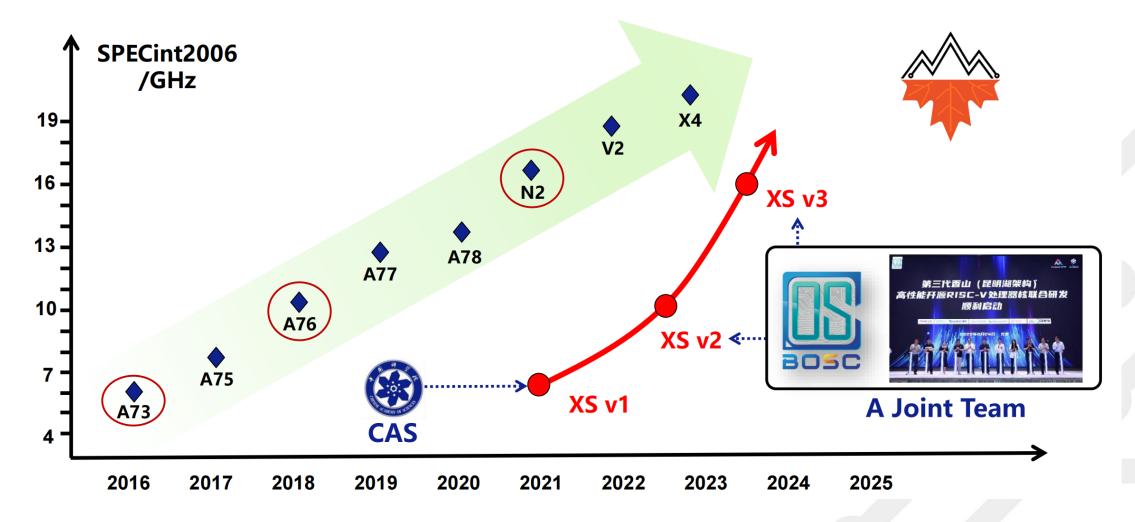


## 面向Palladium的高性能处理器 软硬件协同验证框架部署与加速方法

蔡洛姗 游昆霖

#### Who Are We?





#### 香山:开源高性能 RISC-V 处理器

#### 1st generation: YQH

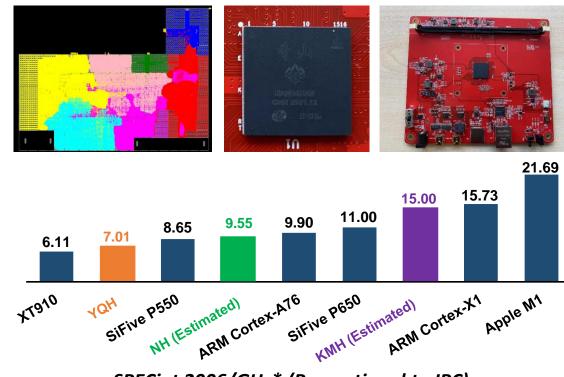
- RV64GC, single-core, superscalar OoO
- 28nm tape-out, 1.3GHz, July 2021
- SPEC CPU2006 7.01@1GHz, DDR4-1600

#### 2<sup>nd</sup> generation: NH

- RV64GCBK, dual-core, superscalar OoO
- 14nm GDS delivery, 2GHz, 2023 Q3
- Estimated\*\* SPECint 2006 19.10@2GHz

#### 3<sup>rd</sup> generation: KMH

- RV64GCBKHV, quad-core, superscalar OoO
- Advanced-node, 3GHz, 1.5x IPC of NH
- Close collaboration with industrial partners



SPECint 2006/GHz\* (Proportional to IPC)



















有道 youdao















## 高性能处理器发展的"灰犀牛": 验证

**50%** 

146%

>50%

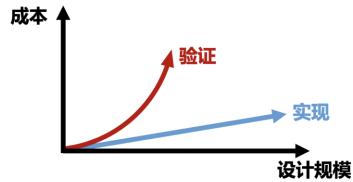
Increase in <u>design</u> engineers since 2007

Increase in <u>verification</u> engineers since 2007

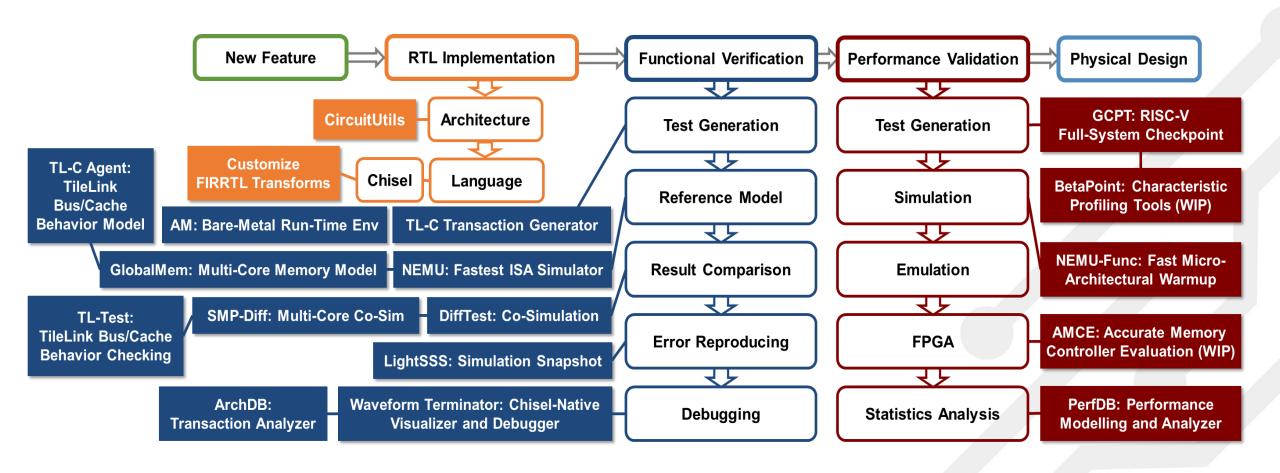
Median project <u>time</u> spent in <u>verification</u>

- ・验证是处理器开发流程中耗时最长的一个阶段
- 随着设计效率的提高,验证将成为处理器开发的瓶颈

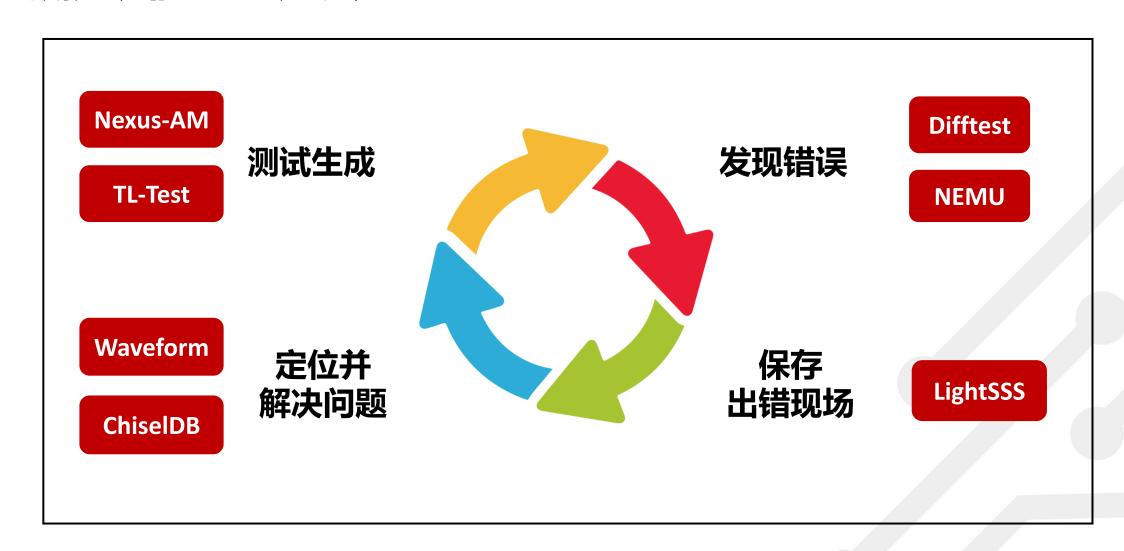
Amdahl's Law: 
$$S_{latency}(s) = \frac{1}{(1-p) + \frac{p}{s}} < \frac{1}{1-p}$$



#### 敏捷设计与验证流程



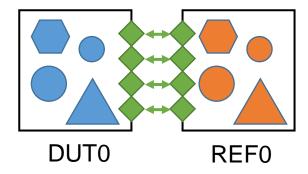
## 敏捷功能验证方法

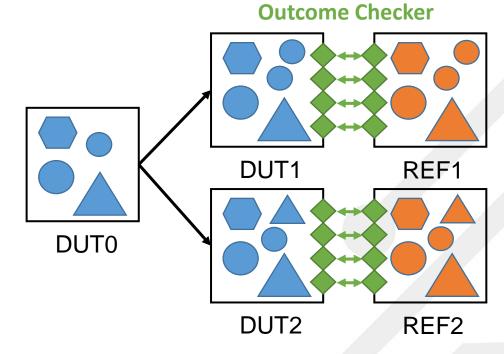


#### 传统验证方法的不敏捷来源

- 核心来源:基于**待测设计与参考模型输出结果的一致性**表征正确性
  - · 待测设计 (design-under-test, DUT): 待验证的硬件设计,通常基于RTL实现
  - · 参考模型 (reference model, REF):被假定为正确的行为模型,通常基于SV/C/C++实现







①参考模型需要包含大量设计细节

②设计不断变化时,参考模型需要不断更新

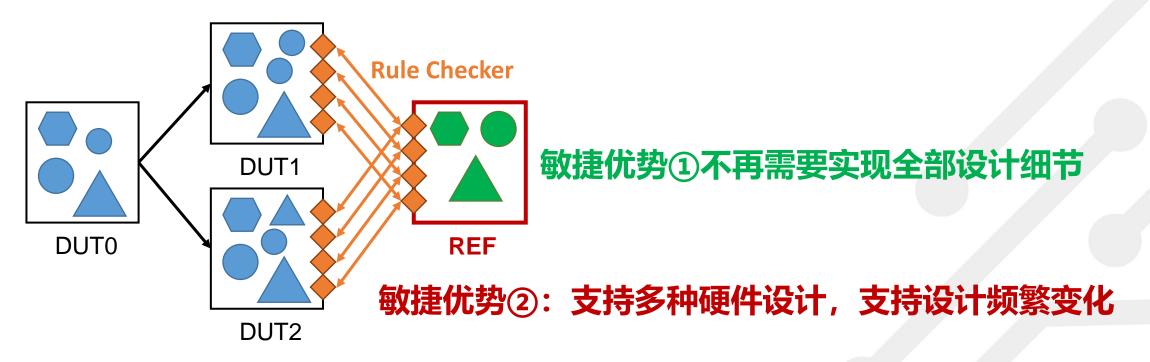


#### 基于规则的敏捷验证方法

•核心思想:基于合法规则的匹配表征正确性

• 不再追求待测设计 (DUT) 与参考模型 (REF) 的完全一致

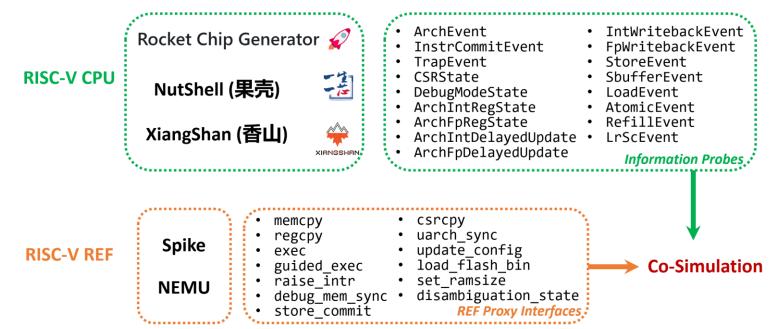
• 原理:在不违背语义正确性的前提下,允许REF跟随DUT的变化

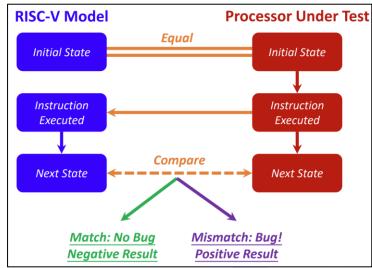


#### Difftest: 指令级协同仿真差分验证框架

#### ・基本流程

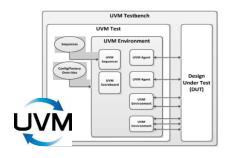
- 处理器仿真产生指令提交/其他状态更新
- 模拟器执行相同的指令
- 利用规则比较两者状态, 报错或继续





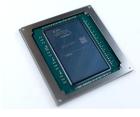


## 多层级的硅前验证平台









#### **Formal Analysis**

- + detect bug early
- + high coverage
- complex for large designs

#### **SW Simulation**

- + flexible
- slow for large designs

#### **Emulation**

- + faster
- + debug like simulation
- cost

#### **FPGA**

- + inexpensive
- + fast
- little debug
- slow compile

**Earlier Flexibility** 

**Higher Speed** 

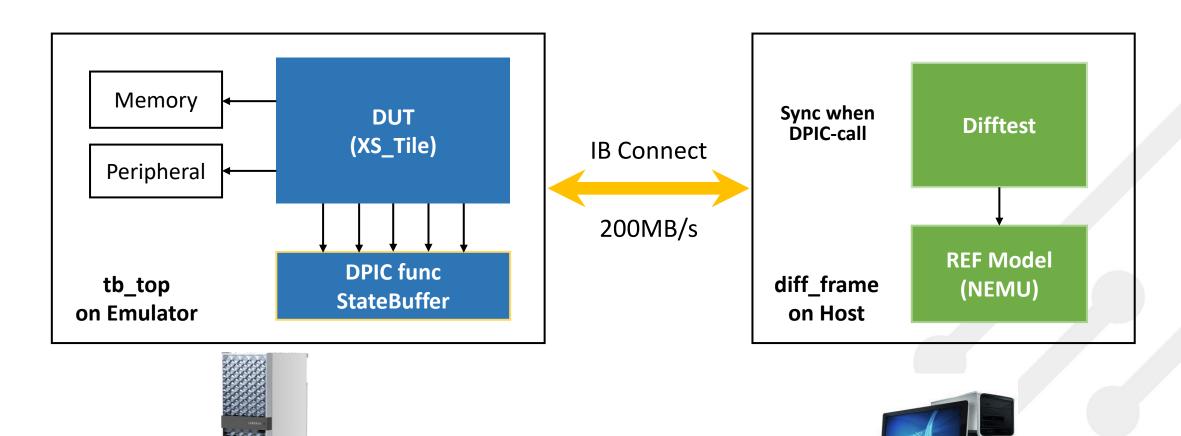


## **Why Palladium**

- 快速编译
- 运行速度快
- 调试手段多样

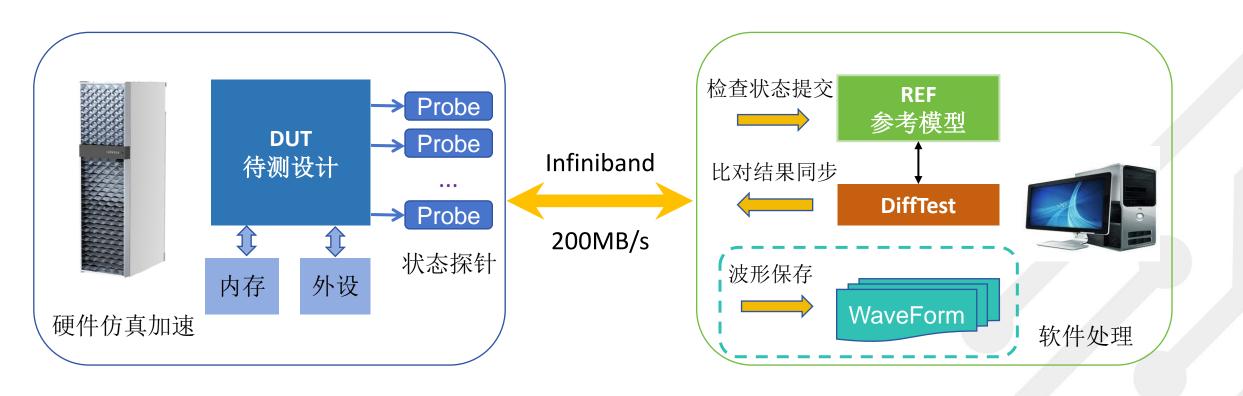
	Verilator	Palladium	FPGA
编译时间	30min	30min	~ 10h
运行速度	3 kHz	1.95 MHz	50 MHz
调试手段	Difftest,	Waveform,	ILA,
	Waveform,	Dynamic Events	JTAG,
	LightSSS,	trigger,	Snapshot
	ChiselDB	SDL (State	
		Definition	
		Language),	
		Difftest	

## 基于 Palladium 的 Difftest 软硬件协同验证

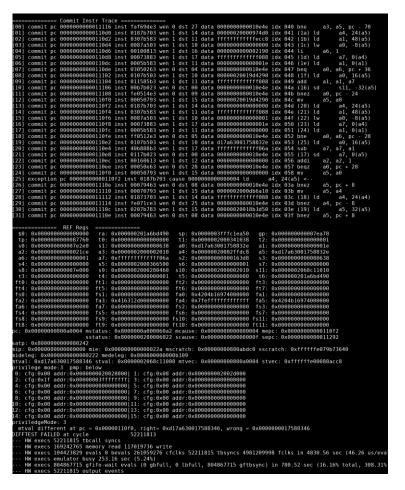




## 基于 Palladium 的 Difftest 软硬件协同验证



## 敏捷验证流程 (演示用例)



指令级错误行为比对及定位

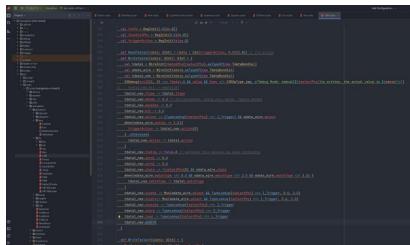
DiffTest 敏捷验证 错误定位 WaveForm 波形调试

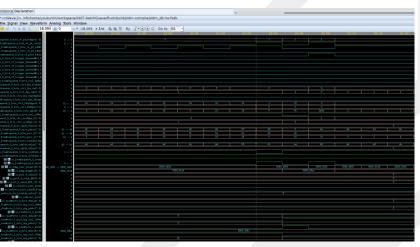
指令集行为规范

设计修改

Chisel

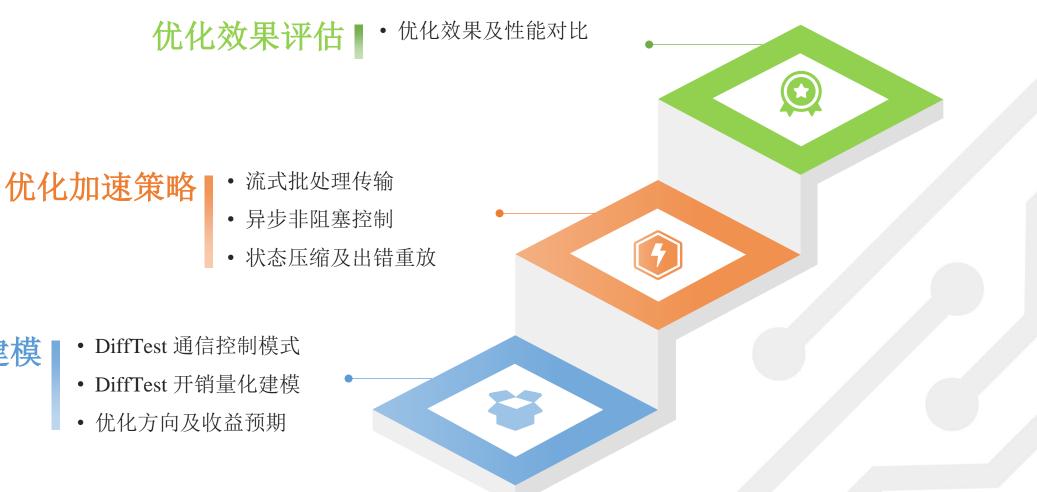
波形实现细节







## 面向 Palladium 的 DiffTest 加速



#### 开销量化建模▮

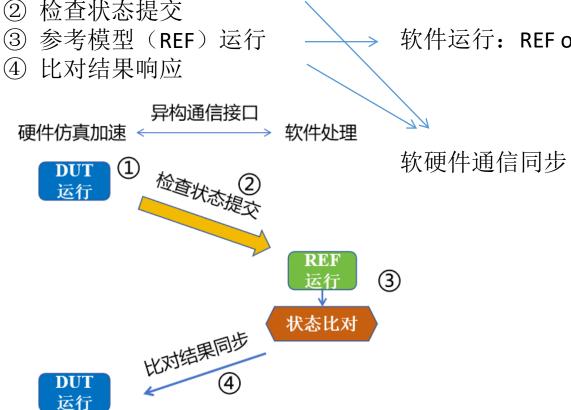
- DiffTest 通信控制模式
- DiffTest 开销量化建模
- 优化方向及收益预期

#### DiffTest 通信控制模式

#### DiffTest 的一次检查流程

- ① 待测设计(DUT)仿真
- ② 检查状态提交

时间



硬件仿真: DUT on Emulator  $(\sim 0.8 \text{ MHz})$ 

软件运行: REF on Host Computer (~ 100 MHz)

传递 DiffTest 所有检查信息

单核运行: 指令提交/寄存器状态/

异常中断/外设访问 ...

多核一致性: 共享内存维护/

Cache 重填检查 ...

状态比对结果相应

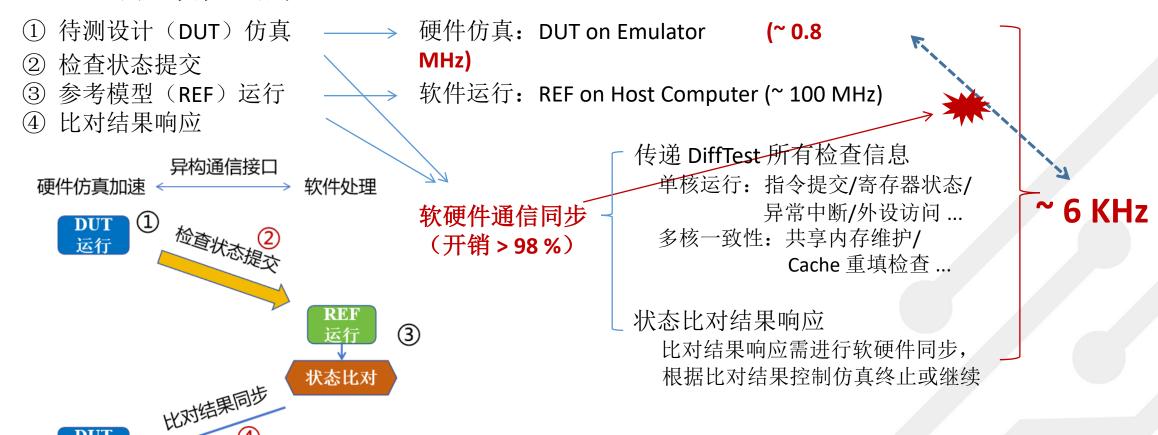
比对结果响应需进行软硬件同步, 根据比对结果控制仿真终止或继续

#### DiffTest 通信控制模式

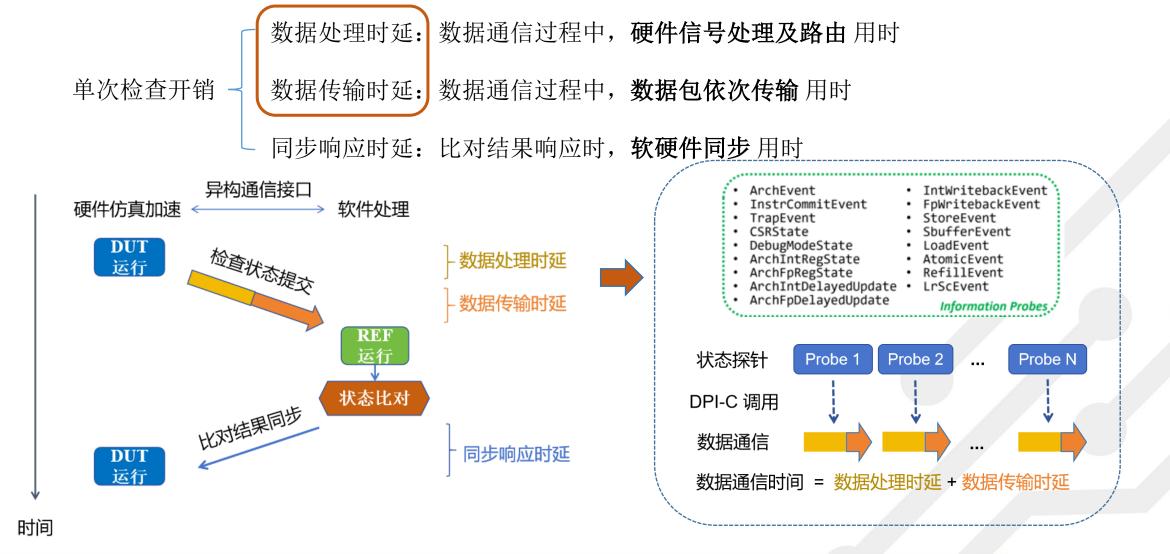
#### DiffTest 的一次检查流程

DUT 运行

时间



## DiffTest 开销量化建模 - 单次检查开销构成



## DiffTest 开销量化建模 - 单次检查开销构成

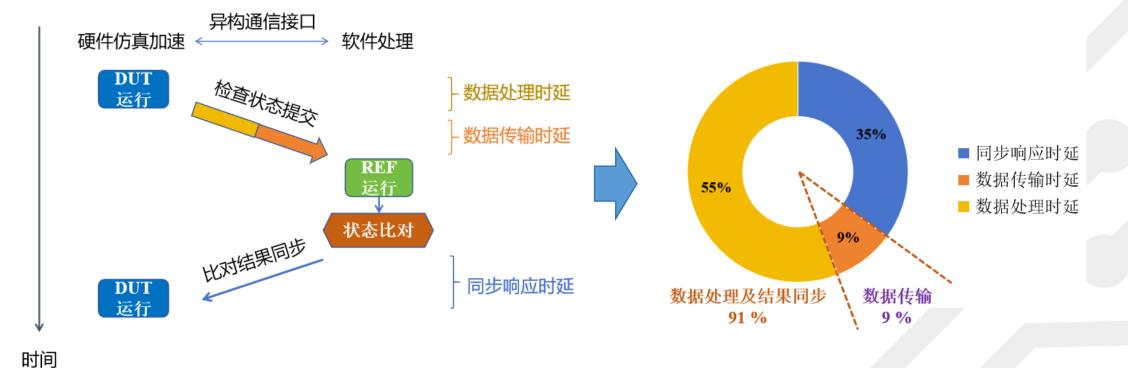
数据处理时延:数据通信过程中,硬件信号处理及路由用时

单次检查开销

数据传输时延:数据通信过程中,数据包依次传输用时

同步响应时延:比对结果响应时,软硬件同步用时

55 %9 %36 %



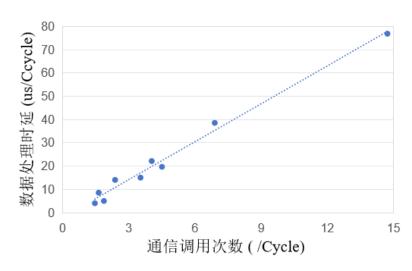
## DiffTest 开销量化建模 - 单次检查开销构成

数据处理时延:数据通信过程中,硬件信号处理及路由用时

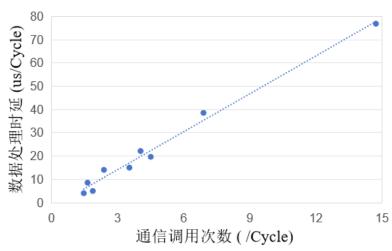
单次检查开销

数据传输时延:数据通信过程中,数据包依次传输用时

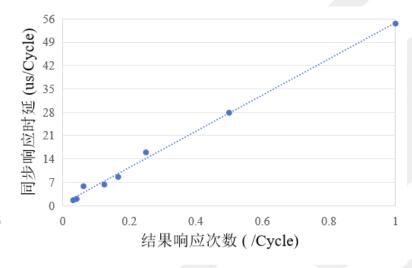
同步响应时延:比对结果响应时,软硬件同步用时



**数据处理时延** 通信调用次数 × 单次处理延迟



**数据传输时延** 传输数据量 × 数据传输速率



**同步响应时延** 结果响应次数 × 单次同步延迟

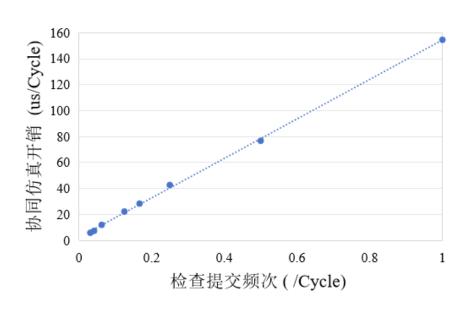


#### DiffTest 开销量化建模 - 检查步骤重复

协同仿真开销

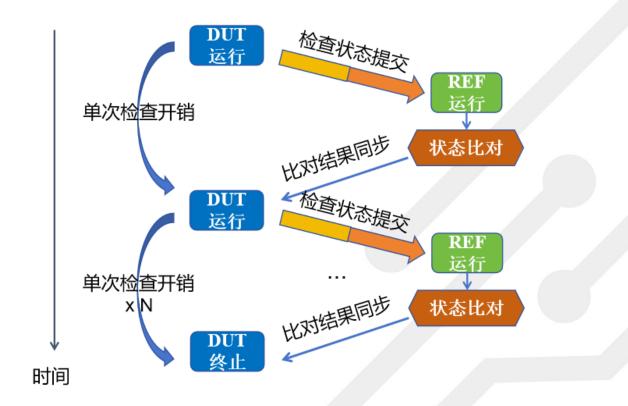
检查提交频次: 平均每个处理器周期 检查状态提交次数

单次检查开销:一次检查状态提交-比对结果同步所需时间



协同仿真开销

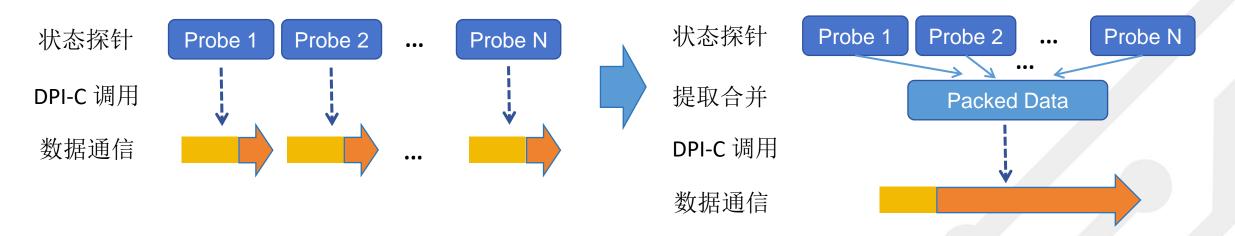
检查提交频次×单次检查开销



#### DiffTest 优化加速策略 - 数据处理时延

(55 %) (9 %) (36 %) (36 %) (10 h) (1

通信调用次数×单次处理延迟



优化思路: 提取合并探针数据,减少通信调用次数

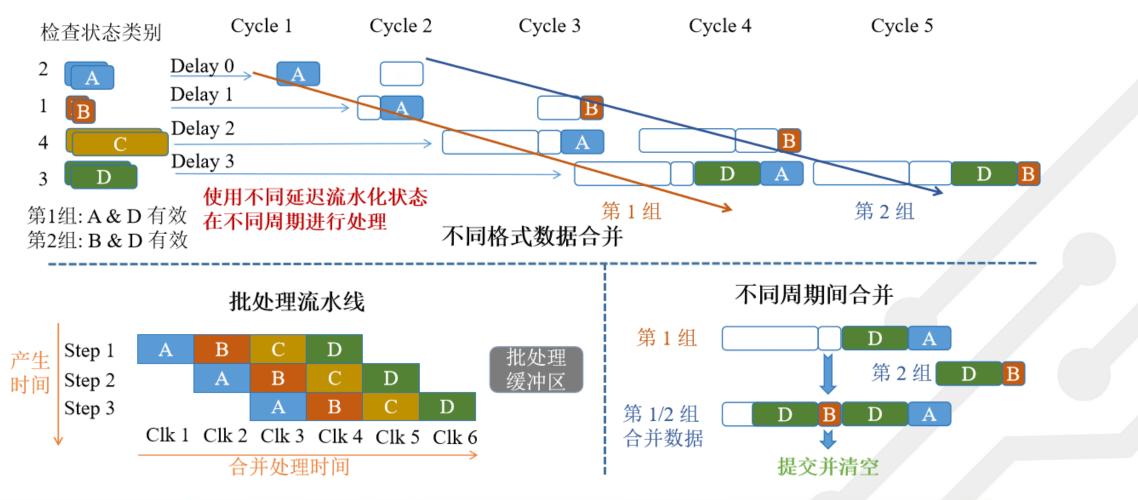
关键问题:一、有效数据提取合并的硬件实现

二、合并数据软件侧拆分及格式恢复



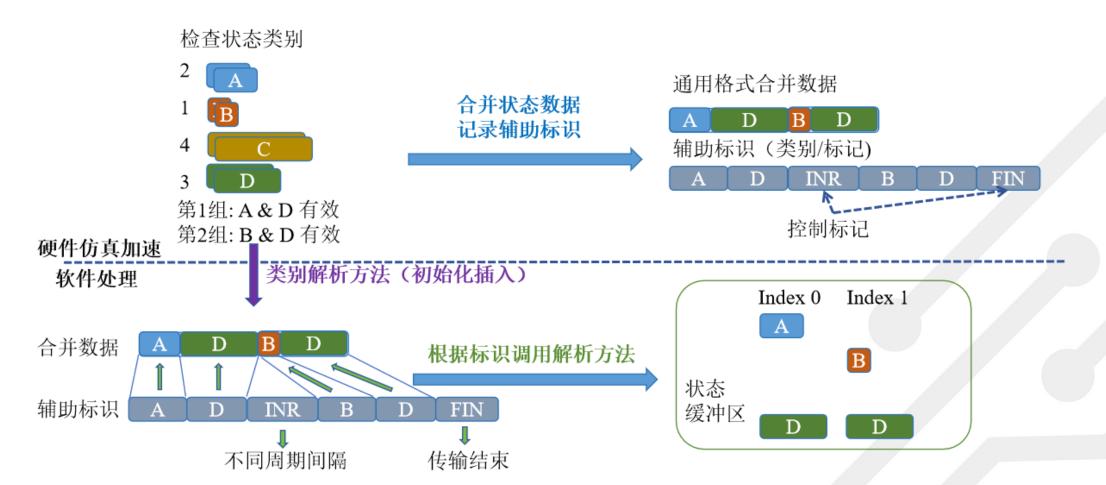
#### DiffTest 优化加速策略 - 流式批处理传输

问题一: 有效数据提取合并的硬件实现 —— 流式有效值合并策略



#### DiffTest 优化加速策略 - 流式批处理传输

问题二: 合并数据在软件侧拆分及格式恢复 —— 异构数据一致性表示

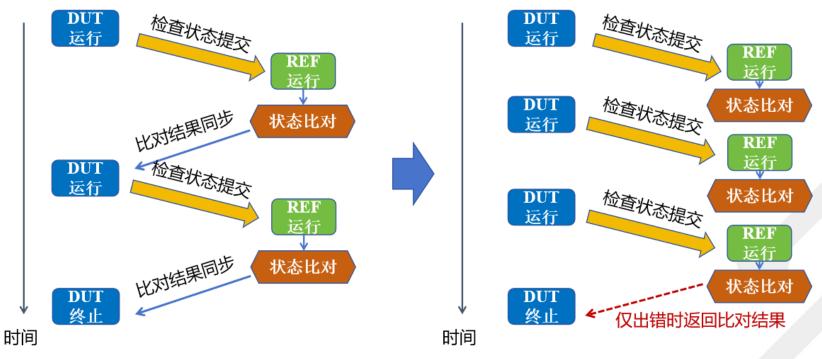


#### DiffTest 优化加速策略 - 同步响应时延

 55 %
 9 %
 36 %

 协同仿真开销 = 检查提交频次 × (数据处理时延 + 数据传输时延 + 同步响应时延)

**结果响应次数** × 单次同步延迟



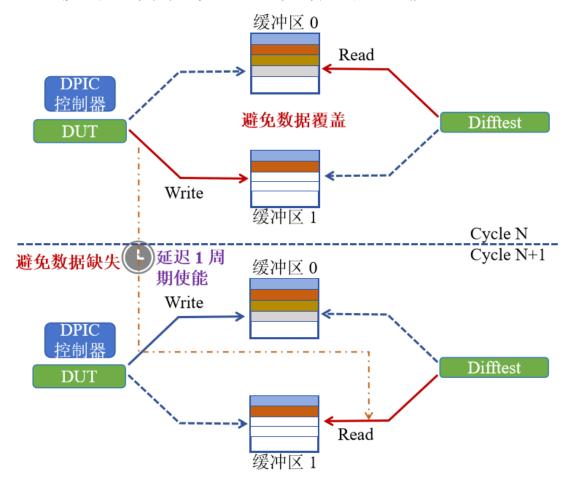
优化思路: 仅在出错时返回结果并终止仿真,减少结果响应次数

关键问题: 缓冲区并行读写的数据丢失或覆盖



#### DiffTest 优化加速策略 - 异步非阻塞控制

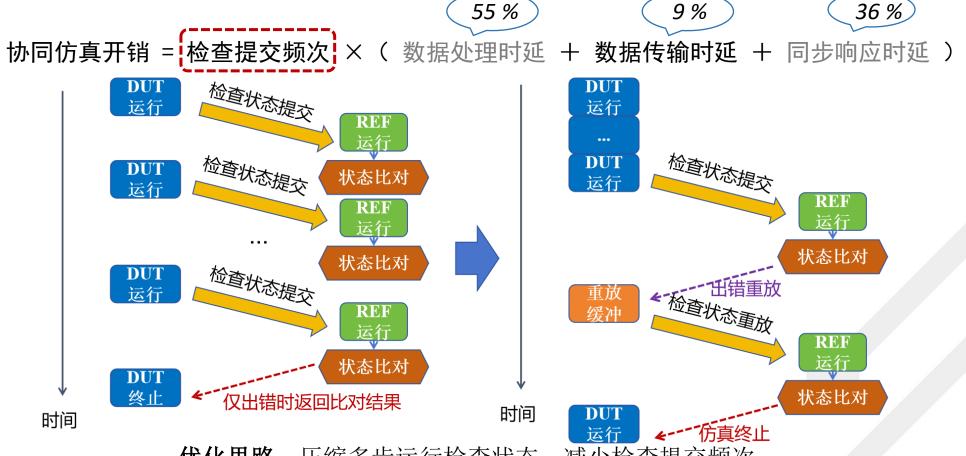
问题:缓冲区并行读写的数据丢失或覆盖 ——"乒乓"缓冲区



软件处理过程中,数据传输至其余缓冲区, **避免数据覆盖** 

数据传输完毕的下一处理器周期 使能软件比对,**避免数据缺失** 

#### DiffTest 优化加速策略 - 检查提交频次



优化思路: 压缩多步运行检查状态,减少检查提交频次

关键问题:一、状态压缩中的依赖关系与不可压缩状态

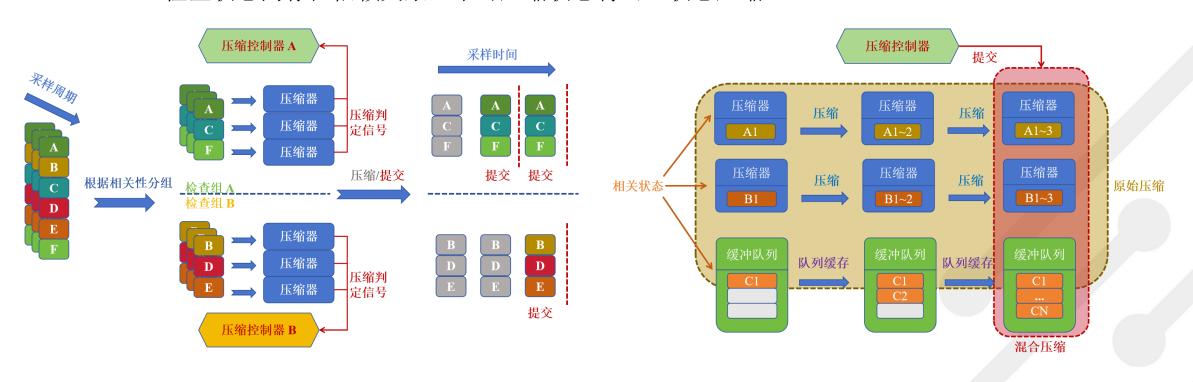
二、出错时压缩状态的调试定位



#### DiffTest 优化加速策略 - 状态压缩及出错重放

#### 问题一: 硬件侧检查状态压缩 —— 基于状态语义的压缩策略

检查状态间存在依赖关系,不可压缩状态将终止状态压缩



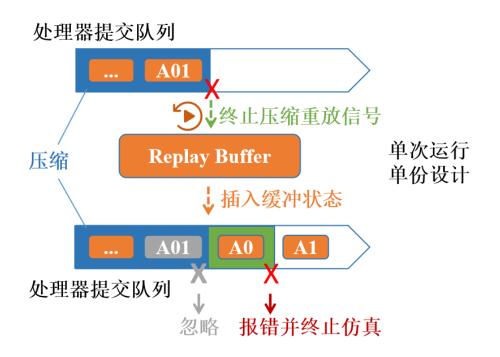
相关性分组,减少状态依赖影响

缓冲 不可压缩状态, 提升压缩率



#### DiffTest 优化加速策略 - 状态压缩及出错重放

问题二: 出错时压缩状态的调试定位 —— 状态缓冲, 出错重放



软件侧返回压缩指令重放范围, 重新提交未压缩指令

## DiffTest 优化效果



▶ 协同仿真通信同步开销: 155 us/Cycle 0.4 us/Cycle 降低 362.3 倍

► 占总仿真时间比例: 99.1 % 18.8 % **己消除主要开销** 

▶ 仿真速度: 6.4 KHz 563.4 KHz **提升 88.7 倍** 



"香山"处理器,完整协同仿真验证,通用验证框架





**216.7** 倍软件仿真性能(AMD 7950X3D, 96M L3, 8 线程 Verilator)



# Thanks 納納