Timing Predictability and How to Achieve It

Jian-Jia Chen

Slides are originally from Prof. Jan Reineke
The Timing Analysis Problem

Set of Software Tasks

Microarchitecture

Timing Requirements

// Perform the convolution.
for (int i=0; i<10; i++) {
    x[i] = a[i]*b[j-i];
    // Notify listeners.
    notify(x[i]);
}
What does the execution time depend on?

- The **input**, determining which path is taken through the program.
What does the execution time depend on?

- The **input**, determining which path is taken through the program.
- The **state of the hardware platform**: Due to caches, pipelining, speculation, etc.
- Interferences from the environment: External interferences as seen from the analyzed task on shared busses, caches, memory.
What does the execution time depend on?

- The **input**, determining which path is taken through the program.
- The **state of the hardware platform**: Due to caches, pipelining, speculation, etc.
- **Interference** from the environment: External interference as seen from the analyzed task on shared busses, caches, memory.
Example of Influence of Corunning Tasks in Multicores

Radojkovic et al. (ACM TACO, 2012) on Intel Atom and Intel Core 2 Quad:

up to 14x slow-down due to interference on shared L2 cache and memory controller
The Need for Models

Predictions about the future behavior of a system are always based on models of the system.

All models are wrong, but some are useful.

George Box (Statistiker)
The Need for Timing Models

The Instruction Set Architecture (ISA) partially defines the behavior of microarchitectures: it abstracts from timing.

How to obtain timing models?
- Hardware manuals
- Manually devised microbenchmarks
- Machine learning

**Challenge:** Introduce HW/SW contract to capture timing behavior of microarchitectures.
Desirable Properties of Systems and their Timing Models

- Predictability
- Analyzability
Predictability

How precisely can programs' execution times on a particular microarchitecture be predicted?

Assuming a deterministic timing model and known initial conditions, can perfectly predict execution time.

But: initial state, inputs, and interference unknown.
How to Increase Predictability?

1. *Eliminate stateful components:*
   - Cache → Scratchpad memory
   - Regular Pipeline → Thread-interleaved pipeline
   - Out-of-order execution → VLIW

**Challenge:** Efficient static allocation of resources.
How to Increase Predictability?

2. **Eliminate interference:** „temporal isolation“

Partition resources:

- **in time**
  - TDMA bus/NoC arbitration,
  - SW scheduling (e.g. PREM)

- **in space**
  - shared cache: in HW or SW
  - SRAM banks (e.g. Kalray MPPA)
  - DRAM banks (e.g. PRET DRAM, PALLOC)

**Challenge:**
*Determine efficient partitioning of resources.*

**Question:** What’s the performance impact?
Analyzeability

How *efficiently* can programs‘ WCETs on a particular microarchitecture be bounded?

WCET analysis needs to consider all inputs, initial HW states, interference scenarios...

...explicitly or implicitly.
How to Increase Analyzability?

1. Eliminate stateful resources: Fewer states to consider
2. Eliminate interference: „temporal isolation“: Can focus analysis on one partition
3. Choose „forgetful“/„insensitive“ components: Different analysis states will quickly converge
Timing Anomalies

Cache Miss = Local Worst Case

Cache Hit

Nondeterminism due to uncertainty about hardware state

leads to

Global Worst Case

Timing Anomalies in Dynamically Scheduled Microprocessors
T. Lundqvist, P. Stenström – RTSS 1999
Timing Anomalies: Example

Scheduling Anomaly

Bounds on multiprocessing timing anomalies
(http://epubs.siam.org/doi/abs/10.1137/0117039)
Timing Compositionality: By Example

**Timing Compositionality** =
Ability to simply sum up timing contributions by different components

Implicitly or explicitly assumed by (almost) all approaches to timing analysis for multi cores and cache-related preemption delays (CRPD).
Timing Compositionality: Benefit

How does compositionality help?

Efficiency of microarchitectural analysis

Integrated Compositional

Uncertainty does not multiply
Conventional Wisdom

Simple in-order pipeline + LRU caches

→ no timing anomalies
→ timing-compositional
Bad News I: Timing Anomalies

We show such a pipeline has timing anomalies:

*Toward Compact Abstractions for Processor Pipelines*

A Timing Anomaly

load ...  
nop  
load r1, ...  
div ...; r1  
---------------  
ret

Hit case:
• Instruction fetch starts before second load becomes ready  
• Stalls second load, which misses the cache

Miss case:
• Second load can catch up during first load missing the cache  
• Second load is prioritized over instruction fetch  
• Loading before fetching suits subsequent execution
Bad News II: Timing Compositionality

Maximal cost of an additional cache miss?

**Intuitively:** main memory latency

**Unfortunately:** ~ 2 times main memory latency
- ongoing instruction fetch may block load
- ongoing load may block instruction fetch
Some References

Enabling Compositionality for Multicore Timing Analysis

MIRROR: Symmetric Timing Analysis for Real-Time Tasks on Multicore Platforms with Shared Resources

A Generic and Compositional Framework for Multicore Response Time Analysis

On the Smoothness of Paging Algorithms

Toward Compact Abstractions for Processor Pipelines

Architecture-Parametric Timing Analysis

Selfish-LRU: Preemption-Aware Caching for Predictability and Performance

Towards Compositionality in Execution Time Analysis - Definition and Challenges

Measurement-based Modeling of the Cache Replacement Policy

PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation

Timing Predictability of Cache Replacement Policies