UBC System on a Chip research lab

# Rapid Synthesis and Simulation of Computational Circuits on an MPPA

**David Grant**

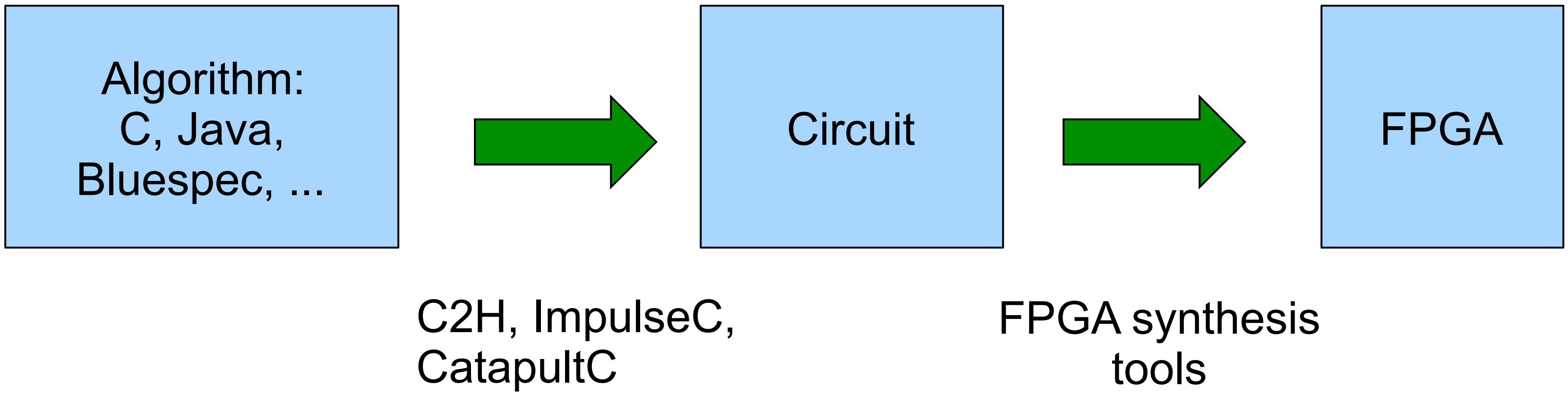
Dr. Guy Lemieux, Graeme Smecher,
Rosemary Francis (U. Cambridge)

# Overview

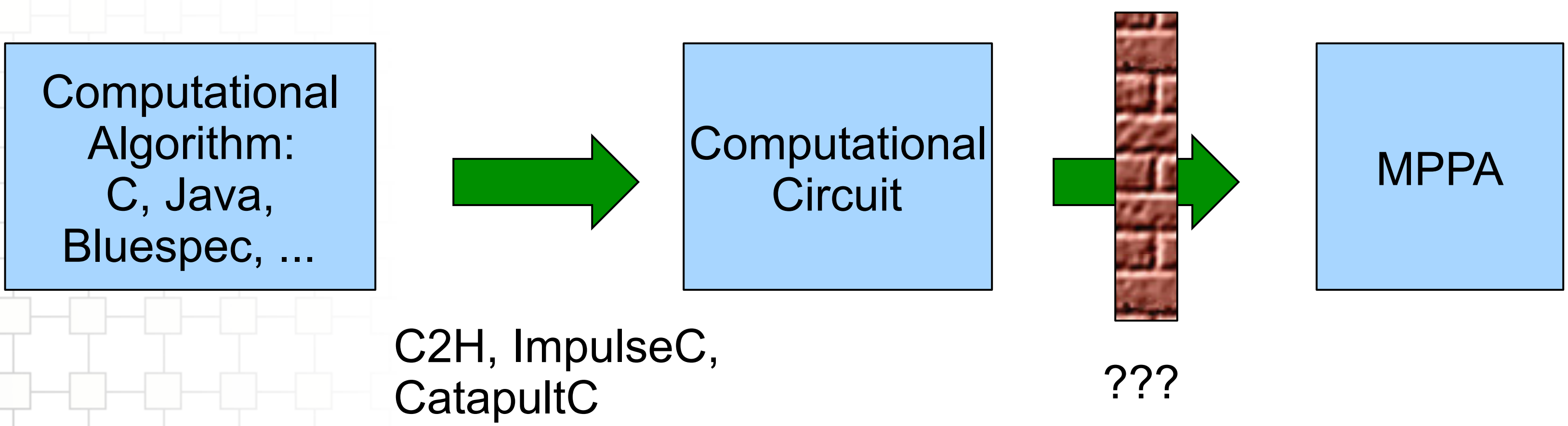
- **Introduction and Motivation**
- MPPA Architecture
- CAD Tools
- Results

# Motivation

- Old and Busted




- New Hotness



# Introduction

- Computational Circuits
  - Software converted to hardware for performance
    - ex. molecular dynamics, rendering, encoding, ...
    - Word-oriented circuits
- Computational Circuits can be Very Large
  - Problem 1 and 2: Long synthesis and long simulation
  - Problem 3: Hard capacity limit in FPGAs
  - Wastes designer productivity
- No Existing Solutions
  - Either fast synthesis or fast simulation, not both

# Introduction

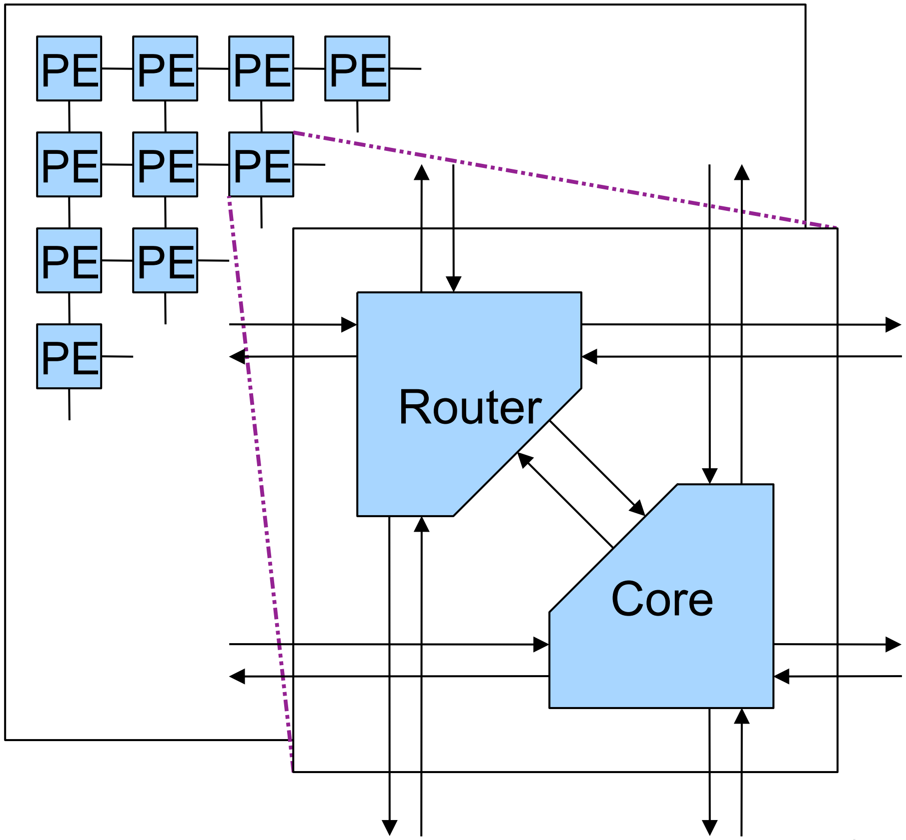
- Objectives
  - Quickly synthesize and simulate computational circuits
  - Achieve a soft capacity limit (trade speed for area)
- Solution
  - MPPA Architecture
    - Time-multiplexed, high speed interconnect
    - Tuned for Verilog execution
  - Fast Tools
    - Behavioural, word-level
    - Target coarse-grained MPPA architecture
  - Scalable: speed vs. area tradeoff

# Overview

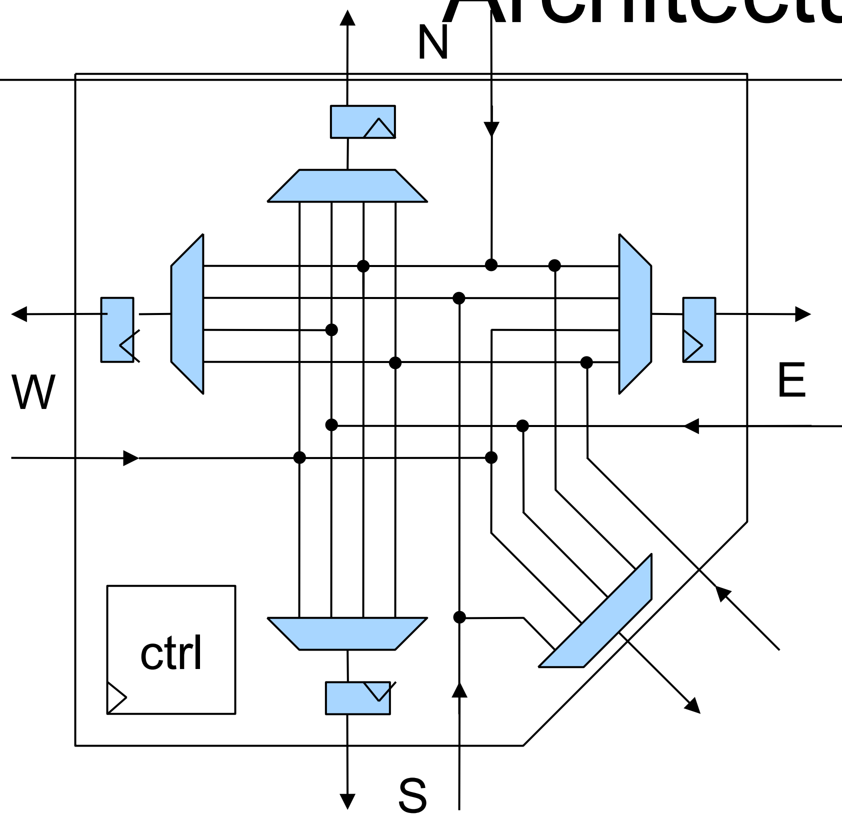
- Introduction and Motiviation
- **MPPA Architecture**
- CAD Tools
- Results

# Architecture

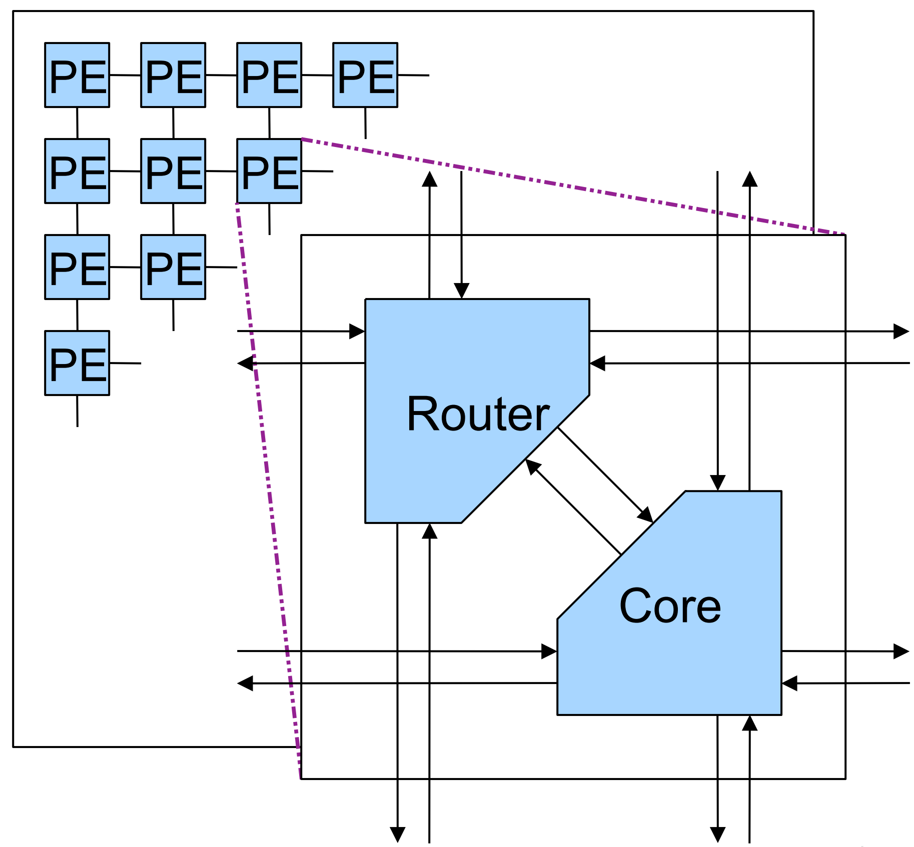
- Architecture
  - ➔ 2D array of processing elements (PEs)
    - PE is a small, fast sequential computer
    - Contains router and core
  - ➔ Pipelined high-bandwidth interconnect
- Execution Model
  - ➔ *n* cycle fixed schedule
    - One user cycle
  - ➔ PEs only communicate with neighbours



# Architecture – PE Router

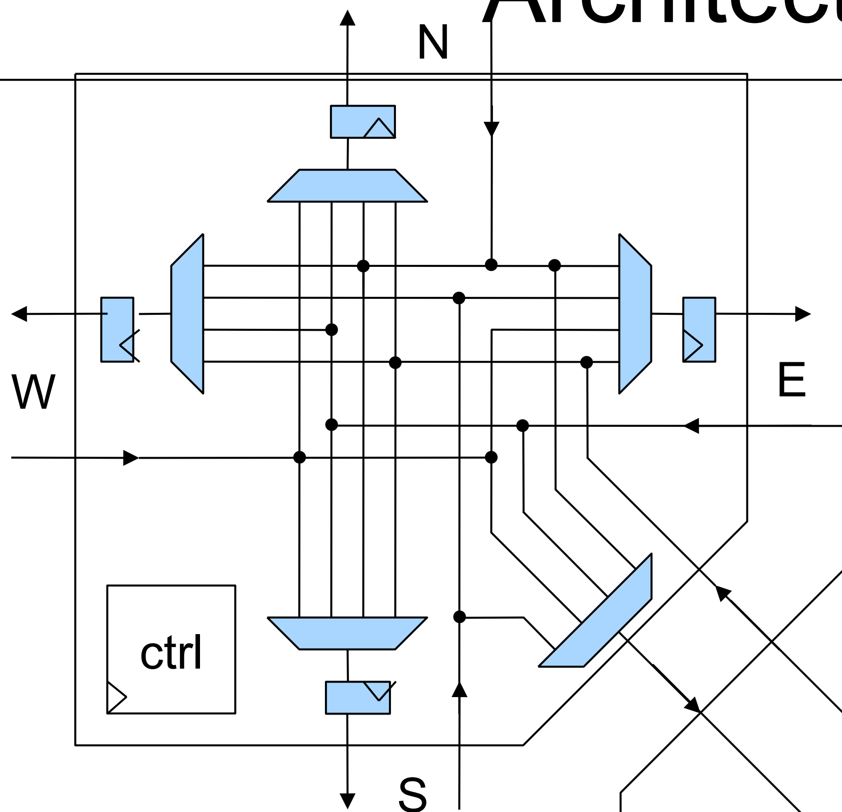



- Fixed static schedule
- 1GHz+ target
- 5x5 data pipelined crossbar
- 1 word/cycle to/from PE

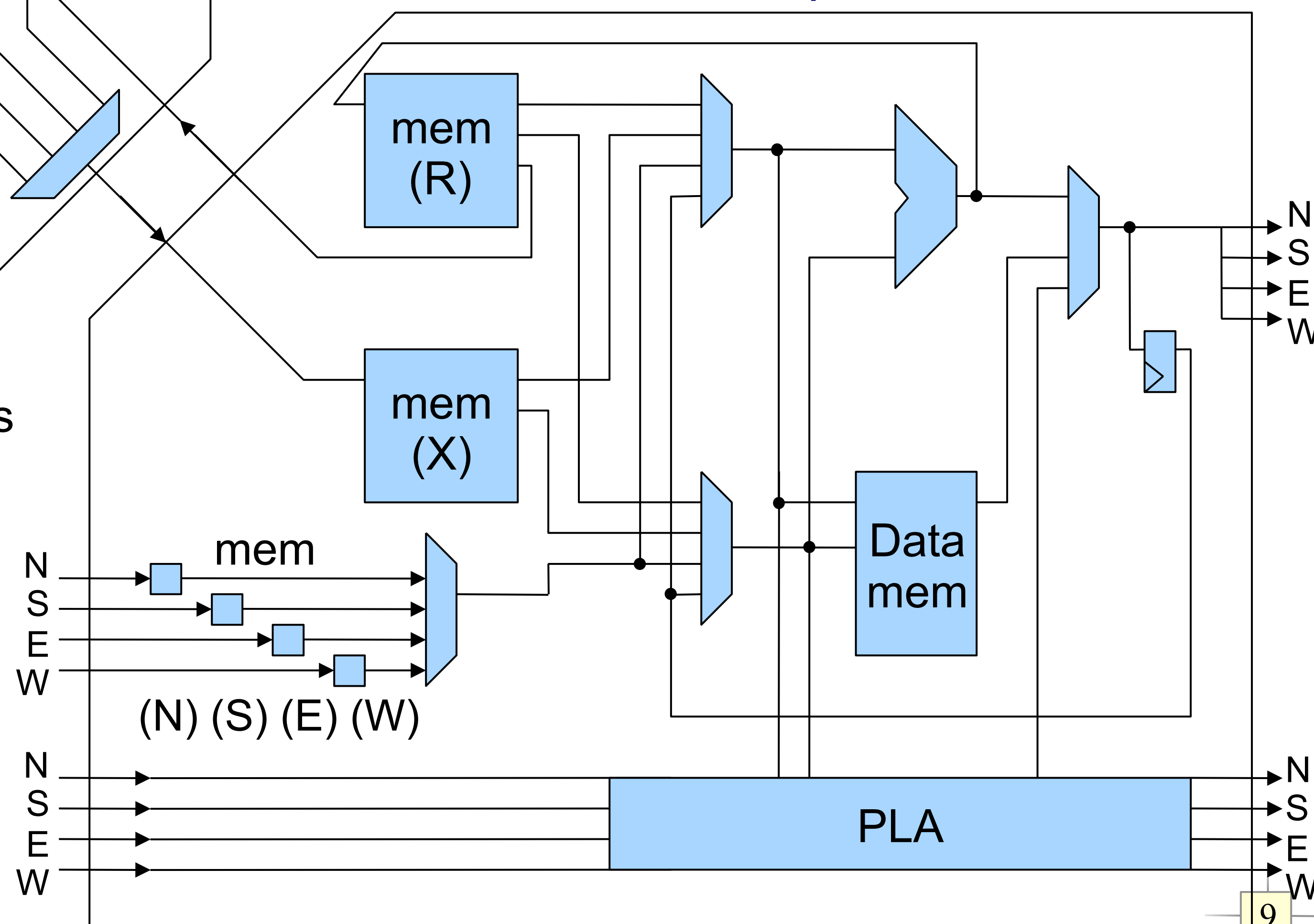



| N | S | E | W | PE | Raddr | Waddr |
|---|---|---|---|---|---|---|
| S | P | | P | | 4 | |
| | E | W | E | | | |
| | | | | N | | 6 |
| ... | | ... | | ... | ... | ... |
| | ... | | ... | | | |

# Architecture – PE Core




- Fixed static schedule
- 1GHz+ target
- Time-mux ALU, data mem
- PLA for bitops



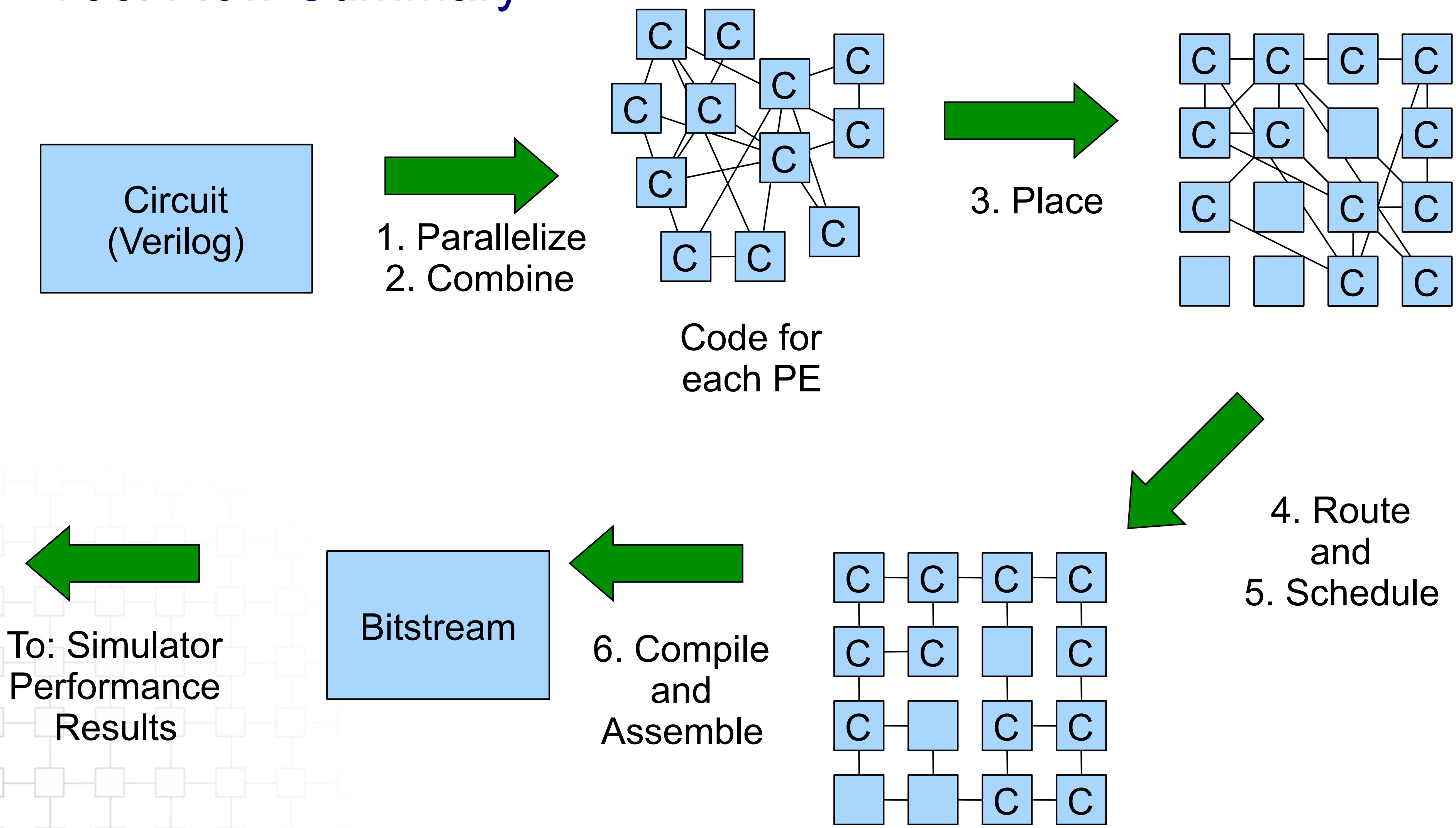

- Direct neighbour connections
- Several memories:
  - R nodes (16 words)
  - X nodes (16 words)
  - N,S,E,W (4 words ea)
  - User Data (2k words)

# Overview

- Introduction and Motivation
- MPPA Architecture
- **CAD Tools**
- Results

# Tools

- Tool Flow Summary

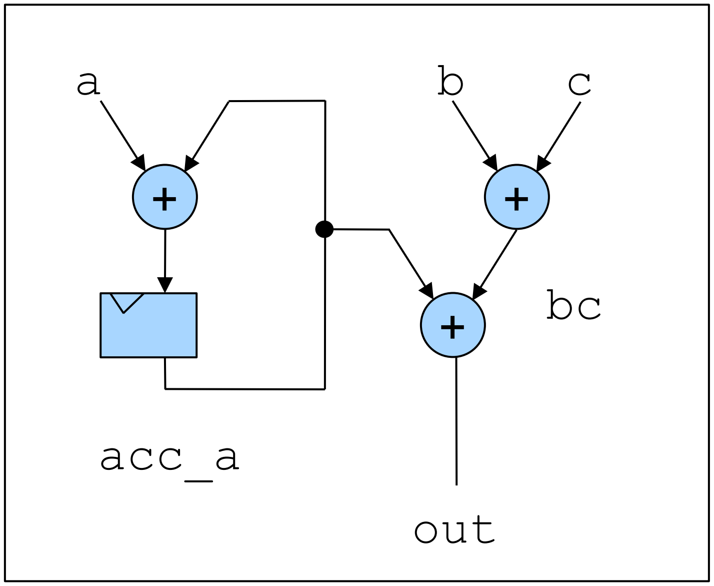


# Tools

- Example
  - Verilog and circuit

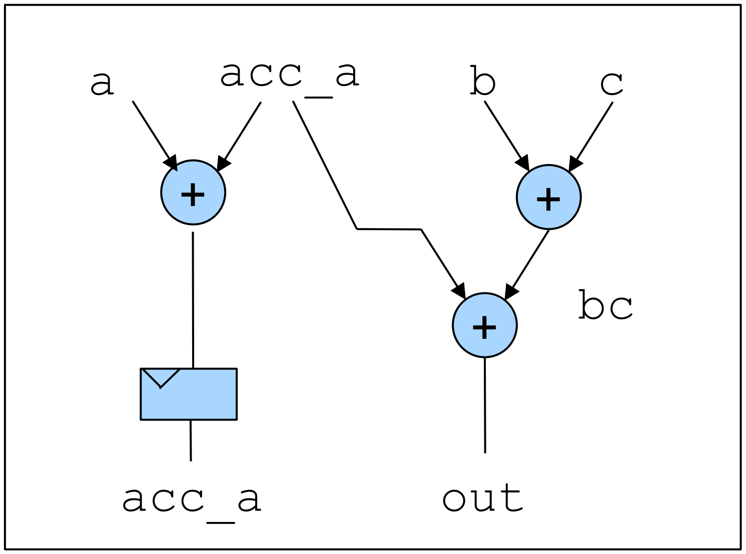
```
always @(posedge clk) begin
    acc_a <= acc_a + a;
end

assign bc = b + c;
assign out = bc + acc_a;
```



# Tools

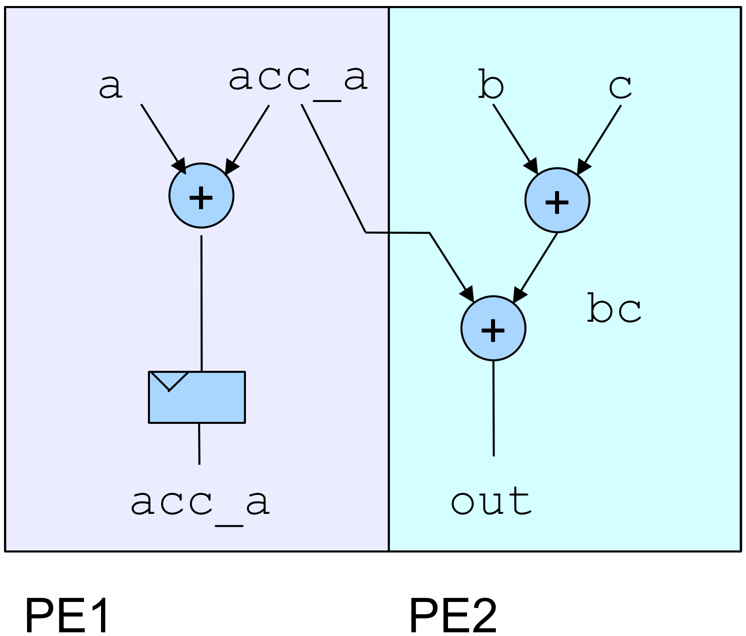
- Parallelize
  - Find parallelism
  - Use Verilator
    - Parse Verilog
    - Construct DFG
    - Optimize
  - Map larger memories into multiple PEs (future work)
  - Example
    - Trivial here, harder with memories and sequential code



# Tools

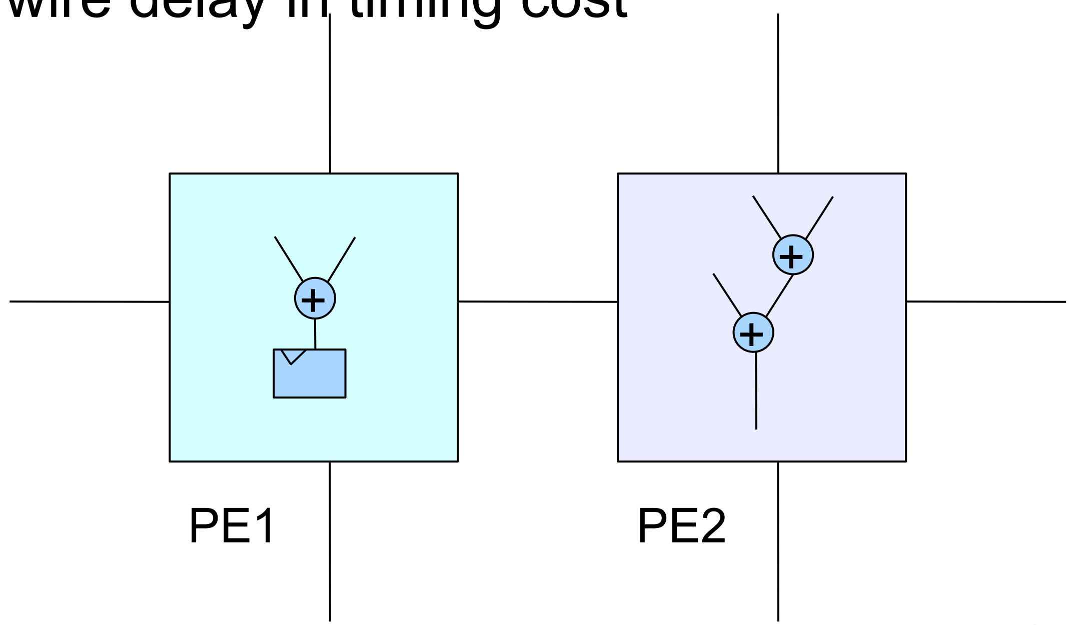
- Combine
  - Cluster graph into PE-sized code segments
  - Minimize sum of communication widths
  - Use hMETIS

  - Example
    - Cluster into 2 PEs



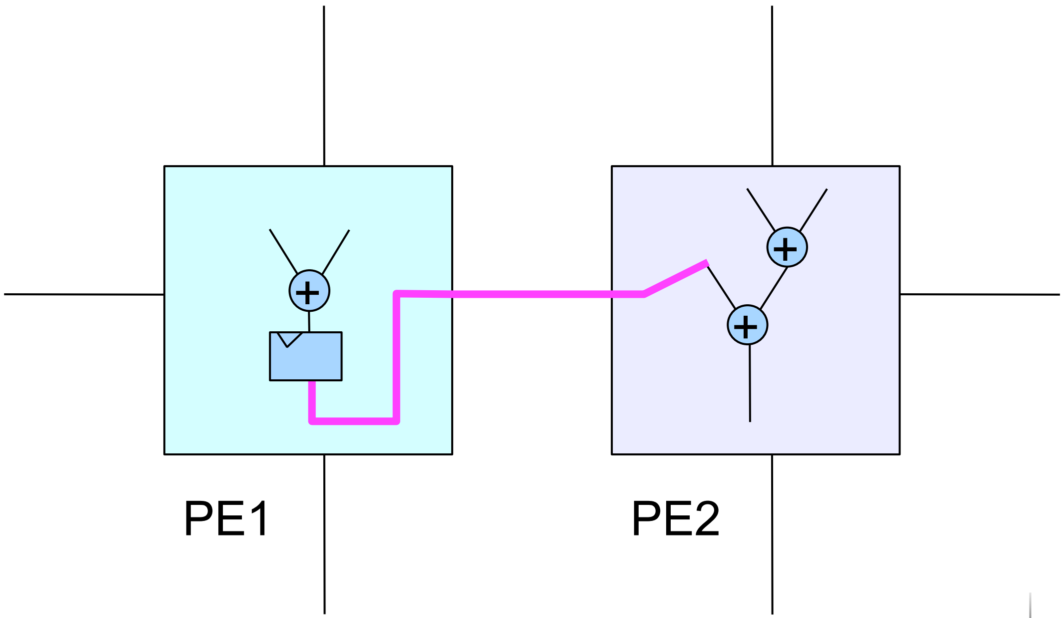
# Tools

- Placement
  - Assign PE code to physical PEs
  - Minimize critical path
  - Place one-bit and multi-bit logic separately
  - Use VPR's timing driven placement
    - Manhattan distance not wire delay in timing cost
  - Example
    - Only one data communication



# Tools

- Routing
  - Connect all communicating PEs
  - Time multiplexed but static (deterministic)
  - Horizontal-then-vertical routing strategy (O(n))
    - Ignore congestion and conflicts
      - Post-scheduling results show very few routing conflicts
  - Example
    - Single route
    - Needs to cross clock boundary



# Tools

- Schedule
  - Assign code to timeslots in PEs (same with route hops)
    - Timeslot oriented algorithm
      - Fast (O(n))
      - Always makes forward progress
  - PE router conflicts
    - Delay data in router 1+ cycles using “hold slots”
  - Node memory implements user registers and wires
    - Wires → memory read after write (RAW)
    - Registers → memory write after read (WAR)

# Tools

- Example – final code / route schedule

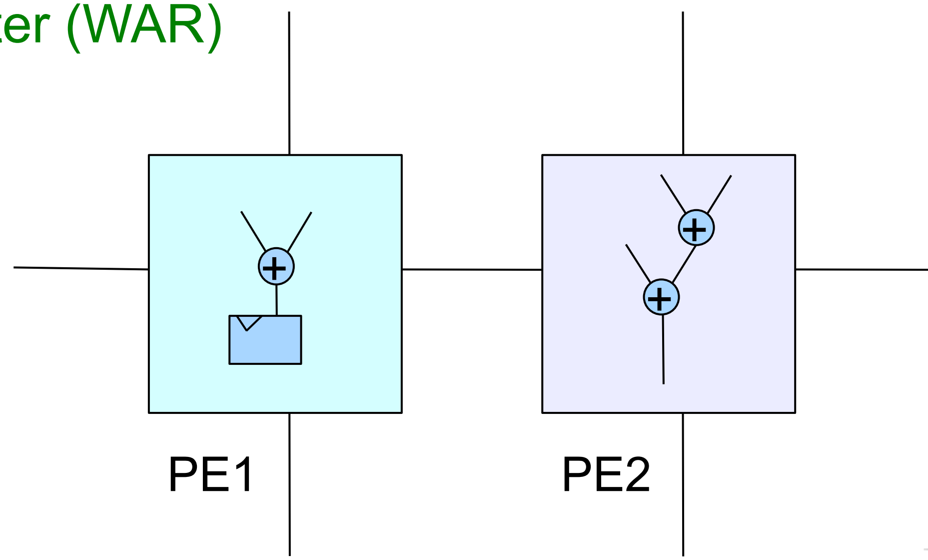
R1 is a wire (RAW)

| Time | PE1 core | PE1 router | PE2 core | PE2 router |
|---|---|---|---|---|
| 0 | IOLOAD R1 | R2 -> E | IOLOAD R1 | |
| 1 | ADD R2,R1,R2 | | IOLOAD R2 | W -> X0 |
| 2 | NOP | | ADD R1,R1,R2 | |
| 3 | NOP | | ADD R1,R1,X0 | |
| 4 | NOP | | IOSTORE R1 | |

R2 is a register (WAR)

```
always @(posedge clk) begin
    acc_a <= acc_a + a;
end

assign bc = b + c;
assign out = bc + acc_a;
```



# Overview

- Introduction and Motiviation
- MPPA Architecture
- CAD Tools
- **Results**

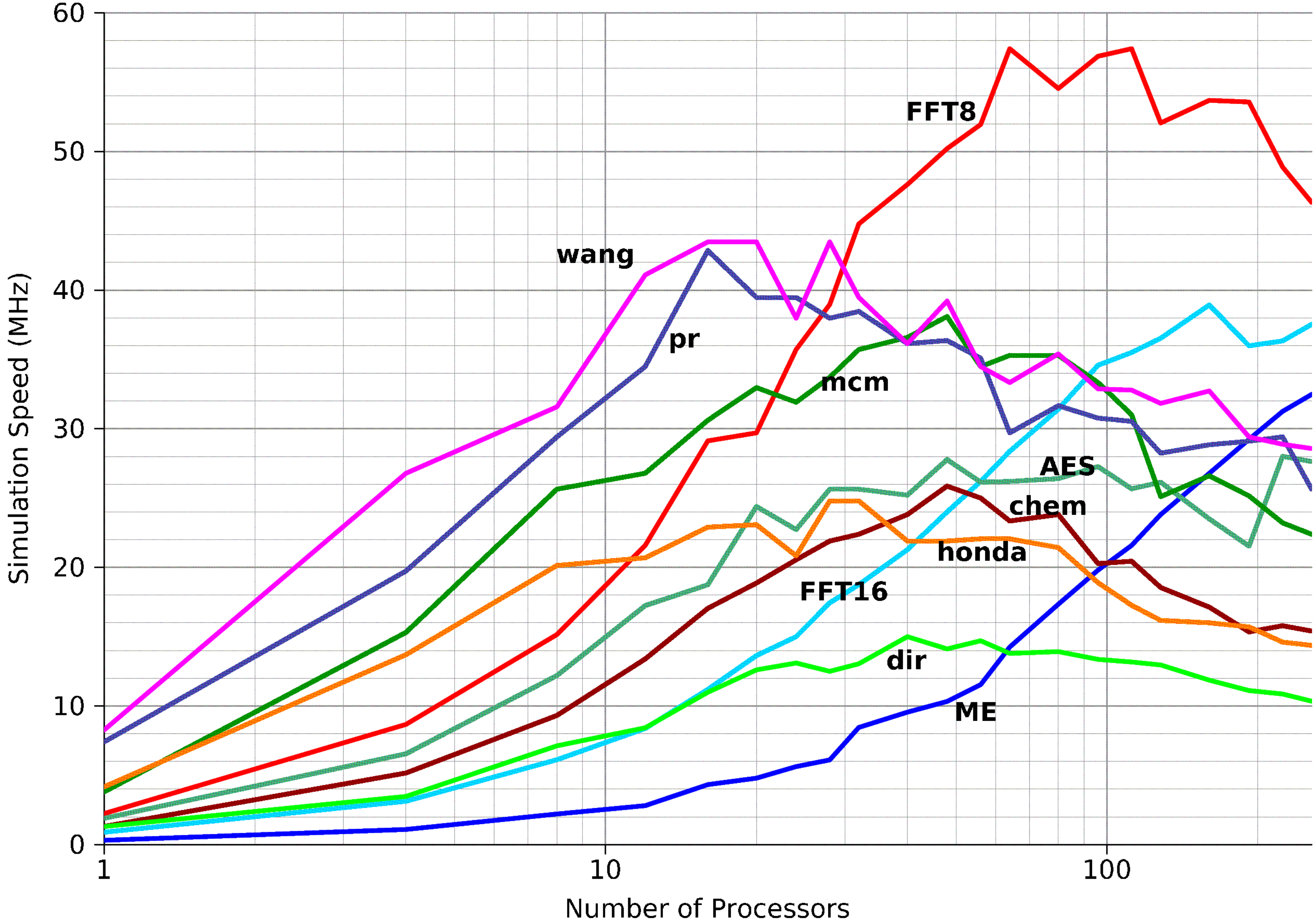
# Results

- Compile Time (seconds)

| Circuit | Verilator | Modelsim | Quartus | RVE |
|---|---|---|---|---|
| AES | 3 | 1 | 148 | 4 |
| pr | 3 | 1 | 228 | 2 |
| wang | 3 | 1 | 182 | 2 |
| honda | 3 | 1 | 202 | 2 |
| mcm | 3 | 1 | 219 | 2 |
| dir | 3 | 1 | 372 | 5 |
| FFT8 | 3 | 1 | 207 | 4 |
| chem | 4 | 1 | 477 | 3 |
| ME | 3 | 1 | 277 | 27 |
| FFT16 | 3 | 1 | 790 | 8 |
| **Geo. Mean** | **3** | **1** | **272** | **3.9** |

# Results

- Simulation speed for 1-256 processors

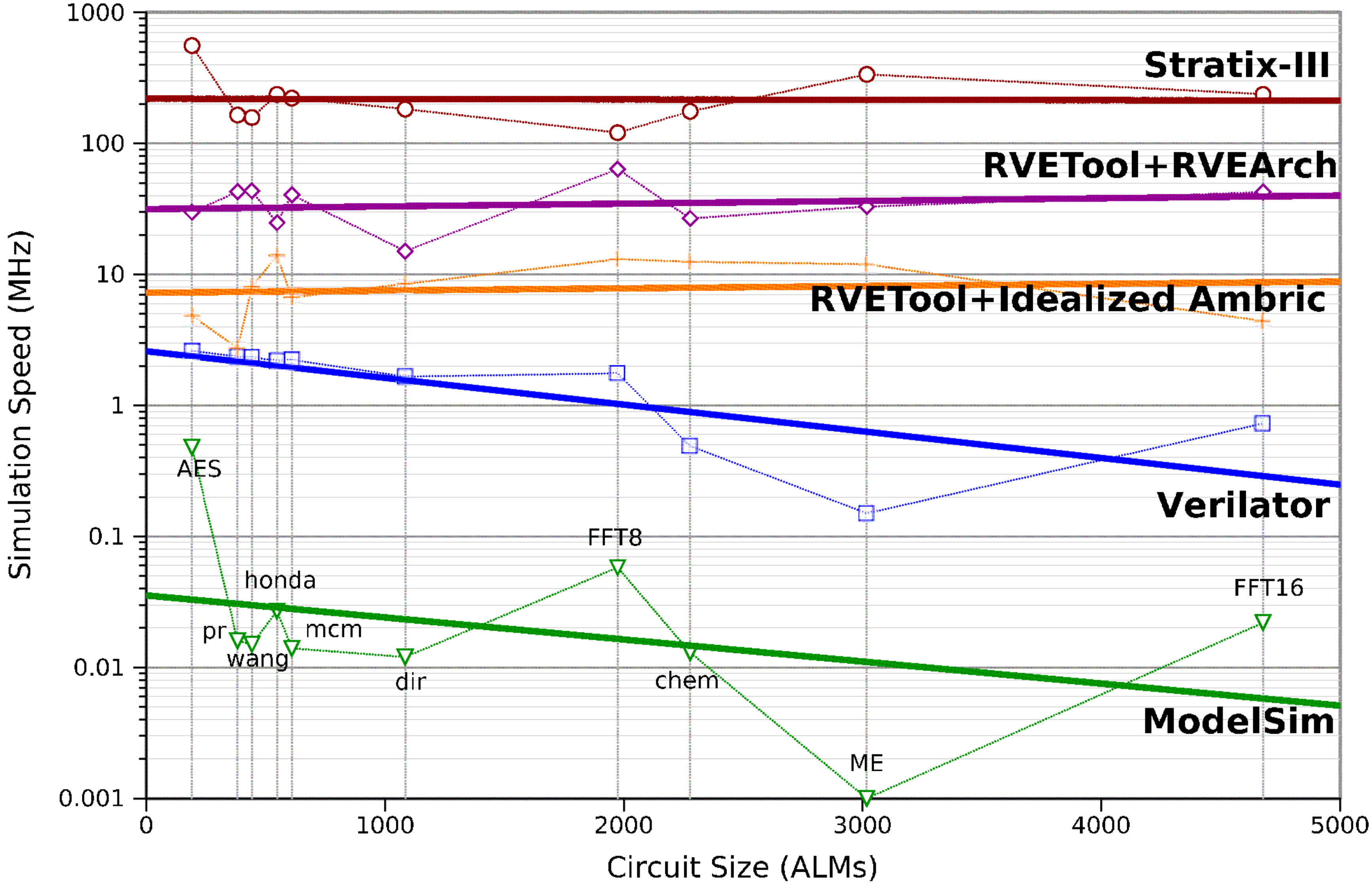


# Results

- Simulation speed for 1-256 processors
  - Larger circuits peak later
    - More parallelism available (bigger circuit)
    - Some slowdown from distant communication
    - Scalable to 10,000+ PEs ?

# Results

- **Best** Simulation Speed Comparison



# Results

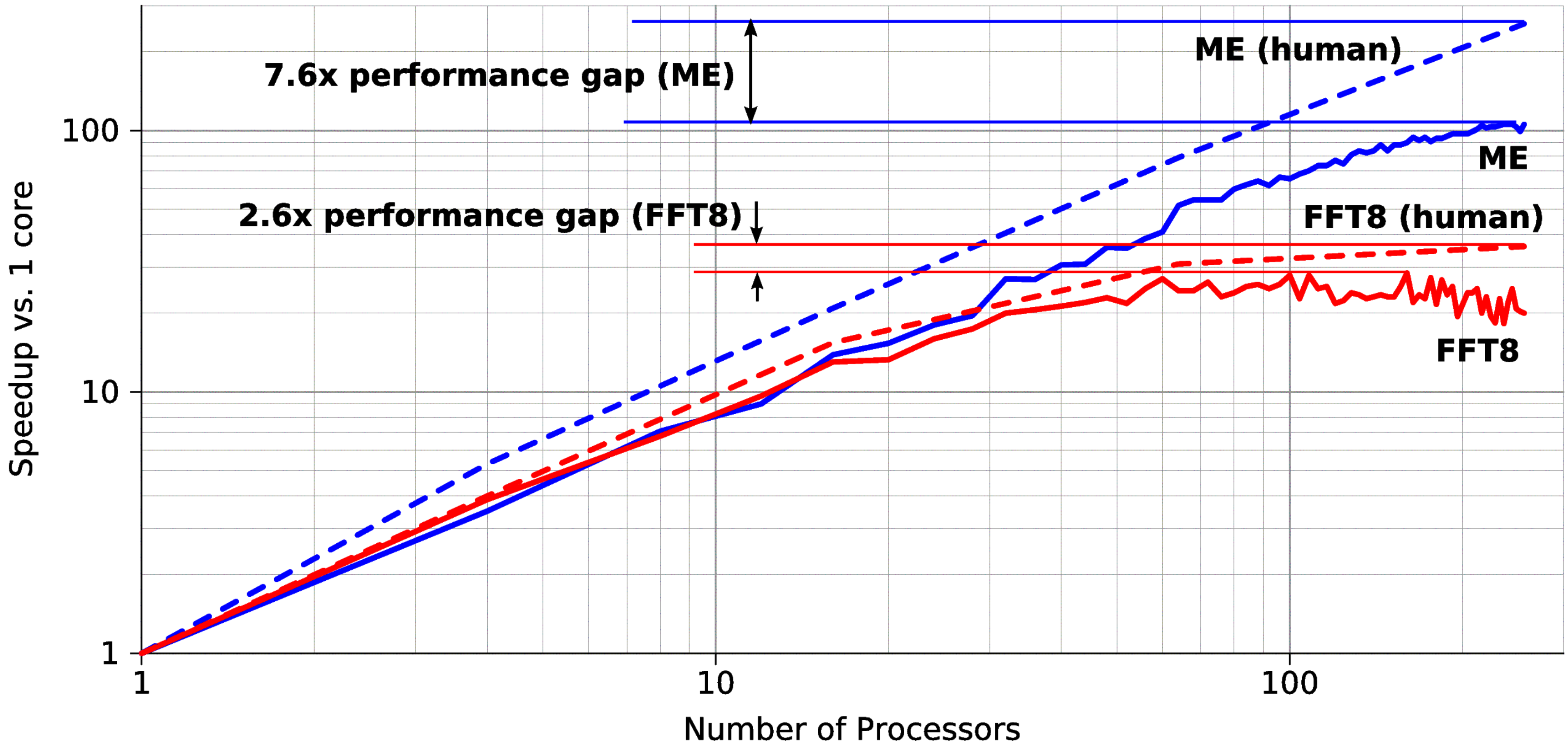
- **Best** Simulation Speed Comparison
  - Speed dependence on circuit size
    - Stratix III and RVETool maintain simulation speed
    - Software simulators get slower
      - Only one processor
  - At best simulation speed
    - RVEArch has 1/4 density of Stratix III @ 1/7$^{th}$ the speed
    - RVEArch can do space/time folding...

# Results

- Space/time folding
  - Best “unfolding” speed
    - RVEARch vs Stratix III
    - 1/4 density @ 1/7$^{th}$ the speed
  - After “folding”
    - 1x density @ 1/12$^{th}$ speed
    - 10x density @ 1/50$^{th}$ speed
      - Still over 30x faster than software simulators
      - Software speed degrades more quickly with bigger circuits

# Results

- Hand-mapped Speedup Gap
  - How good are the tools? (assume: human knows best)



# Status / To-Do

- Current status
  - The tools work
  - Trying to improve Fmax (cluster+place+route+schedule)
  - Adding 1-bit signals (using VPR as a subroutine)
- TODO: Need Verilog Benchmarks
  - The bigger, the better!
  - Right mix of word-wide, 1-bit signals
- TODO: Architectural experiments
  - Focus is on tools right now
  - Good tools will let us explore architecture design space

# Summary

- Main Idea
  - Take a computational circuit and synthesize/simulate it fast
- Custom Architecture
  - Time-multiplexed MPPA
  - High bandwidth, low latency communication
- Fast Tools
  - Behavioural synthesis, word-level
  - Coarse grained architecture
- To Do
  - Lots. But it works so far!

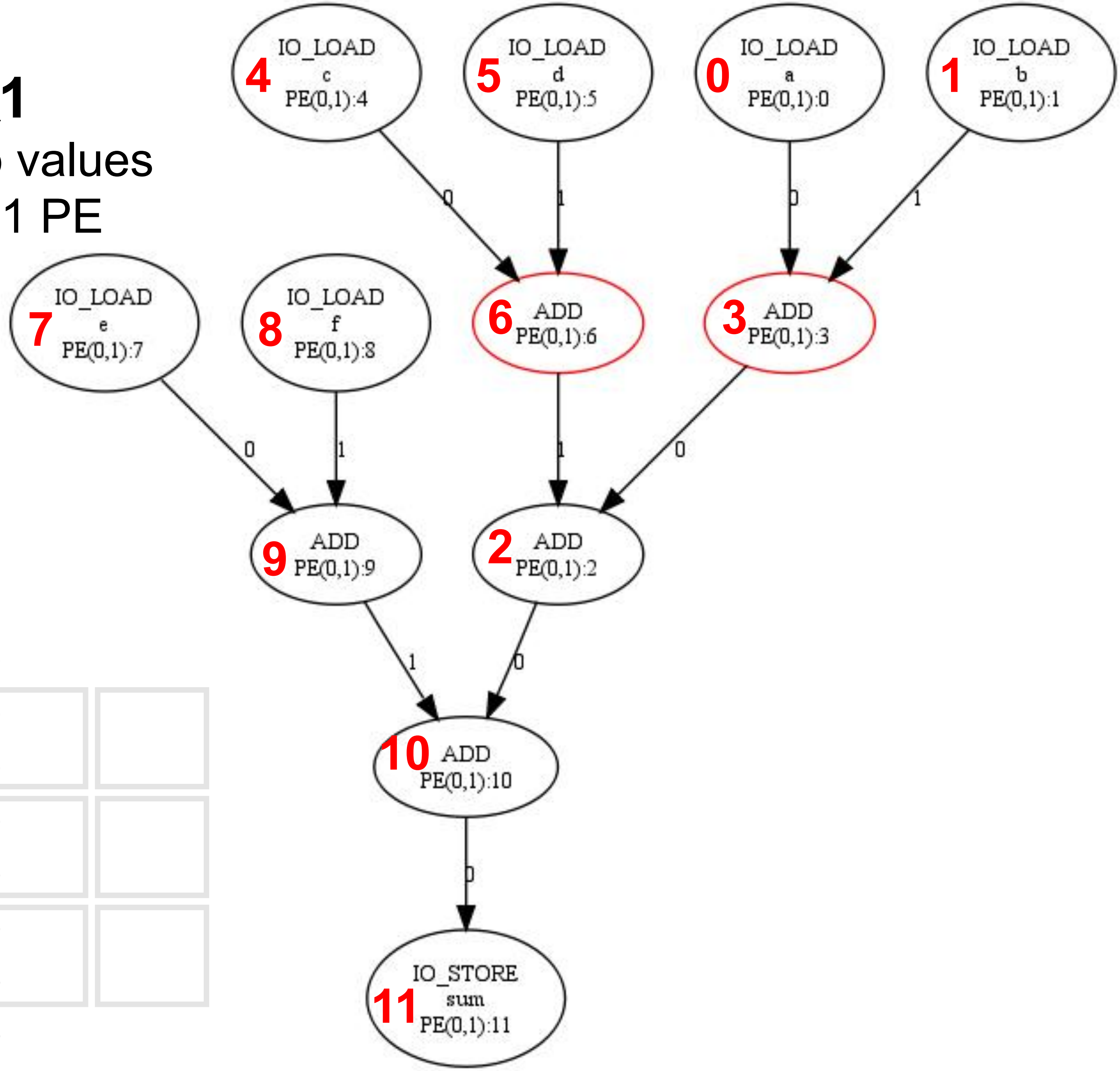
# EOF

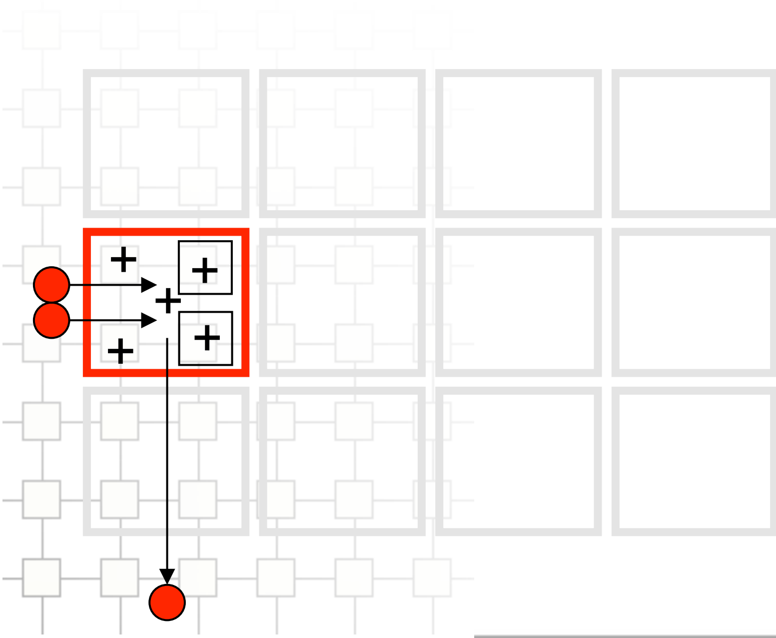
+++NO CARRIER

# Example 1

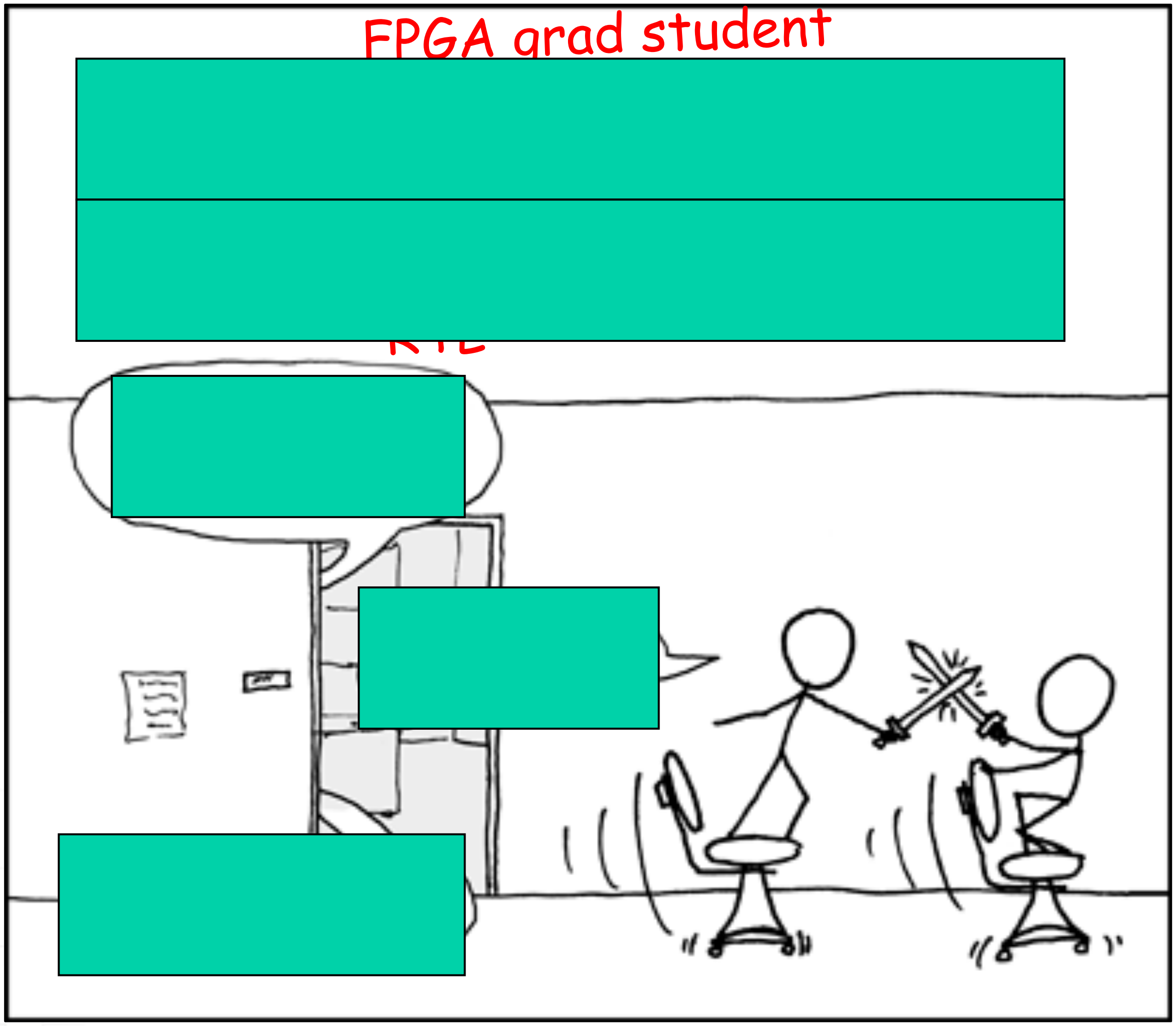
Add 6 x 32b values
Execute on 1 PE

Total of 12 clock cycles



# Motivation




http://xkcd.com/303/

# Goals

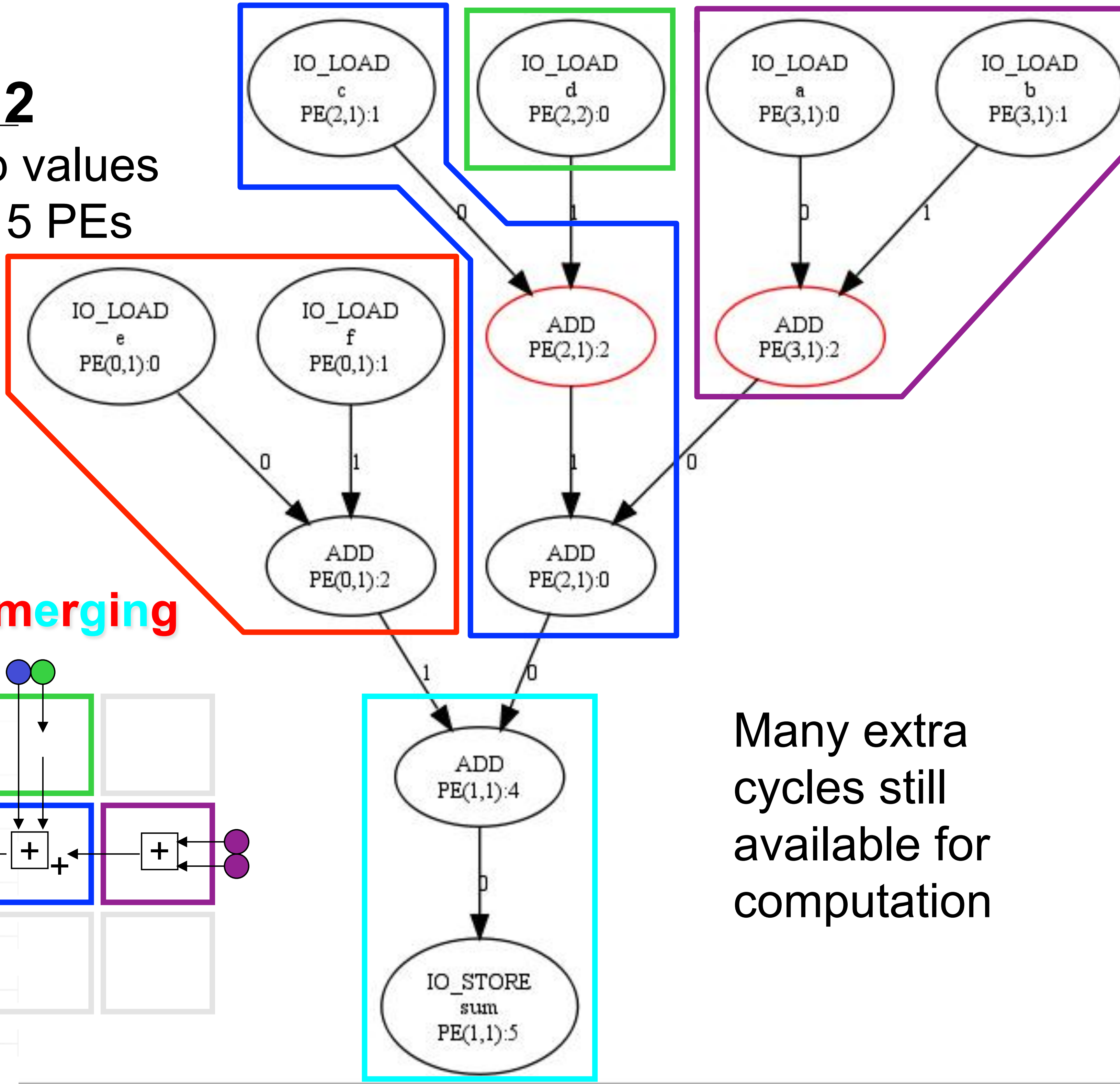
- This work is really about **productivity improvements**
- 10x faster than FPGA CAD tools
  - Minutes, not hours, for large design
  - Compile+Place+Route runtime ~ compiler
- 10x capacity of an FPGA
  - Gracefully increase capacity at expense of speed
- No less than 1/10th the speed of FPGA
  - Slower only at "full capacity"
  - As fast as FPGA at "low capacity"

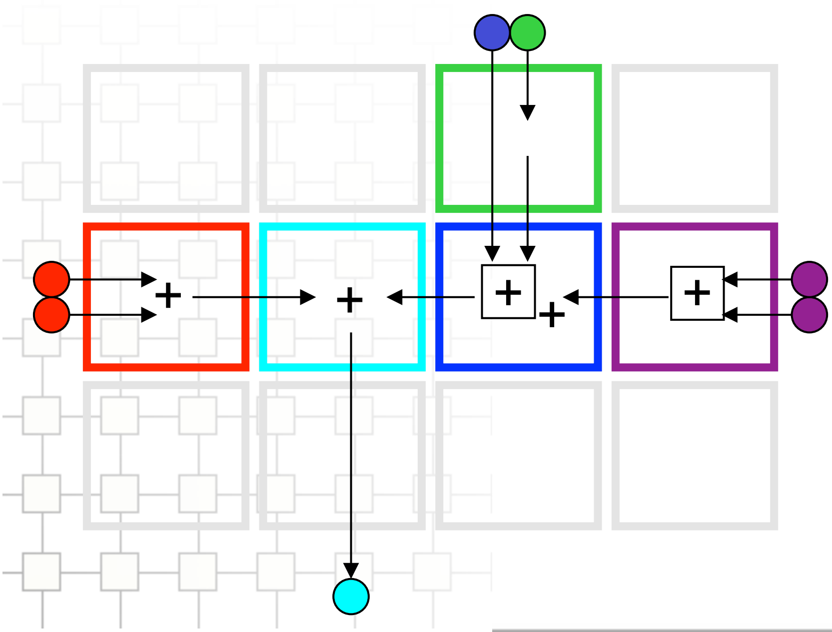
# Example 2

Add 6 x 32b values
Execute on 5 PEs

Total of 6 clock cycles

Minimum is 5 cycles by **merging**



Many extra cycles still available for computation

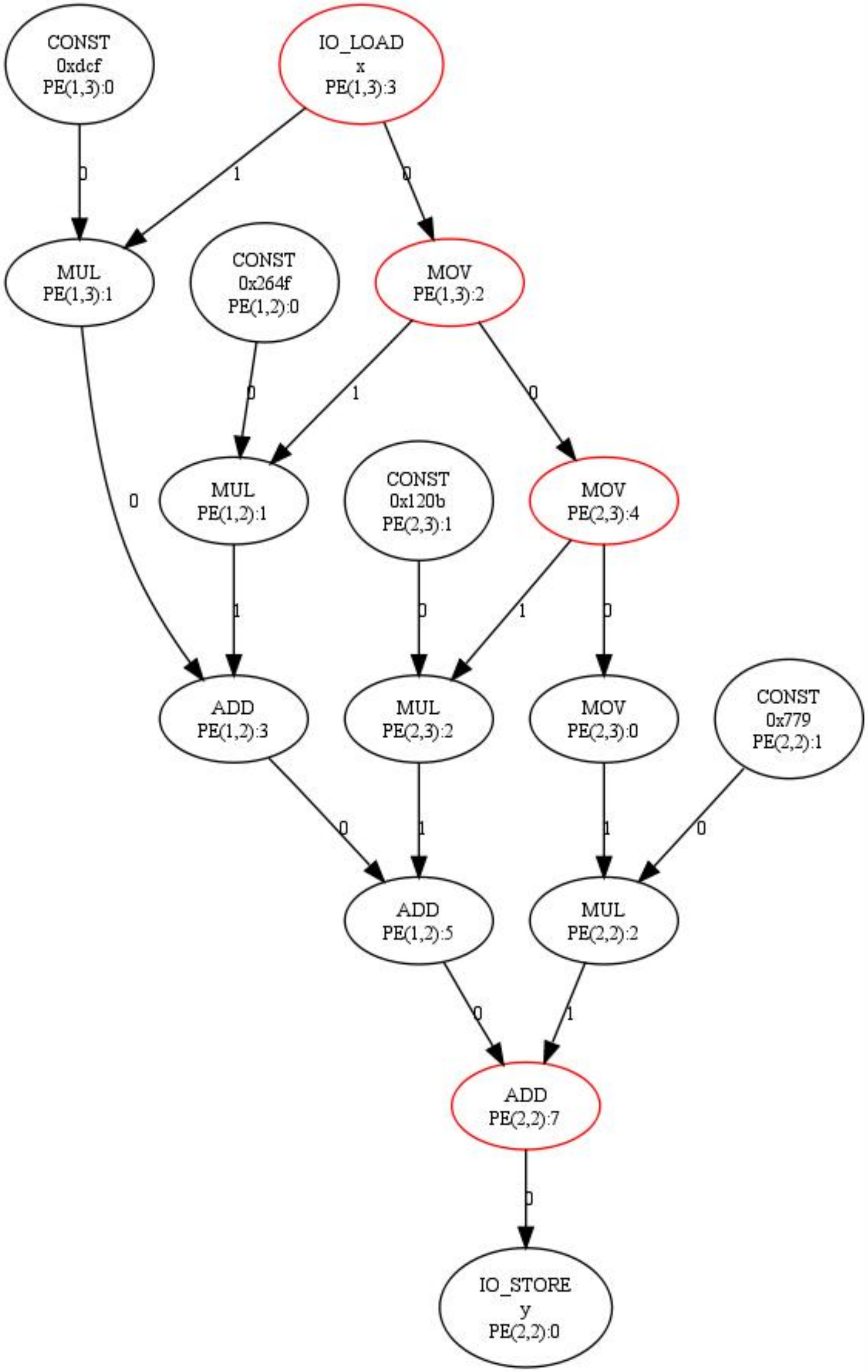
# Example 3

4-tap FIR filter (32b)

Filter is ***not*** pipelined

Executes in 8 cycles

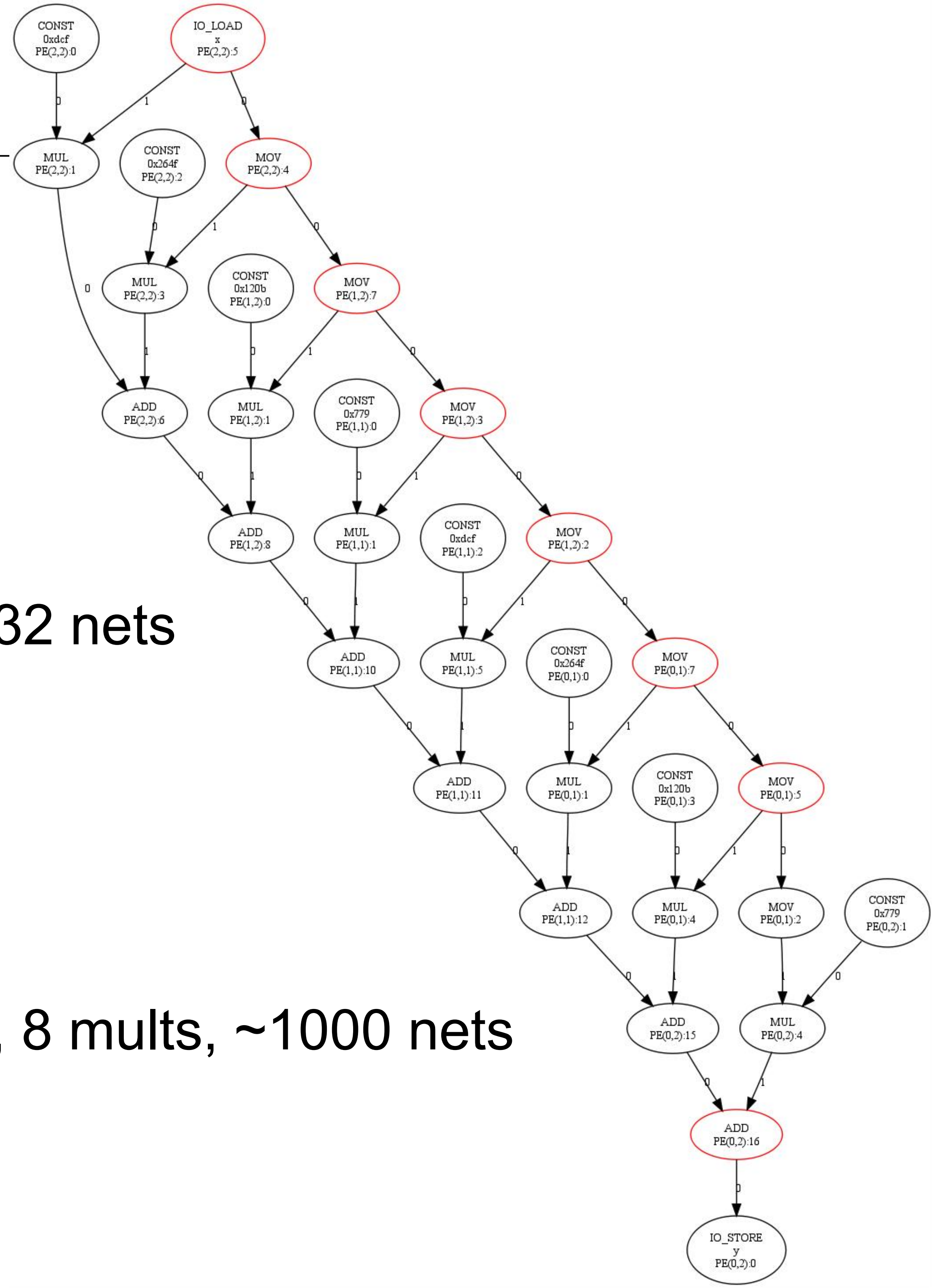
On 4 PEs



# Example 4

## 8-tap FIR filter (32b)

Filter is ***not*** pipelined




Executes in 17 cycles

On 5 PEs

- Place & route ~5 PEs, ~32 nets

Traditional FPGA:

- 7 x 32b adds
- 8 x 32b mults
- 8 x 32b registers
- Place & route ~32 LABs, 8 mults, ~1000 nets

# Overview

- Introduction and Motivation
- MPPA Architecture
- CAD Tools
- Examples
- **Results**

# Overview

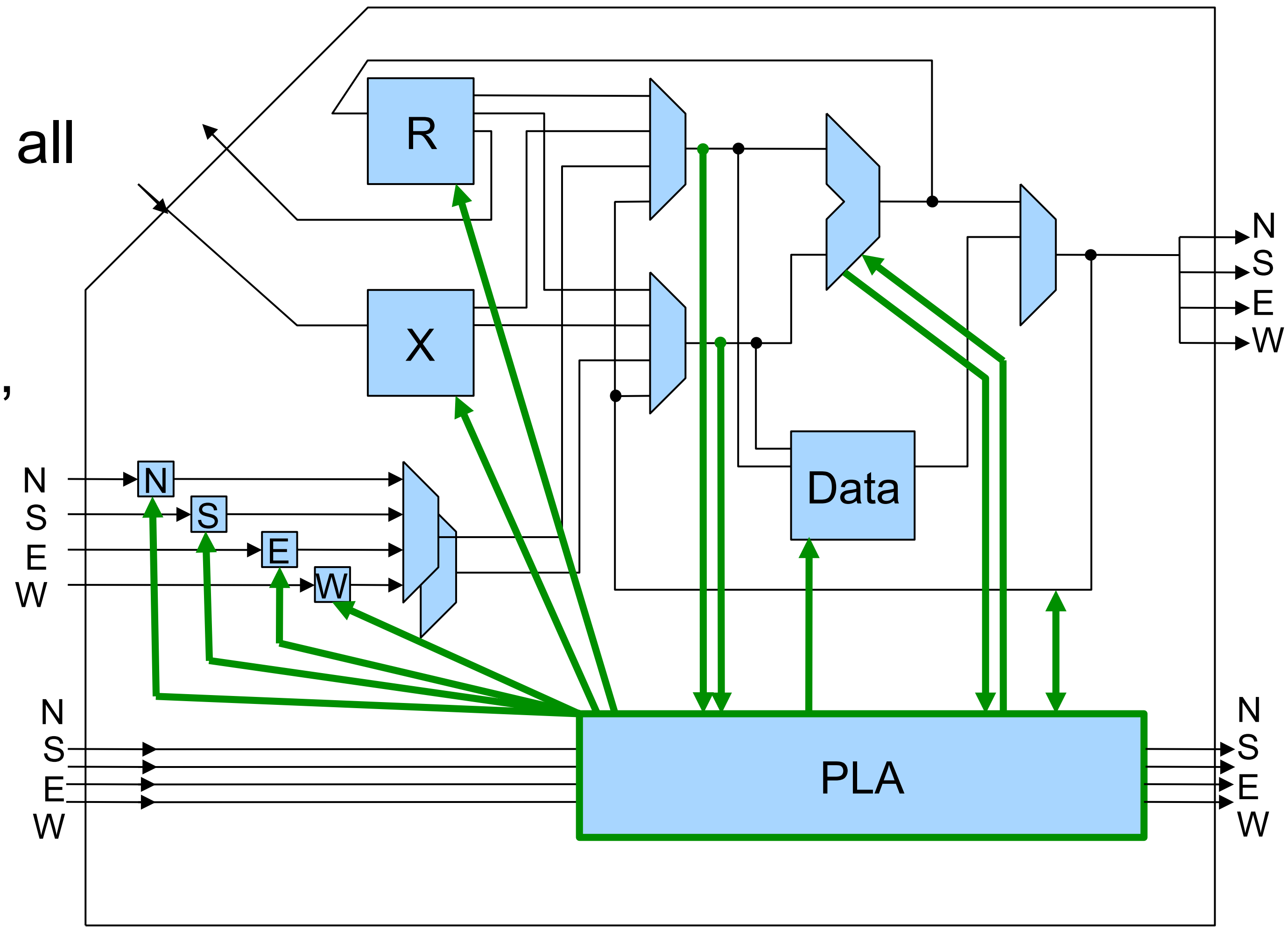
- Introduction
- Architecture
- Tools
- Examples
- Results
- **Bit-level Signals**

# Bit-level Signals

- Problem
  - Compute+communicate 1-bit data on 32-bit datapath
  - Frequently, 1-bit is control & has large fanout
- Idea
  - Separate 1-bit signals and implement in a PLA
    - Mostly control signals
  - Keep word-oriented signals in 32-bit datapath

# Bit-level Signals

- Bit-level PLA signals
  - Write-enable to all memories
  - ALU control (eg: mux select, +/-, ...)
  - ALU flags
  - ALU operands, ALU result



# Bit-level Signals

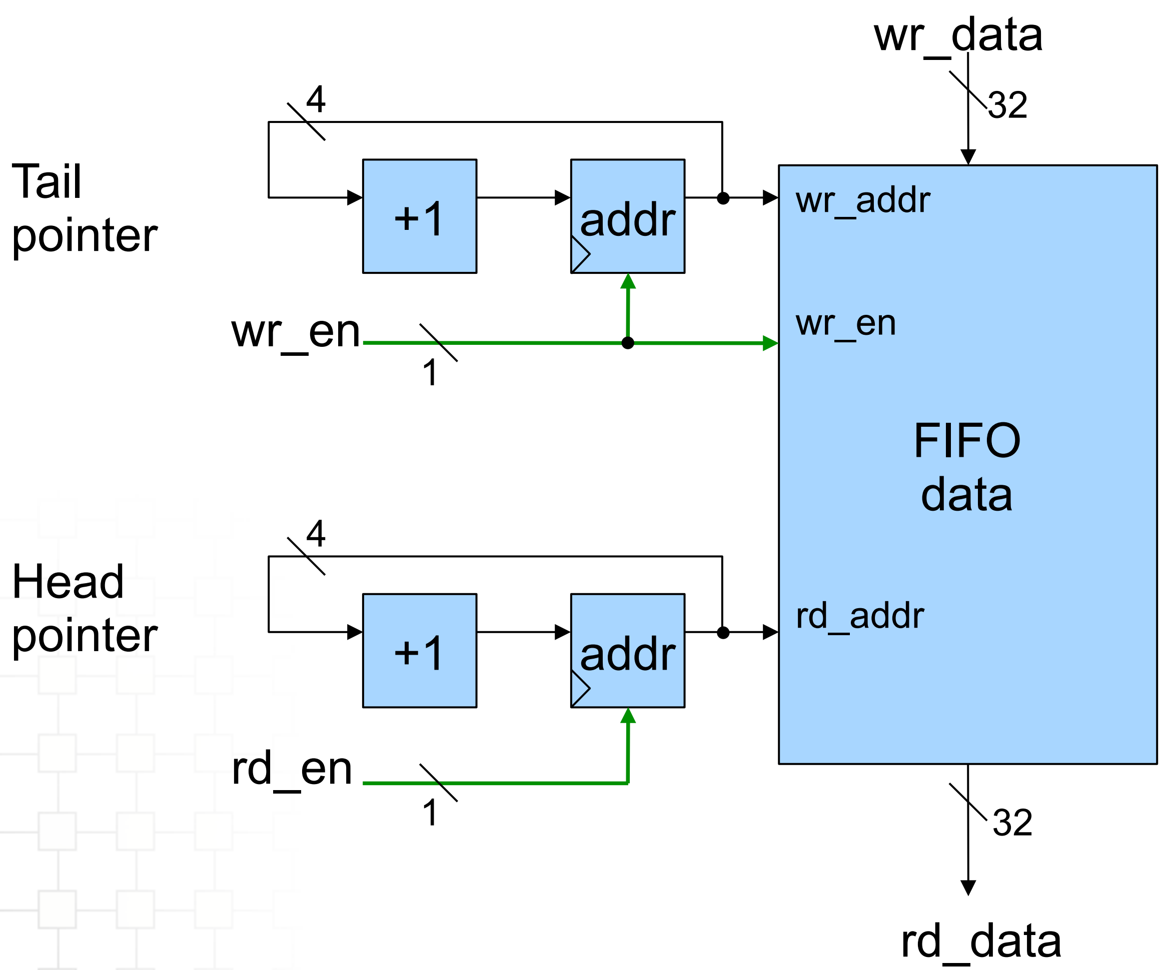
- Problem
  - Compute+communicate 1-bit data on 32-bit datapath
  - Frequently, 1-bit is control & has large fanout
- PLA and bit-oriented interconnect
  - Compute + distribute 1-bit results for control logic
  - Like regular FPGA fabric: not multiplexed, not pipelined
    - Fast, no pipeline delays
    - Avoids control signals becoming timing-critical
  - Timing analyzer determines arrival time
    - Knows when to schedule dependant word operations

# Bit-level Signals

- Two Examples
  - Local: FIFO
    - Bit-level signal stays within PE
  - Global: Deeply pipelined string matching
    - Bit-level signal distributed to many PEs

# Bit-level Signals

- Local example: FIFO



# Bit-level Signals

- Local example: FIFO
  - Single PE code **without** PLA

| Time | PE | Notes |
|---|---|---|
| 0 | `IOLOAD wr_en` | |
| 1 | **`CMP wr_en, 0`** | 32-bit comparison |
| 2 | `IOLOAD wr_data` | |
| 3 | `STORE.CMP [wr_data], wr_data` | Predicated store |
| 4 | `ADDI.CMP wr_addr, wr_addr, 1` | |
| 6 | `IOLOAD rd_en` | |
| 7 | **`CMP rd_en, 0`** | 32-bit comparison |
| 8 | `LOAD.CMP rd_data, [rd_addr]` | |
| 9 | `IOSTORE rd_data` | |
| 10 | `ADDI.CMP rd_addr, rd_addr, 1` | |

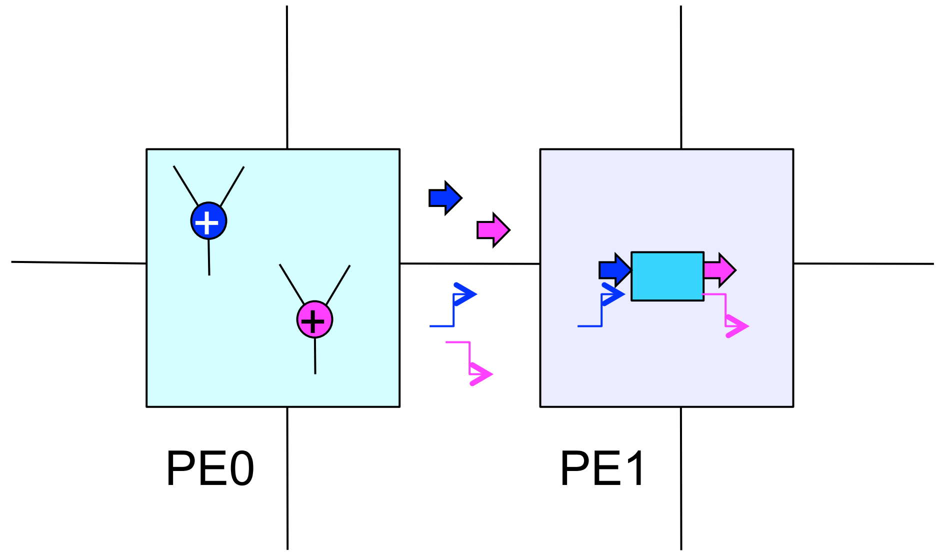
# Bit-level Signals

- Local example: FIFO
  - Single PE code with PLA
    - PLA needs small amount of logic, flip-flops
    - Instructions predicated on PLA bits
    - Reduces schedule length

| Time | PE | Notes |
|---|---|---|
| 0 | `IOLOAD wr_data` | |
| 1 | `STORE.PLA.0 [wr_addr], wr_data` | `PLA.0 holds wr_en` |
| 2 | `ADDI.PLA.0 wr_addr, wr_addr, 1` | |
| 3 | `LOAD.PLA.1 rd_data, [rd_addr]` | `PLA.1 holds rd_en` |
| 4 | `IOSTORE rd_data` | |
| 5 | `ADDI.PLA.1 rd_addr, rd_addr, 1` | |

# Bit-level Signals

- Local example: FIFO
  - Two PE code with PLA
    - PLA in PE1 gets signals from PLA in PE0

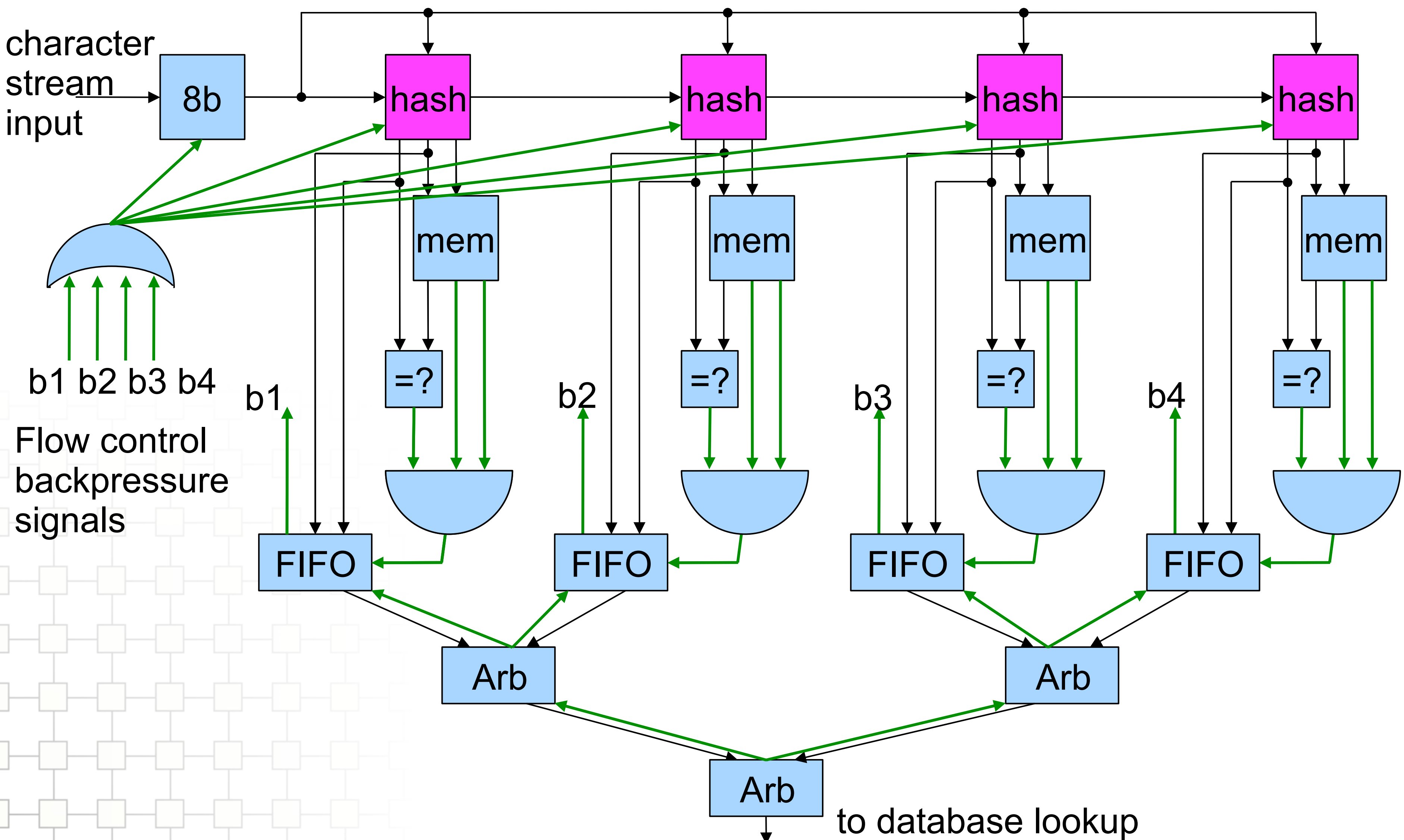



| Time | PE0 (address generator) core | PE0 router | PE1 (data storage) core | PE1 router |
|---|---|---|---|---|
| 0 | | | IOLOAD R1 | |
| 1 | | | STORE.PLA.0W [W0],R1 | |
| 2 | ADDI.PLA.0 R1,R1,1 | ADDI->E0 | LOAD.PLA.1W R2,[W1] | |
| 3 | ADDI.PLA.1 R2,R2,1 | ADDI->E1 | IOSTORE R2 | |

R2 == rd_data

# Bit-level Signals

- Global example: Deeply pipelined string matching



# Bit-level Signals

- Example: Deeply pipelined string matching
  - 1-bit back-pressure to all “hash engines”
  - Everything is feed forward, except back-pressure
    - Excellent for parallelizing/pipelining
    - Back-pressure stalls all engines
  - Use PLA to compute + distribute back-pressure

# Bit-level Signals

- Clustering with Bit-level Signals
  - Two main alternatives (several variations)
    - Which alternative is the best?

1. Pre-cluster 1-bit logic…
    - Remove all datapath signals (> 4 bits) and nodes
    - Cluster 1-bit logic into distinct PEs
      - Separate based on connected components?
      - Break apart components that are too large?
      - Cluster together components that are too small?
    - Re-insert all datapath signals and nodes
    - Cluster datapath nodes around 1-bit clusters
    - Datapath logic is distributed around control logic

# Bit-level Signals

- Clustering with Bit-level Signals

  2. Pre-cluster datapath logic
     - Reverse order of previous approach
     - Separate into 1-bit and datapath netlists
     - Cluster datapath netlist
     - Cluster 1-bit netlist, constrained by datapath clustering solution
     - Control logic is distributed around datapath

# Status / To-Do

- Current status
  - → The tools work
  - → Trying to improve Fmax (cluster+place+route+schedule)
  - → Adding 1-bit signals (using VPR as a subroutine)
- TODO: Need Verilog Benchmarks
  - The bigger, the better!
  - Right mix of word-wide, 1-bit signals
- TODO: Architectural experiments
  - Focus is on tools right now
  - Good tools will let us explore architecture design space

# Comparison to Previous Work

- Lots of MPPAs, coarse-grain arrays, CMPs, ...
  - Thousands of processors
  - A low-latency, high-bandwidth on-chip network
- What's Different?
  - Compiles Verilog!   (compiler/CAD tools)
    - Operate at behavioural level, leverages coarse-grain ALUs
    - 70x faster than FPGAs
  - Runs Verilog!   (execution model)
    - No C model (no global memory, no conditional branches)
    - Each PE has more VLIW-parallelism than Ambric
    - Bit- and word-level resources
    - Soft capacity limit vs FPGAs

# Summary

- Main Idea
  - Take a computational circuit and synthesize/simulate it fast
- Custom Architecture
  - Time-multiplexed MPPA
  - High bandwidth, low latency communication
- Fast Tools
  - Behavioural synthesis, word-level
  - Coarse grained architecture
- To Do
  - Lots.  But it works so far!

# EOF

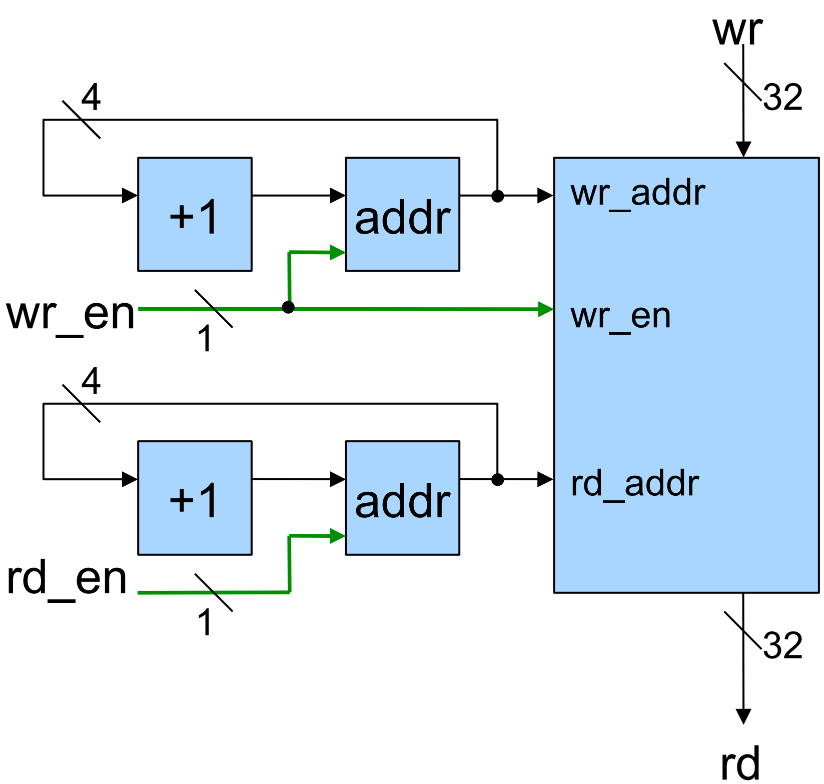
# End

# Bit-level Signals

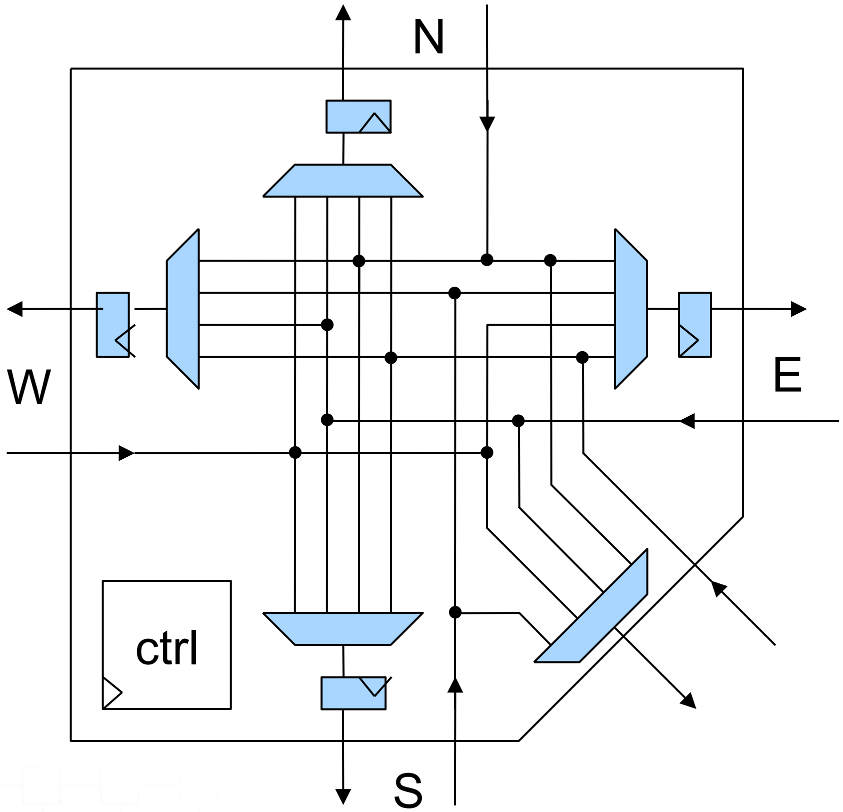
- Example: FIFO

```
always @posedge(clk) begin
    if(wr_en) begin
        mem[wr_addr] <= wr;
        wr_addr <= wr_addr + 1;
    end

    if(rd_en) begin
        rd_addr <= rd_addr + 1;
    end
end
assign rd = mem[rd_addr];
```

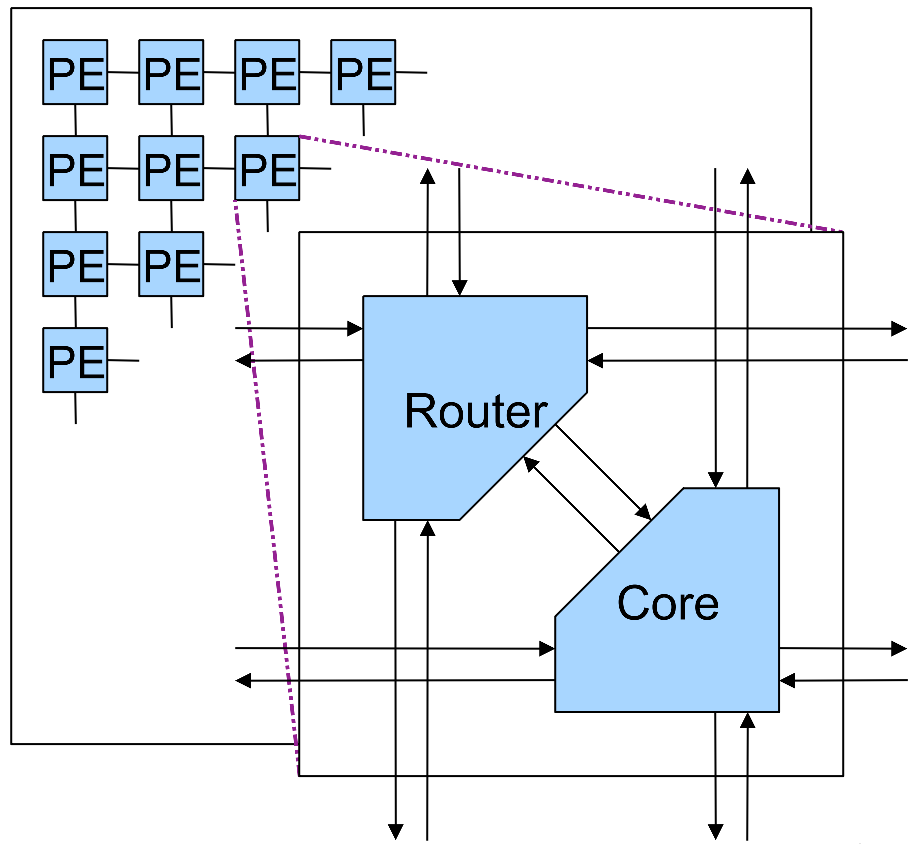


# Architecture – PE Router




| N | S | E | W | PE | Raddr | Waddr |
|---|---|---|---|---|---|---|
| S | P | | P | | 4 | |
| | E | W | E | | | |
| | | | | N | | 6 |
| … | | … | | … | … | … |
| | … | | … | | | |

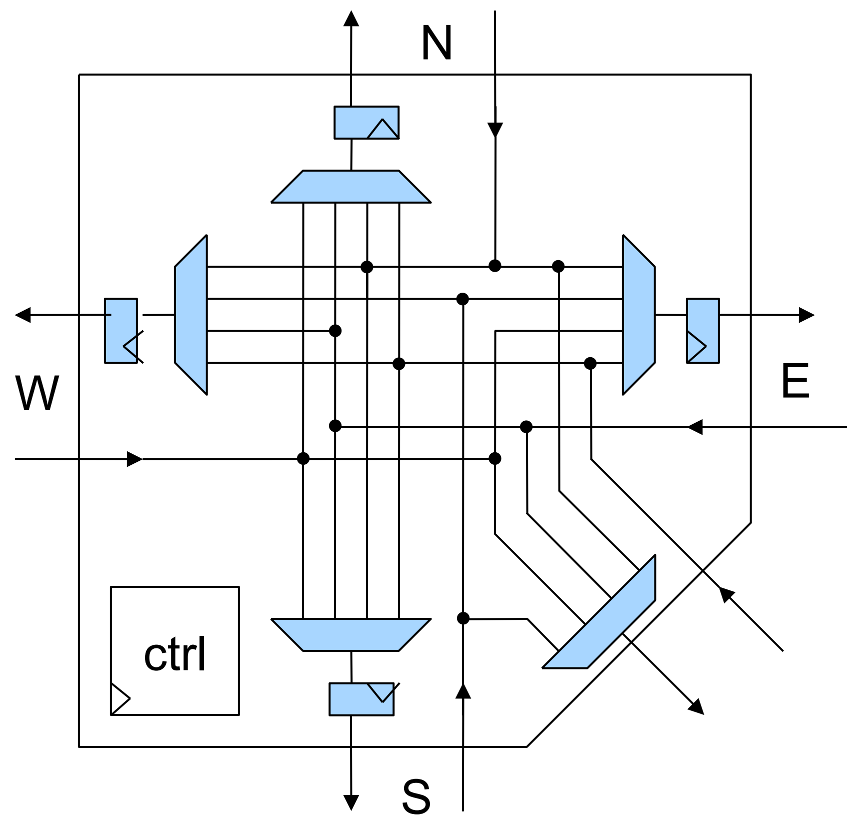
- 5x5 data crossbar
- Fixed static schedule
- Pipelined interconnect
- 1 word/cycle to/from PE



# Architecture

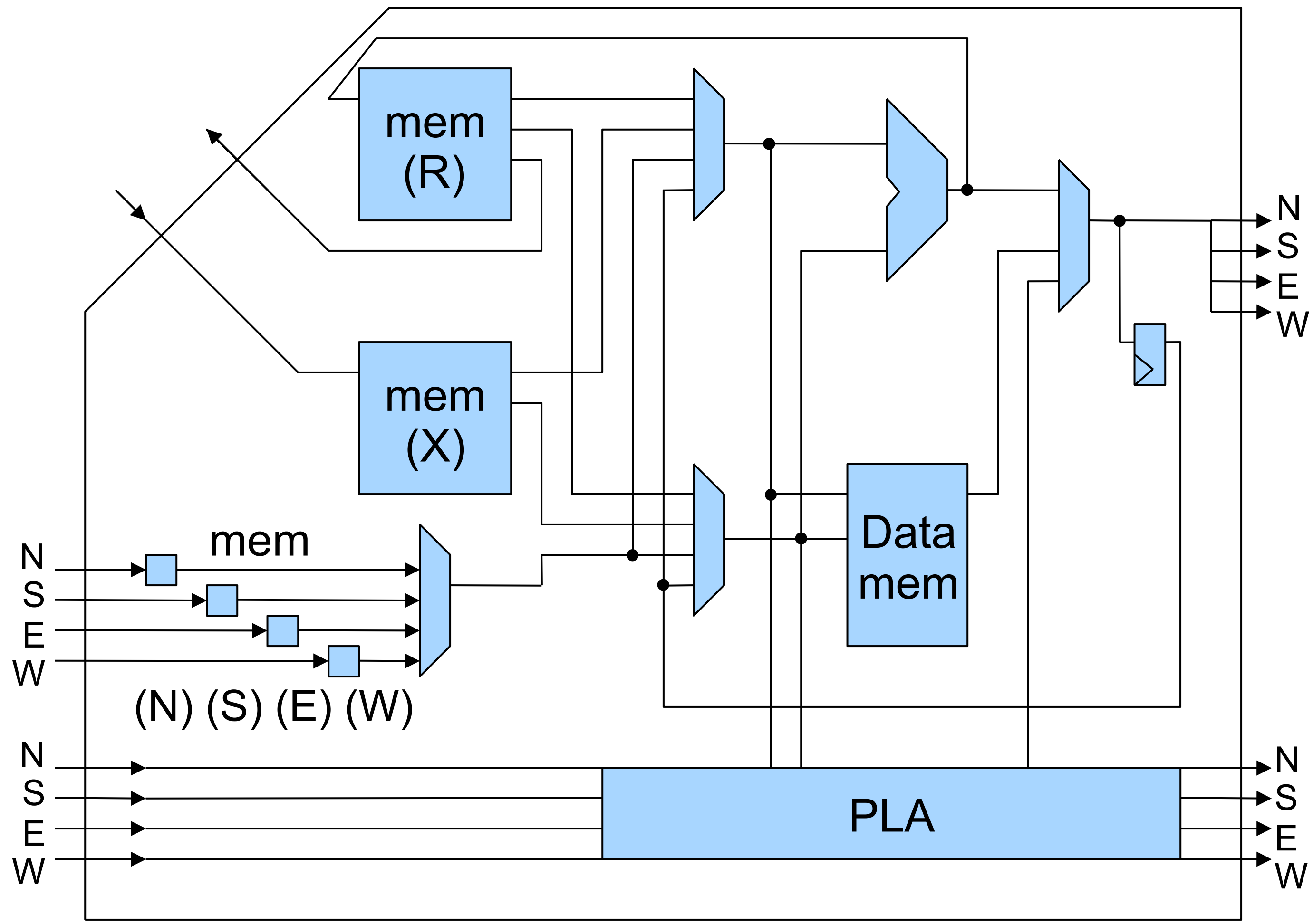
- PE Router
  - 5x5 data crossbar
  - Fixed static schedule
  - Pipelined interconnect
  - Accepts 1 write/cycle

| N | S | E | W | PE | Raddr | Waddr |
|---|---|---|---|---|---|---|
| S | PE | W | PE | | 4 | |
| | | | | N | | 6 |
| ... | ... | ... | ... | ... | ... | ... |



# Architecture

- Fixed static program
- Time-mux ALU
- Direct neighbour connections
- 6 node memories
- Data memory
- PLA for bitops
- 1GHz+ target



# Results

- Space/time folding: Area vs. Performance trade-off
  - FPGA @ 218 MHz

| | | Match FPGA Density | | 10x FPGA Density | |
|---|---|---|---|---|---|
| **Circuit** | **ALMs** | **Req'd PEs** | **Speed (MHz)** | **Req'd PEs** | **Speed** |
| AES | 191 | 4 | 6.6 | 1 | 1.9 |
| pr | 384 | 4 | 19.7 | 1 | 7.4 |
| wang | 442 | 4 | 26.8 | 1 | 8.3 |
| honda | 547 | 8 | 20.1 | 1 | 4.2 |
| mcm | 609 | 8 | 25.6 | 1 | 3.8 |
| dir | 1084 | 12 | 8.4 | 1 | 1.3 |
| FFT8 | 1974 | 28 | 29.0 | 4 | 8.7 |
| chem | 2278 | 28 | 21.9 | 4 | 5.2 |
| ME | 3018 | 36 | 8.4 | 4 | 1.1 |
| FFT16 | 4678 | 60 | 27.9 | 8 | 6.1 |
| Geo.Mean | **954.4** | **12.4** | **17.7** | **1.9** | **3.9** |