

# EasyDiff – 一个高效实用的处理器验证框架

余子濠  
中科院计算所  
2019.11.13@深圳



中国科学院计算技术研究所  
Institute Of Computing Technology Chinese Academy Of Sciences



鹏城实验室

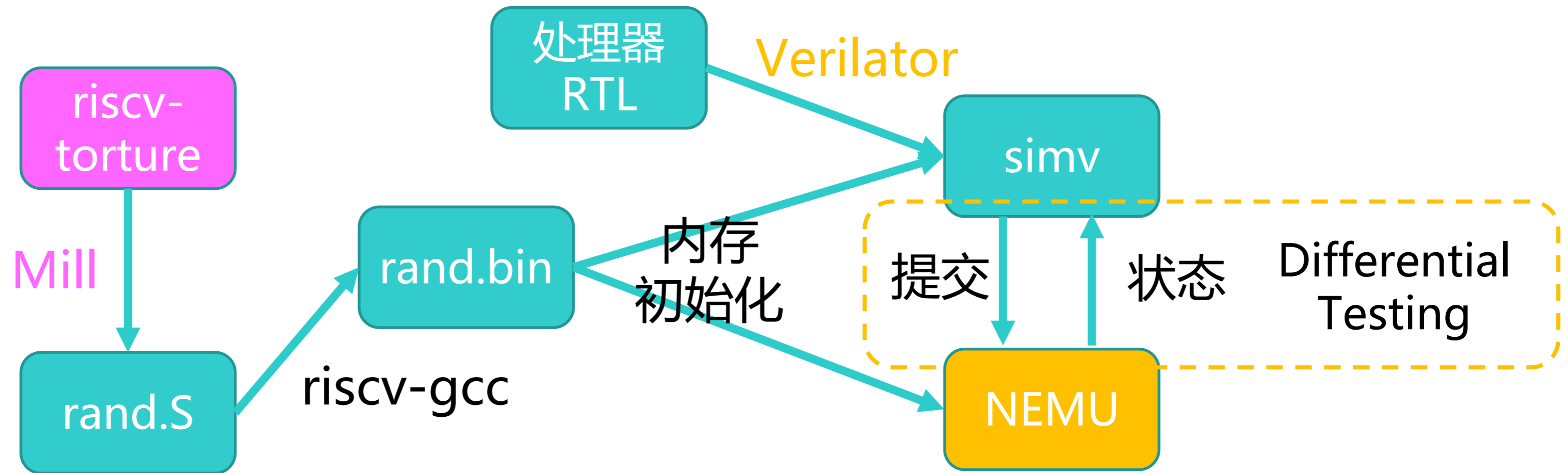
# 随机指令生成

- ▶ 随机指令生成是一种有效的处理器验证方式
  - 一般需要运行200亿条随机指令
- ▶ Google开发团队发布了基于UVM的riscv-dv[1]
  
- ▶ 但riscv-dv具有以下问题
  - **生成速度慢**
    - ▶ SystemVerilog -> C++ -> simv -> 随机指令
  - **定位错误难**
    - ▶ 测试出错时, 需要人工回溯定位出错指令
    - ▶ 指令越多, 回溯越难
  - **成本高**
    - ▶ 需要获得EDA工具(如Synopsys VCS)的授权才能使用
    - ▶ 非生产环境中难以使用

[1] <https://github.com/google/riscv-dv>

# EasyDiff – 一个高效实用的处理器验证框架

- 为了克服riscv-dv的问题, 我们提出了一个**高效实用低成本**的处理器验证框架EasyDiff
  - EasyDiff = riscv-torture + Mill + Verilator + NEMU



# 高效: riscv-torture + Mill

## ► riscv-torture[1]是UCB riscv团队开发的随机指令生成框架

- generator模块 - 随机指令生成
- testrun模块 - 运行测试并和spike[2]模拟器对比结果
- overnight模块 - 重复运行testrun

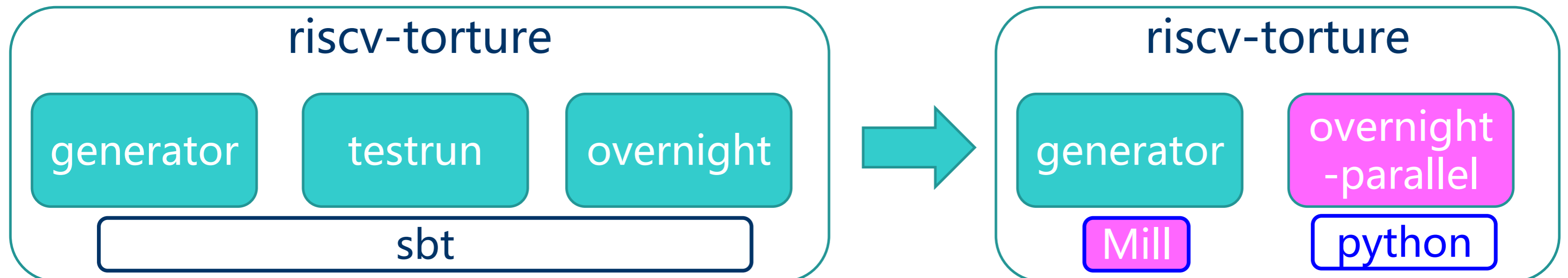
[1] <https://github.com/ucb-bar/riscv-torture>

[2] <https://github.com/riscv/riscv-isa-sim>

[3] <http://www.lihaoyi.com/mill>

## ► 我们的改进

- 使用具有缓存技术的scala构建工具Mill[3], 提高generator的效率
- 编写了并行化测试模块overnight-parallel, 提高在多核服务器上的测试效率



# 实用: Verilator + NEMU

- ▶ Verilator[1]是一个开源的RTL仿真器, 是rocketchip项目的默认仿真器
- ▶ NEMU(NJU Emulator)[2]是南京大学的教学模拟器
  - 支持riscv指令的解释执行
- ▶ 与riscv-torture的testrun模块中的默认仿真器spike相比, NEMU
  - 更简单, 采用面向二年级本科生容易理解的设计
  - 容易与RTL开发进度保持一致, 可在屏蔽未实现功能的情况下进行对比
  - 支持一组指令级别的差分测试(Differential Testing)API, 容易获得NEMU的状态并进行控制

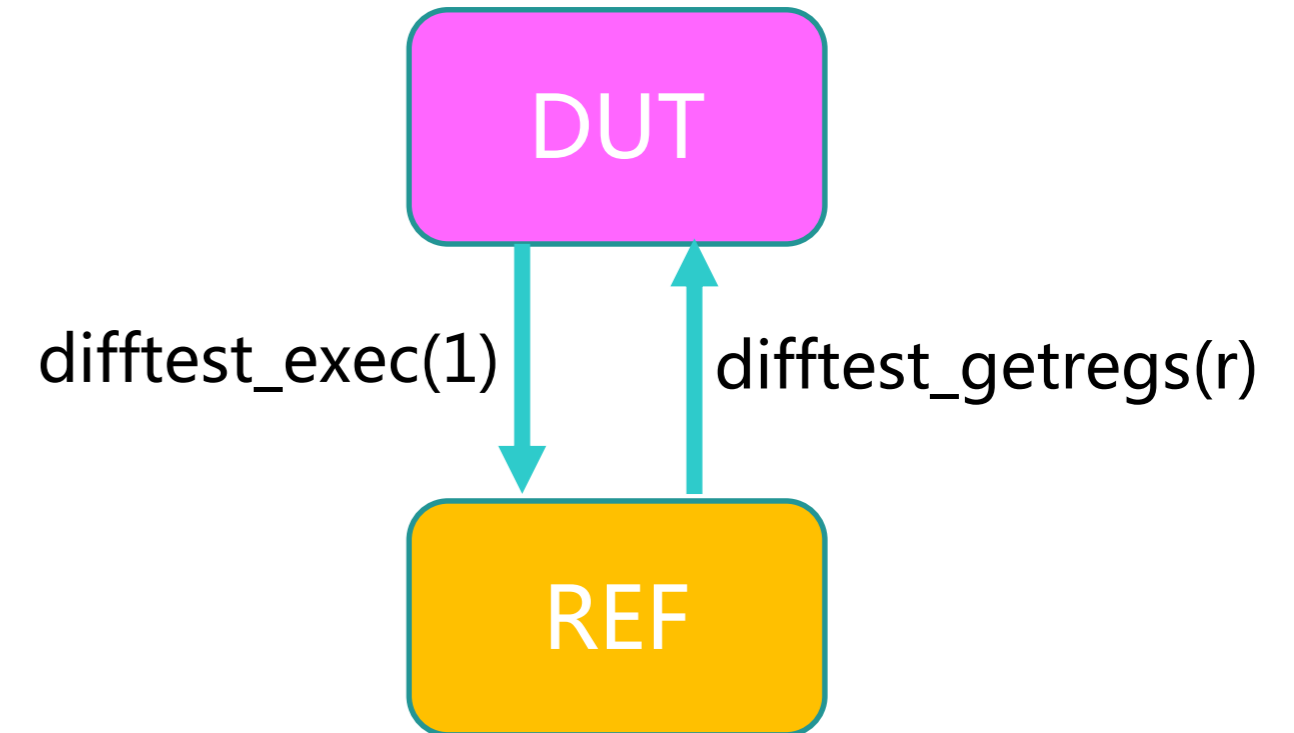
[1] <https://www.veripool.org/wiki/verilator>

[2] <https://github.com/NJU-ProjectN/nemu>

# 实用的秘诀 - 差分测试API

- ▶ 差分测试思想: 对于根据同一规范的两​​种实现, 给定相同的有定义的输入, 它们的行为应当一致

API	说明
<code>difftest_memcpy(dest, src, size)</code>	从src复制size字节到参考实现的内存地址dest中
<code>difftest_getregs(r)</code>	获得参考实现的寄存器状态
<code>difftest_setregs(r)</code>	设置参考实现的寄存器状态
<code>difftest_exec(n)</code>	令参考实现执行n条指令



- ▶ DUT = 测试对象, REF = 参考实现
- ▶ 在NEMU教学实验中, DUT = 学生写的NEMU, REF = QEMU
- ▶ 在EasyDiff中, DUT = RTL代码, REF = NEMU

# 实现差分测试API

- ▶ NEMU的教学框架代码已经提供[1]
- ▶ 只需在仿真驱动代码(simv.cpp)中调用它们

```
while (1) {  
    step();  
    difftest_exec(1);  
    rtl_getregs(&r1);  
    difftest_getregs(&r2);  
    if (r1 != r2) { abort(); }  
}
```

- ▶ 通过运行时动态链接将NEMU链接到simv的地址空间
  - dlopen, dlsym
  - 得益于NEMU的简单, 不依赖libc之外的其他库
- ▶ **在首次检测到寄存器状态不同时, 马上报错!**

```
7 void difftest_memcpy_from_dut(paddr_t dest, void *src, size_t n) {  
8     memcpy(guest_to_host(dest), src, n);  
9 }  
10  
11 void difftest_getregs(void *r) {  
12     memcpy(r, &cpu, DIFFTEST_REG_SIZE);  
13 }  
14  
15 void difftest_setregs(const void *r) {  
16     memcpy(&cpu, r, DIFFTEST_REG_SIZE);  
17 }  
18  
19 void difftest_exec(uint64_t n) {  
20     cpu_exec(n);  
21 }
```

[1] <https://github.com/NJU-ProjectN/nemu/blob/ics2019/src/monitor/diff-test/ref.c>

# 评估案例1 – 指令生成效率

- ▶ 环境: Intel Core i7-9700K服务器
- ▶ 生成10000条随机静态指令, 考察
  - 首次生成(需要编译并运行)用时
  - 再次生成(无需编译, 直接运行)用时, 更常用

框架	首次生成用时(s)	再次生成用时(s)
基于UVM的riscv-dv	49	33
基于sbt的riscv-torture	11	11
EasyDiff	11	1

## ▶ riscv-dv

- 修改配置 = 修改源代码
- SystemVerilog -> C++ -> simv -> 随机指令
- 49 = 16 + 33

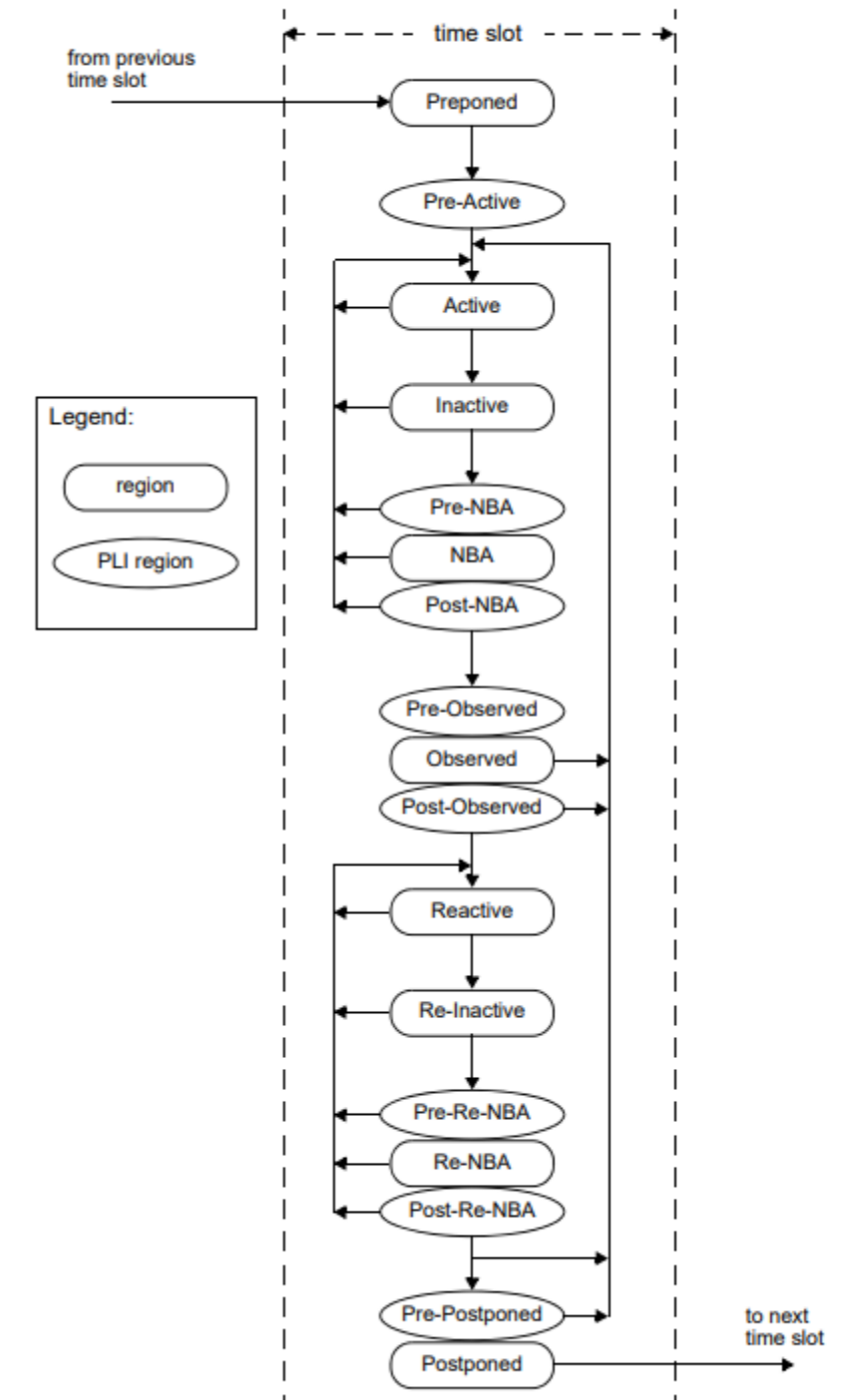
## ▶ riscv-torture

- sbt未采用缓存技术, 再次生成时仍然会触发加载和分析依赖项的工作, 并进行重新编译



# 为什么能快33倍?

- ▶ SystemVerilog的**本质**是一门事件建模语言
- ▶ 用SystemVerilog编写的所有代码, **最终都会被编译成事件和队列的模型, 并以事件驱动的方式来运行**
- ▶ 所以VCS和Verilator生成的c++文件很复杂
- ▶ simv编译得慢, 运行效率也低
  - i++;
    - ▶ C语言 -> add \$1, %eax (一条指令)
    - ▶ SystemVerilog -> 很复杂(右图)
- ▶ 随机指令生成这件事, 通用编程语言就能做得很好
  - 没有必要用事件队列模型来做
  - 即使是在JVM中运行的Scala, 也比SystemVerilog好得多
  - 如果用C语言来写, 效率将会更高



# 评估案例2 – 在线对比效果

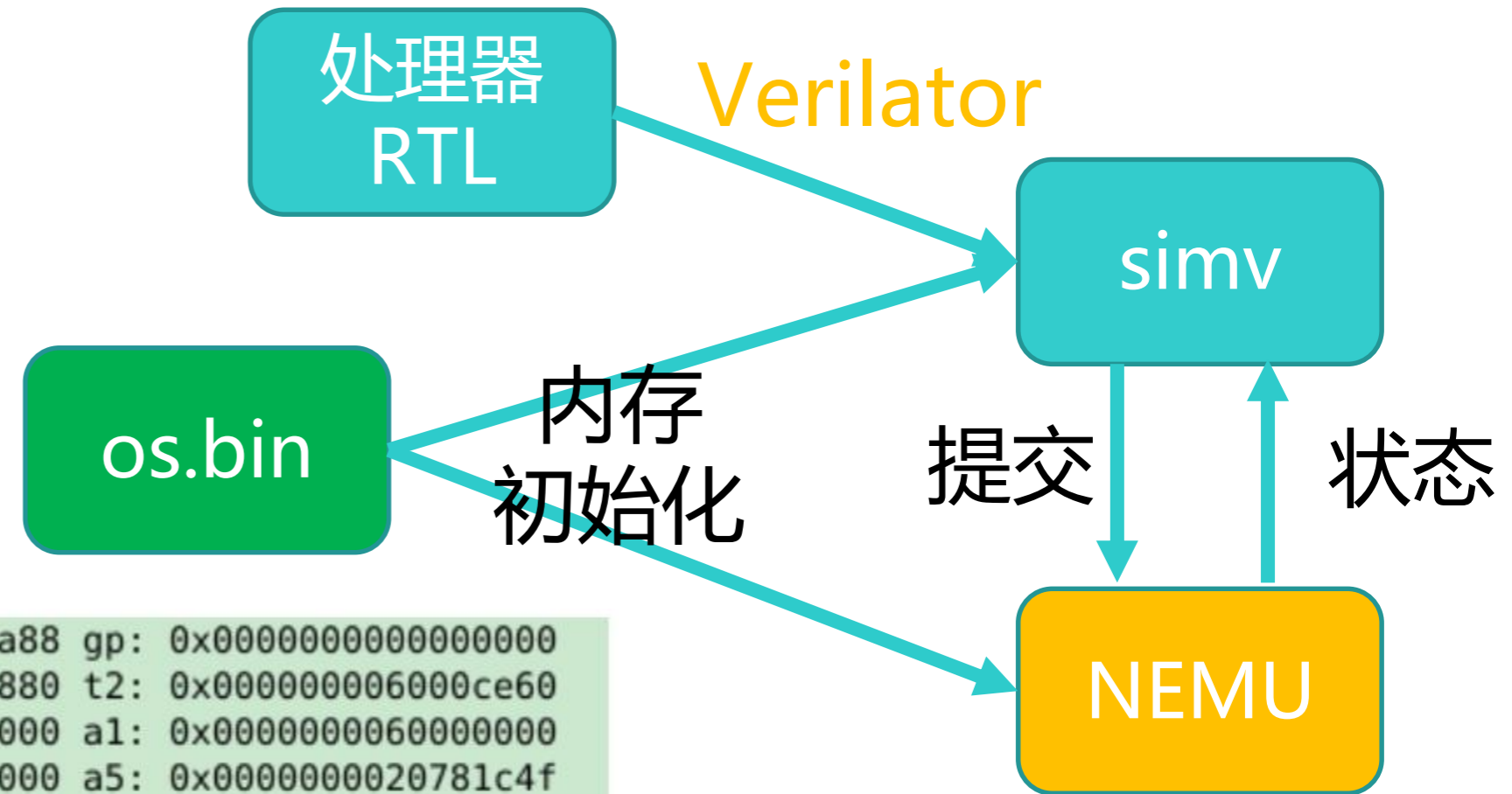
- ▶ 一个指令错误的例子: 未实现除0

```
$0: 0x0000000000000000 ra: 0xffffffffffff9cb sp: 0x0000000000000040 gp: 0x000000000000003f
tp: 0x0000000000000000 t0: 0x0000000000000000 t1: 0x8000000000000000 t2: 0xffffffffffff82e
s0: 0xffffffffffffd9f s1: 0xffffffffffff1d a0: 0x00000000800022a6 a1: 0x8000000000000000
a2: 0x00000000000000493 a3: 0x7fffffffffffffff a4: 0xffffffffffffe52 a5: 0x0000000000000000
a6: 0x8000000000000000 a7: 0xffffffffffffd7f s2: 0x00000000800023c2 s3: 0xffffffffffff82e
s4: 0xffffffffffff s5: 0xffffffffffffe3c s6: 0xffffffffffff65 s7: 0x0000000000000000
s8: 0x7fffffffffffffff s9: 0x80000000800022a6 s10: 0x00000000800022a6 s11: 0x0000000000000000
t3: 0x8000000000000262 t4: 0x7ffffffffffff82e t5: 0x0000000007ffffff t6: 0x00000000000000f7
pc: 0x00000000800000a8
x20 different at pc = 0x800000a4, right= 0xffffffffffff, wrong = 0x0000000000000001
ABORT at pc = 0x800000a8
total guest instructions = 600
instrCnt = 600, cycleCnt = 1468, IPC = 0.408719
Guest cycle spent: 1469
Host time spent: 3ms
```

```
44 80100090: 0f8fbf83 ld t6,248(t6)
45 80100094: 0040006f j 80100098 <_end+0xfda88>
46 80100098: 49300613 li a2,1171
47 8010009c: 9cb00093 li ra,-1589
48 801000a0: d7f00893 li a7,-641
49 801000a4: 02b7da3b divuw s4,a5,a1
50 801000a8: 00002297 auipc t0,0x2
51 801000ac: 5a728293 addi t0,t0,1447 # 801026
52 801000b0: 00100f13 li t5,1
53 801000b4: c0a00213 li tp,-1014
54 801000b8: 00002697 auipc a3,0x2
```

# 评估案例3 – 在线对比效果

- ▶ 一个非指令错误的例子
  - EasyDiff还能跑操作系统



```
$0: 0x0000000000000000 ra: 0x00000000800023a4 sp: 0x0000000081d9da88 gp: 0x0000000000000000
tp: 0x0000000000000000 t0: 0x000000000000000b t1: 0x00000000000c1880 t2: 0x000000006000ce60
s0: 0x0000000081e16000 s1: 0x0000000081e01800 a0: 0x0000000081e16000 a1: 0x0000000060000000
a2: 0x0000000081ea5000 a3: 0x0000000000000000 a4: 0xffffffffffff0000 a5: 0x0000000020781c4f
a6: 0x0000000000000040 a7: 0x0000000000000200 s2: 0x0000000081ea5000 s3: 0x0000000081d96008
s4: 0x0000000000000100 s5: 0x0000000000033000 s6: 0x0000000000000011 s7: 0x00000000000000ff
s8: 0x0000000080004f78 s9: 0x00000000800064e8 s10: 0x0000000000000100 s11: 0x0000000081d9daf8
t3: 0x0000000000000001 t4: 0x0000000000000190 t5: 0x0000000000000190 t6: 0x0000000000000190
pc: 0x0000000080003afc
x15 different at pc = 0x80003af8, right= 0x0000000020781c4f, wrong = 0x0000000020781c0f
ABORT at pc = 0x80003afc
total guest instructions = 27207413
instrCnt = 27207413, cycleCnt = 136717470, IPC = 0.199005
Guest cycle spent: 136717471
Host time spent: 105911ms
```


- ▶ 根据pc值查阅相应代码, 发现在读取页表项 -> access bit实现错误
- ▶ **没有EasyDiff, 这一错误将会继续传播, 造成更多的错误, 调试将非常困难**


# 评估案例3 – 在线对比效果

- ▶ 南京大学本科生在2018年龙芯杯处理器设计大赛中借助差分测试实现乱序处理器

## 取反之前忘记零扩展

```
306 307 def tlb_write(i:UInt):Unit = {
307 308   val entry = tlb(i)
309 +   val mask = ~(pagemask.mask.ZExt(32))
308 310   if(param.pipelineDebug) {
309 311     ... (param.pipelineDebug) {
```

 Fixed bug of insufficient bits of pagemask  
141242068-ouxianfei authored 1 year ago

 differ with nemu when running nanos at 922149 cycle  
141242068-ouxianfei authored 1 year ago

## 基础设施：效果展示

### 突破限制，改进工具

	Verilog	Chisel	NEMU
实现类似功能	500行	150行 (代码密度x3)	N/A
性能仿真时间	20min (大赛官方平台)	30s (速度提升40倍)	0.5s (提升2400倍)

### 七天实现乱序处理器

差分测试：三天内修复了6个乱序执行在特定复杂条件下触发的non-trivial bug  
在2分钟内定位了某TLB bug

- When cache instr trigger
- Set bht 1 history length
- Added --no-gen to avoid
- Added page fault excepti
- Fixed bug of insufficien

### 决赛现场最快速度完成指令添加

5分钟完成代码开发和验证 (第1支完成的队伍)  
综合完毕后一次通过验收

<http://www.nscscc.org/a/wangjie/NSCSCC2018/2018/1010/46.html>

# 评估案例4 – 大规模并行测试

- ▶ 环境: 配备8个Intel Xeon E7-8855 v4(共计112个核心)的服务器
- ▶ 运行EasyDiff的overnight-parallel脚本
- ▶ 统计脚本: **3分钟**通过了**2亿条**随机指令的测试
  
- ▶ 若使用riscv-dv
  - 即使生成2亿条指令, 也需要**5小时**(按服务器性能直接折算)
    - ▶  $200000000/10000/112*33*3/3600=4.91$
  - 由于网络隔离原因, 该服务器暂未配置VCS license, 无法直接运行riscv-dv

# NEMU的一些可能替代方案

## ▶ QEMU和spike

## ▶ 不改代码

– QEMU: 只能通过基于socket通信的GDB协议实现差分测试API

▶ socket通信开销极大, 比动态链接NEMU的方案慢30倍

– spike: 上位机GDB → GDB协议 → OpenOCD → rbb协议 → spike.debug module

▶ 只会比QEMU更慢

## ▶ 想改代码

– QEMU代码太复杂, 即时编译优化不敢随便改, 内存的MemoryRegion结构难理解

– spike代码没那么复杂, 也许花点时间能实现

▶ 但NEMU更简单, 花同样的时间说不定已经写出来了, 而且还能和RTL进展匹配

▶ Keep It Simple, Stupid

# 局限性

## ▶ 中断

- 会改变指令的执行流
- 可以在差分测试API中添加difftest\_raise\_intr(NO)来解决问题

## ▶ 多核

- 挑战: 满足WMO(弱一致性)的不同行为都是正确的, 比较寄存器状态的判断方式不再成立
- 属于内存一致性的研究范畴, EasyDiff目前无法解决
  - ▶ 可以通过内存一致性的工具来进行独立的测试

# 小结

- ▶ 我们提出了一个**高效实用低成本**的处理器验证框架EasyDiff
  - EasyDiff = riscv-torture + Mill + Verilator + NEMU

	riscv-dv	EasyDiff
指令生成速度	<b>慢</b>	快 <b>33</b> 倍
定位错误难度	<b>难</b> , 需要人工定位出错点	<b>瞬间</b> 定位出错点
成本	<b>高</b> , 需要支付EDA证书费用	组件开源 <b>免费</b> , 教学和个人均可用

- ▶ 计算机系统能力可以帮助我们去思考问题的本质, 找到更好的解决方案
- ▶ 开源
  - <https://github.com/LvNA-system/riscv-torture>
  - <https://github.com/LvNA-system/EasyDiff-noop>
  - <https://github.com/LvNA-system/EasyDiff-nemu>