“Simultaneous” Considered Harmful: Modular Parallelism

David P. Reed/SAP Research
7 June 2012

Time is what keeps everything from happening at once...
And space is what keeps everything from happening to me.

- John Wheeler (following Ray Cummings)

Overview

Why must parallel computing be difficult? The world is embarrassingly parallel!

A change in perspective is worth 80 IQ points. (*Alan Kay)*

Calls to action

Principles

Escaping our tangled roots (examples)

Traps	**Better ideas**
Careless primitive design	Deprecate/replace them
Bad modularity	Hide internal effects
Virtualization by parts	Virtualization of wholes

Calls to Action

Accept into your life: “Parallel” is the norm, not radical exception

“Simultaneous-action-at-a-distance” is a bad habit:

Eliminate constructs whose operational semantics feed the habit

Question serializability for defining correctness

“Good modularity principles” should never discuss “simultaneous”

Fix programmer thinking: teach parallel programming *first*.

Reject Amdahl’s Law. It dominates only

because *programs* are conceived as sequential,

not because the *problems* are sequential.

Issues

All important computing systems will evolve, scale up, outlive embodiments

Unfortunate belief from HPC culture: parallel means “tune to bare silicon”

Parallel execution disrupts “clean” modular system designs, due to

Naïve “time” concept: total ordering

Virtualization based on sequential (and hard to reverse) resource binding

Coroutine-based processor multiplexing (time-sharing)

Module composition conceived as order of execution

Write-ordering to memory

Caller waits for callee

Concurrency control needed because concurrent modules interfere with each other

Correctness defined by “serializability” – superscalar processor, DBMS

Compilation and interpretation (hardware, OS, compiler, interpreter, DBMS) resource binding overconstrains execution flexibility, complicating design

Simultaneous should be unspeakable

Lamport (*Time, Clocks, and the Ordering of Events in a Distributed System*) argued: causal ordering is *sufficient* to specify correct behavior and described a system of clocks that defines a total causal ordering

But Einstein (and Goedel) argued for the universe being based on a causal partial ordering – different observers in the system may observe different orderings

A consistent total ordering of all events is not needed to define correctness of any physical system (computing systems are a subset of physical systems).

Total orderings are costly to achieve because they imply simultaneous operations at any distance.

Far too many computing primitives imply simultaneous action across an unbounded system

Semantically problematic primitives

Hardware, OS and language primitives implying *simultaneous distant action*

Semaphores (Dijkstra – based on co-routines in THE O/S)

POSIX shared memory mprotect system call

POSIX open() system call binding a name to a file

Compare-and-swap instruction

Gratuituous implicit interactions (false sharing)

Clean primitives that imply only causal partial ordering

Fork() and join()

Eventcounts and sequencers

Producer-consumer LIFO and FIFO buffers

Multicast send/receive

Write-once, read-many memory cells

Good modularity in a parallel world

Parnas – Information Hiding Principle (hide all information about how a module works "inside" the module)

Cleaning Atomic Action

System R locking paper: Atomic x = "for all y, either x precedes y or y precedes x"

Serializable = "as-if sequential"

Cleaner: nothing about how x is executed can be observed by y, no matter when y is carried out. (no hidden causal ordering)

Simplifies parallel composition of modules when there is no causal connection

Note: does not prevent transparent "caching" for speedup within modules.

Idempotency – without ordering – helps build fault tolerance

X || X || X == X *can be implemented without simultaneous distant action*

Virtualization with parallelism in mind

Virtualization – *reversible* binding of an abstract computing platform to resources of an underlying computing platform of similar or different capability. e.g. virtual memory, virtual processor, virtual machine, virtual network, …

Virtualization (and "emulation") usually enhance flexibility, scalability, fault tolerance, availability, security, etc.

Not true when parallelism involved… why?

Observation: Ordering constraints on resource bindings of virtualization interfere with primitives and modules that imply "simultaneous action at a distance".

This is because virtualization of parts is not virtualization of the whole.

A parallel machine where "simultaneous" is an unspeakable is easy to virtualize.

Implications

Clarify modularity as isolation of internals and separation of concerns, not distinctions of time

Scrub “simultaneous” thinking from our vocabulary and our computing concepts.

Isolate “legacy” systems inside containers where only inputs and outputs are exposed. They don’t scale today, and embed too many problematic concepts.

Start all systems designs as parallel - deprecate sequential execution as a rare special case

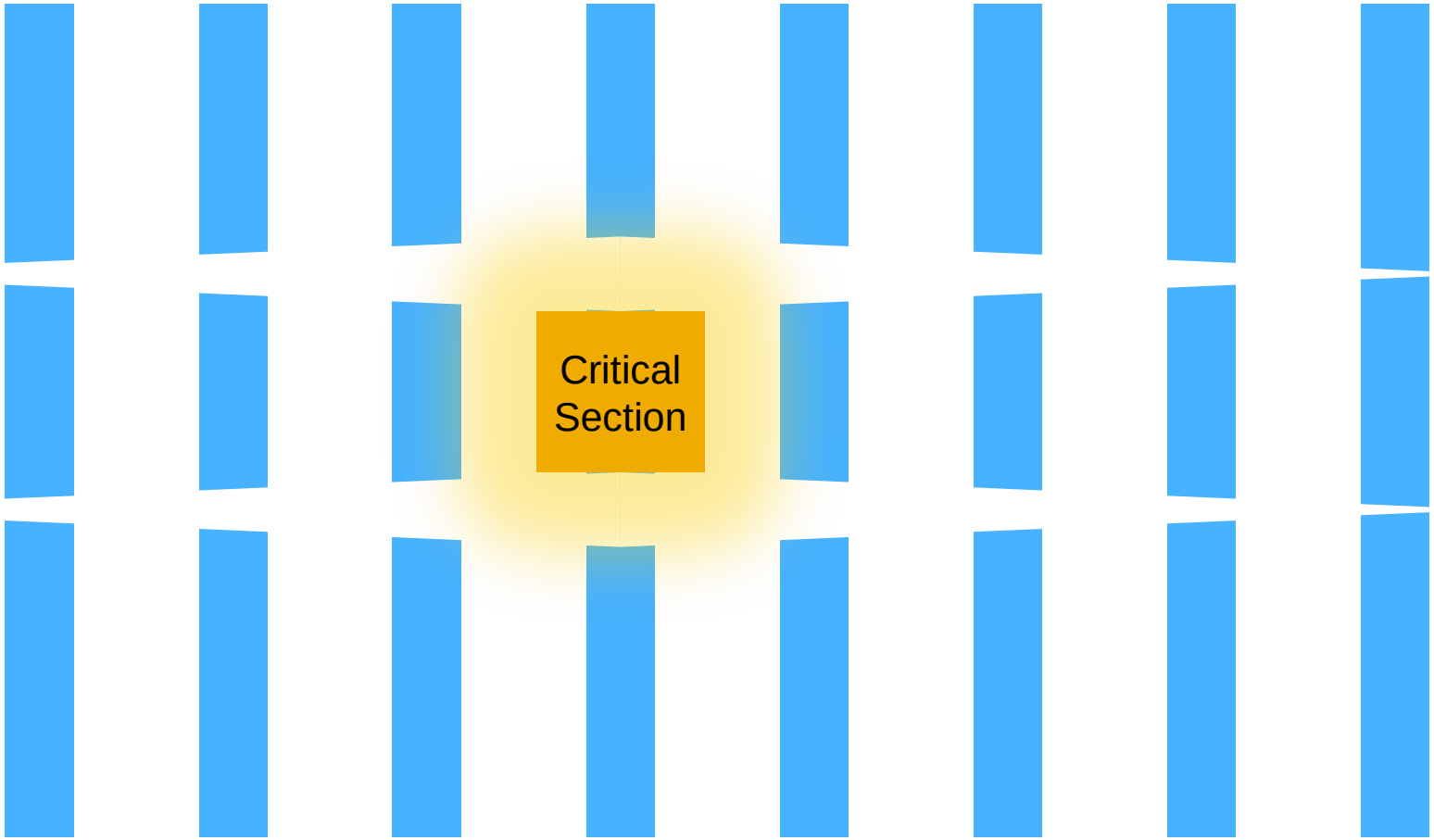
Don’t try to “parallelize” sequential programs – Amdahl’s Law makes it a waste of time to try to undo all the implicit choices made.

Parallel thinking is not any harder than sequential thinking, and ought to be a lot easier, since most activities are parallel.

Thank you! Questions? Debate?

Contact information: david.reed@sap.com

Example: Critical section



Disclaimer

This document contains research concepts from SAP®, and is not intended to be binding upon SAP for any particular course of business, product strategy, and/or development. SAP assumes no responsibility for errors or omissions in this document. SAP does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material