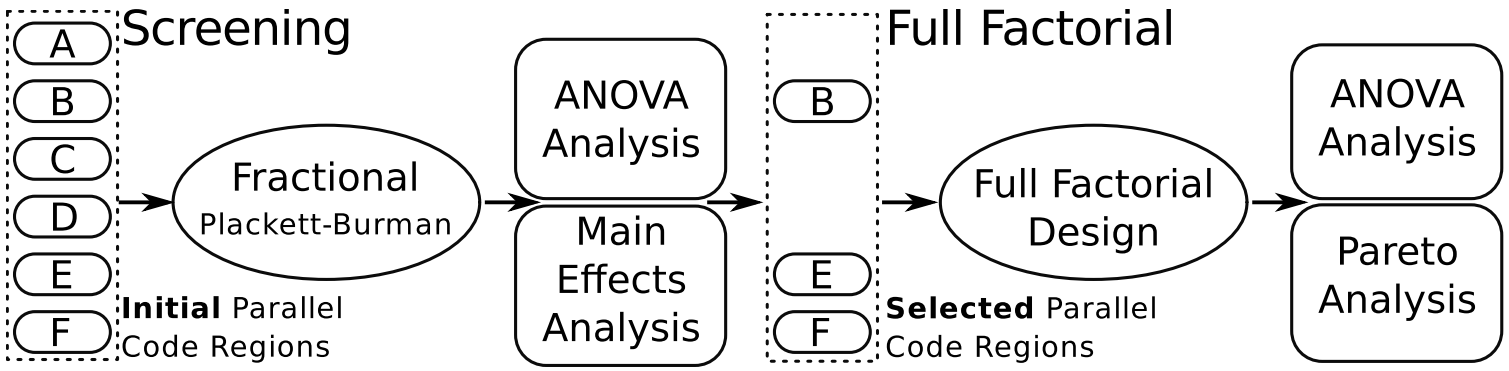
- Graph500 benchmark: 17 code regions (manually marked)
  - Input size 25, it takes roughly six minutes
- Intel Xeon E5-2630: 13 frequency levels (including turbo boost)
- 221 experiments × 10 replications × 6 minutes = 13260 minutes

A faster methodology is required.

Previous Strategy (Millani's Reppar'16 Paper)

DoE-based methodology to accelerate experimental time

- Code regions are letters (A, B, ...) – Factors
- Only two processor frequencies: Low (-) and High (+) – Levels

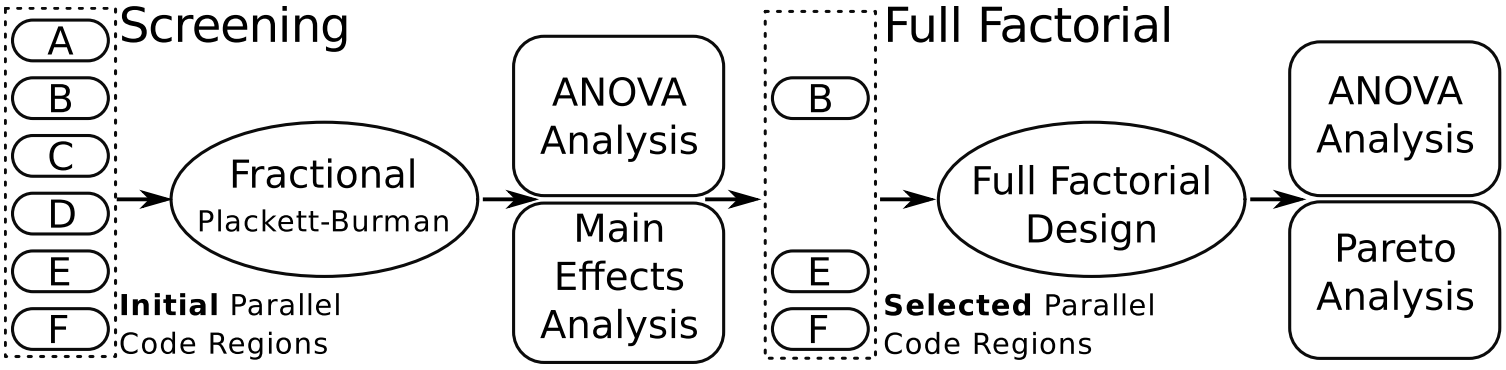


PB Screening: how much each code region affects Time/Energy?
Full Factorial: define all Time-Energy trade-offs, with variability

Previous Strategy (Millani's Reppar'16 Paper)

DoE-based methodology to accelerate experimental time

- Code regions are letters (A, B, ...) – Factors
- Only two processor frequencies: Low (-) and High (+) – Levels



PB Screening: how much each code region affects Time/Energy?
Full Factorial: define all Time-Energy trade-offs, with variability

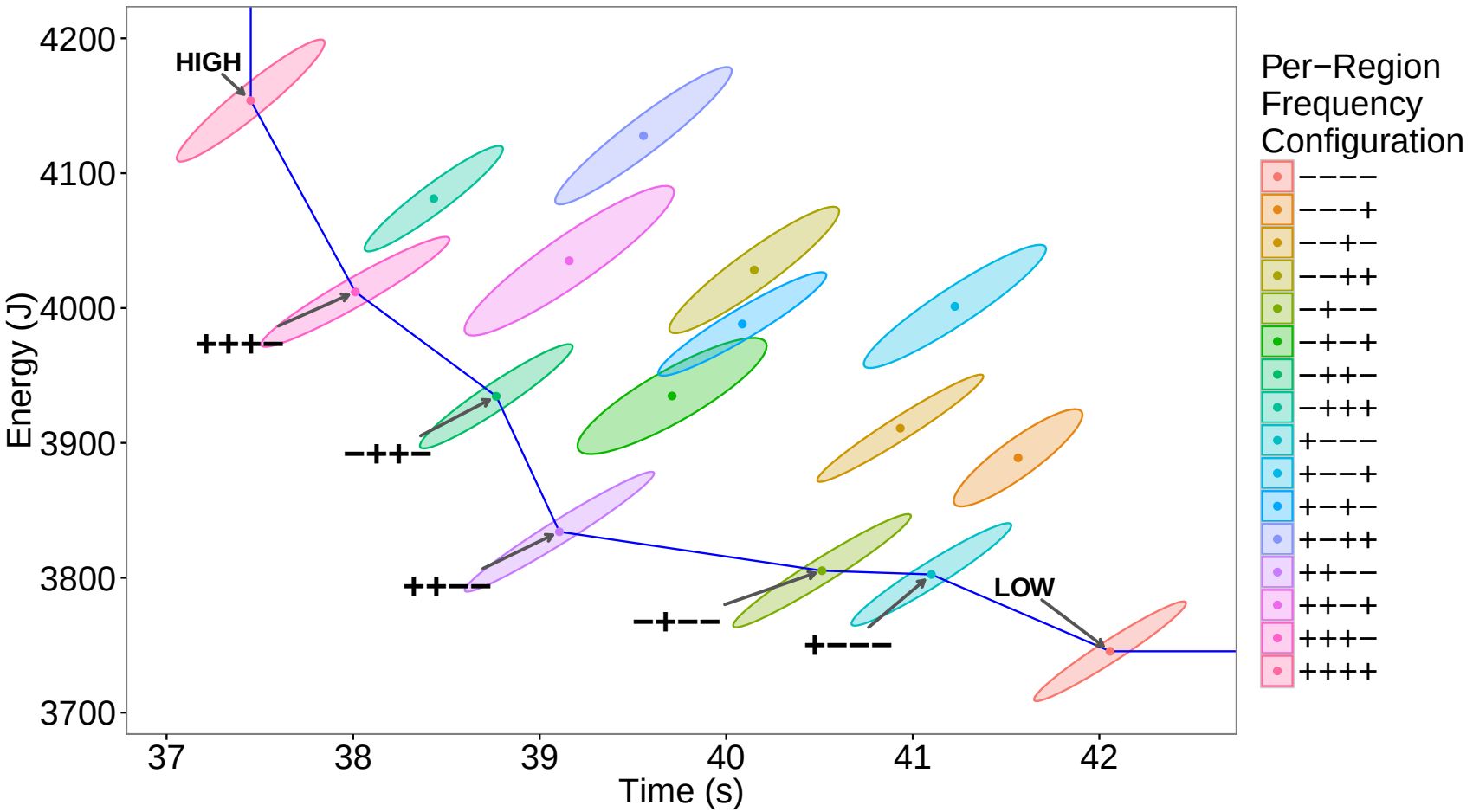
Back to the example: Graph500 17 regions + Intel E5-2630 13 freqs

- 20 $\times$10 replics. + 2^4 $\times$10 replics. = 960$\times$6 = 6960 minutes
  - Disregarding FF, we get only 1200 minutes

Graph500: Full Factorial Results (Pareto)

Ellipses: 99% CI; Pareto points: + and - labels for the E,I,J,L regions

++-- -7.7% energy, +4.4% time against HIGH; +2.4% energy, -7% time against LOW



2 × E5-2630 Sandy Bridge (12 cores); 32GB; GCC5.1.1-O3, RAPL

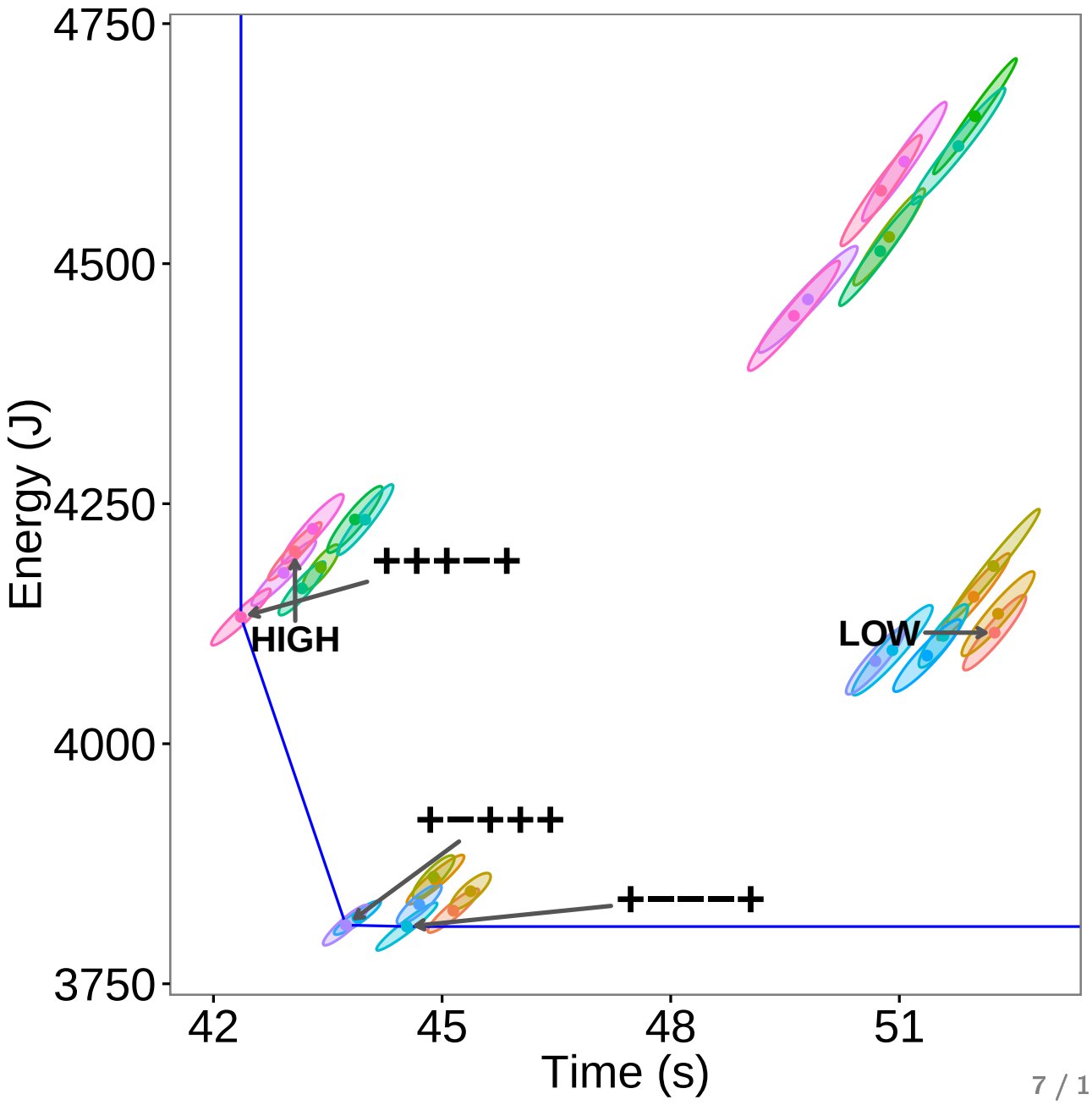
MiniFE: Full Factorial Results (Pareto)

Ellipses 99% CI

\+ and – for the D,F,K,T,Y regions

+−+++

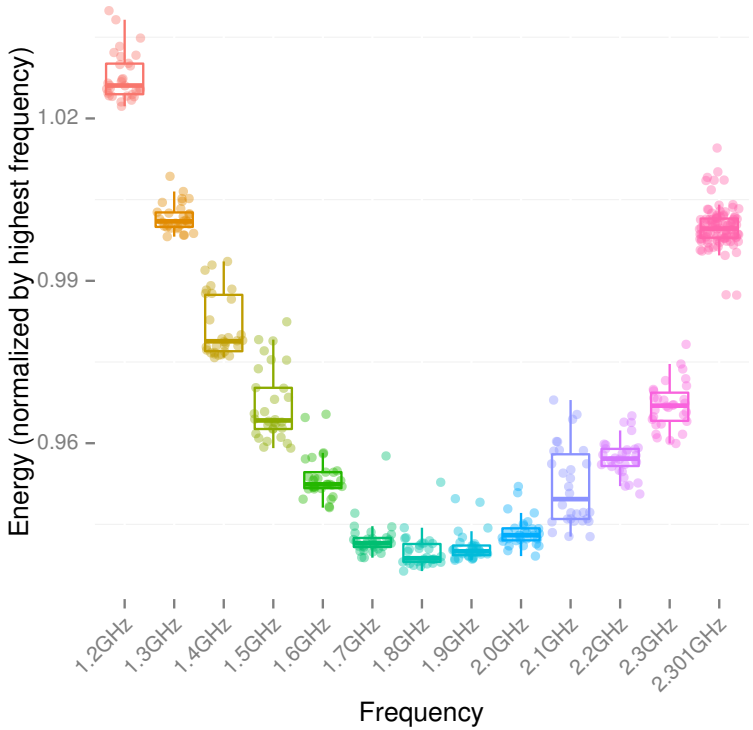
-9.3% energy
+1.5% time
against `HIGH`



Critical View on the Current Approach

Only considers two frequency levels (low and high)

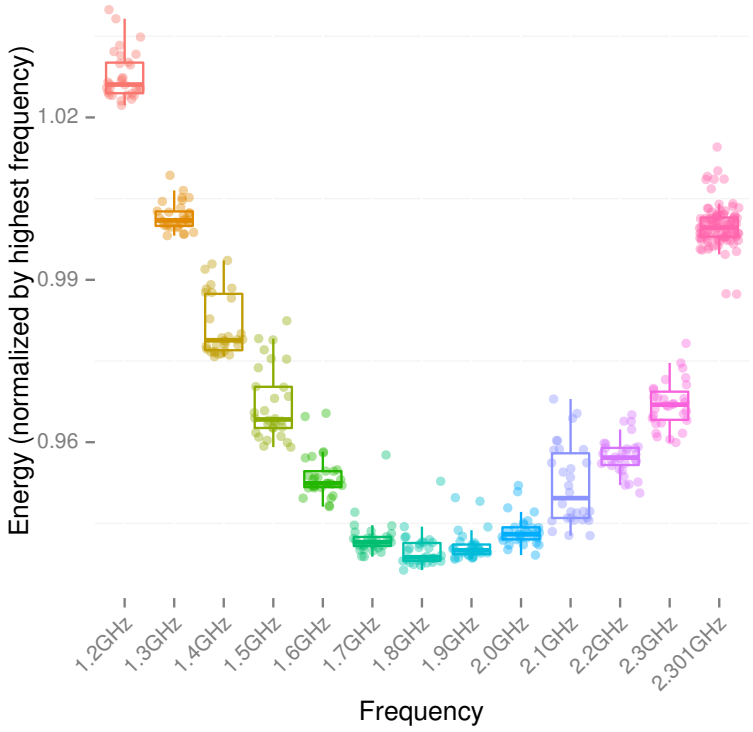
- Showing energy consumption as a function of proc. frequency
- Matrix Multiplication (CPU-bound)



Critical View on the Current Approach

Only considers two frequency levels (low and high)

- Showing energy consumption as a function of proc. frequency
- Matrix Multiplication (CPU-bound)



Is the full factorial step really necessary?

- We have seen no factor iteraction so far

Refining the Strategy + Current Work

Adopt three-level screening designs

- Three processor frequencies; to be more realistic
- Easy to run, hard to analyze

Automatic detection of memory-bound code regions

- Using hardware counters: L2, L3 misses; IPC
  - Use likwid's timeline mode (counters as a function of runtime)
- Gabriel Moro's Master Thesis

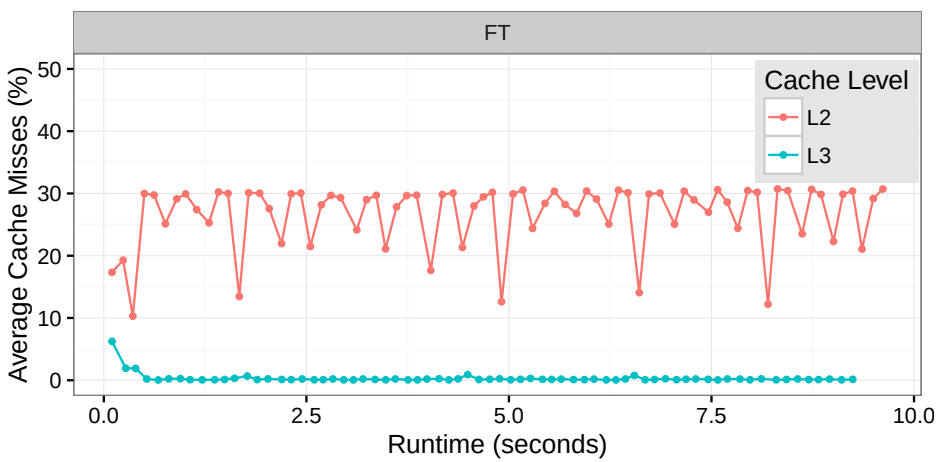
Refining the Strategy + Current Work

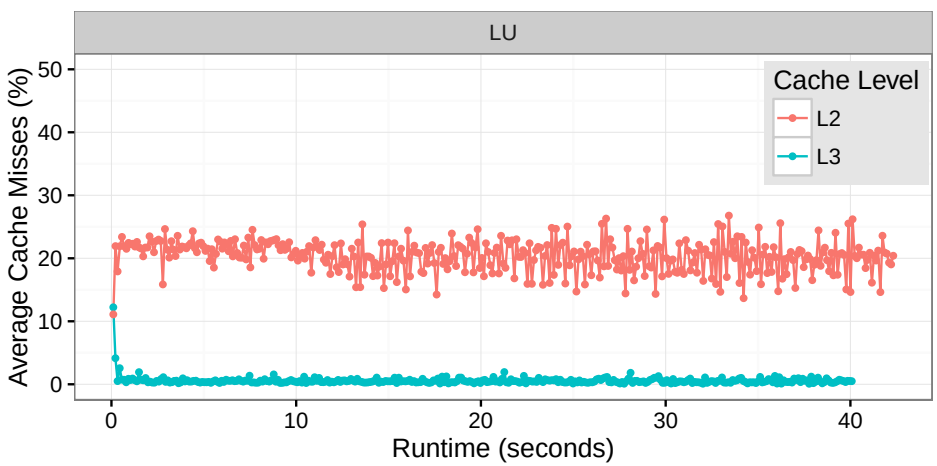
Adopt three-level screening designs

- Three processor frequencies; to be more realistic
- Easy to run, hard to analyze

Automatic detection of memory-bound code regions

- Using hardware counters: L2, L3 misses; IPC
  - Use likwid's timeline mode (counters as a function of runtime)
- Gabriel Moro's Master Thesis





Future Directions + Thanks

Many benchmarks Graph500, MiniFE, Lulesh, Rodinia, Parsec, Mantevo, Coral, Bots

ReDFST https://github.com/lfgmillani/redfst (GPL'd code)

Future Directions + Thanks

Many benchmarks Graph500, MiniFE, Lulesh, Rodinia, Parsec, Mantevo, Coral, Bots
ReDFST https://github.com/lfgmillani/redfst (GPL'd code)

Thanks for your attention. See more details

- http://www.inf.ufrgs.br/~schnorr/