中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES
中国开放指令生态（RISC-V）联盟
China RISC-V Alliance
北京开源芯片研究院
BEIJING INSTITUTE OF OPEN SOURCE CHIP

香山处理器 Tutorial

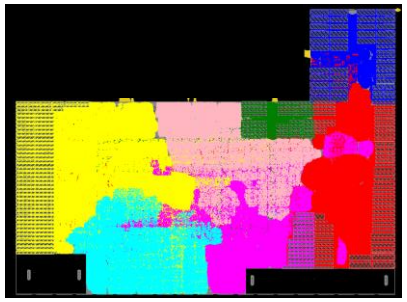
高泽宇、唐浩晋、蔡洛姗、徐泽凡、蒋利杨
中国科学院计算技术研究所

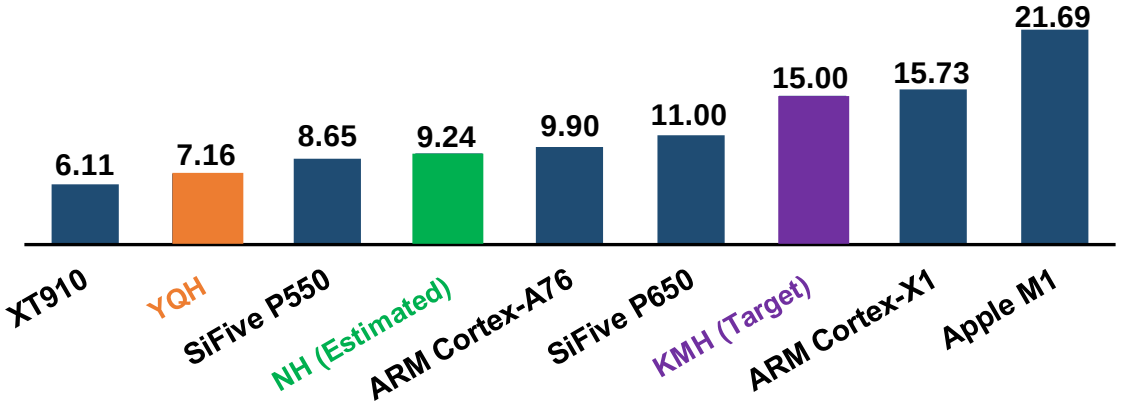
2023.08.24

目录

香山：开源高性能 RISC-V 处理器

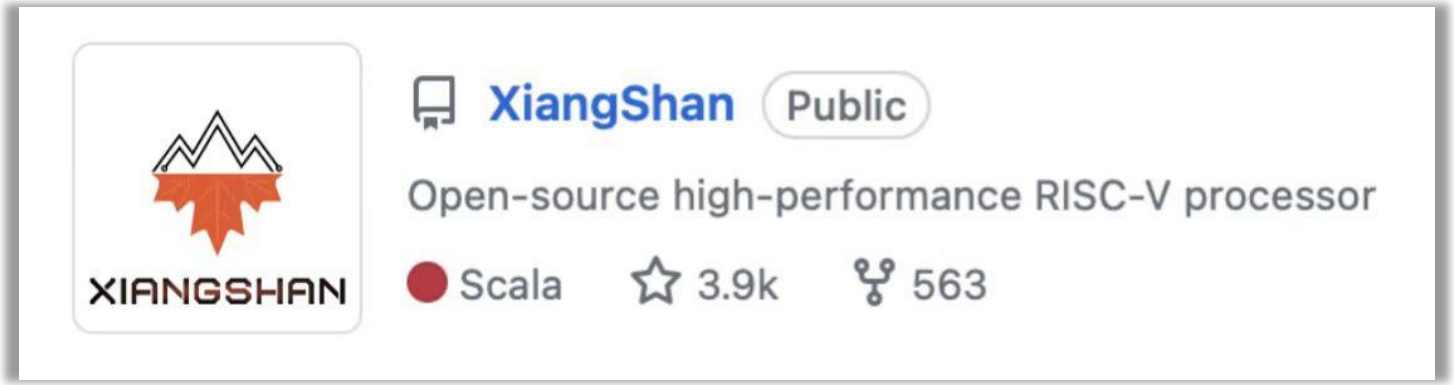
- **第一代：雁栖湖 架构**
  - RV64GC，单核，超标量乱序执行
  - **28nm，1.3GHz，2021年7月**
  - **SPEC CPU2006 7.01@1GHz**
- **第二代：南湖 架构**
  - RV64GCBK，双核，超标量乱序执行
  - **14nm，2GHz，2023年中投片**
  - **预估 SPEC 2006 20@2GHz**
- **第三代：昆明湖 架构**
  - RV64GCBKHV，四核，超标量乱序执行
  - 与工业界企业伙伴紧密协作





香山：开源高性能 RISC-V 处理器

- 使用Chisel硬件设计语言实现
- 包含差分测试框架（DiffTest）、仿真快照、检查点等开发工具
- 建立了包含设计、实现、验证在内的全开源工具敏捷开发流程

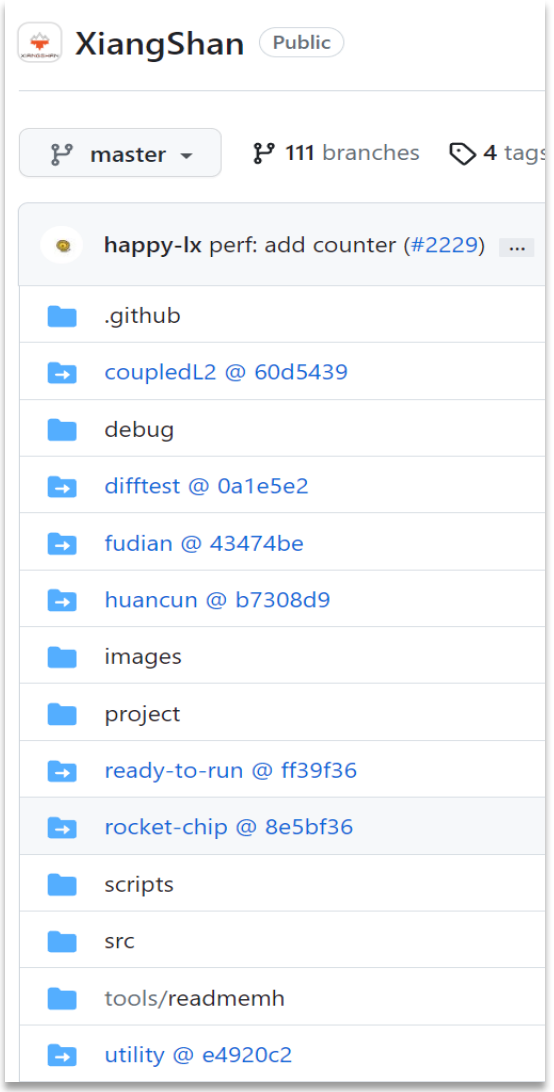


主仓库在全球最大开源项目托管平台 GitHub
已获得超过 3900 个星标，形成超过 500 分支
https://github.com/OpenXiangShan/XiangShan

微信公众号

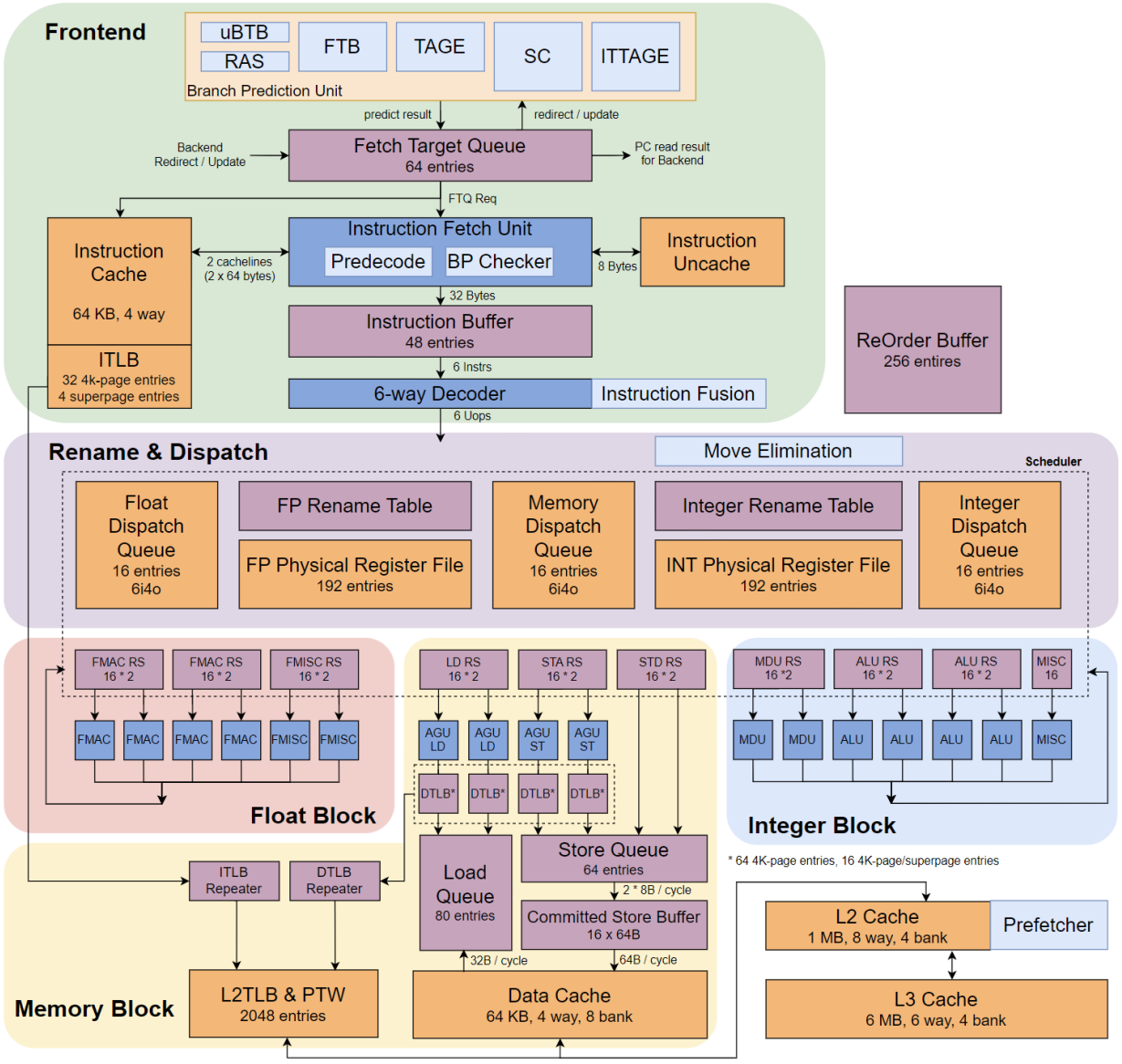
项目结构

```
.
├── scripts               # 生成 Verilog 及仿真使用的一些脚本
├── src                   # 结构设计与验证代码
│   └── main                # 结构设计代码
│       └── scala
│           ├── device          # 仿真用的一些外设
│           ├── system          # SoC 的描述
│           ├── top             # 顶层文件
│           ├── utils           # 一些基础硬件工具库
│           ├── xiangshan       # 香山 CPU 部分的设计代码
│           └── xstransforms    # 一些 FIRRTL Transforms
├── fudian                # 香山浮点子模块
├── huancun               # 香山 L2/L3 缓存子模块
├── difftest              # 香山协同仿真框架
├── ready-to-run          # 预先编译好的 nemu 动态链接库，和一些负载
└── rocket-chip           # 用来获取 Diplomacy 框架（等待上游拆分）
```

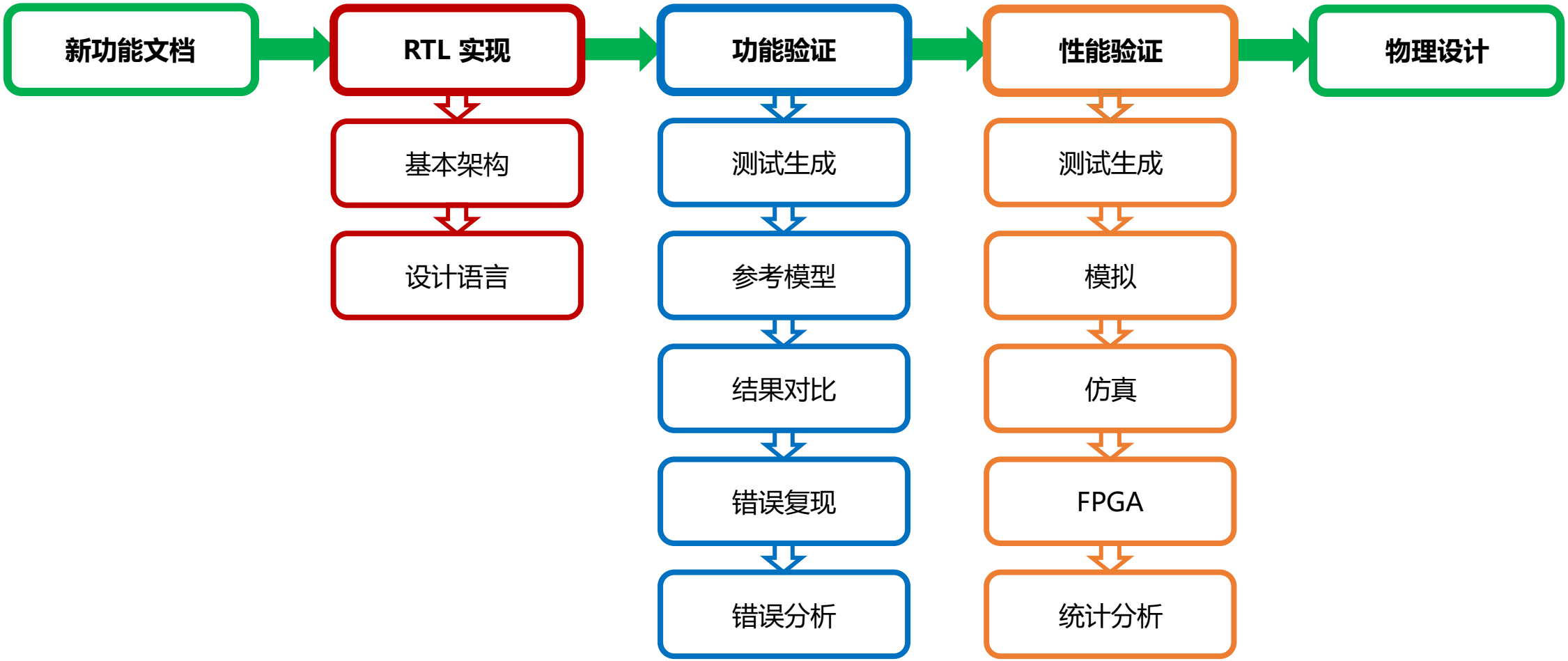


逻辑结构

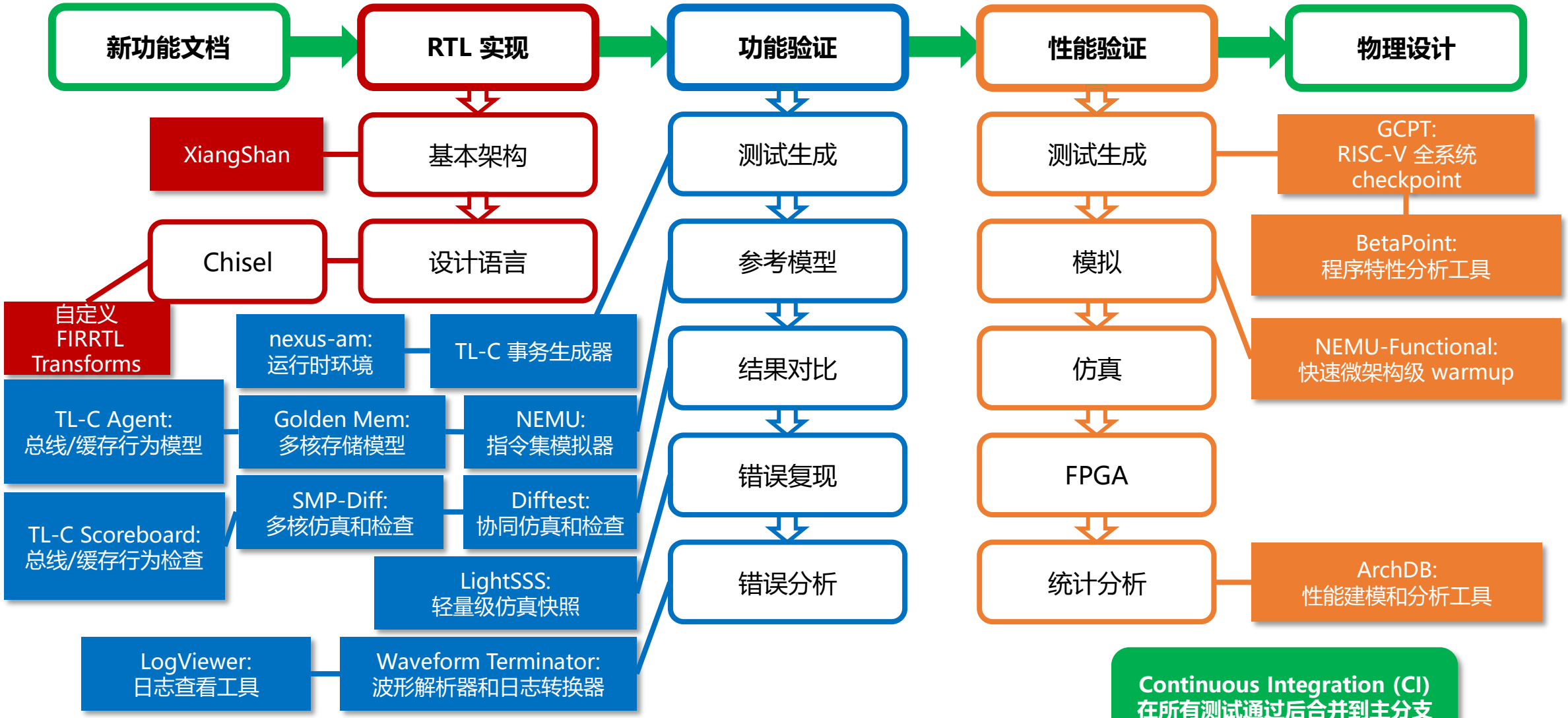
- 前端
  - BPU、FTQ、IFU、IBuffer、ICache
- 后端
  - CtrlBlock、IntBlock、FloatBlock
- 访存
  - MemBlock 、MMU、DCache
- 缓存
  - L2/L3 Cache



敏捷开发流程和工具


新功能文档
RTL 实现
基本架构
设计语言
功能验证
测试生成
参考模型
结果对比
错误复现
错误分析
性能验证
测试生成
模拟
仿真
FPGA
统计分析
物理设计

敏捷开发流程和工具


新功能文档
RTL 实现
功能验证
性能验证
物理设计
XiangShan
基本架构
Chisel
设计语言
自定义 FIRRTL Transforms
nexus-am: 运行时环境
TL-C 事务生成器
TL-C Agent: 总线/缓存行为模型
Golden Mem: 多核存储模型
NEMU: 指令集模拟器
TL-C Scoreboard: 总线/缓存行为检查
SMP-Diff: 多核仿真和检查
Difftest: 协同仿真和检查
LightSSS: 轻量级仿真快照
LogViewer: 日志查看工具
Waveform Terminator: 波形解析器和日志转换器
测试生成
参考模型
结果对比
错误复现
错误分析
测试生成
模拟
仿真
FPGA
统计分析
GCPT: RISC-V 全系统 checkpoint
BetaPoint: 程序特性分析工具
NEMU-Functional: 快速微架构级 warmup
ArchDB: 性能建模和分析工具
Continuous Integration (CI)
在所有测试通过后合并到主分支

在本次 tutorial 中，我们将

- 提供对云服务器的访问
- 准备好香山的开发环境
- 现场演示香山的开发工作流程

仿真

- 环境
- RTL 生成
- RTL 仿真

……

功能验证

- Nexus-AM
- NEMU
- Difftest

……

性能验证

- Perf. counter
- Constin
- Checkpoint

……

- ***需要：网络和 SSH***

演示指令

- shell 命令在**紫色框**中以前缀 $ 显示

```
$ echo "Hello, XiangShan"
$ echo "Have a nice day"
```

- **描述和注释**在**灰色框**中以前缀 # 显示

```
# Please prepare a laptop with an SSH client.
# Next, let's start the demo session !
```

现在试试看吧！

准备工作

- 登录云服务器
  - 对于 Windows 用户（推荐使用 Windows Terminal 和 PowerShell）

```
PS > curl https://openxiangshan.github.io/tutorial.pem -o tutorial.pem
PS > ssh -i "tutorial.pem" guest@123.56.80.238
```

  - 对于 Linux 和 Mac 用户

```
$ curl https://openxiangshan.github.io/tutorial.pem -o tutorial.pem
$ chmod 400 tutorial.pem
$ ssh -i "tutorial.pem" guest@123.56.80.238
```

准备工作

```
# 复制开发环境 xs-env 到您的路径
$ cp -r /opt/xs-env ~/<YOUR_NAME>

# 进入开发环境目录
$ cd ~/<YOUR_NAME>

# 设置环境变量
$ source env.sh
# SET XS_PROJECT_ROOT:                    /home/guest/YOUR_NAME
# SET NOOP_HOME (XiangShan RTL Home):     $XS_PROJECT_ROOT/XiangShan
# SET NEMU_HOME:                          $XS_PROJECT_ROOT/NEMU
# SET AM_HOME:                            $XS_PROJECT_ROOT/nexus-am
```

准备工作

```
# Project Structure
$ tree -d -L 1
.
├── NEMU
├── NEMU-ahead
├── XiangShan
├── XiangShan-Oracle
├── nexus-am
└── tutorial

# 进入 XiangShan 目录
$ cd XiangShan
```

Chisel 编译

- 使用 Verilator 编译 RTL 并构建仿真模拟器

```
$ make emu EMU_THREADS=2 -j8 &   # Run in background

Options:

    CONFIG=TutorialConfig        Configuration of XiangShan

    EMU_THREADS=2                Simulation threads

    EMU_TRACE=1                  Enable waveform dumping

    // WITH_DRAMSIM=1            Enable DRAMSim3 for DRAM simulation

    // WITH_CHISELDB = 1         Enable ChiselDB feature

    // WITH_CONSTANTIN = 1       Enable Constantin feature
```

- 编译大概需要 30 min

准备一个新终端

```
# 打开新的终端，重新登录云服务器
$ ssh -i "tutorial.pem" guest@<server_ip_address>

# 进入开发环境，设置环境变量
$ cd ~/<YOUR_NAME>
$ source env.sh

# 进入 tutorial 目录
$ cd tutorial
# 如没有特殊说明，后续我们都将在这个目录下进行操作
```

使用 Verilator 运行 RTL 仿真

- 使用准备好的 emu，即可运行仿真程序

```
# 运行仿真程序
$ ./emu -i hello.bin --no-diff 2>hello.err

# Some key options:
    -i
    -C / -I / -W
    --diff=PATH / --no-diff
```

这样我们就了解了香山的基本仿真过程。

那么接下来该干什么呢?

我们怎么才能

- **生成所需 workload**
- **查找并修复功能 bug**
- **进行性能分析**
- **进行微架构探索**

敏捷功能验证

Nexus-AM

生成测试

发现错误

difftest

NEMU

Waveform

ChiselDB

定位并
解决问题

保存
出错现场

LightSSS

Nexus-AM: 裸机运行时环境

- **目的**
  - 在没有操作系统的情况下**敏捷地**生成工作负载
  - 为**裸机**（如香山）提供运行时框架
- **Nexus-AM**
  - 轻量且易用
  - 实现了基本的**系统调用**接口和异常处理
  - 支持多种 **ISA 和配置**

Nexus-AM

- **动手试试：**编译 Coremark

```
$ bash build-am.sh
```

```
# cd $AM_HOME/apps/coremark
# make ARCH=riscv64-xs
# ls -l build
#     coremark-riscv64-xs.bin    程序的二进制镜像
#     coremark-riscv64-xs.elf    程序的 ELF 文件
#     coremark-riscv64-xs.txt    程序的反汇编
```

NEMU: ISA 参考设计

- **目的**
  - 可以作为验证的参考模型
  - 简单且高效
- **NEMU**
  - 和 Spike 类似的**指令集模拟器**
  - 经**优化**后，性能与 QEMU 相近
  - 提供了**一系列 API** 来辅助香山比较和验证**微架构**

NEMU

- **动手试试：**编译生成 NEMU

```
$ bash build-nemu.sh
```

```
# cd $NEMU_HOME
# make clean
# make riscv64-xs_defconfig
# make -j      将NEMU编译成裸机，从而可以运行之前步骤的Coremark
# make clean-softfloat
# make riscv64-xs-ref_defconfig
# make         将NEMU编译成香山的参考设计
```

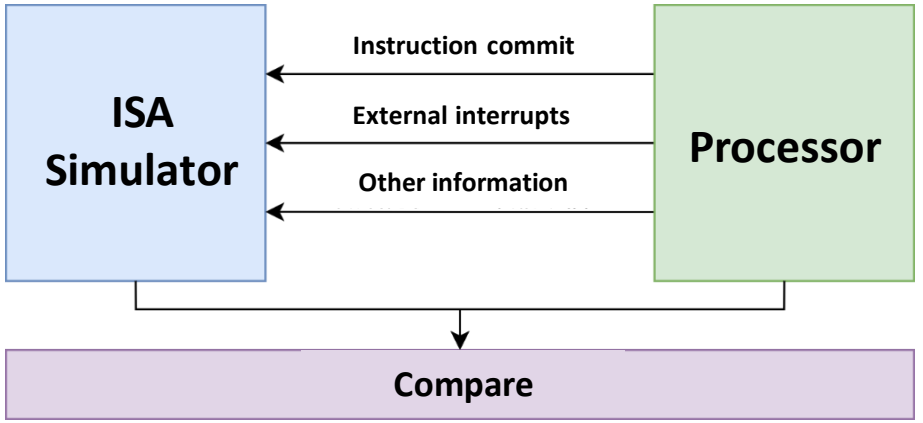
- **动手试试**：在 NEMU 上运行 Coremark

```
$ bash run-nemu.sh
```

```
# cd $NEMU_HOME
# ./build/riscv64-nemu-interpreter -b \                    连续运行，更快
$AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin 运行Coremark
```

Difftest: 差分测试协同仿真框架

- **基本流程**
  - 处理器指令提交/其他信息更新
  - 模拟器执行相同指令
  - 对比待测设计和参考设计间的微架构状态
  - 中止或继续
- **作为处理器验证的基础设施**
  - 向 HDL（如 Chisel/Verilog）提供 API 接口
  - 支持 RTL 模拟器（如 Verilator, VCS）
  - 支持 RISC-V ISS（如 **NEMU**, **Spike**）作为参考设计
- **SMP-Difftest： SMP处理器上的协同仿真**
  - SMP Linux kernel 和多线程程序
  - 在线检查 cache 一致性和内存一致性



Basic architecture of DiffTest

```
while (1) {
  icnt = cpu_step();
  nemu_step(icnt);
  r1s = cpu _getregs();
  r2s = nemu_getregs();
  if (r1s != r2s) { abort(); }
}
```

Online checking

Difftest

- **动手试试**：在香山运行 Coremark，并与 NEMU 进行 difftest

```
# 运行 Coremark 需要约 2 分钟
$ bash run-emu.sh
```

```
# ./emu \
-i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \  使用 coremark.bin
--diff $NEMU_HOME/build/riscv64-nemu-interpreter-so \      与 NEMU difftest
2>perf.out     将标准错误输出重定向到文件
```

Difftest

```
./emu -i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin --diff $NEMU_HOME/build/riscv64-nemu-interpreter-so
Emu compiled at Mar 24 2023, 20:33:03
Using simulated 32768B flash
[warning]no valid flash bin path, use preset flash instead
The image is /home/guest/xs-env-asplos2023/nexus-am/apps/coremark/build/coremark-riscv64-xs.bin
Using simulated 8192MB RAM
NemuProxy using /home/guest/xs-env-asplos2023/NEMU/build/riscv64-nemu-interpreter-so
The first instruction of core 0 has commited. Difftest enabled.
[NEMU] Can not find flash image: (null)
[NEMU] Use built-in image instead
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'flash' at [0x0000000010000000, 0x000000000100fffff]
Running CoreMark for 1 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total time (ms)  : 4288
Iterations       : 1
Compiler version : GCC9.4.0
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xe714
Finised in 4288 ms.
==================================================
CoreMark Iterations/Sec 233
Core 0: HIT GOOD TRAP at pc = 0x80001c52
total guest instructions = 398,372
instrCnt = 398,372, cycleCnt = 517,099, IPC = 0.770398
Seed=0 Guest cycle spent: 517,103 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 77,559ms
```

Difftest

- **动手试试**：用 difftest 触发 bug

```
$ bash run-emu-diff.sh
```

```
# ./emu-bug \
  -i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \
  --diff $NEMU_HOME/build/riscv64-nemu-interpreter-so    # 与 NEMU difftest
```

Difftest

- Difftest 会在比对失败时立即报错
  - 打印 PC、GPRs 等信息

```
============== Commit Group Trace (Core 0) ==============
commit group [00]: pc 0080000cdc cmtcnt 1
commit group [01]: pc 0080000ce0 cmtcnt 1
commit group [02]: pc 0080000ce2 cmtcnt 2
commit group [03]: pc 0080000cea cmtcnt 2
commit group [04]: pc 0080000cf0 cmtcnt 2
commit group [05]: pc 0080000cf4 cmtcnt 1
commit group [06]: pc 0080000cb0 cmtcnt 2
commit group [07]: pc 0080000cb6 cmtcnt 1
commit group [08]: pc 0080000cba cmtcnt 1
commit group [09]: pc 0080000cbc cmtcnt 2
commit group [10]: pc 0080000cc2 cmtcnt 1 ←
commit group [11]: pc 0080000cc4 cmtcnt 2
commit group [12]: pc 0080000ccc cmtcnt 1
commit group [13]: pc 0080000cce cmtcnt 1
commit group [14]: pc 0080000cd2 cmtcnt 2
commit group [15]: pc 0080000cd8 cmtcnt 1
```

```
==============  REF Regs  ==============
  $0: 0x0000000000000000   ra: 0x0000000080001620   sp: 0x000000008000cef0   gp: 0x0000000000000000
  tp: 0x0000000000000000   t0: 0x000000000000001f   t1: 0x0000000080003a08   t2: 0x0000000000000000
  s0: 0x0000000000007fff   s1: 0x0000000000000000   a0: 0x00000000800039e8   a1: 0x0000000000000000
  a2: 0x0000000000000000   a3: 0x0000000080003a18   a4: 0x0000000000000001   a5: 0x0000000000000009
  a6: 0x0000000000000001   a7: 0x0000000080003be4   s2: 0x0000000000000000   s3: 0x0000000000000000
  s4: 0x0000000000000000   s5: 0x0000000000000000   s6: 0x0000000000000000   s7: 0x0000000000000000
  s8: 0x0000000000000000   s9: 0x0000000000000000  s10: 0x0000000000000000  s11: 0x0000000000000000
  t3: 0x0000000080003be4   t4: 0x0000000080003bd8   t5: 0x0000000080003c54   t6: 0x000000000000001f
 ft0: 0xffffffff00000000  ft1: 0xffffffff00000000  ft2: 0xffffffff00000000  ft3: 0xffffffff00000000
 ft4: 0xffffffff00000000  ft5: 0xffffffff00000000  ft6: 0xffffffff00000000  ft7: 0xffffffff00000000
 fs0: 0xffffffff00000000  fs1: 0xffffffff00000000  fa0: 0xffffffff00000000  fa1: 0xffffffff00000000
 fa2: 0xffffffff00000000  fa3: 0xffffffff00000000  fa4: 0xffffffff00000000  fa5: 0xffffffff00000000
 fa6: 0xffffffff00000000  fa7: 0xffffffff00000000  fs2: 0xffffffff00000000  fs3: 0xffffffff00000000
 fs4: 0xffffffff00000000  fs5: 0xffffffff00000000  fs6: 0xffffffff00000000  fs7: 0xffffffff00000000
 fs8: 0xffffffff00000000  fs9: 0xffffffff00000000 fs10: 0xffffffff00000000 fs11: 0xffffffff00000000
 ft8: 0xffffffff00000000  ft9: 0xffffffff00000000 ft10: 0xffffffff00000000 ft11: 0xffffffff00000000
pc: 0x0000000080000cc4 mstatus: 0x8000000a00006000 mcause: 0x0000000000000000 mepc: 0x4ede2b1aa1cb2ef6
                       sstatus: 0x8000000200006000 scause: 0x0000000000000000 sepc: 0x0000000000000000
satp: 0x0000000000000000
mip: 0x0000000000000000 mie: 0x0000000000000000 mscratch: 0x0000000000000000 sscratch: 0x0000000000000000
mideleg: 0x0000000000000000 medeleg: 0x0000000000000000
mtval: 0x0000000000000000 stval: 0x91da2105e8dfee5b mtvec: 0x0000000000000000 stvec: 0x0000000000000000
privilege mode:3  pmp: below
 0: cfg:0x00 addr:0x0000000000000000| 1: cfg:0x00 addr:0x0000000000000000
 2: cfg:0x00 addr:0x0000000000000000| 3: cfg:0x00 addr:0x0000000000000000
 4: cfg:0x00 addr:0x0000000000000000| 5: cfg:0x00 addr:0x0000000000000000
 6: cfg:0x00 addr:0x0000000000000000| 7: cfg:0x00 addr:0x0000000000000000
 8: cfg:0x00 addr:0x0000000000000000| 9: cfg:0x00 addr:0x0000000000000000
10: cfg:0x00 addr:0x0000000000000000|11: cfg:0x00 addr:0x0000000000000000
12: cfg:0x00 addr:0x0000000000000000|13: cfg:0x00 addr:0x0000000000000000
14: cfg:0x00 addr:0x0000000000000000|15: cfg:0x00 addr:0x0000000000000000
priviledgeMode: 3
     a5 different at pc = 0x0080000cc2, right= 0x0000000000000009, wrong = 0x0000000000000000
Core 0: ABORT at pc = 0x80000cd2
total guest instructions = 1639
instrCnt = 1639, cycleCnt = 8726, IPC = 0.187829
Seed=0 Guest cycle spent: 8729 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 1814ms
```

Difftest

- **动手试试：**在 difftest 检测出 bug 后生成波形

```
$ bash run-emu-dumpwave.sh
$ ls ../XiangShan/build | grep "vcd"   # 目录下应该有后缀名为 .vcd 的波形文件
```

```
# ./emu-bug \
-i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \
--diff $NEMU_HOME/build/riscv64-nemu-interpreter-so \
-b 1200 \        生成波形的起始周期。在上一步中，香山在约1600 个周期处报错结束
-e 1700 \        生成波形的结束周期
--dump-wave \    生成波形，之后会在$NOOP_HOME/build 目录下生成*.vcd 文件
```

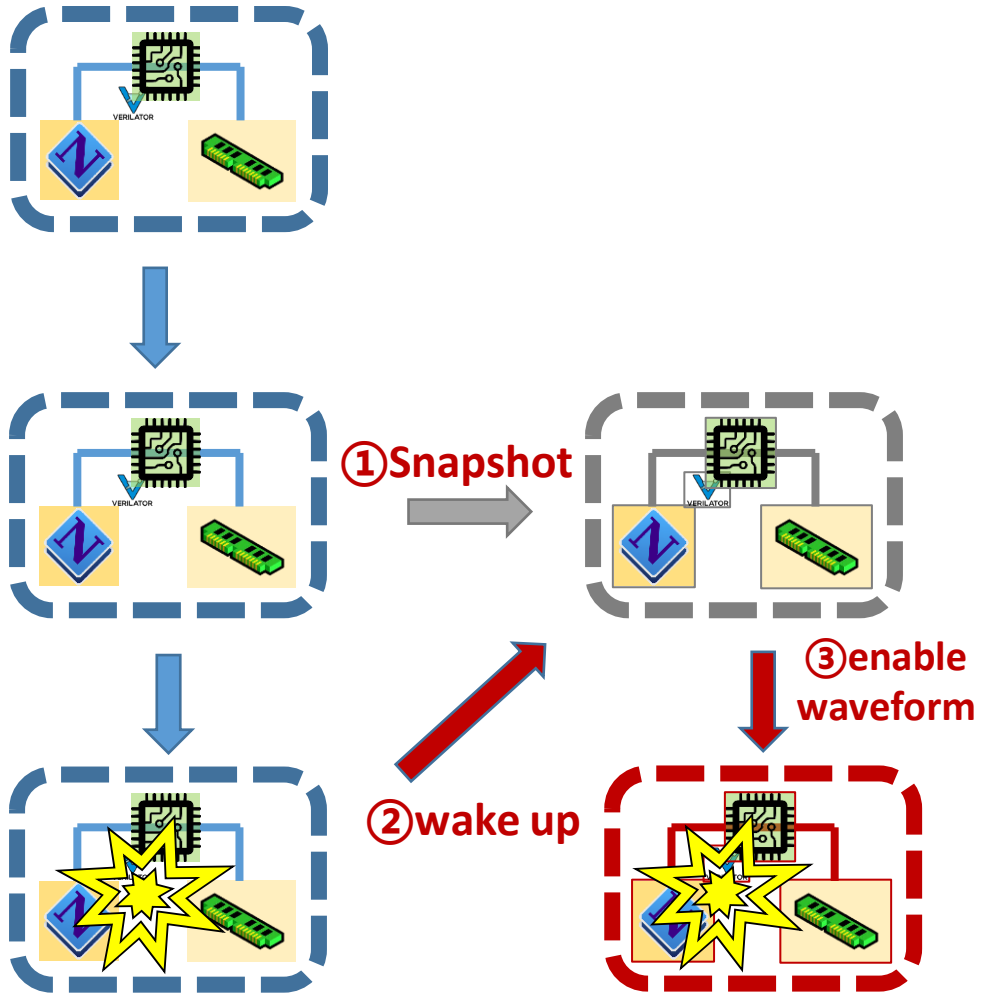
LightSSS: 一种基于内存的轻量级仿真快照

- **重跑仿真很费时间！**
  - **可以通过快照解决**

- **LightSSS：使用 fork() 对进程做快照**
  - Linux Kernel 写时复制机制

- **优点一：可移植性和可扩展性好**
  - 可以对任何外部模块做快照（如 C++）
  - 无需理解外部模块的细节

- **优点二：快照开销低**
  - 生成快照只需约 500 us
  - 远低于 Verilator 提供的 RTL 快照开销



LightSSS

- **动手试试：**用 LightSSS 生成波形，根据需求 debug

```
$ bash run-emu-lightsss.sh
```

```
# ./emu-bug \
-i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \
--diff $NEMU_HOME/build/riscv64-nemu-interpreter-so \
--enable-fork \     不再需要使用复杂的参数
2 > lightsss.err
```

LightSSS

- LightSSS 工作时你应该看到：

```
 4: cfg:0x00 addr:0x0000000000000000| 5: cfg:0x00 addr:0x0000000000000000
 6: cfg:0x00 addr:0x0000000000000000| 7: cfg:0x00 addr:0x0000000000000000
 8: cfg:0x00 addr:0x0000000000000000| 9: cfg:0x00 addr:0x0000000000000000
10: cfg:0x00 addr:0x0000000000000000|11: cfg:0x00 addr:0x0000000000000000
12: cfg:0x00 addr:0x0000000000000000|13: cfg:0x00 addr:0x0000000000000000
14: cfg:0x00 addr:0x0000000000000000|15: cfg:0x00 addr:0x0000000000000000
priviledgeMode: 3
     a5 different at pc = 0x0080000630, right= 0x0000000000000009, wrong = 0x0000000000000000
[FORK_INFO pid(20792)] the oldest checkpoint start to dump wave and dump nemu log...
[FORK_INFO pid(20792)] dump wave to /home/guest/xiangshan/XiangShan/build/2023-08-21@15:23:17_1.vcd...
NEMU dynamic_config update
  - ignore_illegal_mem_access 0
  - debug_difftest 1
The first instruction of core 0 has commited. Difftest enabled.
[NEMU] Can not find flash image: (null)
[NEMU] Use built-in image instead
```

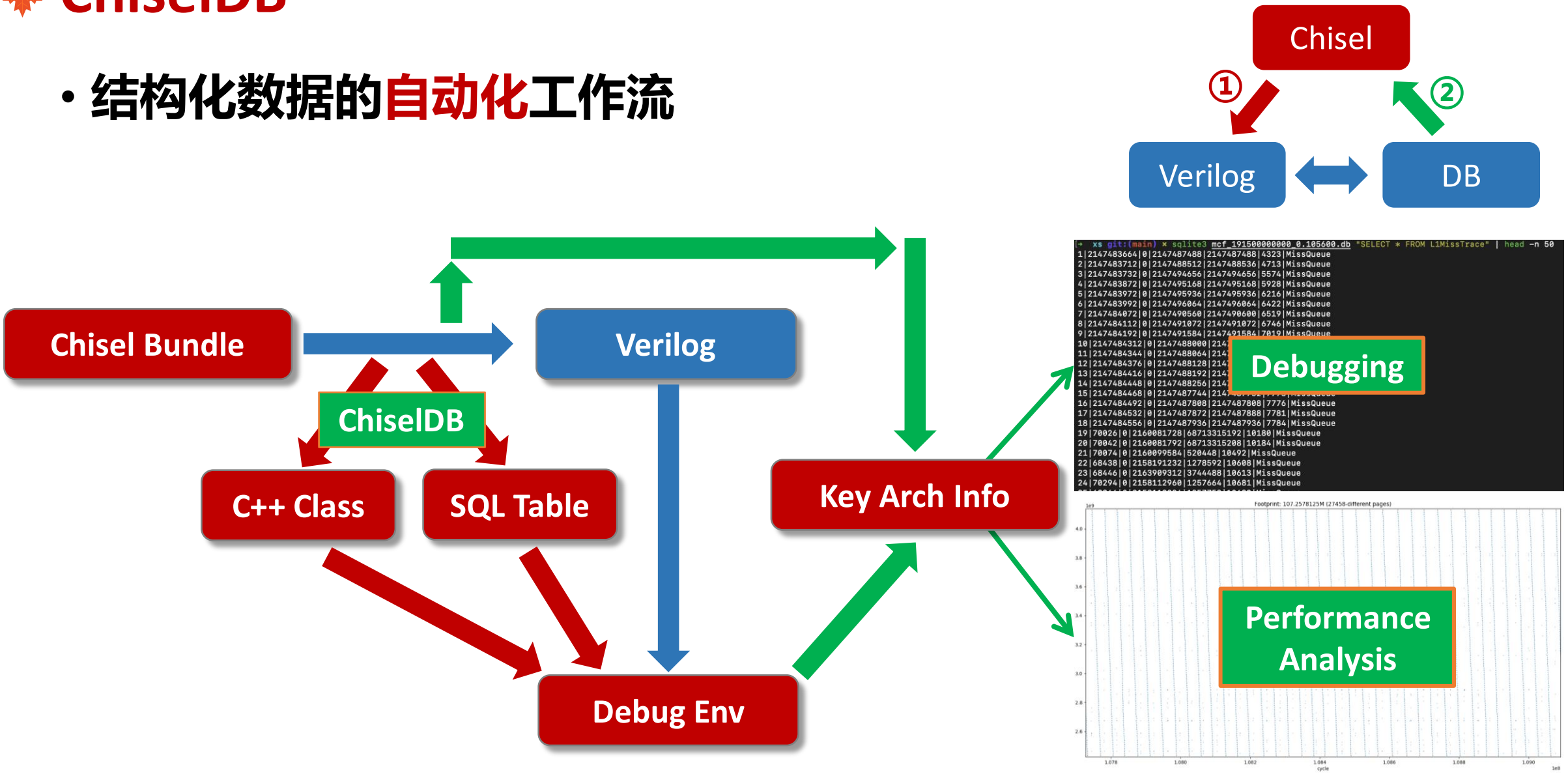
- 此后，仿真会从最近的快照处重启

ChiselDB: Debug 友好的结构化数据库

- **动机**
  - 波形占用空间**大**，且**难以用于**进一步分析
  - 需要分析类似访存事务 trace 的结构化数据
- **ChiselDB**
  - 在硬件层面的模块接口间插入探针
  - DPI-C：在 Chisel 代码中使用 C++ 函数传递数据
  - 在数据库中持久化存储，支持 SQL 查询

ChiselDB

- 结构化数据的**自动化**工作流



ChiselDB

- **源代码：** *XiangShan/utility/ChiselDB.scala*
- **使用方法：** 创建 Table

```
// API: def createTable[T <: Record](tableName: String, hw: T): Table[T]
import huancun.utils.ChiselDB

class MyBundle extends Bundle {
        val fieldA = UInt(10.W)
        val fieldB = UInt(20.W)
}
val table = ChiselDB.createTable("MyTableName", new MyBundle)
```

ChiselDB

- **使用方法：**添加寄存器探针

```
/* APIs
def log(data: T, en: Bool, site: String = "", clock: Clock, reset: Reset)
def log(data: Valid[T], site: String, clock: Clock, reset: Reset): Unit
def log(data: DecoupledIO[T], site: String, clock: Clock, reset: Reset): Unit
*/
table.log(
  data = my_data /* hardware of type T */,
  en = my_cond,   site = "MyCallSite",
  clock = clock,   reset = reset
)
```

ChiselDB

- **实例：** *XiangShan/src/main/scala/xiangshan/cache/mmu/L2TLB.scala*

```
class L2TlbMissQueueDB(implicit p: Parameters) extends TlbBundle {
   val vpn = UInt(vpnLen.W)
}

val L2TlbMissQueueInDB, L2TlbMissQueueOutDB = Wire(new L2TlbMissQueueDB)
L2TlbMissQueueInDB.vpn := missQueue.io.in.bits.vpn
L2TlbMissQueueOutDB.vpn := missQueue.io.out.bits.vpn

val L2TlbMissQueueTable = ChiselDB.createTable(
        "L2TlbMissQueue_hart" + p(XSCoreParamsKey).HartId.toString, new L2TlbMissQueueDB)

L2TlbMissQueueTable.log(L2TlbMissQueueInDB, missQueue.io.in.fire, "L2TlbMissQueueIn", clock, reset)
L2TlbMissQueueTable.log(L2TlbMissQueueOutDB, missQueue.io.out.fire, "L2TlbMissQueueOut", clock, reset)
```

- **动手试试：**分析 Cache 一致性违例

```
# 查看插入的bug – 我们将所有的Release 数据都设置成常量
$ cat cc_err.patch
```

```
--- a/src/main/scala/huancun/noninclusive/SinkC.scala
+++ b/src/main/scala/huancun/noninclusive/SinkC.scala
@@ -93,7 +93,7 @@ class SinkC(implicit p: Parameters) extends
BaseSinkC {
-       buffer(insertIdx)(count) := c.bits.data
+       buffer(insertIdx)(count) := 0xABCDEF.U
```

- **动手试试：**分析 Cache 一致性违例

```
# 编译运行（很费时，因此我们用已经编译好的emu）
# 通常使用`make emu -j4 EMU_THREADS=4 WITH_CHISELDB=1`命令进行编译
$ bash chiseldb_step1_run.sh
```

```
# ./emu-cc-err -i $NOOP_HOME/ready-to-run/linux.bin \
  --diff $NOOP_HOME/ready-to-run/riscv64-nemu-interpreter-so \
  --dump-db 2>linux.err
```

- **动手试试：**分析 Cache 一致性违例

```
# 分析
$ bash chiseldb_step2_analyze.sh
```

```
# 1. sqlite 查询所有在Eaddr(0x80000dc0)上的事务
# 2. 使用脚本解析TLLog
# sqlite3 $NOOP_HOME/build/*.db \
  "select * from TLLOG where ADDRESS=0x80000dc0" | \
  sh $NOOP_HOME/scripts/utils/parseTLLog.sh
```

ChiselDB

```
# 结果: [Time | To_From | Channel | Opcode | Permission | Address | Data(0)]

# 数据成功地从 L1I 传输到了 L2
1364 L2_L1I_0 C ReleaseData Shrink TtoN 80000dc0 689cf9796ec050ef

# 数据成功地从 L2 传输到了 L3
1365 L3_L2_0 C ReleaseData Shrink TtoN 80000dc0 689cf9796ec050ef

# 但是当 L1I 再次请求 Eaddr，从 L3 读取的数据却出错了
1796 L2_L1I_0 A AcquireBlock Grow NtoB 80000dc0
1797 L3_L2_0 A AcquireBlock Grow NtoB  80000dc0
1808 L3_L2_0 D GrantData Cap toT 80000dc0 0000000000abcdef
1809 L2_L1I_0 D GrantData Cap toT 80000dc0 0000000000abcdef

# 因此可以推断 L3 记录 Release Data 时出现了错误
```

敏捷性能验证

Chisel-based prototypes

功能点
代码实现

运行
性能测试

VERILATOR

umd-memsys/
DRAMsim3

DRAMsim3: a Cycle-accurate, Thermal-Capable DRAM Simulator

XSPerf

Constantin

Top-down analysis

性能分析

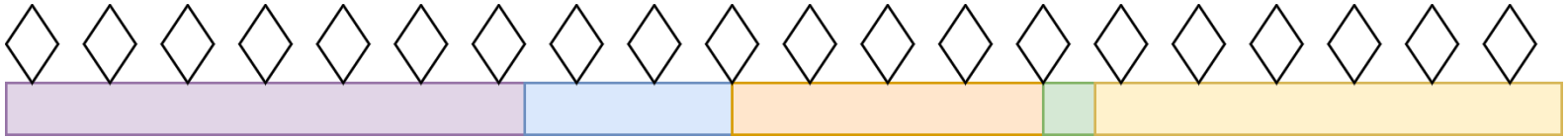
性能评估

RISC-V Checkpoint

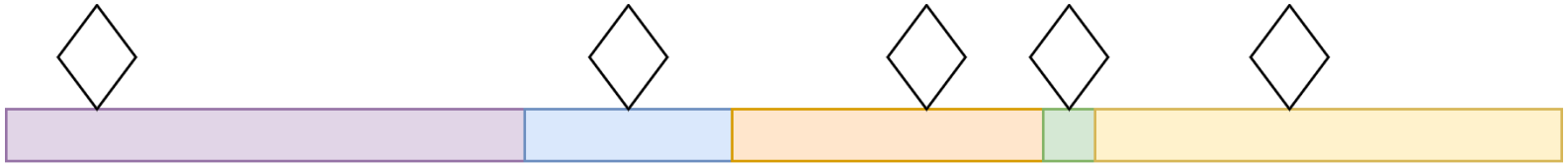
SimPoint

Checkpoint: 提高仿真性能评估并行度

- **Uniform Checkpoint**
  - 将程序等份划分成可并行执行的片段，均一采样



- **Simpoint Checkpoint**
  - 通过聚类分析，选择能够代表程序特性的片段，带权重采样



Checkpoint: Uniform Checkpoint

- step0: 为 checkpoints 准备 NEMU 环境

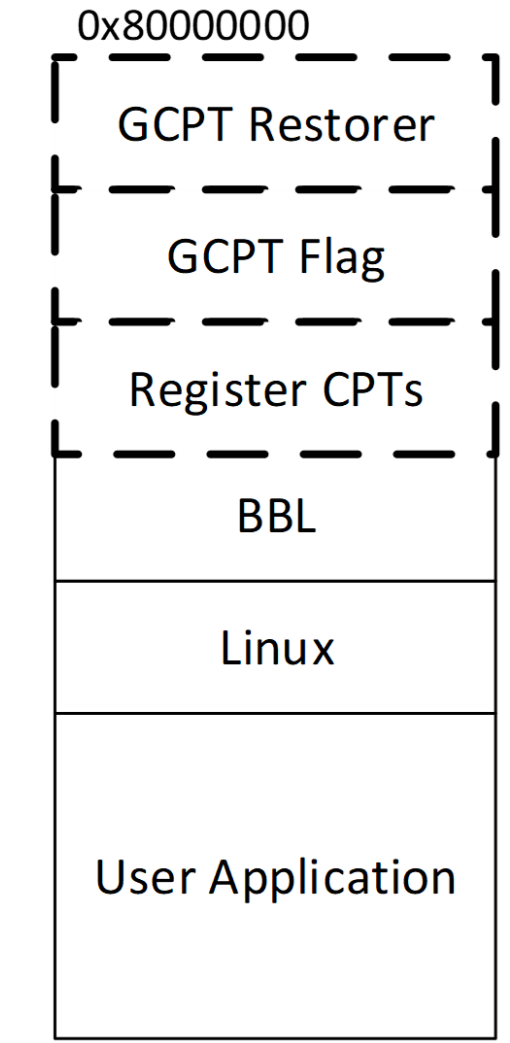
```
$ cd checkpoint      # 进入 checkpoint 目录
$ sh uniform_step0_prepare.sh
```

```
# uniform_step0_prepare.sh

# cd $NEMU_HOME

# git checkout tutorial
# git submodule update --init
# make clean
# make tutorial_defconfig
# make -j4

// 生成 gcpt restorer 的二进制文件
# cd resource/gcpt_restore && make -j4
```



Basic format of RISC-V Checkpoint

Checkpoint: Uniform Checkpoint

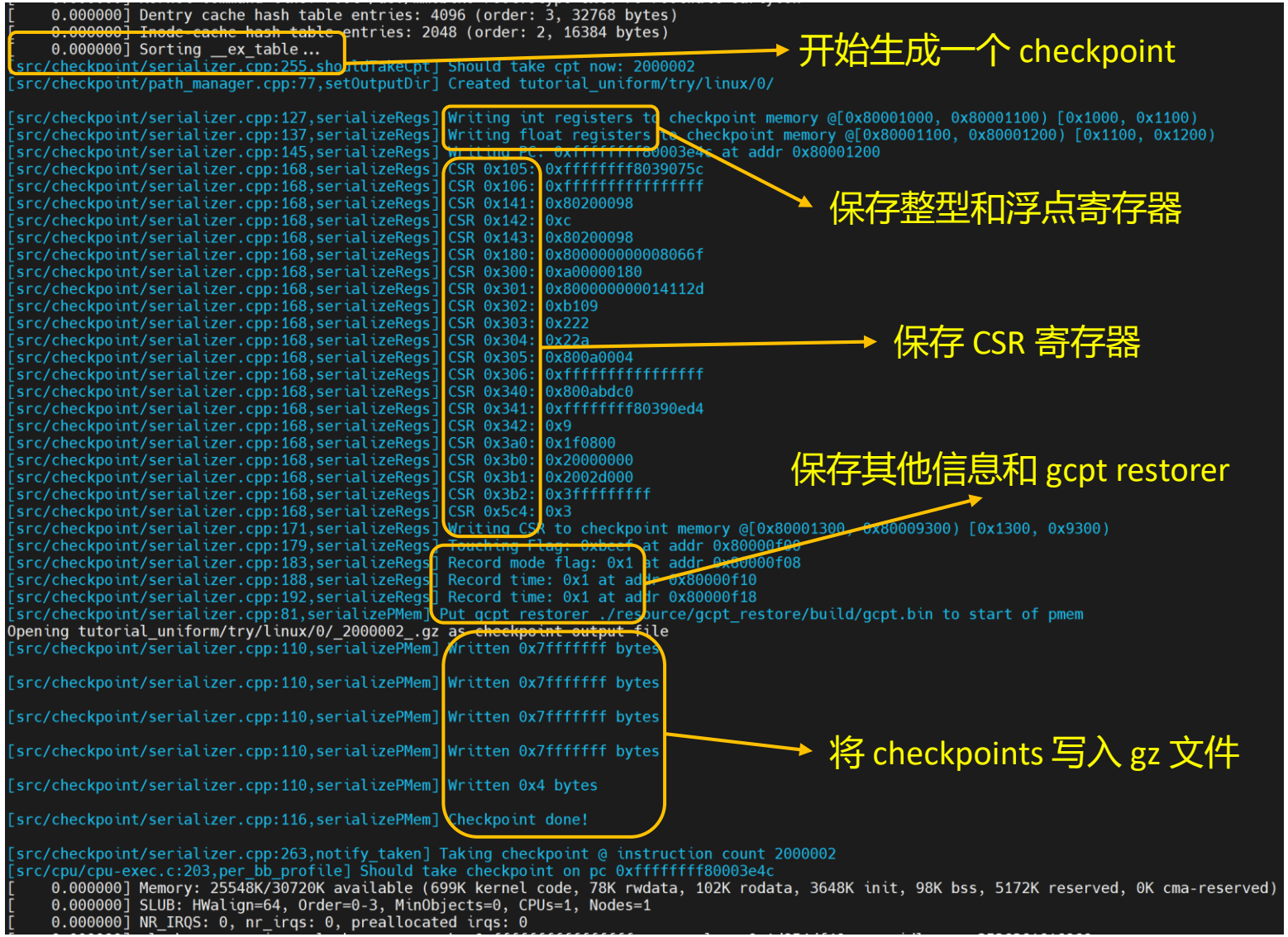
- **step1: 用 NEMU 生成 uniform checkpoints**

```
$ sh uniform_step1_gen.sh
```

```
# uniform_step1_gen.sh

# cd $NEMU_HOME
# rm -rf tutorial_uniform
# ./build/riscv64-nemu-interpreter
#     --cpt-interval 2000000   checkpoint的采样周期
#     -u                       按统一的间隔采样
#     -b                       以批处理模式运行
#     -D tutorial_uniform      指定生成 checkpoints 所在的目录
#     -C try                   指定任务名称
#     -w linux                 指定 workload
#     -r ./resource/gcpt_restore/build/gcpt.bin 上一步生成的 gcpt restorer 所在路径
#     --dont-skip-boot         启动后立即开始采样
#     -I 3000000               执行的最大指令数
#     ./ready-to-run/linux-0xa0000.bin
```

Checkpoint: Uniform Checkpoint



Checkpoint: Uniform Checkpoint

- **step2: 在 NEMU 或 XiangShan 中运行 uniform checkpoint**
- 以 NEMU 为例:

```
$ sh uniform_step2_run_nemu.sh
```

```
# uniform_step2_run_nemu.sh

# cd $NEMU_HOME

# ./build/riscv64-nemu-interpreter
#     -b                    以批处理模式运行
#     --restore             从 CPT FILE 中恢复
#     -I 1000000            执行的最大指令数
#     `find tutorial_uniform/try/linux/0/ -type f -name "*.gz" | tail -1`
                            上一步生成的 checkpoint 的路径
```

Checkpoint: Uniform Checkpoint

```
[src/monitor/monitor.c:103,parse_args] Restoring from checkpoint
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'clint' at [0x0000000038000000, 0x000000003800ffff]
[src/isa/riscv64/init.c:100,init_isa] NEMU will start from pc 0x80000000
[src/monitor/image_loader.c:69,load_img] Loading Gcpt file form cmdline: tutorial_uniform/try/linux/0/_2000002_.gz

[src/monitor/image_loader.c:77,load_img] Loading GZ image tutorial_uniform/try/linux/0/_2000002_.gz
[src/monitor/image_loader.c:69,load_img] Loading Gcpt restorer form cmdline: (null)

[src/monitor/image_loader.c:71,load_img] No image is given. Use the default built-in image/restorer.
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'uartlite' at [0x00000000000003f8, 0x0000000000000404]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'uartlite' at [0x0000000040600000, 0x000000004060000c]
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'rtc' at [0x0000000000000048, 0x000000000000004f]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'rtc' at [0x00000000a1000048, 0x00000000a100004f]
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'screen' at [0x0000000000000100, 0x0000000000000107]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'screen' at [0x0000000040001000, 0x0000000040001007]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'vmem' at [0x0000000050000000, 0x00000000500752ff]
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'keyboard' at [0x0000000000000060, 0x0000000000000063]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'keyboard' at [0x00000000a1000060, 0x00000000a1000063]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'sdhci' at [0x0000000040002000, 0x000000004000207f]
[src/device/sdcard.c:137,init_sdcard] Can not find sdcard image:
[src/monitor/monitor.c:44,welcome] Debug: OFF
[src/monitor/monitor.c:49,welcome] Build time: 19:35:10, Mar 25 2023
Welcome to riscv64-NEMU!
For help, type "help"
[    0.000000] Memory: 25548K/30720K available (699K kernel code, 78K rwdata, 102K rodata, 3648K init, 98K bss, 5172K reserved, 0K cma-reserved)
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[    0.000000] NR_IRQS: 0, nr_irqs: 0, preallocated irqs: 0
[    0.000000] clocksource: riscv_clocksource: mask: 0xffffffffffffffff max_cycles: 0x1d854df40, max_idle_ns: 3526361616960 ns
[    0.000000] console [hvc0] enabled
[    0.000000] console [hvc0] enabled
[    0.000000] bootconsole [early0] disabled
[    0.000000] bootconsole [early0] disabled
[    0.000000] Calibrating delay loop (skipped), value calculated using timer frequency.. 2.00 BogoMIPS (lpj=10000)
[    0.000000] pid_max: default: 4096 minimum: 301
[    0.000000] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
[    0.000000] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)
[    0.010000] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[    0.020000] clocksource: Switched to clocksource riscv_clocksource
```

从检查点恢复并加载 gz 文件

NEMU 将从检查点的位置开始执行

Checkpoint: Simpoint Checkpoint

- **step0: 为 checkpoints 准备 NEMU 环境**

```
$ sh simpoint_step0_prepare.sh
```

```
# simpoint_step0_prepare.sh

# cd $NEMU_HOME

# git checkout cpt-bk
# git submodule update --init
# cd resource/simpoint/simpoint_repo/analysiscode && make simpoint -j4
                                                    生成 simpoint generator 的二进制文件
# cd resource/gcpt_restore && make -j4   生成 gcpt restorer 的二进制文件

# cd $NEMU_HOME
# make clean
# make ISA=riscv64 XIANGSHAN=1 -j4       编译 NEMU
```

Checkpoint: Simpoint Checkpoint

- **step1: 执行一轮 workload，收集程序行为信息**

```
$ sh simpoint_step1_profiling.sh
```

```
# simpoint_step1_profiling.sh

# cd $NEMU_HOME
# rm -rf tutorial_simpoint

# ./build/riscv64-nemu-interpreter
#     -b                                    以批处理模式运行
#     -D $NEMU_HOME/tutorial_simpoint       指定生成 checkpoints 所在的目录
#     -C profiling                          指定任务名称
#     -w stream                             指定 workload
#     --simpoint-profile                    做 simpoint profiling
#     --interval 50000000                   指定 simpoint 的指令间隔
#     $XS_PROJECT_ROOT/tutorial/bin/stream-0xa0000.bin
```

Checkpoint: Simpoint Checkpoint

```
[src/isa/riscv64/exec/special.c,38,exec_nemu_trap] Start profiling, resetting inst count

-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 4 bytes per array element.
-------------------------------------------------------------
Array size = 2000000 (elements), Offset = 0 (elements)
Memory per array = 7.6 MiB (= 0.0 GiB).
Total memory required = 22.9 MiB (= 0.0 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Your clock granularity/precision appears to be 2047 microseconds.
Each test below will take on the order of 102400 microseconds.
   (= 50 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            710.2     0.023438     0.022528     0.024576
Scale:           181.7     0.089884     0.088064     0.092160
Add:             217.0     0.112868     0.110592     0.114688
Triad:           172.3     0.140402     0.139264     0.141312
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-06 on all three arrays
-------------------------------------------------------------
[src/monitor/cpu-exec.c,110,cpu_exec] nemu: HIT GOOD TRAP at pc = 0x0000002aaaaaa5f6
```

NEMU 会在执行 STREAM 程序的过程中进行分析

Checkpoint: Simpoint Checkpoint

- **step2: 进行聚类，得到权重最高的多个程序片段**

```
$ sh simpoint_step2_cluster.sh
```

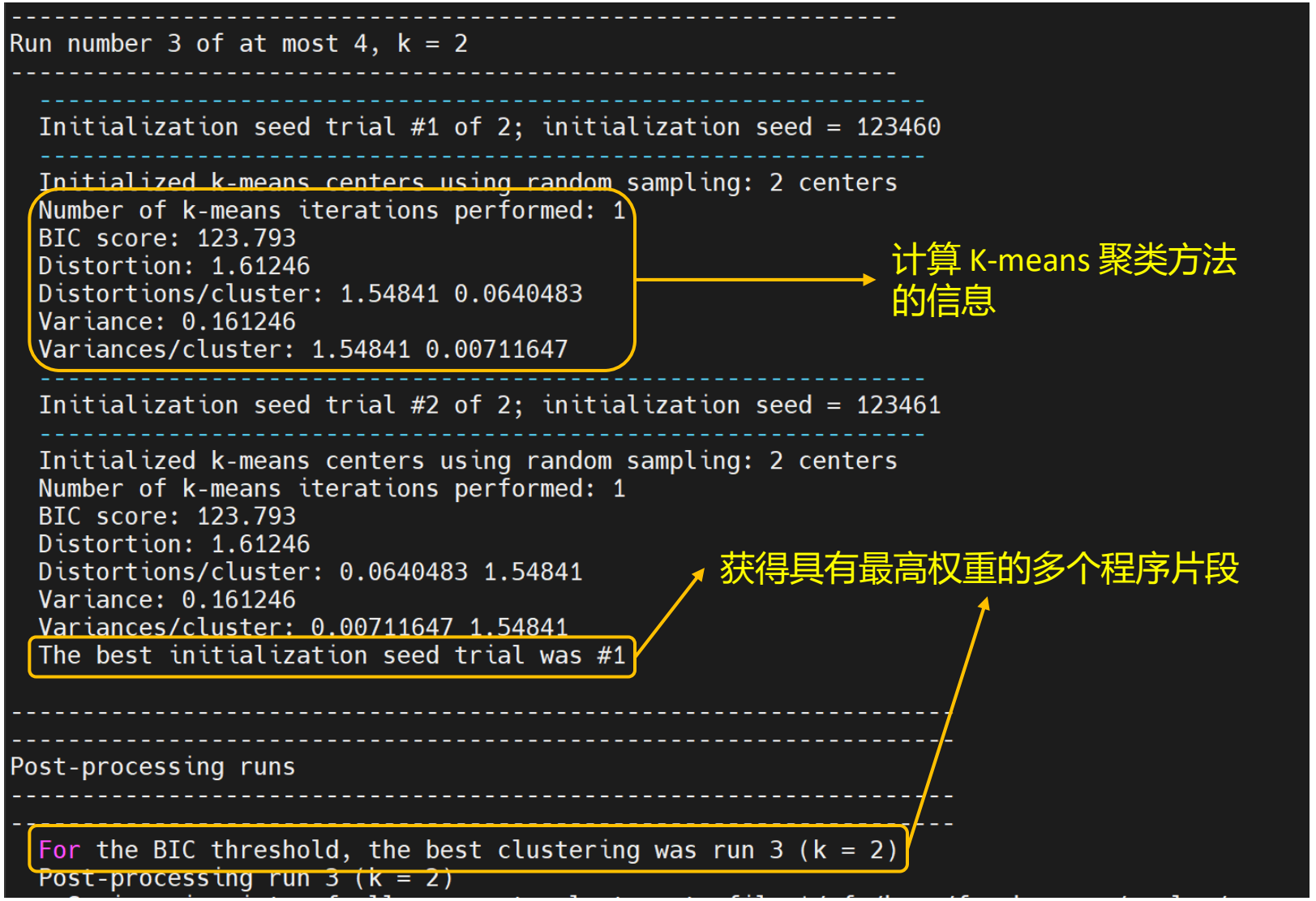
```
# simpoint_step2_cluster.sh

# cd $NEMU_HOME

# mkdir -p $NEMU_HOME/tutorial_simpoint/cluster/stream
# export CLUSTER=$NEMU_HOME/tutorial_simpoint/cluster/stream

# ./resource/simpoint/simpoint_repo/bin/simpoint
#     -loadFVFile ./tutorial_simpoint/profiling/simpoint_bbv.gz  load a frequency vector file
#     -saveSimpoints $CLUSTER/simpoints0                         file to save simpoints
#     -saveSimpointWeights $CLUSTER/weights0                     file to save simpoint weights
#     -inputVectorsGzipped                                       input vectors have been compressed with gzip
#     -maxK 3                 maximum number of clusters to use
#     -numInitSeeds 2         times of different random initialization for each run, taking only the best clustering
#     -iters 1000             maximum number of iterations that should perform
#     -seedkm 123456          random seed for choosing initial k-means centers
#     -seedproj 654321        random seed for random linear projection
```

Checkpoint: Simpoint Checkpoint



Checkpoint: Simpoint Checkpoint

- **step3: 根据聚类结果，生成对应的 Checkpoints**

```
$ sh simpoint_step3_take_cpt.sh          # 大约需要 3 分钟
```

```
# simpoint_step3_take_cpt.sh

# cd $NEMU_HOME
# rm -rf $NEMU_HOME/tutorial_simpoint/stream/take_cpt

# ./build/riscv64-nemu-interpreter
#     -b                                  以批处理模式运行
#     -D tutorial_simpoint                指定生成 checkpoints 所在的目录
#     -C take_cpt                         指定任务名称
#     -w stream                           指定 workload
#     -S ./tutorial_simpoint/cluster      simpoints 聚类结果所在目录
#     --checkpoint-interval 50000000      指定 simpoint 的指令间隔
#     $XS_PROJECT_ROOT/tutorial/bin/stream-0xa0000.bin
```

Checkpoint: Simpoint Checkpoint

Checkpoint: Simpoint Checkpoint

- **step4: 在 NEMU 或 XiangShan 中运行 simpoint checkpoint**
- 以 XiangShan 为例：

```
$ sh simpoint_step4_run_xs.sh
```

```
# simpoint_step4_run_xs.sh

$XS_PROJECT_ROOT/tutorial/emu
    -i `find $NEMU_HOME/tutorial_simpoint/ -type f -name "*_.gz" | tail -1`
    --diff $NOOP_HOME/ready-to-run/riscv64-nemu-interpreter-so
    --max-cycles=100000
    2>simpoint.err
```

XSPerf: 仿真性能计数器

- **两种类型**
  - **Accumulation**：基础的累加型计数器
  - **Histogram**：统计数值分布

Accumulation:

[PERF][time=10000] ctrlBlock.rob: commitInstr, 1500

[PERF][time=10000] ctrlBlock.rob: waitLoadCycle, 800

Histogram:

[PERF][time=10000] l2cache: acquire_period_10_20, 15

[PERF][time=10000] l2cache: acquire_period_20_30, 200

[PERF][time=10000] l2cache: acquire_period_30_40, 60

仿真性能计数器

- **使用方法**
  - 在想要加计数器的位置添加代码 `XSPerfAccumulate('name', signal)` 或 `XSPerfHistogram('name', signal)`
  - 默认会在仿真结束后打印输出
  - 设置 `DebugOptions(EnablePerfDebug = false)` 可以关闭计数器

```
# 回到 tutorial 目录
$ cd ..
$ bash run-emu-perf.sh
```

```
# ./emu -i hello.bin \
  --diff $NEMU_HOME/build/riscv64-nemu-interpreter-so 2>perf.err
```

```
$ cat perf.err
```

仿真性能计数器

```
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.ctrlBlock.decode: utilization,                1183
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.bpu.predictors.components_3.tables_0.wrbypass: wrbypass_hit,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.icache.missUnit.entries_1: entryPenalty1,                  714
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.ctrlBlock.decode: waitInstr,                414
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.bpu.predictors.components_3.tables_0.wrbypass: wrbypass_miss,                     0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.ctrlBlock.decode: stall_cycle,                 393
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.icache.missUnit.entries_1: entryReq1,                   14
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.bpu.predictors.components_3.tables_1.wrbypass: wrbypass_hit,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend: FrontendBubble,               4085
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.icache.missUnit.entries_0: entryPenalty0,                  248
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.bpu.predictors.components_3.tables_1.wrbypass: wrbypass_miss,                     2
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: utilization,                1826
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.icache.missUnit.entries_0: entryReq0,                    6
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_0_1,               2029
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_1_2,                541
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_fire,                    8
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_2_3,                 45
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_stall,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_3_4,                107
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.exuBlocks.scheduler.rs_4.staRS_0.oldestSelection: oldest_override_0,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_PutFullData_fire,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_4_5,                 44
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.exuBlocks.scheduler.rs_4.staRS_0.oldestSelection: oldest_same_as_selected_0,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_PutFullData_stall,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_5_6,                 85
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.exuBlocks.scheduler.rs_4.staRS_0.oldestSelection: oldest_override_1,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_PutPartialData_fire,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_6_7,                 13
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.exuBlocks.scheduler.rs_4.staRS_0.oldestSelection: oldest_same_as_selected_1,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_PutPartialData_stall,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_7_8,                 27
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.memBlock.dtlb_st_tlb_st.entries: port0_np_sp_multi_hit,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_ArithmeticData_fire,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: full,                862
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.memBlock.dtlb_st_tlb_st.entries: port1_np_sp_multi_hit,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_ArithmeticData_stall,                    0
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: exHalf,                125
```

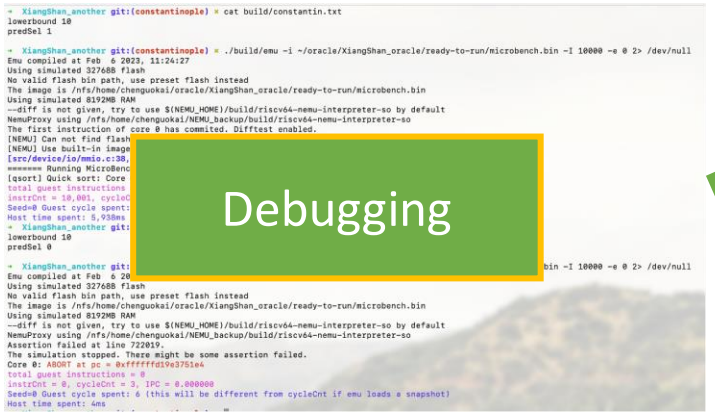
更多解析脚本：https://github.com/OpenXiangShan/env-scripts/blob/main/perf/perf.py

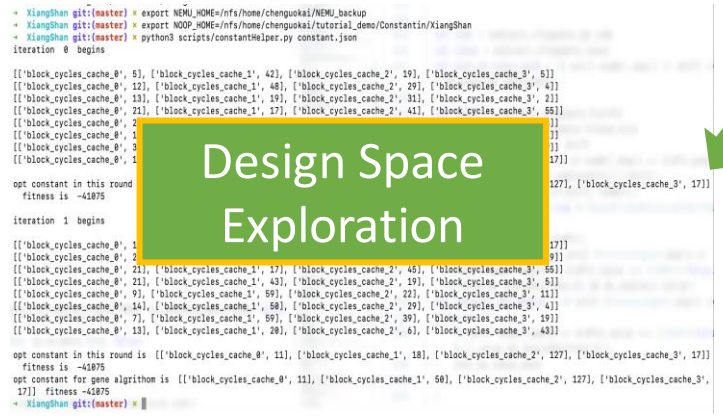
Constantin: 运行时参数调控框架

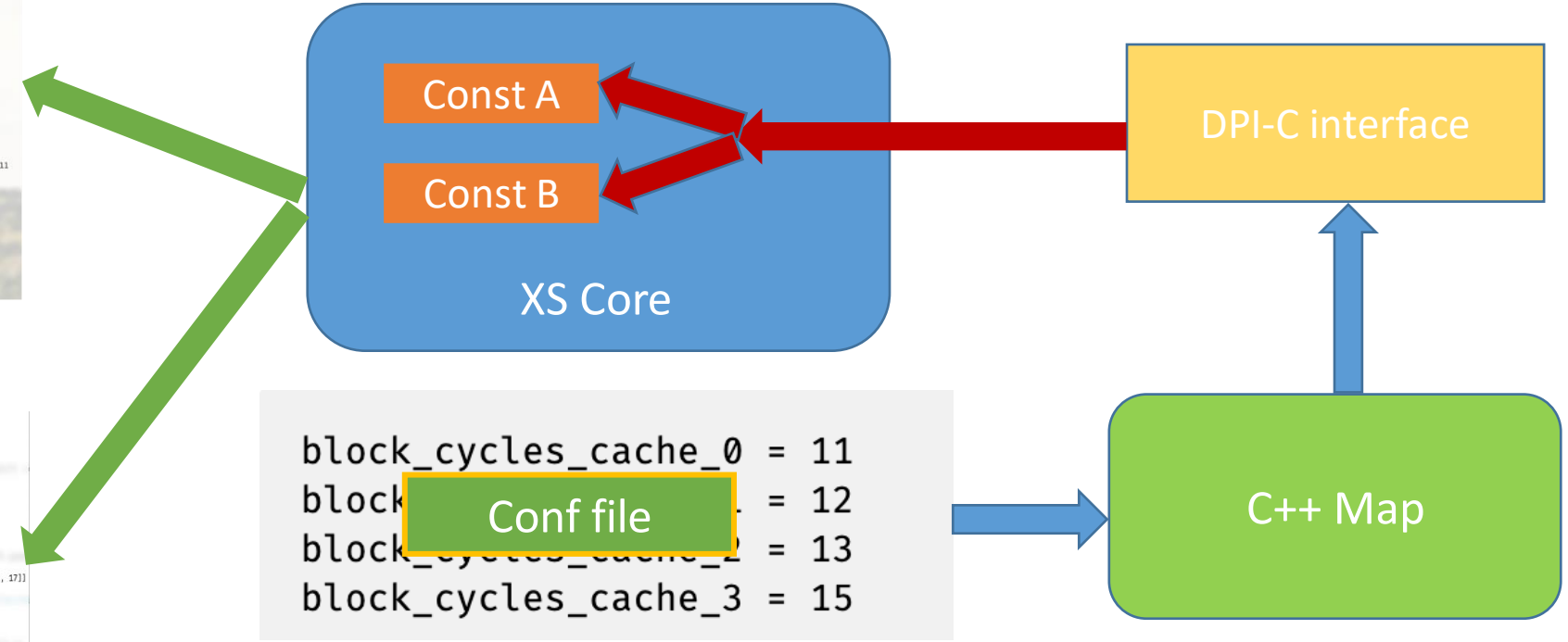
- **动机**
  - 在不同参数配置下测试、对比性能
  - 发现 Bug 后重新编译很耗时
  - 在运行时改变常量参数值
- **设计方法**
  - 在 Chisel 代码中，利用 DPI-C 接口调用 C++ 函数
  - 可以在仿真运行时而不是编译时配置信号值

Constantin

- 常量值仅在运行时保持恒定







Constantin

- **使用方法：**创建信号

```
// API: def createRecord(constName: String): UInt
import utility.Constantin

val block_cycles_cache = RegInit(VecInit(Seq(11.U(ReSelectLen.W),
18.U(ReSelectLen.W), 127.U(ReSelectLen.W), 17.U(ReSelectLen.W))))
block_cycles_cache(0) := Constantin.createRecord("block_cycles_cache_0")
block_cycles_cache(1) := Constantin.createRecord("block_cycles_cache_1")
block_cycles_cache(2) := Constantin.createRecord("block_cycles_cache_2")
block_cycles_cache(3) := Constantin.createRecord("block_cycles_cache_3")
```

Constantin

- **使用方法：**配置文件

```
// 格式：信号名 取值
// 默认路径: ${NOOP_HOME}/build/constantin.txt

block_cycles_cache_0 11
block_cycles_cache_1 18
block_cycles_cache_2 127
block_cycles_cache_3 17
```

Constantin

- **动手试试：** 自动求解最优的参数值

```
# Patch: 打开 AutoSolving
$ cat utility.patch
```

```
- def AutoSolving = false
+ def AutoSolving = true
```

```
# Patch: 创建目标信号
$ cat xs.patch
```

```
+ block_cycles_cache(0) := Constantin.createRecord("block_cycles_cache_0")
+ block_cycles_cache(1) := Constantin.createRecord("block_cycles_cache_1")
+ block_cycles_cache(2) := Constantin.createRecord("block_cycles_cache_2")
+ block_cycles_cache(3) := Constantin.createRecord("block_cycles_cache_3")
```

Constantin

- **动手试试：**自动求解最优的参数值

```
# 由于编译很费时，我们只在这里展示命令

# 应用前面的两个 patches
# cd $XS_PROJECT_ROOT/XiangShan && git apply ../tutorial/xs.patch
# cd utility && git apply ../../tutorial/utility.patch

# 编译
# make emu CONFIG=TutorialConfig EMU_THREADS=1 \
  EMU_TRACE=1 WITH_CONSTANTIN=1 -j8 # Enables support in Difftest framework
```

Constantin

- **动手试试：**自动求解最优的参数值

```
# 手动设置常量的赋值
$ sh constantin-basic.sh
```

```
# cat constantin.txt | ${XS_PROJECT_ROOT}/tutorial/emu-constantin \
  -i ./ready-to-run/linux.bin \
  --diff ./ready-to-run/riscv64-nemu-interpreter-so 2>constantin.err
# 从配置文件中读取常数值到emu
4
block_cycles_cache_0 11
block_cycles_cache_1 12
block_cycles_cache_2 13
block_cycles_cache_3 15
```

Constantin

- **动手试试：**自动求解最优的参数值

```
# 自动参数求解的配置文件
$ cat constantin.json
```

- **参数包含**
  - 信号的性质
  - 自动求解目标
  - 遗传算法的参数
  - XS 模拟器的参数

Constantin

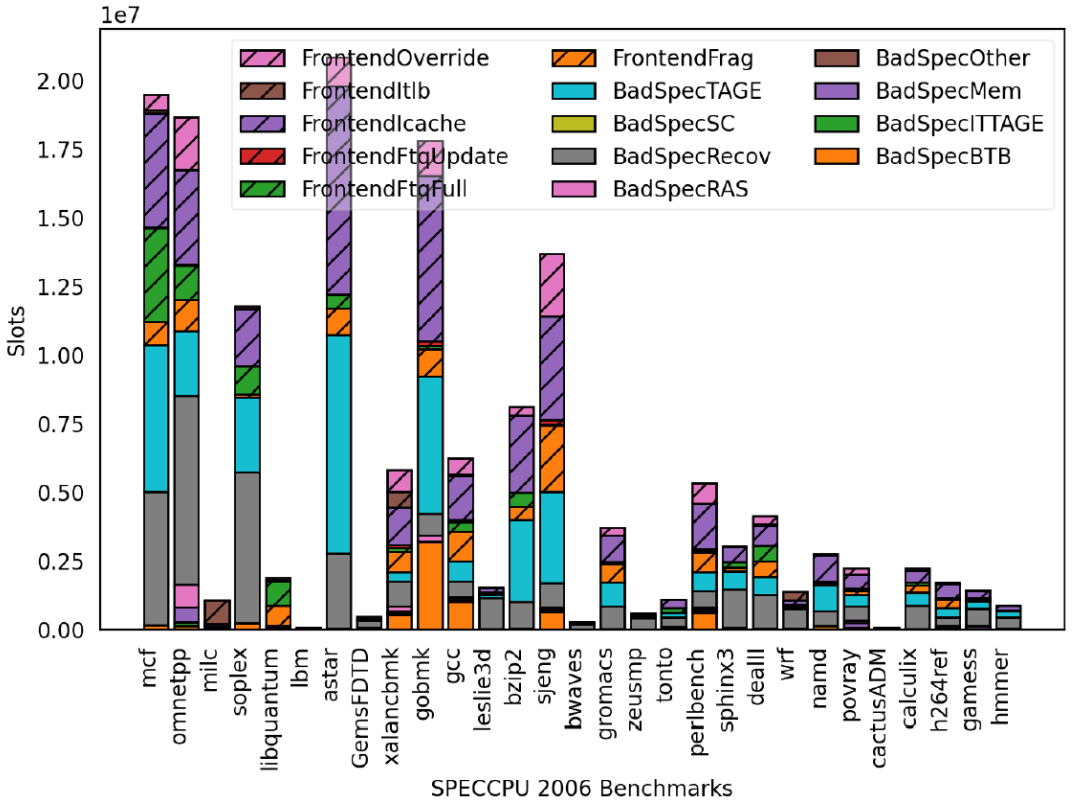
- **动手试试：**自动求解最优的参数值

```
# 运行自动求解器，输出当前得到的最优解
$ sh constantin-solve.sh
```

```
# 生成的最优参数结果
opt constant for gene algrithom is
[['block_cycles_cache_0', XXX], ['block_cycles_cache_1', XXX],
 ['block_cycles_cache_2', XXX],  ['block_cycles_cache_3', XXX]]  fitness 0
```

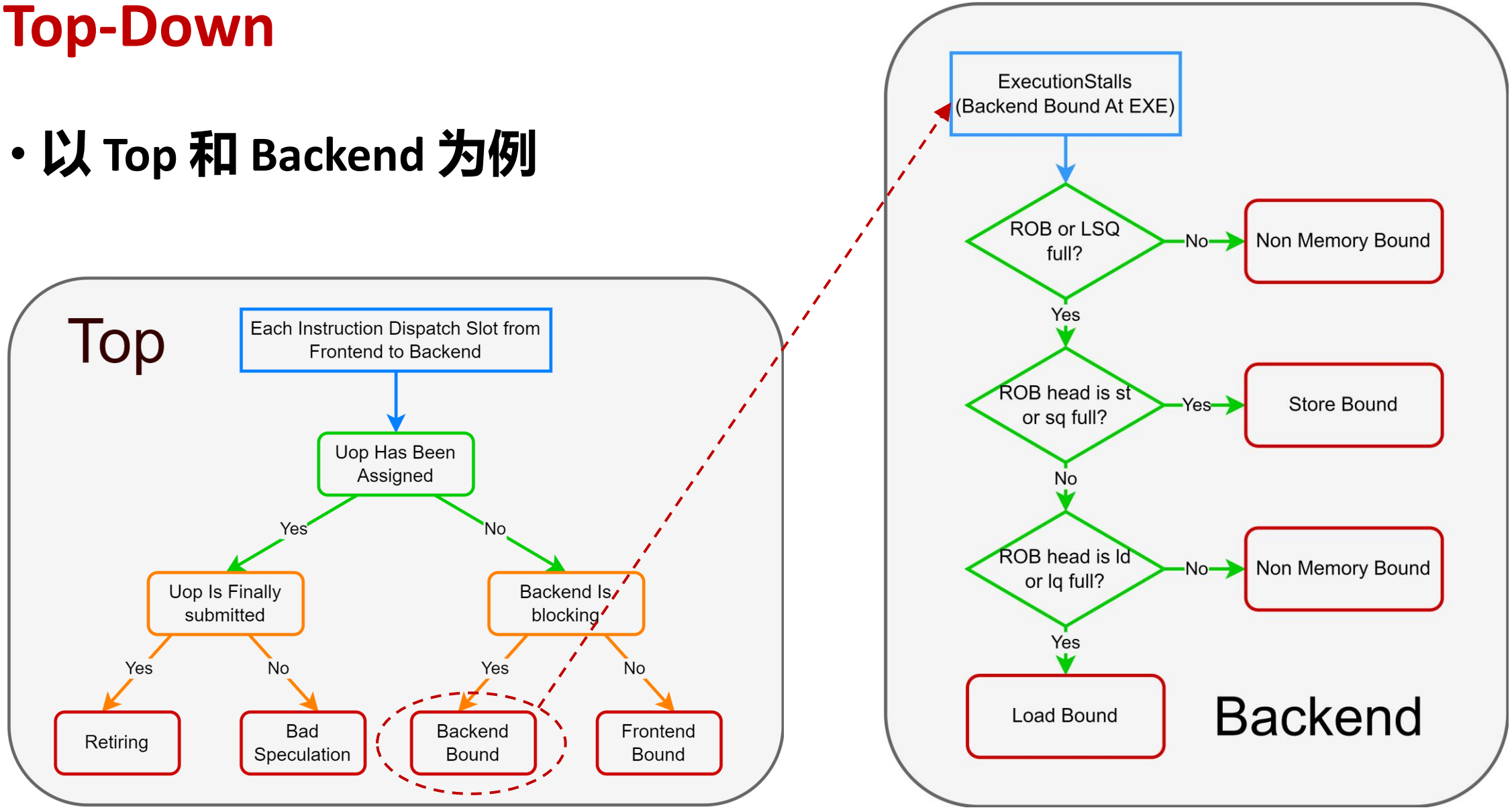
Top-Down: 性能分析模型

- **目的**
  - 将分散的性能事件**有层次地**组织起来
  - **精确**计算性能事件对处理器性能的影响
- **Top-Down on XiangShan**
  - 对 RISC-V 指令集进行**针对性优化适配**
  - 针对香山的微架构进行**性能计数器**优化
  - 进一步完善 Top-Down 模块的**层次化设计**
  - 不会造成性能事件的重复或遗漏
  - 不需要提前假设性能事件的阻塞周期



Top-Down

- 以 Top 和 Backend 为例



Top-Down

- 在 RTL 代码中设置性能计数器

Dispatch.scala

```
val stallReason = Wire(chiselTypeOf(io.stallReason.reason))
// ...
TopDownCounters.values.foreach(ctr =>
  XSPerfAccumulate(ctr.toString(), PopCount(stallReason.map(_ === ctr.id.U)))
)
```

- 运行仿真并收集性能计数器数据

Top-Down

- 用 Top-Down 方法进行层次分析

```
env-scripts: perf/top_down/configs.py


xs_coarse_rename_map = {
  'OverrideBubble': 'MergeFrontend',
  'FtqFullStall': 'MergeFrontend',
  ‘IntDqStall': 'MergeCoreDQStall',  # 将性能计数器归因到 Top-Down 的各个层次之中
  'FpDqStall': 'MergeCoreDQStall'
}
```

- 获取分析结果并进行有针对性的优化

Top-Down

- **动手试试：使用 Top-Down 工具分析阻塞原因**

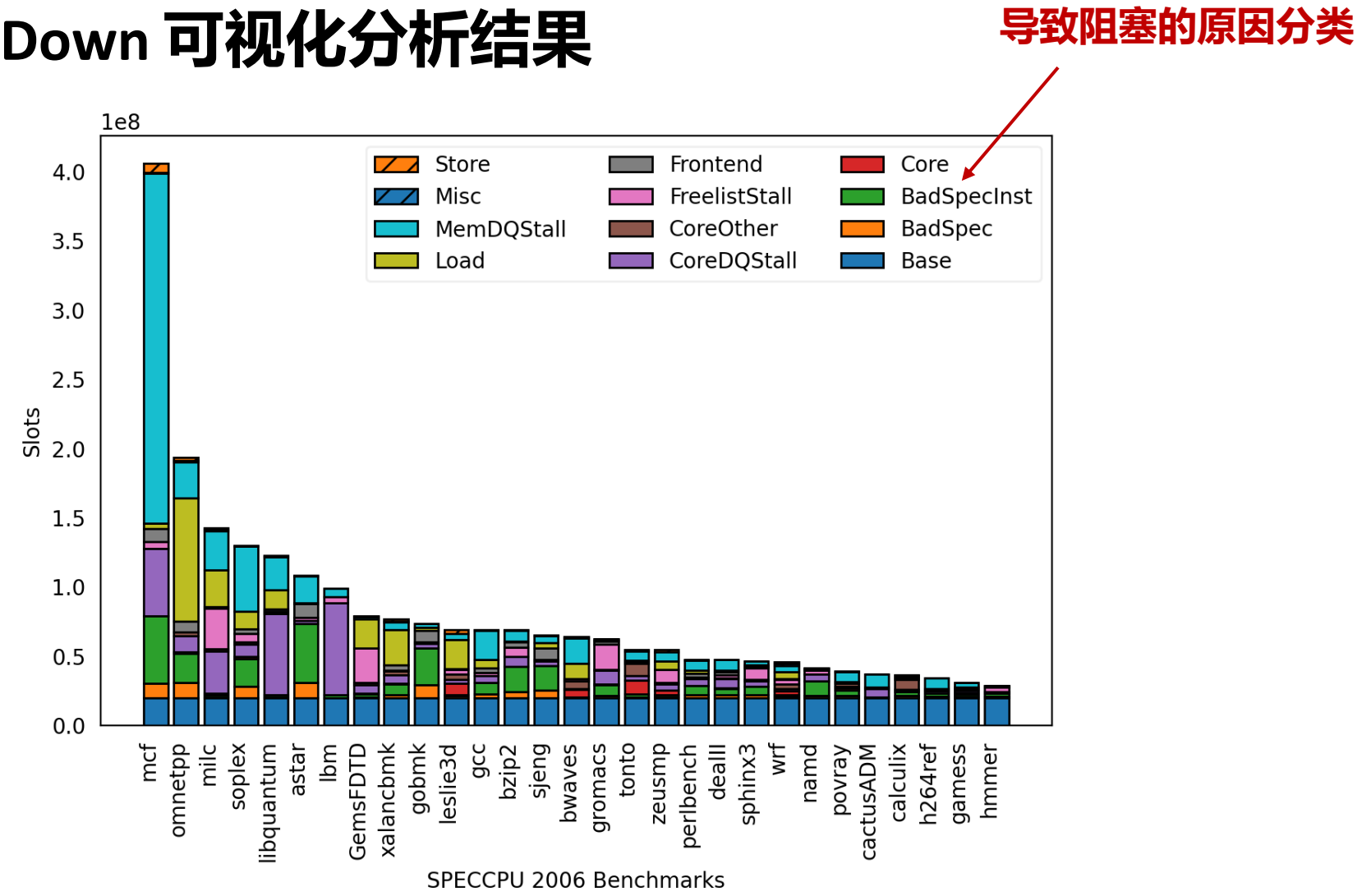
```
# 运行 Top-Down 分析工具
$ bash topdown.sh
```

```
# cd ${XS_PROJECT_ROOT}/env-scripts/perf/top_down && \
  python3 top_down.py \
  -s /opt/SPEC06_EmuTasks_topdown # 指定性能计数器路径

# ls ${XS_PROJECT_ROOT}/env-scripts/perf/top_down/results
result.png  results.csv  results-weighted.csv # 输出分析数据和可视化分析结果
```

Top-Down

- 示例：Top-Down 可视化分析结果



运行编译结果

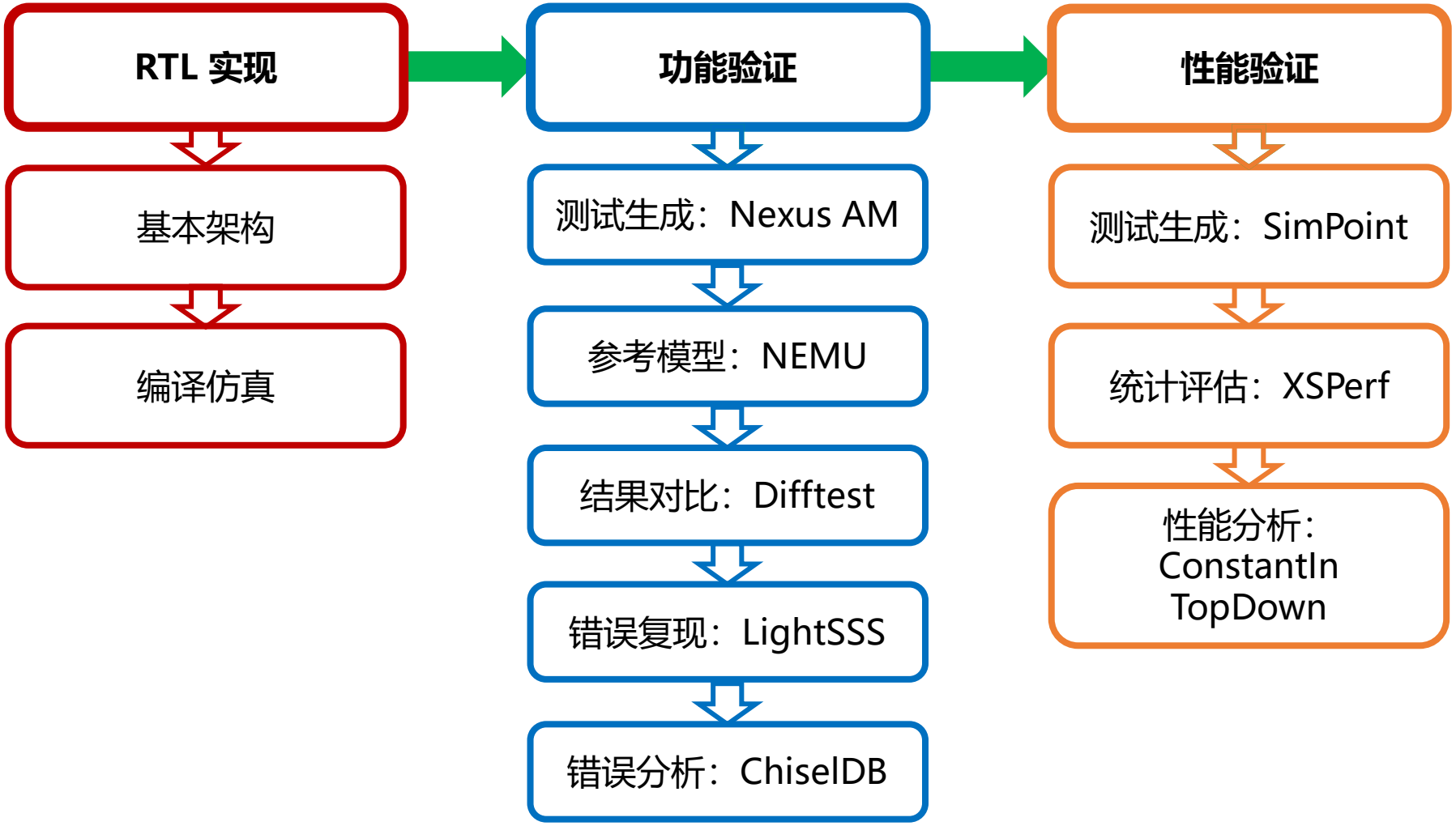
- 回到最开始编译 emu 的 shell 界面
- 可以运行刚刚编译完成的 emu 仿真

```
$ cd $NOOP_HOME
$ ./build/emu --help          (automatically runs with EMU_THREADS threads)
# Some Options:
    -i                        Workload to run
    -C / -I / -W              Max cycles /Max Insts / Warmup Insts
    --diff=PATH / --no-diff   Compare with NEMU for difftest / disable difftest
```

Example： *(use tab for command completion)*

```
$ ./build/emu -i ../tutorial/hello.bin --diff ./ready-to-run/riscv64-nemu-
interpreter-so 2> perf.err
```

小结 – 香山敏捷开发流程与工具



欢迎加入我们！