VISA: A State-Based, Hierarchical, Architecture-Independent Random Test Generation Environment for High-Performance Multiprocessors

Presenter: Neil R. McKenzie, AMD

Neil.McKenzie@AMD.com

Task: design and implement a new random test generator (RTG)

- Sources of test stimuli
  - Random tests: ~95%
  - Directed tests: ~5%
  - Formal tools: < 1%
- Goals for new RTG
  - Customization
  - State-awareness
  - ISA independence
  - Leverage experienced team

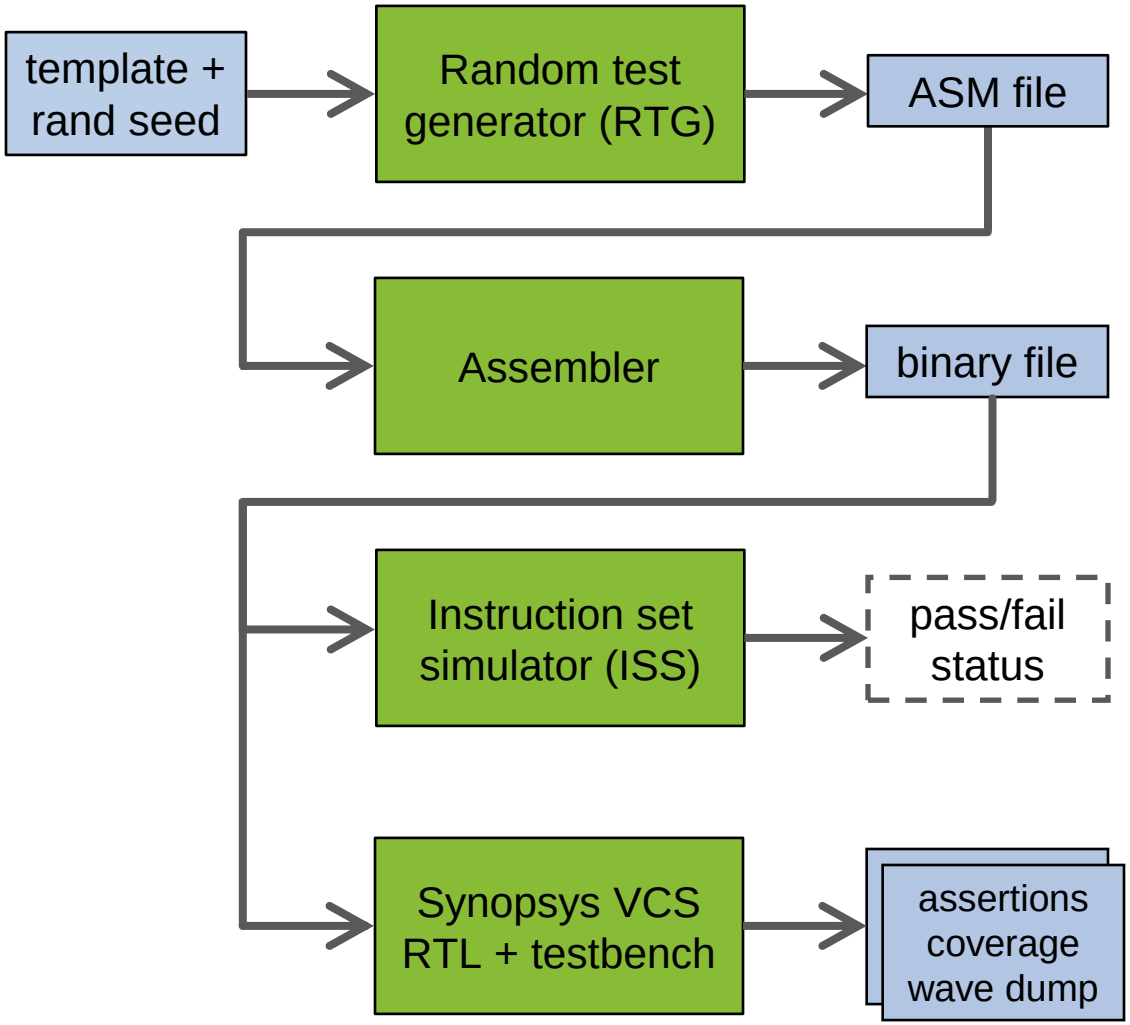
McKenzie

AMD

Agenda

- **State-aware random test generation**
- Ruby language
- VISA organization
- Hierarchical structure of templates
- Speed of test generation
- Summary, conclusions and future work

Standard RTG flow



1. Test generation

2. Assembly

3. Qualification

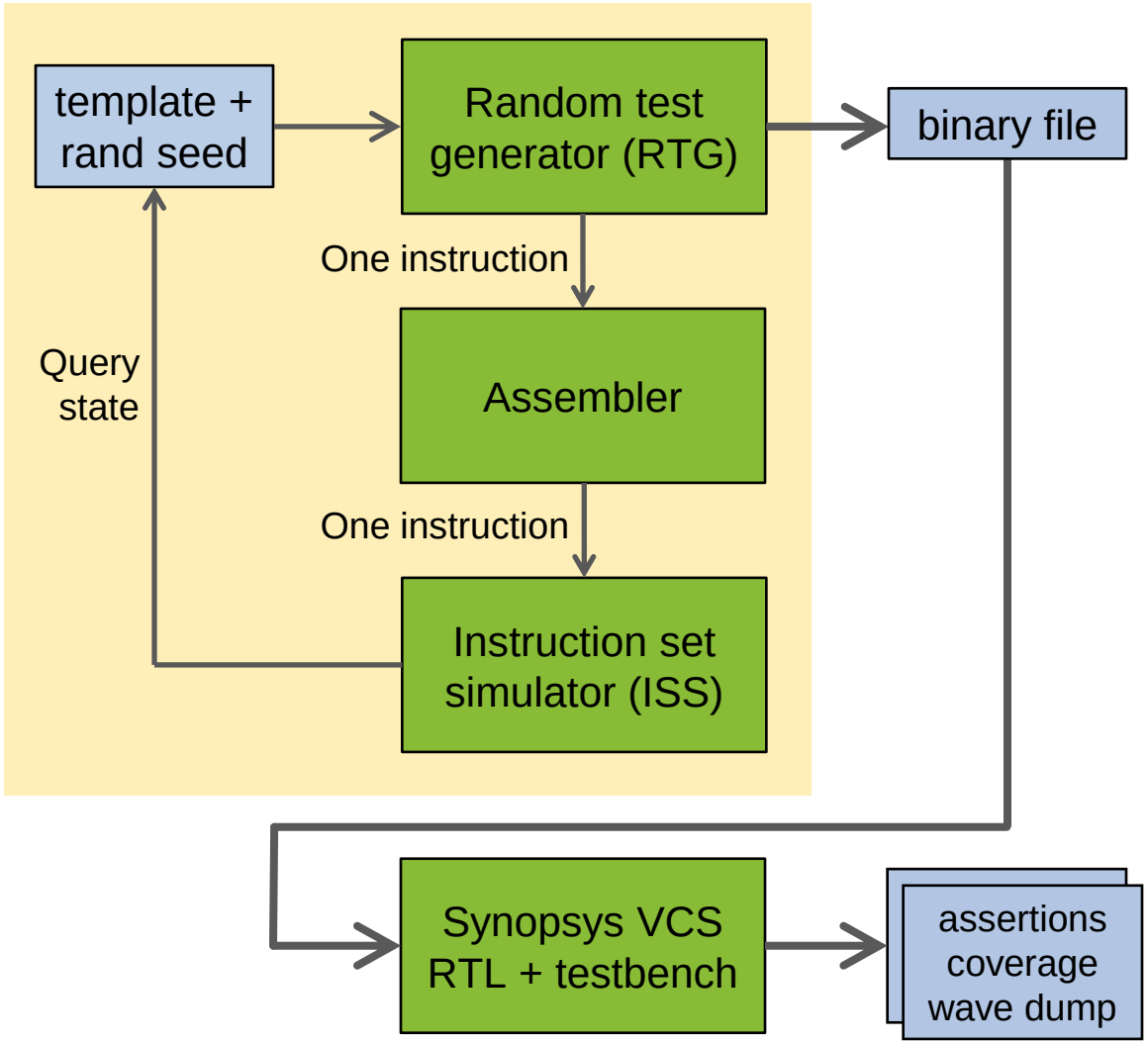
4. RTL simulation

McKenzie
AMD

State-aware test generation

- Embed the instruction set simulator (ISS) directly into the random test generator
  - User templates query the state of the ISS
  - Make code generation decisions on-the-fly
- Examples
  - Generate different code for different threads
  - Generate different code based on the carry flag

State-aware RTG flow



1. Test generation

2. RTL simulation

McKenzie
AMD

Impact of state-awareness

- Quality: verify coverage at test generation time
  - Exceptions
  - Carry and overflow bits, FP subnormals, etc.
  - Cache and TLB state transitions
  - Automatically generate self-checking code
- Efficiency: eliminate redundant set-up code
  - Converge coverage using fewer, shorter testcases
- Tradeoff: reduced speed of test generation
  - Still a net win because RTL simulation is slower than the speed of test generation

Agenda

- State-aware random test generation
- **Ruby language**
- VISA organization
- Hierarchical structure of templates
- Speed of test generation
- Summary, conclusions and future work

Ruby language: the best things in life are free

- Interpreted (like Perl, Python, Tcl, etc.)
  - Rapid prototyping [Hartson & Smith 89]
  - Immediate feedback for iterative refinement
- Object oriented (like Smalltalk)
  - Everything is an object, even integers
  - Classes, inheritance, virtual method invocation, polymorphism, data encapsulation
- Seamless dynamic linking with C++ .so files
  - C++ code runs 50-100x faster

Agenda

- State-aware random test generation
- Ruby language
- **VISA organization**
- Hierarchical structure of templates
- Speed of test generation
- Summary, conclusions and future work

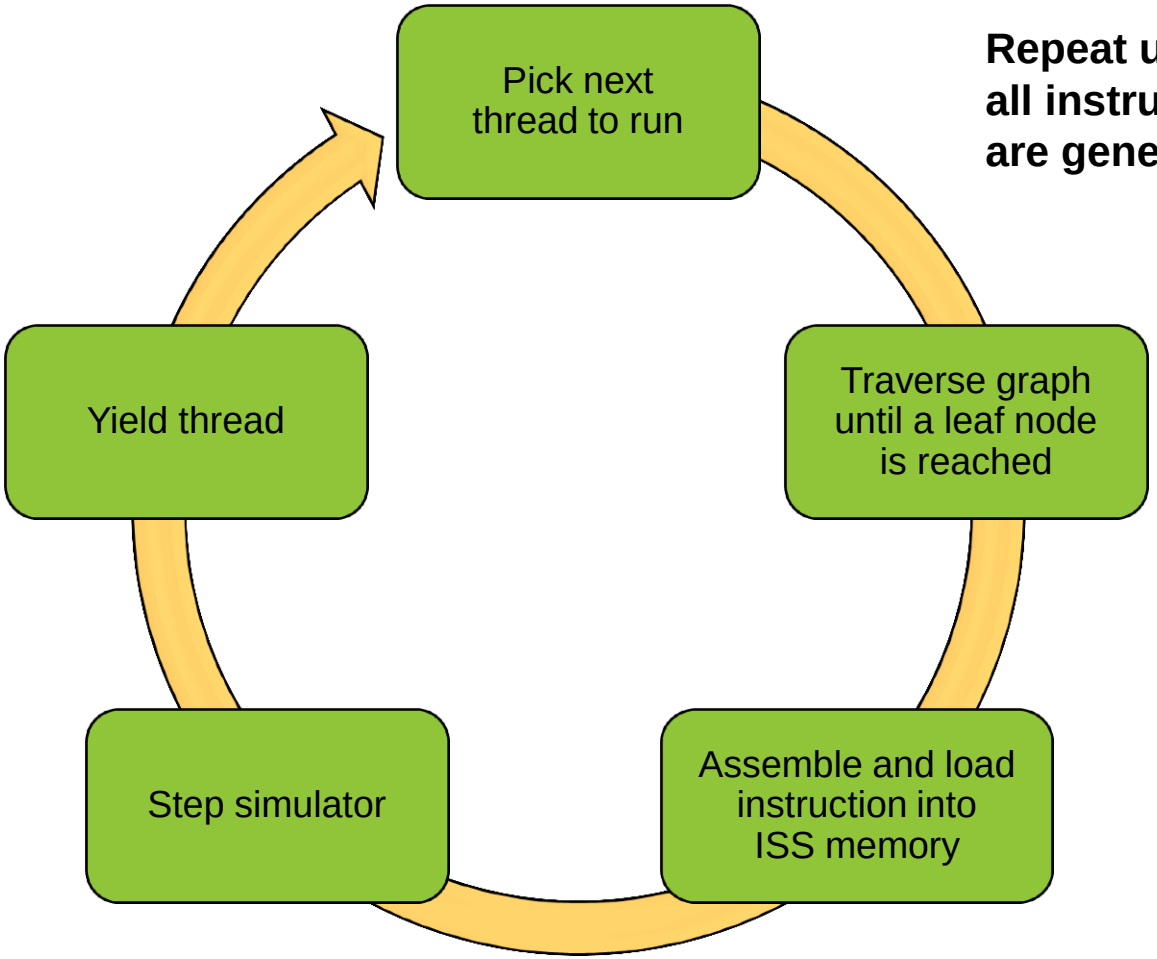
VISA components

- **Front-end (Ruby)**
  - Project file: arch defaults
  - Instruction table
  - Top-level template
  - Generator library
- **VISA core (Ruby)**
  - Test manager
  - Thread manager
  - Instruction manager
  - Assembler
- **Back-end (C++)**
  - Instruction set simulator
  - Memory manager
  - Paging manager
  - Test emitter

VISA flow

- Load C++ shared object files and Ruby libraries
- Create an instance of the top-level template class
  - Invoke its initialize method (like a C++ constructor)
  - Allocate page tables, memory regions, static data
- Create an array of Ruby fibers (one per test thread)
  - Each fiber begins executing the template's main method
  - Fibers are explicitly blocked and resumed
- Execute main loop
- Emit test (assembly or binary file)
- Emit TMI file: Test Meta Information

VISA main loop



Repeat until all instructions are generated

McKenzie

Agenda

- State-aware random test generation
- Ruby language
- VISA organization
- **Hierarchical structure of templates**
- Speed of test generation
- Summary, conclusions and future work

Hierarchical structure

- A template is a set of directed graphs (digraphs)
  - One digraph per thread
  - Digraphs can be reused or customized
  - Traverse each digraph in depth-first order
- Ruby classes implement generators
  - A generator object instance is a node of the digraph
- The main and generate methods specify edges to other nodes of the digraph
  - Syntax: **`gen`** *generator_object*

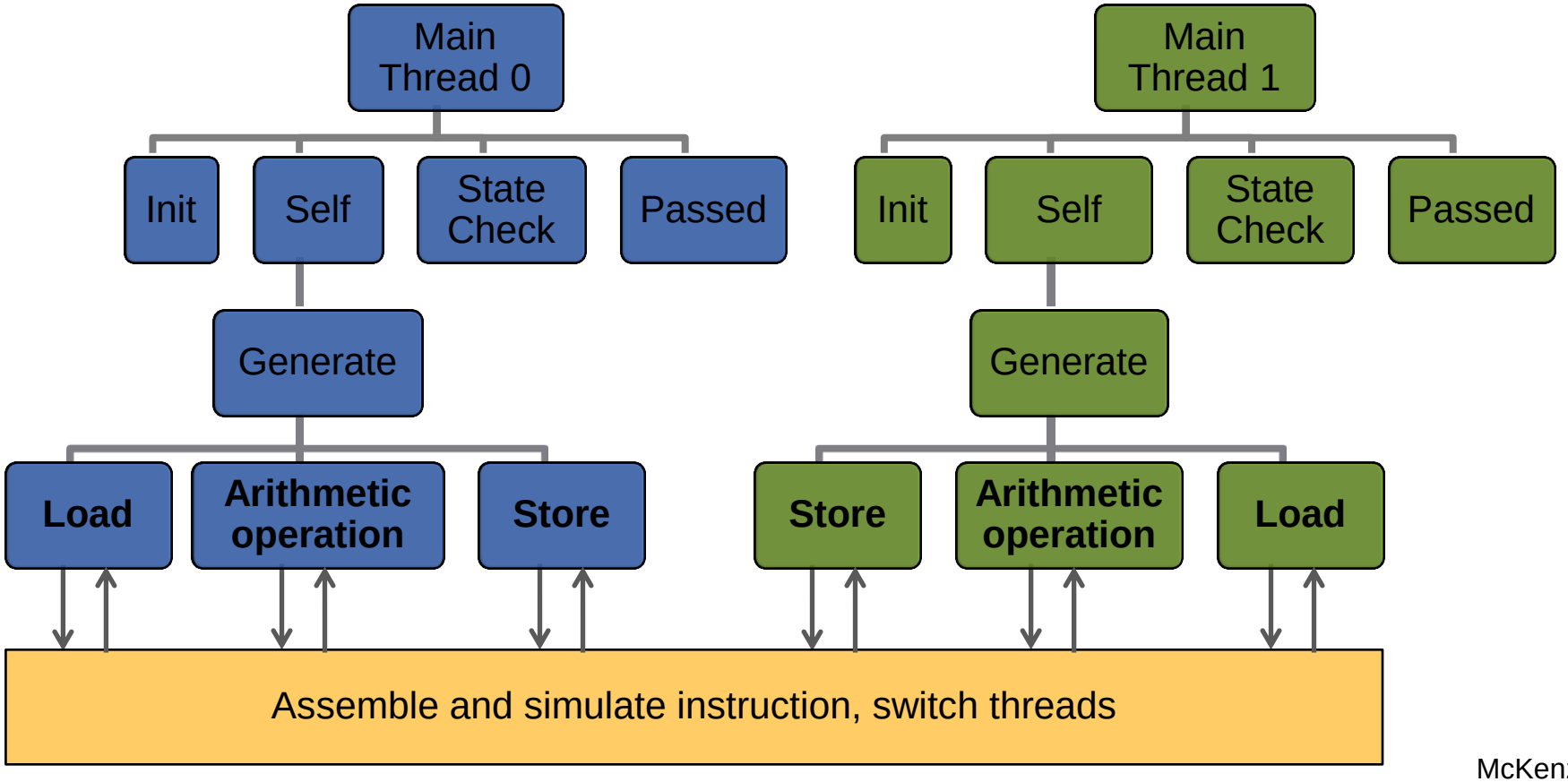
NullTest template


```
1   class NullTest < TestGenerator
2   # NullTest has a trivial initialize method
3     def initialize(name, threads)
4       super(name)
5     end

6   # NullTest has an empty generate method
7     def generate(state)
8     end

9   # The main method is common to all test templates
10    def main(state)
11      thread = state.get_id
12      gen InitGenerator.new(name: "init_thread#{thread}")
13      gen self
14      gen StateCheckGenerator.new("check_#{thread}")
15      gen PassedGenerator.new("passed_thread#{thread}")
16    end
17  end
```

Polymorphism: `gen` handles object by type

- Leaf node object (from instruction table)
  - Assemble and simulate one instruction
  - Yield thread and pick a different thread
- Array object
  - Traverse each array element in order
- Hash object (grab bag)
  - Select one item using a weighted random distribution
- Custom generator object
  - Invoke the object's generate method
  - Make decisions based on simulator's state

Structure of load-store template



McKenzie

Load-store template code


```
18  class LoadStoreTest < NullTest
19    def initialize(name, threads)
20      super(name)
21      @array = [
22        Inst_LOAD_Reg32_Addr32.new(…),
23        Inst_ARITH_Reg32_Reg32.new(…),
24        Inst_STORE_Reg32_Addr32.new(…)
25      ]
26    end

27    def generate(state)
28      thread = state.get_id
29      if thread == 0
30        gen @array
31      else
32        gen @array.reverse
33      end
34    end
35  end
```

Create array of instructions

Customize for each thread

McKenzie

TMI file output for thread 0


```
0x0009488bd3a7:
 :mnemonic:  "rex: mov rdi, 0x00000006768b7c7b"
 :bytes:     0x48, 0xbf, 0x7b, 0x7c, 0x8b, 0x76, 0x00, …
0x0009488bd3b1:
 :mnemonic:  "op: mov cx, [rdi]”
 :bytes:     0x66, 0x8b, 0x0f

0x0009488bd3b4:
 :mnemonic:  "sbb eax, esi"
 :bytes:     0x1b, 0xc6

0x0009488bd3b6:
 :mnemonic:  "rex: mov rcx, 0x000000001fdf9c71"
 :bytes:     0x48, 0xb9, 0x71, 0x9c, 0xdf, 0x1f, 0x00, …
0x0009488bd3c0:
 :mnemonic:  "mov [rcx], edi"
 :bytes:     0x89, 0x39
```

Load

Arithmetic operation

Store

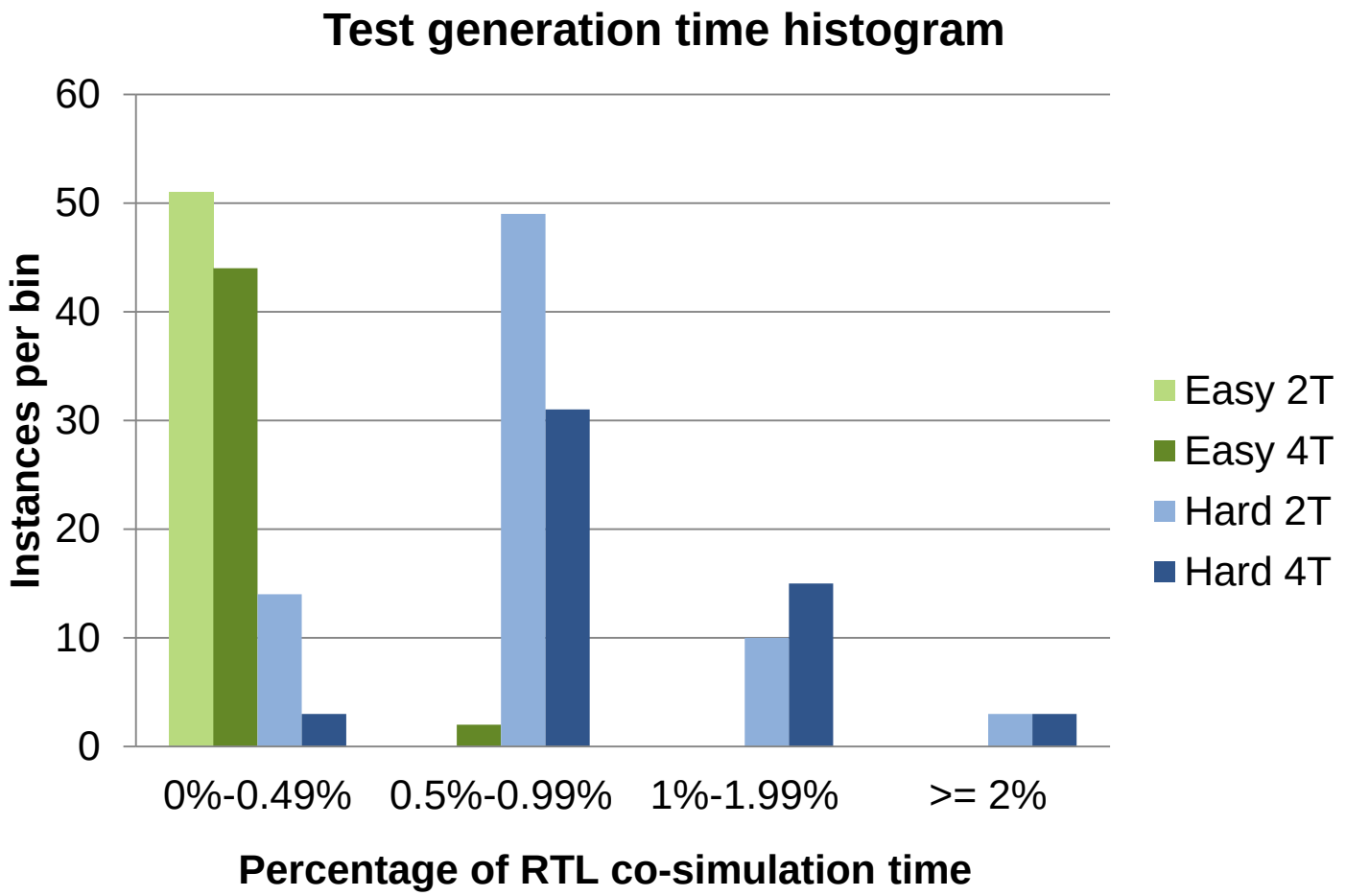
Generator library: fragments and scenarios

- Random instruction selection
- Instruction selection by regular expression
- Ad-hoc instruction generation (e.g. illegal instructions)
- Stride and region pre-fetching
- Thread synchronization
- Mutual exclusion
- Loops, branches (misprediction)
- Self-modifying code
- Multiprocessor cache coherence
- Exception handlers
- Physical and virtual address aliasing
- Many, many more

Agenda

- State-aware random test generation
- Ruby language
- VISA organization
- Hierarchical structure of templates
- **Speed of test generation**
- Summary, conclusions and future work

Speed of test generation



Agenda

- State-aware random test generation
- Ruby language
- VISA organization
- Hierarchical structure of templates
- Speed of test generation
- **Summary, conclusions and future work**

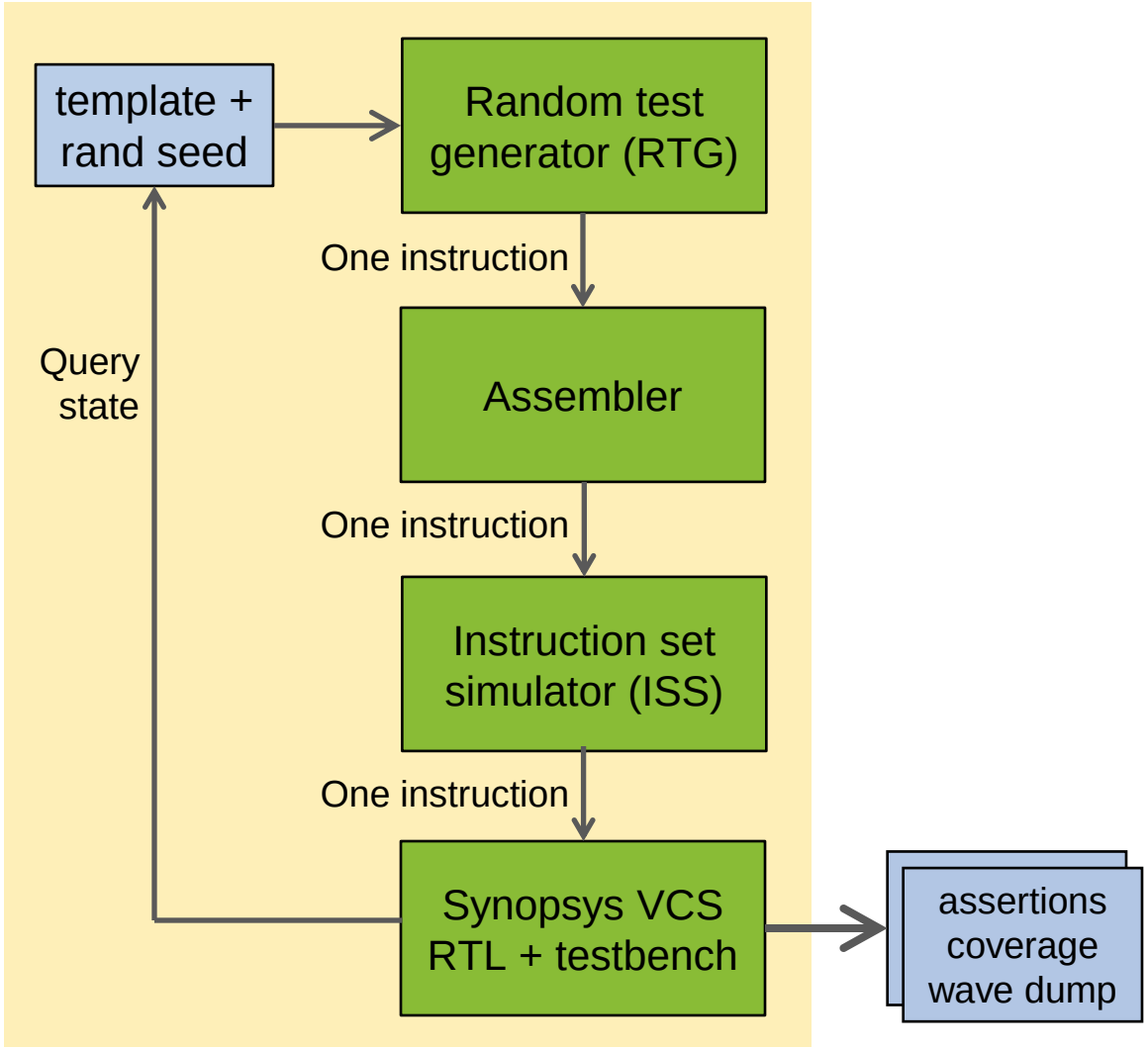
Summary and conclusions

- State-awareness improves quality of testcases
  - Verify feature coverage at test generation time
  - Reduce length and total number of testcases
  - Reduce overall cost of closing coverage
- Template structure: hierarchy + polymorphism
  - Reuse or customize code for each thread
  - Graph structure allows arbitrarily complex templates
  - We are continually adding generators to library

Summary and conclusions

- Ruby: 
  - Use one language for tool development and for templates
  - Interpreted: immediate feedback for iterative refinement
  - Seamless integration with C++ shared object modules
  - ISA-specific modules are loaded at run-time
  - One language expert is sufficient
- Speed of test generation: 
  - ~1% of RTL simulation time
  - Sensitive to the complexity of the template
  - Insensitive to the number of threads

Future RTG flow



McKenzie

Thank you

Synopsys Users Group
SILICON VALLEY 2013

McKenzie

AMD