A Fast Instruction Set Simulator for RISC-V

Maxim.Maslov@esperantotech.com
Vadim.Gimpelson@esperantotech.com
Nikita.Voronov@esperantotech.com
Dave.Ditzel@esperantotech.com

Esperanto Technologies, Inc.

5th RISC-V Workshop
November 30, 2016

Background

Esperanto is a stealth mode startup designing chips with RISC-V.

Esperanto wanted a fast RISC-V ISA simulator capable of:

- Running large applications with minimal (<5x) slowdown
- Running large number of threads with good scalability
- Providing flexibility in testing instruction extensions
- Providing flexibility in gathering performance data

Fast simulation is a key productivity tool

- Gives a fast compile, run and test/debug loop prior to silicon
- Current simulators were judged to be too slow
- Undertook project to modify Eltechs ExaGear to run RISC-V

Motivation

Evaluated two existing options: QEMU and Spike

Would either be sufficient?

We compiled several tests from Spec2006 benchmark both for x86-64 and RISC-V with the same compiler version and options: GCC 6.1.0 –O2

Comparison of native, QEMU and Spike runtimes

Benchmark	x86-64 native (in seconds)	QEMU system mode (in seconds)	Spike pk (in seconds)
099.go	8.6	585	2,294
401.bzip2	475	Failed	41,043
410.bwaves	444	Failed	48,859
416.gamess	66*	Failed	16,045
445.gobmk	64*	2145	10,778
435.gromacs	464	26040	Failed

* Only one subtask

Result of Spike and QEMU evaluation

- Spike is a very slow simulator: 86x – 267x times slower than native
- Spike user mode (pk) requires static binaries – not applicable in some cases
- QEMU system mode was not fast enough:
  - 34x – 68x times slower than native
- QEMU had no user mode support for RISC-V when we started the evaluation

Fast Simulator Design

User mode approach

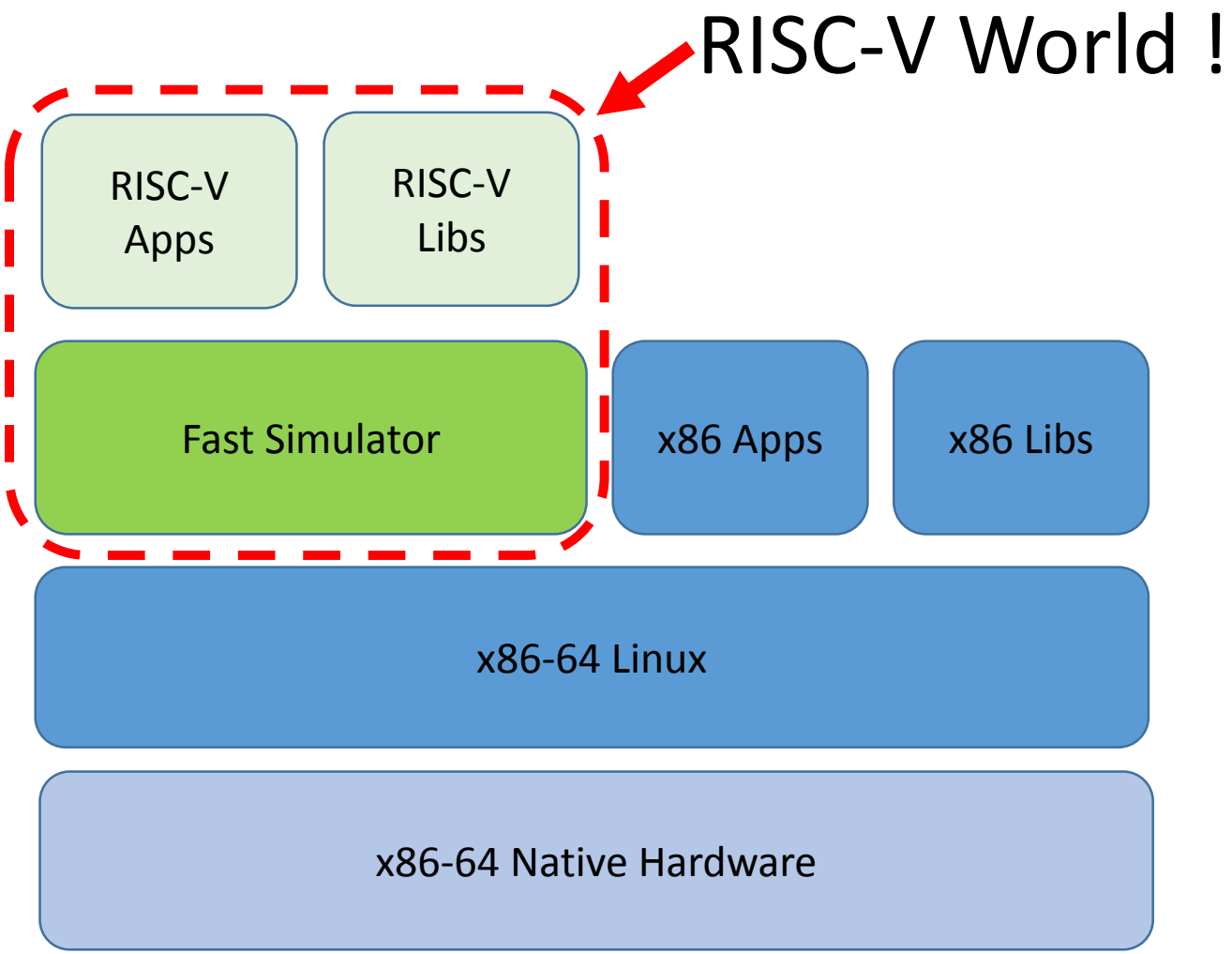
"User mode" emulation means we only need to run the RISC-V instructions in the application, not the OS.

The OS still runs compiled to the native hardware.

User mode approach advantages:

- Faster emulation (does not need sophisticated software MMU techniques)
- Does not need stable RISC-V kernel right now
- Kernel code does not blur performance results

User mode approach

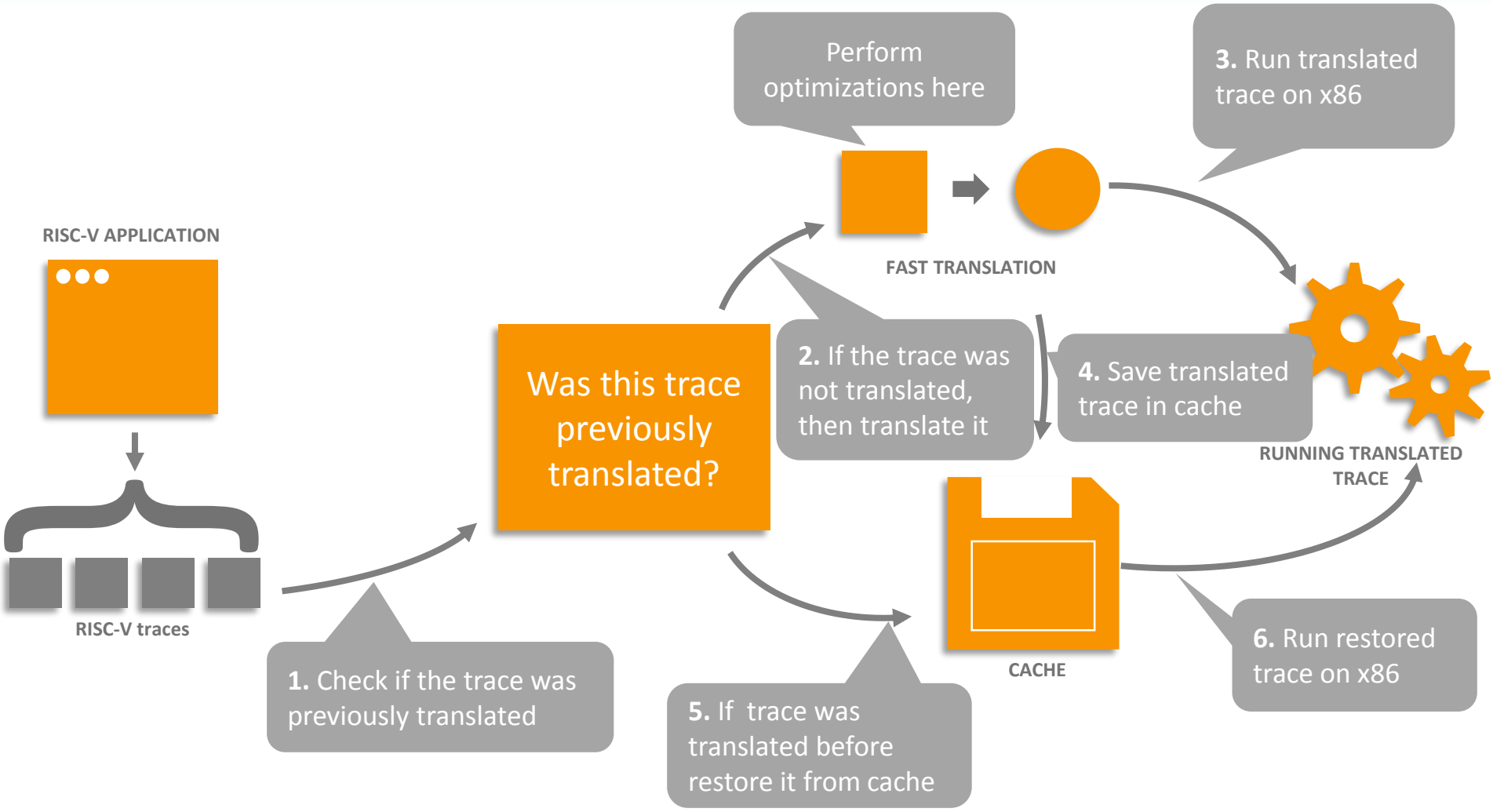


User mode key points

- Fast Simulator can only execute Linux RISC-V ELF-binaries with user mode
- RISC-V code is translated into corresponding x86-64 instructions
- RISC-V Linux system calls are translated into corresponding x86-64 Linux system calls
- RISC-V applications can only see a RISC-V world

  (some kind of chroot), but can communicate with host via kernel (for example, using sockets)

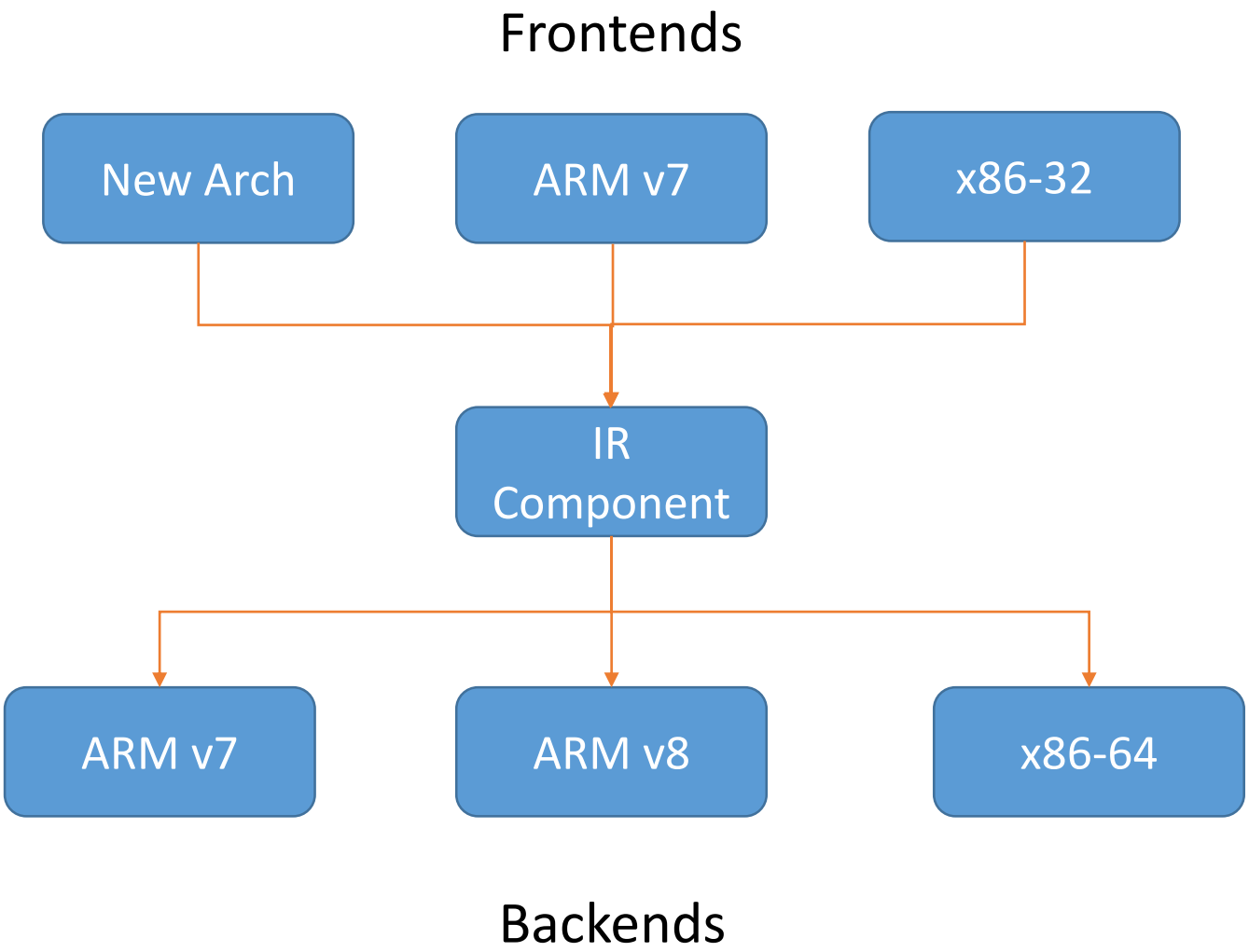
Fast Simulator Translation Flow



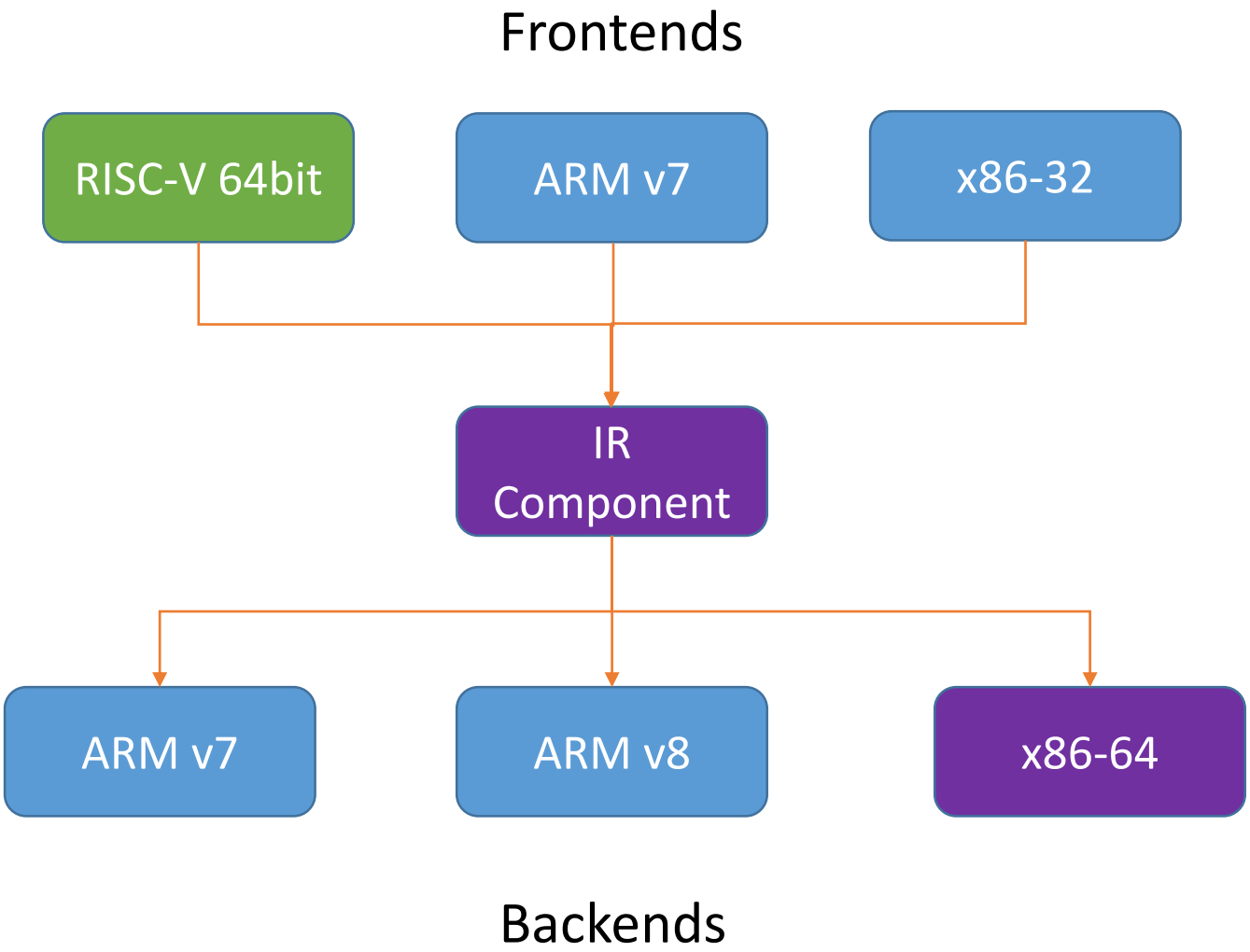
Fast Simulator Translation Flow Key Points

- Fast Simulator translates traces
- All compiled traces are saved in a Translation Cache and are then reused
- Several optimizations are applied, including efficient register allocation, peephole optimization, dynamic jump cache, etc.
- Floating point calculations directly use hardware FPU and FPU registers

Fast Simulator Design



Fast Simulator Design



Fast Simulator Design Benefits

- x86-64 backend was implemented previously and we just reused it
- Intermediate Representation (IR) was also reused
- All reused components are very reliable due to exhaustive testing in previous products
- For first reasonable version we had to implement only RISC-V frontend and tune other components
- Many optimizations worked without changes

Development time frame

- About 2 months for a first reliable version which is able to run full Spec2006 benchmark
- About one month of performance tuning the optimizations to get the presented numbers

As a Result

- Fast development
- High reliability
- High performance

Performance results

Performance results

- Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz
- GCC 6.1.0, –O2 optimization level
- RISC-V toolchain available on 24 OCT 2016
- QEMU 2.7.50 (RISC-V user mode appears!)
- Spec2006 benchmarks

CINT2006
Fast Sim/x86-64 performance comparison

	x86-64	Fast Sim	
Spec name	(in seconds)	(in seconds)	Ratio
400.perlbench	359	1168	3.25
401.bzip2	475	1188	2.50
403.gcc	318	890	2.80
429.mcf	376	469	1.25
445.gobmk	447	1473	3.30
456.hmmer	427	1363	3.19
458.sjeng	494	1745	3.53
462.libquantum	550	787	1.43
464.h264ref	537	1986	3.70
471.omnetpp	383	719	1.88
473.astar	401	721	1.80
483.xalancbmk	301	828	2.75
GeoMean			**2.47**

CFP2006
Fast Sim/x86-64 performance comparison

Spec name	x86-64 (in seconds)	Fast Sim (in seconds)	Ratio
410.bwaves	444	1213	2.73
416.gamess	66	235	3.56
433.milc	441	1134	2.57
434.zeusmp	411	1938	4.72
435.gromacs	464	1360	2.93
436.cactusADM	763	4756	6.23
437.leslie3d	432	1595	3.69
444.namd	363	1263	3.48
447.dealII	310	956	3.08
450.soplex	314	667	2.12
453.povray	169	737	4.36
454.calculix	710	4149	5.84
459.GemsFDTD	516	1126	2.18
465.tonto	656	2126	3.24
470.lbm	490	1244	2.54
481.wrf	589	2280	3.87
482.sphinx3	733	3704	5.05
GeoMean			**3.48**

CINT2006
QEMU user mode/Fast Sim performance comparison

Spec name	Fast Sim (in seconds)	QEMU user mode (in seconds)	Ratio
400.perlbench	1168	3334	2.85
401.bzip2	1188	1478	1.24
403.gcc	890	1766	1.98
429.mcf	469	506	1.08
445.gobmk	1473	2385	1.62
456.hmmer	1363	1609	1.18
458.sjeng	1745	3042	1.74
462.libquantum	787	921	1.17
464.h264ref	1986	4492	2.26
471.omnetpp	719	2050	2.85
473.astar	721	971	1.35
483.xalancbmk	828	2060	2.49
GeoMean			**1.71**

CFP2006
QEMU user mode/Fast Sim performance comparison

Spec name	Fast Sim (in seconds)	QEMU user mode (in seconds)	Ratio
410.bwaves	1213	10127	8.35
416.gamess	235	1362	5.80
433.milc	1134	9306	8.21
434.zeusmp	1938	10104	5.21
435.gromacs	1360	16519	12.15
436.cactusADM	4756	24871	5.23
437.leslie3d	1595	9292	5.83
444.namd	1263	17074	13.52
447.dealII	956	4987	5.22
450.soplex	667	1739	2.61
453.povray	737	4325	5.87
454.calculix	4149	47720	11.50
459.GemsFDTD	1126	12004	10.66
465.tonto	2126	16752	7.88
470.lbm	1244	14201	11.42
481.wrf	2280	17689	7.76
482.sphinx3	3704	24365	6.58
GeoMean			**7.29**

CINT2006
Spike pk/Fast Sim performance comparison

	Fast Sim	Spike pk	
Spec name	(in seconds)	(in seconds)	Ratio
400.perlbench	1168	55111	47.18
401.bzip2	1188	41040	34.55
403.gcc	890	failed	
429.mcf	469	4762	10.15
445.gobmk	1473	71583	48.60
456.hmmer	1363	32425	23.79
458.sjeng	1745	102748	58.88
462.libquantum	787	10245	13.02
464.h264ref	1986	failed	
471.omnetpp	719	19965	27.77
473.astar	721	11651	16.16
483.xalancbmk	828	failed	
GeoMean			**26.56**

CFP2006
Spike pk/Fast Sim performance comparison

Spec name	Fast Sim (in seconds)	Spike pk (in seconds)	Ratio
410.bwaves	1213	48859	40.28
416.gamess	235	16045	68.28
433.milc	1134	29221	25.77
434.zeusmp	1938	35506	18.32
435.gromacs	1360	failed	
436.cactusADM	4756	199003	41.84
437.leslie3d	1595	26769	16.78
444.namd	1263	44916	35.56
447.dealII	956	24788	25.93
450.soplex	667	7861	11.79
453.povray	737	37731	51.20
454.calculix	4149	159790	38.51
459.GemsFDTD	1126	30716	27.28
465.tonto	2126	91514	43.05
470.lbm	1244	39644	31.87
481.wrf	2280	failed	
482.sphinx3	3704	failed	
GeoMean			**30.92**

Performance summary

- Fast Simulator is only 2.5 times slower than native on SpecInt2006 and 3.5 times on SpecFPU2006

  (3x in average and 6.2x in the worst case)

- Fast Simulator is 1.7 times faster than QEMU user mode on SpecInt2006 and 7.3 times faster on SpecFPU2006

  (4x in average and up to 13.5x on some tests)

- Fast simulator ~30 times faster than Spike

Why is Fast Simulator so fast?

Use of a “performance oriented” architecture:

- Compiler style Intermediate Representation

  (allows implementing more optimizations)
- Smart trace collection
- Optimized x86-64 backend
- Many of optimization tricks in other components are enabled by default
- Hardware floating point emulation

Good Multithreading Scalability

Intel(R) Xeon Phi(TM) CPU 7210 @ 1.30GHz , 64 cores

GeoBenchmark:
$ mig_benchmark 1001 1001 1

Number of threads	x86-64 seconds	Ratio	Fast Sim seconds	Ratio
1	32.262	**1**	154.477	**1**
2	16.136	**2.00**	77.322	**2.00**
4	8.084	**3.99**	38.811	**3.98**
8	4.07	**7.93**	19.525	**7.91**
16	2.049	**15.75**	9.799	**15.76**
32	1.051	**30.70**	5.082	**30.40**
64	0.536	**60.19**	2.614	**59.10**

Future plans

Still a proprietary simulator under development

Future plans may include:

- Additional optimizations to improve performance
- Improved floating point precision support
- Support for additional extensions
- Support for additional metrics
- Improved debugging options
- System Mode ?

Summary

- Internal working name is ExaGear-RVx
- ExaGear-RVx is an extremely fast simulator
  - Only 2.5 (int) to 3.5x(float) slower than native execution
  - Up to 10+ times faster(4x in average) than QEMU user mode
- Good multithreading scalability
- Industrial level reliability
- Reasonable design flexibility – Allows quick support for hardware ISA changes