EPFL

Hardware-Accelerated, Fine-Grain BSP

James Larus

School of Computer and Communication Sciences

EPFL

EMERALD Workshop
PODC/SPAA

June 21, 2024

École polytechnique fédérale de Lausanne

PhD students

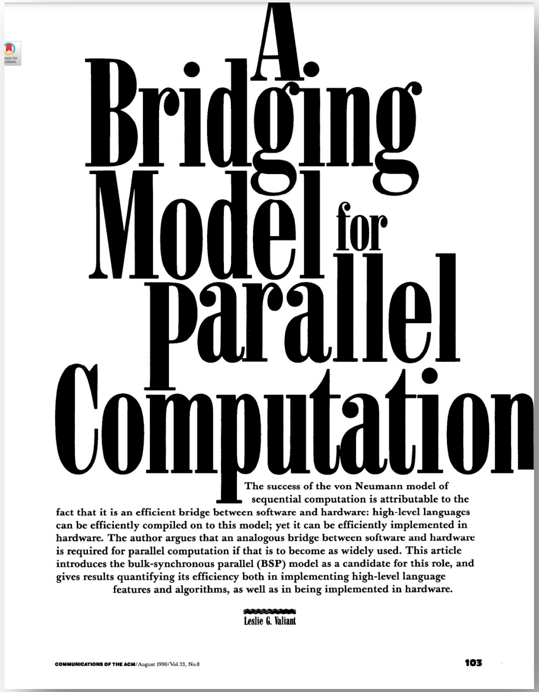
Building Chips Faster: Hardware-Compiler Co-Design for Accelerated RTL Simulation

Currently at MangoBoost

Highly Parallel RTL Simulation

Finishing summer 2024

Bulk-Synchronous Parallel computation model



A Bridging Model for Parallel Computation

The success of the von Neumann model of sequential computation is attributable to the fact that it is an efficient bridge between software and hardware: high-level languages can be efficiently compiled on to this model; yet it can be efficiently implemented in hardware. The author argues that an analogous bridge between software and hardware is required for parallel computation if that is to become as widely used. This article introduces the bulk-synchronous parallel (BSP) model as a candidate for this role, and gives results quantifying its efficiency both in implementing high-level language features and algorithms, as well as in being implemented in hardware.

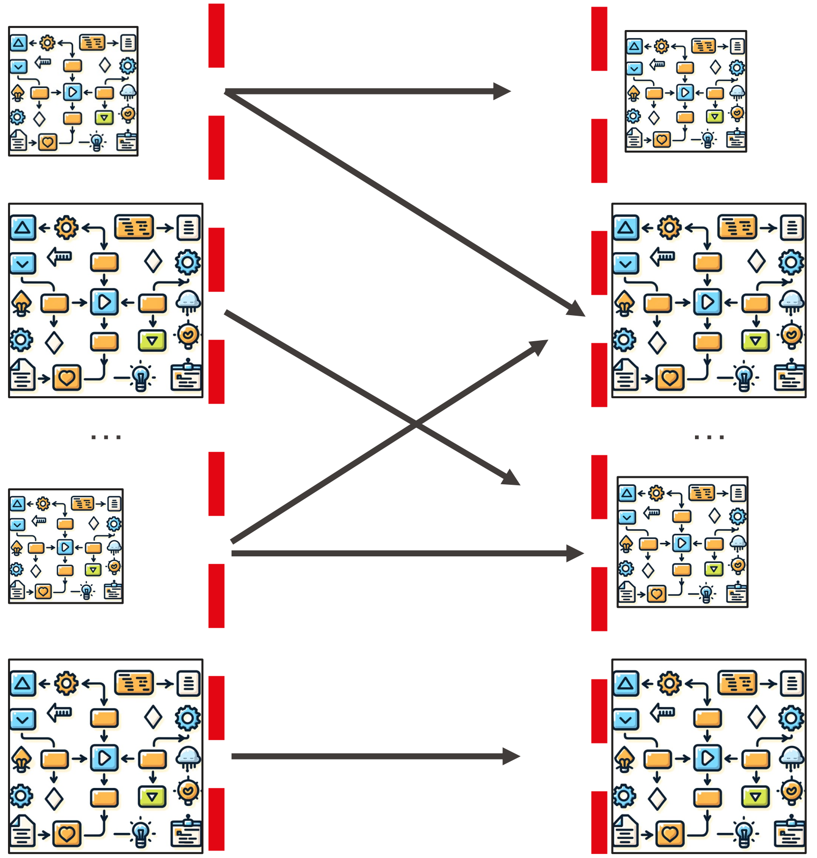
Leslie G. Valiant

COMMUNICATIONS OF THE ACM/August 1990/Vol.33, No.8 103

- BSP
  - Leslie Valiant, CACM 1990
  - Cited 5500+ times
- Popular parallel programming model
  - Barriers widely used before BSP
  - Valiant formalized and named

BSP

Compute **Barrier** Communicate **Barrier** Compute …

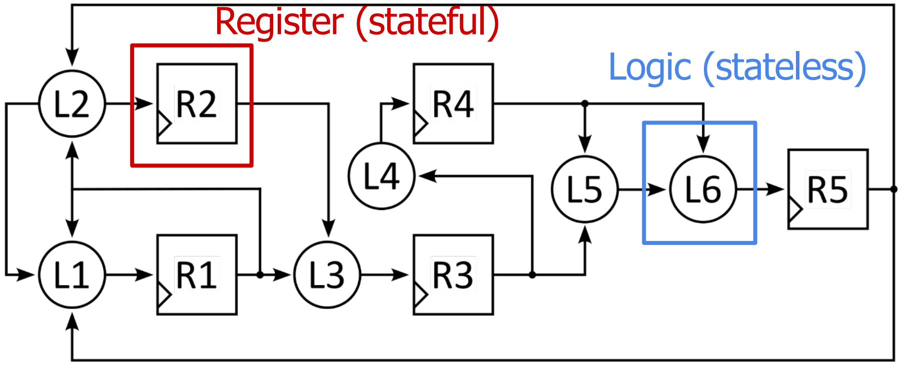


Can eliminate one barrier with double buffering

RTL hardware design
(application domain)

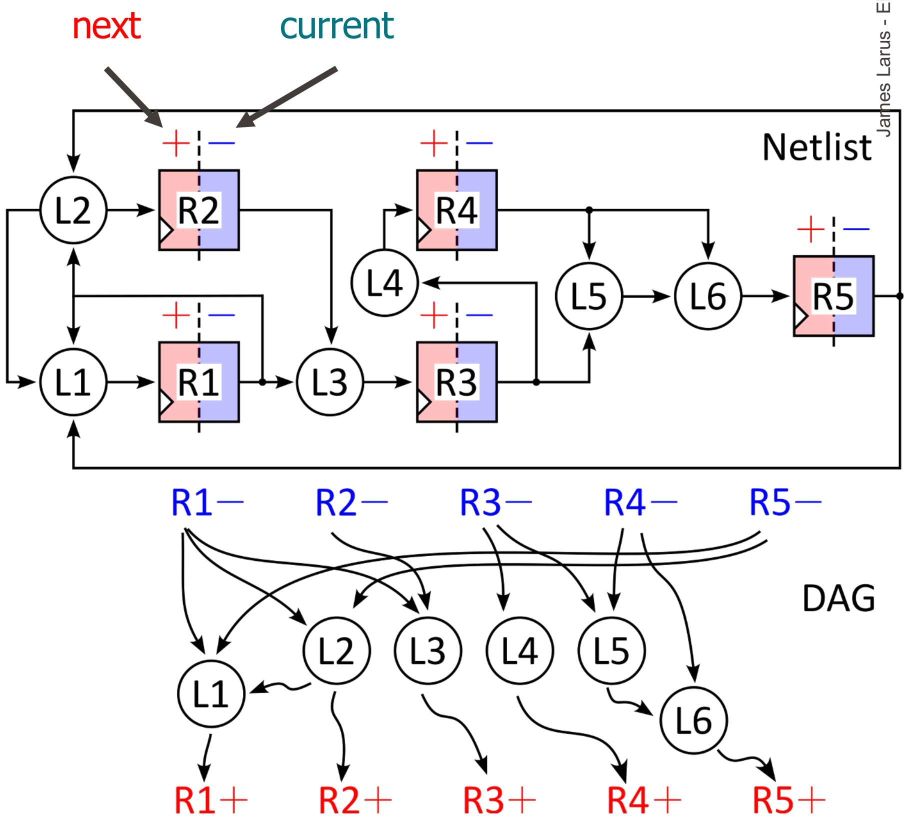
- Register transfer level (RTL)
  - Stateful registers/memories
  - Stateless logic
  - Update state at clock edge

- Hardware description language (HDL)
  - Describe digital circuits
  - e.g., Verilog and VHDL



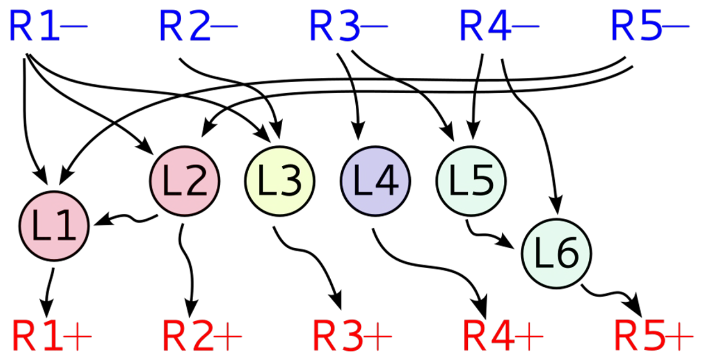
Cycle-accurate simulation

- Evaluate (simulate) RTL
- Registers contain **current** and **next** values
- At each cycle
  - compute **next** values
- At clock edge
  - **current** ← **next**

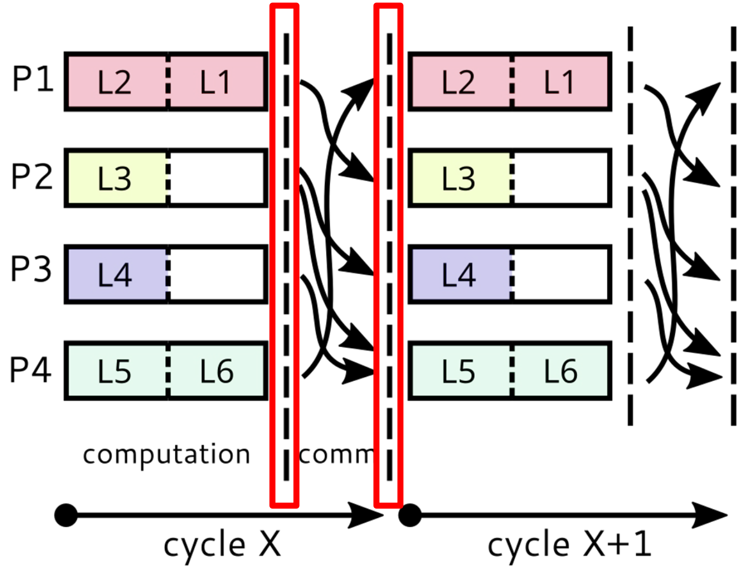


BSP simulation

- **Computation**
  - Processor computes slice(s) of netlist
  - No inter-slice dependencies



- **Communication**
  - Producers send values to consumers

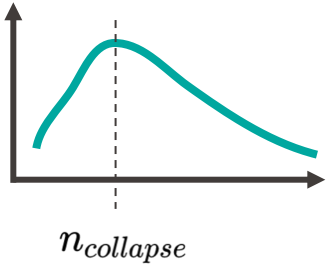


Widely seen that parallel simulation is slow on x64

- Simulation rate on shared-memory, general-purpose computer

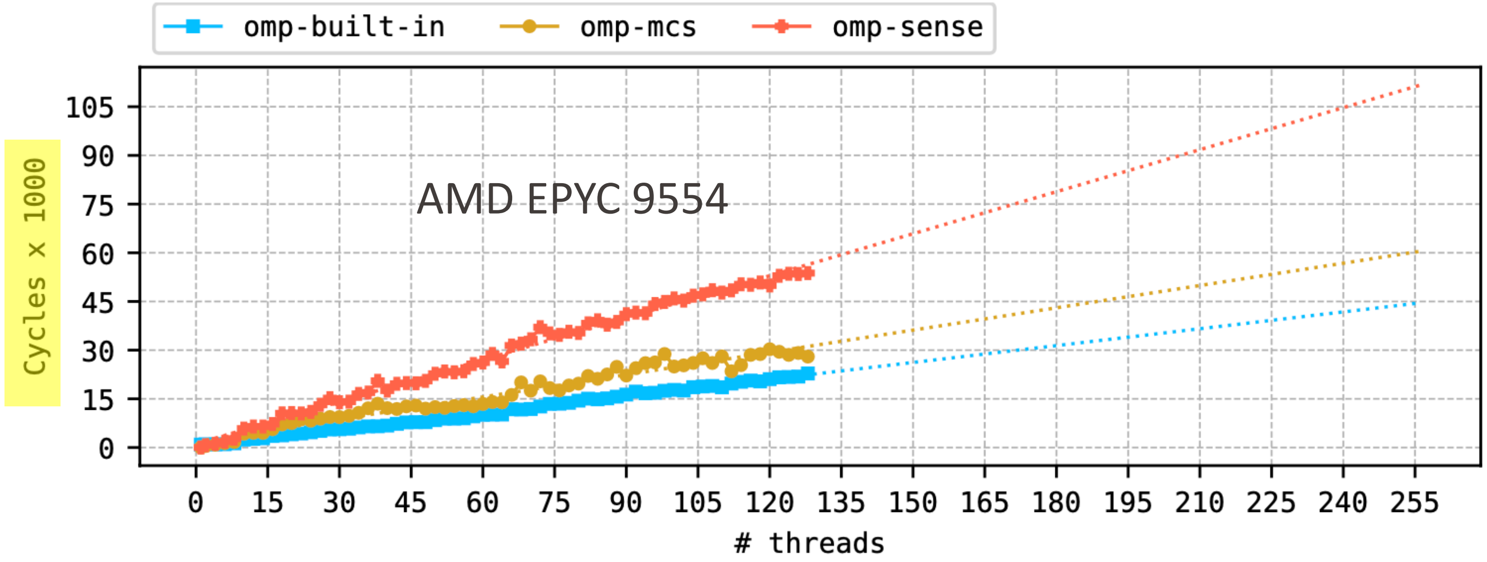
$$\underbrace{r}_{\text{Simulation rate}}(\overset{\text{\# cores}}{N}) = \frac{\overset{\text{Host clock speed}}{f}}{\underbrace{W/N}_{\text{Work per core}}+\underbrace{(N-1)B}_{\text{Cost of barrier}}}$$

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

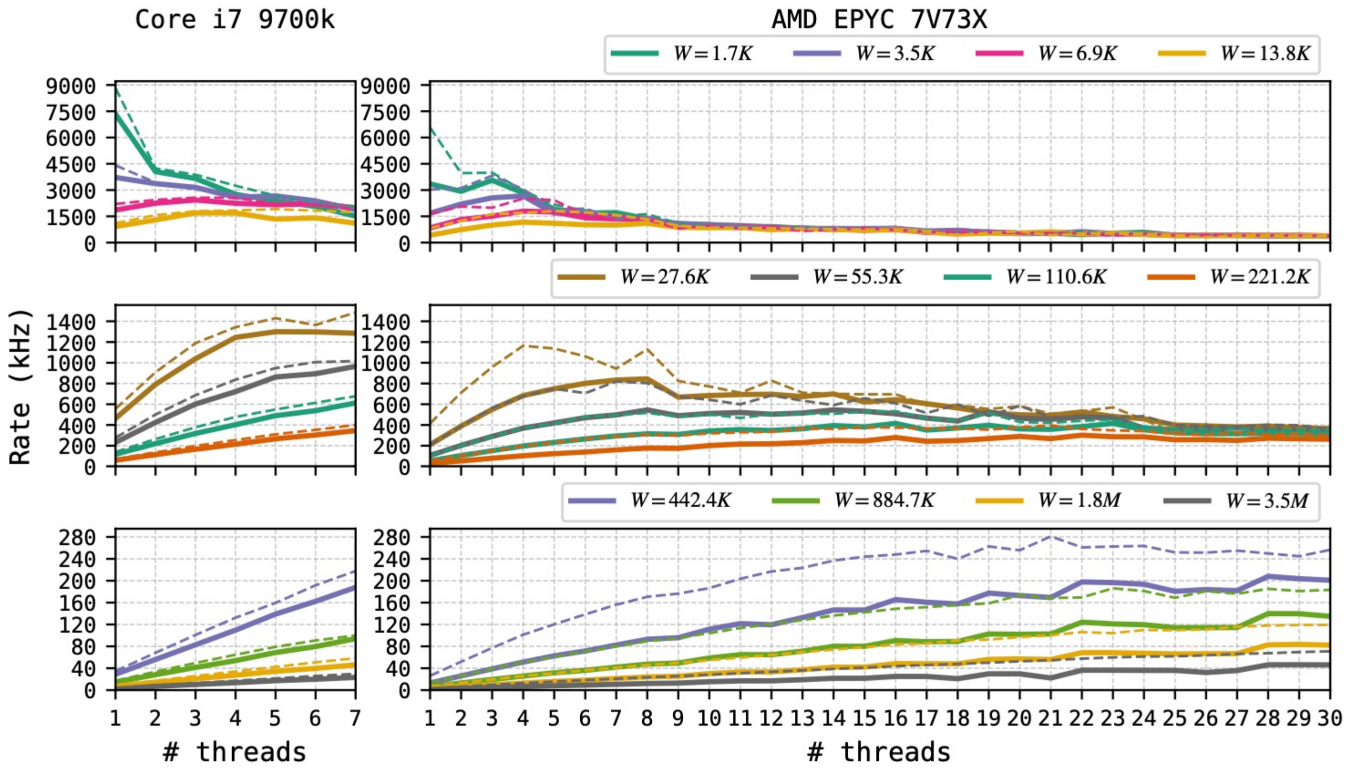


EPFL

Barriers are costly



Effect on BSP simulation



Fine grain

Medium grain

Large grain

Taming synchronization

# cores

Host clock speed

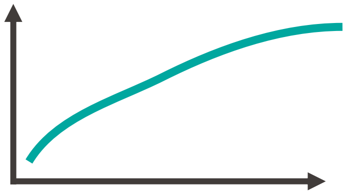
$$r(N) = \frac{f}{W/N + \qquad B}$$

Simulation rate

Work per core

Cost of barrier

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

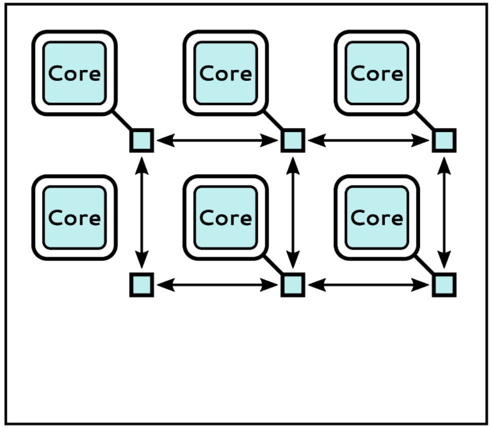


Manticore simulation processor

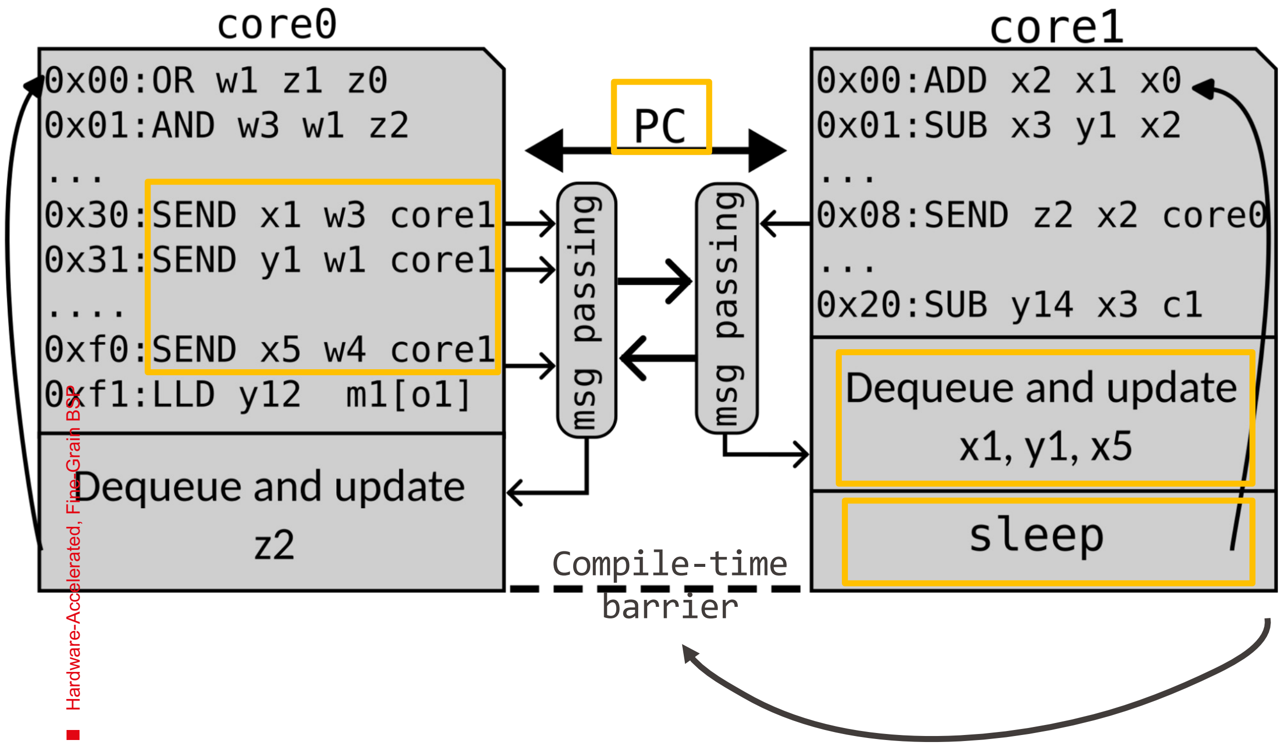
- Problem
  - High overhead of runtime synchronization
  - Limits scaling to tens of cores
- Goal
  - Scale to hundreds or thousands of cores
- **Manticore**
  - Statically schedule computation and communication
    - Eliminate runtime overhead
  - Requires machine with deterministic behavior

Manticore design

- Static BSP
  - Statically scheduled message passing
  - Statically scheduled barriers
- Lock-step execution (cores + NoC)
  - Static local memories
  - Predication, not branches
- Global stall hides non-deterministic events
  - DRAM access or user interaction
- Similar to MIT RAW machine
  - Logic simulation has little dynamic behavior



Static BSP execution on Manticore



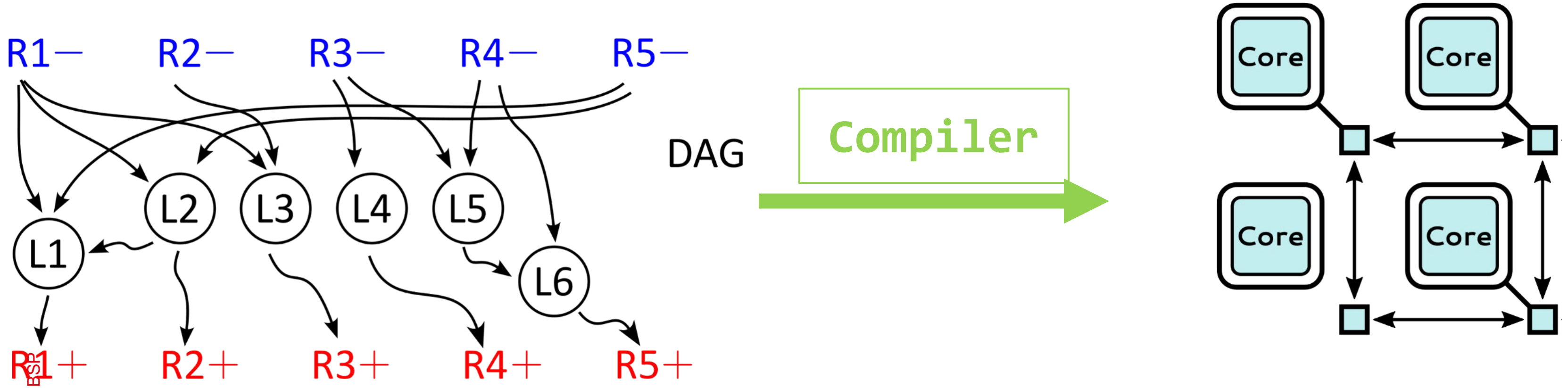
Lock-step execution
- Same PC on all cores
- Different code on each

Message-passing
- Overlap computation and communication
- Schedule network traffic
- Delay remote updates

Compile-time arrive-await barrier
- “NOP” delays until straggler finishes

No runtime synchronization

From RTL to parallel execution



Manticore's hardware is fast

- Processors and network are simple and easily pipelined
- 225+ cores on medium-sized FPGA
- Heavily optimized to run at 475 Mhz

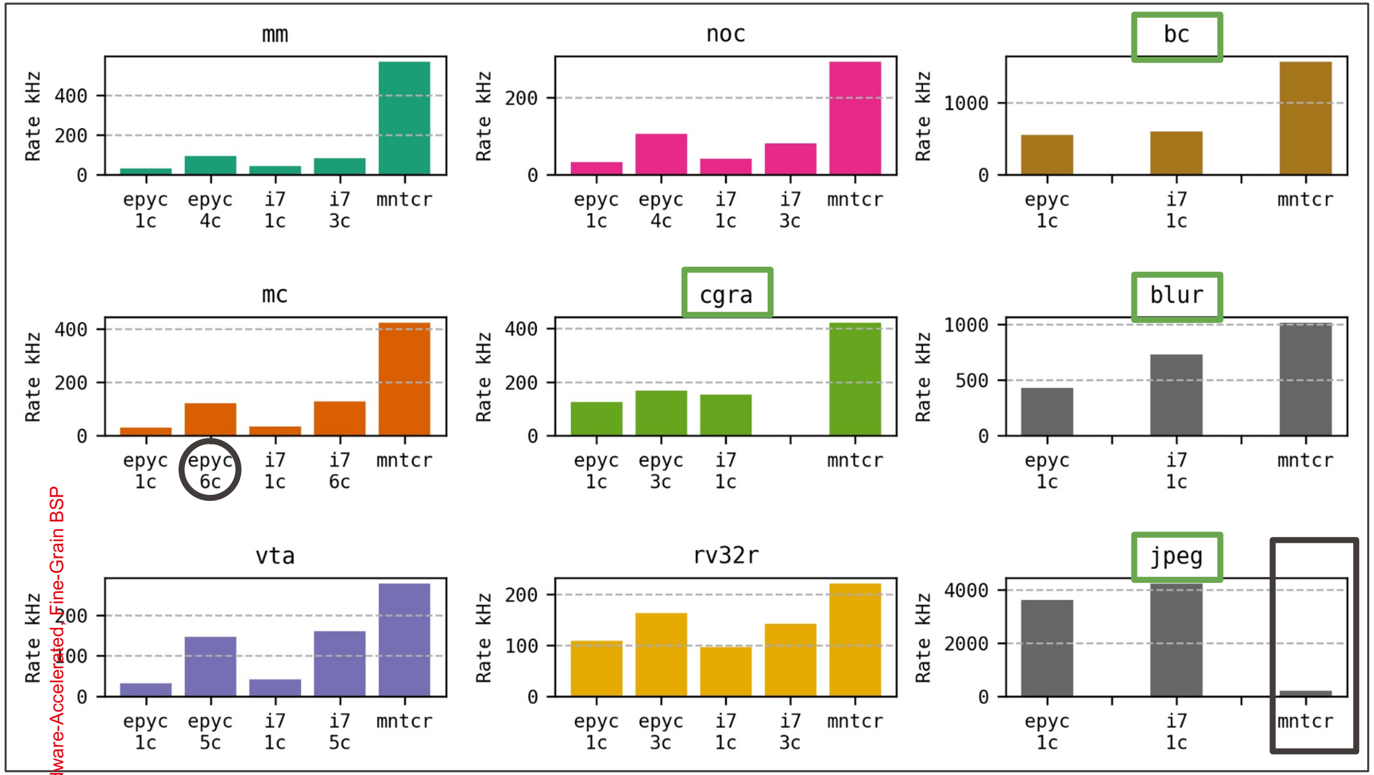
EPFL

Evaluation

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

EPFL

Simulation rate



Did not scale with Verilator

At best scales up to 6 cores with Verilator

jpeg is sequential

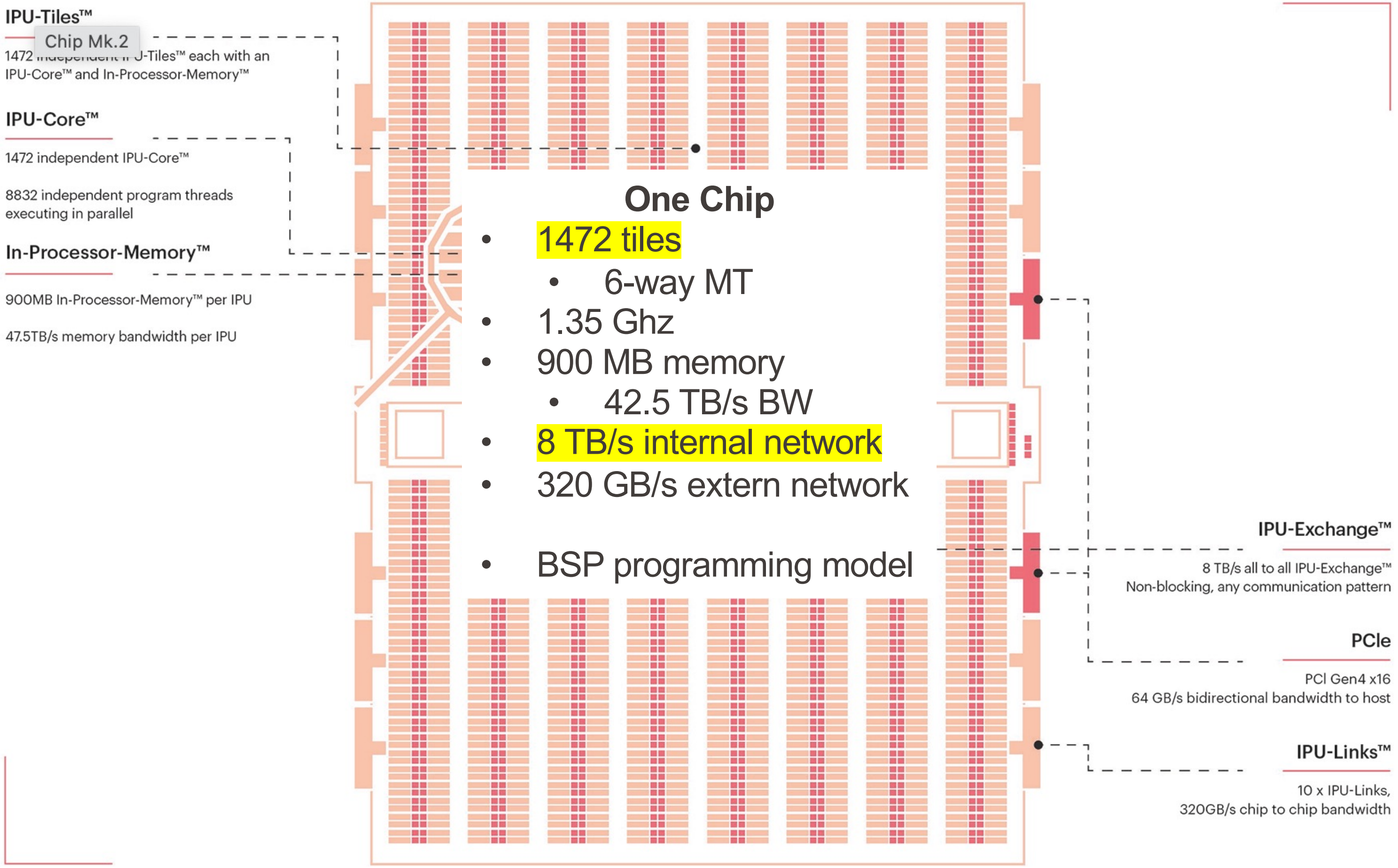
Scaling up

- Can RTL simulation run on thousands of processors?

- Manticore constrained by size of FPGAs and our design decisions
  - ASICs have different tradeoffs than FPGAs
  - Decided not to build v2
  - Did not want to simulate a simulator

- Luckily, found suitable commercial system
  - Designed for BSP computation

Graphcore IPU


IPU-Tiles™
Chip Mk.2
1472 independent IPU-Tiles™ each with an IPU-Core™ and In-Processor-Memory™
IPU-Core™
1472 independent IPU-Core™
8832 independent program threads executing in parallel
In-Processor-Memory™
900MB In-Processor-Memory™ per IPU
47.5TB/s memory bandwidth per IPU
IPU-Exchange™
8 TB/s all to all IPU-Exchange™
Non-blocking, any communication pattern
PCIe
PCI Gen4 x16
64 GB/s bidirectional bandwidth to host
IPU-Links™
10 x IPU-Links,
320GB/s chip to chip bandwidth

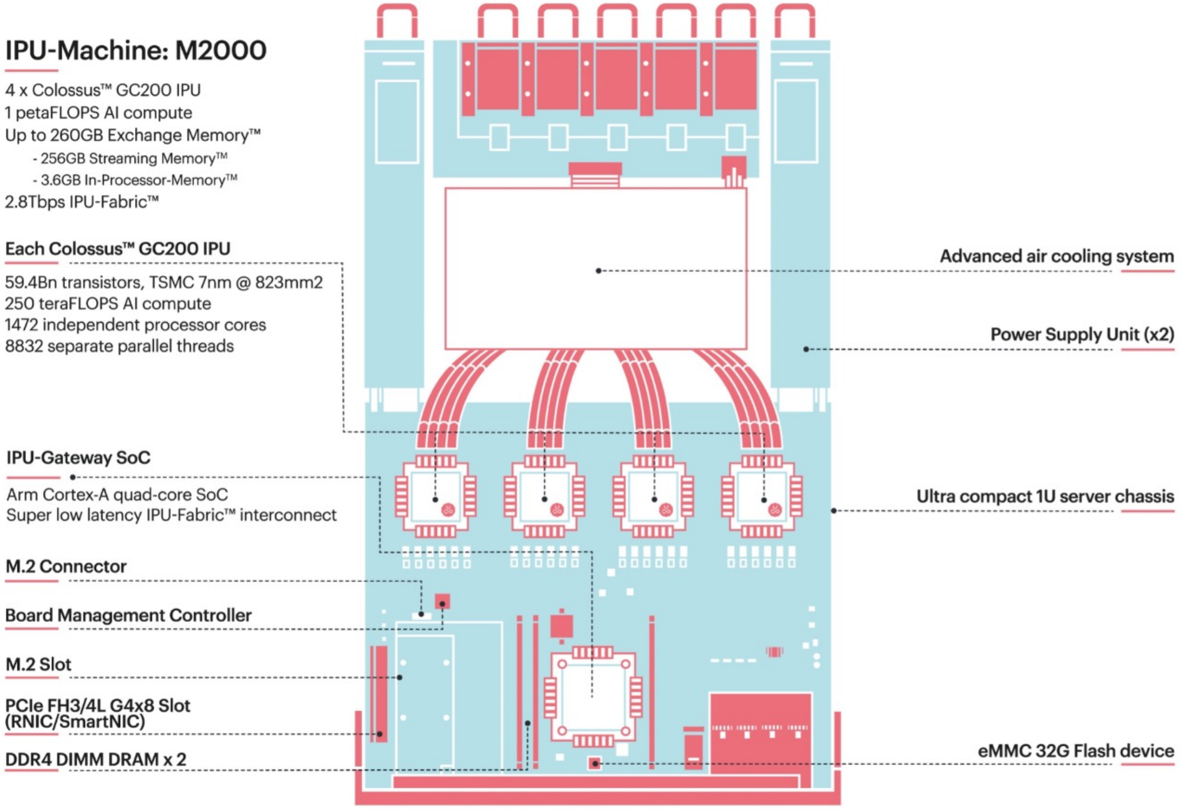
One Chip

- 1472 tiles
  - 6-way MT
- 1.35 Ghz
- 900 MB memory
  - 42.5 TB/s BW
- 8 TB/s internal network
- 320 GB/s extern network

- BSP programming model

https://www.graphcore.ai/products/ipu

IPU systems



IPU-Machine: M2000

4 x Colossus™ GC200 IPU
1 petaFLOPS AI compute
Up to 260GB Exchange Memory™
- 256GB Streaming Memory™
- 3.6GB In-Processor-Memory™

2.8Tbps IPU-Fabric™

Each Colossus™ GC200 IPU

59.4Bn transistors, TSMC 7nm @ 823mm2
250 teraFLOPS AI compute
1472 independent processor cores
8832 separate parallel threads

IPU-Gateway SoC

Arm Cortex-A quad-core SoC
Super low latency IPU-Fabric™ interconnect

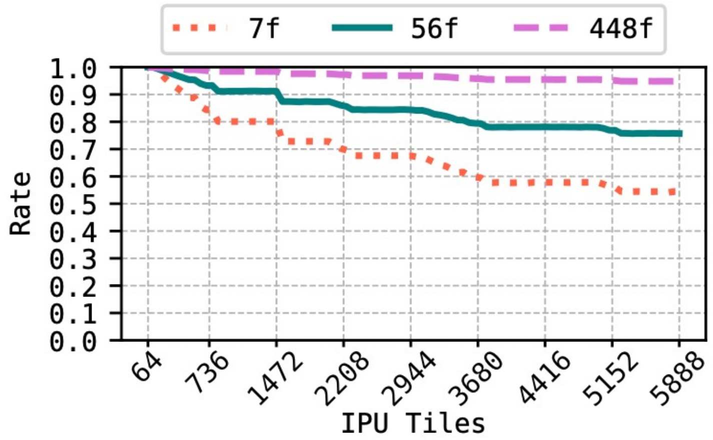
M.2 Connector
Board Management Controller
M.2 Slot
PCIe FH3/4L G4x8 Slot (RNIC/SmartNIC)
DDR4 DIMM DRAM x 2
Advanced air cooling system
Power Supply Unit (x2)
Ultra compact 1U server chassis
eMMC 32G Flash device

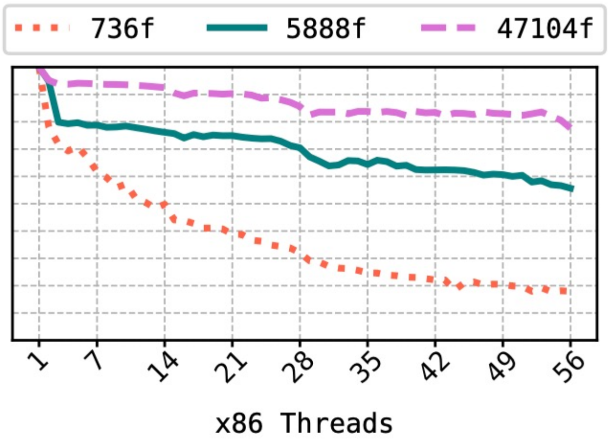
https://docs.graphcore.ai/projects/graphcore-ipu-m2000-datasheet/en/latest/overview.html

Parendi

- Verilog compiler and runtime for Graphcore M-2000
  - Based on Verilator
  - Open source

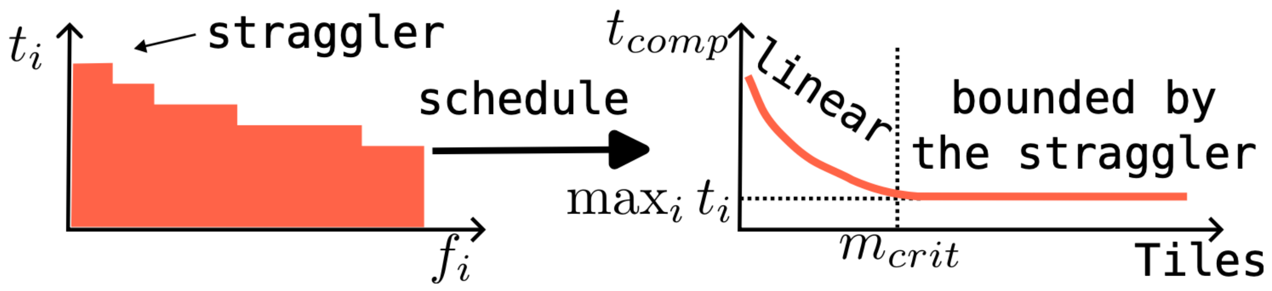
Low overhead synchronization

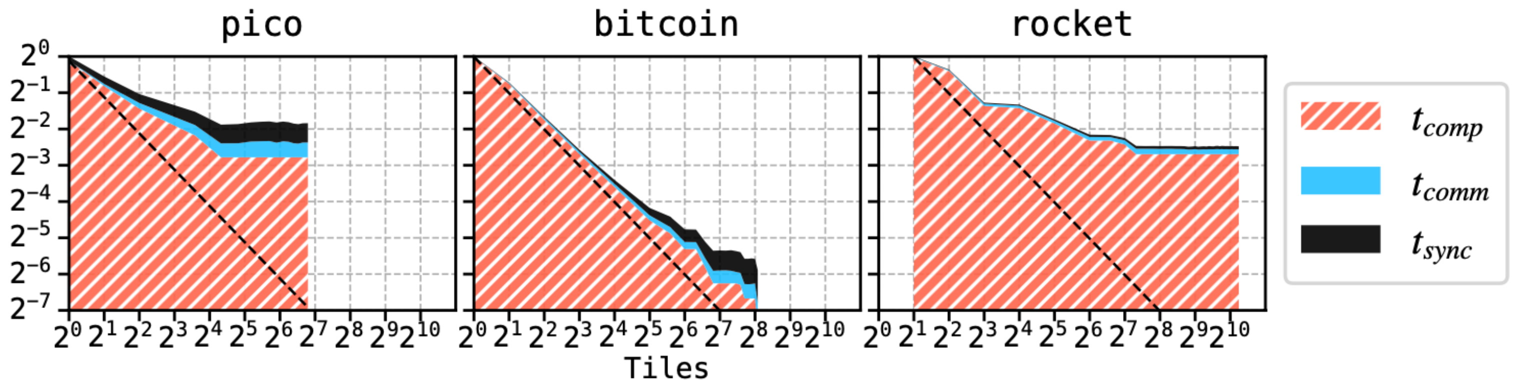




f = xorshift32 pseudo-random number generators (PRNG) — 3 XORs and 3 shifts

Non-uniform tasks limit speedup





Scaling requires large designs

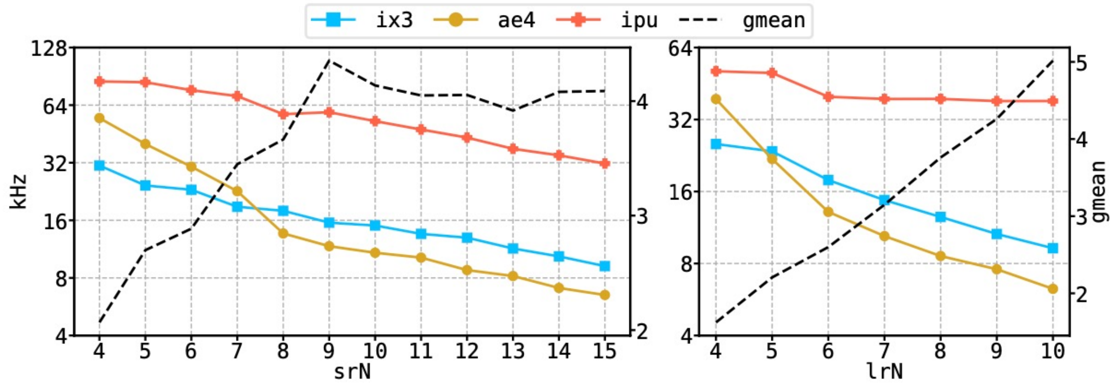
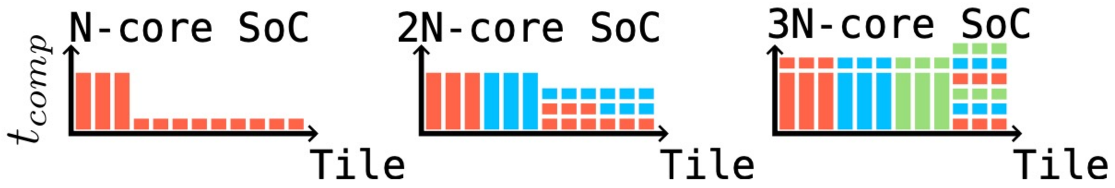
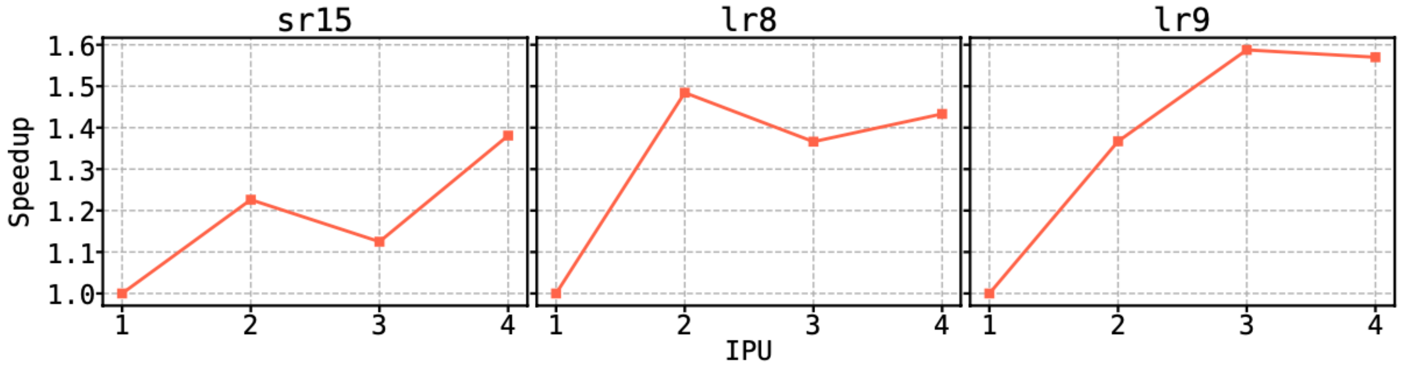


Figure 5.18: PARENDI vs. Verilator in coping with increasing design size. The left axis shows the best simulation rate in base-2 logarithmic scale. Right axis shows the gmean speedup of PARENDI against Verilator (dashed lines).



Non-uniform communication



Conclusion

- General-purpose parallel computers poorly support fine-grain BSP
  - Synchronization overhead limits scaling
- Manticore enables scalable parallel RTL simulation
  - Hundreds of cores
  - Built compiler and FPGA accelerator
  - Compile-time synchronization and communications
  - Simple HW enables many cores and high clock rate
- Architecture can directly support fine-grain BSP
  - Graphcore IPU is a great example of what is possible with an ASIC
  - More general, perhaps more scalable, than Manticore
  - Non-uniform communication costs are new scheduling challenge
- Massively parallel RTL simulation is possible!

Questions?

Open source:

- https://github.com/ManticoreRTL
- https://github.com/epfl-vlsc/parendi