Temporal Decoupling – Are “Fast” and “Correct” Mutually Exclusive?

Dr. Jakob Engblom, Intel Stockholm, Sweden

jakob.engblom@intel.com
TEMPORAL DECOUPLING
Temporal Decoupling
Necessary Technique

**Less overhead**
- Less time spent in simulator kernel
- More time spent in models
- Fewer trips through sim kernel code
- A fast Virtual Platform (VP) has to get to much less than 10 host instructions per target instruction

**More locality**
- Data
- Code
- Better effect from Just-in-Time (JIT) compilers and Virtualization Technology (VT) acceleration
Experiments on Wind River Simics®

• All experiments performed using Simics

• Instruction Set Simulator (ISS) modes:
  – Interpreter
  – Just-in-time (JIT) compiler
    • Like all other fast simulators in the world
  – Direct execution (VMP)
    • Uses Intel® Virtualization Technology for Intel® 64 and IA-32 architectures (Intel® VT-x) to run IA code directly on the host
FAST?
Example: Boot

Relative Performance vs Time Slice Length – Ubuntu* 16 Linux Boot

Notes:
• Target system: 4 x IA Core i7-9xx, 2000 MHz, 16 GB RAM
• Booting Ubuntu* Linux 16.04
• Virtual boot time to graphical desktop is 24s, out of which 12s are low load

*Other names and brands may be claimed as the property of others.
Example: Compute Program

**Relative Performance vs Time Slice Length - User-Level Compute Program**

**Notes:**
- Target system: 4xIA Core i7-9xx, 2000 MHz, 16 GB RAM
- Running Ubuntu* 16.04, and on top of that a threaded compute program that uses all target cores at basically 100%
- Runs for 19s of target time

*Other names and brands may be claimed as the property of others.
Compute-Intense program on an 8-core NXP QorIQ P4080*

Note:
- Only JIT, as this is cross-target
- Graph starts at 10, not 1
- Peak at 100k instructions

*Other names and brands may be claimed as the property of others.
Observations

• VMP benefits from longer time slices – up to 1M instructions
  – Still less than 1/1000 of a second for a typical 2-4GHz processor core

• JIT plateaus around 10k instructions
  – Standard observation going back to mid-2000s

• Good default is 100k instructions, might increase from there
CORRECT?
Temporal Decoupling = Information Latency

“Variant 1” in the paper, with devices running in their own time quanta
Temporal Decoupling = Information Latency

“Variant 2” in the paper, with devices running inside the time quanta of the processors accessing them
The Classic Bathtub

Freescale MPC8572* Linux boot, effect of time quantum length

- Board
  - SoC
  - SW
  - OS
  - Core

- Host execution time (s)
- Total instructions executed

- Time quantum length (cycles)

*Other names and brands may be claimed as the property of others.*
Race Conditions - Micro

- **Microbenchmark**
  - Each thread loops, load-modify-write as quick as possible on a shared variable
- **Longer time quantum** =
  - Fewer races seen
  - But: Races still happen
- **Not really representative of real software**
Race Conditions - Macro

- Varying time quantum length proven way to find errors
  - Order of events
  - Interleave of software operations
- Scriptable & deterministic
  - Example: http://blogs.windriver.com/tools/2012/12/debugging-simics-on-simics.html
  - Some 30 tests to replicate error

*Other names and brands may be claimed as the property of others.
Ping-Pong Protocol, no Time-Out

Virtual time (s) vs Network latency (µs)

- VP is running at 1x real-time speed
- VP is running at 60x real-time speed

Tftp total transfer time vs network latency

- host time (s)
- virtual time (s)
Ping-Pong Protocols with Time-Out

On hardware, this is expected path

If the time slice is longer than the time-out
Ping-Pong Protocols with Time-Out (2)

• Reduce the time quantum...

• ... or insert stalls to avoid reducing the time quantum

Make the ping operation take a long time so that processor B sees it and replies before the check for the reply & the time-out starts
Example: Fairness Test

4 x threads: Loop:
Take lock
Inc counter
Release lock
Wait N

Thread unbalance, for time slice 200k cycles

With a long time quantum and short wait, we can see that one thread dominates the lock

Thread unbalance, for time slice 100 cycles

With a short time quantum, there is much less unfairness – especially once wait is longer than time quantum
Most of the Time, Speed is Key

We need 100s of billions of instructions to be run for most real setups

Server model

User land

Database program

Linux* Distro

Server UEFI

96GB RAM

Core

Core

Processor socket 1

PCH

Disk

10G Eth

96GB RAM

Core

Core

Processor socket 2

User land

SpecJEnterprise* driver utility

Application server (payload)

Java* Virtual Machine (JVM)

Linux* Distro

Server UEFI

96GB RAM

Core

Core

Processor socket 1

PCH

Disk

10G Eth

96GB RAM

Core

Core

Processor socket 2

*Other names and brands may be claimed as the property of others.
When do we Need to Dial Down?

• Results (timing) skew when accessing shared resources:
  – *Time quantum < shortest relevant observable delay*
  – Example: Cache studies: time slice lower than last-level cache penalty is "OK"

• Software time-out:
  – Unit reports time-out unless a reply is seen within a short time
  – ... unless we can solve it some other way (stall)

• Software expecting close timing between closely-coupled cores
Questions?