



Computational Circuits in a Time-Multiplexed Processor Array

David Grant

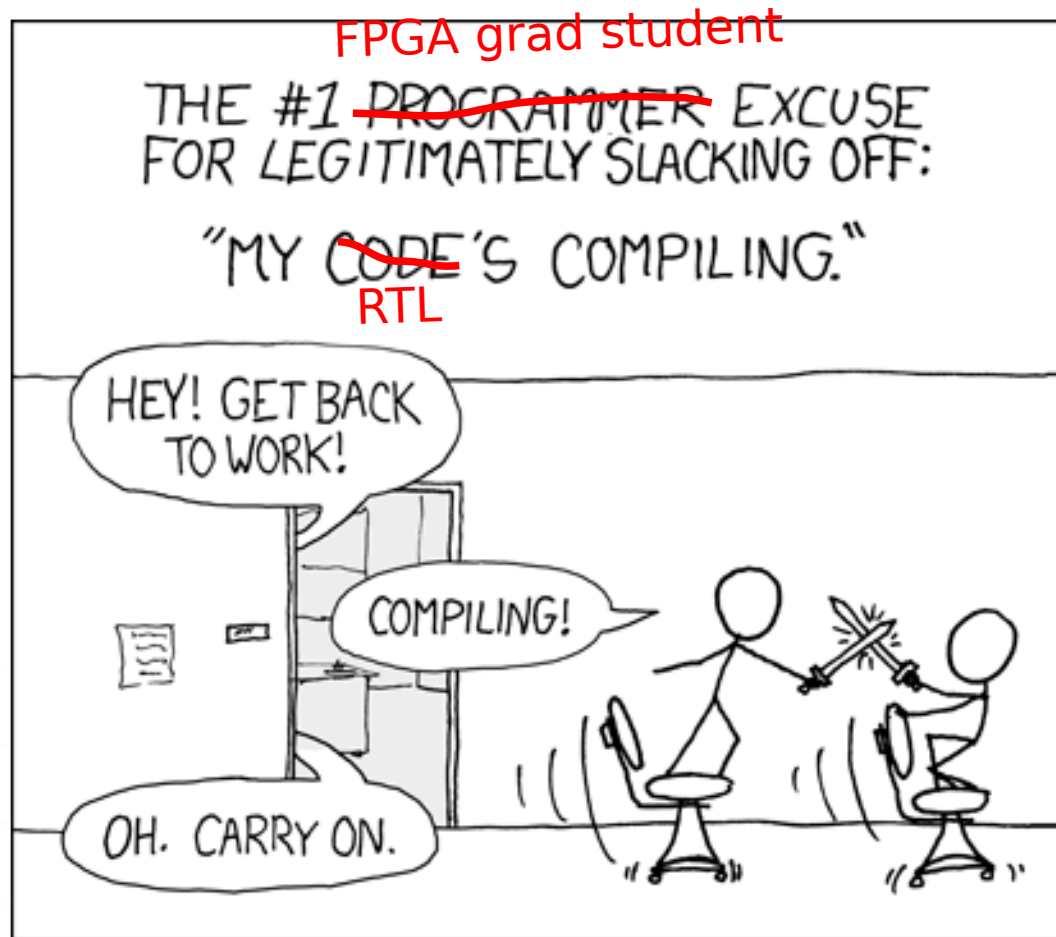
Supervisor: Dr. Guy Lemieux

University of British Columbia, SOC Lab

davidg@ece.ubc.ca

University of Toronto, Oct 9, 2008

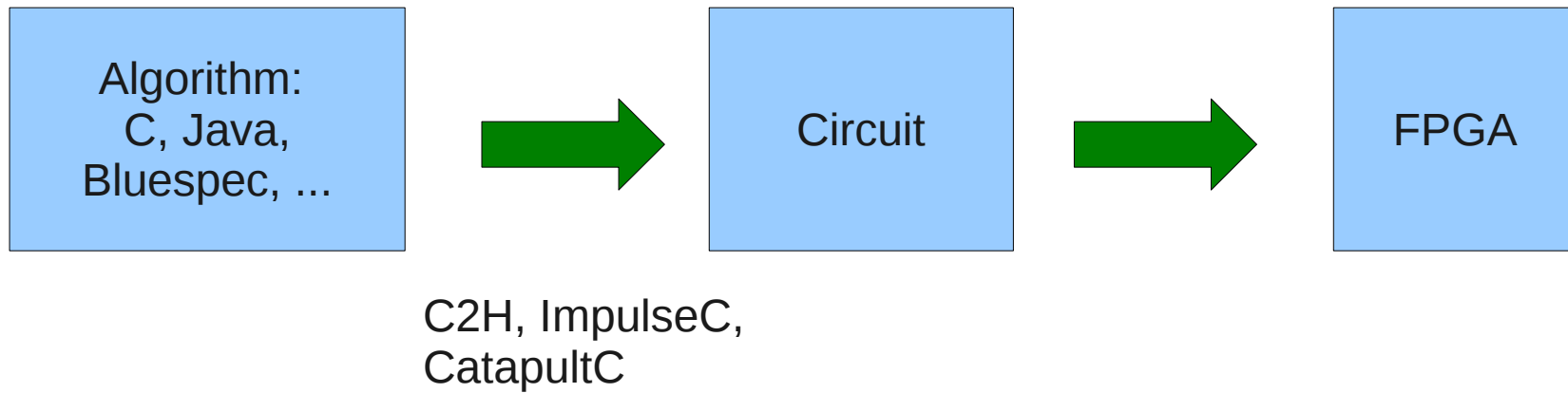
Motivation



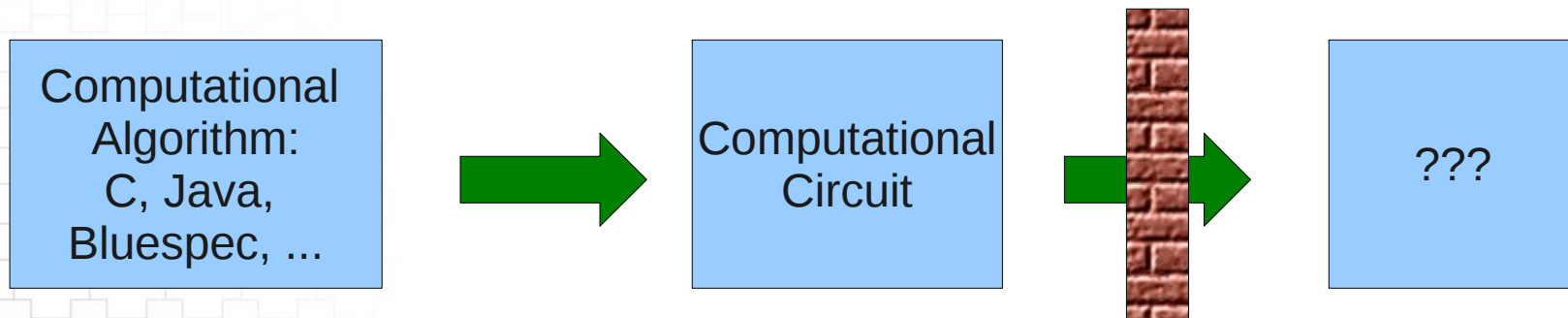
<http://xkcd.com/303/>

Motivation

- Old and busted



- New hotness





Introduction

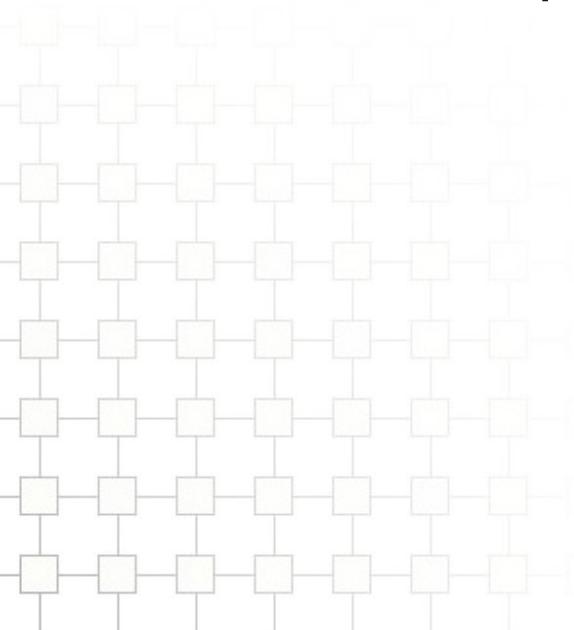
- **Computational Circuits**
 - Software converted to hardware for performance
 - ex. molecular dynamics, rendering, encoding, ...
 - Word-oriented circuits

 - They are not
 - Bit oriented circuits
 - ex. logic simulator
 - FPGAs will always rule these



Introduction

- **Computational Circuits can be very large**
 - Synthesis and Simulation have long runtimes
 - Wastes designer productivity
- **Objective**
 - Quickly compile and run computational circuits
 - Build a spatial computer





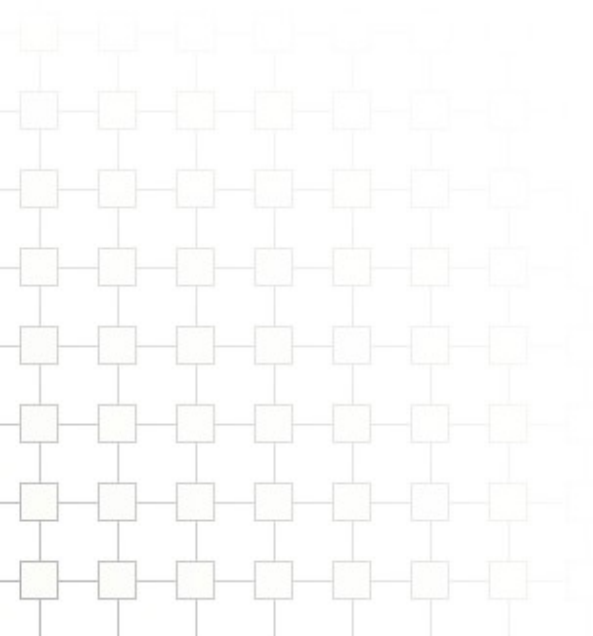
Introduction

- **Computational Circuits on FPGAs**
 - FPGAs bring simulation speed to $\frac{1}{4}$ ASIC
 - Problem 1: **Long Synthesis Time**
 - Synthesis down to gate level
 - Hinders incremental debugging and development
 - Problem 2: **Hard Capacity Limit**
 - Software accustomed to “infinite resources” model
 - Run a few more instructions
 - Add a few MB of memory
 - Add a few million gates ???



Introduction

- Architecture and Tools to implement computational circuits
 - Custom Architecture
 - Fast Tools
- Architecture+Tools address **synthesis time** and **capacity** problems





Introduction

- Custom Architecture
 - Array-of-processors architecture
 - Time-multiplexed (pipelined) interconnect
 - Time-multiplexed ALUs in each PE
 - Capacity vs. speed tradeoff
- Tools
 - Behavioural compilation, word level synthesis
 - Synthesis stops at words, not gates
 - Place-and-route PEs (1,000 instead of 100,000)



Introduction

- **Goals**
 - 10x faster than FPGA CAD tools
 - Minutes, not hours, for large design
 - Compile+Place+Route runtime like a software 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”



Overview

- Introduction
- **Architecture**
- Tools
- Preliminary Results
- Status

Architecture

- **Architecture Summary**

- 2D Array of Processing Elements (PEs)

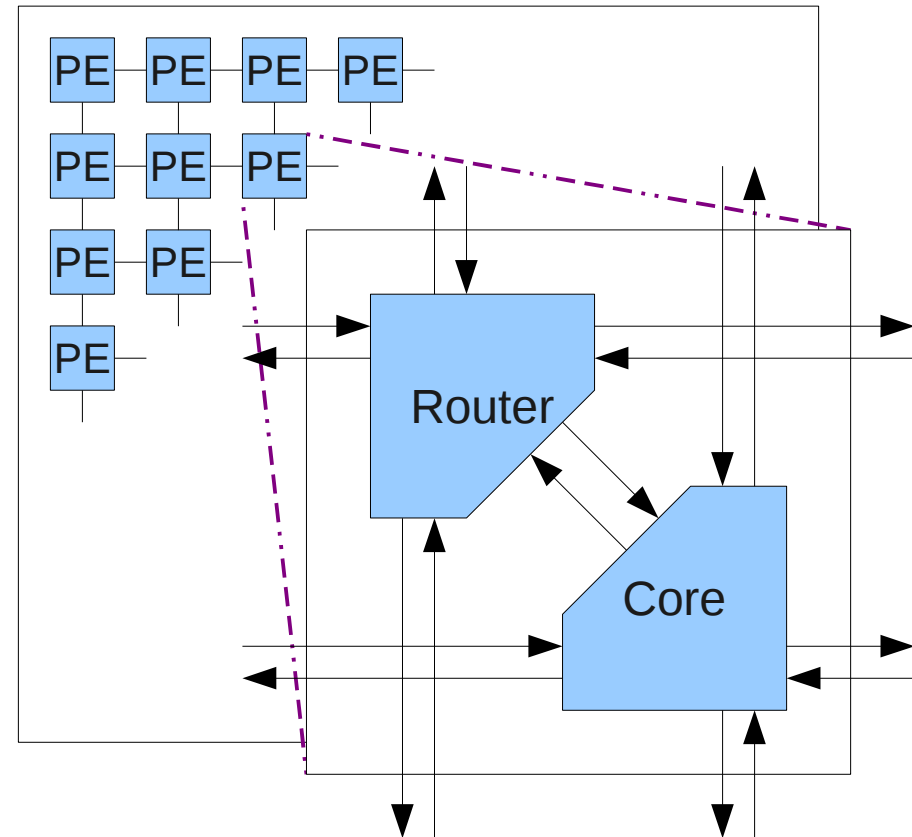
- PE is a small, fast sequential computer
- 3 GHz

- Mesochronous clock network

- Known phase mismatch between neighbours is negligible

- n cycle fixed schedule

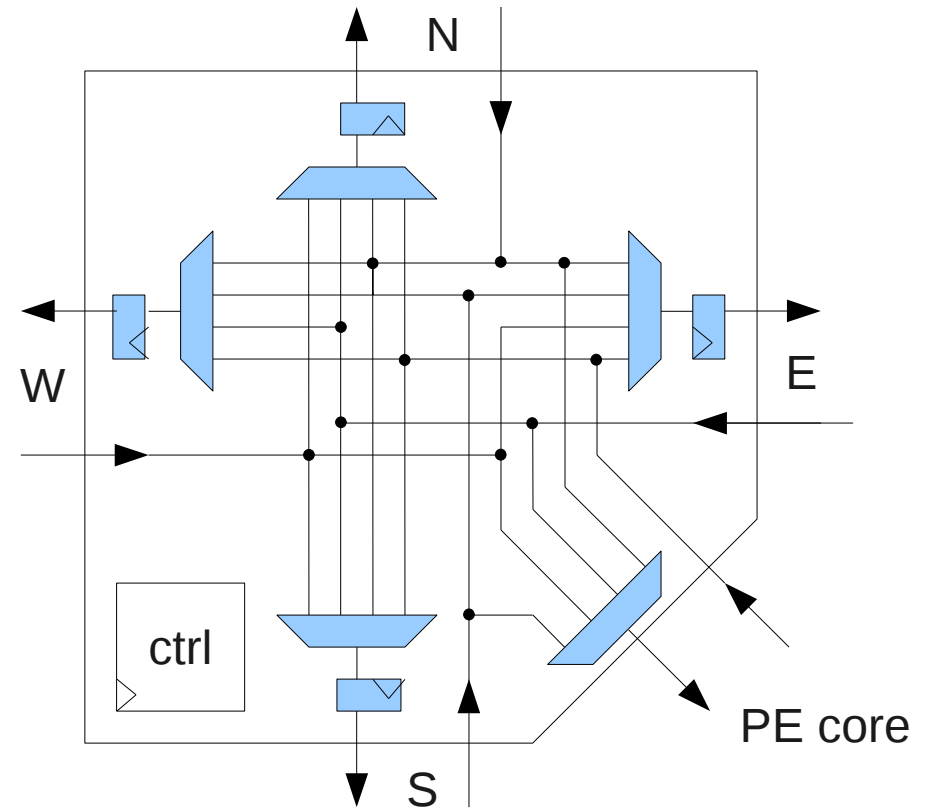
- One user cycle



- PE Router

- 5 x 5 data crossbar
- Fixed static schedule
- Pipelined interconnect
- PE core accepts 1 write/cycle

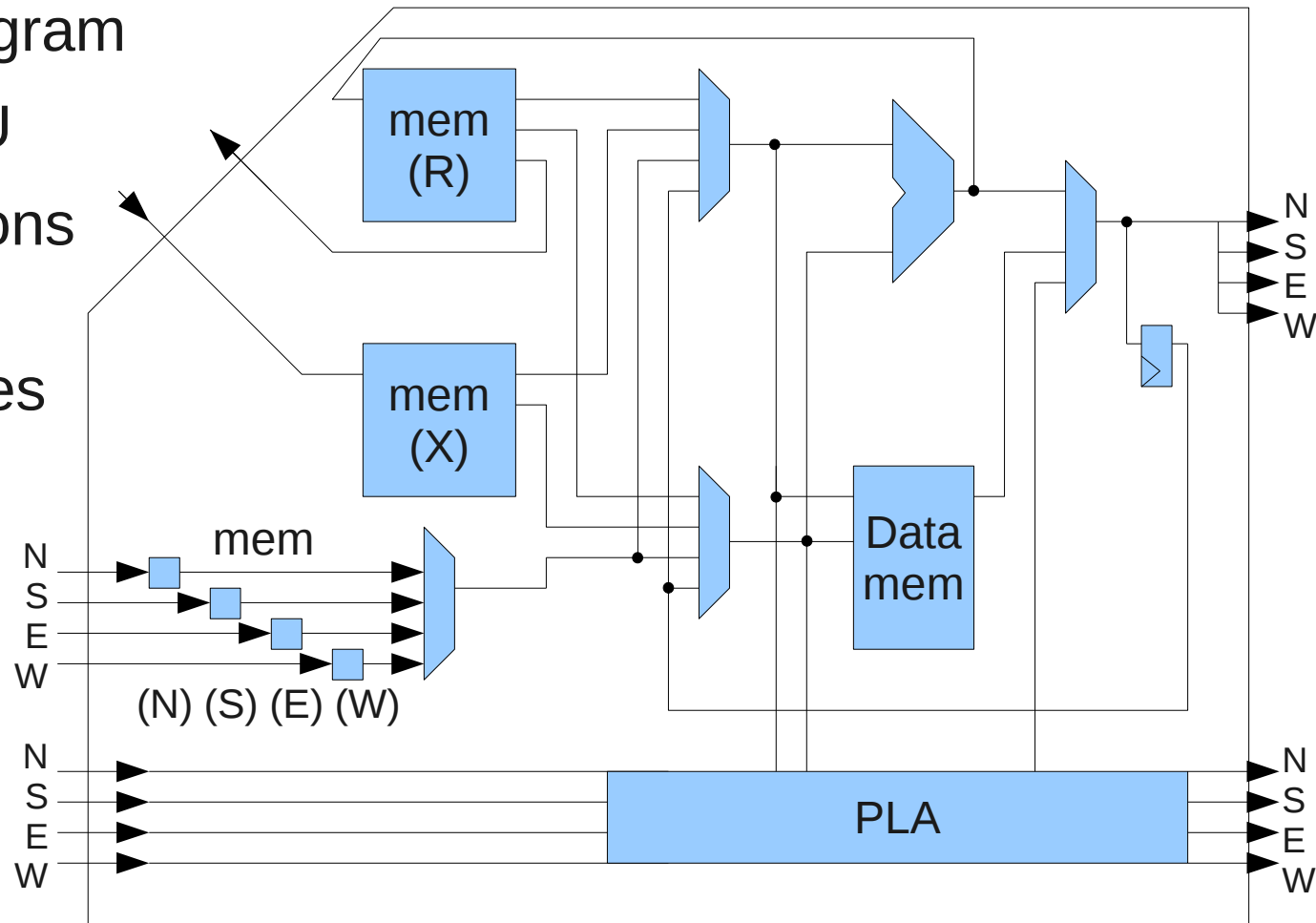
N	S	E	W	PE	Raddr	Waddr
S	PE		PE		4	
		W		N		6
...



Architecture

- PE Core

- Fixed static program
- 1 time-mux ALU
- Direct connections to neighbours
- 6 node memories
- Data memory
- PLA for bitops



- Architecture questions

- PE core

- Node memory sizes?
- Memory size (data, instruction)?
- Heterogeneous (e.g., some columns have more memory)?
- ALU width? 2-way static superscalar?
- PLA size? Use LUTs instead?

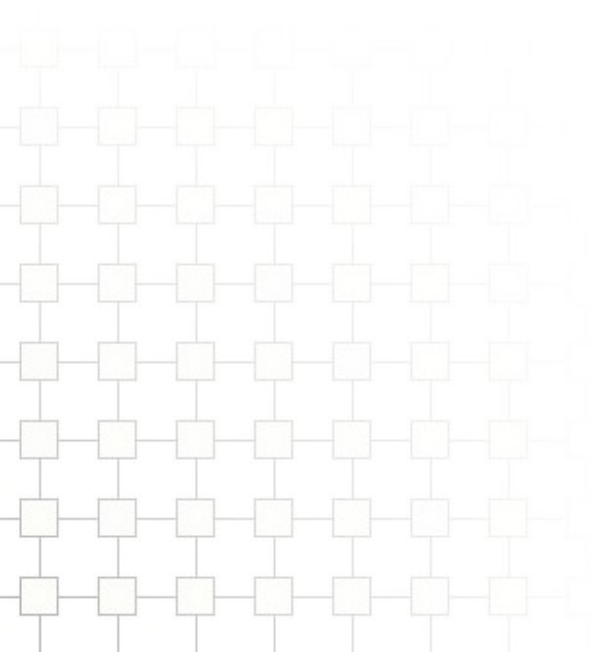
- PE router

- Bus widths?
- Number of routers per PE?
- Serial interconnect?



Overview

- Introduction
- Architecture
- **Tools**
- Preliminary Results
- Status



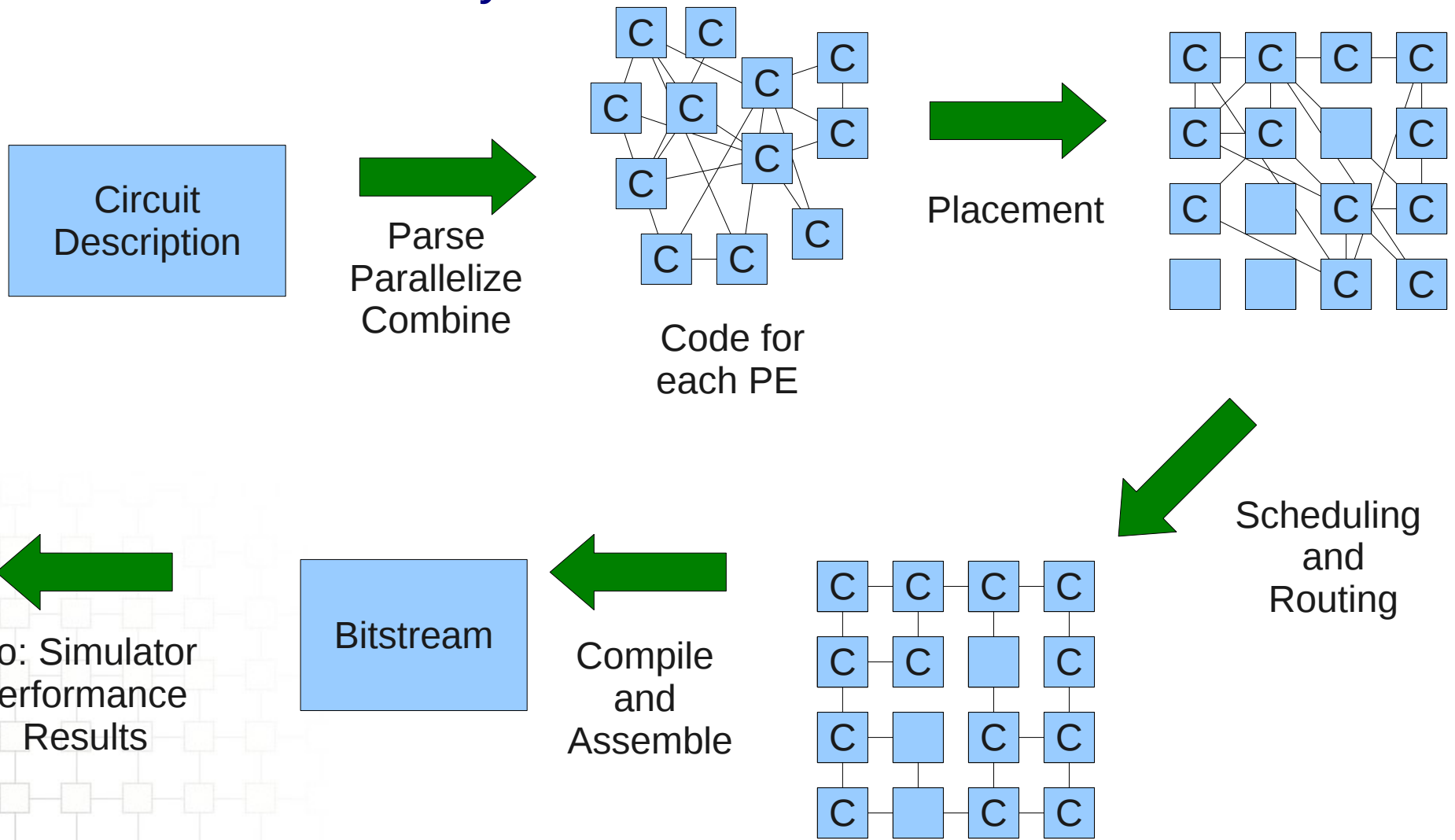
- **Fast tools**

- Behavioural compilation and word level synthesis
- ~1000 PEs vs. ~100,000 CLBs
- Fast algorithms, less sensitive to quality

- **Tool flow**

- Parallelize
- Combine
- Placement
- Schedule and Route
- Code Generation

- Tool Flow Summary



Tools - Example

- **Input**

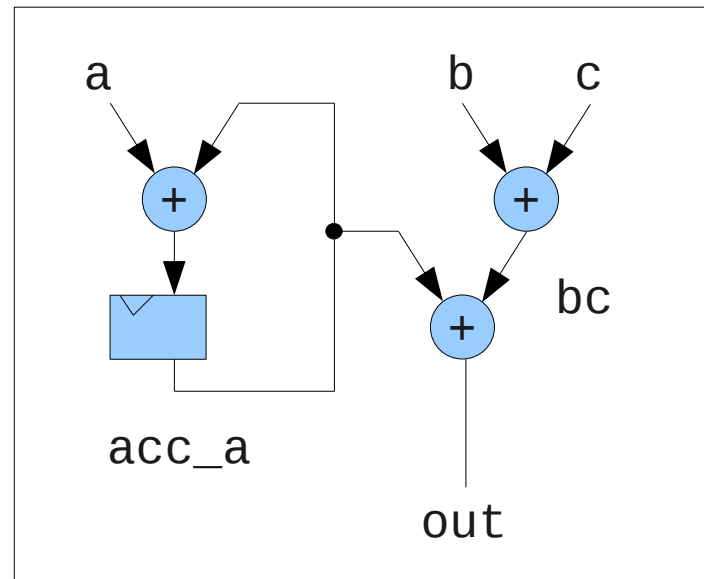
- Verilog

- 32-bit signals

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

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

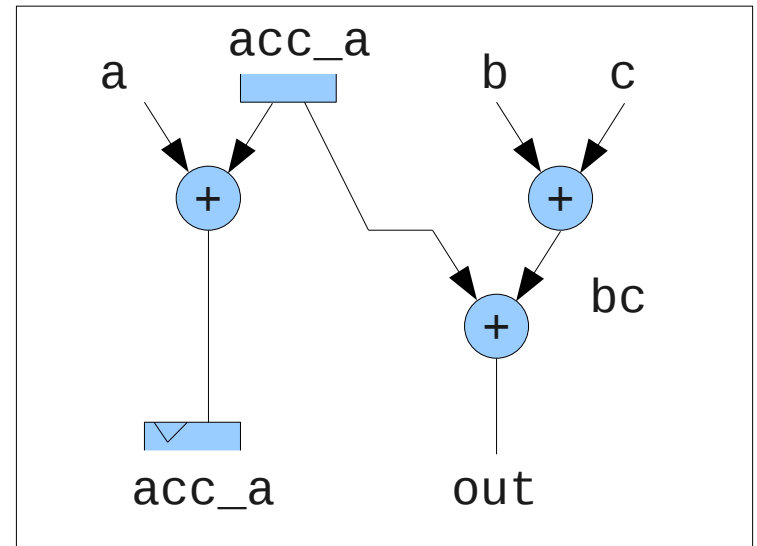
- Behavioural Circuit



Tools - Example

- **Parallelize**

- Parallelize to maximum, word ops
- Initial version: Use Verilator and post-process
- Map registers into virtual inputs/outputs
- Separate word-level and bit-level signals
- Map large user memory to multiple PEs



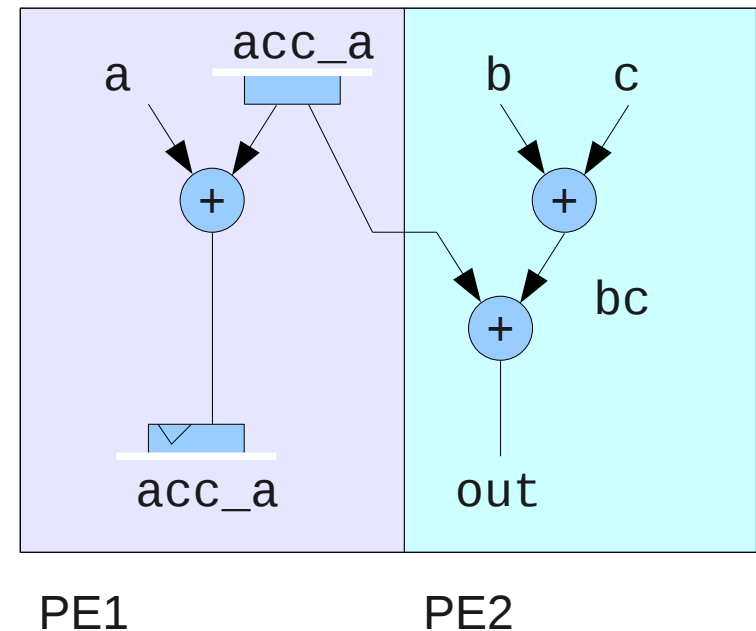
- **Example**

- Parallelization is trivial
- Harder with memories and sequential code

Tools - Example

- **Combine (Clustering)**
 - Minimize sum of communication data widths
 - Initial version: greedy algorithm ($O(n)$)
 - Parallelize+Combine are one tool
 - Don't over-parallelize if not needed

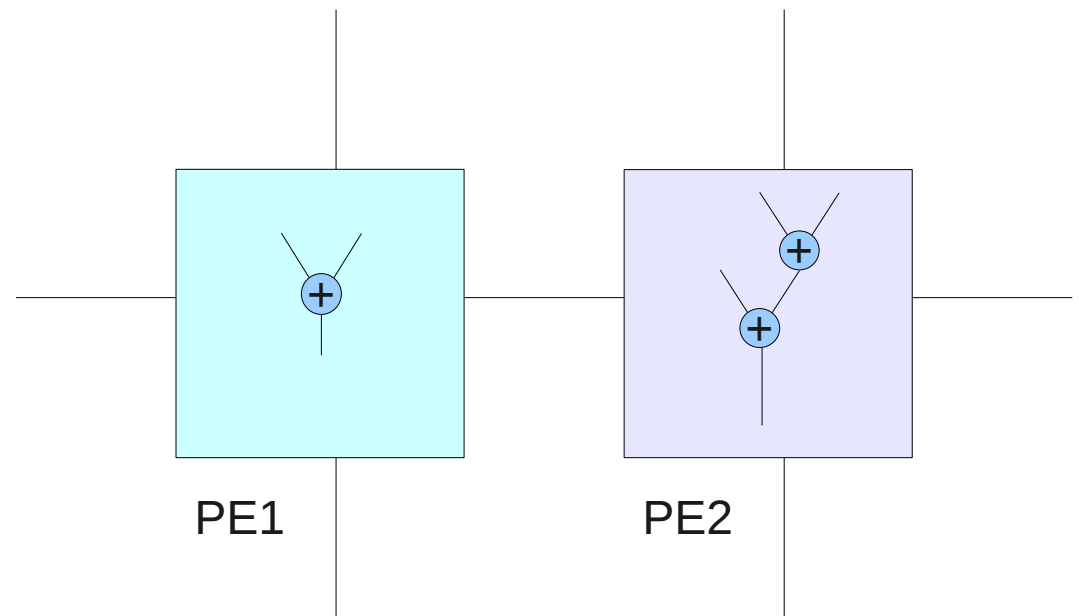
- **Example**
 - Cluster into 2 PEs
 - This is actually a poor clustering but good for the example



Tools - Example

- Placement

- Assign code for each PE to a physical PE
- Initial version: simulated annealing
 - Cost is sum of manhattan distances for all sources to sinks
- Aware of time-multiplexed links



- Example

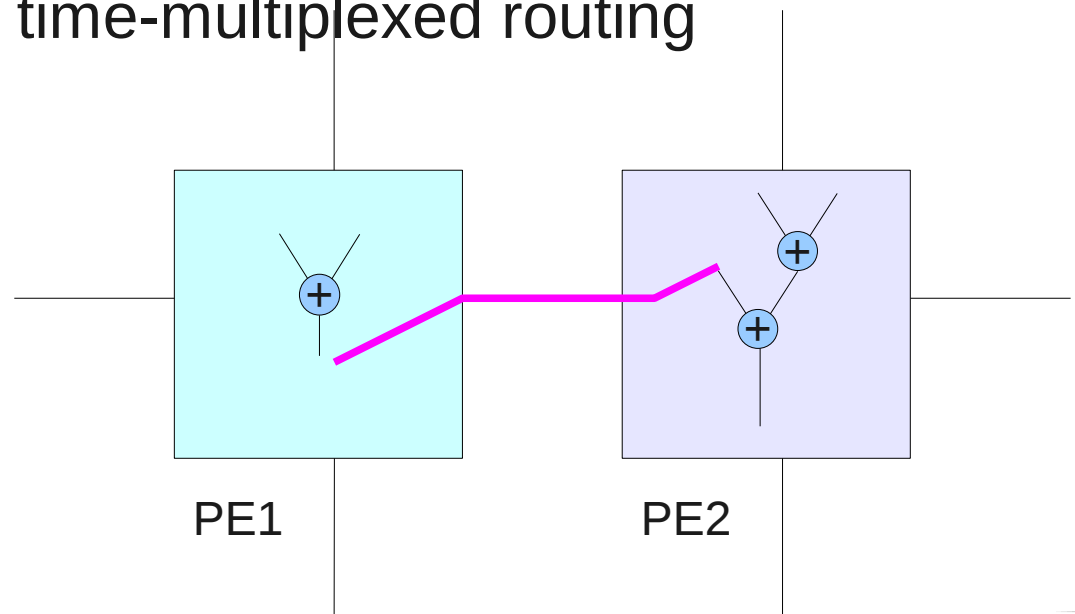
- Only one data communication

Tools - Example

- Router
 - Route data between PEs
 - Initial version: horizontal-then-vertical router ($O(n)$)
 - Like FPGA routing
 - Time multiplexed but static
 - deterministic
 - Future work: aware of time-multiplexed routing

- Example

- Single route
- Needs to cross clock cycle





Tools - Example

- Scheduler
 - Scheduler and Router are one tool
 - Assign code and route hops to timeslots
 - Initial version: timeslot-by-timeslot ($O(n)$), start at slot 0
 - PE core resource conflicts
 - Push back schedule
 - Future work: reorder code
 - PE router conflicts
 - Delay route



Tools - Example

- Scheduler
 - Use node memory to implement registers and wires
 - Memory write followed by read → wire
 - Memory read followed by write → register
 - Problem: Router needs a code schedule, but scheduler needs a routing
 - Placement tool provides an *initial* schedule for the router

Tools - Example

- Example – final code/route schedule

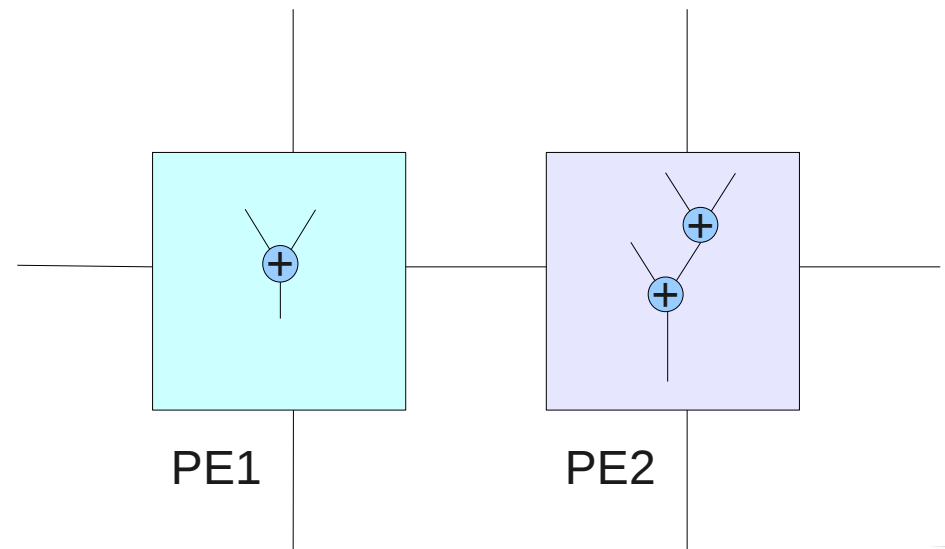
Time	PE1		PE2	
	core	router	core	router
0	ILOAD R1	R2 -> E	ILOAD R1	
1	ADD R2, R1, R2		ILOAD R2	W -> X0
2	NOP		ADD R1, R1, R2	
3	NOP		ADD R1, R1, X0	
4	NOP		IOSTORE R1	

```

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

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

```





Tools - Example

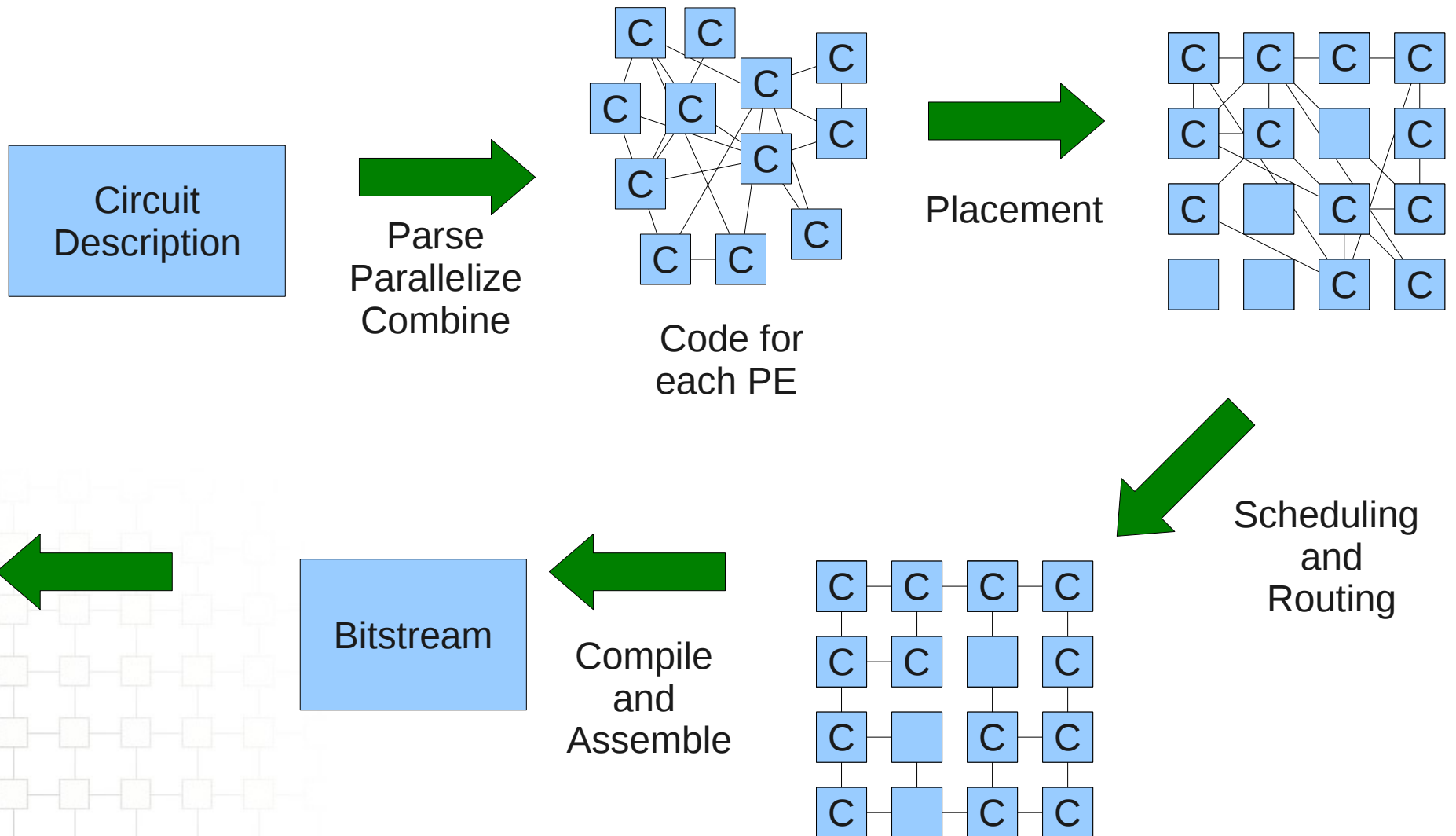
- Example – final code/route schedule

	Our Tools	QuartusII
Target	Ours	Cyclonell
Compile Time	1.6s	43s
Speed	~600MHz	264 Mhz

- 600 Mhz = 5 system clocks at 3 GHz

- R2 is a register in PE1, but R1 is a wire
- R1 and R2 are both wires in PE2

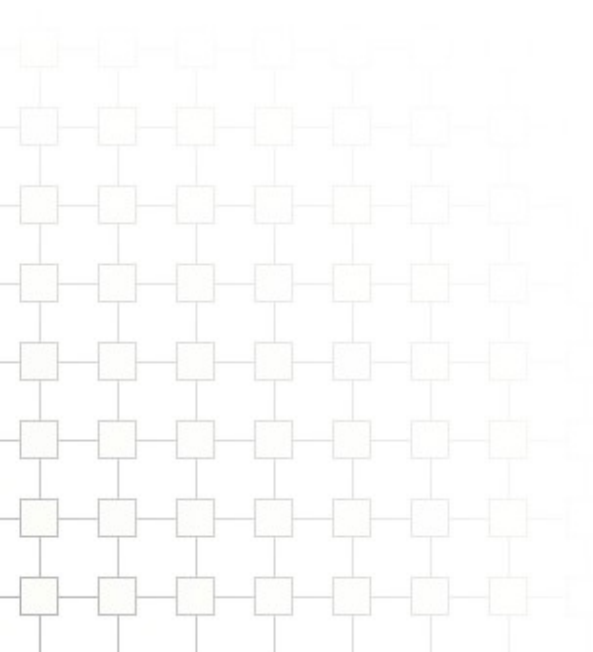
- Code Generation





Overview

- Introduction
- Architecture
- Tools
- **Preliminary Results**
- Status





Preliminary Results

- Motion Estimation

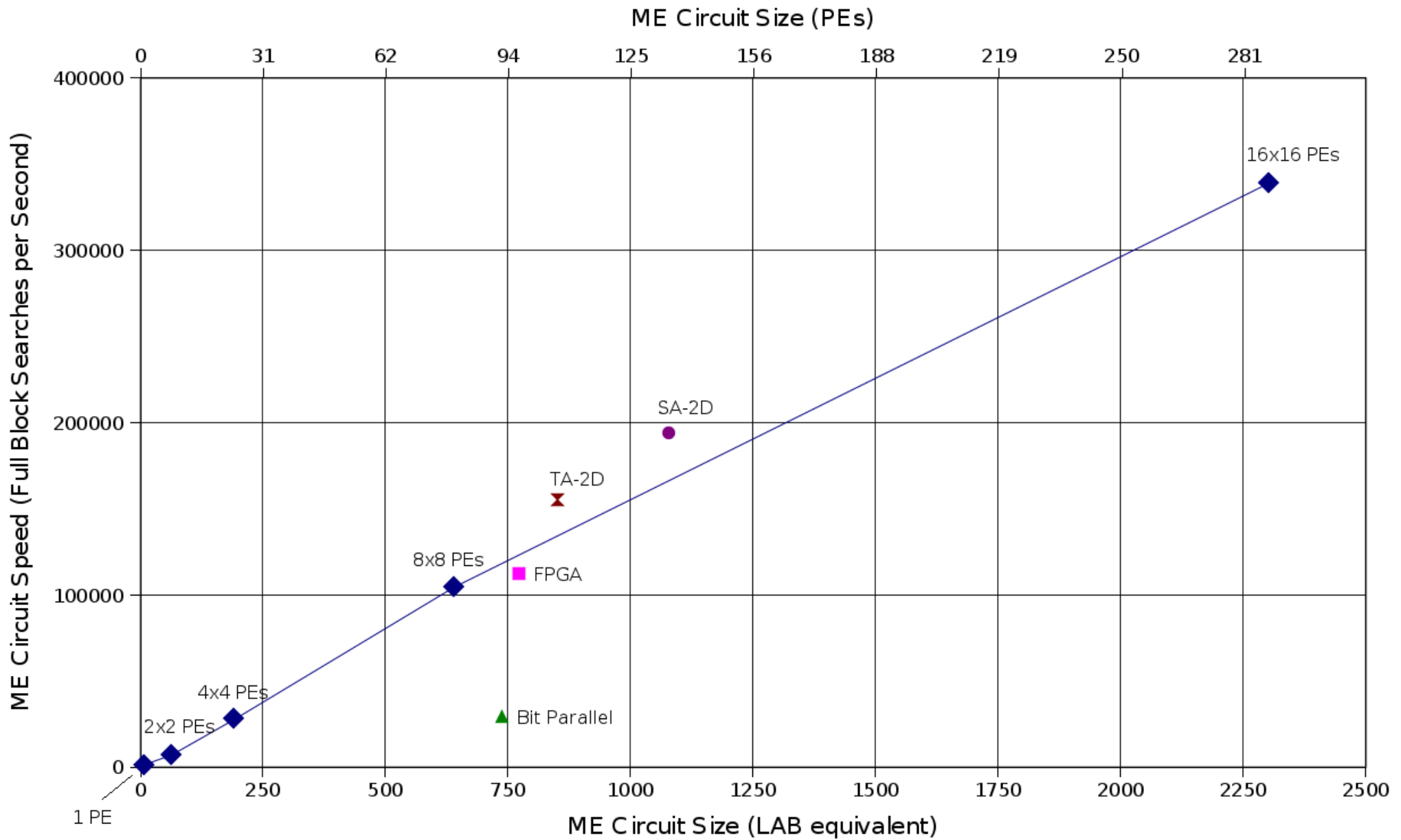
- 16x16 reference block (**R**)
- Search a 32x32 pixel image (**M**) for the best match
- Best match is the lowest sum of absolute differences (**SAD**)

```
for x=-15 to 31
  for y=-15 to 31
    SAD=0
    for i=0 to 15
      for j=0 to 15
        SAD += abs( R[i,j] - M[i+x, j+y] )
      end
    end
    (check for highest SAD)
  end
end
```

- Highly parallelizable



Preliminary Results





Preliminary Results

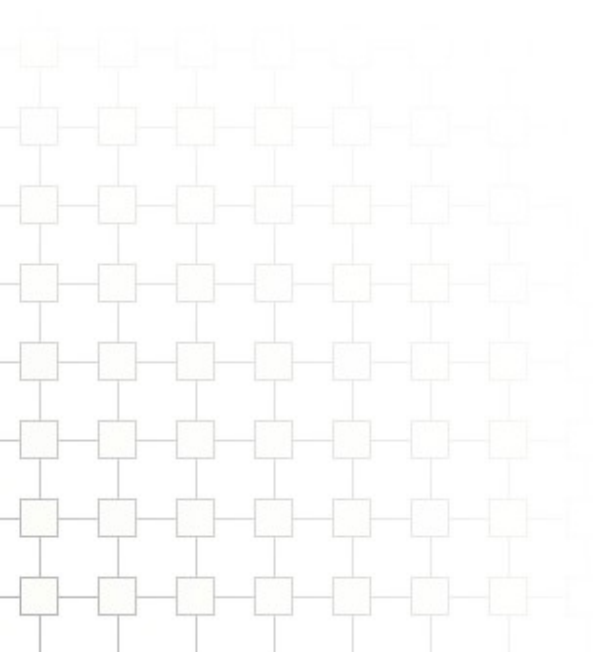
- Motion Estimation Data

Architecture	Actual Pes	LABeq	User Mhz	Full Block Searches/sec	Search Speedup
1 PE	1	8	3	1324	0.012
2x2 PEs	8	64	16	7036	0.063
4x4 PEs	24	192	61	27710	0.248
8x8 PEs	80	640	231	104424	0.930
16x16 PEs	288	2304	750	339188	3.033
FPGA: Ours		773	247	111817	1.000
FPGA: Bit Parallel		738	738	29296	0.262
FPGA: TA-2D		851	239	154592	1.382
FPGA: SA-2D		1079	227	193686	1.732



Overview

- Introduction
- Architecture
- Tools
- Preliminary Results
- **Status**





Status / To Do List

1. Develop a Verilog benchmark

- Series of simple benchmarks (eg, FIR, motion est., ...)
- One complex benchmark
- Help!!!!

2. Modify Verilator to expose parallelism, extract 1-bit signals

- Parallelism found, 1-bit signals not yet separated

3. Modify Verilator to add Combine step

4. Implement Placement

5. Implement Schedule and Route

- (Initial version done by Rosemary Francis)

6. Architecture experiments



Thanks

- **Main Idea**
 - Take a computational circuit and compile+run it fast on a spatial architecture
- **Custom Architecture**
 - Time-multiplexed array-of-processors
 - High bandwidth, low latency communication
- **Fast Tools**
 - Behavioural compilation
 - Coarse grained architecture
- **To Do**
 - Lots. But it works so far!