Simics® Accelerator: Creating a Parallel Program out of a Serial Problem

Jakob Engblom, PhD
Technical Marketing Manager, Virtutech
jakob@virtutech.com

virtutech

Overview

- We have what looks like a *stubbornly sequential* problem that we need to make go faster
- Rethink and slightly redefine the problem domain
- Out comes a *decently concurrent* solution

Stubbornly Sequential	Embarrasingly Parallel (Conveniently Concurrent)	Decently Concurrent
• Programs and algorithms with hard-to-break sequential behavior • The best solution appears to be a sequential algorithm • Ex: Shortest path	• Obvious and trivial parallelism • Many independent computations • "Embarrassing if you cannot make a scalable parallel solution" • Ex: Signal processing on multiple streams, media codecs, web servers, web search, routing, packet scan	• Problem domain offers parallelism • But not as parallel as the convenient problems • Each thread can be quite different from other threads • Ex: Multiple independent programs, game engines (AI, physics, graphics, …)

http://amdahlslaw.blogspot.com/2008/07/take-1000-out-of-my-pocket-for-thinking.html,
http://www.chipdesignmag.com/martins/2008/08/04/inspired-by-amdahls-competition/

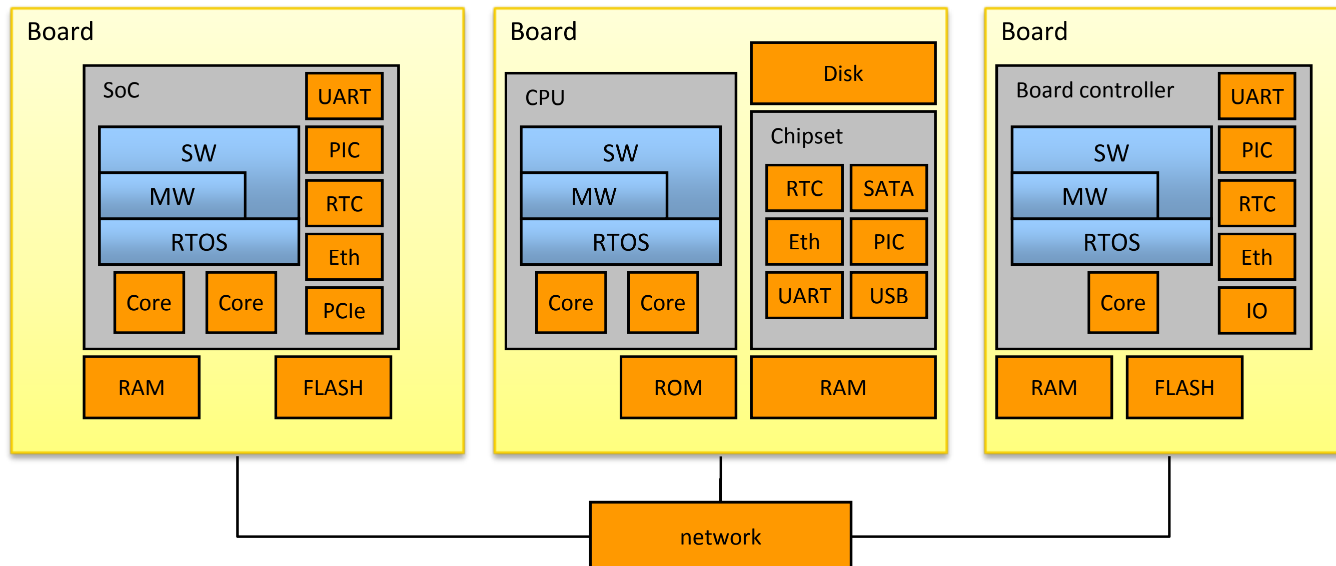
virtutech

The Problem Domain

- Simulating a computer system, any computer system
- Run the same binary code as the target system
- Multiple processors, multiple machines
- Execution speed the primary goal

The Problem Domain

- Defined, repeatable, and deterministic semantics
  - The simulator provides more control than physical hardware
- Target hardware system is inherently parallel
- Naïve semantics:
  - Clearly define the order of execution of units
  - Interleave all simulation units on a cycle-by-cycle basis



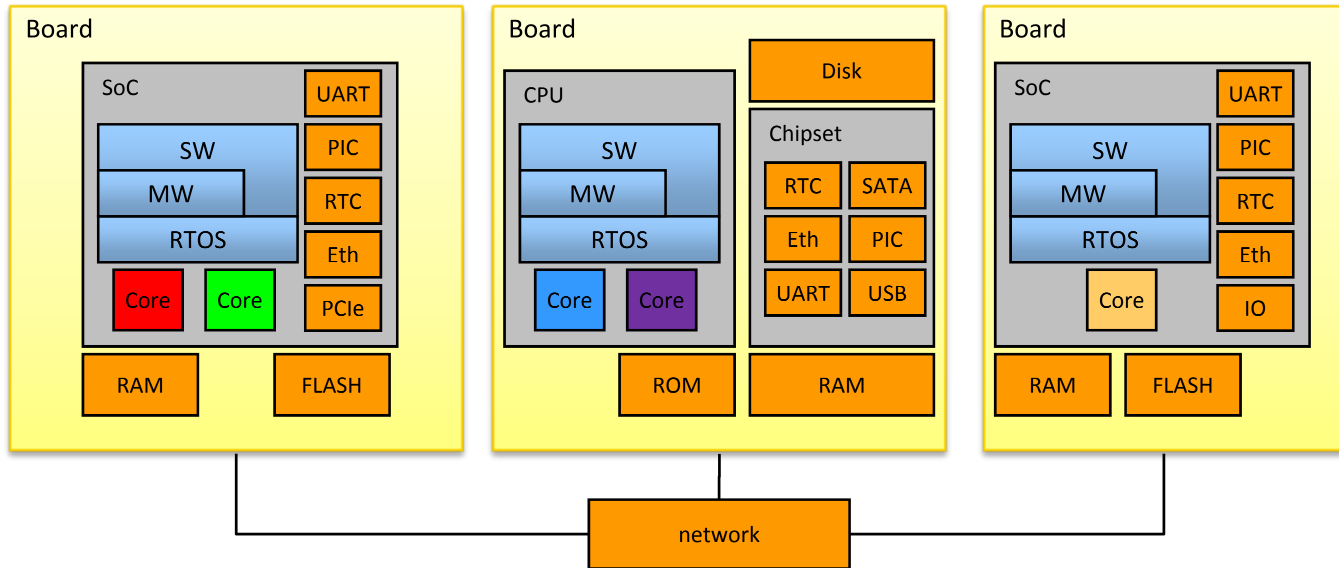
virtutech

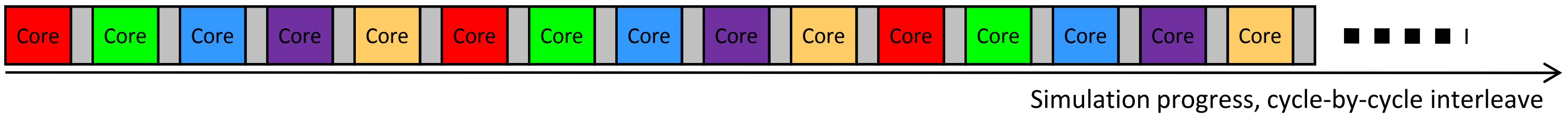
Rethink of Semantics Part 1

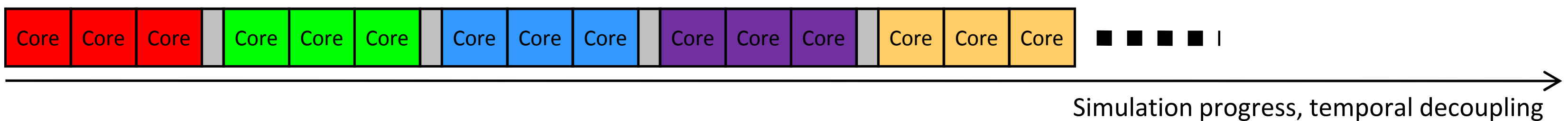
- Observations:
  - Focus is on processors executing code
  - Processors only rarely observes what other processors do
  - Software is tolerant to timing variations
  - Processors consume > 95% or simulation time
  - Device models driven by processors
- Optimization:
  - Make device models passive transaction-driven objects
  - Device models complete their work in a single step
  - **Temporal Decoupling**: run each processor for a long time in each step, do not switch on every cycle
- Result:
  - Enhanced locality, greatly increased performance
  - Works well with almost all software loads
  - “Classic fast Simics”

virtutech

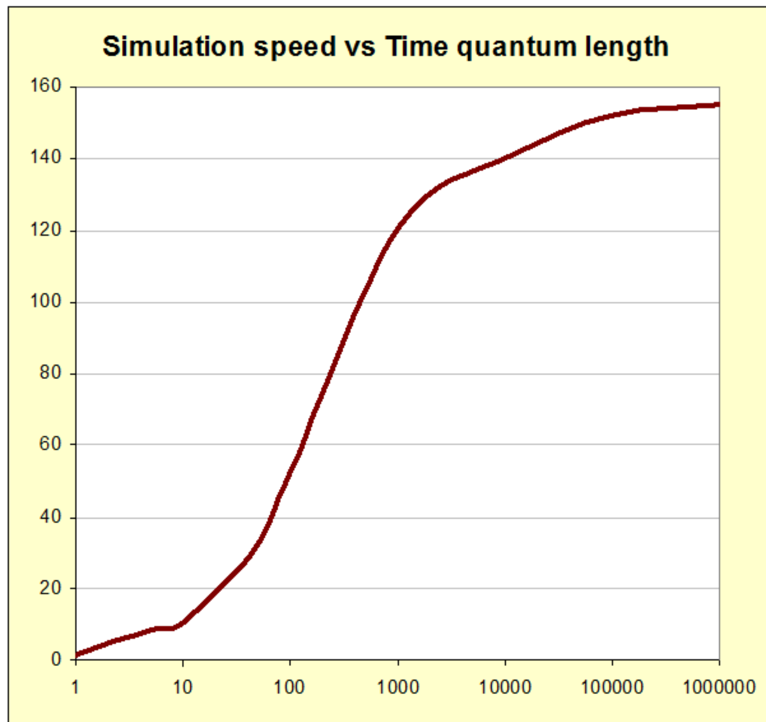
Temporal Decoupling







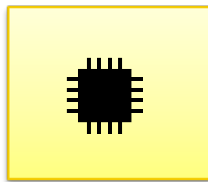
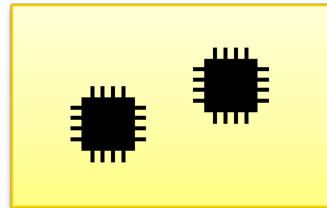
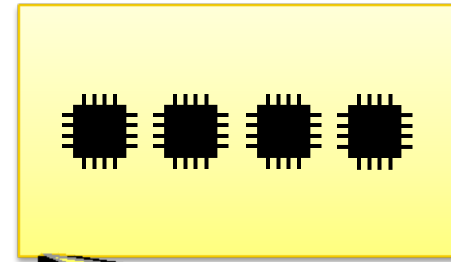
Temporal Decoupling Speed Impact



- Experimental data
  - 4 virtual PPC440 boards
  - Booting Linux
    - Which is a particularly hard workload, lots of device accesses
  - Execution quanta of 1, 10, 100, ... 1000_000 cycles
- Notable points:
  - 10x performance increase from 10 to 1000 quantum
  - +30% from 1000 to 1000_000 quantum

...But Still Sequential

- Deterministic and defined semantics paramount
  - Same execution order and parallel interleave each time

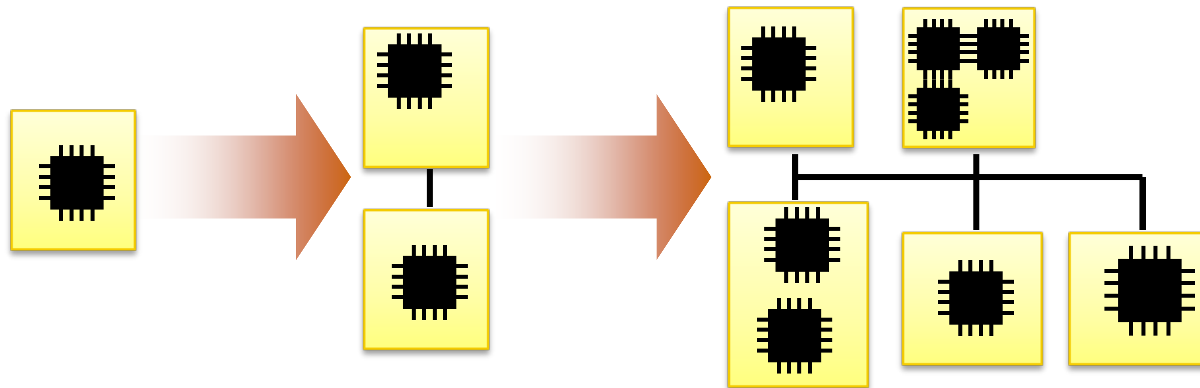
We considered the main problem to be speeding up simulation of tightly-coupled shared-memory or shared-devices systems

- Analysis: synchronization so frequent it would negate any benefit from a parallel execution
  - 1000 to 10000 instructions for processors sharing memory
  - Would mean a synchronization frequency of 100’s of KHz in host time

virtutech

The Target Domain Changes!

- The target systems that we encountered changed
  - From single boards with single processors
  - To multiple boards, each with one or more processors



- This changes the game
  - Significant demands for simulation performance
  - Each board physically quite separate from other boards
  - Boards quite loosely coupled, compared to the coupling between hardware on the same board

virtutech

A Fully Parallel Simulation Solution

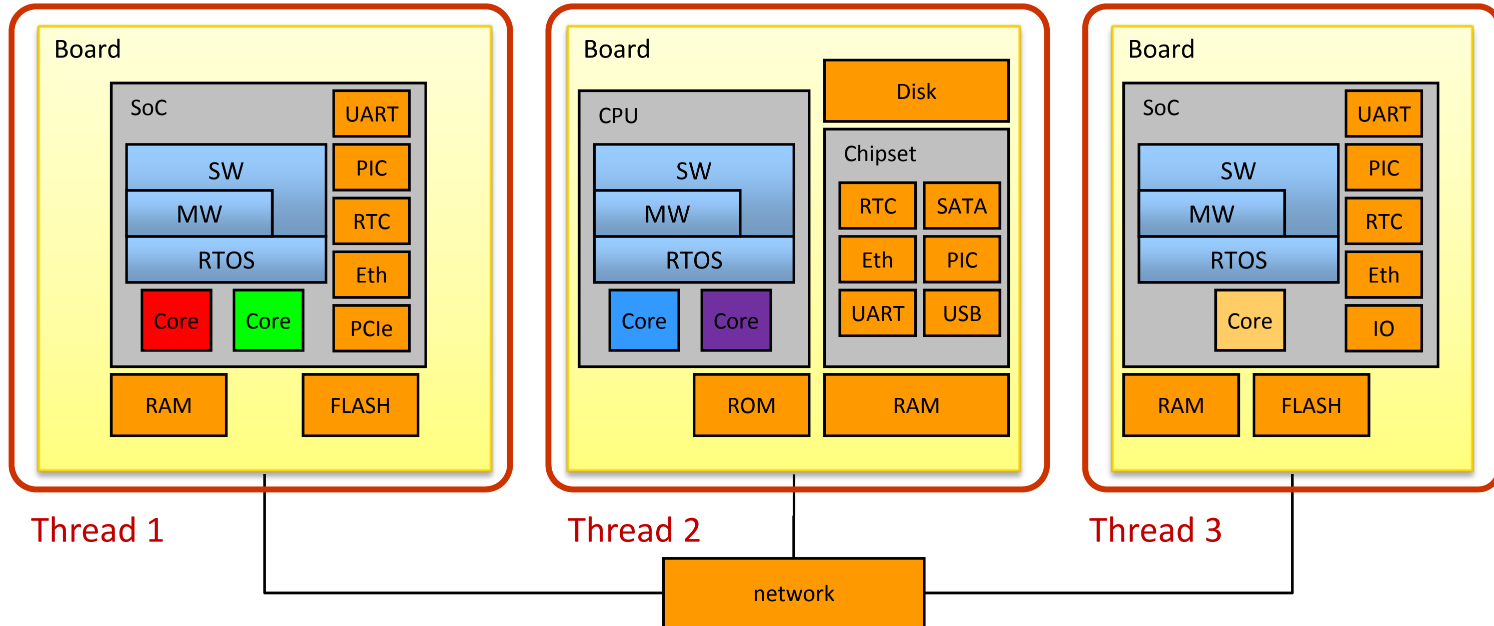
- Put each board or machine on its own thread
  - Use network links as the natural point to divide the simulation
  - Take advantage of high latency of network links
    - This has previously been done when using distributed simulation with Simics, which paved the way for parallel
- Use temporal decoupling within each thread
  - Each board can contain multiple processors
  - Shorter synchronization times than between boards
  - Introduces a hierarchy of synchronization
- Requirements:
  - Simulation results independent of host, # of host cores
  - Run on a single processor defines the semantics

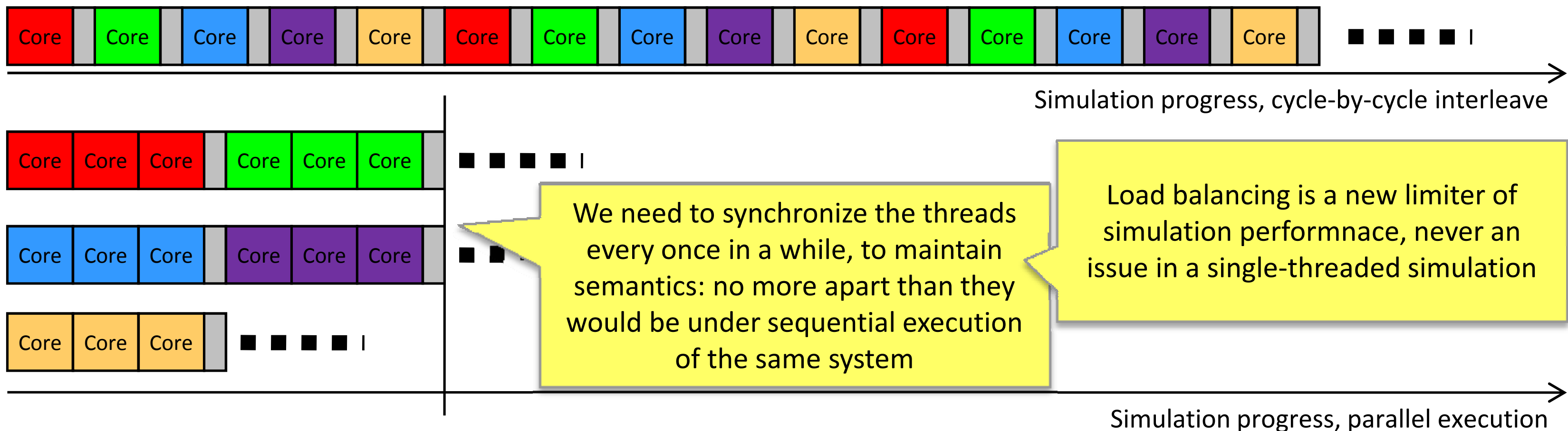
virtutech

Semantic Requirements

	VmWare	IBM CECSim	Serial Simics	Parallel Simics
Multiple machines on single host processor	Yes	No	Yes	Yes
Multiple machines on SMP host	Yes	No	No	Yes
SMP on SMP host	Yes	Yes	No	No
Host-independent execution semantics	No	No	Yes	Yes
Repeatable execution	No	No	Yes	Yes
Reverse execution	No	No	Yes	Yes
Checkpointing	Yes	No	Yes	Yes
Target architecture	X86	zSeries	Power Archictecure, x86, MIPS, SPARC, ARM, C64, H8, MSP430, Alpha, IA64, ...	

Parallel Execution





virtutech

Impact on Simulation Models

- Simics infrastructure
  - Several helper threads for a long time, thread safety was already present in many places
  - Multiple active simulation threads required some changes to how simulation is controlled and inspected
  - Scripting environment obviously affected
- Processors & Memories
  - Provided by VT
  - Thread-safe checks in the code
  - Semantics not really affected, same as temporal decoupling
- Network links
  - Only a few different types, mostly done by VT
  - Passes data between simulation threads
  - *Only* simulation unit straddling parallel execution units
  - Have to be written with awareness of simulation semantics
  - Existing distributed simulation support formed the basis

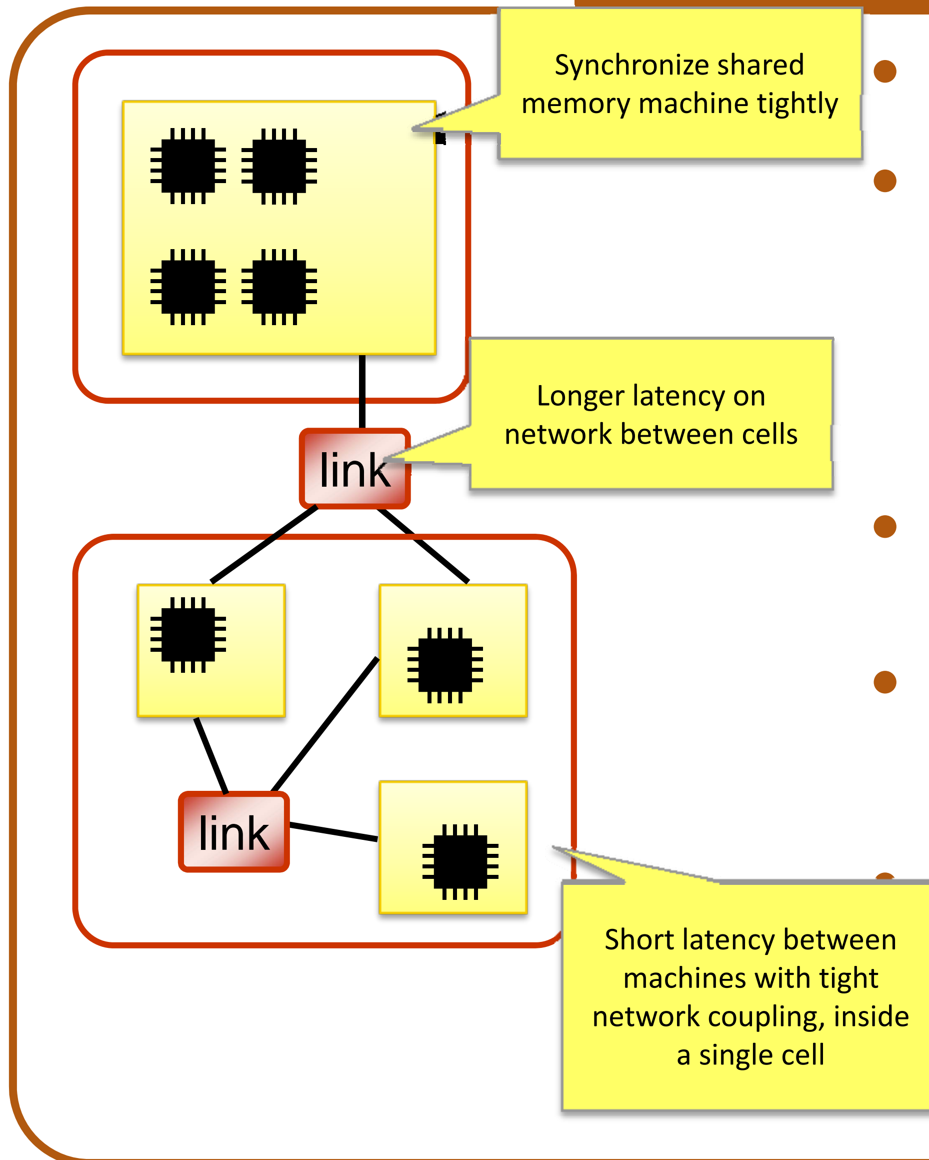
virtutech

Impact on Simulation Models

- Device models
  - The bulk of Simics models, written by VT, users, 3rd parties
  - Semantics of simulation same as temporal decoupling
  - Each device local to one thread, sees a sequential world
  - Any device shared between several processors requires the simulation to run all these processors in the same thread
  - Main work: ensure that code is thread-safe
    - Run in true concurrency with other Simics code
    - Run with other instances of the same model
    - No data shared between instances
    - Has to be validated model-by-model
- The fact that Simics already used temporal decoupling helped parallelization tremendously
  - Parallel execution a "minor tweak" to temporal decoupling
  - Many models already thread-safe without any changes
  - Changes to models usually small

virtutech

Hierarchical Synchronization

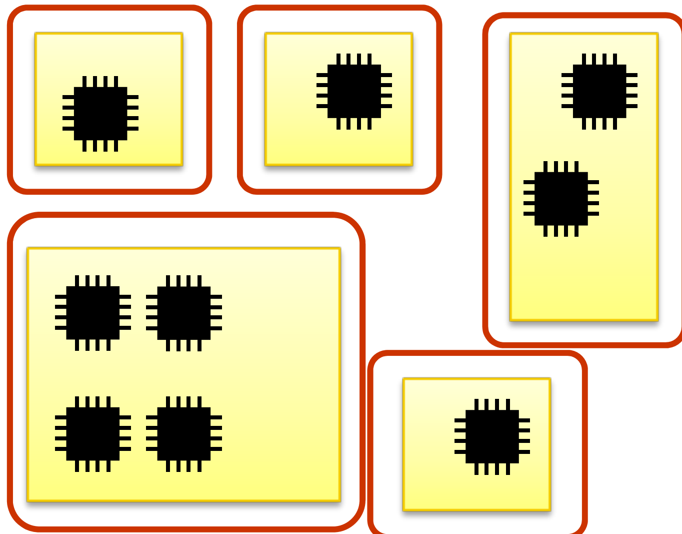


- Deterministic semantics
  - Regardless of host # cores
- Periodic synchronization between different cells and target machines
  - Puts a minimum latency on communication propagation
  - Synch interval determines simulation results, not number of execution threads in Simics
- Latency within a cell:
  - 1000-10000 cycles
  - Works well for SMP OS
- Latency between cells:
  - 10 to 1000 ms
  - Works well for latency-tolerant networks
- Builds on current Simics experience in temporally decoupled simulation
  - Tried-and-tested, only executing faster on a multicore host

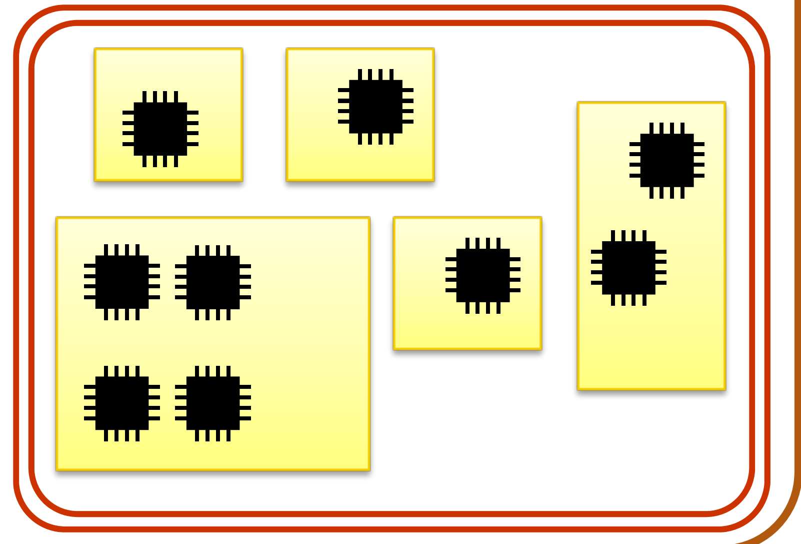
virtutech

Achievable Parallelism

- Maximal model
  - One thread per board
  - Threads will be quite unbalanced, as boards have different load
  - OS schedules threads on all processors on host
  - Quick initial implementation to get off the ground



- Controlled model
  - Only a few threads, with several boards each
  - Simics balances the boards onto the threads
  - Worker pool algorithm
    - Strictly more powerful
  - Share a multicore host between Simicses



virtutech

Multithreading Performance Results

- Performance of Simics multithreading depends on
  - Target system characteristics
  - Software latency requirements
  - Target system load balance
  - Target system communications patterns

- Synthetic experiments and lab experience
  - Single-thread performance not affected
    - Simics works just as well as before on a single core
    - No impact on idle loop simulation
  - Up to 10x Simics 3.2 performance
    - Threading revealed bottlenecks in Simics
    - 8-core host, 64 target machines, no communication
  - Up to 6x scaling on 8-core host
    - Pretty respectable

virtutech

Summary: Lessons

Rethink domain

- What looks stubbornly sequential might be parallel with minor semantic tweaks
- Know your use cases!

Device-local semantics

- Simplify user-provided code
- Essentially sequential
- ”Share nothing” = no locking
- Parallelism hidden in the framework

Host-independent semantics

- Result of execution independent on degree of parallelism
- Very important for Simics
- Generally useful for debug

Modular construction

- Roll out parallelism over time
- Immediate benefits
- Test piece-by-piece
- A non-thread-safe module only affects simulations when used

Questions?