SVM：用可综合方法实现 RISC-V 处理器的高效验证

徐易难

中国科学院计算技术研究所

2025年7月

处理器验证是芯片开发的瓶颈环节之一

206%

2007年至今
验证工程师增长

5:1

验证：设计
工程师数量比

86%

首次流片未成功的
芯片项目占比

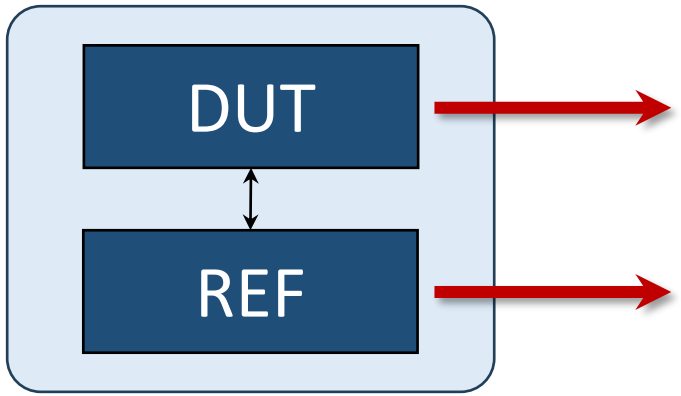
75%

超过预期时间的
芯片项目占比

尽管投入大量资源，芯片验证质量和效率仍未达预期

来源：https://resources.sw.siemens.com/en-US/white-paper-2024-wilson-research-group-ic-asic-functional-verification-trend-report/

背景：协同仿真验证



待验证处理器
RTL (Verilog, ...)

指令集模拟器
Software (C++, ...)

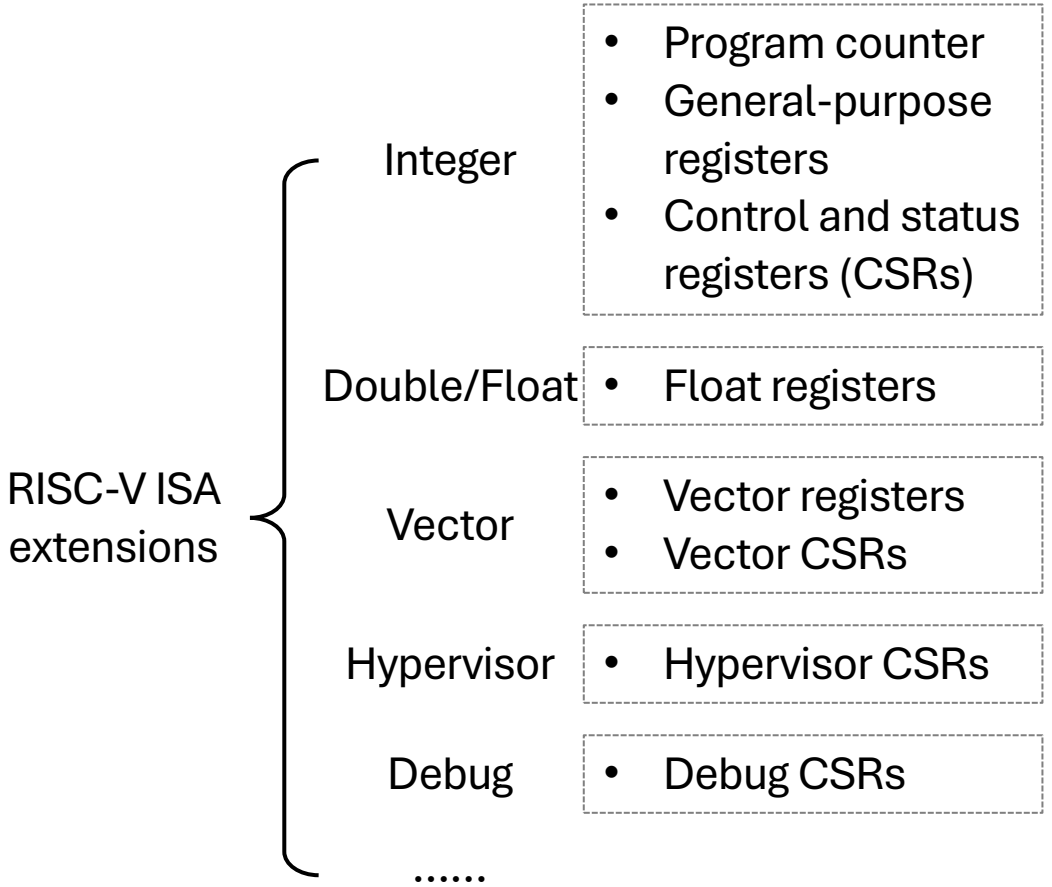
协同仿真
Co-Simulation

发展趋势1：指令集复杂度提升

- **RISC-V 指令集复杂度正在迅速膨胀**

- **以 RVA23 Profile 为例**
  - 33 个必选扩展
  - 830 页指令集手册
  - 相比 2019 年手册篇幅翻倍

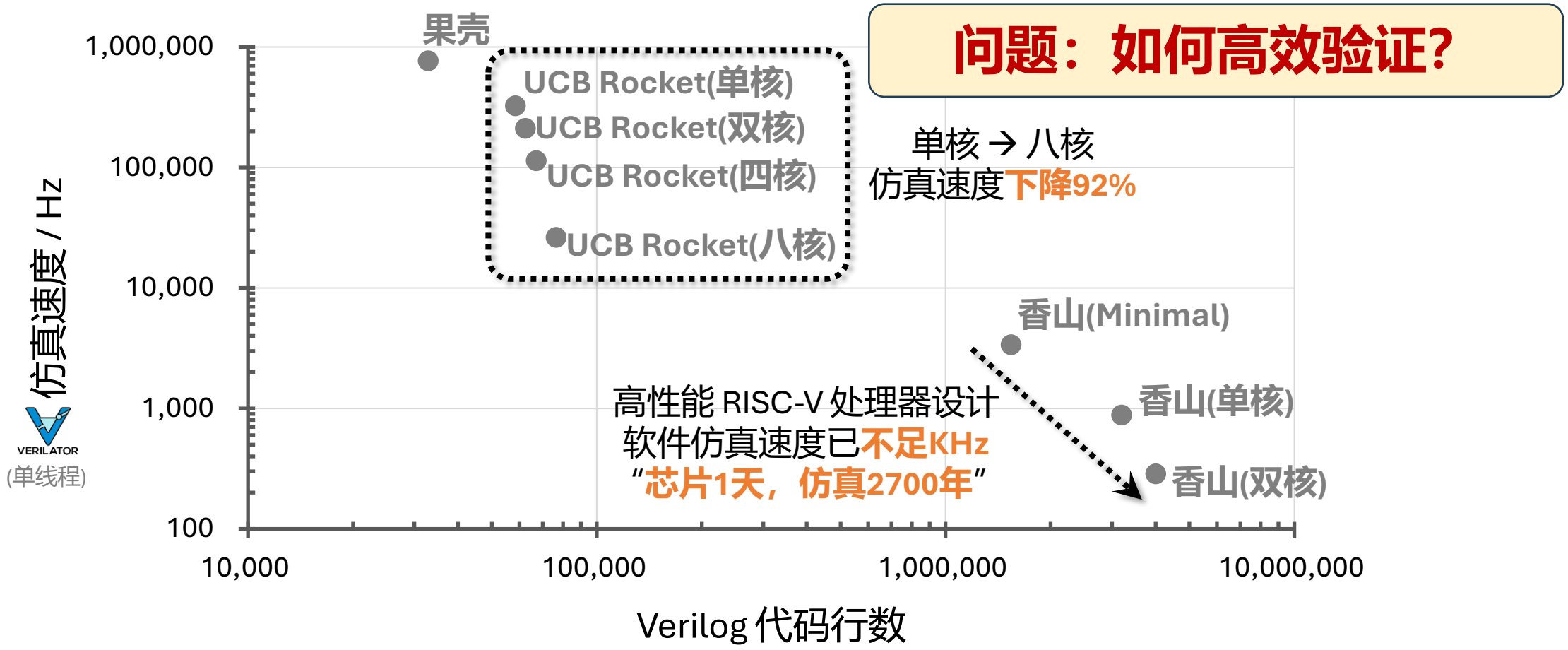
- 功能点、状态量、可选行为、……



问题：如何验证?

发展趋势2：电路仿真速度慢

- 最常使用**软件仿真**，但当**处理器规模扩大**，其**仿真速度大幅下降**



基于硬件仿真平台的处理器验证加速

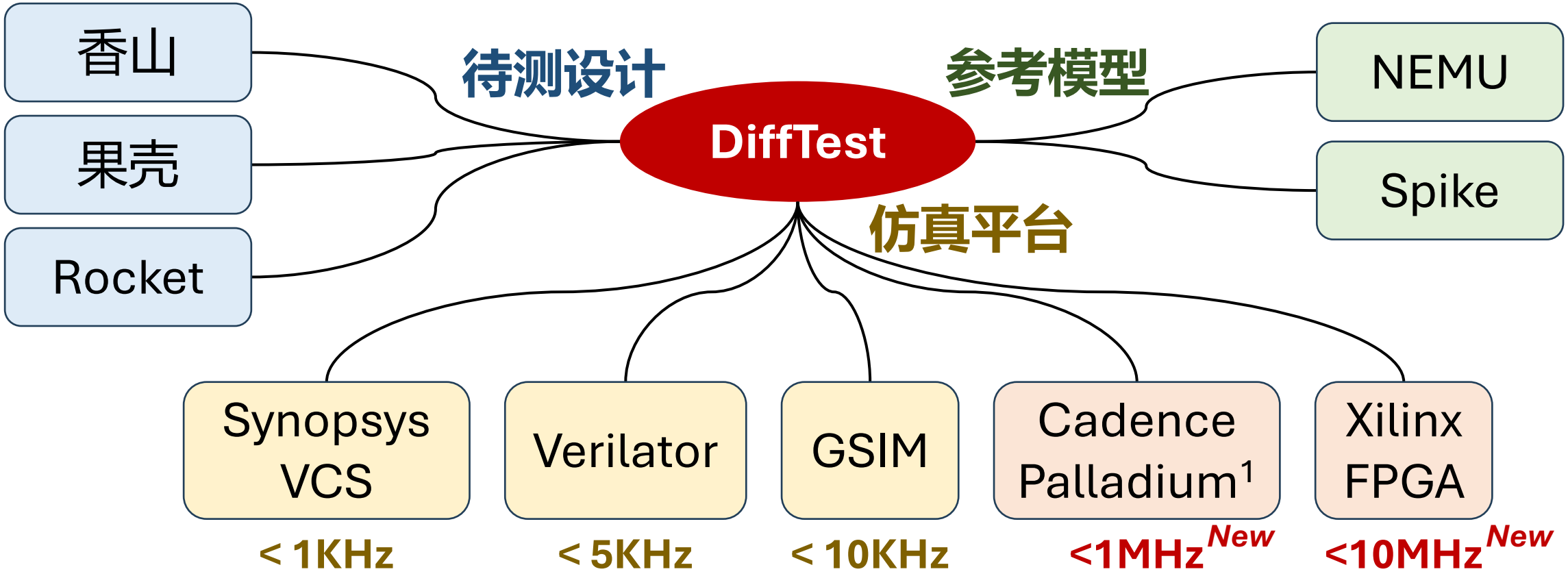
- **动机**：**硬件仿真平台**数量级地加速电路仿真，在此基础上优化DUT-REF整体速度
- **特点**：将 REF 部署于 Host 环境，以**软硬件(RTL-Host)通信**为核心
  - **硬件**(RTL侧)：利用仿真加速器、FPGA等加速 **DUT 电路仿真**
  - **软件**(Host侧)：仍使用x86、ARM宿主机运行 **REF 软件逻辑**
  - **通信**：利用PCIe、以太网、InfiniBand(IB)等**连接手段**在软硬件间传递数据

- Emulator、FPGA等
- 速度 1~100 MHz

- x86/ARM CPU
- 速度 >> 100 MHz

DiffTest：基于Emulator/FPGA的处理器验证加速

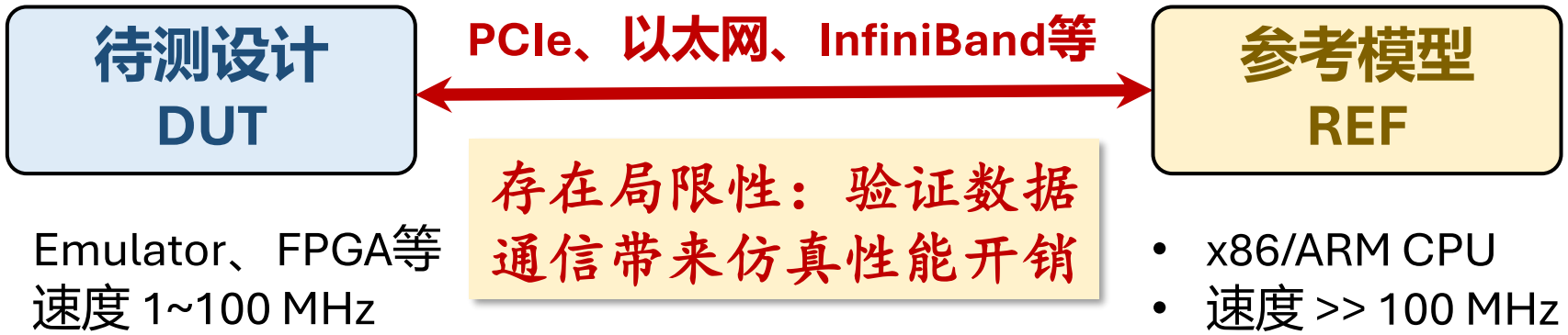
- **香山团队长期维护 DiffTest 开源验证框架，现已支持硬件仿真加速**
  - https://docs.xiangshan.cc/zh-cn/latest/tools/difftest/



[1]面向Palladium的高性能处理器软硬件协同验证框架部署与加速方法. CadenceLIVE中国用户大会 2024.

基于硬件仿真平台的处理器验证加速

- **动机**：**硬件仿真平台**数量级地加速电路仿真，在此基础上优化DUT-REF整体速度
- **特点**：将 REF 部署于 Host 环境，以**软硬件(RTL-Host)通信**为核心
  - **硬件**(RTL侧)：利用仿真加速器、FPGA等加速 **DUT 电路仿真**
  - **软件**(Host侧)：仍使用x86、ARM宿主机运行 **REF 软件逻辑**
  - **通信**：利用PCIe、以太网、InfiniBand(IB)等**连接手段**在软硬件间传递数据



RTL-Host 架构下的通信开销问题

FireSim FPGA 平台 1024 节点仿真 @ 3.4MHz

hardware-software co-design. As an example, we demonstrate automatically generating and deploying a target cluster of 1,024 3.2 GHz quad-core server nodes, each with 16 GB of DRAM, interconnected by a 200 Gbit/s network with 2 microsecond latency, which simulates at a 3.4 MHz processor clock rate (less than 1,000x slowdown over real-time). In

```
17  extern "C" void dromajo_step(int      hart_id,
18                               uint64_t pc,
19                               uint32_t insn,
20                               uint64_t wdata,
21                               uint64_t mstatus) {
```

Dromajo每次指令检查仅使用 **256 bit** 数据

理想与现实的巨大差距

6.3.2 Fromajo Co-simulation. *Sonic*BOOM is also integrated with the Dromajo [1] co-simulation tool. Dromajo checks that the committed instruction trace matches the trace generated by a software architectural simulator. Fromajo integrates Dromajo with a FireSim FPGA-accelerated *Sonic*BOOM simulation, enabling co-simulation at over 1 MHz, orders of magnitude faster than a software-only co-simulation system. Fromajo revealed several latent bugs related to interrupt handling and CSR management.

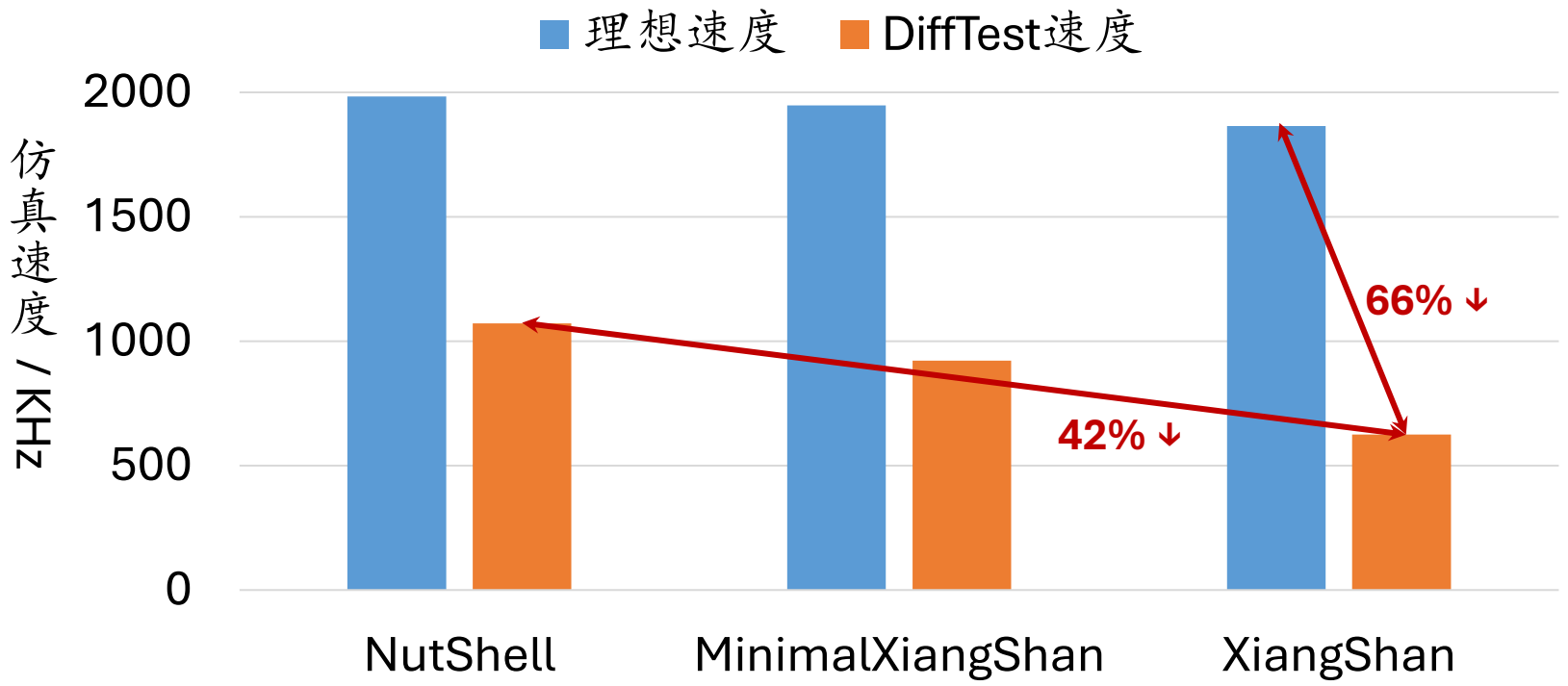
SonicBOOM 在 FireSim 上仅实现 **~1MHz 的验证速度**

```
54  interface rvviTrace
55  #(
56      parameter int ILEN   = 32,  // Instruction length in bits
57      parameter int XLEN   = 32,  // GPR length in bits
58      parameter int FLEN   = 32,  // FPR length in bits
59      parameter int VLEN   = 256, // Vector register size in bits
60      parameter int NHART  = 1,   // Number of harts reported
61      parameter int RETIRE = 1    // Number of instructions that can retire during valid event
62  );
63      //
64      // RISCV output signals
65      //
66      wire                    clk;                                    // Interface clock
67      wire                    valid      [(NHART-1):0][(RETIRE-1):0]; // Valid event
68      wire [63:0]             order      [(NHART-1):0][(RETIRE-1):0]; // Unique event order count (no gaps or reuse)
69
70      wire [(ILEN-1):0]       insn       [(NHART-1):0][(RETIRE-1):0]; // Instruction bit pattern
71      wire                    trap       [(NHART-1):0][(RETIRE-1):0]; // State update without instruction retirement
72      wire                    debug_mode [(NHART-1):0][(RETIRE-1):0]; // Retired instruction executed in debug mode
73
74      // Program counter
75      wire [(XLEN-1):0]       pc_rdata   [(NHART-1):0][(RETIRE-1):0]; // PC of instruction
76
77      // X Registers
78      wire [31:0][(XLEN-1):0] x_wdata    [(NHART-1):0][(RETIRE-1):0]; // X data value
79      wire [31:0]             x_wb       [(NHART-1):0][(RETIRE-1):0]; // X data writeback (change) flag
80
81      // F Registers
82      wire [31:0][(FLEN-1):0] f_wdata    [(NHART-1):0][(RETIRE-1):0]; // F data value
83      wire [31:0]             f_wb       [(NHART-1):0][(RETIRE-1):0]; // F data writeback (change) flag
84
85      // V Registers
86      wire [31:0][(VLEN-1):0] v_wdata    [(NHART-1):0][(RETIRE-1):0]; // V data value
87      wire [31:0]             v_wb       [(NHART-1):0][(RETIRE-1):0]; // V data writeback (change) flag
88
89      // Control and Status Registers
90      wire [4095:0][(XLEN-1):0] csr      [(NHART-1):0][(RETIRE-1):0]; // Full CSR Address range
```

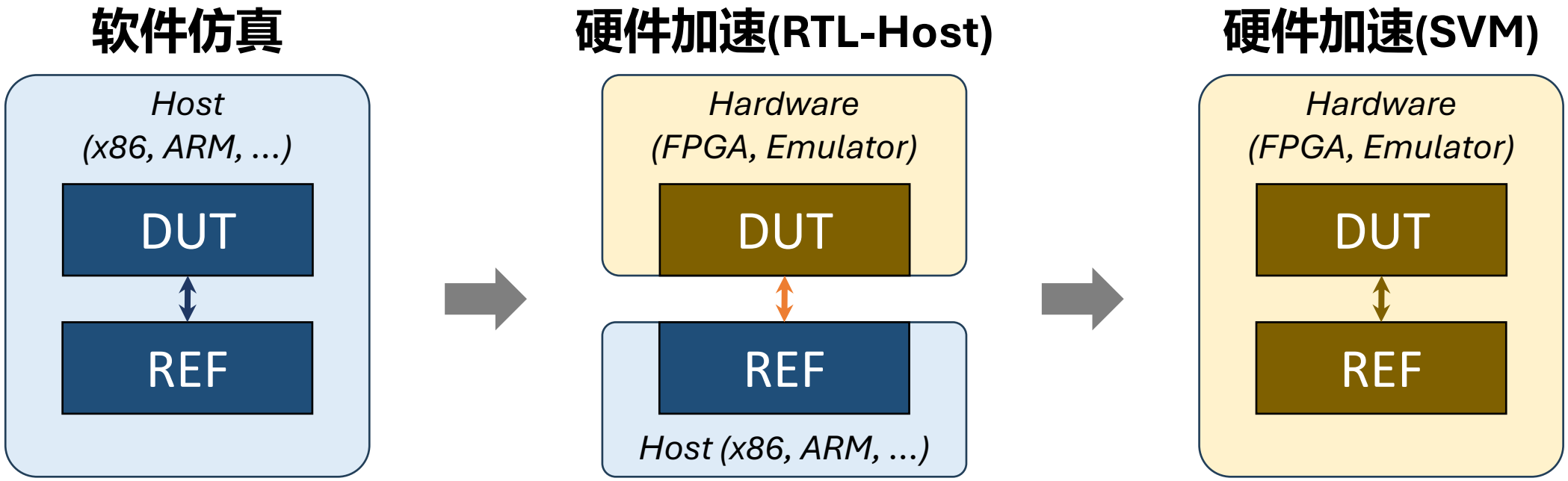
更完善的 Imperas RVVI 检查依赖每次 **145,637 bit** 数据
（针对支持浮点和向量运算的32-bit RISC-V CPU）

RTL-Host 架构下的通信开销问题

- DiffTest 当前对**复杂香山处理器**的验证加速效果**不理想**
- 例：Cadence Palladium 应用效果
  - 尽管已**减少了 99.8%** 的通信需求，仍**未达到理想速度**，且**规模可扩展性差**



本工作：可综合验证方法 (SVM)



Native Slow

Fast yet Link-Constrained

Native Fast

本工作：可综合验证方法 (SVM)

- **另一种不同思路**：**可综合硬件代码**实现所有的协同仿真验证逻辑
  - 完全实现为硬件逻辑后，原有验证数据通信转为片上连线，无开销

- Emulator、FPGA等
- 速度 1~100 MHz

- Emulator、FPGA等

- **实现 SVM 面临的挑战**
  - **编码层**：指令语义复杂，如何确保**硬件 REF 的正确性**?
  - **架构层**：硬件实现约束多，如何确保**硬件 REF 的执行效率**?
  - **运行时**：硬件调试困难，如何提升 **SVM 框架的可调试性**?

挑战1：REF 的电路代码实现

- 现有 REF 通常为软件指令集模拟器，**设计简洁、规整**，保障可靠性
  - 使用 C/C++ 提供的高层次数据结构进行描述
  - 使用更方便的编程方式，如封装、抽象等，降低出错风险

类别	案例	软件描述方式	
指令	add	指令模板 insns	`WRITE_RD(sext_xlen(RS1 + RS2))`
CSR	访问权限	CSR 基类 csr_t	`csr_t::verify_permissions`
	读		`cst_r::read`
	写		`csr_t::write`
常量	ADD指令掩码	常量列表 riscv-opcodes	`#define MATCH_ADD 0x33`
运算	提取/写入字段	标准接口 decode.h	`get_field/set_field`

挑战1：REF 的电路代码实现

- 现有 REF 通常为软件指令集模拟器，**设计简洁、规整**，保障可靠性
  - 使用 C/C++ 提供的高层次数据结构进行描述
  - 使用更方便的编程方式，如封装、抽象等，降低出错风险

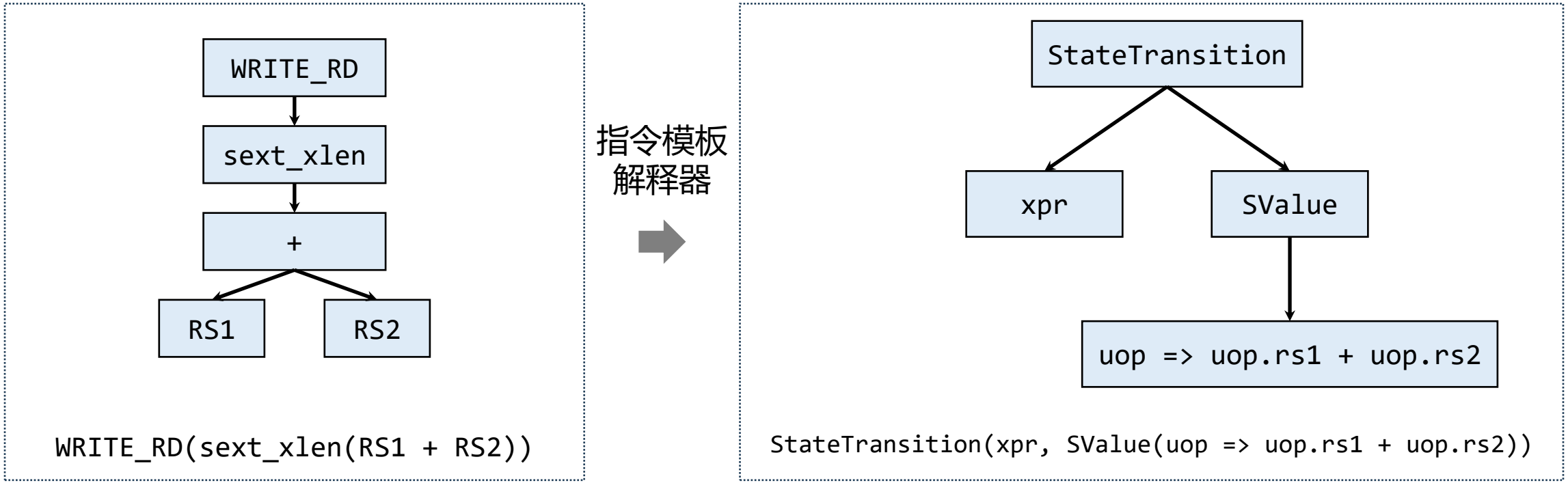
- **问题**：缺少支持**可综合 REF 的高效硬件描述方式**

类别	案例	软件描述方式		硬件描述方式
指令	add	指令模板 insns	`WRITE_RD(sext_xlen(RS1 + RS2))`	???
CSR	访问权限	CSR 基类 csr_t	`csr_t::verify_permissions`	???
	读		`cst_r::read`	
	写		`csr_t::write`	
常量	ADD指令掩码	常量列表 riscv-opcodes	`#define MATCH_ADD 0x33`	???
运算	提取/写入字段	标准接口 decode.h	`get_field/set_field`	???

关键技术1：语义代码迁移技术

- **思路**：**自动化转换**软件 REF 中的关键指令集语义信息

- **例**：构造**指令操作树**完成**指令功能**的迁移



(a) Spike

(b) 本工作

关键技术1：语义代码迁移技术

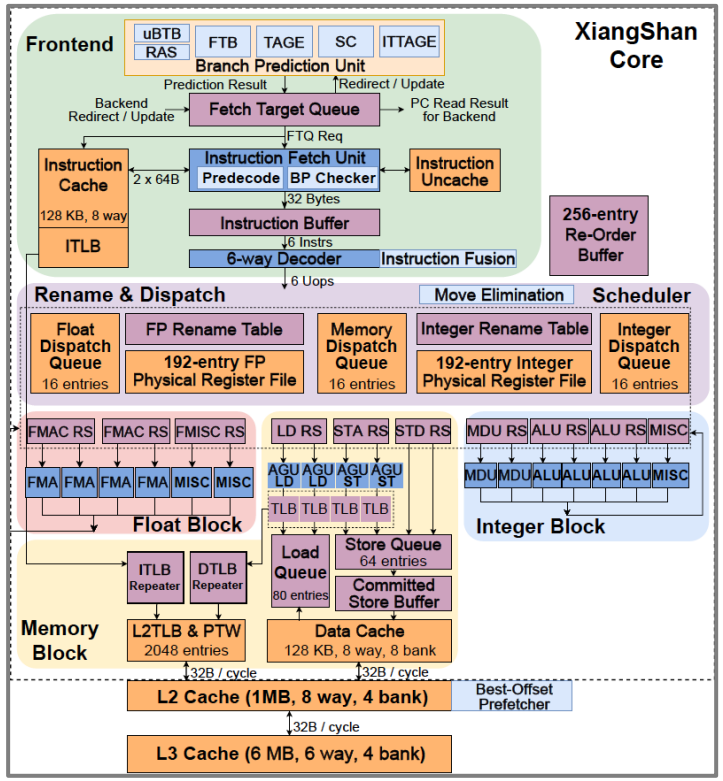
- 支持**指令功能、控制和状态寄存器(CSR)、常量**等语义信息自动迁移
  - 部分提供了和 Spike 类似的 RISC-V REF 编程接口

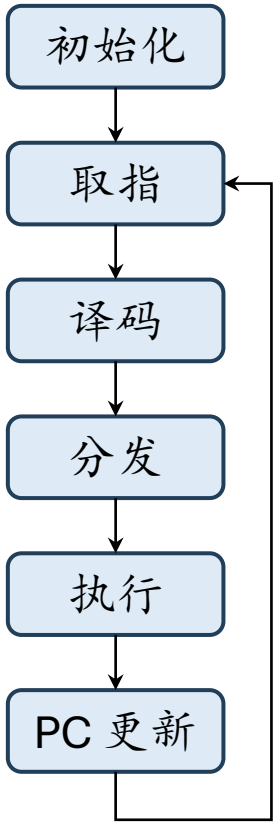
类别	案例	软件描述方式		硬件描述方式
指令	add	指令模板 insns	`WRITE_RD(sext_xlen(RS1 + RS2))`	**指令模板解释器**
CSR	访问权限	CSR基类 csr_t	`csr_t::verify_permissions`	**CSR基类 CSRDef**
	读		`cst_r::read`	
	写		`csr_t::write`	
常量	ADD指令掩码	常量列表 riscv-opcodes	`#define MATCH_ADD 0x33`	**字符串插值**
运算	提取/写入字段	标准接口 decode.h	`get_field/set_field`	**标准接口**

挑战2：硬件 REF 的执行效率

- 协同仿真验证的**指令吞吐**取决于 DUT 和 REF 两者中较小的那个
  - DUT 设计复杂，微结构细节丰富，指令执行效率高
  - **REF 须设计简洁**以保障功能可靠，如何提升其**执行效率**？

DUT 微结构设计复杂





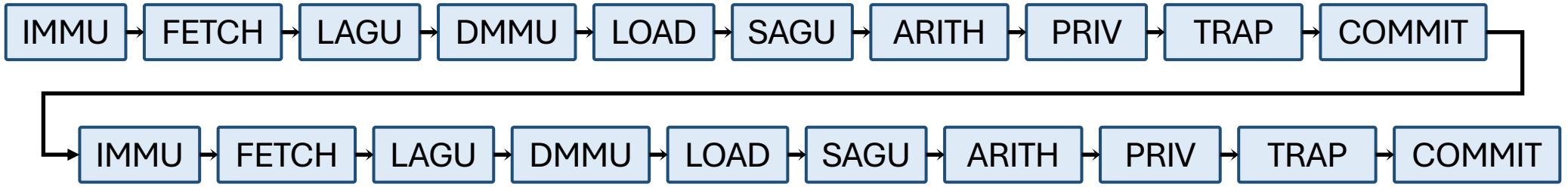
REF 微结构设计简单

关键技术2：硬件参考模型(SRef)设计

- **工作流程**：当 DUT 提交 *N* 条指令，SRef 执行 *N* 条指令，对比结果

关键技术2：硬件参考模型(SRef)设计

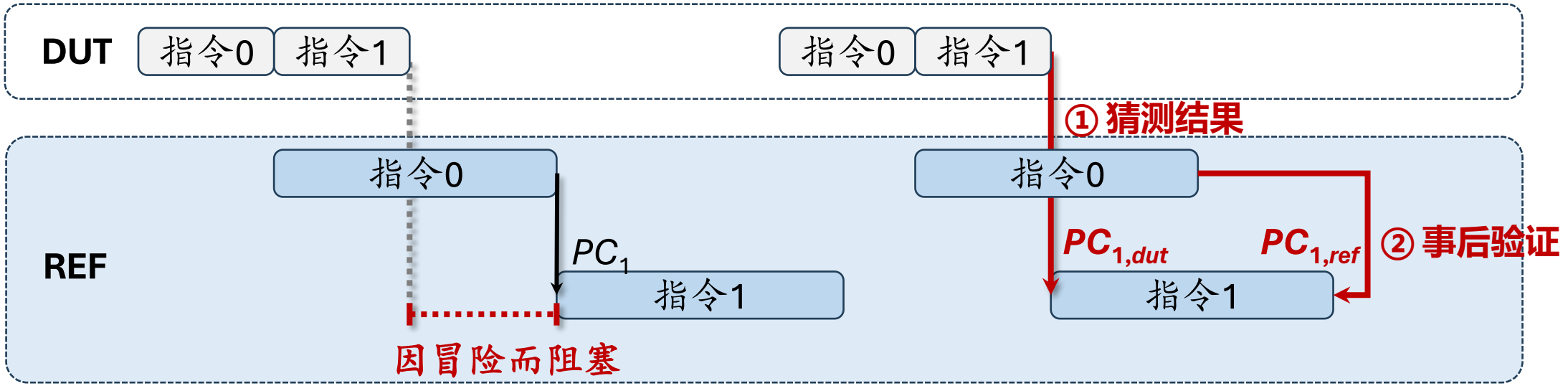
- **工作流程**：当 DUT 提交 *N* 条指令，SRef 执行 *N* 条指令，对比结果
- **简单架构设计：全流水、无阻塞**
  - **预期**：在 *N* * pipeline_length 时钟周期内， SRef 一定能完成指令执行
  - 如果预期成立，那么 SRef 能和任意性能(IPC)的处理器进行协同仿真



例：DUT 提交 2 条指令后，SRef 依次通过各流水级完成 2 条指令执行

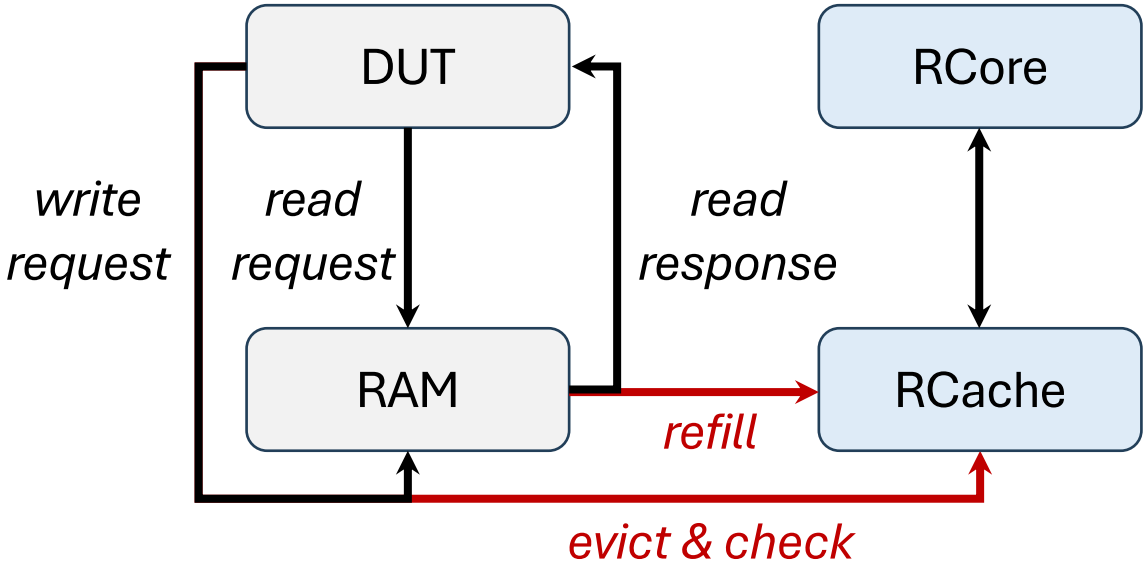
关键技术2：参考模型的主流水线(RCore)设计

- **问题1**：指令间存在**控制和数据冒险(Hazard)**，造成流水线阻塞
- **解决思路**：利用 DUT 信息的**推测执行**机制
  - 利用 DUT 的执行结果，通过**猜测结果 + 事后验证**，消除控制和数据依赖



关键技术2：参考模型的缓存系统(RCache)设计

- **问题2**：内存访问存在**结构冒险**，不足以支撑 REF **访存带宽**需求
- **解决思路**：利用 DUT 访存结果，尽可能避免重复内存访问
  - 等同于：RCache 仅保存和 DUT RAM 不相同的数据块（REF/DUT RAM 差集）
  - DUT 读：内存返回的数据一定是正确的，根据地址回填 RCache
  - DUT 写：确认 RCache 中存在相同数据块并标记为驱逐，数据不同则报错



挑战3：可综合验证框架的可调试性

- 软件验证框架支持**断言、错误日志、计数器**等基本调试方式
  - 但硬件环境缺乏对此的原生支持

调试需求	典型软件机制	现有硬件支持
断言	assert	**有硬件原语，不可综合**
错误日志	display, printf	
计数器	counter	**无硬件原语**
异常处理	signal, sig_handler	
	coredump	

关键技术3：可综合验证调试技术

- 软件验证框架支持**断言、错误日志、计数器**等基本调试方式
  - 但硬件环境缺乏对此的原生支持
- **解决思路**：**REF 可被转换为独立执行的通用 CPU 用于调试**
  - 基于硬件化断言、结果检查等条件进行报错，并切换 REF 状态
  - 验证模式：用于 DUT 执行结果的正确性检查
  - 调试模式：用于出错后的错误现场调试

调试需求	典型软件机制	本工作
断言	assert	**硬件化断言提取器**
错误日志	display, printf	**调试模式 + 串口**
计数器	counter	**硬件化计数器**
异常处理	signal, sig_handler	**状态控制和检查器**
	coredump	**状态记录器**

SVM 工具链

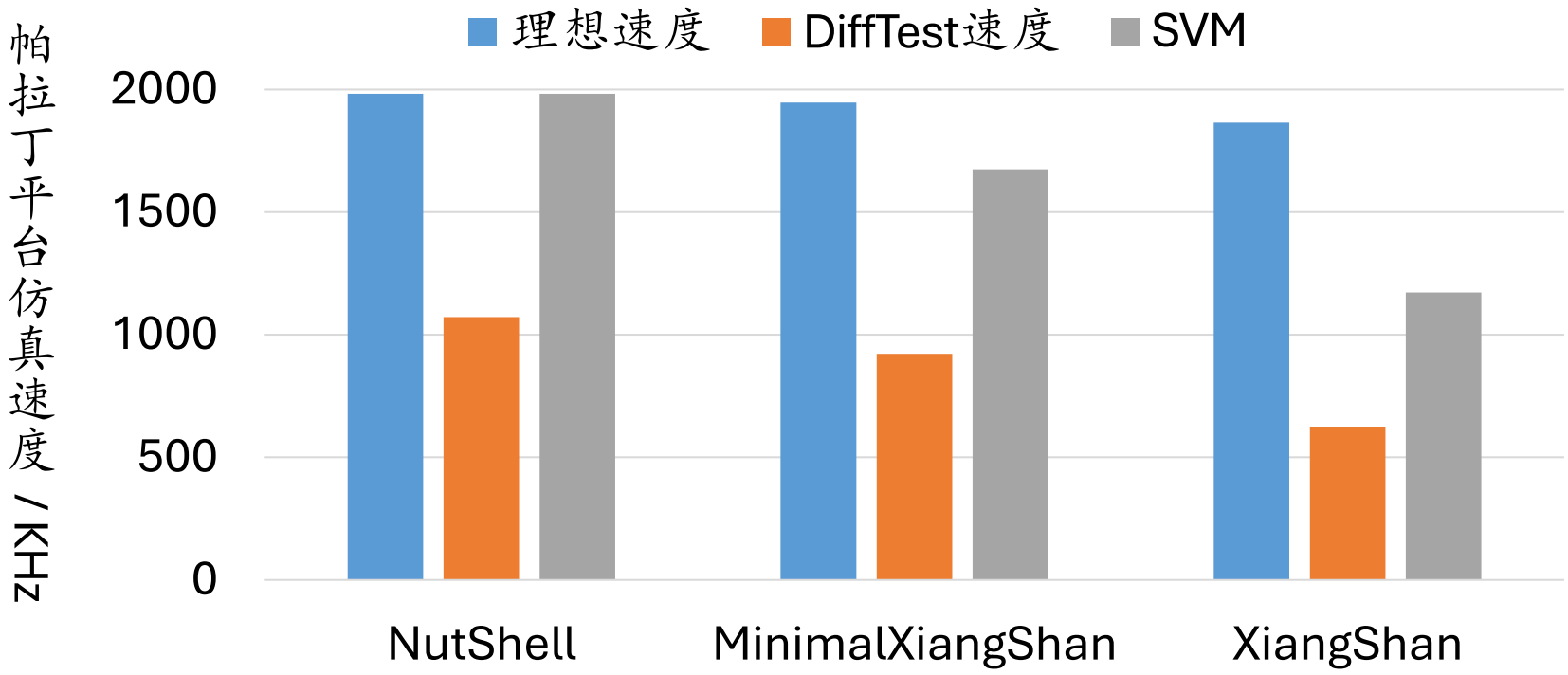
- 本工作实现了一套面向RISC-V处理器的**可综合验证工具链**
  - 即将整理开源

- **兼容 DiffTest[1] 的用户侧接口**，接管其**仿真逻辑**
  - 支持不同配置的处理器，如香山、果壳等
  - 支持 RISC-V 指令集的 I、M、A、C、B 扩展指令
  - 支持 RISC-V 指令集的 M、S、U 特权模式和对应特权操作
  - 支持自动解析 Spike 指令集模拟器源码中的语义信息

[1] https://github.com/OpenXiangShan/difftest

实验评估：接近硬件平台理想仿真速度

- 将 SVM 用于不同配置的**香山、果壳处理器**协同仿真验证
  - **FPGA**[1]：果壳，使用 512KB RCache，速度达 **60MHz**，比 DiffTest **快 10 倍**
  - **帕拉丁**：果壳最高达 **1.9MHz**，与理想速度一致，比 DiffTest **快 90%**



[1] 由于容量限制，当前Xilinx zu19eg FPGA仅实现果壳处理器的SVM验证

谢谢！请批评指正！欢迎交流！