

# Manticore

Hardware-Accelerated RTL Simulation with  
Static Bulk-Synchronous Parallelism

---

Mahyar Emami, Sahand Kashani, Keisuke Kamahori, Sepehr  
Pourghannad, Ritik Raj, and James R. Larus

# RTL Simulation

- **Register Transfer Level (RTL) simulation** is at the heart of functional verification in accelerator design
- Big problem  $\Rightarrow$  RTL simulation is slow
  - Simulating **1 second** of a design could take a **week!**
  - Increasingly important issue:
    - More and more accelerators are being built today
    - Chips are becoming larger and larger

## Research Question:

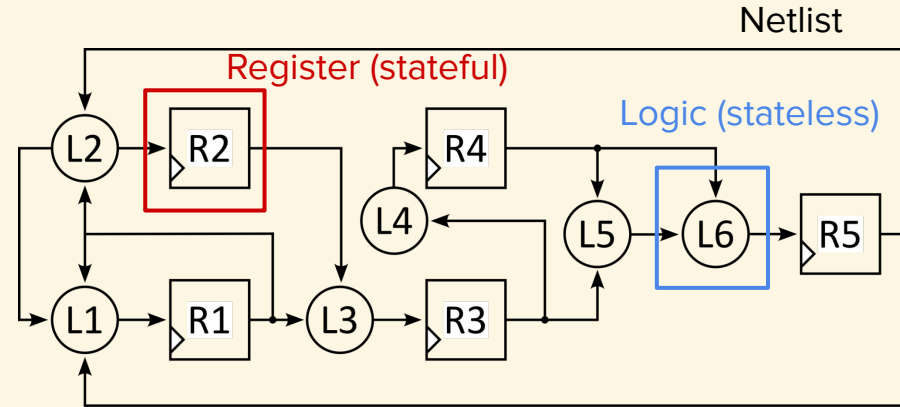
**Can we leverage the inherent parallelism of RTL to speed up its simulation to hundreds of cores?**

# Background

# What is RTL?

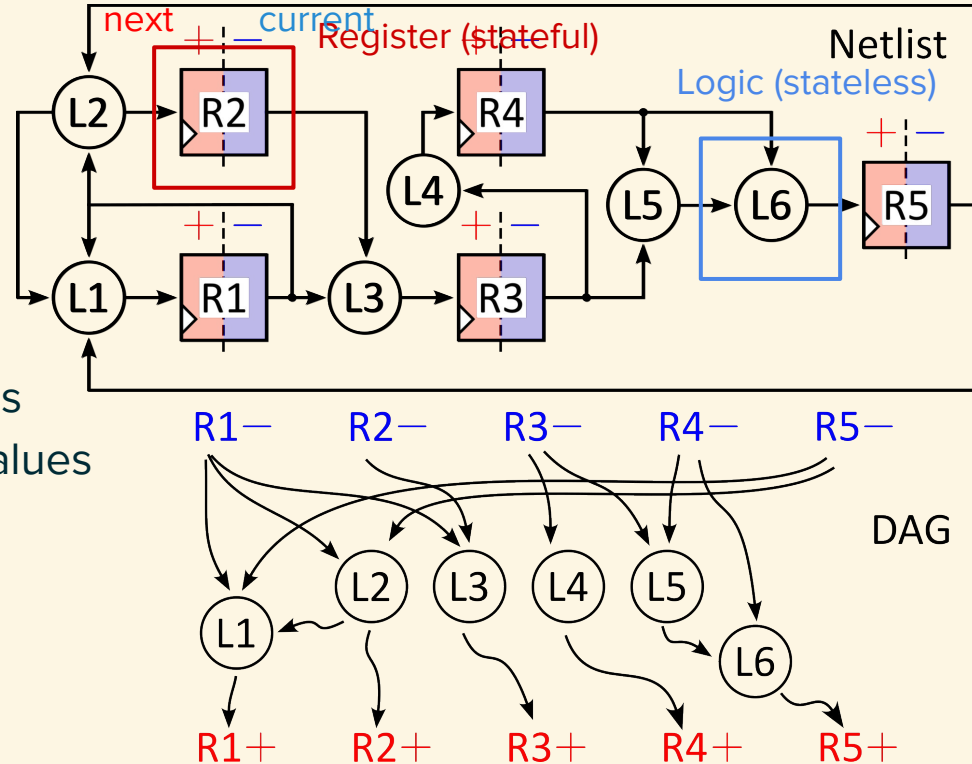
Digital circuits are described in hardware description languages like Verilog and VHDL at the Register Transfer Level (RTL) abstraction

- Stateful registers / memories
- Stateless logic in between



# Cycle-accurate simulation

- How to simulate?
  - Split registers
    - **current** and **next** value
  - At each cycle
    - Compute **next** register values
    - Overwrite **current** register values
    - Repeat

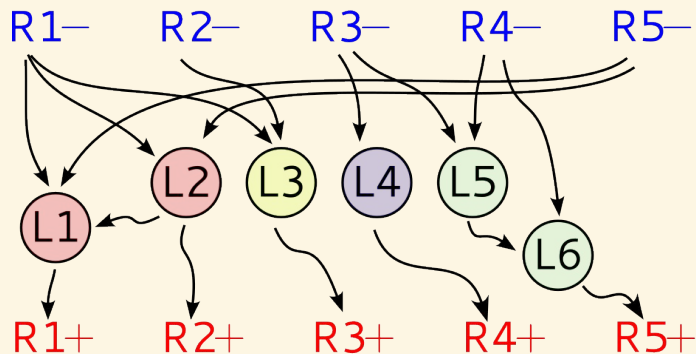


# Bulk-synchronous parallel simulation

RTL simulation lends itself well to BSP

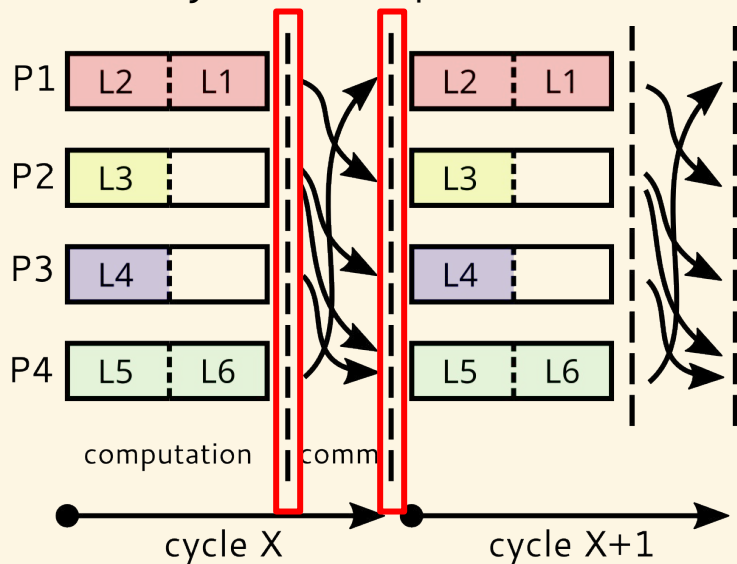
## Computation phase

- Each processor computes its program partition independently
- Synchronize processors



## Communication phase

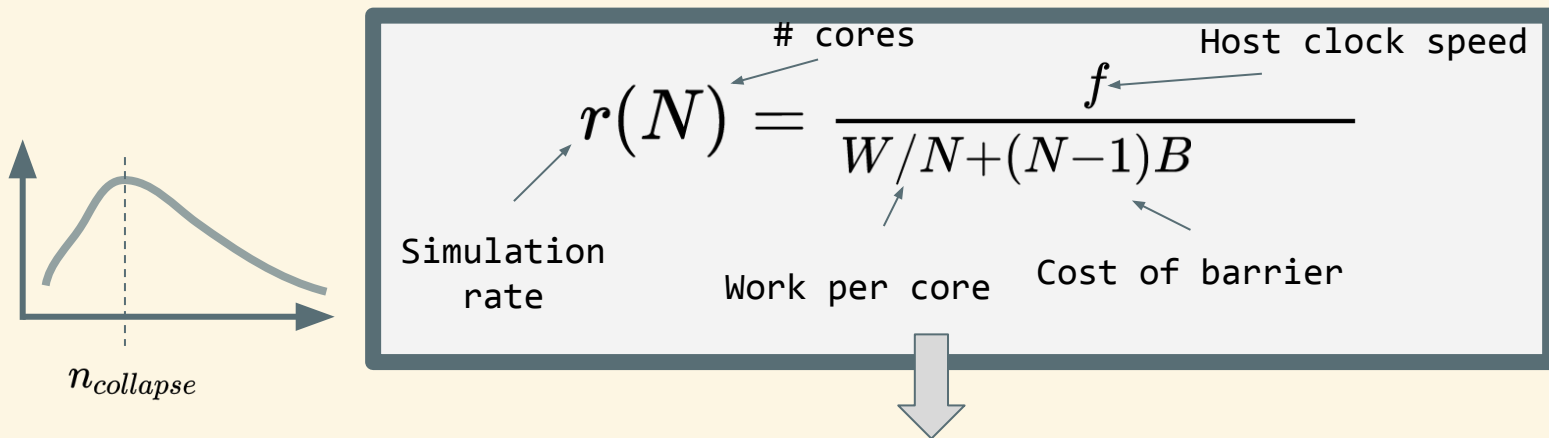
- Producers send their values to consumers
- Synchronize processors



# Bounding software simulation speed

# Parallel Simulation is Doomed on x86!

- Simulation rate on shared-memory **general purpose** machines

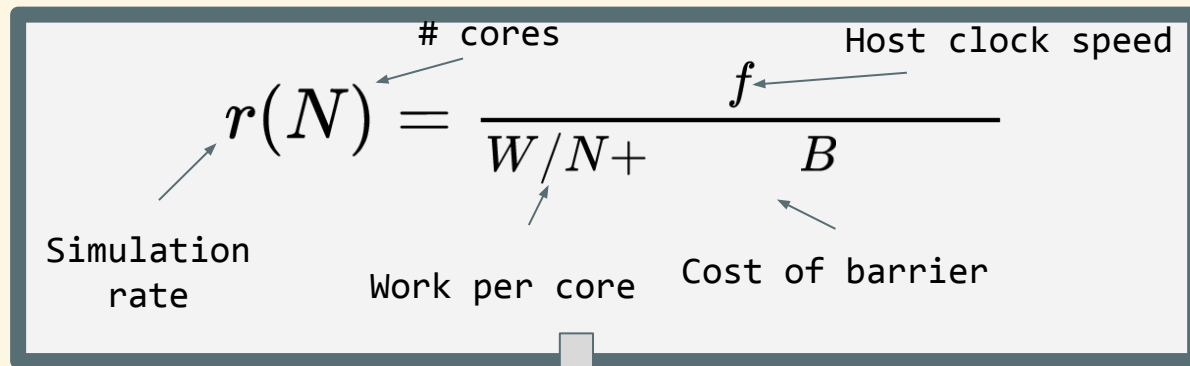


$$\left. \frac{dr}{dN} \right|_{n_{collapse}} = 0 \rightarrow n_{collapse} = \sqrt{W/B}$$

See paper for an experimental demonstration!



# Taming Synchronization Cost


$$r(N) = \frac{f \cdot N}{W + B \cdot N}$$

Simulation rate

# cores

Host clock speed

Work per core

Cost of barrier

$$\frac{dr}{dN} > 0$$

We don't really need a barrier if we could schedule all operations at compile-time!

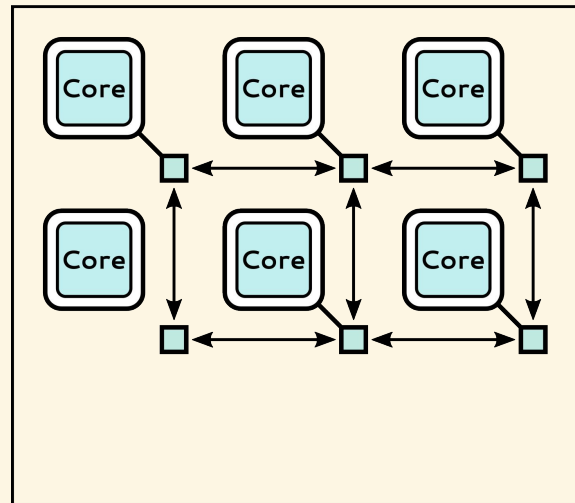
# Manticore Architecture

# Key idea

- Problem
  - Runtime overhead of synchronization
  - Limits scaling to **tens** of cores
- Goal
  - Scale simulation to **hundreds or thousands** of cores
- Solution
  - Eliminate runtime synchronization
  - **Statically schedule all cores**
  - Requires a machine with deterministic behavior

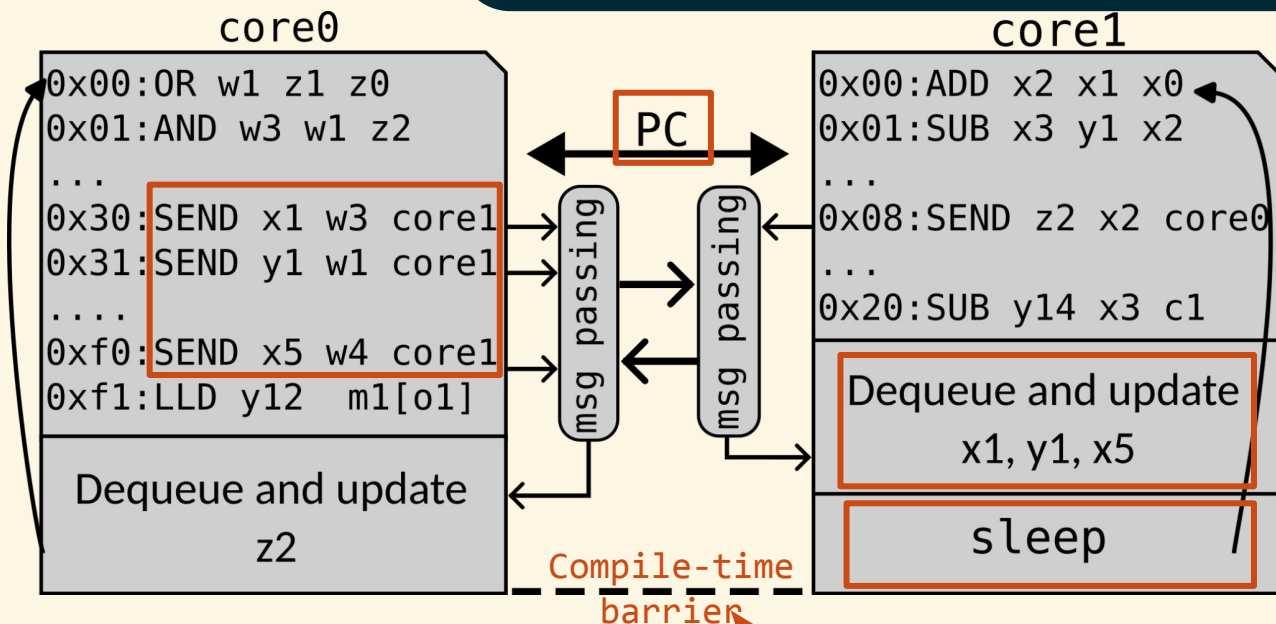
# Manticore architecture: 10,000-foot view

- Static BSP  $\Rightarrow$  native message passing
- **Lock-step execution (cores + NoC)**
  - local memories
  - predication
- Global stall for non-deterministic events
  - See paper for more



# Static BSP execution with Manticore

See paper for march details!



Lock-step execution

- Same PC on all cores

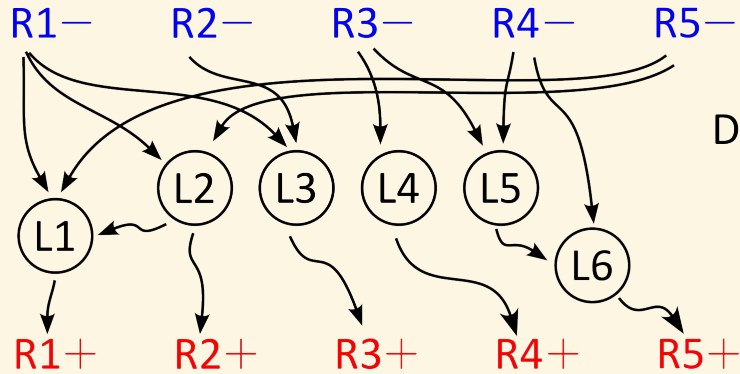
Message-passing:

- Mixed computation and communication
- Delayed updates

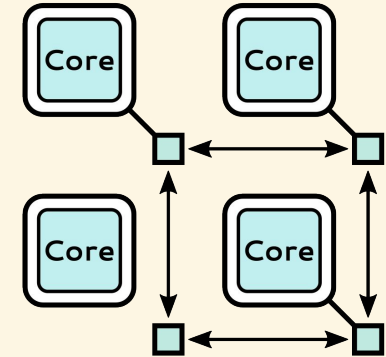
Compile-time arrive-await barrier

- “NOP” until straggler is done
- No runtime synchronization

# From RTL to parallel execution



DAG **Compiler** →



Manticore's hardware relies **entirely** on its compiler for:

- Parallelizing the netlist
- Scheduling instructions (data hazards)
- Scheduling messages (message delivery)

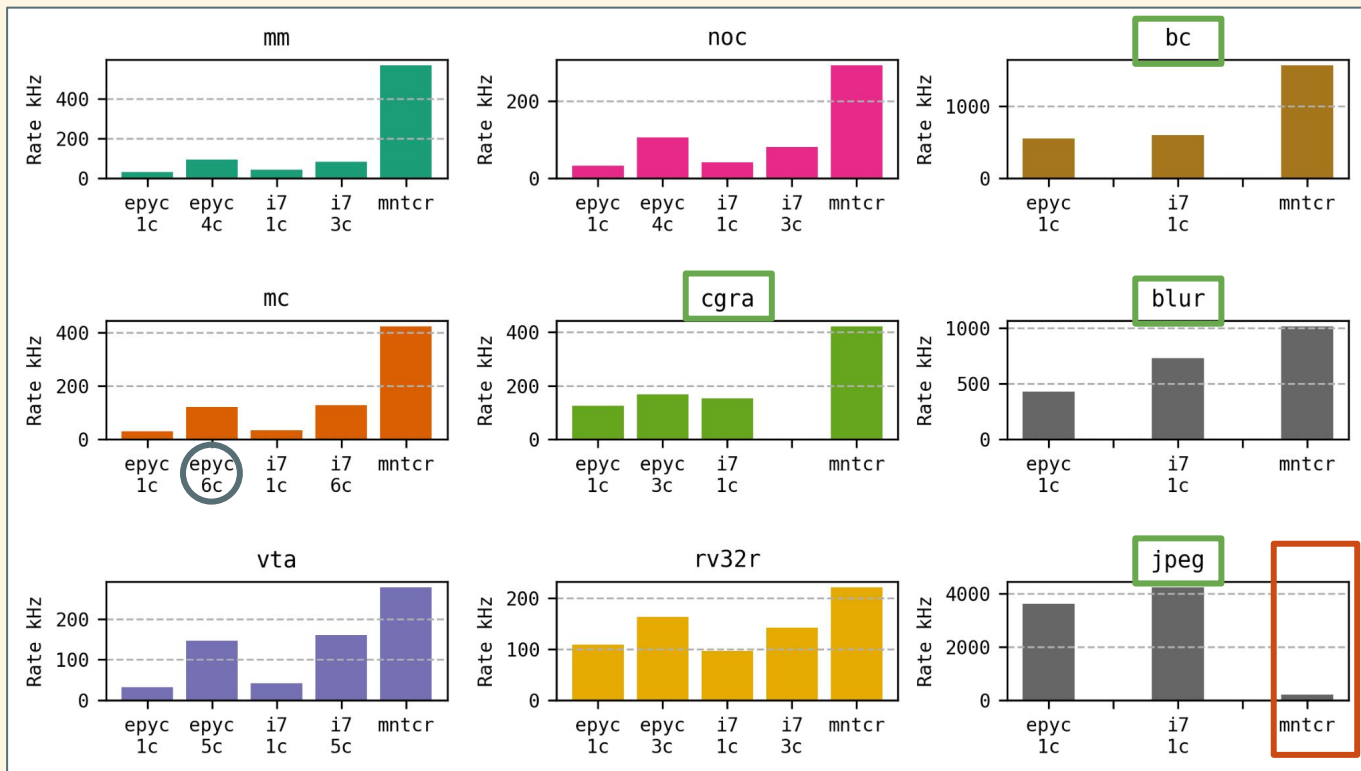
# Evaluation

# Hardware setup

	Verilator v5.006 (Feb 2023)		Manticore
Hardware	AMD EPYC 7V73X	Intel Core i7 9700K	<b>Prototyped</b> on Xilinx Alveo U200
# cores	<b>120</b> (dual socket)	<b>8</b>	<b>225</b>
Freq. GHz	<b>3.0–3.5</b>	<b>4.6–4.9</b> (overclocked)	<b>0.475</b>
SRAM (MiB)	259.6	14.5	18.45
Released	Q1 2022	Q4 2018	–



# Simulation Rate



Few did not scale at all with Verilator

At best scales up to only 6 cores with Verilator

jpeg is sequential

# Conclusion

- General-purpose multicores have poor thread scaling in RTL simulation
  - Synchronization overheads limit scaling to **tens** of cores
- We propose Manticore: an architecture for scalable parallel RTL simulation
  - **Hundreds** of cores
- Key ideas
  - A deterministic machine that allows implementing Static BSP
  - Static BSP replaces **runtime synchronization** with **compile-time synchronization**
  - Statically schedule **entire machine**