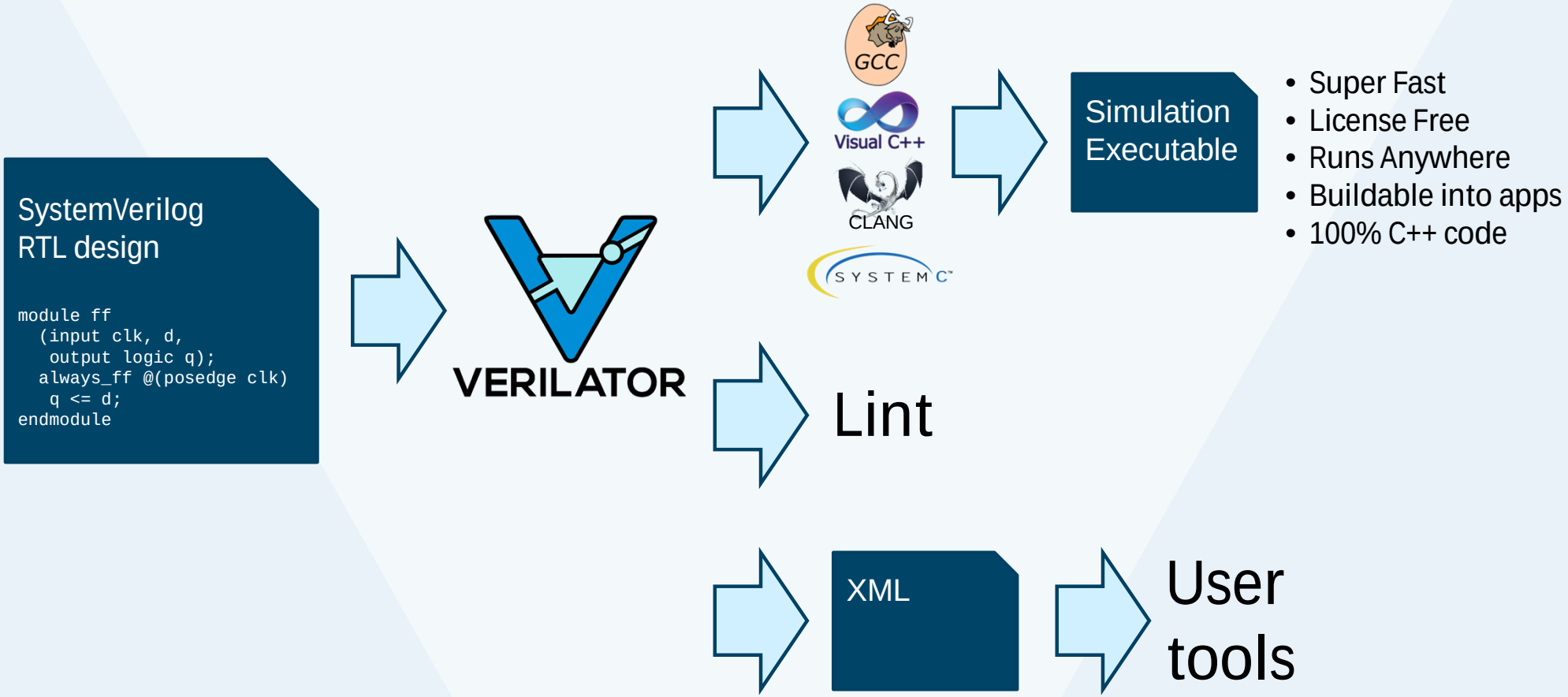
VERILATOR

Verilator, Accelerated:

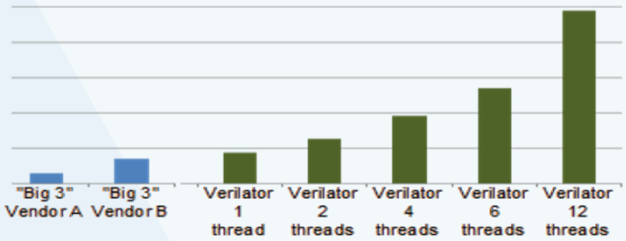
Accelerating development, and case study of accelerating performance

Wilson Snyder, 2020

Verilator


SystemVerilog
RTL design
module ff
(input clk, d,
output logic q);
always_ff @(posedge clk)
q <= d;
endmodule
VERILATOR
GCC
Visual C++
CLANG
SYSTEMC
Simulation
Executable
Super Fast
License Free
Runs Anywhere
Buildable into apps
100% C++ code
Lint
XML
User
tools

Why Verilator?



Fast

- Outperforms many commercial simulators
- Single- and multi-threaded output models

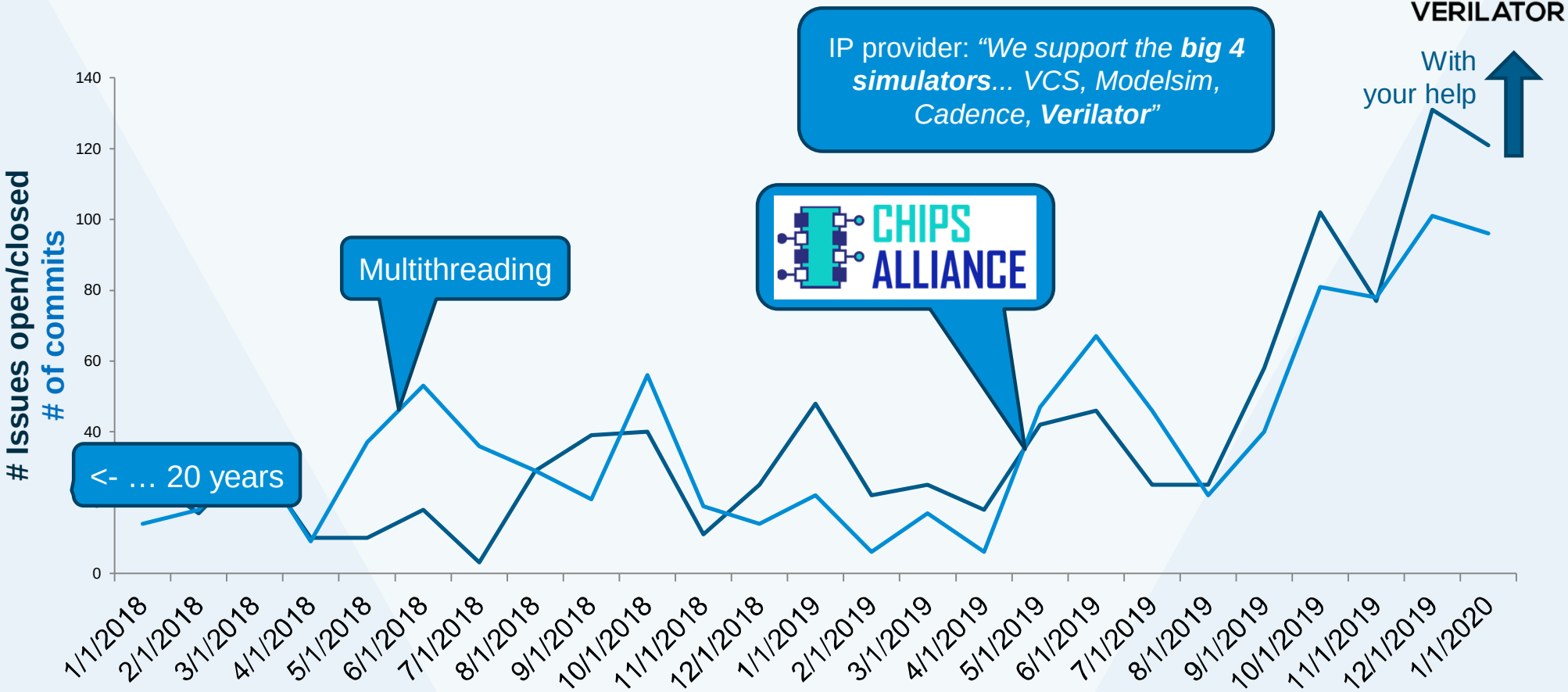
Widely Used

- Wide industry and academic deployment
- Out-of-the-box support from Arm, and RISC-V vendor IP

Community Driven, Professionally Supported, & Openly Licensed

- Guided by the CHIPS Alliance and Linux Foundation
- Professional support available through partners
- Open, and free as in both speech and beer
- More simulation for your verification budget

Accelerated Development



Improvements Last 3 Months

Language

- Dynamic arrays
- Bounded queues
- Interface parameter arrays
- String .atoi, character indexing
- Immediate cover
- $dumpfile
- type(expression) and $typename
- "|->“

& Others. Thanks ALL!

Performance

- split_var for UNOPTFLAT
- Verilator binary speedups
- FST trace speedups

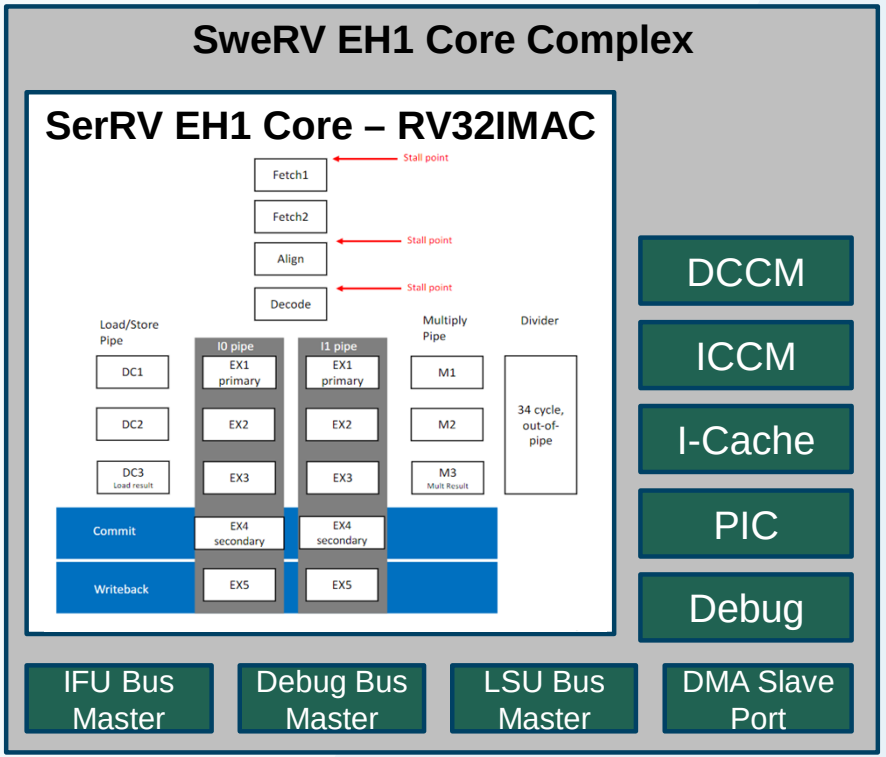
Usability & Lint

- Docker images
- Attributes in config files
- Lint misused defines
- Lint misused genvars
- --protect-lib improvements

Case Study: Accelerate Compile & Simulation

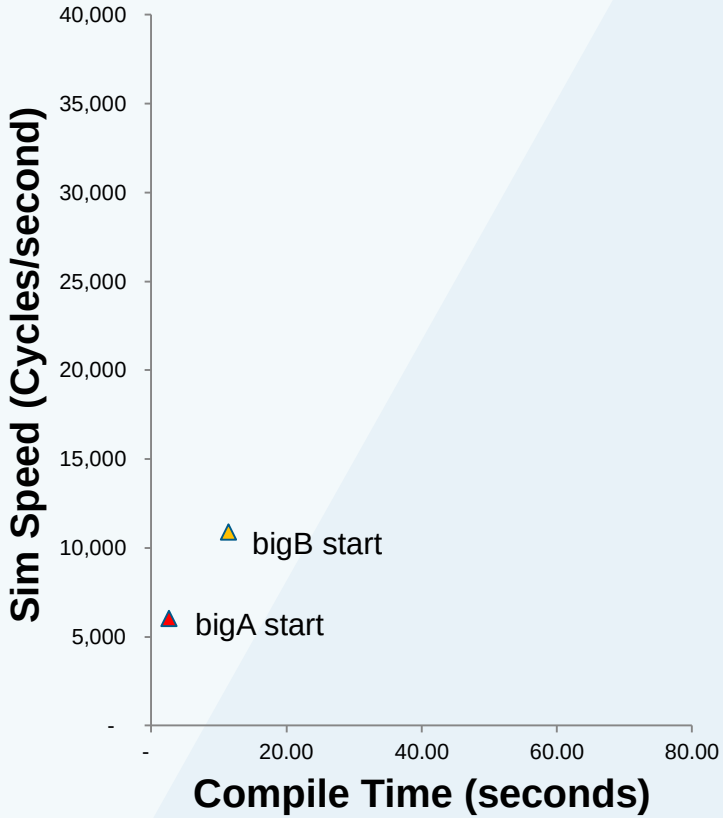
Case Study: Accelerate Compile & Simulation

- Western Digital/CHIPS Alliance's SweRV EH1
- Simulate cmark.hex file in distribution



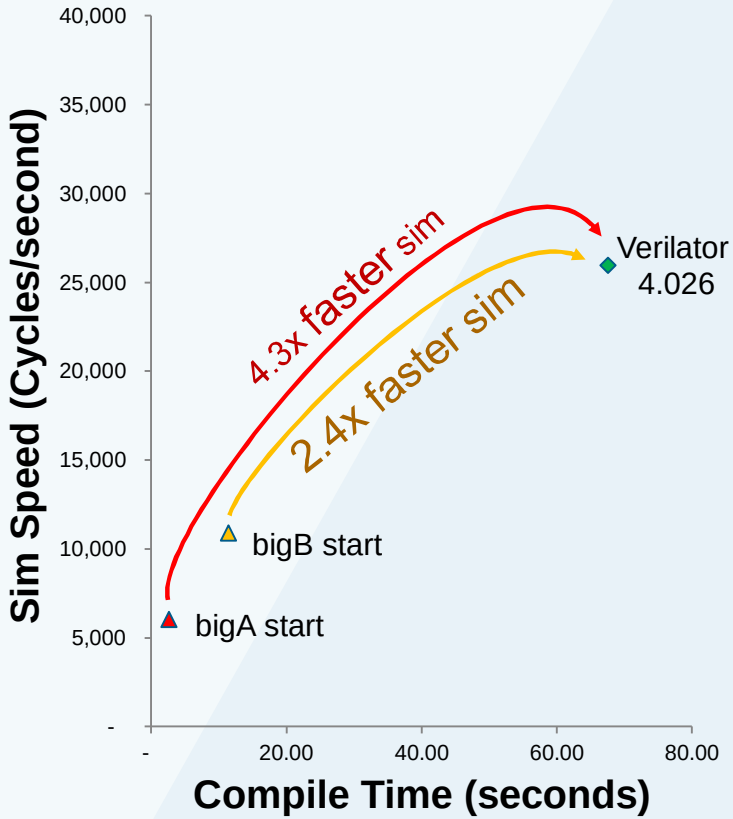
0: Commercial Starting Point

- BigA & BigB Closed-Source Simulators
  - Wait for license not included ☺



0: Starting Point

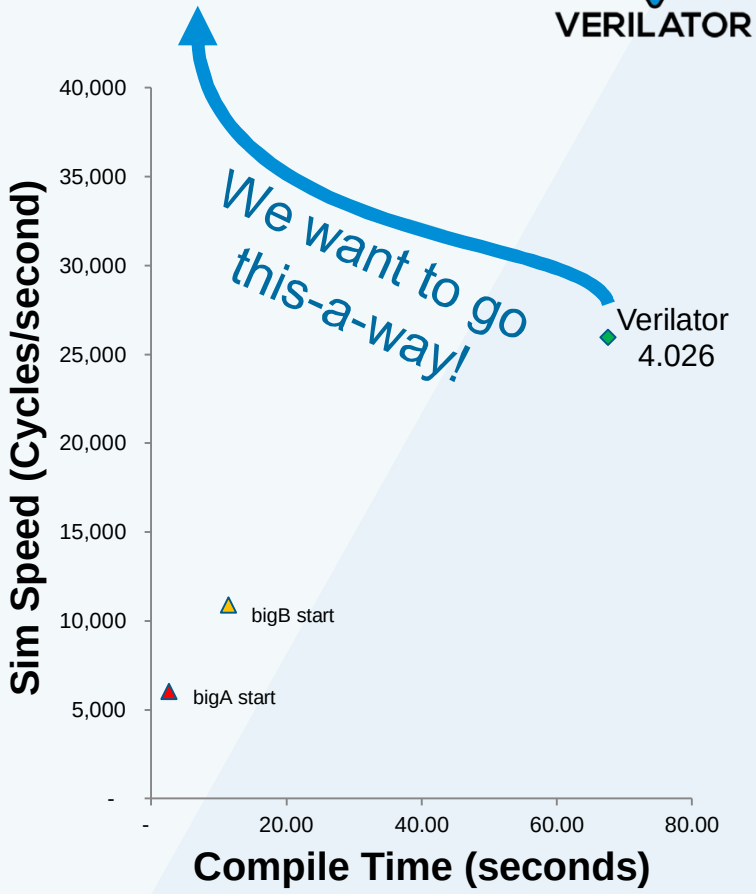
- BigA & BigB Closed-Source Simulator
  - Wait for license not included ☺
- Verilator
  - Verilate: 7.5 sec
  - GCC Compile: 60.1 sec ☹
    - 5.9 to 25 times slower than BigA/B
  - Run: 26,000 cps ☺
    - 2.4 to 4.3 times faster than BigA/B



0: The Goal

Our Topic:

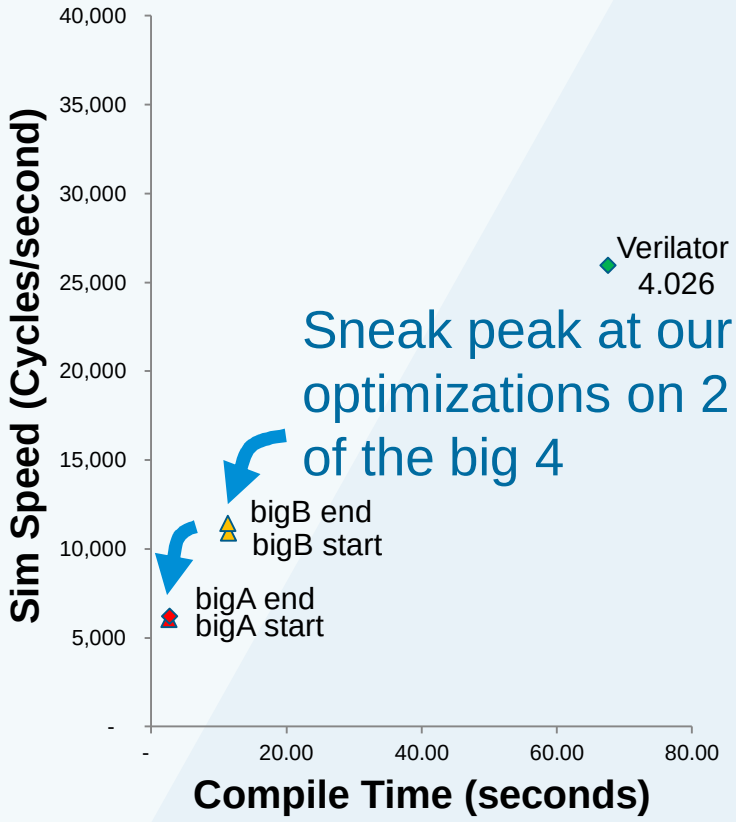
Can Verilator do better?



0: Peek At Results

In Fairness…

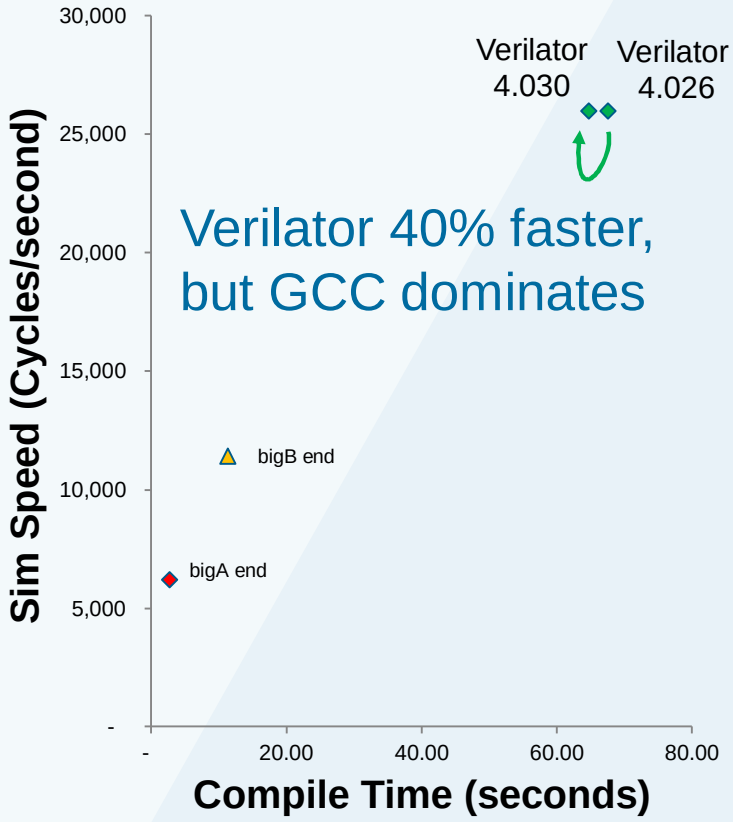
Some of what we discuss helps the closed-sourced simulators



1: Use Latest Verilator

- Upgrade to 4.030
  - Recent community runtime improvements

New!

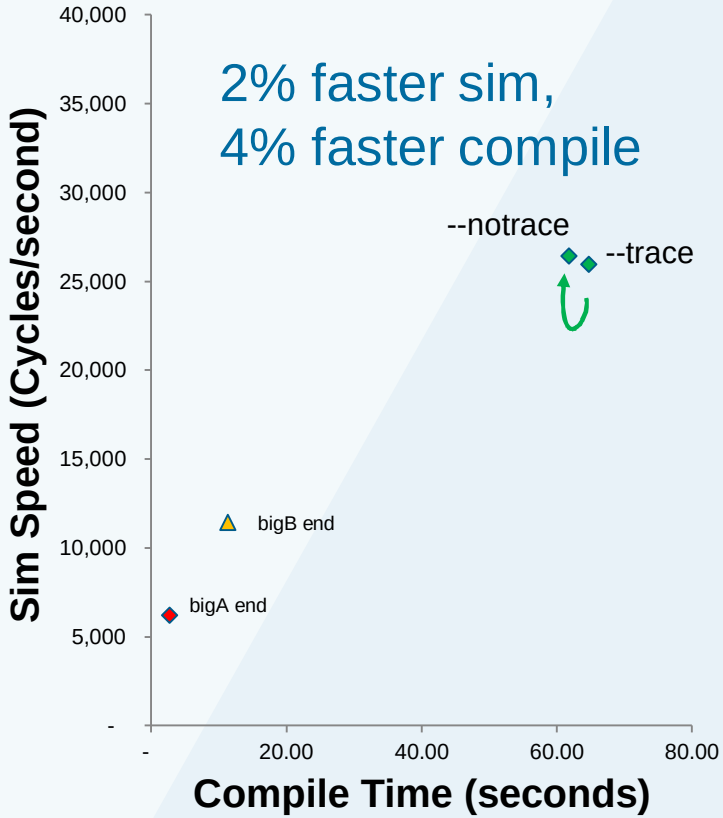


2: Turn off Tracing

- Found in logfile:

```
verilator … -trace
```

- Remove `-trace`. Tracing is great for debug, but reduces optimizations

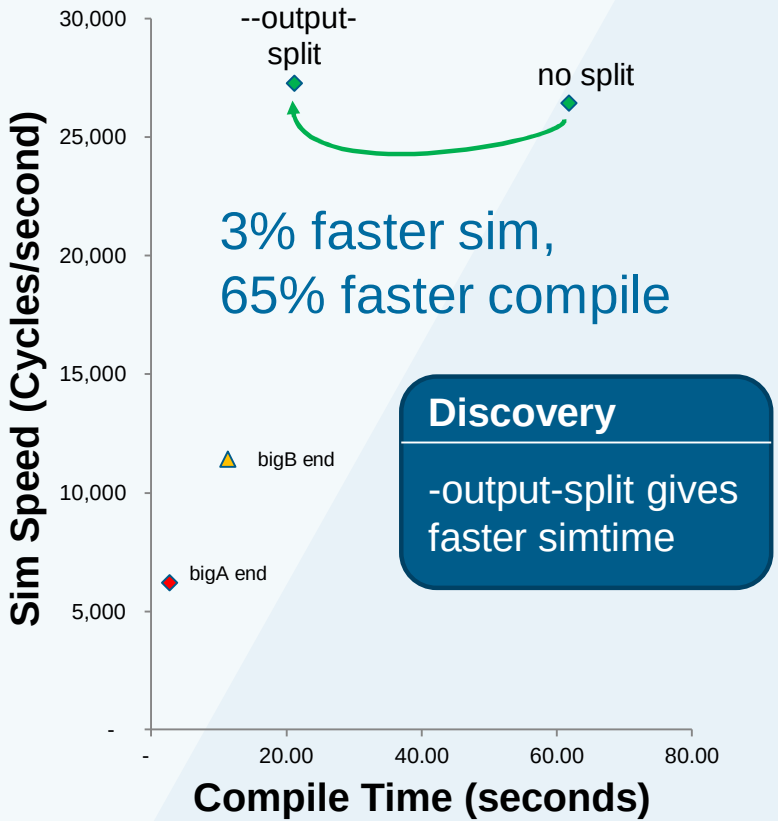


3: Use Parallel Make

- Make larger number of smaller C++ files so can compile in parallel

```
$ verilator … –output-split 15000
$ make –j 8 VM_PARALLEL_BUILDS=1 …
```

- Use `icecream` or similar to distribute compiles across serversm (not shown)



4: Use CCache

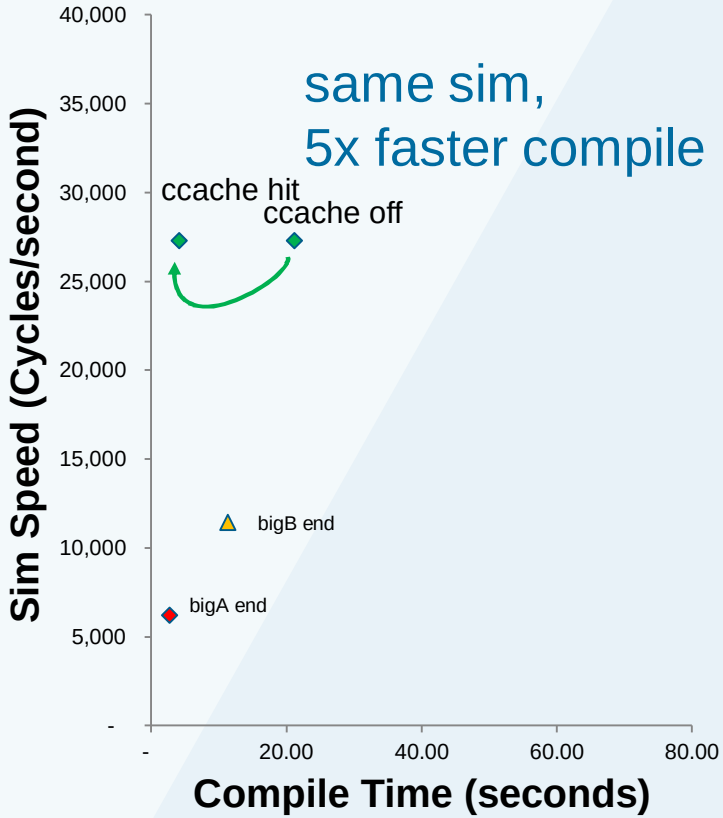
- Cache compiles so rebuild faster

```
$ export CCACHE=ccache
$ make –j 10 …
```

- Small Verilog changes should give ccache hit on many files
- Further slides will not include this gain

Help Wanted

Improve Verilator to make ccache hit more often



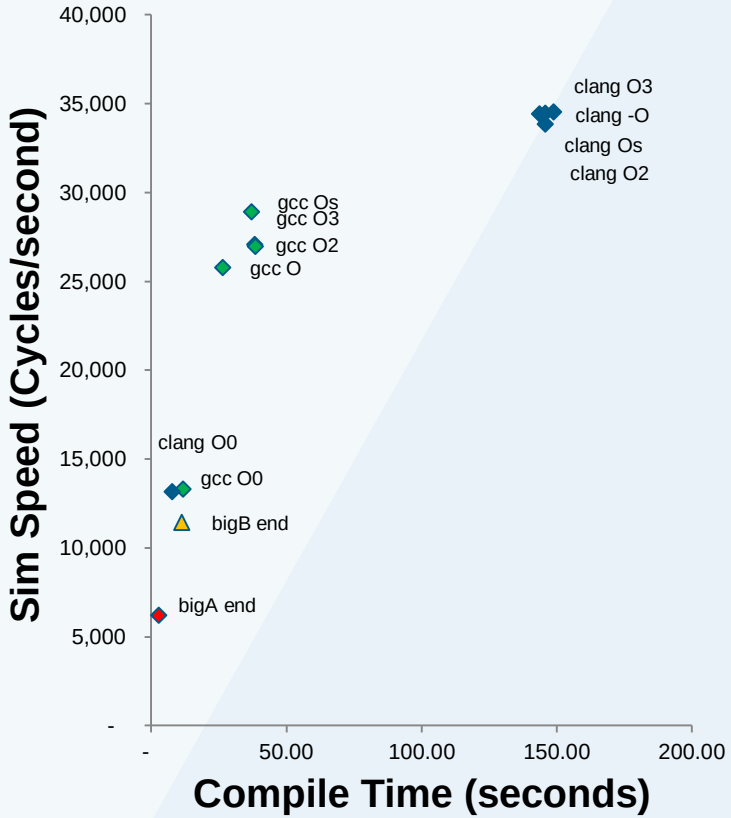
5: Compiler & Compiler Optimization

- Use compiler optimization

```
$ verilator … -CFLAGS -O0
```

  – Also -O, -O2, -O3 -Os

  – clang instead of GCC



6: Compiler Profile-driven Optimization

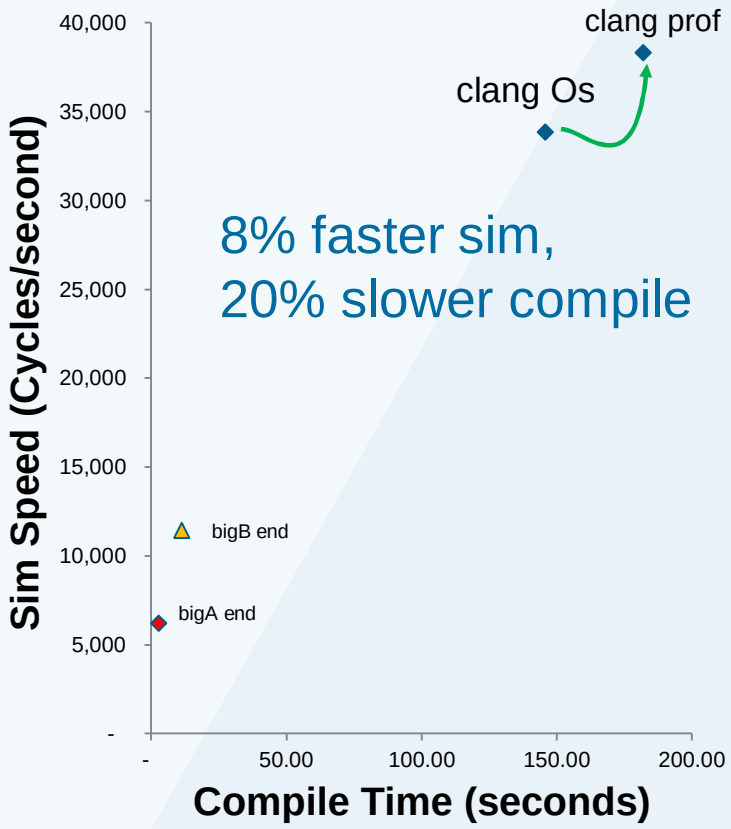
- Use profile-driven optimization

  `$ clang … -fprofile-generate`

  Run simulation to collect profile

  `$ clang … -fprofile-use` *`datfile`*

- Great gain, but painful 2-step process
- Further slides will not include this gain



7: Fix UNOPTFLAT

- Logfile contains

  `verilator … --Wno-UNOPTFLAT`

- UNOPTFLAT warns about potential performance issues
  1. Remove –Wno-UNOPTFLAT
  2. Add –report-unoptflat
  3. Optionally view graphs of paths
  4. Add /*verilator split_var*/ New!

Result - no performance gained (oh well!)

8: Analyze run-time performance

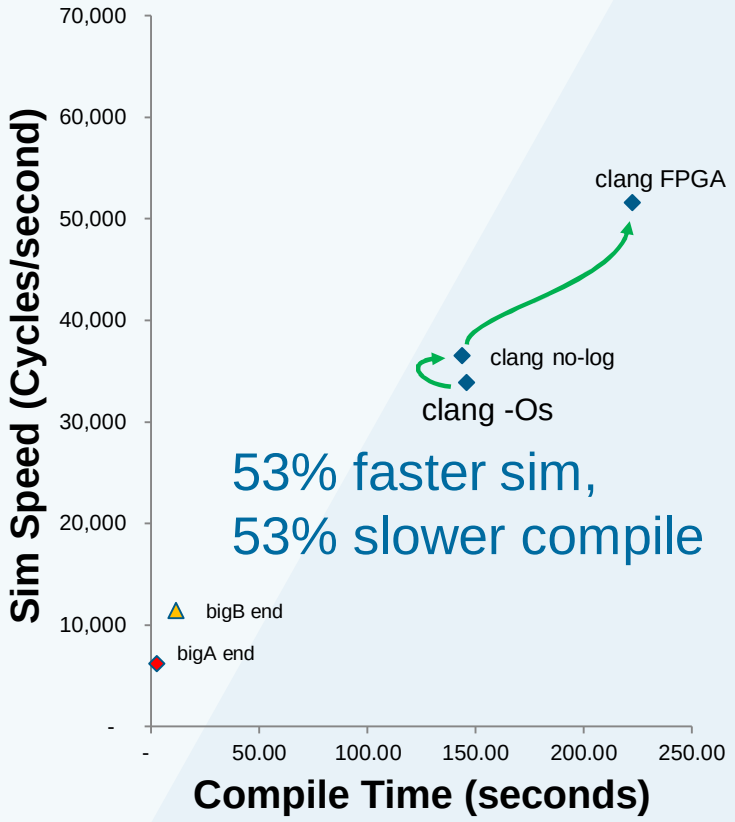
```
$ verilator … -prof-cfuncs
```

```
  0.15%  C++
 29.03%  Common code under Vtb_top
  0.62%  VLib
 68.72%  Verilog Blocks under Vtb_top
…
  0.51%  _vl_vsformat (1)
  0.45%  dec_tlu_ctl.sv:2495 (2)
  0.22%  dec_decode_ctl.sv:2507 (3)
```

1. Format because of instruction logging
2. Dec_tlu_ctl large statement
3. Use RV_FPGA_OPTIMIZE for faster flops

Help Wanted

Optimize 2 decoder

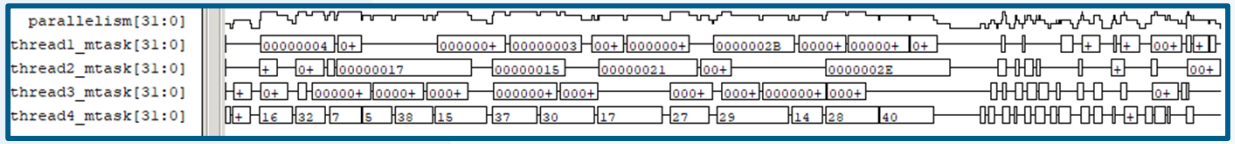


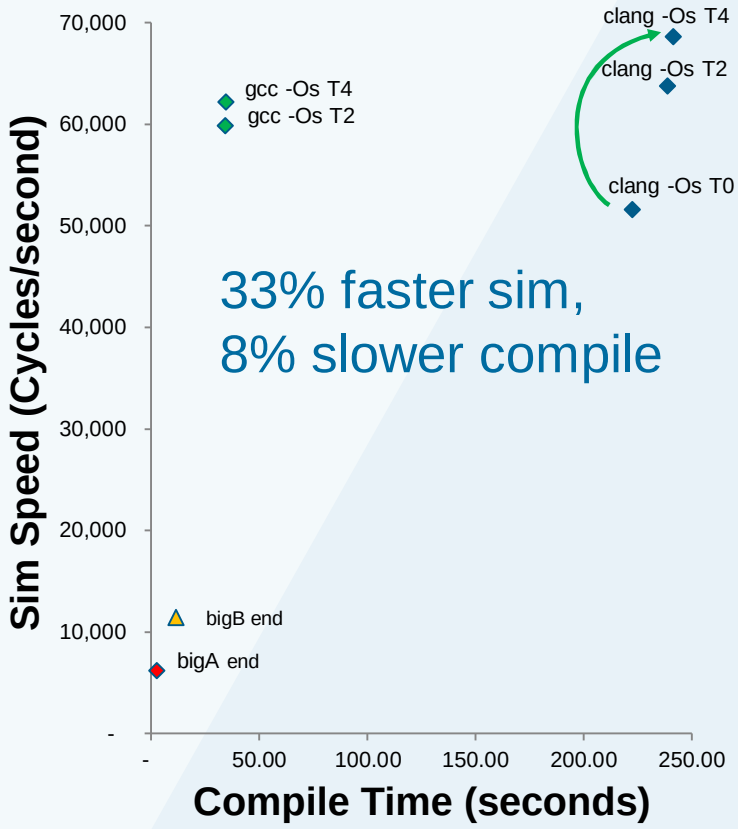
9: Threading

- Try threading <u>after</u> optimizing single-threaded

```
$ verilator … -threads 4
$ numactl –C 0,2,4,6 obj/sim_exe
```

Verilator_gantt report



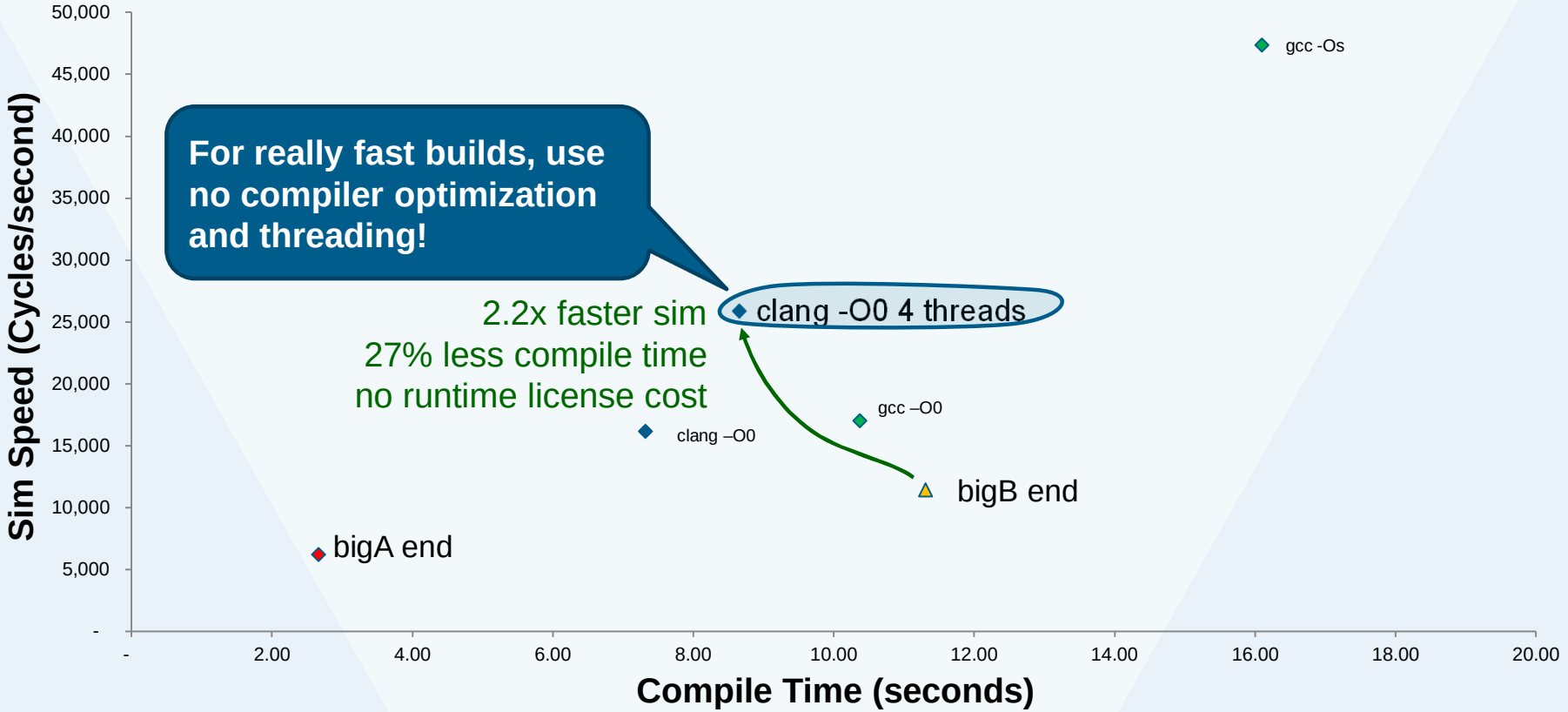


33% faster sim,
8% slower compile

Help Wanted

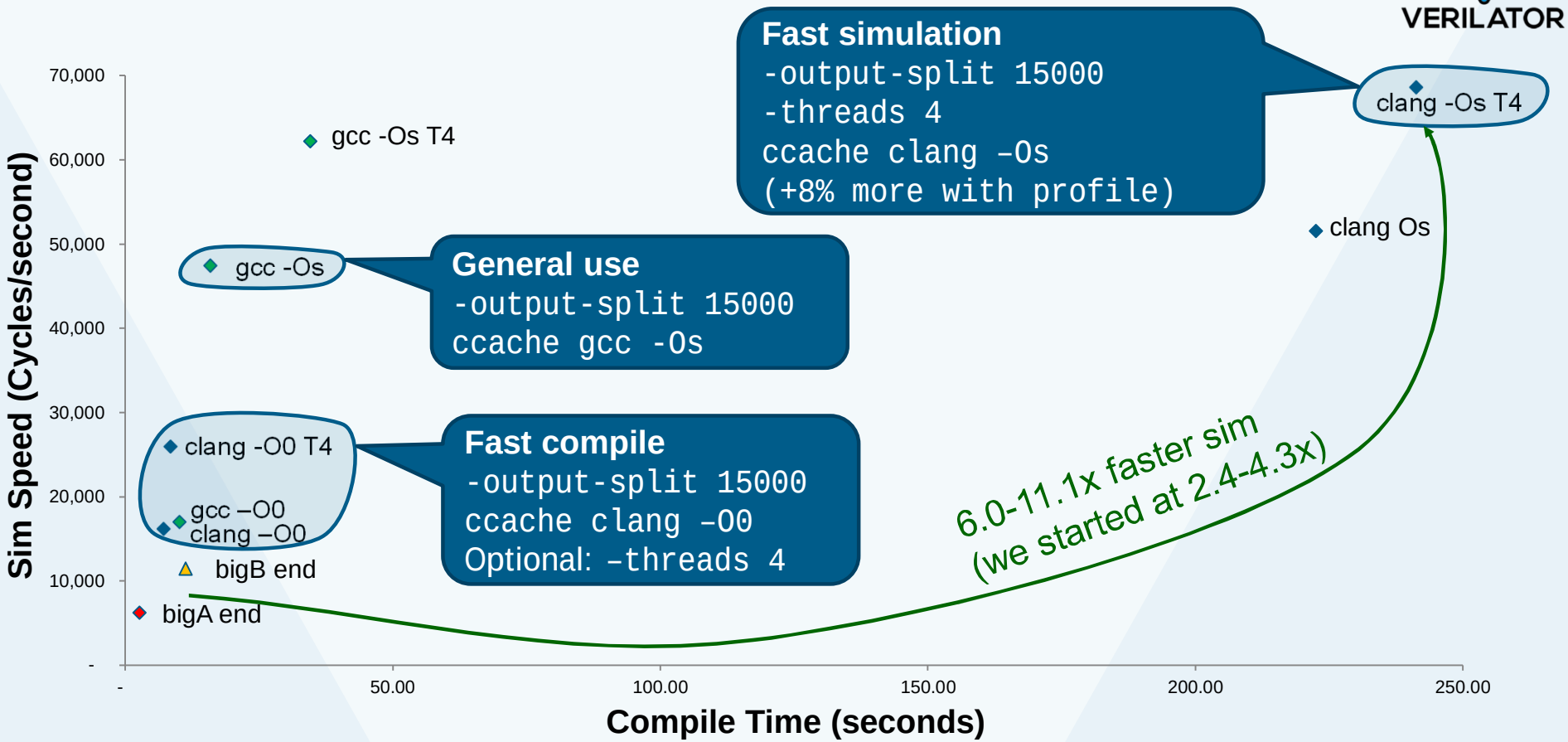
Profile-driven thread feedback optimization

Discovery: Clang Threading with –O0



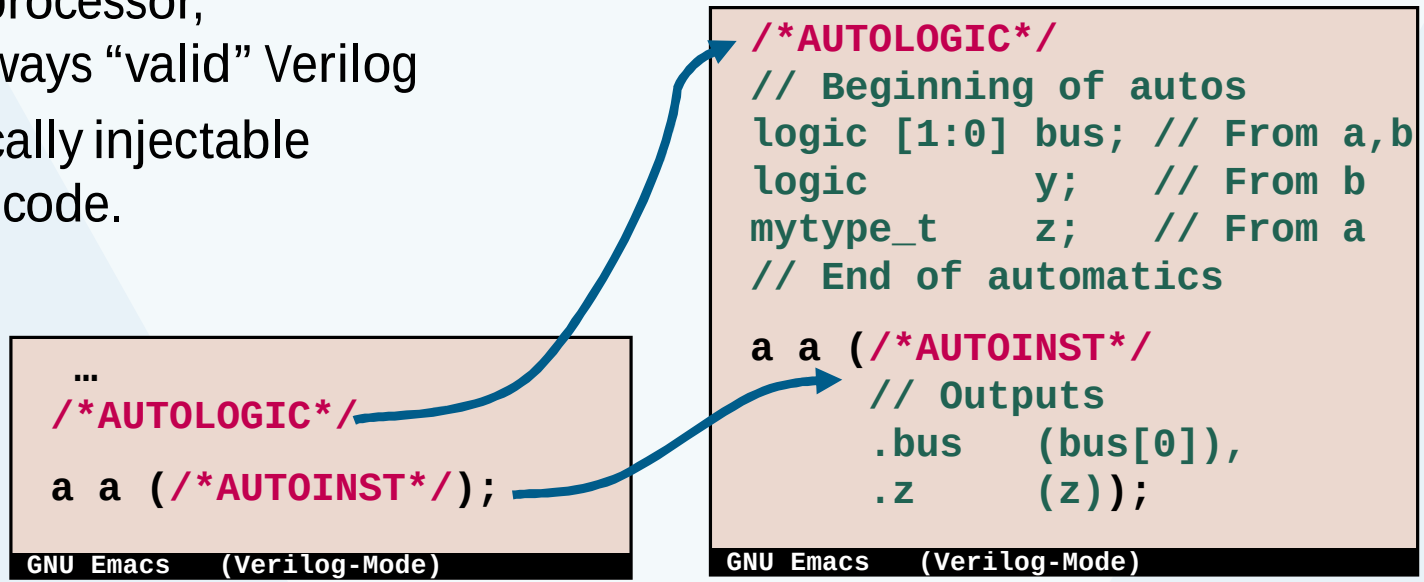
Conclusion

Conclusion: Choose Your Operating Point



Recommended: Verilog-Mode for Emacs

- Thousands of users, including most IP houses
- Fewer lines of code to edit means fewer bugs
- Indents code correctly, too
- Not a preprocessor,
  code is always “valid” Verilog
- ☆ Automatically injectable
  into older code.



Resources

- Verilator and open source design tools at http://www.veripool.org
  - Downloads
  - Bug Reporting
  - User Forums
  - News (add yourself as a watcher to see releases)
  - Presentations at http://www.veripool.org/papers/

Thanks to all past and future contributors!


VERILATOR