欢迎大家关注香山并加群讨论

香山技术讨论 QQ 群

香山 tutorial 文档主页

香山技术讨论微信四群

北京开源芯片研究院
BEIJING INSTITUTE OF OPEN SOURCE CHIP

香山处理器 Tutorial

冯浩原[1] 林志达[1] 马月骁[1] 刘泽昊[1] 陈泱宇[2] 李燕琴[1]

[1]中国科学院计算技术研究所 [2]重庆大学

2024 年 8 月 22 日

目录

一、香山处理器概述

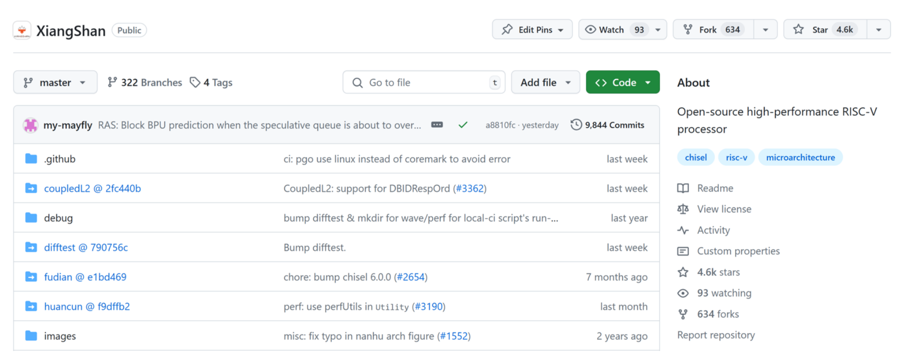
二、香山处理器的功能验证方法及演示

三、香山处理器的性能评估方法及演示

香山：开源高性能处理器

- **第一代架构：雁栖湖**
  - 2020/6：RTL 设计的第一个提交
  - 2021/7：**28nm 流片，1.3GHz**
  - 性能：**SPEC CPU2006 7.01分@1GHz, DDR4-1600**
- **第二代架构：南湖**
  - 2021/5：开始设计探索和架构设计
  - 2023/4：GDSII 交付
  - **14nm 流片，SPEC CPU2006 ～20分@2GHz**
- **第三代架构：昆明湖**
  - 新的指令集扩展（虚拟化、向量等）
  - 优化设计的流水线部件
  - CHI-CoupledL2 缓存
  - 性能：**SPEC CPU2006 ～45分@3GHz**

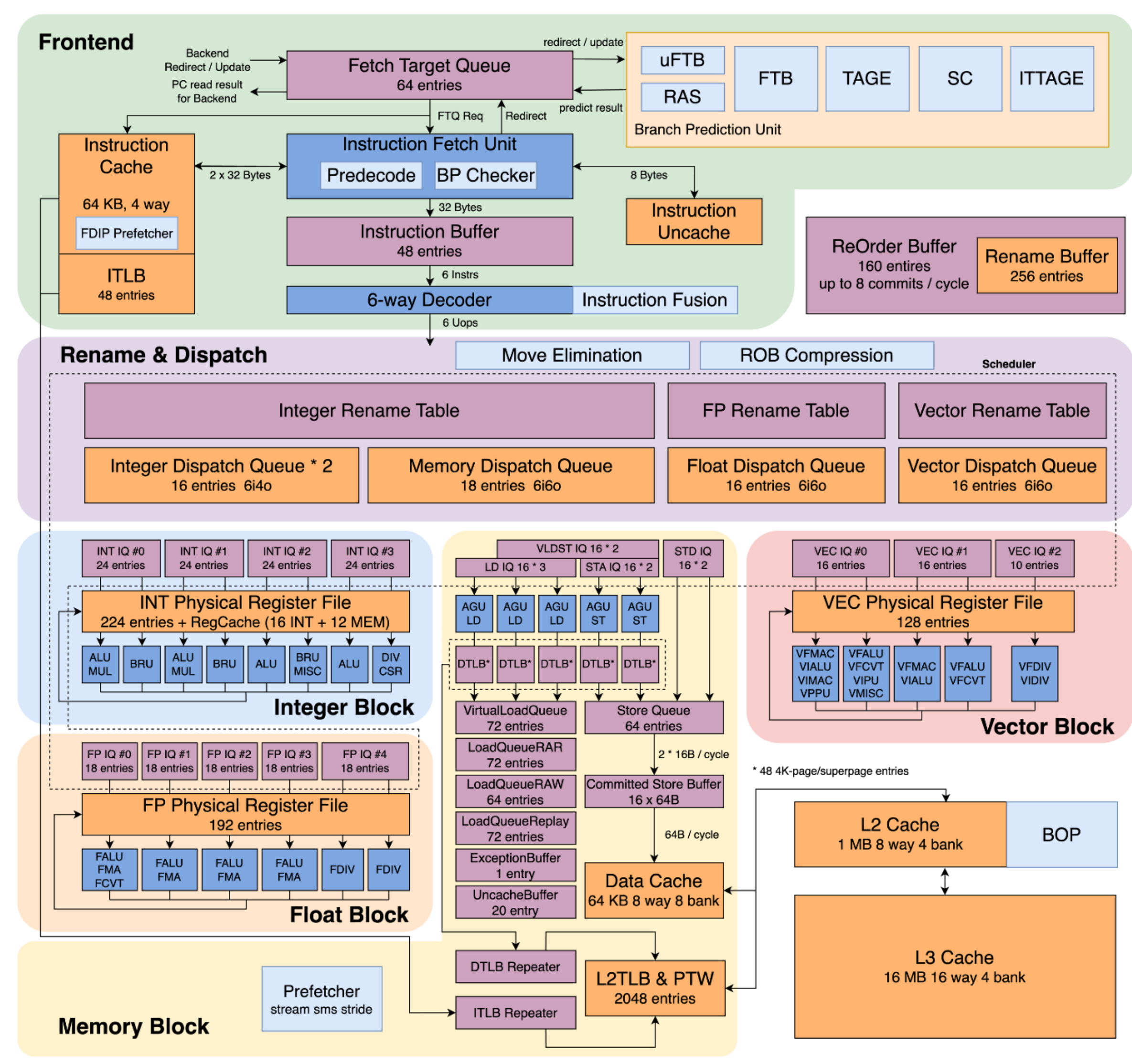
北京香山



> 4.5K stars, > 630 forks on GitHub

昆明湖架构总览

- 6 宽度重命名
- 多级覆盖分支预测
- 224 INT + 192 FP + 128 VEC 物理寄存器
- 160 ROB + 256 RAB（RenAme Buffer）
- 64 KB ICache / DCache
- 1 MB L2 Cache
- FDIP 指令预取
- stream / sms / stride / bop / tp 预取器
- 48-entry ITLB / DTLB + 2048-entry L2TLB



开源开放的处理器芯片研发能力体系

- 香山核心价值是构建一套**芯片敏捷设计基础设施**，缩短迭代优化周期
- **开源开放能力体系**，联合企业加速处理器研发节奏，支持按需快速定制芯片

成果开源

能力开放

新特性提出 → RTL实现 → 功能验证 → 性能验证 → 物理设计验证

RTL实现：基础架构；设计语言

XiangShan
开源硬件平台

Chisel
敏捷设计语言

Transforms
代码转换工具

nexus-am
简易运行时环境

TL-C总线请求
生成器

TL-C Agent
总线/缓存
标准行为模型

Golden Mem
多核内存模型

NEMU
指令集模拟器

TL-C Scoreboard
总线行为/数据
检查框架

SMP-Diff
多核验证框架

Difftest
差分验证框架

LightSSS
仿真进程快照

LogViewer
日志可视化工具

Waveform Terminator
波形解析工具

功能验证：测试生成；参考实现；结果对比；出错现场复现；出错现场分析

性能验证：测试生成；仿真平台；仿真加速器平台(WIP)；FPGA平台；性能数据分析

GCPT
全系统状态检查点

BetaPoint
程序特征采样方法(WIP)

NEMU-Functional
快速结构预热方法(WIP)

KFDB
性能建模与分析工具

自动化持续集成测试
完成后进入主线

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

- 服务器登录
- 开发环境准备
- RTL 生成
- RTL 仿真

……

功能验证

- Nexus-AM
- NEMU
- Difftest
- TL-Test

……

性能验证

- Checkpoint
- GEM5
- Top-Down
- Constantin

……

- ***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
  - 对于 Linux 和 Mac 用户，请直接打开终端

```
> ssh guest@tutorial.xiangshan.cc    # 输入这行命令后会提示“...... password:”
> xiangshan-2024                     # 请输入密码：xiangshan-2024
```

  - 可以看到

```
$ guest@xiangshan-tutorial:~$
```

  - 请输入

```
$ guest@xiangshan-tutorial:~$ pwd    # 输入 pwd，查看当前目录
/home/guest                          # 应输出您位于 /home/guest 目录下
```

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
# SET TLT_HOME:                           $XS_PROJECT_ROOT/tl-test-new
# SET gem5_home:                          $XS_PROJECT_ROOT/gem5
```

准备工作

```
# Project Structure

$ tree -d -L 1

.
├── DRAMsim3
├── gem5
├── NEMU
├── nexus-am
├── NutShell
├── tl-test-new
├── tutorial
├── verilator
└── XiangShan

# 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 -j8 # 约 20 min

Options:
    CONFIG=MinimalConfig        # 香山的配置，已默认指定为 MinimalConfig
    // EMU_THREADS=4            # 仿真的线程数，默认为 1 即可
    // EMU_TRACE=1              # 启用波形生成，默认不生成波形
    // WITH_DRAMSIM=1           # 启用模拟 DRAM 仿真的 DRAMSim3
    // WITH_CHISELDB = 1        # 开启 ChiselDB 特性
    // WITH_CONSTANTIN = 1      # 开启 Constantin 特性
```

- 编译大概需要 20 min

准备一个新终端

```
# 打开新的终端，重新登录云服务器
$ ssh guest@tutorial.xiangshan.cc # 输入这行命令后会提示“...... password:”
$ xiangshan-2024                  # 请输入密码：xiangshan-2024

# 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

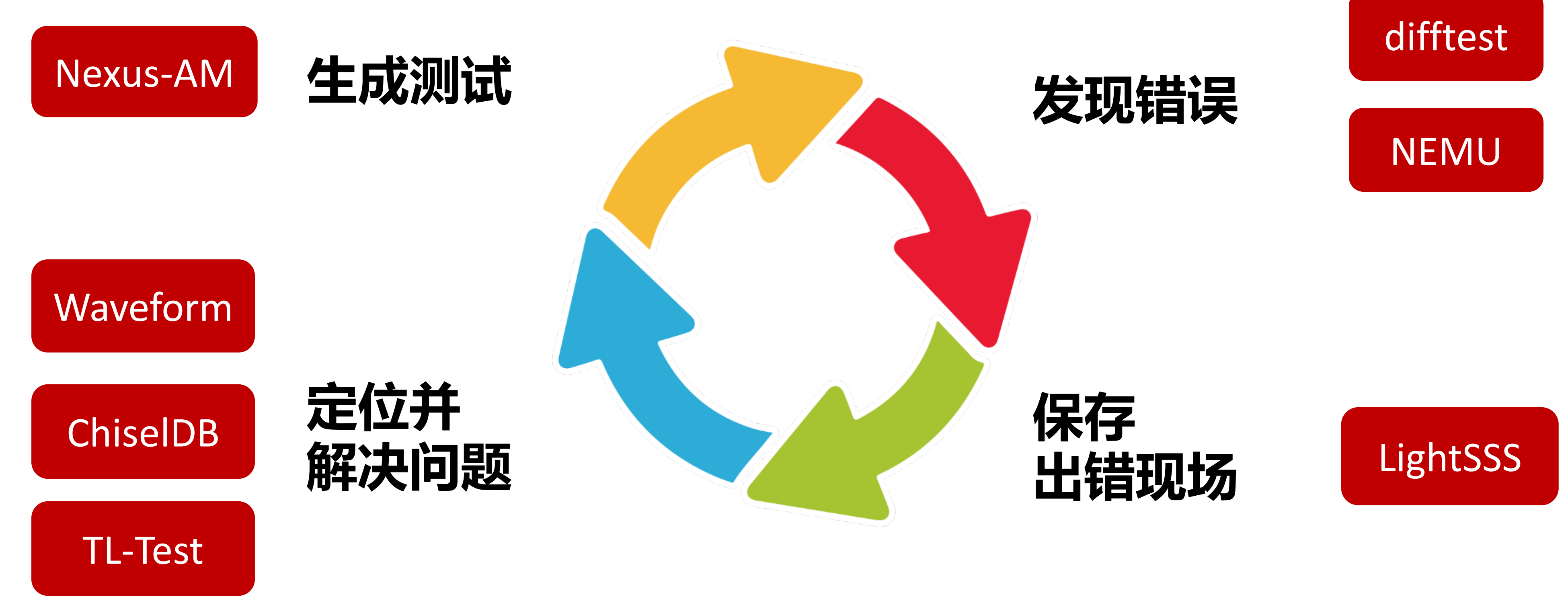
```
# 运行仿真程序
$ cd p1-basic-func
$ ./emu -i hello.bin --no-diff 2>hello.err

# 一些常用的选项:
    -i                        # 运行的 workload
    -C / -I                   # 最大周期数 / 最大指令数
    --diff=PATH / --no-diff   # 作为 REF 的参考设计路径 / 不开启 difftest
```

基本仿真流程完毕

- **这样我们就完成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: 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 \
  ITERATIONS=1 LINUX_GNU_TOOLCHAIN=1 TOTAL_DATA_SIZE=400
  设置迭代次数，工具链，以及数据大小
# 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 -j          将NEMU 编译成香山的参考设计
```

NEMU

- **动手试试**：在 NEMU 上运行 Coremark

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

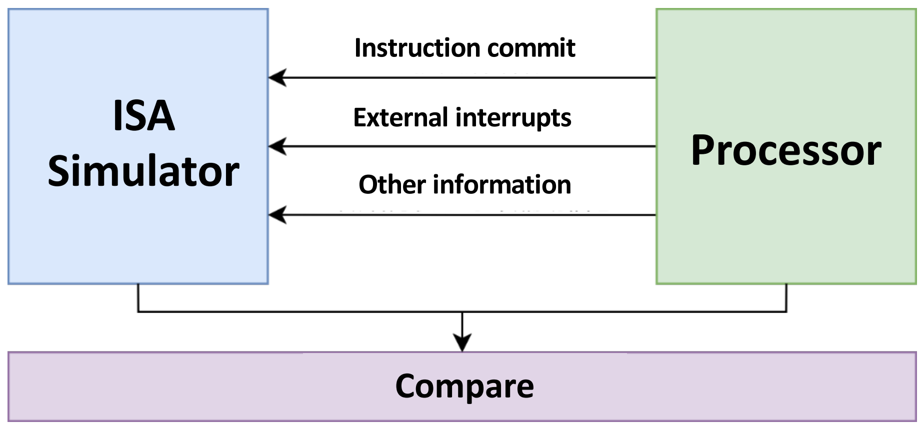
```
# cd $NEMU_HOME
# ./build/riscv64-nemu-interpreter -b \
$AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin
```

连续运行，更快

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
  - 支持应用于硬件仿真加速器
- **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1 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.err     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 2>perf.err
emu compiled at Aug 19 2024, 22:33:45
[WARNING] /home/guest/linzhida/XiangShan/build/constantin.txt does not exist, so all constants default to init
ialized values.
Using simulated 32768B flash
Using simulated 8192MB RAM
The image is /home/guest/linzhida/nexus-am/apps/coremark/build/coremark-riscv64-xs.bin
The reference model is /home/guest/linzhida/NEMU/build/riscv64-nemu-interpreter-so
The first instruction of core 0 has commited. Difftest enabled.
Running CoreMark for 1 iterations
400 performance run parameters for coremark.
CoreMark Size    : 133
Total time (ms)  : 200
Iterations       : 1
Compiler version : GCC11.4.0
seedcrc          : 0xe97b
[0]crclist       : 0x31be
[0]crcmatrix     : 0x0000
[0]crcstate      : 0x7bd3
[0]crcfinal      : 0x31be
Finished in 200 ms.
==================================================
CoreMark Iterations/Sec 5000
Core 0: HIT GOOD TRAP at pc = 0x80001eb4
Core-0 instrCnt = 39606, cycleCnt = 68161, IPC = 0.581065
Seed=0 Guest cycle spent: 68165 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 35280ms
```

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 0080002a7a cmtcnt 8
commit group [01]: pc 0080002a8a cmtcnt 10
commit group [02]: pc 008000108a cmtcnt 8
commit group [03]: pc 0080000e70 cmtcnt 9
commit group [04]: pc 00800015ae cmtcnt 10
commit group [05]: pc 0080000e78 cmtcnt 9
commit group [06]: pc 00800015c2 cmtcnt 9
commit group [07]: pc 0080000e7c cmtcnt 9
commit group [08]: pc 0080000e68 cmtcnt 9
commit group [09]: pc 0080000eaa cmtcnt 9
commit group [10]: pc 0080000e70 cmtcnt 9
commit group [11]: pc 00800015dc cmtcnt 10
commit group [12]: pc 0080001600 cmtcnt 13
commit group [13]: pc 008000163e cmtcnt 18
commit group [14]: pc 0080001aa6 cmtcnt 12 <--
commit group [15]: pc 0080002a6e cmtcnt 8
```

```
==============  REF Regs  ==============
  $0: 0x0000000000000000   ra: 0x00000000800015dc   sp: 0x000000008000bf00   gp: 0x0000000000000000
  tp: 0x0000000000000000   t0: 0x0000000000000000   t1: 0x0000000000000001   t2: 0x0000000000000009
  s0: 0x0000000000000000   s1: 0x0000000000000000   a0: 0x0000000000000085   a1: 0x000000008000bf18
  a2: 0x0000000000000002   a3: 0x0000000000000002   a4: 0x0000000000000002   a5: 0x0000000000000007
  a6: 0x0000000000000001   a7: 0x0000000000000003   s2: 0x0000000000000000   s3: 0x0000000000000000
  s4: 0x0000000000000000   s5: 0x0000000000000000   s6: 0x0000000000000000   s7: 0x0000000000000000
  s8: 0x0000000000000000   s9: 0x0000000000000000  s10: 0x0000000000000000  s11: 0x0000000000000000
  t3: 0x0000000080003b10   t4: 0x0000000000000001   t5: 0x000000008000bda1   t6: 0x0000000000000031
 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: 0x0000000080001aa0 mstatus: 0x8000000a00006000 mcause: 0x0000000000000000 mepc: 0x0000000000000000
                       sstatus: 0x8000000200006000 scause: 0x0000000000000000 sepc: 0x0000000000000000
```

```
privilegeMode: 3
     a3 different at pc = 0x0080001aa6, right= 0x0000000000000002, wrong = 0x0000000000000000
Core 0: ABORT at pc = 0x80000cce
Core-0 instrCnt = 1280, cycleCnt = 11879, IPC = 0.107753
Seed=0 Guest cycle spent: 11882 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 6335ms
```

Difftest

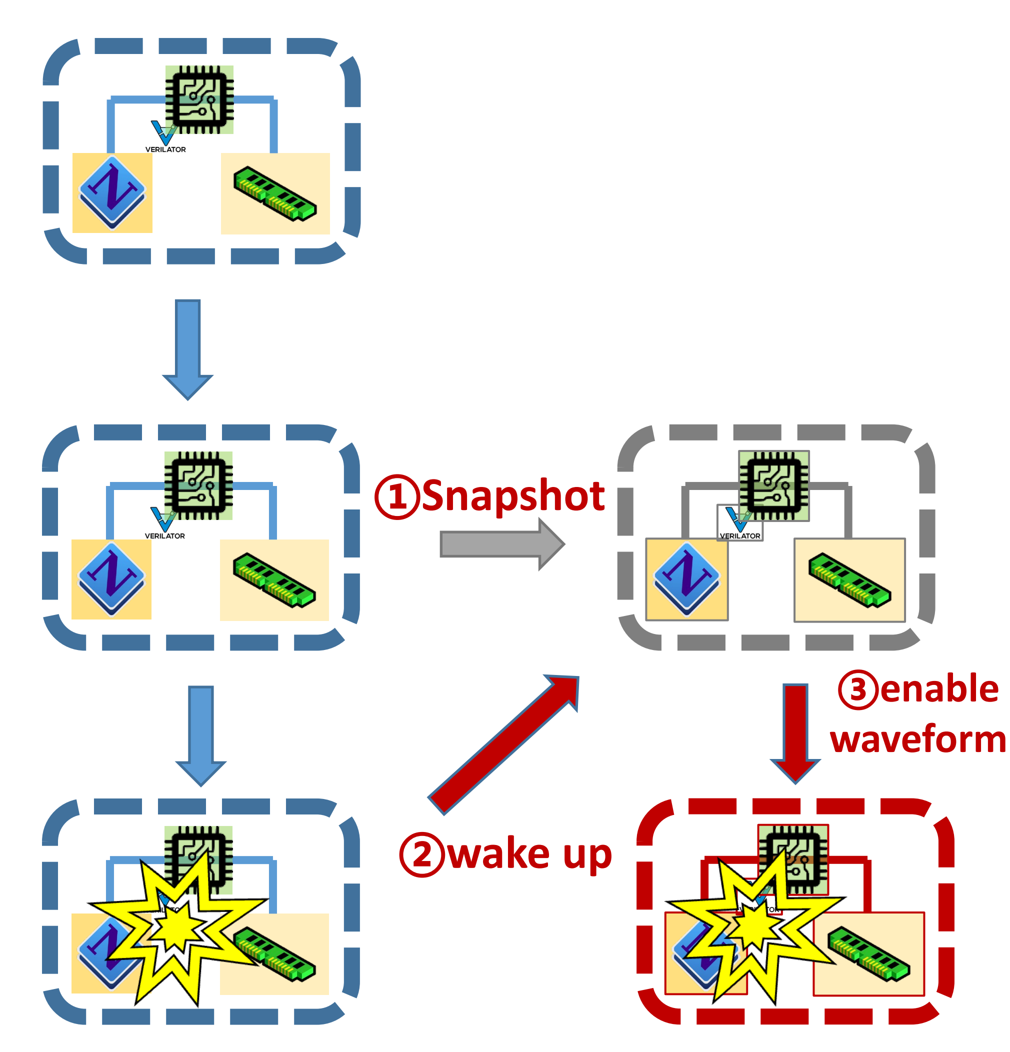
- **动手试试**：在 difftest 检测出 bug 后生成波形

```
$ bash run-emu-dumpwave.sh
$ ls $NOOP_HOME/build/*.vcd -lh   # 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 11000 \        生成波形的起始周期。在上一步中，香山在约 12000 个周期处报错结束
-e 13000 \        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 \
2 > lightsss.err
```

不再需要使用复杂的参数

LightSSS

- LightSSS 工作时你应该看到

```
privilegeMode: 3
     a3 different at pc = 0x0080001aa6, right= 0x0000000000000002, wrong = 0x0000000000000000
[FORK_INFO pid(17919)] the oldest checkpoint start to dump wave and dump nemu log...
[FORK_INFO pid(17919)] dump wave to /home/guest/linzhida/XiangShan/build/2024-08-21@10:12:24_8373.vcd...
Running CoreMark for 1 iterations

============== Commit Group Trace (Core 0) ==============
commit group [00]: pc 0080002a7a cmtcnt 8
commit group [01]: pc 0080002a8a cmtcnt 10
commit group [02]: pc 008000108a cmtcnt 8
commit group [03]: pc 0080000e70 cmtcnt 9
commit group [04]: pc 00800015ae cmtcnt 10
```

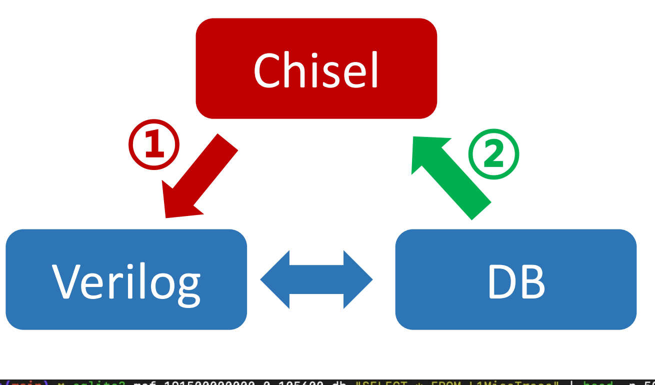
- 此后，仿真会从最近的快照处重启

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



Chisel Bundle

Verilog

ChiselDB

C++ Class

SQL Table

Key Arch Info

Debug Env

Debugging

Performance Analysis

ChiselDB

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

```scala
// 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)
```

ChiselDB：辅助调试工具

- **TL-Log：持久化的总线事务/数据库**
  - 基于 ChiselDB 技术，将运行中的总线事务记录到 SQLite3 数据库中
  - 支持离线出错检索以及性能分析

时间戳	监控器名称	通道	Opcode	权限升降	状态转换信息	SourceID	SinkID	地址	数据			
116854134	L2_L1I_0	A	AcquireBlock	Grow	NtoB	1	0	803d3680	0000000000000000	0000000000000000	0000000000000000	0000000000000000
116854146	L2_L1I_0	D	GrantData	Cap	toT	1	16	803d3680	fa843783f9043903	06090063f8f43c23	f8f4382300093783	378300093023c3bd
116854147	L2_L1I_0	D	GrantData	Cap	toT	1	16	803d3680	03e32e07bb030089	f0ef8526c881f49b	bb0300893783bbdf	855a010925832e07
123191840	L2_L1I_0	C	ReleaseData	Shrink	TtoN	0	0	803d3680	fa843783f9043903	06090063f8f43c23	f8f4382300093783	378300093023c3bd
123191841	L2_L1I_0	B	Probe	Cap	toN	0	0	803d3680	0000000000000000	0000000000000000	0000000000000000	0000000000000000
123191841	L2_L1I_0	C	ReleaseData	Shrink	TtoN	0	0	803d3680	03e32e07bb030089	f0ef8526c881f49b	bb0300893783bbdf	855a010925832e07
123191848	L2_L1I_0	D	ReleaseAck	Cap	toT	0	31	803d3680	0000000020fab05b	0000000020fab45b	0000000020fab85b	0000000020fabc5b
123191851	L3_L2_0	C	ReleaseData	Shrink	TtoN	31	0	803d3680	fa843783f9043903	06090063f8f43c23	f8f4382300093783	378300093023c3bd
123191852	L3_L2_0	C	ReleaseData	Shrink	TtoN	31	0	803d3680	03e32e07bb030089	f0ef8526c881f49b	bb0300893783bbdf	855a010925832e07
123191862	L3_L2_0	D	ReleaseAck	Cap	toT	31	16	803d3680	86a6f7043583dcd5	f0ef856a865a8762	3583b7654481e17f	865a86a68762f704
123191863	L2_L1I_0	C	ProbeAck	Shrink	TtoN	0	0	803d3680	0000000000000000	0000000000000000	0000000000000000	0000000000000000
123191878	L3_L2_0	C	Release	Report	NtoN	30	0	803d3680	5597bf494981aa09	8522cc2585930000	f84ef15dda0f00ef	00194703b775e84a

示例：数据库中一组违背缓存一致性的请求序列

ChiselDB

- **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 (L2->L3)数据都设置成常量
$ cd ../p2-chisledb && cat cdb_err.patch
```

```
--- a/huancun/src/main/scala/huancun/noninclusive/SinkC.scala
+++ b/huancun/src/main/scala/huancun/noninclusive/SinkC.scala

@@ -99,7 +99,7 @@ class SinkC(implicit p: Parameters) extends BaseSinkC {
-       buffer(insertIdx)(count) := c.bits.data
+       buffer(insertIdx)(count) := 0xABCDEF.U
```

ChiselDB

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

ChiselDB

- **动手试试**：分析 Cache 一致性违例
- Difftest 报错信息
  - 出错地址：0x800419c0

```
The first instruction of core 0 has commited. Difftest enabled.
cacheid=1,idtfr=0,realpaddr=0x800419c0: Refill test failed!
addr: 800419c0
Gold: 000000000000000000000000000000000000000080041a70000000008001619800000000080041a50000000000000000000000000000000000000000000000000
Core: 000000000abcdef000000000000000000000000000000000000000000000000000000000000000000000000080041a50000000000000000000000000000000000000000000000000
Core 0: ABORT at pc = 0x80013966
Core-0 instrCnt = 17,709, cycleCnt = 50,670, IPC = 0.349497
```

ChiselDB

- **动手试试**：分析 Cache 一致性违例

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

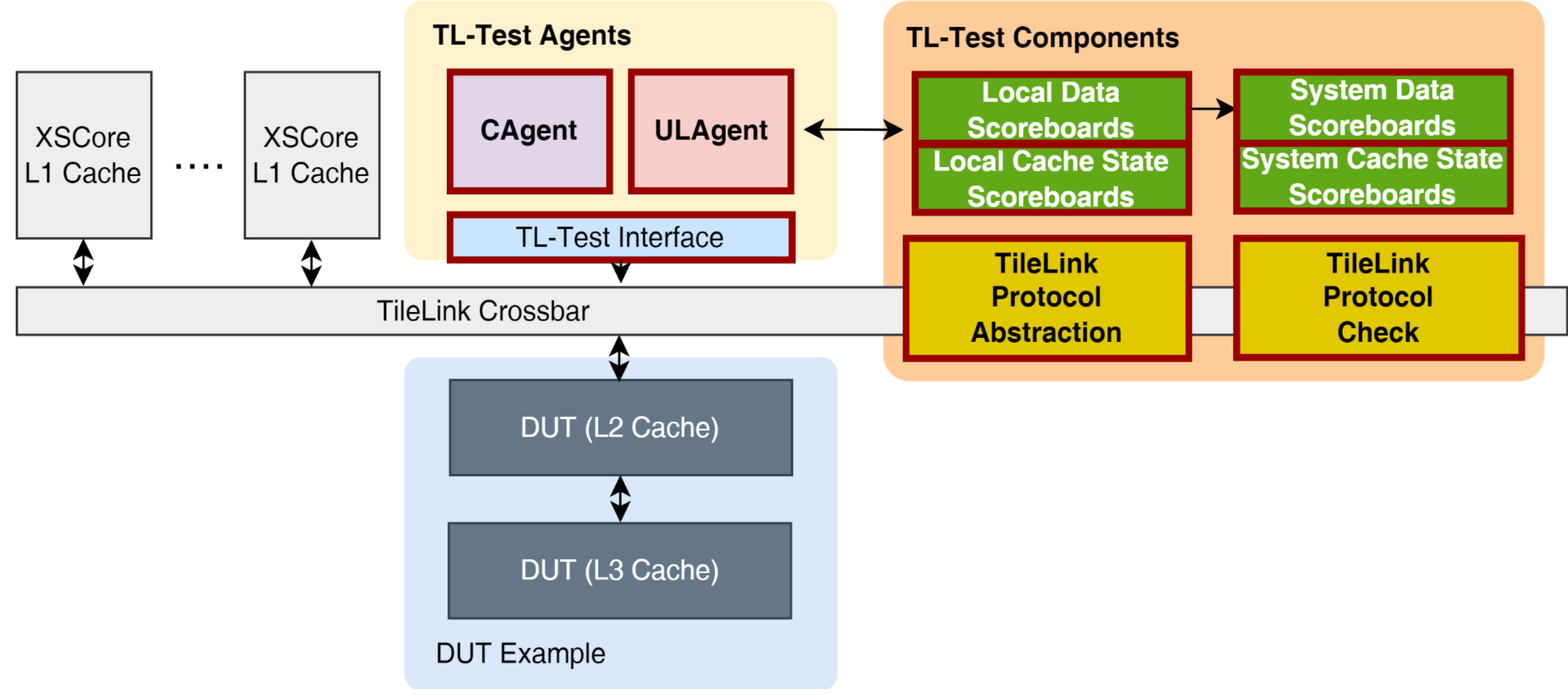
```
# 1. sqlite 查询所有在 Eaddr(0x800419c0) 上的事务
# 2. 使用脚本解析 TLLog
# sqlite3 $NOOP_HOME/build/*.db \
  "select * from TLLOG where ADDRESS=0x800419c0" | \
  sh $NOOP_HOME/scripts/utils/parseTLLog.sh
```

ChiselDB

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

# 数据成功地从 L1D 传输到了 L2
28189 L2_L1D_0 C ReleaseData Shrink TtoN 800419c0
0000000000000000 0000000000000000 0000000080041a70 0000000080016198

# 数据成功地从 L2 传输到了 L3
39637 L3_L2_0 C ProbeAckData Shrink TtoN 800419c0
0000000000000000 0000000000000000 0000000080041a70 0000000080016198

# 但是当 L1D 再次请求 Eaddr，从 L3 读取的数据却出错了
47371 L2_L1D_0 A AcquireBlock Grow NtoB 800419c0
47377 L3_L2_0 A AcquireBlock Grow NtoB 0 1 800419c0
47413 L3_L2_0 D GrantData Cap toT 800419c0
0000000000abcdef 0000000000000000 0000000000000000 0000000000000000
# 因此可以推断 L3 记录 Release Data 时出现了错误
```

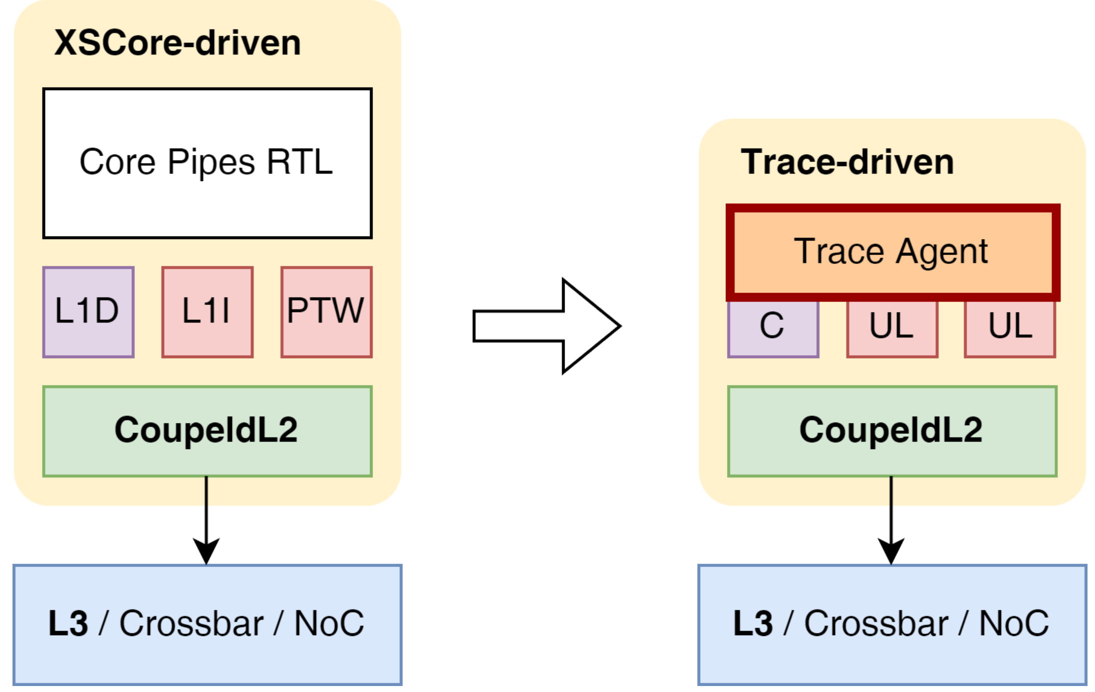
TL-Test：基于 TileLink 的多功能缓存验证框架

- **作为缓存验证的基础设施**
  - 支持 **TL 协议正确性**的检查；支持**缓存一致性**检查
  - 支持**快速构造测试用例**；实现**可约束随机测试**



TL-Test：辅助调试工具

- **TL-Trace：基于 Trace 生成激励**
  - 将处理器核的访存 trace 转换为测试激励
  - 支持功能性问题的快速复现；性能问题的快速调试



TL-Test

```
# Project Structure

$ cd $TLT_HOME && tree -d -L 1

.
├── assets
├── configs      # 测试配置
├── dut          # 开箱即用的DUT对象
├── main         # TL-Test源码
├── run          # 编译后可执行文件以及运行后产生的波形、log
├── scripts      # 脚本
└── tutorial

# 进入 TL-Test tutorial 目录

$ cd $XS_PROJECT_ROOT/tutorial/p3-tl-test
```

TL-Test

- **动手试试**：分析 Cache 一致性违例

```
# 查看插入的bug – 我们将Release (L1->L2) 数据设置错误偏移
$ cat tlt_err.patch
```

```
--- a/tl-test-new/dut/CoupledL2/src/main/scala/coupledL2/GrantBuffer.scala
+++ b/tl-test-new/dut/CoupledL2/src/main/scala/coupledL2/GrantBuffer.scala
@@ -87,7 +87,7 @@ class GrantBuffer(implicit p: Parameters) extends
LazyModule {
-       d.data := data
+       d.data := data << 8
```

TL-Test

- **动手试试：**分析 Cache 一致性违例

```
# 编译运行（比较费时，因此我们用已经编译好的TL-Test框架）
# 通常使用 `make coupledL2-test-l2l3 THREADS_BUILD=16 CXX_COMPILER=clang++-17` 命令进行编译
# 单核 coupledL2-test-l2l3 ；双核 coupledL2-test-l2l3l2

$ bash tltest_step1_run.sh
```

```
# cd $TLT_HOME/run && ./tltest_v3lt 2>&1 | tee tltest_v3lt.log
```

TL-Test

- **动手试试**：分析 Cache 一致性违例
- TL-Test报错信息
  - 出错地址：0x16000

```
[1740] [tl-test-new-INFO] #0 L2[0].C[0] [data complete D] [GrantData toT] source: 0x1, addr: 0x16000, alias: 0, data: [ 00 d6 ... 00 75 ... ]

dut: [ 00 d6 ... 00 75 ... ]
ref: [ d6 26 ... 75 0d ... ]
[1740] [tl-test-new-ERROR] [tlc_assert failure at int GlobalBoard<unsigned long>::data_check(TLLocalContext *, const uint8_t *, const uint8_t *, std::string) [T = unsigned long]:263]
[1740] [tl-test-new-ERROR] [tlc_assert failure from system #0]
[1740] [tl-test-new-ERROR] info: Data mismatch from status SB_VALID!
```

TL-Test

- **动手试试**：分析 Cache 一致性违例

```
# 分析
$ bash tltest_step2_analyze.sh
```

```
# 1. grep 查询所有在 Eaddr(0x16000) 上的事务
# 2. 可使用 TL-Log 辅助调试

# grep “addr: 0x16000” $TLT_HOME/run/tltest_v3lt.log
```

TL-Test

```
# 结果: [Time | Core | Channel | Opcode | Source | Address | Data]

# 数据成功地从 L1D 传输到了 L2
[800] L2[0].C[0] [fire C] [ReleaseData TtoN] source: 0xa, addr: 0x16000, data: [ d6 26 ... ]
[802] L2[0].C[0] [fire C] [ReleaseData TtoN] source: 0xa, addr: 0x16000, data: [ 75 0d ... ]

# 但是当 L1D 再次请求 Eaddr，从 L2返回的数据却出错了
[844]   L2[0].C[0] [fire A] [AcquireBlock NtoT] source: 0x1, addr: 0x16000
[1738] L2[0].C[0] [fire D] [GrantData toT] source: 0x1, addr: 0x16000, data: [ 00 d6 ... ]
[1740] L2[0].C[0] [fire D] [GrantData toT] source: 0x1, addr: 0x16000, data: [ 00 75 ... ]
# 因此可以推断 L2 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

umd-memsys/
DRAMsim3

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

XSPerf

Constantin

Top-down analysis

性能分析

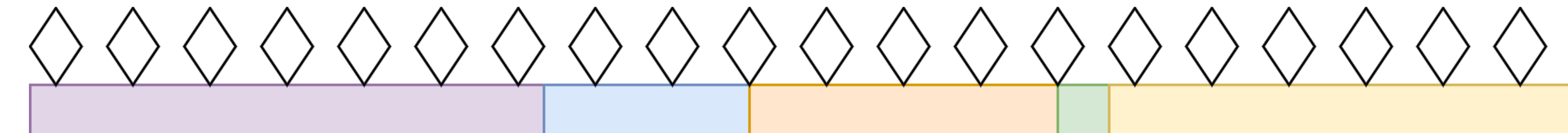
性能评估

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: Simpoint Checkpoint

- step0: 为 checkpoints 准备 NEMU 环境

```
$ cd $XS_PROJECT_ROOT/tutorial/p4-checkpoint
$ bash simpoint_step0_prepare.sh
```

```
# simpoint_step0_prepare.sh
# cd $NEMU_HOME
# git submodule update --init
# cd $NEMU_HOME/resource/simpoint/simpoint_repo
# make clean && make                          生成 simpoint generator 的二进制文件

# cd $NEMU_HOME
# make clean
# make riscv64-xs-cpt_defconfig && make -j 8  编译NEMU

# cd $NEMU_HOME/resource/gcpt_restore
# rm -rf $XS_PROJECT_ROOT/tutorial/part5-checkpoint/gcpt
# make -C $NEMU_HOME/resource/gcpt_restore/   \ 生成 gcpt restorer 的二进制文件
        O=$XS_PROJECT_ROOT/tutorial/part5-checkpoint/gcpt    \ 结果输出目录
        GCPT_PAYLOAD_PATH=/$XS_PROJECT_ROOT/tutorial/part5-checkpoint
/bin/stream_100000.bin
```

0x80000000

GCPT Restorer
GCPT Flag
Register CPTs
BBL
Linux
User Application

Basic format of RISC-V Checkpoint

Checkpoint: Simpoint Checkpoint

- **step1: 执行一轮 workload，收集程序行为信息**

```
$ bash simpoint_step1_profiling.sh
```

```
# simpoint_step1_profiling.sh
# source simpoint_env.sh                    配置环境变量

# rm -rf $RESULT

# $NEMU ${BBL_PATH}/${workload}.bin \   指定workload
#     -b                            \   以批处理模式运行
#     -D $RESULT                    \   指定生成checkpoints 所在的目录
#     -C $profiling_result_name     \   指定任务结果名称
#     -w stream                     \   指定workload
#     --simpoint-profile            \   做simpoint profiling
#     --cpt-interval ${interval}    \   指定 simpoint 的指令间隔 ：50000000
#     > >(tee $log/stream-out.txt) 2> >(tee ${log}/stream-err.txt)
                                        指定log输出路径，用于后续脚本分析
```

Checkpoint: Simpoint Checkpoint

```
-------------------------------------------------------------
Your clock granularity/precision appears to be 26 microseconds.
Each test below will take on the order of 1023153 microseconds.
   (= 39352 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:           941.2     0.085167     0.084997     0.085384
Scale:          215.4     0.371746     0.371488     0.371877
Add:            275.9     0.435193     0.434995     0.435358
Triad:          256.4     0.468193     0.467980     0.468407
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-06 on all three arrays
-------------------------------------------------------------
[src/monitor/monitor.c:195,parse_args] Doing Simpoint Profiling
[src/checkpoint/path_manager.cpp:54,init] Cpt id: -1
[src/checkpoint/path_manager.cpp:72,setSimpointProfilingOutputDir] Created /home/guest/fenghaoyuan/tutorial/p4-checkpoint/simpoint_result/simpoint-profiling/stream/

[src/checkpoint/simpoint.cpp:83,init] Doing simpoint profiling with interval 50000000
[src/checkpoint/serializer.cpp:378,next_index] set next index 0
[src/checkpoint/path_manager.cpp:91,setCheckpointingOutputDir] donot set checkpoint path without Checkpoint mode
[src/isa/riscv64/init.c:191,init_isa] NEMU will start from pc 0x80000000
[src/monitor/image_loader.c:203,load_img] Loading image (checkpoint/bare metal app/bbl) form cmdline: /home/guest/fenghaoyuan/tutorial/p4-checkpoint/gcpt/build/gcpt.
```

NEMU 会在执行 STREAM 程序的过程中
以50000000指令间隔进行分析

Checkpoint: Simpoint Checkpoint

- step2: 进行聚类，得到权重最高的多个程序片段

```
$ bash simpoint_step2_cluster.sh
```

```
# simpoint_step2_cluster.sh

# export CLUSTER=$RESULT/cluster/${workload} && mkdir -p $CLUSTER
# mkdir -p $LOG_PATH/cluster_logs/cluster

# random1=`head -20 /dev/urandom | cksum | cut -c 1-6`
# random2=`head -20 /dev/urandom | cksum | cut -c 1-6`

# $NEMU_HOME/resource/simpoint/simpoint_repo/bin/simpoint    \
#     -loadFVFile $PROFILING_RES/${workload}/simpoint_bbv.gz \    加载profiling结果：频率向量文件
#     -saveSimpoints $CLUSTER/simpoints0                     \    simpoints文件保存路径
#     -saveSimpointWeights $CLUSTER/weights0                 \    simpoint 权重文件保存路径
#     -inputVectorsGzipped                                   \    输入向量文件已使用gzip 压缩
#     -maxK 3                   \   使用最大聚类数
#     -numInitSeeds 2           \   每次运行时采用不同随机初始化的次数，只选取其中最好的聚类结果
#     -iters 1000               \   最大迭代次数
#     -seedkm ${random1}        \   初始k-means随机种子
#     -seedproj ${random2}      \   随机线性投影的随机种子
#     > >(tee $log/${workload}-out.txt) 2> >(tee $log/${workload}-err.txt)   指定log输出路径，用于后续脚本分析
```

Checkpoint: Simpoint Checkpoint

```
------------------------------------------------------------
Run number 2 of at most 4, k = 3
------------------------------------------------------------
  ------------------------------------------------------------
  Initialization seed trial #1 of 2; initialization seed = 634853
  ------------------------------------------------------------
  Initialized k-means centers using random sampling: 3 centers
  Number of k-means iterations performed: 1
  BIC score: -27.9287
  Distortion: 7.02268
  Distortions/cluster: 0.542115 3.14007 3.34049
  Variance: 0.702268
  Variances/cluster: 0.271058 1.04669 0.668099
  ------------------------------------------------------------
  Initialization seed trial #2 of 2; initialization seed = 634854
  ------------------------------------------------------------
  Initialized k-means centers using random sampling: 3 centers
  Number of k-means iterations performed: 4
  BIC score: -10.3726
  Distortion: 6.04602
  Distortions/cluster: 5.29257 0.753145 0.000309052
  Variance: 0.604602
  Variances/cluster: 0.661571 0.753145 0.000309052
  The best initialization seed trial was #2
 -----------------------------------------------------------------
 Post-processing runs
 -----------------------------------------------------------------
 -----------------------------------------------------------------
  For the BIC threshold, the best clustering was run 2 (k = 3)
```

计算 K-means 聚类方法的信息

获得具有最高权重的多个程序片段

Checkpoint: Simpoint Checkpoint

- step3: 根据聚类结果，生成对应的 Checkpoints

```
$ bash simpoint_step3_genspt.sh        # 大约需要 2 分钟
```

```
# simpoint_step3_genspt.sh

# export CLUSTER=$RESULT/cluster
# mkdir -p $LOG_PATH/checkpoint_logs

# $NEMU ${BBL_PATH}/${workload}.bin \
#     -b                            \  以批处理模式运行
#     -D $RESULT                    \  指定生成checkpoints 所在的目录
#     -C heckpoint                  \  指定任务名称
#     -w stream                     \  指定workload名称
#     -S $CLUSTER                   \  simpoints 聚类结果所在目录
#     --cpt-interval $interval      \  指定 simpoint 的指令间隔：50000000
#     > >(tee $log/stream-out.txt) 2> >(tee $log/stream-err.txt)
                                      指定log输出路径，用于后续脚本分析
```

Checkpoint: Simpoint Checkpoint

```
[src/profiling/profiling_control.c:21,reset_inst_counters] Start profiling, resetting inst count from 275933546 to 1, (n_remain_total will not be cl
eared)

[src/checkpoint/serializer.cpp:325,instrsCouldTakeCpt] Should take cpt now: 914 next point 0
[src/checkpoint/serializer.cpp:176,serializeRegs] Writing int registers to checkpoint memory @[0x800edde0, 0x800edee0) [0xedde0, 0xedee0)
[src/checkpoint/serializer.cpp:184,serializeRegs] Writing float registers to checkpoint memory @[0x800edee8, 0x800edfe8) [0xedee8, 0xedfe8)
[src/checkpoint/serializer.cpp:195,serializeRegs] Writing Vector registers to checkpoint memory @[0x800edee8, 0x800edfe8) [0xfdff8, 0xfe1f8)
[src/checkpoint/serializer.cpp:204,serializeRegs] Writing PC: 0x3f9e18a552 at addr 0x800edb8
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x105: 0xffffffff801f7318
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x106: 0x7
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x10c: 0x1
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x140: 0xffffffd880179480
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x141: 0x3f9e19b524
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x142: 0xf
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x143: 0x3f9e2b51d0
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x180: 0x8000700000100c5b
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x200: 0x200000000
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x300: 0xa000066a2
[src/checkpoint/serializer.cpp:227,serializeRegs] CSR 0x301: 0x80000000003411af
[src/checkpoint/serializer.cpp:254,serializeRegs] Touching Flag: 0xbeef at addr 0x800ecdb0
[src/checkpoint/serializer.cpp:258,serializeRegs] Record mode flag: 0x0 at addr 0x800ecdc0
[src/checkpoint/serializer.cpp:263,serializeRegs] Record time: 0x0 at addr 0x800ecdc8
[src/checkpoint/serializer.cpp:267,serializeRegs] Record time: 0x0 at addr 0x800ecdd0

[src/checkpoint/serializer.cpp:127,serializePMem] Written 0x7fffffff bytes

[src/checkpoint/serializer.cpp:127,serializePMem] Written 0x7fffffff bytes

[src/checkpoint/serializer.cpp:127,serializePMem] Written 0x7fffffff bytes

[src/checkpoint/serializer.cpp:127,serializePMem] Written 0x7fffffff bytes

[src/checkpoint/serializer.cpp:127,serializePMem] Written 0x4 bytes

[src/checkpoint/serializer.cpp:161,serializePMem] Checkpoint done!
```

开始生成 checkpoint

保存整型，浮点和向量寄存器以及pc

保存 CSR 寄存器

保存其他信息

将 checkpoints 写入 gz 文件

Checkpoint: Simpoint Checkpoint

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

```
$ bash simpoint_step4_run_xs.sh
```

```
# simpoint_step4_run_xs.sh (约1 min)

# ./emu \
#   -i `find $RESULT/checkpoint/stream -type f -name "*_.gz" | tail -1` \
                        运行的workload路径
#   --diff $NOOP_HOME/ready-to-run/riscv64-nemu-interpreter-so \
                        开启比对的REF标准设计路径
#   --max-cycles=50000 \      执行最大指令周期数
#   2>simpoint.err
```

Checkpoint: Simpoint Checkpoint

- **Step5：为 GEM5/XiangShan 批量运行 checkpoints 提供配置文件**

```
$ python3 simpoint_step5_dumpresult.py
```

- 收集profiling/checkpoint信息，生成用于GEM5执行的list文件

```
# checkpoint.lst
stream_5 stream/5 0 0 20 20
stream_11 stream/11 0 0 20 20
stream_1 stream/1 0 0 20 20
负载名称 | checkpoint路径 | 跳过指令数(默认0) | 功能预热指令数(默认0) | 细节预热指令数(默认20) | 采样指令数(默认20)
```

- 收集cluster信息，生成用于XiangShan执行的json文件

```
# cluster.json
{"stream": {"insts": "654228721",
"points": {"7": "0.615385", "0": "0.153846", "11": "0.230769"}}}
负载名称 | 指令数 | checkpoint序号 | checkpoint权重
```

Checkpoint: Uniform Checkpoint

- **Step0：为 checkpoints 准备 NEMU 环境**

```
$ cd $XS_PROJECT_ROOT/tutorial/p4-checkpoint
$ bash simpoint_step0_prepare.sh
```

- **Step1：用 NEMU 生成 uniform checkpoints**

```
$ bash uniform_cpt.sh
```

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

```
$ bash uniform_run_nemu.sh
```

XS-Gem5: 架构性能评估工具

- **目的**
  - 高效迭代微架构新特性
  - 快速进行端到端性能评估
- **XS-Gem5**
  - 在主线 Gem5 O3CPU 基础上与香山微架构尽可能进行对齐
  - 支持 Difftest 与 Checkpoint
  - 增加用于香山性能分析的相关脚本

XS-Gem5: 架构性能评估工具

- **动手试试**：编译生成香山 Gem5

```
$ cd $XS_PROJECT_ROOT/tutorial/p5-xs-gem5 # 进入 XS-Gem5 tutorial 目录
$ bash 0-gem5_prepare.sh
```

```
prepare_gem5() {
    pushd $gem5_home && \
    cd ext/dramsim3 && \ # 构建DRAMSim3
    git clone git@github.com:umd-memsys/DRAMSim3.git DRAMsim3 && \
    cd DRAMsim3 && mkdir -p build && cd build && cmake .. && make -j `nproc` && \
    popd
}
build_gem5() {
    pushd $gem5_home && \
    scons build/RISCV/gem5.opt --gold-linker -j `nproc` && \
    popd
}
```

XS-Gem5: 架构性能评估工具

- **动手试试：在 Gem5 上运行裸机应用**

```
$ bash 1-gem5_run_coremark.sh
```

```
pushd $gem5_home && \
export GCBV_REF_SO=$NEMU_HOME/build/riscv64-nemu-gem5-ref-so && \
mkdir -p util/xs_scripts/coremark && \ # 建立 CoreMark 工作目录
cd util/xs_scripts/coremark && \
$gem5_home/build/RISCV/gem5.opt $gem5_home/configs/example/xiangshan.py \
--raw-cpt \
--generic-rv-cpt=$NOOP_HOME/ready-to-run/coremark-2-iteration.bin && \
popd
```

XS-Gem5: 架构性能评估工具

ecs-user@xiangshan-tutorial:~/chenyangyu/xs-env/tutorial/p5-xs-gem5

```
coremark-2-iteration.bin to pmem 0x7f72654ee000
build/RISCV/mem/physical.cc:608: info: First 4 bytes are 0x13 0x4 0x0 0x0
build/RISCV/sim/system.cc:561: info: Restored from Xiangshan RISC-V Checkpoint
**** REAL SIMULATION ****
build/RISCV/sim/simulate.cc:194: info: Entering event queue @ 0.  Starting simulation...
build/RISCV/cpu/base.cc:1266: warn: Start memcpy to NEMU from 0x7f72654ee000, size=8589934592
build/RISCV/cpu/base.cc:1269: warn: Start regcpy to NEMU
build/RISCV/dev/serial/uartlite.cc:35: warn: Write to other uartlite addr 12 is not implemented
Running CoreMark for 2 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total time (ms)  : 110
Iterations       : 2
Compiler version : GCC10.2.0
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x72be
Finised in 110 ms.
==================================================
CoreMark Iterations/Sec 18181
Exiting @ tick 313394625 because m5_exit instruction encountered when simulating XS
~/chenyangyu/xs-env/tutorial/p5-xs-gem5
→ p5-xs-gem5 git:(tutorial-2024) ✗
```

XS-Gem5: 架构性能评估工具

- 动手试试：查看 Gem5 性能计数器

```
$ bash 2-gem5_counter.sh
```

```
cat $gem5_home/util/xs_scripts/coremark/m5out/stats.txt
```

XS-Gem5: 架构性能评估工具

ecs-user@xiangshan-tutorial:~/chenyangyu/xs-env/tutorial/p5-xs-gem5 ⌥⌘1

```
➜  p5-xs-gem5 git:(tutorial-2024) ✗ ./2-gem5_counter.sh | shuf -n 20 | cut -f1 -d'#'
system.l3.prefetcher.pfHitInWB                     0
system.cpu.dcache.prefetcher.berti.pfUnused_srcs::11            0
system.l3.prefetcher.pfHitInCache_srcs::10            0
system.cpu.dcache.ReadReq.misses::cpu.data          161
system.cpu.iew.renameStallReason::DTlbStall           10
system.cpu.mmu.dtb.l2sptlbUnusedRemove             0
system.l2_caches.InvalidateReq.missRate::cpu.data            1
system.cpu.mmu.l2_shared.writeL2l3TlbMisses           0
system.cpu.dcache.prefetcher.spp.pfUseful_srcs::9            0
system.cpu.branchPred.commitControlSquashLatencyDist::2            0      0.00%      0.00%
system.l2_caches.ReadExReq.mshrMisses::cpu.dcache.prefetcher            8
system.cpu.branchPred.tage.bank_0.updateTableMispreds::1         211
system.cpu.dcache.prefetcher.bop_large.pfUseful_srcs::9            0
system.cpu.dcache.prefetcher.bop_learned.pfUsefulSpanPage            0
system.cpu.dcache.prefetcher.pfUseful              0
system.cpu.dcache.prefetcher.berti.pfHitInWB_srcs::3            0
system.cpu.dcache.prefetcher.cmc.pfHitInCache_srcs::1            0
system.mem_ctrls.bwInstRead::cpu.inst        39821998
system.l2_caches.demandMisses::cpu.data            49
system.cpu.icache.overallAvgMissLatency::cpu.inst 52512.614493
➜  p5-xs-gem5 git:(tutorial-2024) ✗
```

Cache MPKI 分析

- **动手试试：运行 Cache MPKI 分析脚本**

```
$ bash 3-gem5_cache.sh
```

```
pushd gem5_data_proc && \
mkdir -p results && \
export PYTHONPATH=`pwd` && \
python3 batch.py \
  -s /home/share/xs-model-l1bank \
  -o gem5-cache-example.csv \
  --cache && \
python3 simpoint_cpt/compute_weighted.py \
  -r gem5-cache-example.csv \
  -j /home/share/xs-model-l1bank/cluster-0-0.json \
  -o weighted.csv
```

注：为了更好地展示，这里采用SPEC CPU 2006 Checkpoint运行结果作为示例。

Cache MPKI 分析

ecs-user@xiangshan-tutorial:~/chenyangyu/xs-env/tutorial/p5-xs-gem5 ⌥⌘1

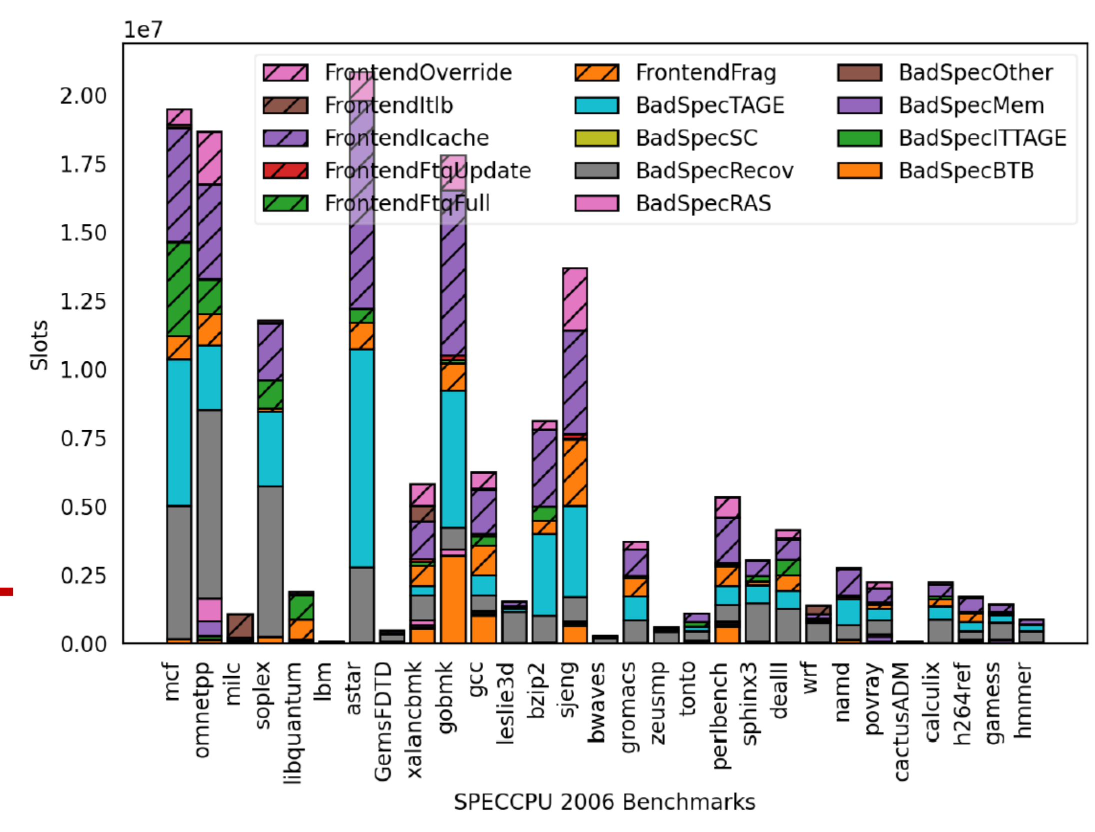
```
['GemsFDTD', 'astar', 'bwaves', 'bzip2', 'cactusADM', 'calculix', 'dealII', 'gamess', 'gcc', 'gobmk', 'gromacs', 'h264ref', 'hmmer', 'lbm', 'leslie3d', 'libquantum', 'mcf', 'milc', 'namd', 'omnetpp', 'perlbench', 'povray', 'sjeng', 'soplex', 'sphinx3', 'tonto', 'wrf', 'xalancbmk', 'zeusmp']
```

	Cycles	Insts	L1D.MPKI	L2.MPKI	...	l3_acc	l3_miss	cpi	coverage
mcf	3.027e+07	2.000e+07	270.673	100.326	...	2.294e+06	698680.709	1.513	1.0
omnetpp	1.774e+07	2.000e+07	57.713	47.026	...	1.026e+06	492397.438	0.887	1.0
astar	1.618e+07	2.000e+07	27.252	15.890	...	3.543e+05	48744.997	0.809	1.0
gobmk	1.155e+07	2.000e+07	5.485	1.525	...	3.856e+04	14055.195	0.577	1.0
soplex	1.119e+07	2.000e+07	33.391	25.666	...	9.042e+05	175472.264	0.559	1.0
milc	1.071e+07	2.000e+07	68.772	27.739	...	9.773e+05	702851.431	0.535	1.0
bzip2	9.521e+06	2.000e+07	12.069	4.226	...	9.716e+04	4578.537	0.476	1.0
gromacs	9.386e+06	2.000e+07	32.104	0.658	...	1.976e+04	7220.444	0.469	1.0
gcc	9.261e+06	2.000e+07	13.926	15.001	...	4.285e+05	114394.979	0.463	1.0
sjeng	9.021e+06	2.000e+07	1.439	1.211	...	2.422e+04	20958.788	0.451	1.0
bwaves	8.071e+06	2.000e+07	6.480	6.880	...	5.694e+05	89426.320	0.404	1.0
perlbench	7.967e+06	2.000e+07	4.548	1.431	...	3.134e+04	19389.243	0.398	1.0
lbm	7.920e+06	2.000e+07	8.029	19.248	...	6.969e+05	122392.254	0.396	1.0
zeusmp	7.450e+06	2.000e+07	12.428	5.687	...	1.894e+05	85596.788	0.373	1.0
tonto	6.759e+06	2.000e+07	5.334	2.342	...	8.166e+04	5793.963	0.338	1.0
calculix	6.720e+06	2.000e+07	0.932	0.340	...	8.947e+03	3756.312	0.336	1.0
GemsFDTD	6.559e+06	2.000e+07	12.818	14.021	...	5.250e+05	374770.295	0.328	1.0
leslie3d	6.552e+06	2.000e+07	21.274	10.611	...	3.842e+05	144669.463	0.328	1.0
xalancbmk	6.515e+06	2.000e+07	19.989	11.652	...	3.246e+05	38865.836	0.326	1.0
namd	6.494e+06	2.000e+07	14.142	0.212	...	5.496e+03	4312.841	0.325	1.0
povray	6.162e+06	2.000e+07	17.487	0.195	...	3.944e+03	3190.261	0.308	1.0
dealII	5.944e+06	2.000e+07	8.144	1.365	...	9.003e+04	10105.618	0.297	1.0
sphinx3	5.736e+06	2.000e+07	14.614	11.872	...	3.751e+05	20540.266	0.287	1.0
h264ref	4.941e+06	2.000e+07	3.710	1.137	...	3.053e+04	10893.001	0.247	1.0
wrf	4.908e+06	2.000e+07	2.839	1.865	...	1.151e+05	12708.207	0.245	1.0
hmmer	4.489e+06	2.000e+07	1.042	0.818	...	2.653e+04	494.690	0.224	1.0
cactusADM	4.474e+06	2.000e+07	2.154	2.749	...	9.884e+04	34987.191	0.224	1.0
libquantum	4.390e+06	2.000e+07	1.185	1.214	...	5.120e+05	69486.561	0.220	1.0
gamess	4.261e+06	2.000e+07	2.287	0.169	...	3.496e+03	2324.679	0.213	1.0

```
[29 rows x 17 columns]
~/chenyangyu/xs-env/tutorial/p5-xs-gem5
→ p5-xs-gem5 git:(tutorial-2024-gem5-reorder) ✗
```

Top-Down: 性能分析模型

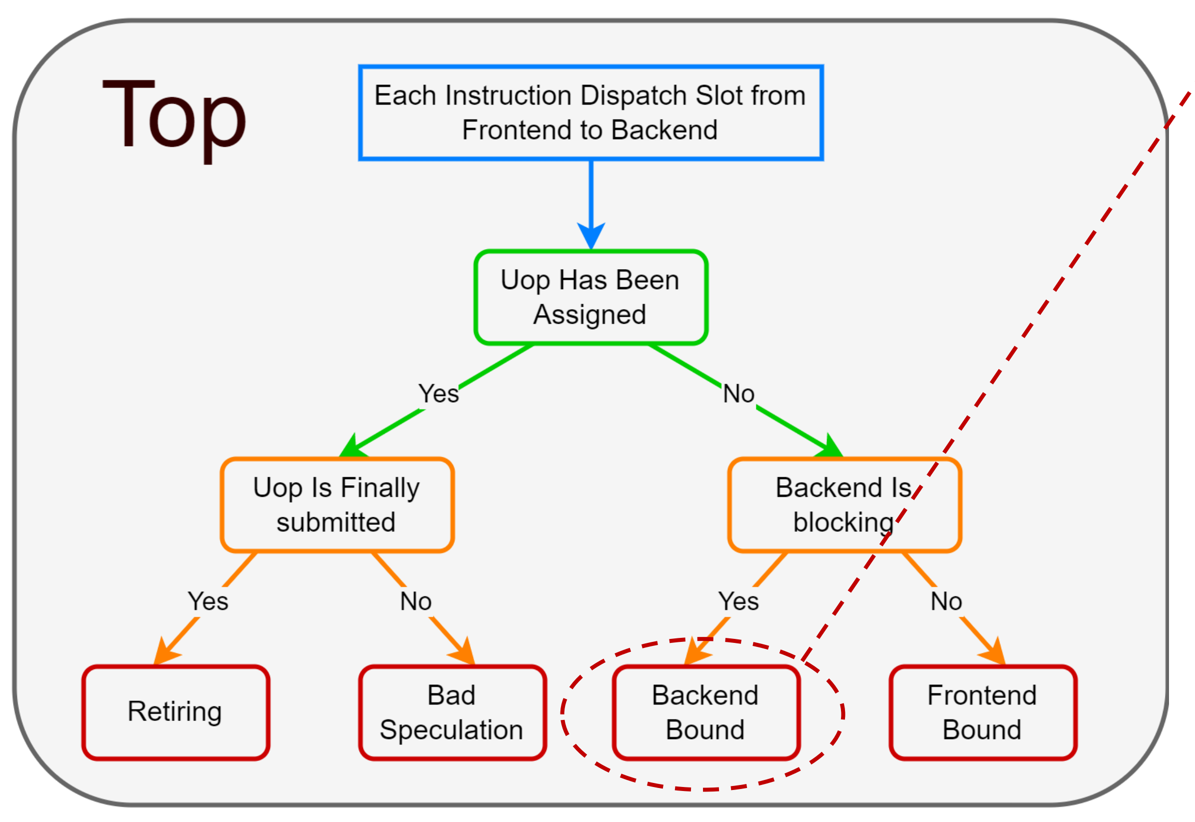
- **目的**
  - 将分散的性能事件**有层次地**组织起来
  - **精确**计算性能事件对处理器性能的影响
- **Top-Down on XiangShan**
  - 对 RISC-V 指令集进行**针对性优化适配**
  - 针对香山的微架构进行**性能计数器**优化
  - 进一步完善 Top-Down 模块的**层次化设计**
  - 不会造成性能事件的重复或遗漏
  - 不需要提前假设性能事件的阻塞周期

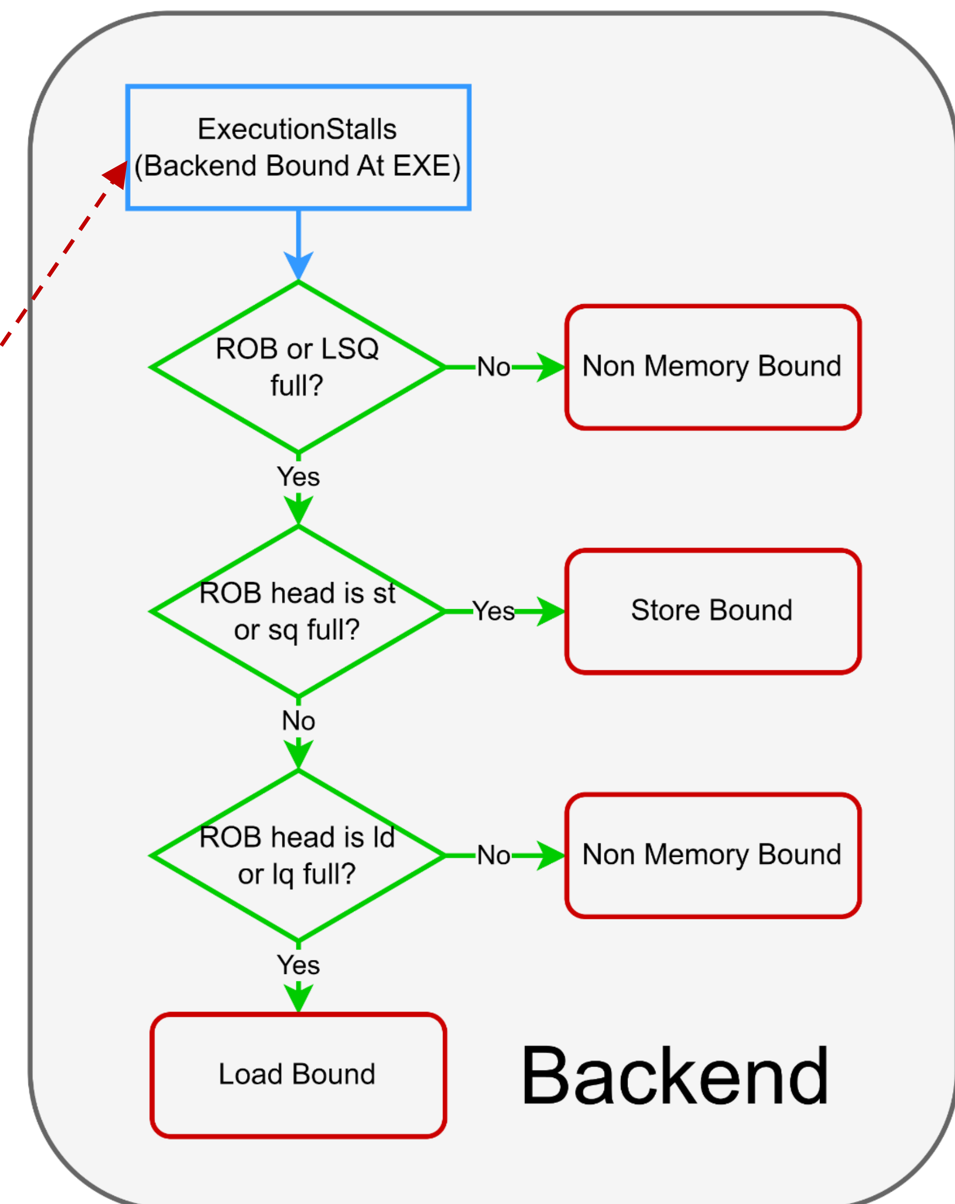


[1] A Top-Down method for performance analysis and counters architecture

Top-Down

- 以 Top 和 Backend 为例

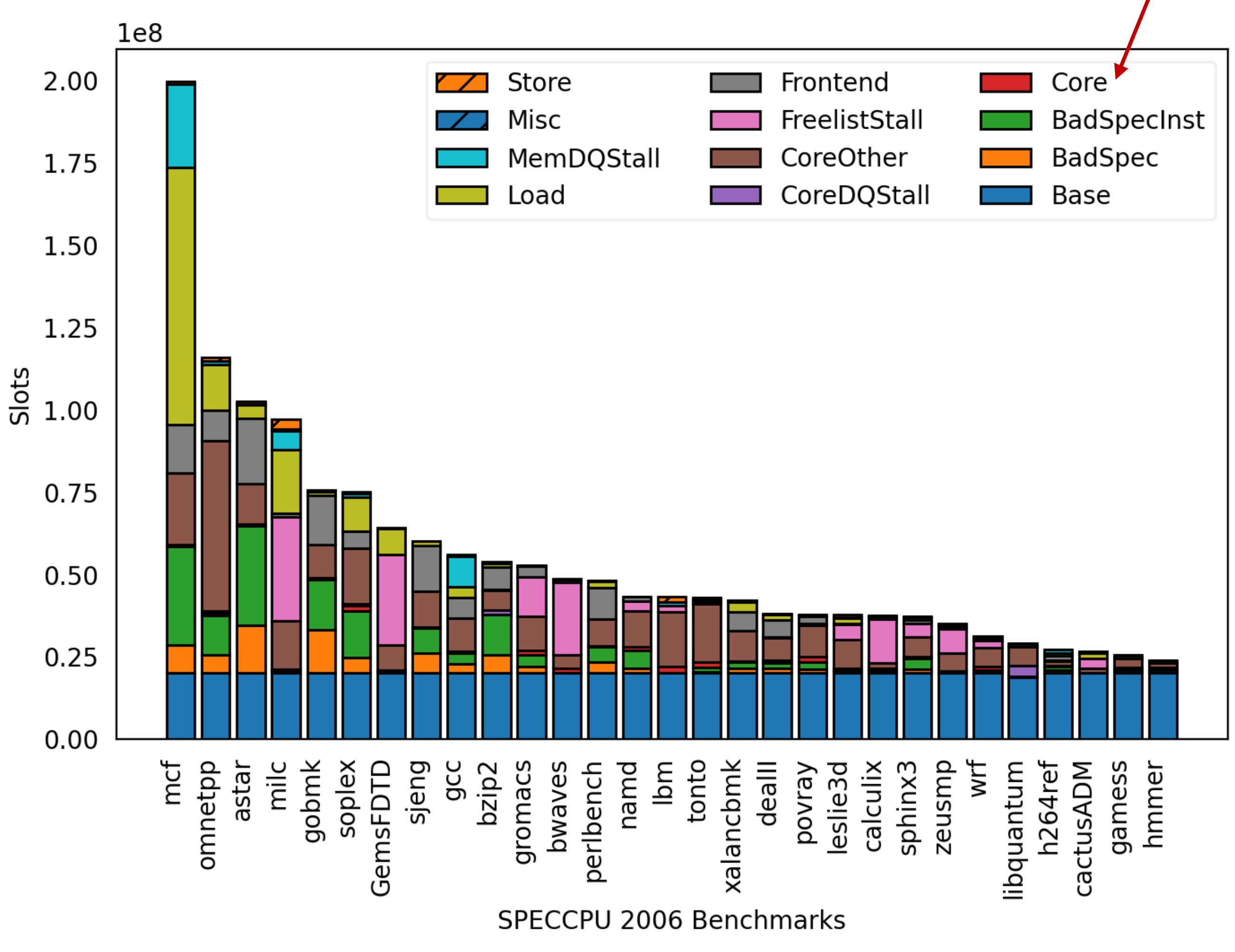




Top-Down

- 示例：Top-Down 可视化分析结果

导致阻塞的原因分类



Top-Down

- 动手试试：在 gem5 中查看 Top-Down 结果

```
$ bash 4-gem_topdown.sh
```

```
pushd gem5_data_proc && \
python3 batch.py \
  -s $gem5_home/util/xs_scripts/coremark \
  -t --topdown-raw && \
popd
```

Top-Down

ecs-user@xiangshan-tutorial:~/chenyangyu/xs-env/tutorial/p5-xs-gem5 ⌥⌘1

```
system.l2_caches.demandMisses::cpu.data            49
system.cpu.icache.overallAvgMissLatency::cpu.inst 52512.614493
➜  p5-xs-gem5 git:(tutorial-2024) ✗ ./3-gem_topdown.sh
~/chenyangyu/xs-env/tutorial/p5-xs-gem5/gem5_data_proc ~/chenyangyu/xs-env/tutorial/p5-xs-gem5
workload: m5out, point: 0, segments: 1
[('m5out_0', '/home/ecs-user/chenyangyu/xs-env/gem5/util/xs_scripts/coremark/m5out/stats.txt')]
Process finished job: m5out_0
/home/ecs-user/chenyangyu/xs-env/gem5/util/xs_scripts/coremark/m5out/stats.txt
{'m5out_0': {'OtherFetchStall': 0, 'OtherStall': 0, 'CommitSquash': 328812, 'ResumeUnblock': 0, '
OtherMemStall': 0, 'Atomic': 0, 'MemCommitRateLimit': 2058, 'MemNotReady': 1576428, 'MemSquashed'
: 0, 'StoreMemBound': 0, 'StoreL3Bound': 0, 'StoreL2Bound': 0, 'StoreL1Bound': 930, 'LoadMemBound
': 4746, 'LoadL3Bound': 0, 'LoadL2Bound': 0, 'LoadL1Bound': 756, 'InstNotReady': 12, 'SerializeSt
all': 63390, 'InstSquashed': 130470, 'InstMisPred': 0, 'FetchBufferInvalid': 0, 'SquashStall': 33
504, 'FragStall': 1721534, 'TrapStall': 0, 'IntStall': 0, 'BpStall': 186960, 'DTlbStall': 6, 'ITl
bStall': 0, 'IcacheStall': 282756, 'NoStall': 951427, 'ipc': 0.66608, 'Insts': 626869, 'Cycles':
941131, 'point': '0', 'workload': 'm5out', 'bmk': 'm5out'}}
         Atomic  BpStall  CommitSquash  Cycles  ...    bmk    ipc  point  workload
m5out_0       0   186960        328812  941131  ...  m5out  0.666      0     m5out

[1 rows x 37 columns]
~/chenyangyu/xs-env/tutorial/p5-xs-gem5
➜  p5-xs-gem5 git:(tutorial-2024) ✗
```

SPEC CPU 算分

- **动手试试：运行 SPEC CPU 算分脚本**

```
$ bash 5-gem5_spec06_score.sh
```

```
pushd gem5_data_proc && \
mkdir -p results && \
export PYTHONPATH=`pwd` && \
python3 batch.py -s /home/share/xs-model-l1bank \
  -o gem5-score-example.csv && \
python3 simpoint_cpt/compute_weighted.py \
  -r gem5-score-example.csv \
  -j /home/share/xs-model-l1bank/cluster-0-0.json \
  --score score.csv
```

注：为了更好地展示，这里采用SPEC CPU 2006 Checkpoint运行结果作为示例。

SPEC CPU 算分

tmux ⌥⌘1

```
wrf          250.853   11170.0   14.843        [25/2118]
leslie3d     214.233    9400.0   14.626        1.0
gamess       451.597   19580.0   14.452        1.0
namd         210.250    8020.0   12.715        1.0
tonto        260.012    9840.0   12.615        1.0
hmmer        260.274    9330.0   11.949        1.0
gromacs      201.133    7140.0   11.833        1.0
perlbench    289.646    9770.0   11.244        1.0
sjeng        366.934   12100.0   10.992        1.0
gobmk        320.031   10490.0   10.926        1.0
astar        229.743    7020.0   10.185        1.0
bzip2        410.138    9650.0    7.843        1.0
calculix     490.428    8250.0    5.607        1.0
score.csv
================ SPEC06 =================
================ Int =================
                time  ref_time    score  coverage
perlbench    289.646    9770.0   11.244        1.0
bzip2        410.138    9650.0    7.843        1.0
gcc          167.812    8050.0   15.990        1.0
mcf          141.107    9120.0   21.544        1.0
gobmk        320.031   10490.0   10.926        1.0
hmmer        260.274    9330.0   11.949        1.0
sjeng        366.934   12100.0   10.992        1.0
libquantum   148.529   20720.0   46.500        1.0
h264ref      428.160   22130.0   17.229        1.0
omnetpp      139.743    6250.0   14.908        1.0
astar        229.743    7020.0   10.185        1.0
xalancbmk     85.528    6900.0   26.892        1.0
Estimated Int score per GHz: 15.118648674723131
Estimated Int score @ 3.0GHz: 45.35594602416939
================ FP =================

h264ref      428.160   22130.0   17.229        1.0
omnetpp      139.743    6250.0   14.908        1.0
astar        229.743    7020.0   10.185        1.0
xalancbmk     85.528    6900.0   26.892        1.0
Estimated Int score per GHz: 15.118648674723131
Estimated Int score @ 3.0GHz: 45.35594602416939
================ FP =================
             time  ref_time    score   coverage
bwaves     175.907   13590.0   25.752        1.0
gamess     451.597   19580.0   14.452        1.0
milc       142.358    9180.0   21.495        1.0
zeusmp     196.183    9100.0   15.462        1.0
gromacs    201.133    7140.0   11.833        1.0
cactusADM  252.748   11950.0   15.760        1.0
leslie3d   214.233    9400.0   14.626        1.0
namd       210.250    8020.0   12.715        1.0
dealII     144.375   11440.0   26.413        1.0
soplex     136.694    8340.0   20.337        1.0
povray      93.311    5320.0   19.005        1.0
calculix   490.428    8250.0    5.607        1.0
GemsFDTD   176.662   10610.0   20.019        1.0
tonto      260.012    9840.0   12.615        1.0
lbm        149.280   13740.0   30.681        1.0
wrf        250.853   11170.0   14.843        1.0
sphinx3    350.359   19490.0   18.543        1.0
Estimated FP score per GHz: 16.548846674376477
Estimated FP score @ 3.0GHz: 49.64654002312943
================ Overall ================
Estimated overall score per GHz: 15.941323843383495
Estimated overall score @ 3.0GHz: 47.82397153015049
~/chenyangyu/xs-env/tutorial/p5-xs-gem5
➜  p5-xs-gem5 git:(tutorial-2024-gem5-reorder) ✗

[0] 0:[tmux]*                          "xiangshan-tutorial" 21:44 20-Aug-24
```

RTL Top-Down

- 在 RTL 代码中设置性能计数器

```
XiangShan/src/main/scala/xiangshan/backend/dispatch/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

RTL Top-Down

- 用 Top-Down 方法对 RTL 计数器进行层次分析

```
XiangShan/scripts/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

RTL Top-Down

- **动手试试：使用 Top-Down 工具分析 RTL 上的阻塞原因**

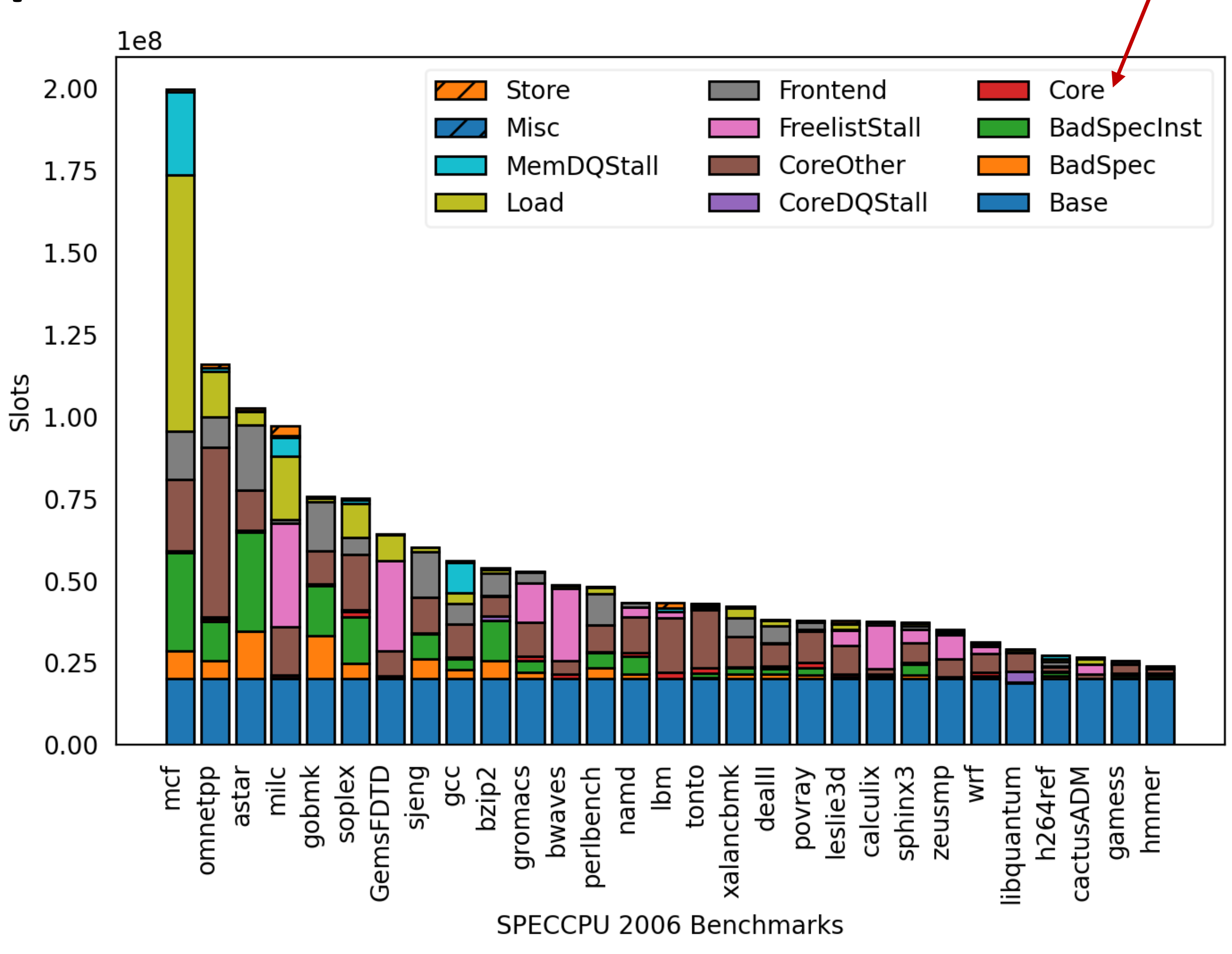
```
# 运行 Top-Down 分析工具（约 40s）
$ cd $XS_PROJECT_ROOT/tutorial/p6-xs-perf && bash rtl-top-down.sh
```

```
# cd ${NOOP_HOME}/scripts/top-down && \
  python3 top_down.py \
  -s /opt/SPEC06_EmuTasks_topdown \ # 指定性能计数器路径
  -j /opt/SPEC06_EmuTasks_topdown.json # 指定性能 json 文件

# ls ${NOOP_HOME}/scripts/top-down/results
result.png  results.csv  results-weighted.csv # 输出分析数据和可视化分析结果
```

RTL Top-Down

- 结果示例：Top-Down 可视化分析结果

导致阻塞的原因分类



XSPerf: RTL 仿真性能采集

- **多种类型**
  - **Accumulation**：基础的累加型计数器（log 打印）
  - **Histogram**：统计数值分布（log 打印）
  - **Rolling**：采集片段计数以分析性能变化（ChiselDB 存储）

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
```

XSPerf: RTL 仿真性能采集

- **基于 log 打印的使用方法**
  - 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

```
# tutorial/p6-xs-perf/
$ bash xs-perf-log.sh
```

```
# 查看 p1-basic-func 生成的结果
# cat ${XS_PROJECT_ROOT}/tutorial/p1-basic-func/perf.err
```

XSPerf: RTL 仿真性能采集

- 基于 log 打印的使用方法

```
[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

XSPerf: RTL 仿真性能采集

- **Rolling 的使用方法**
  - 添加代码 *XSPerfRolling('name', perfCnt, granularity, clock, reset)*
  - 编译新增 *WITH_CHISELDB=1 WITH_ROLLINGDB=1*，运行新增 *–dump-db*

```
# 由于编译执行很费时，我们只在这里展示命令
$ bash xs-perf-prepare.sh
```

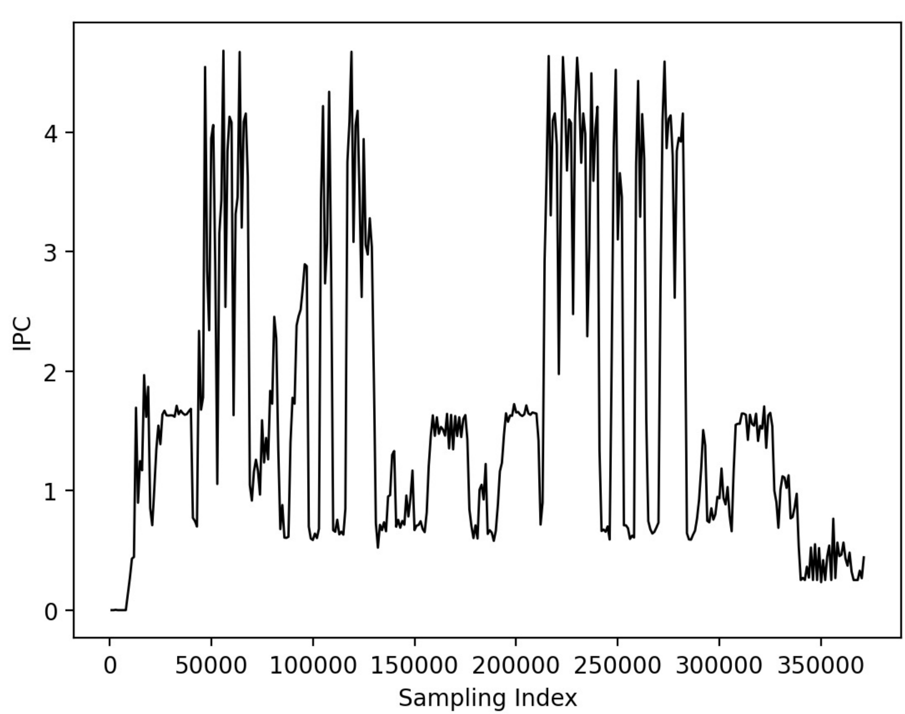
```
# cd ${NOOP_HOME}
# make clean
# make emu EMU_THREADS=4 WITH_CHISELDB=1 WITH_ROLLINGDB=1 -j8 \
#     PGO_WORKLOAD=${NOOP_HOME}/ready-to-run/coremark-2-iteration.bin \
#     PGO_MAX_CYCLE=10000 PGO_EMU_ARGS=--no-diff LLVM_PROFDATA=llvm-profdata
# ./build/emu -i ./ready-to-run/coremark-2-iteration.bin \
#     --diff ./ready-to-run/riscv64-nemu-interpreter-so --dump-db
# cp `find $NOOP_HOME/build/ -type f -name "*.db" | tail -1` \
#     ${XS_PROJECT_ROOT}/tutorial/p6-xs-perf/xs-perf-rolling.db
```

XSPerf: RTL 仿真性能采集

- **Rolling 的使用方法**
  - 查看结果并绘制性能片段采集的曲线图

```
# tutorial/p6-xs-perf/
$ bash xs-perf-rolling.sh
```

```
# cd ${NOOP_HOME}/scripts/rolling
# python3 rollingplot.py
${XS_PROJECT_ROOT}/tutorial/xs-
perf/XsPerfRolling-test.db ipc
# ls ${NOOP_HOME}/scripts/rolling/results
```



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
  - RTL 内更改参数再重新编译很耗时
  - 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前配置信号值

Constantin

- 常量值仅在运行时保持恒定

block_cycles_cache_0 = 11
block_cycles_cache_1 = 12
block_cycles_cache_2 = 13
block_cycles_cache_3 = 15

Conf file

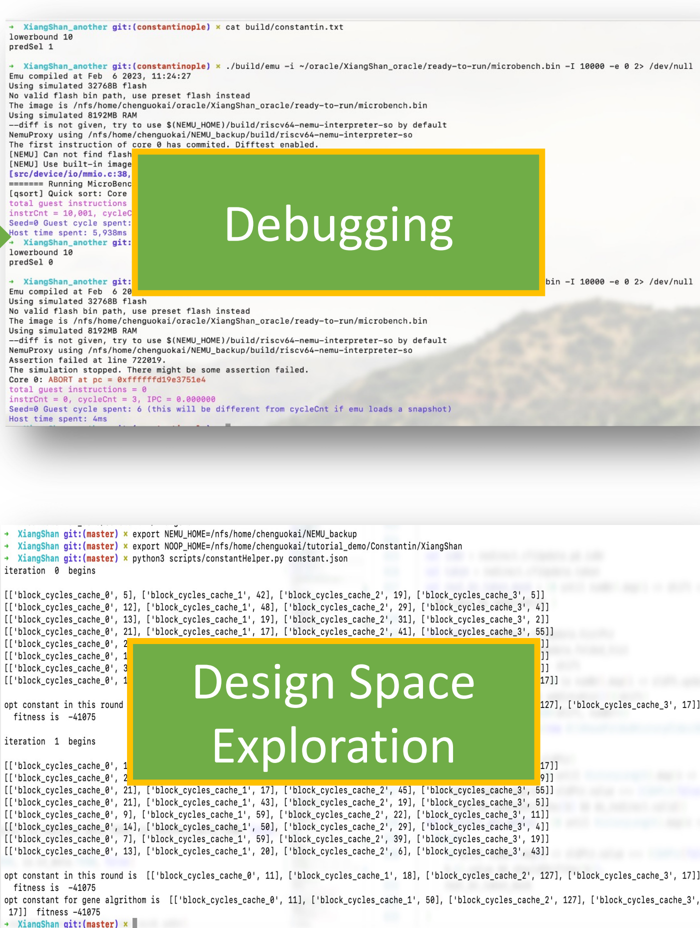
C++ Map

DPI-C interface

XS Core

Const A

Const B



Constantin

- **使用方法**：创建信号

```
// API: def createRecord(constName: String, initValue: BigInt = 0): UInt
import utility.Constantin
val setDqLatency = Constantin.createRecord("DelayQueueLatency"+name,
dQLatency)
```

- **使用方法**：配置文件

```
// 格式：信号名 取值
// 默认路径: ${NOOP_HOME}/build/constantin.txt
DelayQueueLatencyvbop 177
DelayQueueLatencypbop 242
```

Constantin

- **动手试试**：自动求解最优的参数值

```
# Patch: 打开 AutoSolving
$ cd $XS_PROJECT_ROOT/tutorial/p7-constantin && cat utility.patch
```

```
--- a/src/main/scala/utility/Constantin.scala
+++ b/src/main/scala/utility/Constantin.scala
@@ -21,7 +21,7 @@ import chisel3.util._
trait ConstantinParams {
def UIntWidth = 64
- def AutoSolving = false
+ def AutoSolving = true
```

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 patch
# cd $XS_PROJECT_ROOT/XiangShan/utility
# git apply ../../tutorial/utility.patch

# 编译
# make emu EMU_THREADS=4 WITH_CONSTANTIN=1 EMU_OPTIMIZE="" -j200
```

Constantin

- **动手试试**：自动求解最优的参数值

```
# 手动设置常量的赋值, tutorial/p7-constantin
$ sh step1-basic.sh
```

```
# 从标准输入（或管道）中读取常数值到emu
# wld=${XS_PROJECT_ROOT}/tutorial/${dir}/maprobe-riscv64-xs.bin
# cst=${XS_PROJECT_ROOT}/tutorial/${dir}/my_constantin.txt
# cat $cst | ./emu -i $wld -I 1000 \
#     --diff ${NOOP_HOME}/ready-to-run/riscv64-nemu-interpreter-so \
#     2> ${XS_PROJECT_ROOT}/tutorial/${dir}/constantin.err
```

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 my_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（约2min）
$ sh step2-solve.sh
```

```
# 脚本内容
# python3 ${NOOP_HOME}/scripts/constantHelper.py \
    ${XS_PROJECT_ROOT}/tutorial/${dir}/my_constantin.json

# 结果输出：生成的最优参数
opt constant for gene algrithom is
[['DelayQueueLatencyvbop’, 78], ['DelayQueueLatencypbop’, 80]]  fitness
54660
```

运行编译结果

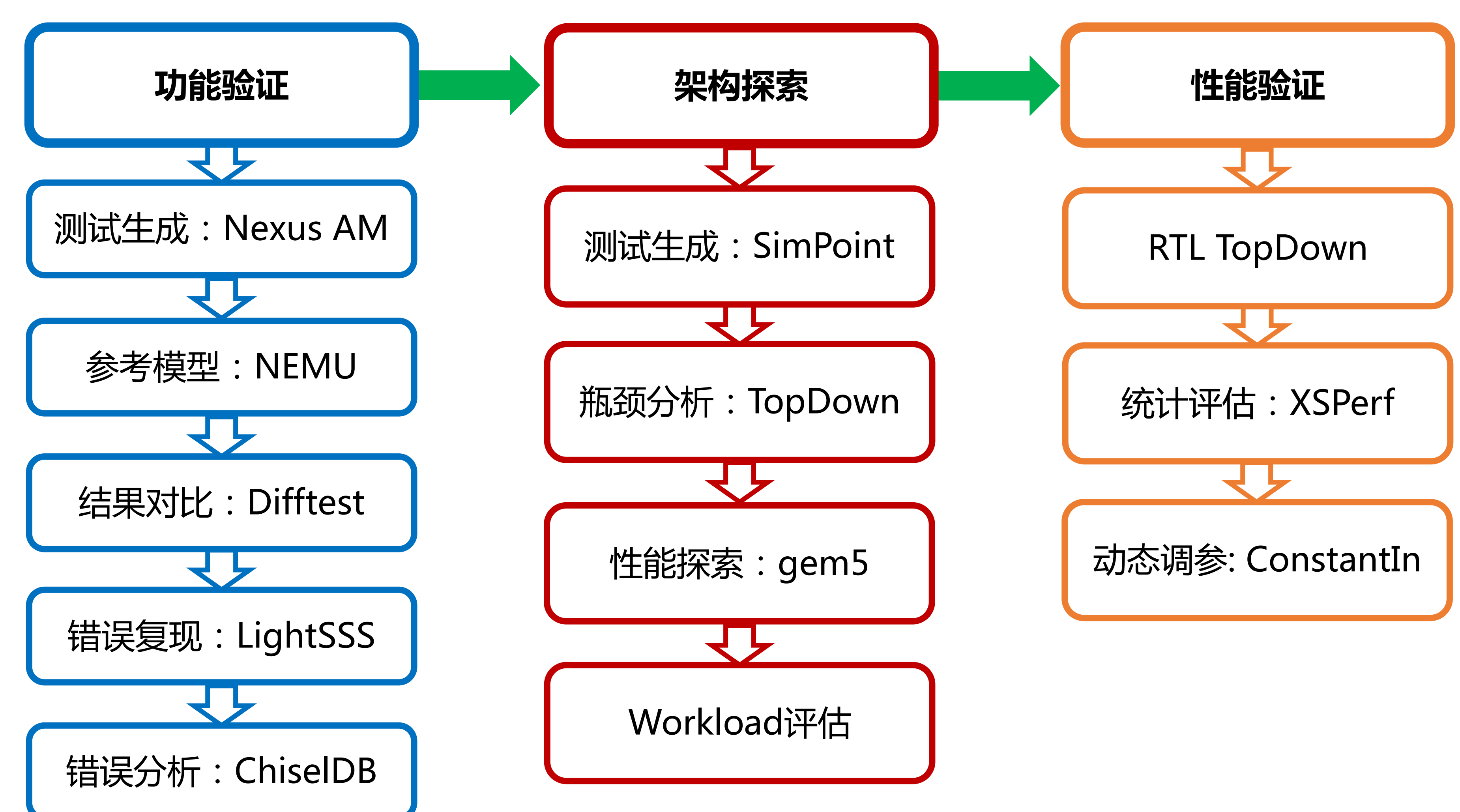
- 回到最开始编译 emu 的 shell 界面
- 可以运行刚刚编译完成的 emu 仿真

```
$ cd $NOOP_HOME
$ ./build/emu --help          (执行时自动使用 EMU_THREADS 线程)
# 部分参数：
    -i                        用到的 workload
    -C / -I / -W              最大周期数 / 最大指令数 / 预热指令数
    --diff=PATH / --no-diff   与 NEMU 做差分测试 / 关闭差分测试
```

示例：使用 tab 键进行命令补全

```
$ ./build/emu -i ../tutorial/p1-basic-func/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



北京开源芯片研究院
BEIJING INSTITUTE OF OPEN SOURCE CHIP

欢迎加入我们！

香山技术讨论 QQ 群

香山 tutorial 文档主页

香山技术讨论微信四群