VERIPOOL

VERILATOR

ORConf 2018

Verilator 4.0:
Open Simulation Goes Multithreaded

Wilson Snyder

Veripool.org

wsnyder@wsnyder.org

Updated 2018-10-22

Agenda

- Introduction to Verilator
- Version 4
- Multithreading
- Benchmarking
- Conclusion
- Q & A   (Please hold questions until end)

Introduction to Verilator

Verilator is a Compiler

- Verilator compiles synthesizable Verilog into C++
  - Matches synthesis rules, not simulation rules
  - Time delays ignored (a <= #{n} b;)
  - Only two state simulation (and tri-state busses)
  - Unknowns are randomized (better than Xs)

- Creates C++/SystemC wrapper

- Creates own internal interconnect
  - Plays several tricks to get good, fast code

Example: Linting a Verilog Module

- Lint check your code

```
verilator –lint-only -Wall Convert.v
```

(Verilator is the only open-source Verilog Lint tool known to the presenter)

Convert.v

```
module Convert;
   input clk
   input [31:0] data;
   output [31:0] out;

   initial $display(“Hello flip-flop”);
   always_ff @ (posedge clk)
      out = data;
endmodule
```

```
%Warning-BLKSEQ, Blocking
assignments (=) in sequential
(flop or latch) block; suggest
delayed assignments (<=).
```

Example: Translation

- Translate to a C++ class (also can do SystemC module – not shown)

```
verilator -cc Convert.v
```

Convert.v

```
module Convert;
   input clk
   input [31:0] data;
   output [31:0] out;

   initial $display(“Hello flip-flop”);
   always_ff @ (posedge clk)
      out <= data;
endmodule
```

obj_dir/VConvert.h (made by Verilator)

```
#include "verilated.h"

class VConvert {
   bool      clk;
   uint32_t  data;
   uint32_t  out;

   void eval();
   void final();
}
```

- Verilog top module becomes a C++ class (obj_dir/V*Convert*.h)
- Inputs and outputs map directly to basic C++ types (not objects)

Verilator in Standard IP

- From the world’s most popular cellphone CPU IP:

```
reg clk_en_lat /*verilator clock_enable*/;
```

  - Verilation works “out-of-the-box”
  - Thanks!

- Supported by many open-source HW IPs
  - Thanks!

- Under common use internally in other major CPU design centers
  - Thanks!

Verilator Version 4

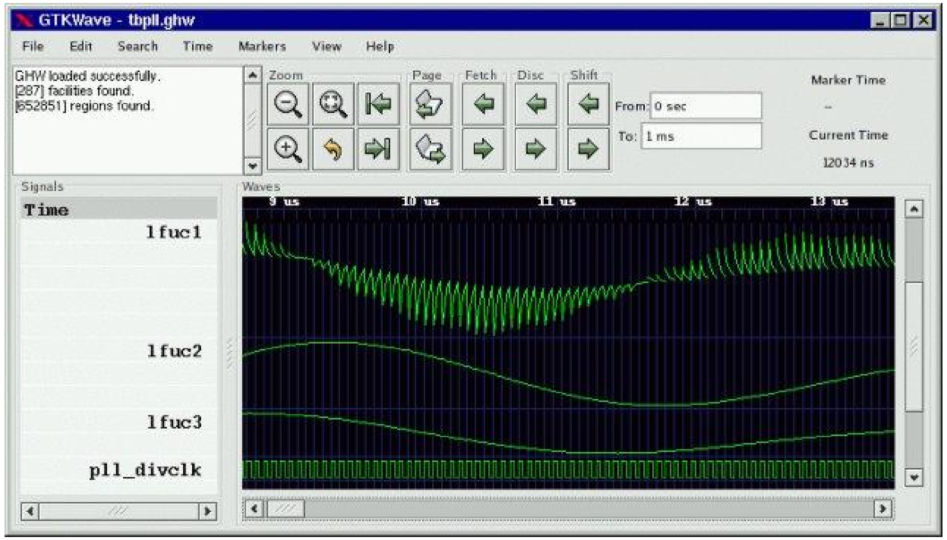
Thanks to all whom have contributed

Ahmed El-Mahmoudy, David Addison, Tariq B. Ahmad, Nikana Anastasiadis, Hans Van Antwerpen, Vasu Arasanipalai, Jens Arm, Sharad Bagri, Andrew Bardsley, Matthew Barr, Geoff Barrett, Julius Baxter, Jeremy Bennett, Michael Berman, Victor Besyakov, David Binderman, Johan Bjork, David Black, Tymoteusz Blazejczyk, Daniel Bone, Gregg Bouchard, Christopher Boumenot, Nick Bowler, Byron Bradley, Bryan Brady, Charlie Brej, J Briquet, Lane Brooks, John Brownlee, Jeff Bush, Lawrence Butcher, Ted Campbell, Chris Candler, Lauren Carlson, Donal Casey, Sebastien Van Cauwenberghe, Terry Chen, Enzo Chi, Robert A. Clark, Allan Cochrane, John Coiner, Laurens van Dam, Gunter Dannoritzer, Ashutosh Das, Bernard Deadman, John Demme, Mike Denio, John Deroo, Philip Derrick, Joe DErrico, John Dickol, Ruben Diez, Danny Ding, Ivan Djordjevic, Jonathon Donaldson, Sebastian Dressler, Alex Duller, Jeff Dutton, Usuario Eda, Chandan Egbert, Joe Eiler, Ahmed El-Mahmoudy, Trevor Elbourne, Robert Farrell, Eugen Fekete, Fabrizio Ferrandi, Brian Flachs, Andrea Foletto, Bob Fredieu, Duane Galbi, Christian Gelinek, Glen Gibb, Shankar Giri, Dan Gisselquist, Sam Gladstone, Amir Gonnen, Chitlesh Goorah, Xuan Guo, Neil Hamilton, Jannis Harder, Junji Hashimoto, Thomas Hawkins, Robert Henry, David Hewson, Jamey Hicks, Joel Holdsworth, Hiroki Honda, Alex Hornung, David Horton, Jae Hossell, Alan Hunter, James Hutchinson, Jamie Iles, Ben Jackson, Shareef Jalloq, Krzysztof Jankowski, HyungKi Jeong, Iztok Jeras, James Johnson, Christophe Joly, Franck Jullien, James Jung, Mike Kagen, Arthur Kahlich, Kaalia Kahn, Guy-Armand Kamendje, Vasu Kandadi, Patricio Kaplan, Ralf Karge, Dan Katz, Sol Katzman, Jonathan Kimmitt, Olof Kindgren, Dan Kirkham, Sobhan Klnv, Gernot Koch, Soon Koh, Steve Kolecki, Brett Koonce, Wojciech Koszek, Varun Koyyalagunta, David Kravitz, Roland Kruse, Sergey Kvachonok, Ed Lander, Steve Lang, Stephane Laurent, Walter Lavino, Christian Leber, Igor Lesik, John Li, Eivind Liland, Yu Sheng Lin, Charlie Lind, Andrew Ling, Paul Liu, Derek Lockhart, Arthur Low, Stefan Ludwig, Dan Lussier, Fred Ma, Duraid Madina, Julien Margetts, Mark Marshall, Alfonso Martinez, Yves Mathieu, Patrick Maupin, Jason McMullan, Elliot Mednick, Wim Michiels, Miodrag Milanovic, Wai Sum Mong, Sean Moore, Dennis Muhlestein, John Murphy, Richard Myers, Dimitris Nalbantis, Bob Newgard, Cong Van Nguyen, Paul Nitza, Pete Nixon, Lisa Noack, Mark Nodine, Andreas Olofsson, James Pallister, Brad Parker, Maciej Piechotka, David Pierce, Dominic Plunkett, David Poole, Mike Popoloski, Rich Porter, Niranjan Prabhu, Usha Priyadharshini, Mark Jackson Pulver, Prateek Puri, Marshal Qiao, Chris Randall, Anton Rapp, Josh Redford, Odd Magne Reitan, Frederic Requin, Alberto Del Rio, Oleg Rodionov, Paul Rolfe, Arjen Roodselaar, Jan Egil Ruud, John Sanguinetti, Galen Seitz, Salman Sheikh, Mike Shinkarovsky, Rafael Shirakawa, Jeffrey Short, Rodney Sinclair, Steven Slatter, Brian Small, Wilson Snyder, Alex Solomatnikov, Wei Song, Art Stamness, John Stevenson, Patrick Stewart, Rob Stoddard, Todd Strader, John Stroebel, Sven Stucki, Emerson Suguimoto, Gene Sullivan, Renga Sundararajan, Yutetsu Takatsukasa, Peter Tengstrand, Wesley Terpstra, Rui Terra, Stefan Thiede, Gary Thomas, Kevin Thompson, Ian Thompson, Mike Thyer, Hans Tichelaar, Steve Tong, Michael Tresidder, Holger Waechtler, Stefan Wallentowitz, Shawn Wang, Paul Wasson, Greg Waters, Thomas Watts, Eugene Weber, David Welch, Thomas J Whatson, Leon Wildman, Gerald Williams, Trevor Williams, Jeff Winston, Joshua Wise, Clifford Wolf, Johan Wouters, Junyi Xi, Ding Xiaoliang, Jie Xu, Mandy Xu, Luke Yang, and Amir Yazdanbakhsh.

Thanks to them, and any we’ve missed including, and any who whished to remain anonymous.

New in v4

- Single threaded performance enhancements
- Standard runtime option parsing
- Natively write FST files for GTKWave viewing
- Multithreading (later slides)

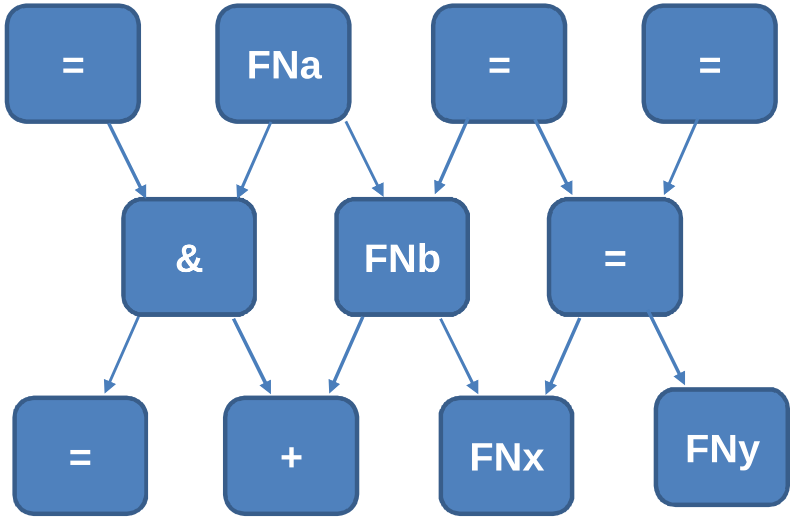


GHW loaded successfully.
[287] facilities found.
[652851] regions found.

Multithreading

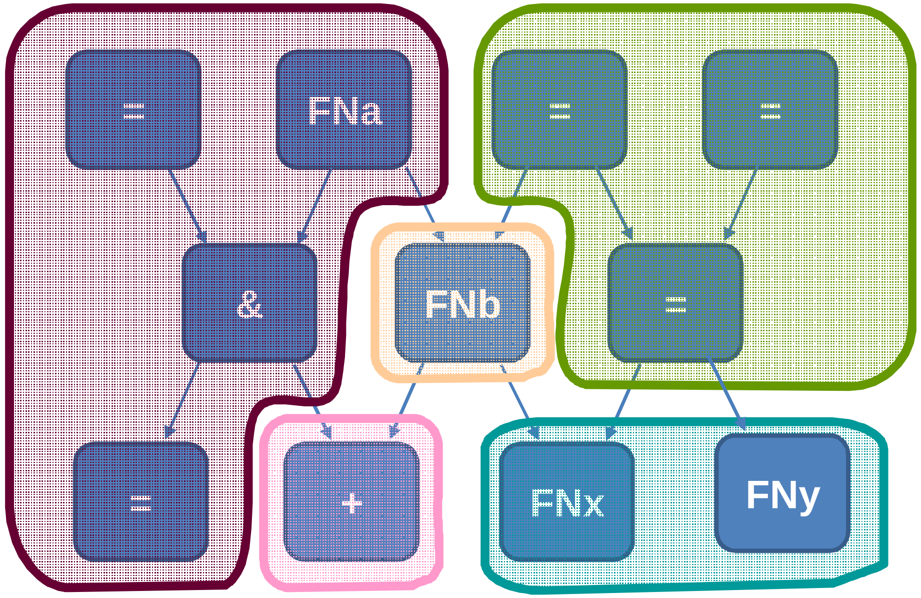
Multithreading Algorithm Begins

Begin with graph of millions of statement-level “micro tasks”...



Coarsen the DAG

Color nodes into tens of "Macro Tasks"

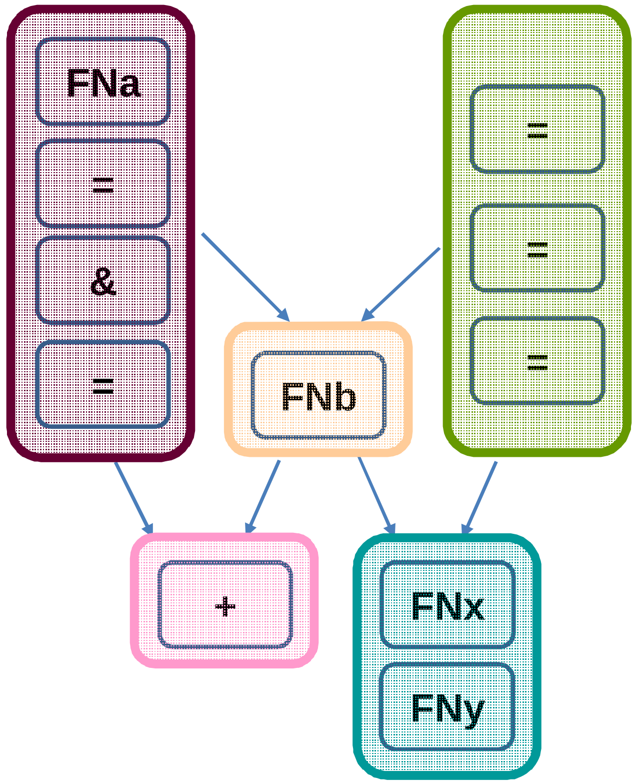


Merge micro tasks, only macro tasks remain

Merge micro tasks so only macro tasks remain

Typical result: *millions* of micro-tasks merge into *tens* of macro-tasks

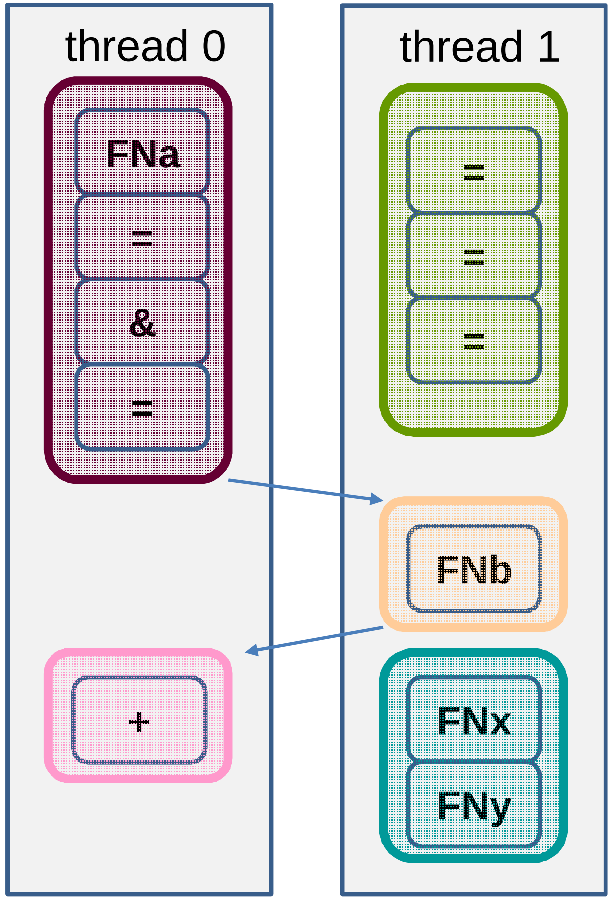
Runtime synchronization becomes affordable



Schedule macro tasks onto threads

Schedule macro tasks onto threads

- Static assignment, using guess of runtime
- Future improvement to reassign dynamically using feedback
- Every-cycle dynamic runtime assignment was tried but slow! Future research area



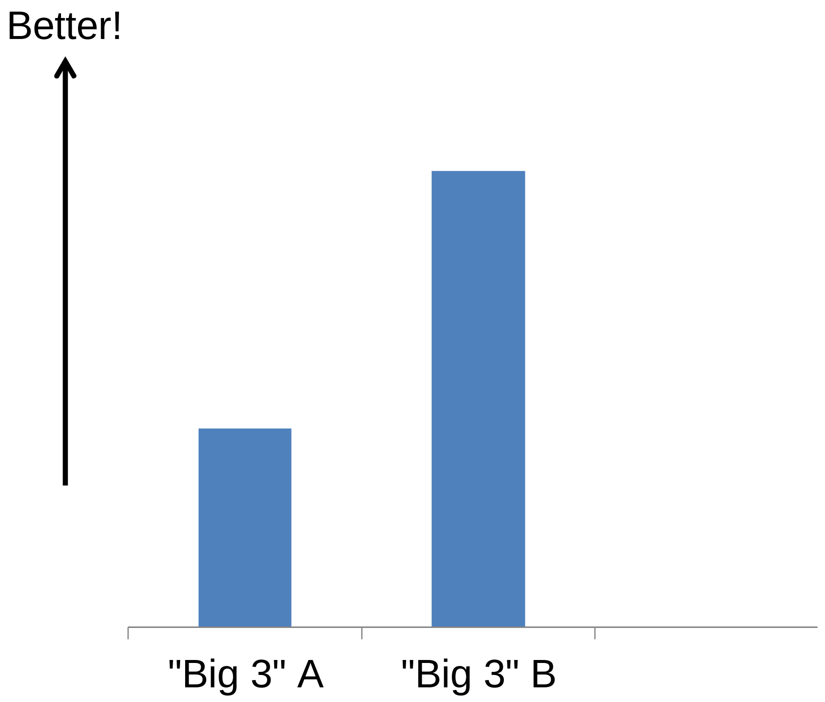
Benchmarking

Benchmarks

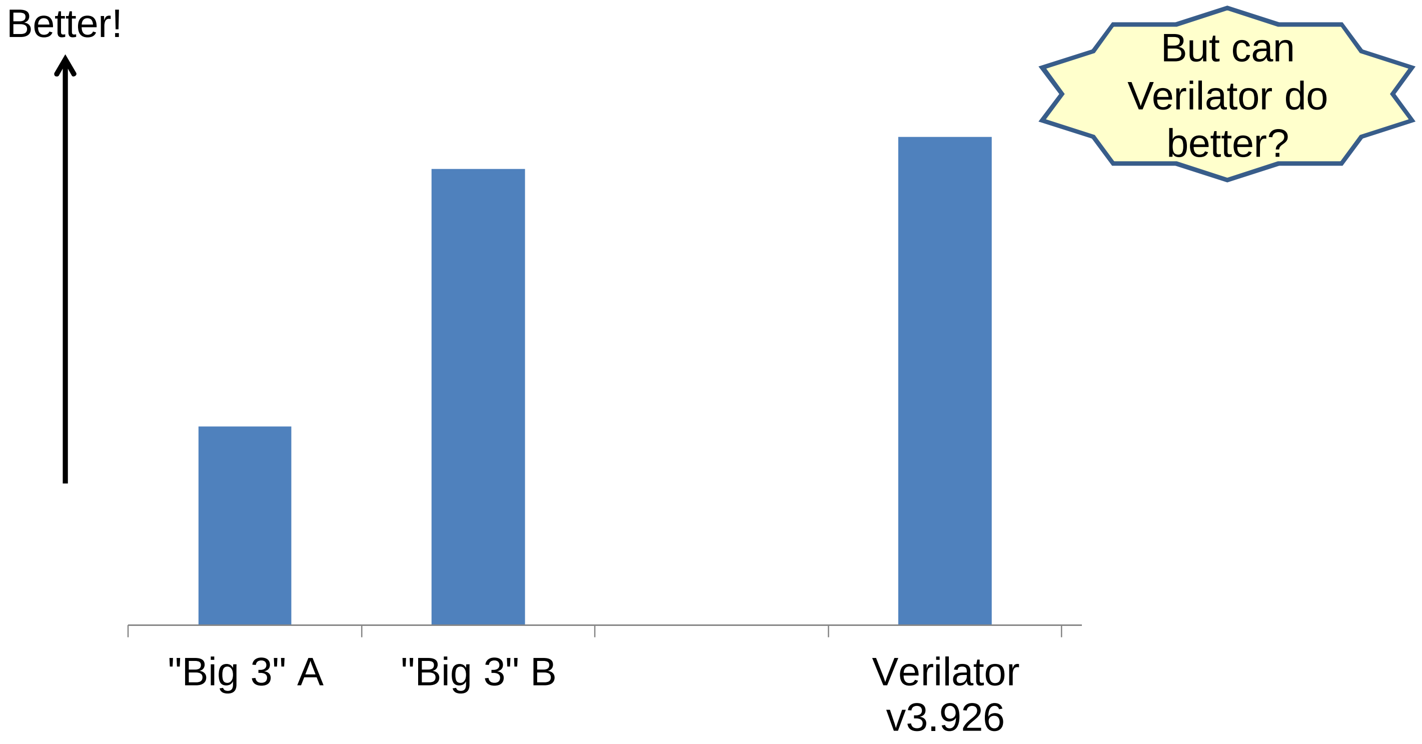
- Packet Architects 1.6 Tbit Router IP
  - (Thanks to them for allowing publication of this data)
- Verilog compiled out-of-the-box
  - No specialized tuning, no hints
- Your mileage will vary…

Packet Architects AB

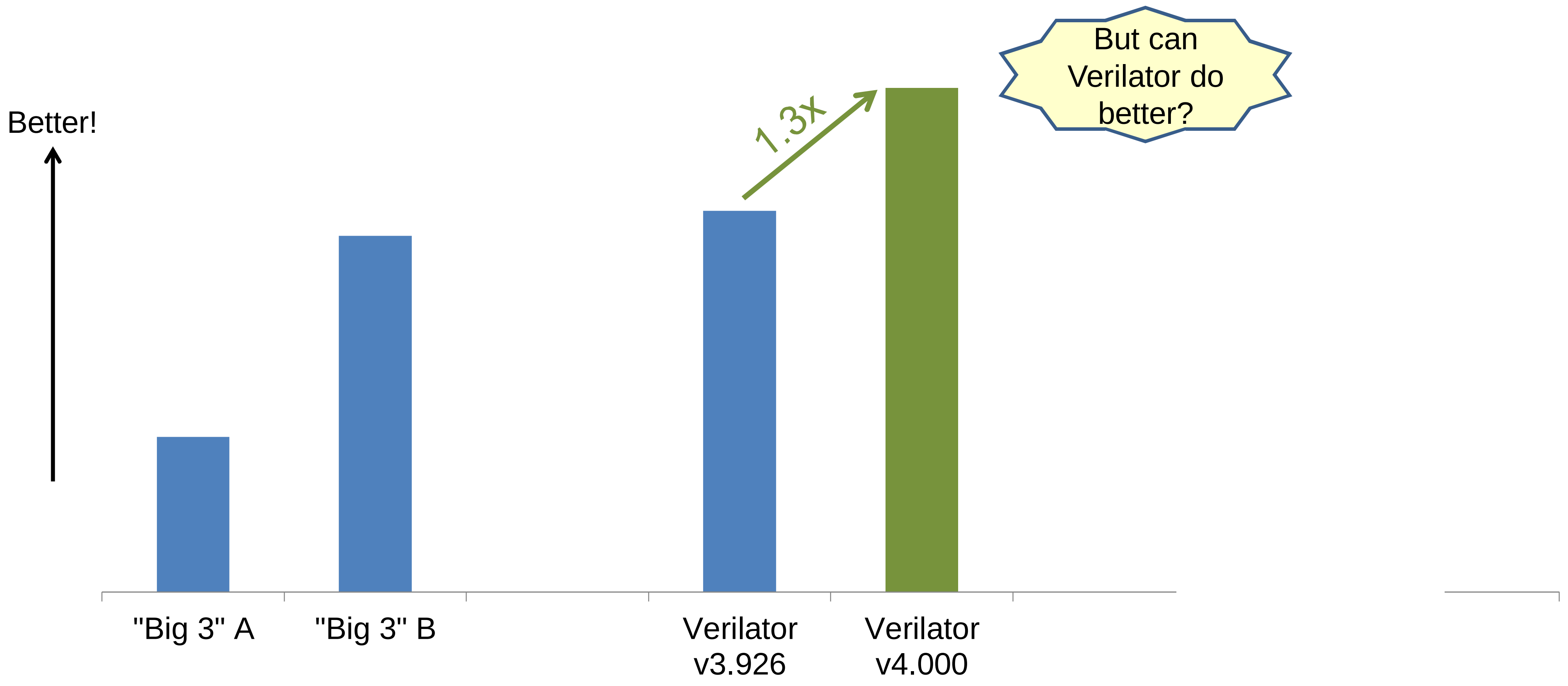
Verilator Performance vs. Big 3



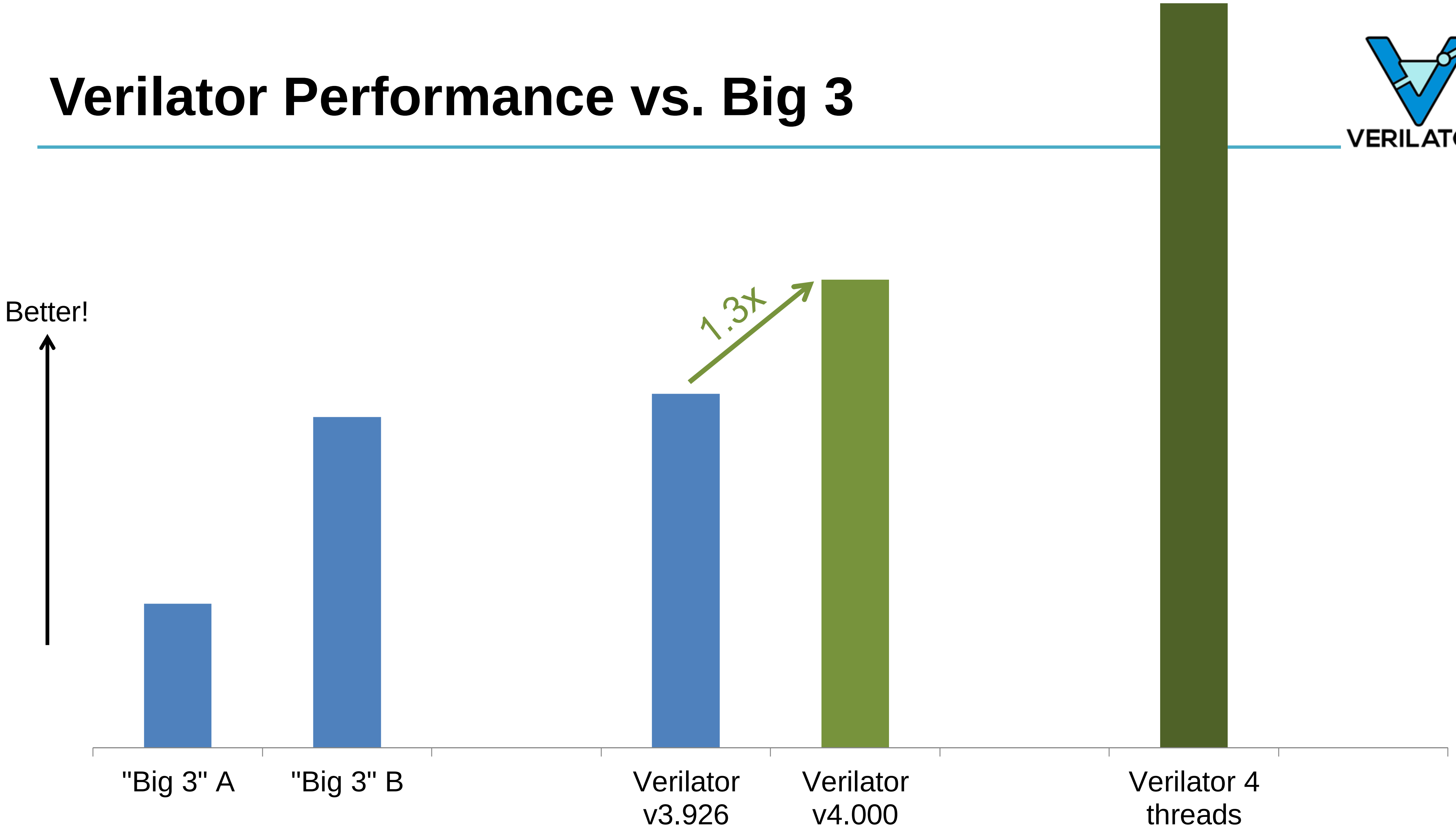
Verilator Performance vs. Big 3



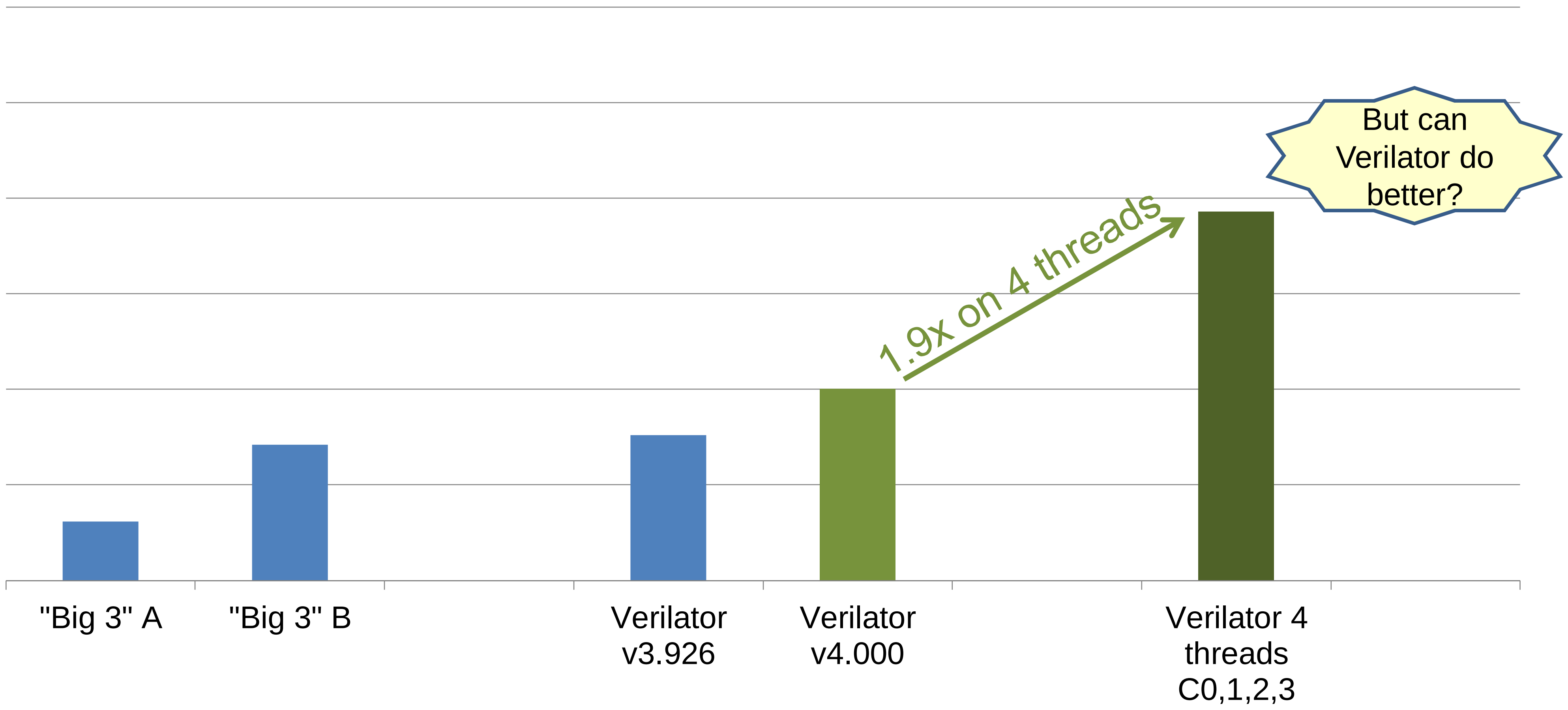
Verilator Performance vs. Big 3



Verilator Performance vs. Big 3



Verilator Performance vs. Big 3



Beware Which Cores Are Used

```
$ lscpu
Architecture:          x86_64
CPU(s):                8
Socket(s):             2
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              16896K
NUMA node0 CPU(s):     0,2,4,6
NUMA node1 CPU(s):     1,3,5,7
```

Got 4 threads but crossing sockets

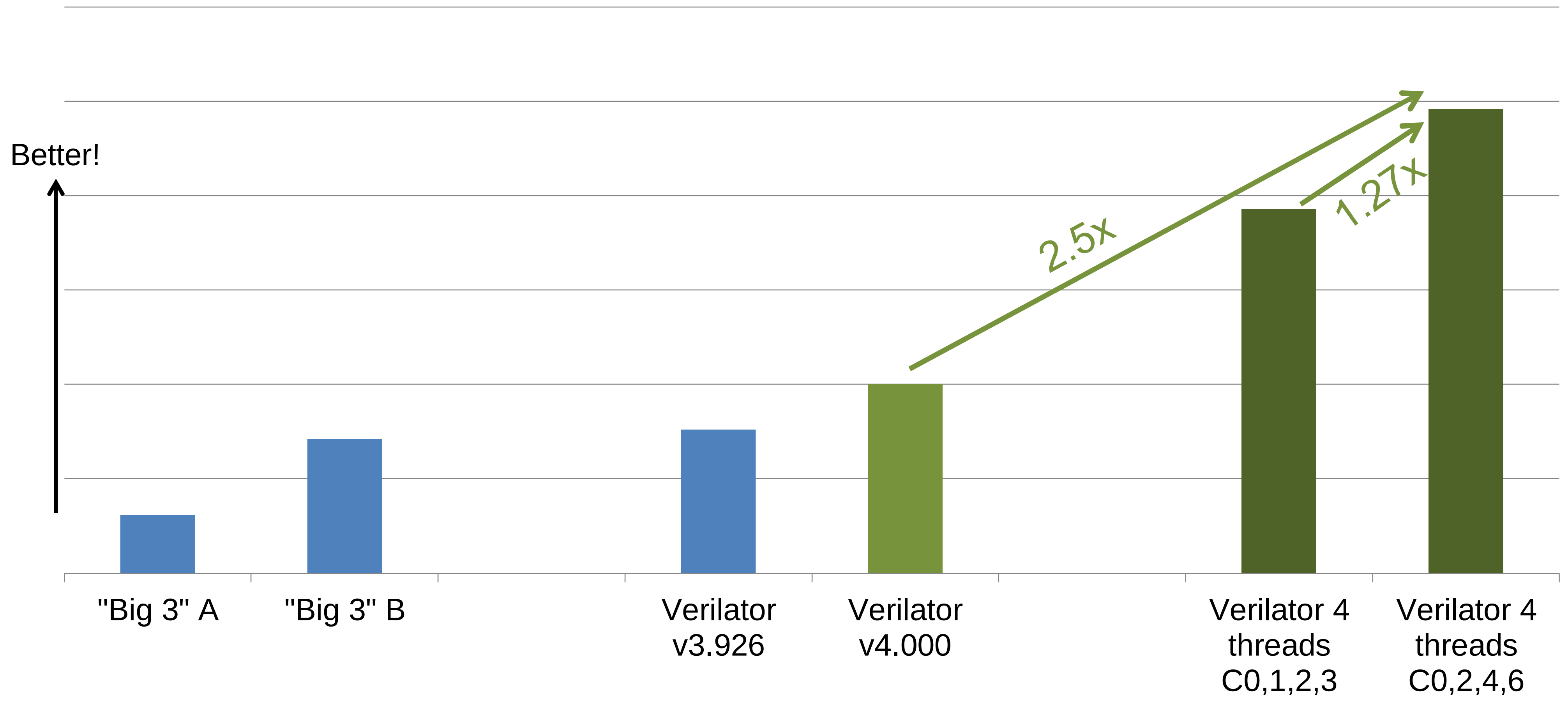
```
NUMA node0 CPU(s):     0,2,4,6
NUMA node1 CPU(s):     1,3,5,7
```

Want 4 threads on same socket

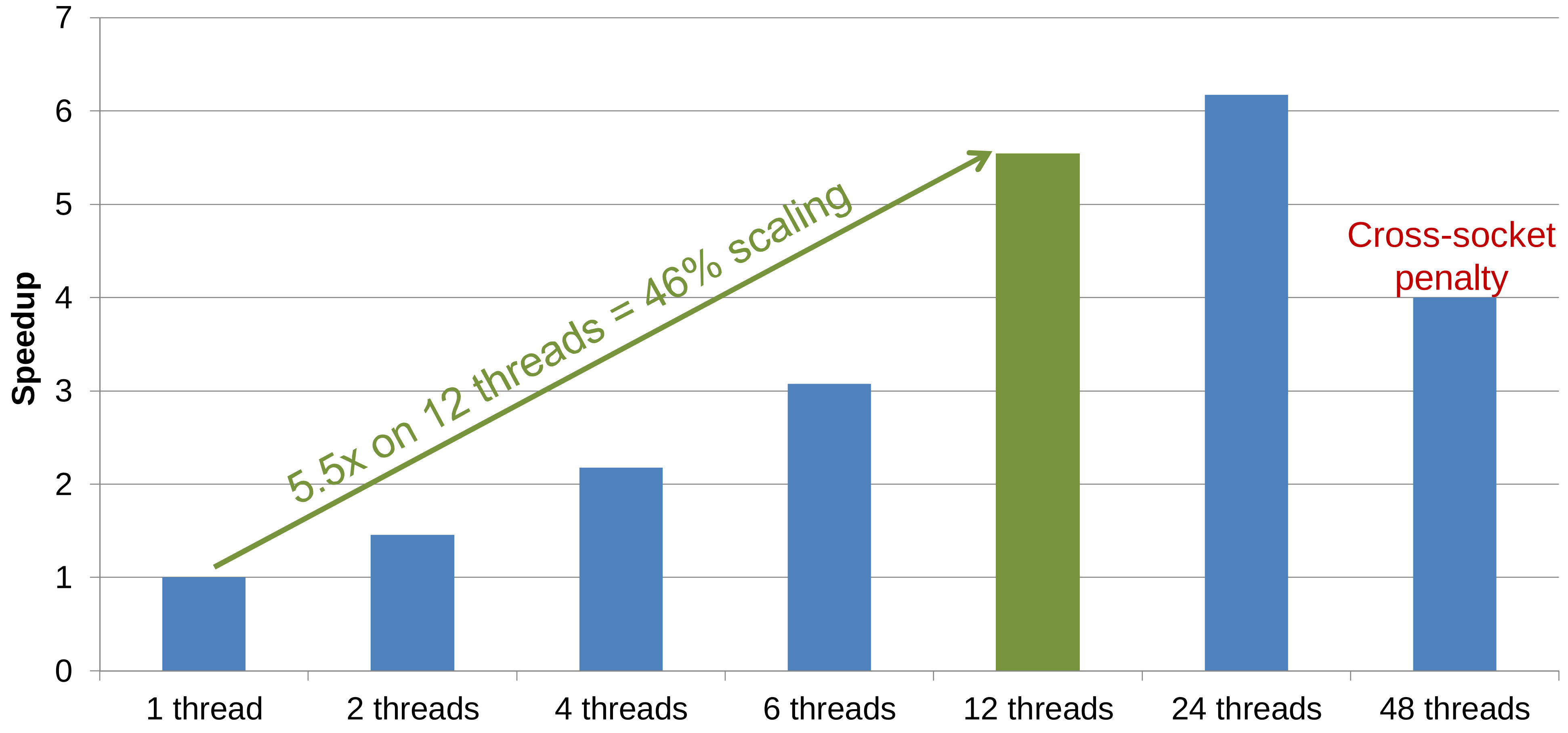
Similar issue may exist getting two hyperthreaded CPUs on same physical CPU. Thus always use numactl.

```
$ numactl –C 0,2,4,6 sim_executable
```

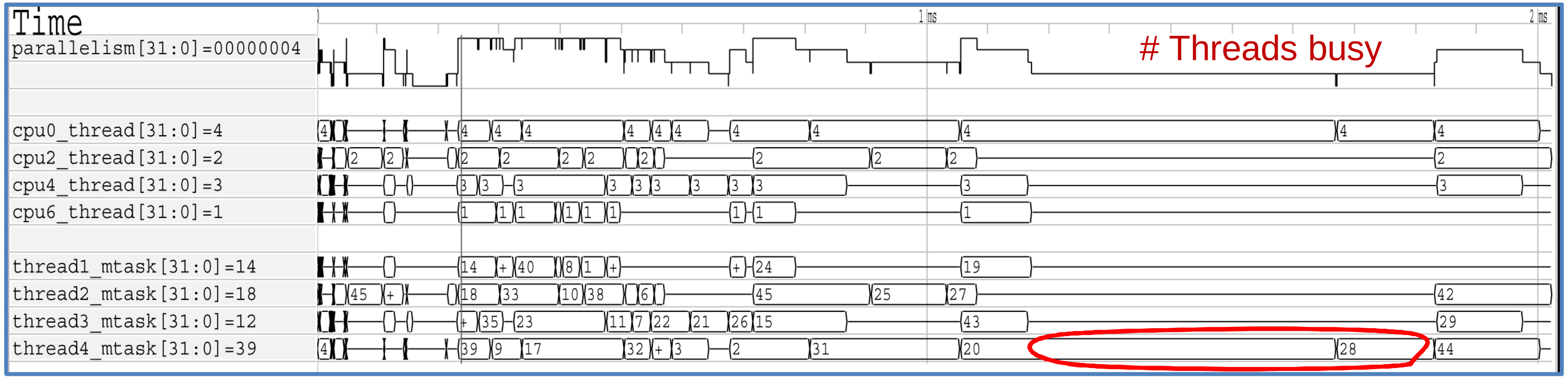
Verilator Performance vs. Big 3



Verilator Multithreaded Performance on 48 core machine



Verilator Gantt Report (4 threads)



```
Analysis:
  Total threads                = 4
  Total mtasks                 = 45
  Total eval time              = 2021242 rdtsc ticks
  Longest mtask time           = 614664 rdtsc ticks
  All-thread mtask time        = 4212854 rdtsc ticks
  Longest-thread efficiency    = 30.4%
  All-thread efficiency        = 52.1%
  All-thread speedup           = 2.1
```

Long period of single thread active;
Mtask 20 is taking too long.
Area for future improvements/tuning

Conclusion

Conclusion: Adopt Verilator

- Supported
  - Continual language improvements
  - Growing support network for 20+ years
  - Run faster than major simulators
- Open Source Helps You
  - Easy to run on laptops or SW developer machines
  - Get bug fixes in minutes rather than months
  - Greatly aids commercial license negotiation
- Keep your Commercial Simulators
  - SystemVerilog Verification, analog models, gate SDF, etc.

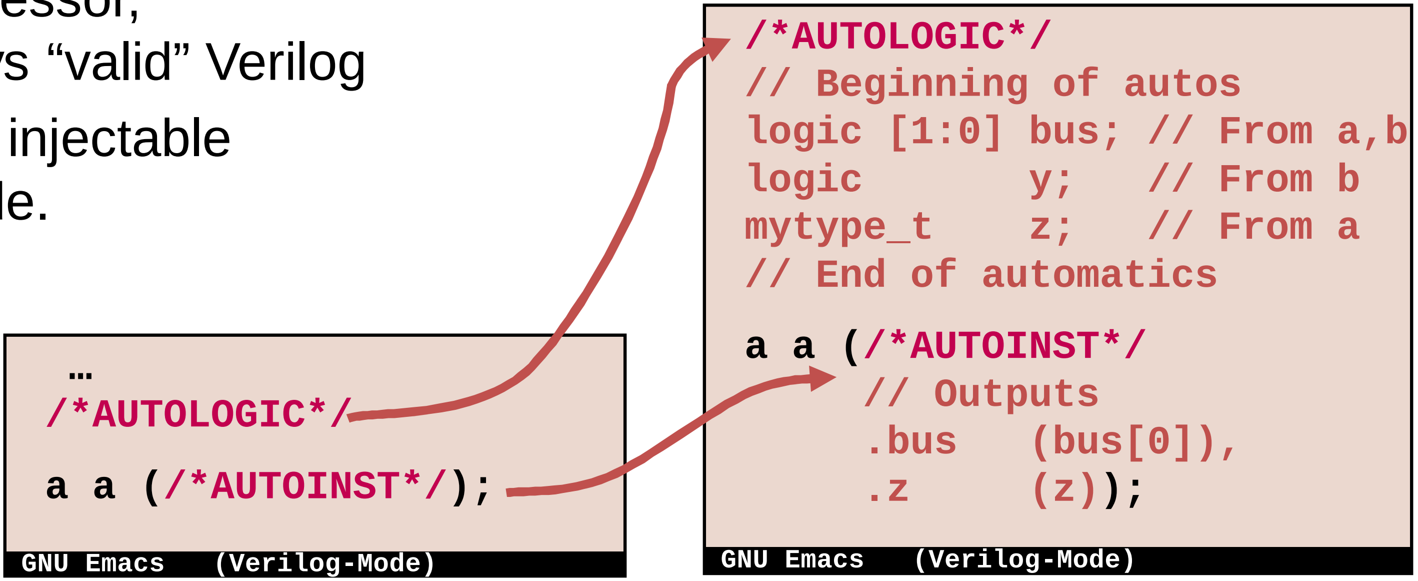
Contributing Back

- The value of Open Source is in the Community!
- Use Forums and Bug Reporting
- Try to submit a patch yourself
  - Many problems take only a few hours to resolve yourself; often less time than packaging up a test case for an EDA company!
  - Even if just documentation fixes!
  - Great experience for the resume!
- Advocate

Also at Veripool: 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.



```
 …
/*AUTOLOGIC*/

a a (/*AUTOINST*/);
```

```
/*AUTOLOGIC*/
// Beginning of autos
logic [1:0] bus; // From a,b
logic       y;   // From b
mytype_t    z;   // From a
// End of automatics

a a (/*AUTOINST*/
     // Outputs
     .bus   (bus[0]),
     .z     (z));
```

Sources

- 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)
  - These slides at http://www.veripool.org/papers/