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

中国开放指令生态（RISC-V）联盟
China RISC-V Alliance

香山开源处理器社区
XiangShan Open-Source Processor Community

XiangShan: An Open-Source High-Performance RISC-V Processor and Infrastructure for Architecture Research

The XiangShan Team

MICRO'24@Austin, USA

November 02, 2024

What we will cover in this tutorial

- **Introduce of XiangShan (13:00 – 13:40, 40 minutes)**

- **CPU Microarchitecture (13:50 – 15:00, 70 minutes)**
  - *Design and implementation – How to implement novel ideas on XiangShan*
  - Frontend: branch prediction and instruction fetch
  - Backend: out-of-order scheduler, execution units
  - Load/Store Unit: LSQ, pipelines, TLBs, data caches
  - L2/L3 caches and prefetchers

- **Development workflows (15:30 – 17:00, 90 minutes)**
  - *Introduction of their usages – How to develop on XiangShan with MinJie*
  - Simulation and Debugging
  - Research Demo

In this tutorial, we will

- Provide access to cloud servers
- Prepare the XiangShan development environment for you
- Go through the development workflows including:

Simulation

- Server login
- Environment
- RTL Generation
- RTL Simulation

……

Func. Verification

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

……

Perf. Verification

- Checkpoint
- Gem5
- Top-Down
- Constantin

……

- ***Required: machines with network and SSH***

Demo Instructions

- **Shell commands** are highlighted in **brown box** with prefix $

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

- **Description and notes** are presented in **grey box** with prefix #

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

Let's Start!

Prerequisites

- Login to the provided cloud server
  - Windows (Windows Terminal / PowerShell is recommended)

```
PS > ssh guest@tutorial.xiangshan.cc
# Password: xiangshan-2024
```

  - Linux / Mac

```
$ ssh guest@tutorial.xiangshan.cc
# Password: xiangshan-2024
$ guest@xiangshan-tutorial:~$ pwd
/home/guest
```

For **offline users**, please refer to https://github.com/OpenXiangShan/xs-env/tree/tutorial-new

Prerequisites

```
# Copy tutorial environment to your dir based on your name
$ cp -r /opt/xs-env ~/<YOUR_NAME>

# Enter your dir
$ cd ~/<YOUR_NAME>

# Set up environment variables
# DO IT AGAIN when opening a new terminal
$ 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
```

Prerequisites

```
# Project Structure
$ tree -d -L 1
.
├── DRAMsim3
├── gem5
├── NEMU
├── nexus-am
├── NutShell
├── tl-test-new
├── tutorial
└── XiangShan

# Enter XiangShan directory
$ cd XiangShan
```

Chisel Compilation

- **Compile RTL and build simulator with Verilator**

```
$ make emu -j4
```

Options:

CONFIG=MinimalConfig	*Configuration of XiangShan*
// EMU_THREADS=4	*Simulation threads*
// EMU_TRACE=1	*Enable waveform dump*
// WITH_DRAMSIM=1	*Enable DRAMSim3 for DRAM simulation*
// WITH_CHISELDB = 1	*Enable ChiselDB feature*
// WITH_CONSTANTIN = 1	*Enable Constantin feature*

- **Compilation might take ~20 mins**

Open Another Terminal

```
# login to the cloud server again
$ ssh guest@tutorial.xiangshan.cc
# Password: xiangshan-2024

# Enter your dir and set up
$ cd ~/<YOUR_NAME> && source env.sh

# Enter tutorial for following operations
$ cd tutorial
```

Run RTL Simulation with Verilator

- **After building, we can run the simulator**

```
# run pre-built simulator
$ cd p1-basic-func
$ ./emu -i hello.bin --no-diff 2>hello.err

# Some key options:
    -i                              # Workload to run
    -C / -I                         # Max cycles  /Max Insts
    --diff=PATH / --no-diff         # Path of Reference Model / disable difftest
```

Great!

We have learned the basic simulation process of Xiangshan.

Once we get the RTL simulator ready, what's next?

How can we

- **generate desired workloads**
- **find and fix functional bugs**
- **do performance analysis**
- **do research on micro-architecture**

MinJie Functional Verification

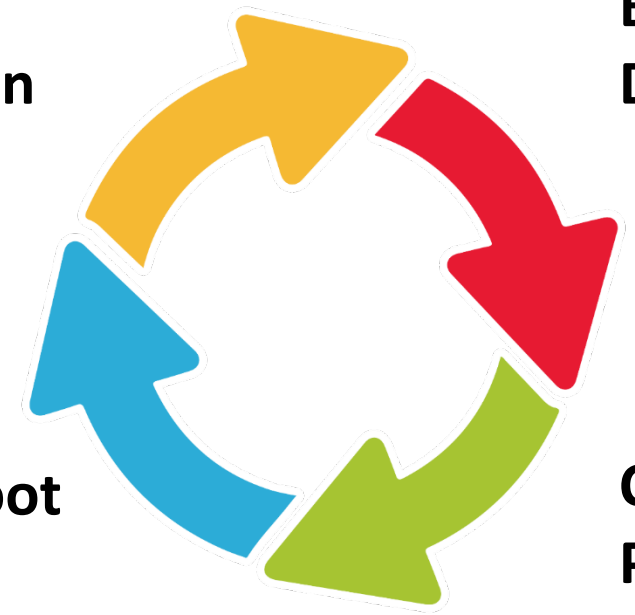
Nexus-AM

Test Generation

Bug Detection

Difftest

NEMU



Waveform

ChiselDB

Troubleshoot & Fix bugs

Context Preservation

LightSSS

Nexus-AM: Bare Metal Runtime

- **Purpose**
  - Generate workloads **agilely** without an OS
  - Provide runtime framework for **bare metal machines** like XiangShan
- **We present a bare metal runtime called Nexus-AM**
  - Light-weight, easy to use
  - Implement basic **system call** interfaces and exception handlers
  - Support multiple **ISAs and configurations**

Nexus-AM

- **Hands-on:** build Coremark workload
- **Showcase**

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

```
# cd $AM_HOME/apps/coremark
# make ARCH=riscv64-xs \
  ITERATIONS=1 LINUX_GNU_TOOLCHAIN=1 TOTAL_DATA_SIZE=400
# ls -l build
#    coremark-riscv64-xs.bin    Binary image of the program
#    coremark-riscv64-xs.elf    ELF of the program
#    coremark-riscv64-xs.txt    Disassembly of the program
```

MinJie Functional Verification

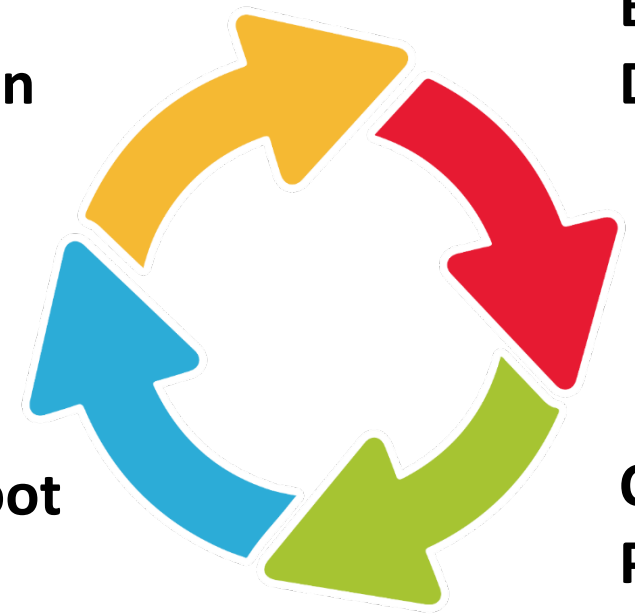
Nexus-AM

Test Generation

Bug Detection

Difftest

NEMU



Waveform

ChiselDB

Troubleshoot & Fix bugs

Context Preservation

LightSSS

NEMU: ISA REF

- **Purpose**
  - Golden model for verification
  - Simple yet high performance
- **We present NEMU**
  - An **instruction set simulator**, similar to Spike
  - Through **optimization techniques**, achieve performance similar to QEMU.
  - **A set of APIs** is provided to assist XiangShan in comparing and verifying the **architectural states**

NEMU

- **Showcase**

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

```
# cd $NEMU_HOME
# make clean
# make riscv64-xs_defconfig
# make -j      build NEMU as the bare metal machine, can run the Coremark from the previous step
# make clean-softfloat
# make riscv64-xs-ref_defconfig
# make -j        build NEMU as the reference model for XiangShan
```

NEMU

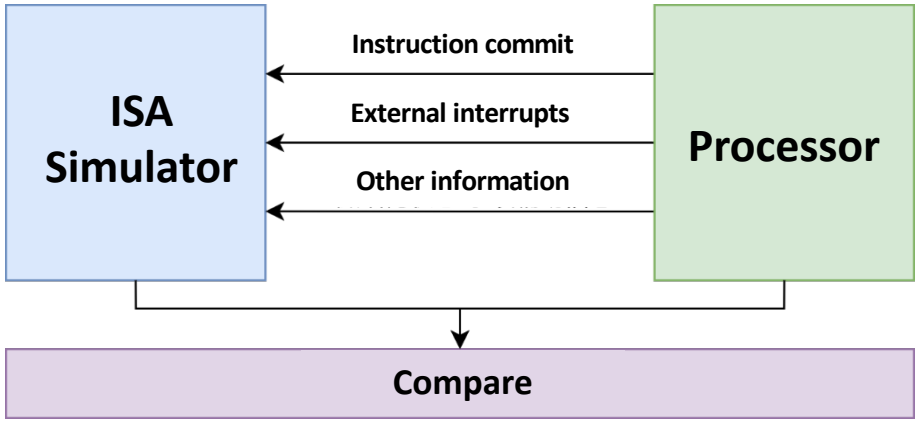
- **Hands-on:** run Coremark workload on NEMU
- **Showcase**

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

```
# cd $NEMU_HOME
# ./build/riscv64-nemu-interpreter -b \                    run in batch mode, faster
$AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin       set workspace to Coremark
```

Difftest: ISA Co-simulation Framework

- **Basic flows**
  - Instructions commit/other states update
  - The simulator executes the same instructions
  - Compare the architectural states between DUT and REF
  - Abort or continue
- **Provided as verification infrastructure for processors**
  - APIs for HDLs such as Chisel/Verilog
  - RTL simulators such as Verilator, VCS
  - RISC-V ISS such as **NEMU**, **Spike**
- **SMP-Difftest: co-simulation on an SMP processor**
  - SMP Linux kernel and multi-threading programs
  - Online checking of cache coherency and memory consistency



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

- **Hands-on:** run Coremark workload on XiangShan, difftest with NEMU
- **Showcase**

```
# Running Coremark takes about 1 minutes
$ bash run-emu.sh
```

```
# ./emu \
-i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \   use coremark.bin
--diff $NEMU_HOME/build/riscv64-nemu-interpreter-so \       difftest with NEMU
2>perf.out     redirect stderr to file
```

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

- **Hands-on:**
  - We injected a bug in a pre-built XiangShan processor
  - Now, let's trigger it by Difftest
- **Showcase**

```
$ 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    # difftest with NEMU
```

Difftest

- Difftest will report the error once failed
  - Dump info like PC, GPRs, etc.

```
============== Commit Group Trace (Core 0) ==============
commit group [00]: pc 0080000872 cmtcnt 9
commit group [01]: pc 0080000838 cmtcnt 9
commit group [02]: pc 00800015de cmtcnt 10
commit group [03]: pc 0080001614 cmtcnt 13
commit group [04]: pc 008000163e cmtcnt 19
commit group [05]: pc 0080001aaa cmtcnt 11 <--
commit group [06]: pc 00800028d8 cmtcnt 8
commit group [07]: pc 0080002a96 cmtcnt 8
commit group [08]: pc 0080002aa6 cmtcnt 10
commit group [09]: pc 0080000162 cmtcnt 8
commit group [10]: pc 0080000838 cmtcnt 10
commit group [11]: pc 00800015b0 cmtcnt 9
commit group [12]: pc 008000083e cmtcnt 9
commit group [13]: pc 008000082a cmtcnt 9
commit group [14]: pc 0080000860 cmtcnt 9
commit group [15]: pc 0080000830 cmtcnt 9
```

```
==============  REF Regs  ==============
  $0: 0x0000000000000000   ra: 0x00000000800015da   sp: 0x0000000008000bf00   gp: 0x0000000000000000
  tp: 0x0000000000000000   t0: 0x0000000000000000   t1: 0x0000000000000001   t2: 0x0000000000000009
  s0: 0x0000000000000000   s1: 0x0000000000000000   a0: 0x0000000000000085   a1: 0x000000008000bf18
  a2: 0x0000000000000002   a3: 0x0000000000000085   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: 0x0000000008000bda1   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: 0x0000000080001aa2 mstatus: 0x8000000a00006000 mcause: 0x0000000000000000 mepc: 0x0000000000000000
                       sstatus: 0x8000000200006000 scause: 0x0000000000000000 sepc: 0x0000000000000000
```

```
privilegeMode: 3
     a1 different at pc = 0x0080001aaa, right= 0x000000008000bf18, wrong = 0x000000008000bf20
     a3 different at pc = 0x0080001aaa, right= 0x0000000000000085, wrong = 0x0000000000000000
Core 0: ABORT at pc = 0x80000f24
Core-0 instrCnt = 1,296, cycleCnt = 11,810, IPC = 0.109738
Seed=0 Guest cycle spent: 11,813 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 5,376ms
```

Difftest

- **Hands-on:** Dump waveform after Difftest detects the bug
- **Showcase**

```
$ bash run-emu-dumpwave.sh
$ ls $NOOP_HOME/build/*.vcd -lh   # There should be a .vcd waveform
```

```
# ./emu-bug \
-i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \
--diff $NEMU_HOME/build/riscv64-nemu-interpreter-so \
-b 11000 \      the begin cycle of the wave, you may find XiangShan trap at 12000 cycles in previous step
-e 13000 \      the end cycle of the wave
--dump-wave \   set this option to dump wave, you will find *.vcd on $NOOP_HOME/build
```

MinJie Functional Verification

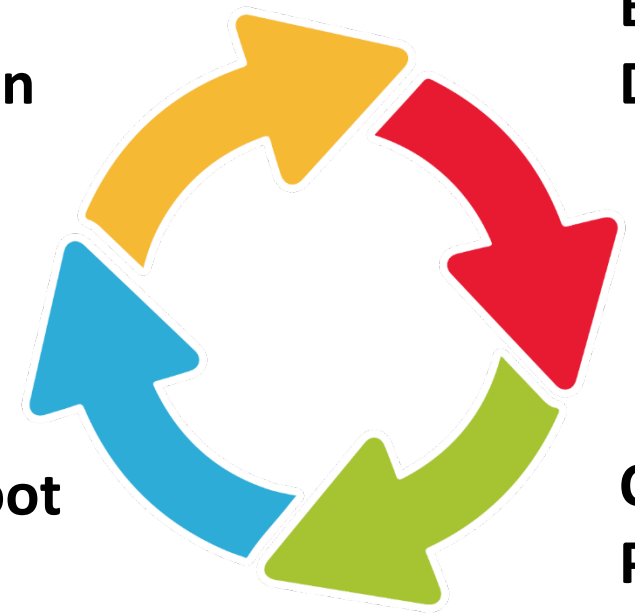
Nexus-AM

Test Generation

Bug Detection

Difftest

NEMU



Waveform

ChiselDB

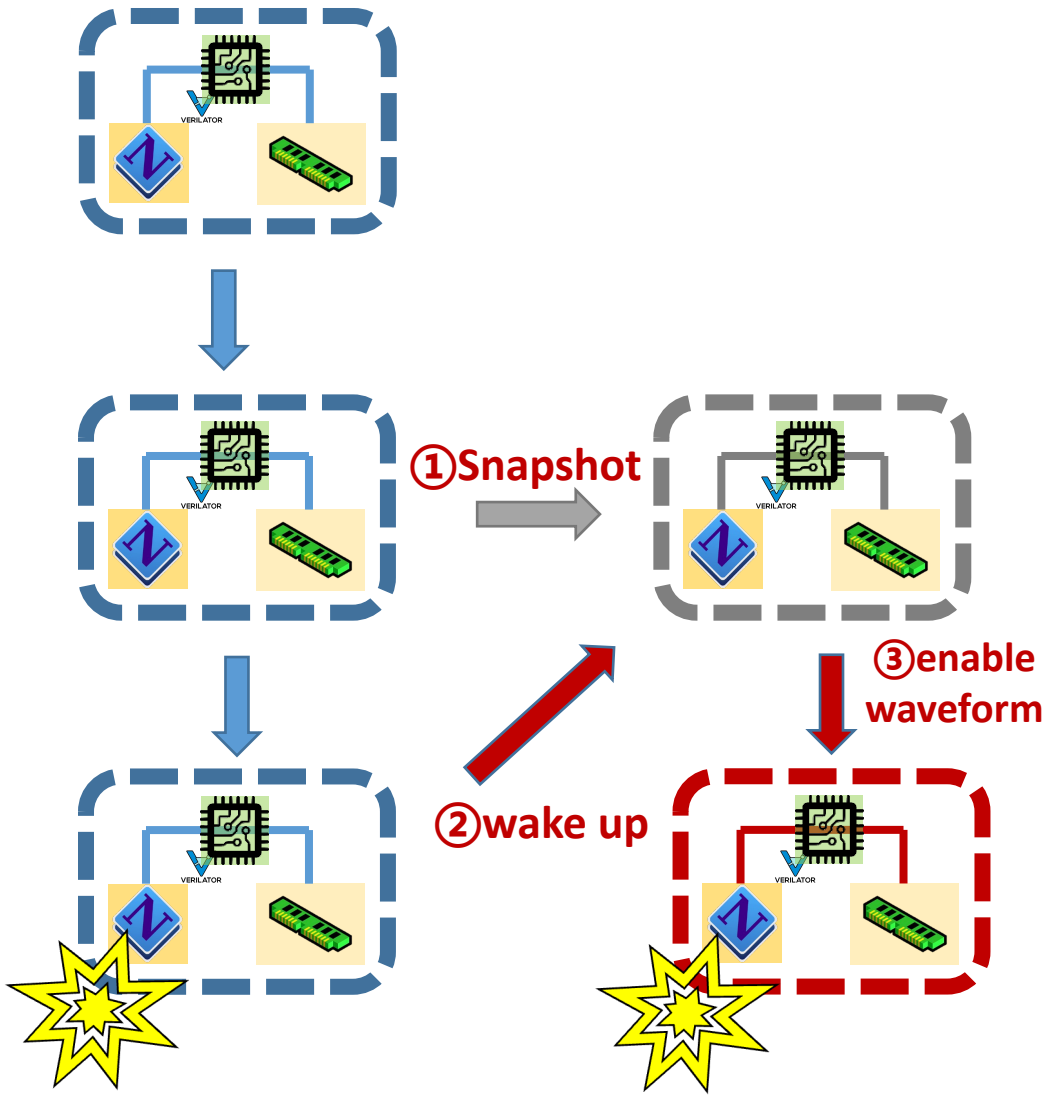
Troubleshoot & Fix bugs

Context Preservation

LightSSS

LightSSS: Light-weight Simulation SnapShot

- **Re-run simulation is time-consuming!**
  - **Snapshot is the way out**
- **LightSSS: snapshots of the process with fork()**
  - Copy-on-write on Linux Kernel
- **Advantage 1: Good portability and scalability**
  - Snapshots for any external models (such as C++)
  - No need to understand details of external models
- **Advantage 2: Low overhead of snapshots**
  - ~500us for taking a snapshot
  - Far less overhead than RTL snapshots by Verilator



LightSSS

- **Hands-on:** use lightSSS to dump waveform
- **Showcase**

```
$ 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
```

No longer need to use complex parameters

LightSSS

- LightSSS is working if you see:

```
privilegeMode: 3
     a1 different at pc = 0x0080001aaa, right= 0x000000008000bf18, wrong = 0x000000008000bf20
     a3 different at pc = 0x0080001aaa, right= 0x0000000000000085, wrong = 0x0000000000000000
[FORK_INFO pid(29770)] the oldest checkpoint start to dump wave and dump nemu log...
[FORK_INFO pid(29770)] dump wave to /home/guest/liyanqin-tmp/XiangShan/build/2024-11-02@23:40:18_9208.vcd...
ing CoreMark for 1 iterations

============== Commit Group Trace (Core 0) ==============
commit group [00]: pc 0080000872 cmtcnt 9
commit group [01]: pc 0080000838 cmtcnt 9
commit group [02]: pc 00800015de cmtcnt 10
commit group [03]: pc 0080001614 cmtcnt 13
commit group [04]: pc 008000163e cmtcnt 19
commit group [05]: pc 0080001aaa cmtcnt 11 <--
```

- After that, simulation will start again from the existing latest snapshot

MinJie Functional Verification

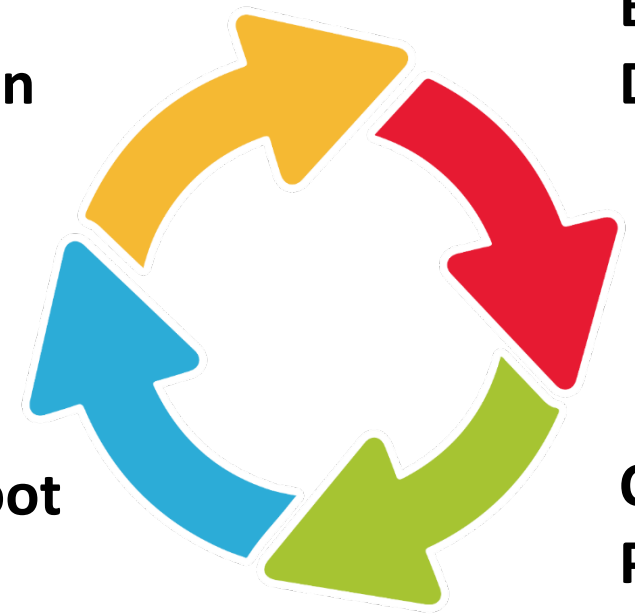
Nexus-AM

Test Generation

Bug Detection

Difftest

NEMU



Waveform

ChiselDB

Troubleshoot & Fix bugs

Context Preservation

LightSSS

ChiselDB: Debug-friendly Structured Database

- **Motivation:**
  - Waveforms are **large** in size and **hard to apply** further analysis
  - Need to analyze structured data like memory transaction trace
- **We present ChiselDB**
- **Highlights:**
  - Inserting probes between module interfaces in hardware
  - DPI-C: Using C++ function in Chisel code to transfer data
  - Persist in database, SQL queries supported

ChiselDB

- **Design source code:** *XiangShan/utility/src/main/scala/utility/ChiselDB.scala*
- **Usage:** Create 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

- **Usage:** Register probes

```
/* 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

- **Example:** *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

- **TL-Log：Persistence Transactions/Database**
  - Record bus transactions to SQLite3 database using ChiselDB
  - Support offline bug detect and performance analysis

Time	Name	Channel	Opcode	Permission	State	SourceID	SinkID	Address	Data
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	fa8437839043903 06090063843c23 84382300093783 378300093023c3bd
116854147	L2_L1I_0	D	GrantData	Cap	toT	1	16	803d3680	03e32e07bb030089 f0ef8526c881f49b b60300893783bbdf 855a010925832e07
123191840	L2_L1I_0	C	ReleaseData	Shrink	TtON	0	0	803d3680	fa8437839043903 06090063843c23 84382300093783 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 b60300893783bbdf 855a010925832e07
123191848	L2_L1I_0	D	ReleaseAck	Cap	toT	0	31	803d3680	0000000020ab05b 0000000020fab45b 0000000020fab85b 0000000020fabc5b
123191851	L3_L2_0	C	ReleaseData	Shrink	TtoN	31	0	803d3680	fa8437839043903 06090063843c23 84382300093783 378300093023c3bd
123191852	L3_L2_0	C	ReleaseData	Shrink	TtoN	31	0	803d3680	03e32e07bb030089 f0ef8526c881f49b b60300893783bbdf 855a010925832e07
123191862	L3_L2_0	D	ReleaseAck	Cap	toT	31	16	803d3680	86a6f7043583dcd5 f0ef856a865a8762 3583b7654481e17 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 001947036775e84a

Example: requests sequence in database that violate cache coherence

ChiselDB

- **Hands-on: Analysis of Cache Coherence Violation**

```
# Inject bug – We set all Release Data (L2->L3) as a constant value
$ cd ../p2-chiseldb && cat cdb_err.patch
```

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

- **Hands-on: Analysis of Cache Coherence Violation**

```
# simulate using pre-built emu (ChiselDB enabled)
$ 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    # set this argument to dump data base
  2>linux.err
```

ChiselDB

- **Hands-on: Analysis of Cache Coherence Violation**
- **Difftest Info**
  - Error Address：0x800419c0

```
The first instruction of core 0 has commited. Difftest enabled.
cacheid=1,idtfr=0,realpaddr=0x800419c0: Refill test failed!
addr: 800419c0
Gold: 00000000000000000000000000000000000080041a70000000080016198000000080041a5000000000000000000000000000000000000000000000000
Core: 000000000abcdef0000000000000000000000000000000000000000000000000000080041a5000000000000000000000000000000000000000000000000
Core 0: ABORT at pc = 0x80013966
Core-0 instrCnt = 17,709, cycleCnt = 50,670, IPC = 0.349497
```

ChiselDB

- **Hands-on: Analysis of Cache Coherence Violation**

```
# Analyze
$ bash chiseldb_step2-analyze.sh
```

```
# 1. sqlite query for all transactions on Eaddr
# 2. use script to parse TLLog
# sqlite3 $NOOP_HOME/build/*.db \
  "select * from TLLOG where ADDRESS=0x800419c0" | \
  sh $NOOP_HOME/scripts/utils/parseTLLog.sh
```

ChiselDB:

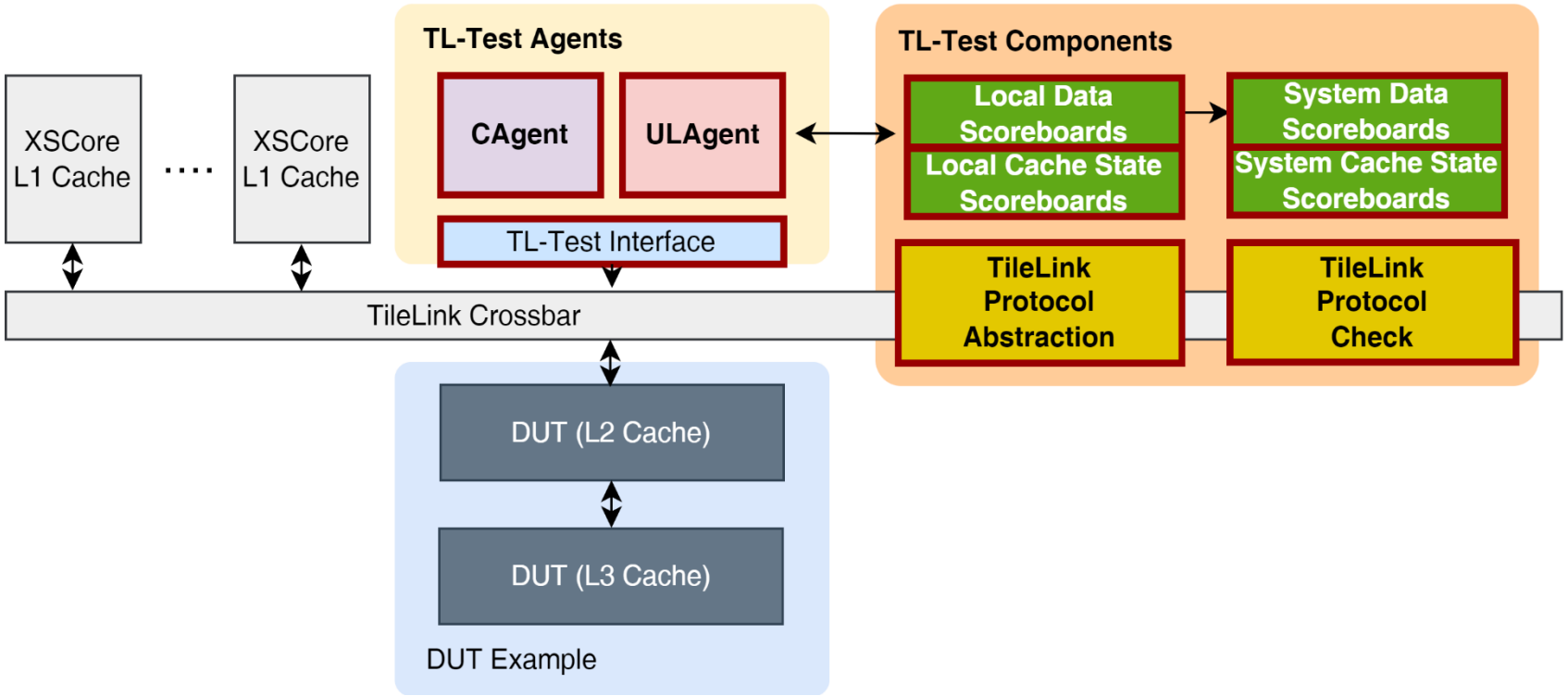
```
# Result: [Time | To_From | Channel | Opcode | Permission | Address | Data]

# Data successfully transferred from L1D to L2
28189 L2_L1D_0 C ReleaseData Shrink TtoN 800419c0
0000000000000000 0000000000000000 0000000080041a70 0000000080016198
# Data successfully transferred from L2 to L3
39637 L3_L2_0 C ProbeAckData Shrink TtoN 800419c0
0000000000000000 0000000000000000 0000000080041a70 0000000080016198


# But when L1D acquires Eaddr again, data loaded from L3 is wrong
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
# So there must be something wrong when L3 records Release Data
```

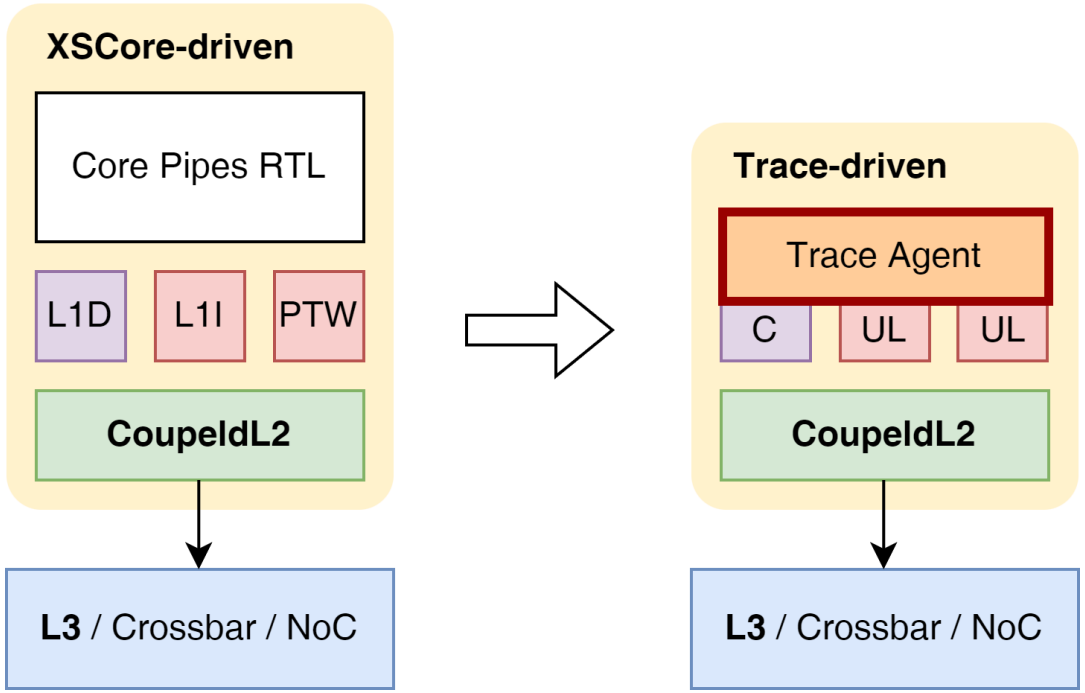
TL-Test: Cache Verification Framework

- **As an infrastructure for cache verification**
  - Support **Tilelink protocol** and **cache coherence** check
  - **Fast test cases generation** and **constrained random test**



TL-Test

- **TL-Trace: Trace-based testbench generation**
  - Convert cache access trace into testbench
  - Fast reproduction of functional bugs and analysis of performance problems



TL-Test

```
# Project Structure
$ cd $TLT_HOME && tree -d -L 1
.
├── assets
├── configs
├── dut
├── main
├── run
├── scripts
└── tutorial

# Enter TL-Test tutorial
$ cd ../tutorial/p3-tl-test
```

TL-Test

- **Hands-on: Analysis of Cache Coherence Violation**

```
# Inject bug – We wrongly shift the Release Data (L1->L2)
$ 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

- **Hands-on: Analysis of Cache Coherence Violation**

```
# We provide a pre-compiled TL-Test
# Run the TL-Test
$ bash tltest_step1_run.sh
```

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

TL-Test

- **Hands-on: Analysis of Cache Coherence Violation**
- TL-Test Info
  - Error Address： 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

- **Hands-on: Analysis of Cache Coherence Violation**

```
# Analyze
$ bash tltest_step2_analyze.sh
```

```
# 1. grep all transactions on Eaddr(0x16000)
# 2. use TL-Log to help debugging

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

TL-Test

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

 # Data successfully transferred from L1D to 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 ...]

 # But when L1D acquires Eaddr again, data loaded from L2 is wrong

[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 ...]

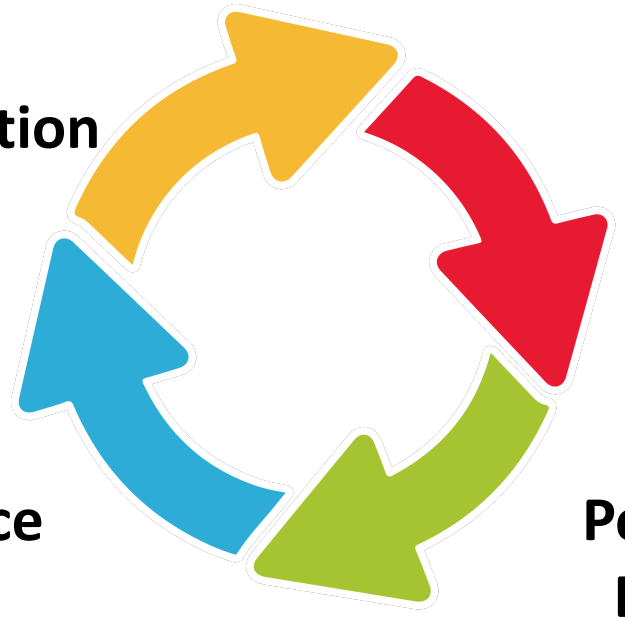
[1740] L2[0].C[0] [fire D] [GrantData toT] source: 0x1, addr: 0x16000, data: [00 75 ...]

 # So there must be something wrong when L2 records Release Data

MinJie Performance Verification

Chisel-based prototypes

Implementation



Run Tests

gem5

VERILATOR

umd-memsys/
DRAMsim3
DRAMsim3: a Cycle-accurate, Thermal-Capable DRAM Simulator

XSPerf

Constantin

Top-Down

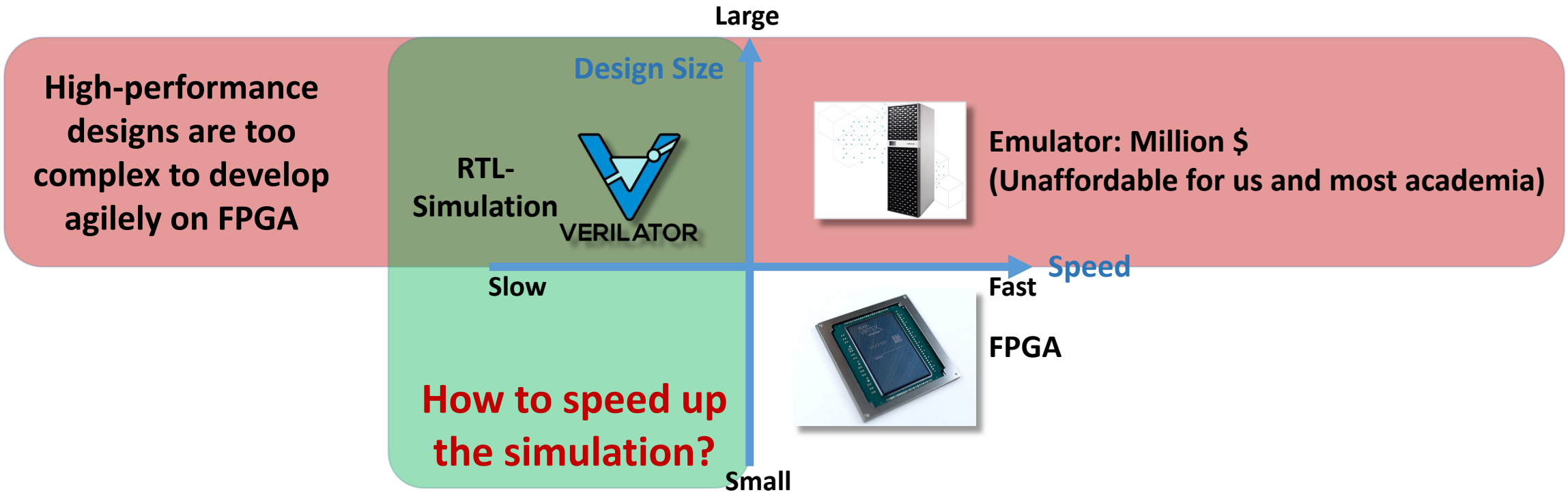
Performance Analysis

Performance Evaluation

RISC-V Checkpoint

SimPoint

Checkpoint: Agile Performance Evaluation

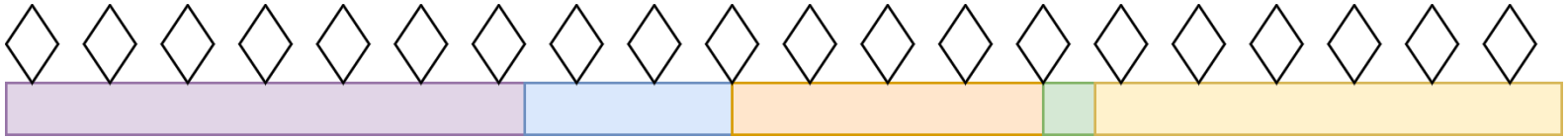


Time for performance modelling of single-core XiangShan on SPEC CPU2006

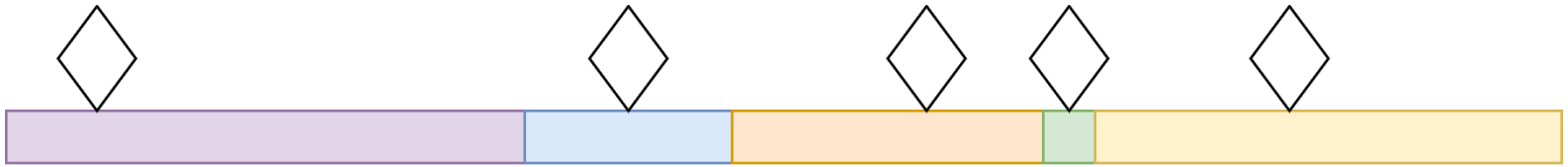
	RTL-simulation	FPGA	RTL-Simulation w/ Checkpoint
Compile	20 minutes	5 hours	20 minutes
Simulation	958 years@2KHz (2K CPS)	7 days@100MHz	5.5 hours with enough x86 servers
			Our Approach

Checkpoint: Agile Performance Evaluation

- Uniform Checkpoint
  - Divide a program to equal segments for parallel simulation with uniform sampling



- SimPoint Checkpoint
  - Select representative segments with cluster analysis and weighted sampling



Checkpoint: SimPoint Checkpoint

- **Step 0: Prepare NEMU environment for SimPoint checkpoint**

```
$ cd ../p4-checkpoint && bash simpoint_step0-prepare.sh
```

```
# simpoint_step0_prepare.sh (~23s)

# cd $NEMU_HOME
# git submodule update --init
# cd $NEMU_HOME/resource/simpoint/simpoint_repo
# make clean && make                # generate simpoint generator binary

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

# cd $NEMU_HOME/resource/gcpt_restore
# rm -rf $XS_PROJECT_ROOT/tutorial/part5-checkpoint/gcpt
# make -C $NEMU_HOME/resource/gcpt_restore/ \     # generate gcpt restorer binary
        O=$XS_PROJECT_ROOT/tutorial/part5-checkpoint/gcpt \  # directory of results
        GCPT_PAYLOAD_PATH=$XS_PROJECT_ROOT/tutorial/part5-checkpoint
/bin/stream_100000.bin
```

Checkpoint: SimPoint Checkpoint

- **Step 1: Execute the workload and collect program behavior**

```
$ bash simpoint_step1-profiling.sh
```

```
# simpoint_step1_profiling.sh (~18s)
# source simpoint_env.sh                    # configure environment variables

# rm -rf $RESULT

# $NEMU ${BBL_PATH}/${workload}.bin \       # specify workload
#     -b                            \       # run with batch mode
#     -D $RESULT                    \       # directory where the checkpoint is generated
#     -C $profiling_result_name     \       # name of this task
#     -w stream                     \       # name of workload
#     --simpoint-profile            \       # is simpoint profiling
#     --cpt-interval ${interval}    \       # simpoint interval instructions: 50000000
#     > >(tee $log/stream-out.txt) 2> >(tee ${log}/stream-err.txt)  # redirect stdout and stderr
```

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 will do profiling while executing workload STREAM at intervals of 50,000,000 instructions.

Checkpoint: SimPoint Checkpoint

- **Step 2: Cluster and obtain multiple representative slices**

```
$ 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 \    # load a frequency vector file
#     -saveSimpoints $CLUSTER/simpoints0                     \    # file to save simpoints
#     -saveSimpointWeights $CLUSTER/weights0                 \    # file to save simpoints 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 ${random1}       \   # random seed for choosing initial k-means centers
#     -seedproj ${random2}     \   # random seed for random linear projection
#     > >(tee $log/${workload}-out.txt) 2> >(tee $log/${workload}-err.txt)
                                  # redirect stdout and stderr
```

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)
```

Compute the information of K-means clustering

Obtain multiple program representative checkpoints

Checkpoint: SimPoint Checkpoint

- **Step 3: Generate corresponding Checkpoints based on clustering results**

```
$ bash simpoint_step3_genspt.sh          # It may take about 2 minutes
```

```
# simpoint_step3_genspt.sh

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

# $NEMU ${BBL_PATH}/${workload}.bin \
#     -b                            \  run with batch mode
#     -D $RESULT                    \  directory where the checkpoint is generated
#     -C checkpoint                 \  name of this task
#     -w stream                     \  name of workload
#     -S $CLUSTER                   \  simpoints results director
#     --cpt-interval $interval      \  simpoint interval instructions: 50,000,000
#     > >(tee $log/stream-out.txt) 2> >(tee $log/stream-err.txt)
                                       redirect stdout and stderr
```

Checkpoint: SimPoint Checkpoint

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

[src/checkpoint/serializer.cpp,100,serializeRegs] Writing int registers to checkpoint memory @[0x80001000, 0x80001100) [0x1000, 0x1100)
[src/checkpoint/serializer.cpp,110,serializeRegs] Writing float registers to checkpoint memory @[0x80001100, 0x80001200) [0x1100, 0x1200)
[src/checkpoint/serializer.cpp,118,serializeRegs] Writing PC: 0xffffffe000720f20 at addr 0x80001200
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x105: 0xffffffe000697cc8
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x106: 0xffffffffffffffff
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x141: 0x200013417e
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x142: 0x8
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x180: 0x800000000000f805d
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x300: 0xa00000022
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x301: 0x800000000014112d
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x302: 0xb109
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x303: 0x222
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x304: 0x22a
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x305: 0x800a0004
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x306: 0xffffffffffffffff
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x340: 0x800abdc0
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x341: 0xffffffe0008d6eb2
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x342: 0x2
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x3a0: 0x1f0800
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x3b0: 0x20028000
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x3b1: 0x2002d000
[src/checkpoint/serializer.cpp,141,serializeRegs] CSR 0x3b2: 0xffffffffffffffff
[src/checkpoint/serializer.cpp,144,serializeRegs] Writing CSR to checkpoint memory @[0x80001300, 0x80009300) [0x1300, 0x9300)
[src/checkpoint/serializer.cpp,152,serializeRegs] Touching Flag: 0xbeef at addr 0x80000f00
[src/checkpoint/serializer.cpp,156,serializeRegs] Record mode flag: 0x1 at addr 0x80000f08
[src/checkpoint/serializer.cpp,160,serializeRegs] Record time: 0x1 at addr 0x80000f10
[src/checkpoint/serializer.cpp,164,serializeRegs] Record time: 0x1 at addr 0x80000f18
[src/checkpoint/serializer.cpp,52,serializePMem] Created tutorial_simpoint/take_cpt/stream_0_0.083333/0/

Opening tutorial_simpoint/take_cpt/stream_0_0.083333/0/_0_.gz as checkpoint output file
[src/checkpoint/serializer.cpp,83,serializePMem] Written 0x7fffffff bytes

[src/checkpoint/serializer.cpp,83,serializePMem] Written 0x7fffffff bytes

[src/checkpoint/serializer.cpp,83,serializePMem] Written 0x7fffffff bytes

[src/checkpoint/serializer.cpp,83,serializePMem] Written 0x7fffffff bytes

[src/checkpoint/serializer.cpp,83,serializePMem] Written 0x4 bytes

[src/checkpoint/serializer.cpp,89,serializePMem] Checkpoint done!
```

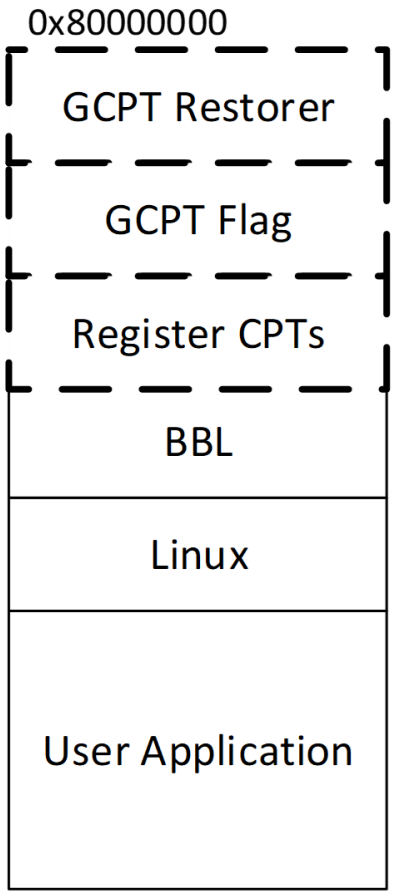
Start making checkpoint

Save int & float registers

Save CSR registers

Save other information

write checkpoints into .gz



Basic format of RISC-V Checkpoint

Checkpoint: SimPoint Checkpoint

- **Step 4: Run the SimPoint checkpoint on NEMU or XiangShan**
- Here we take XiangShan as an example

```
$ bash simpoint_step4_run_xs.sh
```

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

# ./emu \
#    -i `find $RESULT/checkpoint/stream -type f -name "*_.gz" | tail -1` \
                                        get the path of workload
#   --diff $NOOP_HOME/ready-to-run/riscv64-nemu-interpreter-so \
                                        Enable the reference standard design path for difftest
#   --max-cycles=50000 \        Maximum execution instruction count
#   2>simpoint.err
```

Checkpoint: SimPoint Checkpoint

- **Step 5: Provide configuration files for batch running on Gem5/XiangShan**

```
$ python3 simpoint_step5_dumpresult.py
$ ls -l simpoint_result/checkpoint
```

  - Collect profiling/checkpoint information to generate a list file for Gem5 execution

```
# 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
Workload Name | Checkpoint Path | Instructions to Skip (default 0) | Functional Warmup Instructions (default 0) | Detailed Warmup Instructions (default 20M) | Sample Instructions (default 20M)
```

  - Collect cluster information to generate a JSON file for XiangShan execution

```
# cluster.json
{"stream": {"insts": "654228721",
"points": {"7": "0.615385", "0": "0.153846", "11": "0.230769"}}}
Workload Name | Instruction Count | Checkpoint Index | Checkpoint Weight
```

Checkpoints: Uniform Checkpoint

- **Step 0: Prepare NEMU environment for checkpoints**

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

- **Step 1: Generate uniform checkpoints using NEMU**

```
$ bash uniform_cpt.sh
```

- **Step 2: Run uniform checkpoint on NEMU or XiangShan**
- Taking NEMU as an example:

```
$ bash uniform_run_nemu.sh
```

XSPerf: Simulation Performance Collection

- **Purpose**
  - Collect simulation performance data under different requirements
- **XSPerf**
  - Multiple types for various fine-grained performance analysis
    - **Accumulation**: basic accumulator counter (log with stderr)
    - **Histogram**: count the distribution (log with stderr)
    - **Rolling**: rolling curve collection and visualization (ChiselDB)

XSPerf: Accumulate & Histogram

- **Example**

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: Accumulate & Histogram

- **Usage**
  - Add *XSPerfAccumulate('name', signal)* or *XSPerfHistogram('name', signal)*
  - **By default**, it is printed after simulation finishes
  - Set *DebugOptions(EnablePerfDebug = false)* to turn off
- **Showcase**

```
$ cd ../p5-xs-perf && bash xs-perf-log.sh | head -n 20
```

```
# print the log result of p1-basic-func
# cat ${XS_PROJECT_ROOT}/tutorial/p1-basic-func/perf.err
```

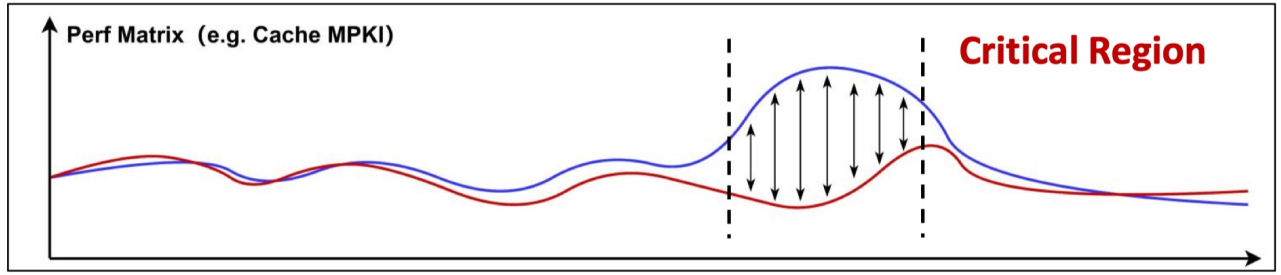
XSPerf: Accumulate & Histogram

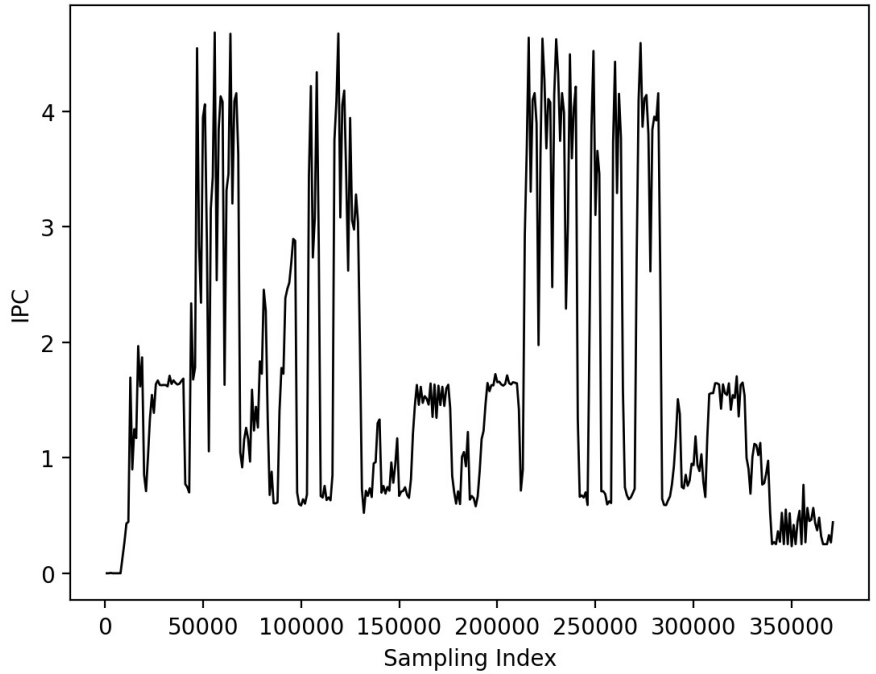
```
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.ctrlBlock.decode: utilization,                1183
[PERF ][time=              2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.bpu.predictors.components_3.tables_0.wrbypass: wrbypass_hit,                    0
[PERF ][time=              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
```

Find more parsing scripts at https://github.com/OpenXiangShan/env-scripts/blob/main/perf/perf.py

XSPerf: Rolling

- **Advantages**
  - Visualization of performance analysis
  - Detailed performance differences among segments
  - Help identify parameter-sensitive critical regions





XSPerf: Rolling

- **Usage**
  - Add *XSPerfRolling('name', perfCnt, granularity, clock, reset)* wherever you want
  - Compile with *WITH_CHISELDB=1 WITH_ROLLINGDB=1* , run with *--dump-db*
- **Showcase**

```
# Build emu with rolling (time consuming, use pre-built emu instead)
# 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/p5-xs-perf/xs-perf-rolling.db
```

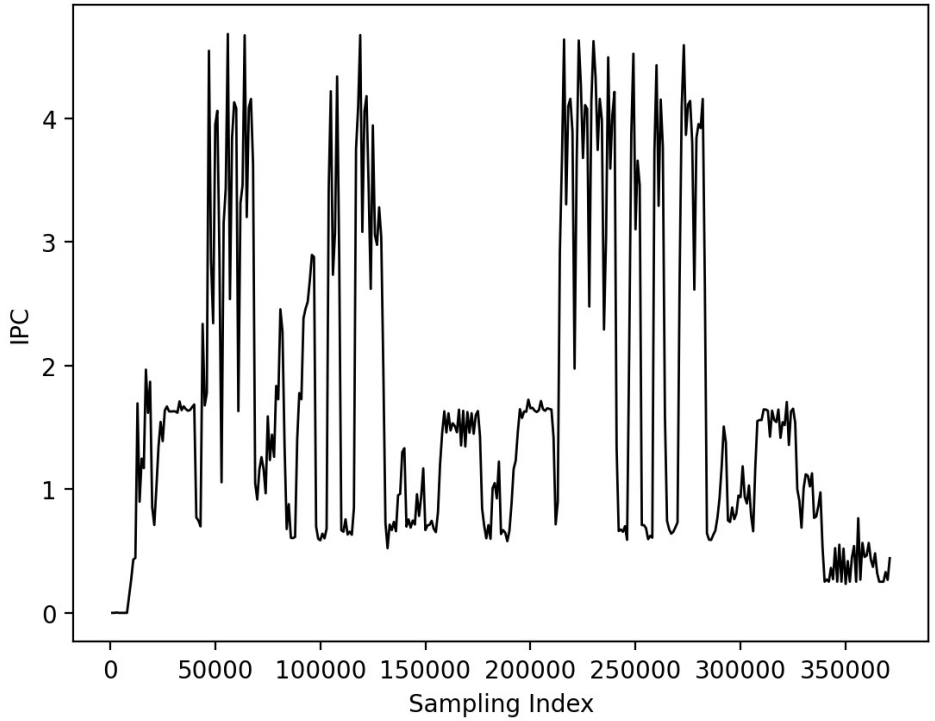
XSPerf: Rolling

- **Hands-on:** check the results and plot the curve of performance segments

```
# tutorial/p5-xs-perf/
$ bash xs-perf-rolling.sh
```

```
# cd ${NOOP_HOME}/scripts/rolling
# python3 rollingplot.py
${XS_PROJECT_ROOT}/tutorial/p6-xs-
perf/XsPerfRolling-test.db ipc
# ls ${NOOP_HOME}/scripts/rolling/results

perf.png
```



rolling curve example

Top-Down: Method for Performance Analysis

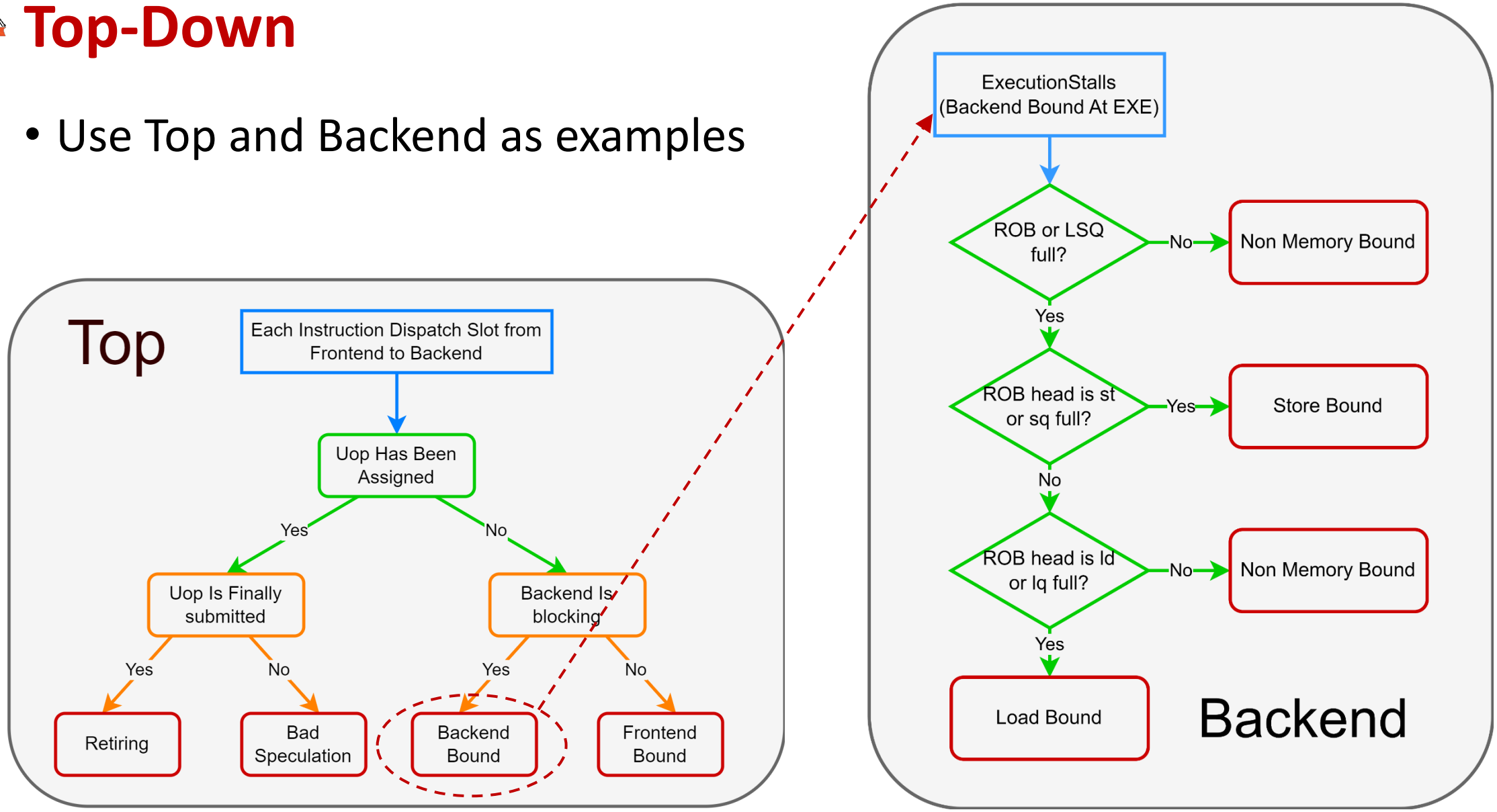
- **Purpose**
  - Organize scattered performance events in a **hierarchical manner**
  - **Accurately** calculate one event's impact on processor performance

- We apply the **Top-Down** approach to XS-Gem5 and XiangShan
  - Complete **targeted optimization adaptation** for RISC-V instruction set
  - Optimize **performance counters** according to XiangShan microarchitecture
  - Further refine the **hierarchical design** of Top-Down model
  - Without missing or duplicating any events
  - Without assuming the blocking cycles of performance events in advance

Yasin, Ahmad. "A Top-Down Method for Performance Analysis and Counters Architecture." 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014, pp. 35–44, https://doi.org/10.1109/ISPASS.2014.6844459.

Top-Down

- Use Top and Backend as examples



Top-Down in RTL

- Setting Performance Counters in RTL Code

```
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)))
)
```

- Do simulation and collect performance counter data

Top-Down in RTL

- Analyze the performance counter data through Top-Down method

```
XiangShan/scripts/top-down/configs.py


xs_coarse_rename_map = {
  'OverrideBubble': 'MergeFrontend',
  'FtqFullStall': 'MergeFrontend',
  'IntDqStall': 'MergeCoreDQStall',  # attribute perf-counters to Top-Down hierarchy
  'FpDqStall': 'MergeCoreDQStall’
}
```

- Obtain analysis results and make targeted optimizations

Top-Down in RTL

- **Hands-on:** analyze the congestion causes in RTL using the Top-Down tool

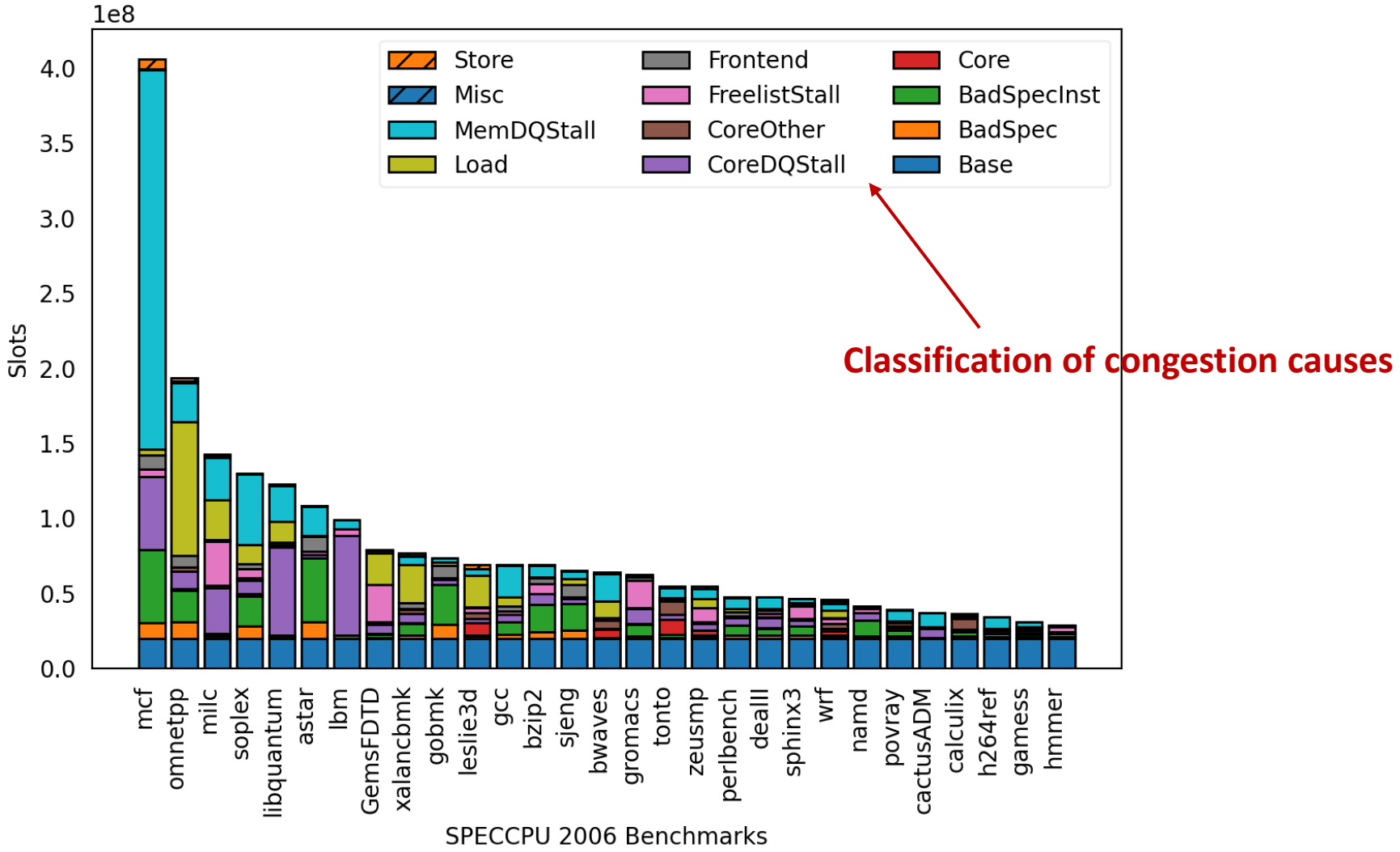
```
#run the Top-Down analysis scripts (it takes ~2min)
$ bash rtl-top-down.sh
```

```
# cd ${NOOP_HOME}/scripts/top-down && \
  python3 top_down.py \
  -s /opt/SPEC06_EmuTasks_topdown \    #path of performance counter results
  -j /opt/SPEC06_EmuTasks_topdown.json #json file of checkpoints configuration

# ls ${NOOP_HOME}/scripts/top-down/results
result.png  results.csv  results-weighted.csv #output of visual analysis results
```

Top-Down in RTL

- Result Example: Top-Down visual analysis results



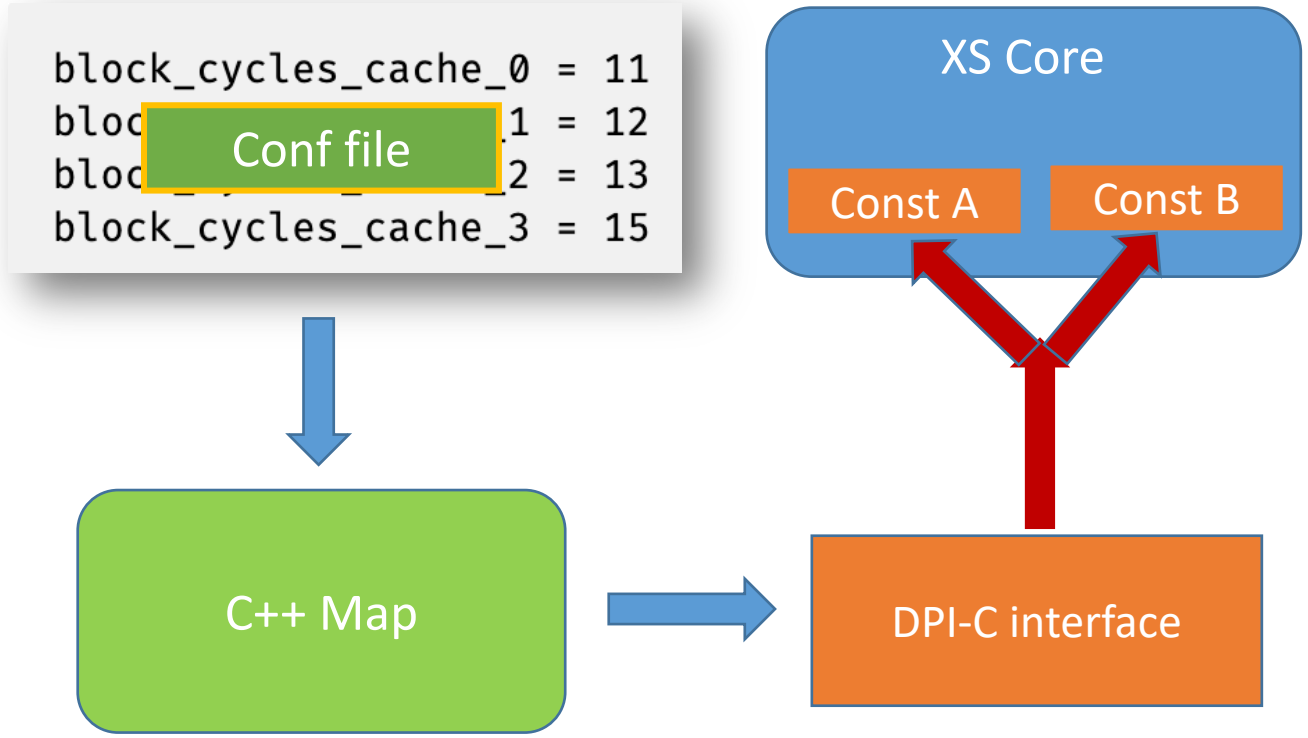
ConstantIn

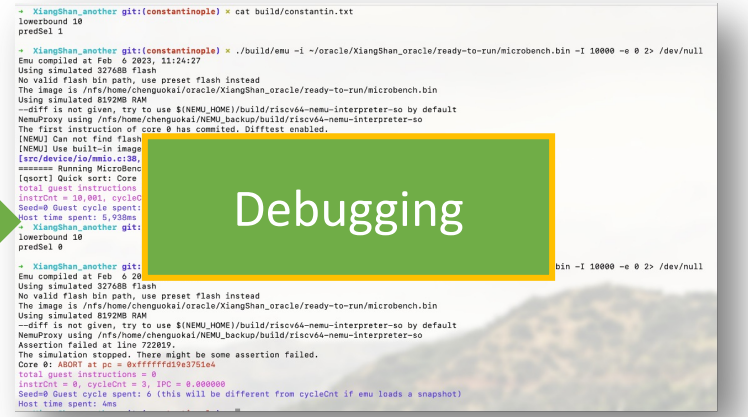
- **Motivation**
  - Want to test the performance under different parameters
  - But re-compilation is time consuming
  - Can we change parameters (constants) at runtime?

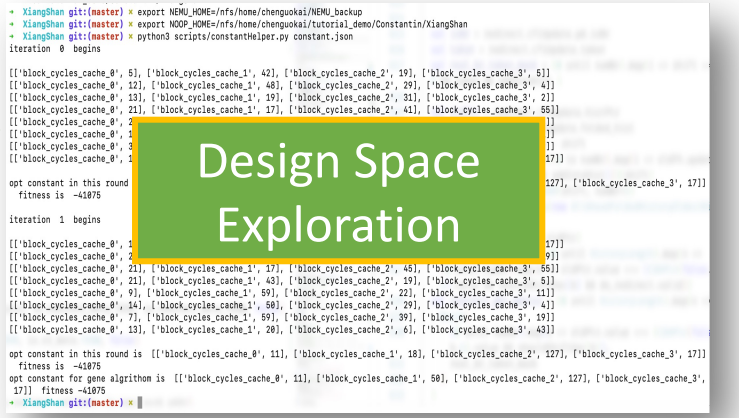
- **ConstantIn**
  - DPI-C: Using C++ function in Chisel code (through Verilog using BlackBox)
  - Signal values configured at runtime rather than compile time

ConstantIn

- Constants can be changed when initializing
- And remain unchanged during runtime







ConstantIn

- **Usage:** Create signals

```
# XiangShan/coupledL2/src/main/scala/coupledL2/prefetch/TemporalPrefetch.scala
```

```
139   require(cacheParams.hartIds.size == 1)
140   val hartid = cacheParams.hartIds.head
141   // 0 / 1: whether to enable temporal prefetcher
142   private val enableTP = WireInit(Constantin.createRecord("enableTP"+hartid.toString, initValue = 1.U))
143   // 0 ~ N: throttle cycles for each prefetch request
144   private val tpThrottleCycles = WireInit(Constantin.createRecord("tp_throttleCycles"+hartid.toString, initValue = 4.U(3.W)))
145   // 0 / 1: whether request to set as trigger on meta hit
146   private val hitAsTrigger = WireInit(Constantin.createRecord("tp_hitAsTrigger"+hartid.toString, initValue = 1.U))
147   // 1 ~ triggerQueueDepth: enqueue threshold for triggerQueue
148   private val triggerThres = WireInit(Constantin.createRecord("tp_triggerThres"+hartid.toString, initValue = 1.U(3.W)))
149   // 1 ~ tpEntryMaxLen: record threshold for recorder and sender (storage size will not be affected)
150   private val recordThres = WireInit(Constantin.createRecord("tp_recordThres"+hartid.toString, initValue = tpEntryMaxLen.U))
151   // 0 / 1: whether to train on vaddr
152   private val trainOnVaddr = WireInit(Constantin.createRecord("tp_trainOnVaddr"+hartid.toString, initValue = 0.U))
153   // 0 / 1: whether to eliminate L1 prefetch request training
154   private val trainOnL1PF = WireInit(Constantin.createRecord("tp_trainOnL1PF"+hartid.toString, initValue = 0.U))
```

ConstantIn

- **Hands-on:** Pass constant via standard input stream
  - **By default**, set constant via *${NOOP_HOME}/build/constantin.txt*
  - **For better demonstration**, it is changed to stdin (corresponding modifications can refer to p6-constantin/*utility.patch*)

```
$ cd ../p6-constantin
```

```
# bash step0-build.sh     # Build emu with constantin (time consuming, use pre-built emu instead)
```

```
$ bash step1-basic.sh     # Run emu and pass constant
please input total constant number
2
please input each constant ([constant name] [value])
DelayQueueLatencyvbop 175
DelayQueueLatencypbop 175
```

ConstantIn

- **Feature:** *AutoSolving* for automatically finding better constant values
- **Process**
  - Enable *AutoSolving* & prepare signals you need
  - Compile the emu with *WITH_CONSTANTIN=1*
  - Run basic demonstration
  - Set parameters for *AutoSolving* in a configuration file
  - Automatically find

ConstantIn

- **Feature:** *AutoSolving* for automatically finding better constant values

```
# Automatical parameter solver configuration file
$ cat my_constantin.json
```

- **Parameters defined**
  - properties of signals
  - target of *AutoSolving*
  - parameters of genetic algorithm
  - parameters of XS simulator

```
{
    "constants": [
        {
            "name": "DelayQueueLatencyvbop",
            "width": 8,
            "guide": 256,
            "init": 177
        },
        {
            "name": "DelayQueueLatencypbop",
            "width": 8,
            "guide": 256,
            "init": 242
        }
    ],
    "opt_target": [
        {"l2cache.topDown: L2MissMatch,": {"policy" :"min", "baseline" :409} },
        {"backend.ctrlBlock.rob: clock_cycle,": {"policy" :"min", "baseline" :217759} }
    ],

    "population_num": 4,
    "iteration_num": 2,
    "crossover_rate": 50,
    "mutation_rate": 50,

    "emu_threads": 8,
    "concurrent_emu": 4,
    "max_instr": 100000,
    "seed": 3888,
    "work_load": "${XS_PROJECT_ROOT}/tutorial/p7-constantin/maprobe-riscv64-xs.bin"
}
```

ConstantIn

- **Hands-on:** *AutoSolving* example

```
# Run auto solver, output will be the optimal constant currently found
$ bash step2-solve.sh
```

```
#  ~2min
# The solver generates optimal parameters
opt constant in this round is  [['DelayQueueLatencyvbop', 8], ['DelayQueueLatencypbop', 72]]  fitness is  54674
opt constant for gene algrithom is  [['DelayQueueLatencyvbop', 9], ['DelayQueueLatencypbop', 73]]  fitness 54674
```

XS-Gem5: Architecture Performance Evaluation Tool

- **Purpose**
  - Efficiently iterate on microarchitecture features
  - Conduct fast end-to-end performance evaluations
- **XS-Gem5**
  - Align with XiangShan as much as possible based on Gem5 O3CPU
  - Support Difftest and Checkpoint
  - Add scripts for XiangShan performance analysis

XS-Gem5

- **Hands-on:** compile and build XS-Gem5

```
$ cd ../p7-xs-gem5 # Enter XS-Gem5 tutorial directory
$ bash 0-gem5_prepare.sh # ~8min, time consuming, use pre-built emu instead
$ cd /home/guest/liyanqin/tutorial/p7-xs-gem5 && export gem5_home=`pwd`
```

```
prepare_gem5() {
    pushd $gem5_home && \
    cd ext/dramsim3 && \ # build 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
}
```

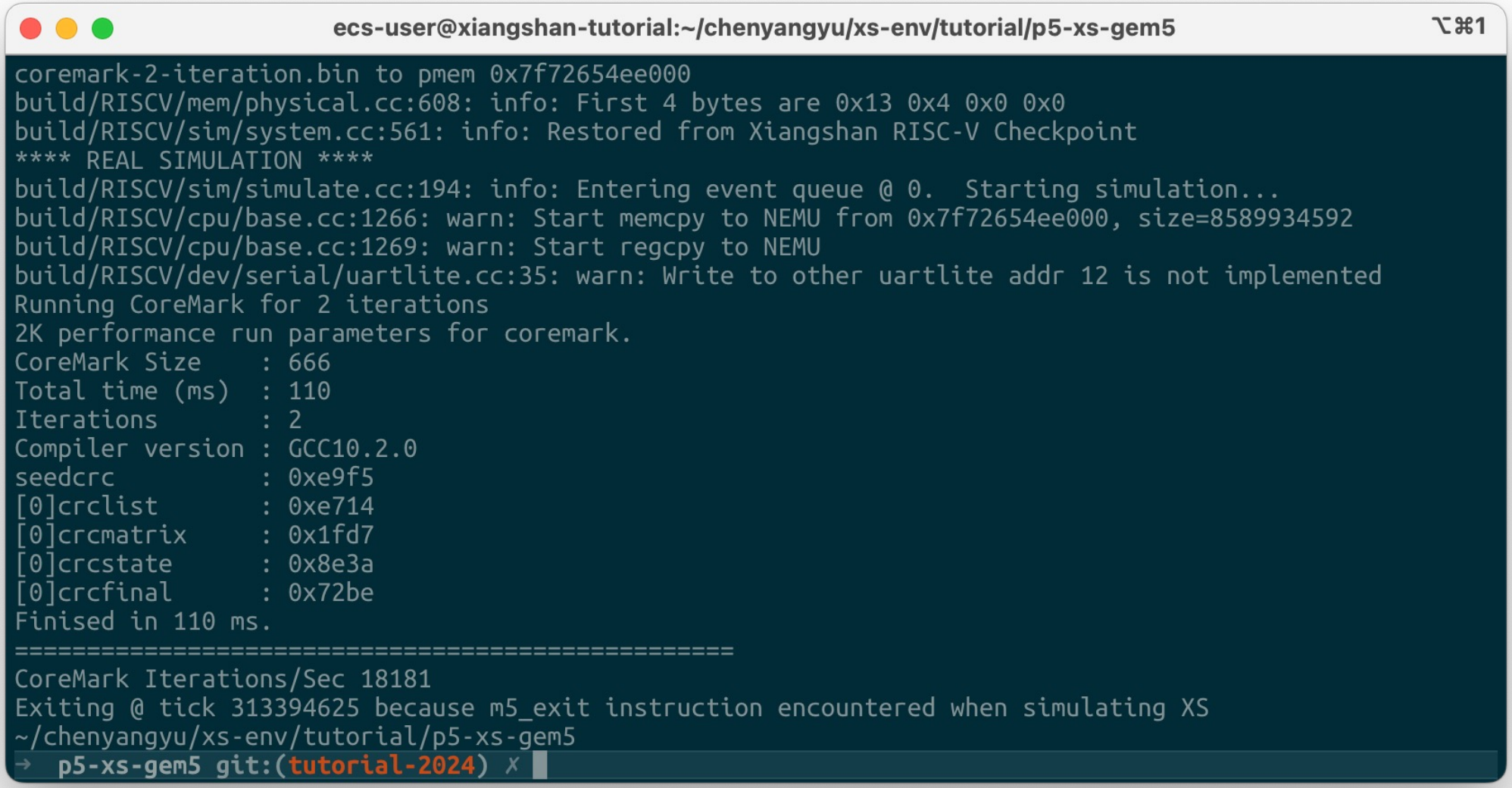
XS-Gem5

- **Hands-on:** run bare-metal workload on XS-Gem5

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

```
#~9s
pushd $gem5_home && \
export GCBV_REF_SO=$NEMU_HOME/build/riscv64-nemu-gem5-ref-so && \
mkdir -p util/xs_scripts/coremark && \ # Setup CoreMark working directory
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


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

- **Hands-on:** analyze the performance counters

```
$ bash 2-gem5_counter.sh | head -n 20
```

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

XS-Gem5

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) ✗
```

XS-Gem5: Performance Analysis Examples

- **Hands-on:** run the Cache MPKI analysis script

```
$ 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
```

Note: here we take the results from SPEC CPU 2006 checkpoints as an example for better showcase.

Cache MPKI

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

```
['GemsFDTD', 'astar', 'bwaves', 'bzip2', 'cactusADM', 'calculix', 'dealII', 'gamess', 'gcc', 'gobmk', 'gromacs', 'h26
4ref', 'hmmer', 'lbm', 'leslie3d', 'libquantum', 'mcf', 'milc', 'namd', 'omnetpp', 'perlbench', 'povray', 'sjeng', 's
oplex', '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) ✗
```

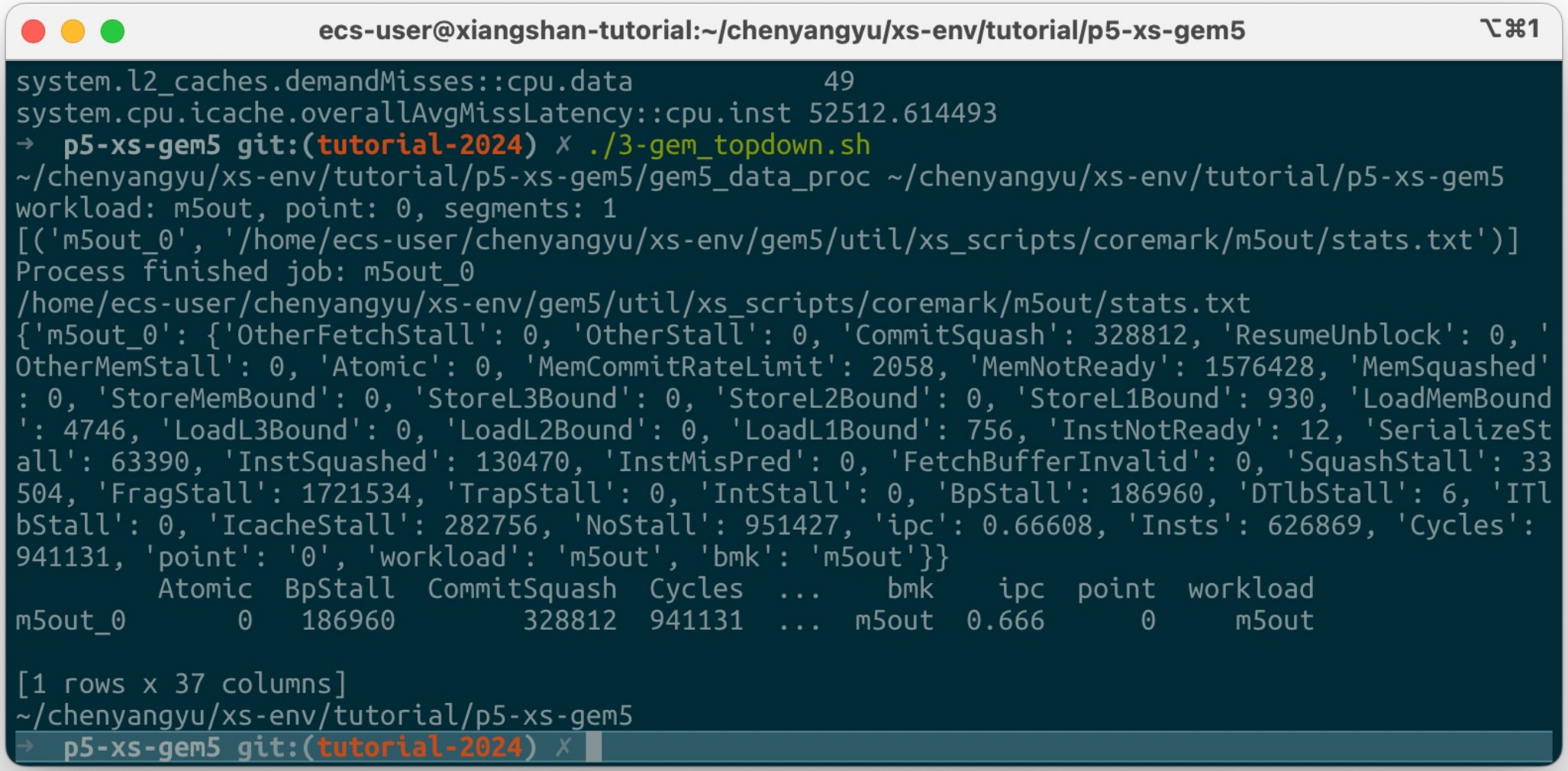
XS-Gem5: Performance Analysis Example

- **Hands-on:** run the Top-Down analysis script

```
$ 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) ✗

XS-Gem5: Performance Analysis Example

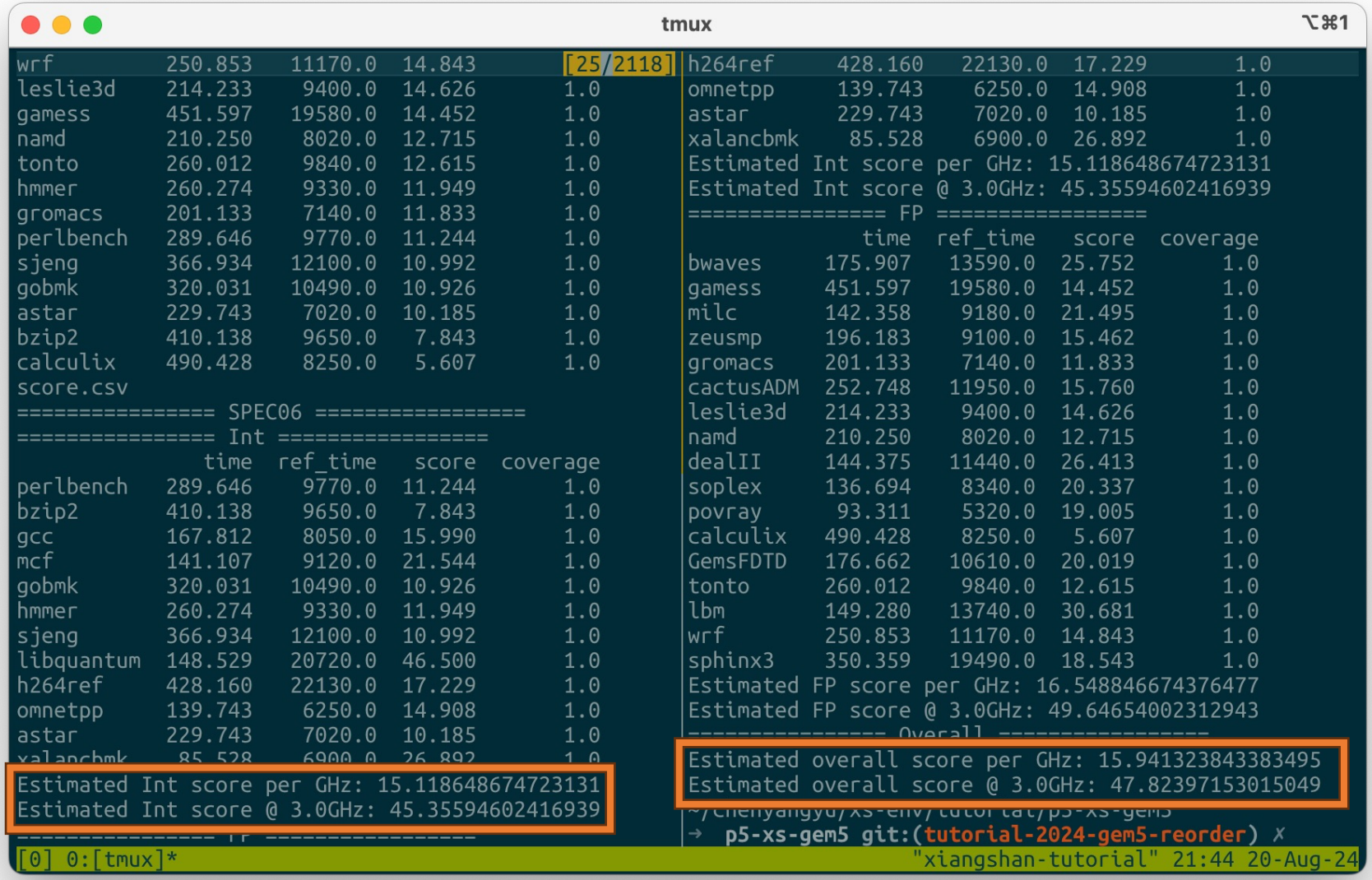
- **Hands-on:** run the SPEC CPU score calculation script

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

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

Note: here we take the results from SPEC CPU 2006 checkpoints as an example for better showcase.

SPEC CPU score



Callback: Check your own emu

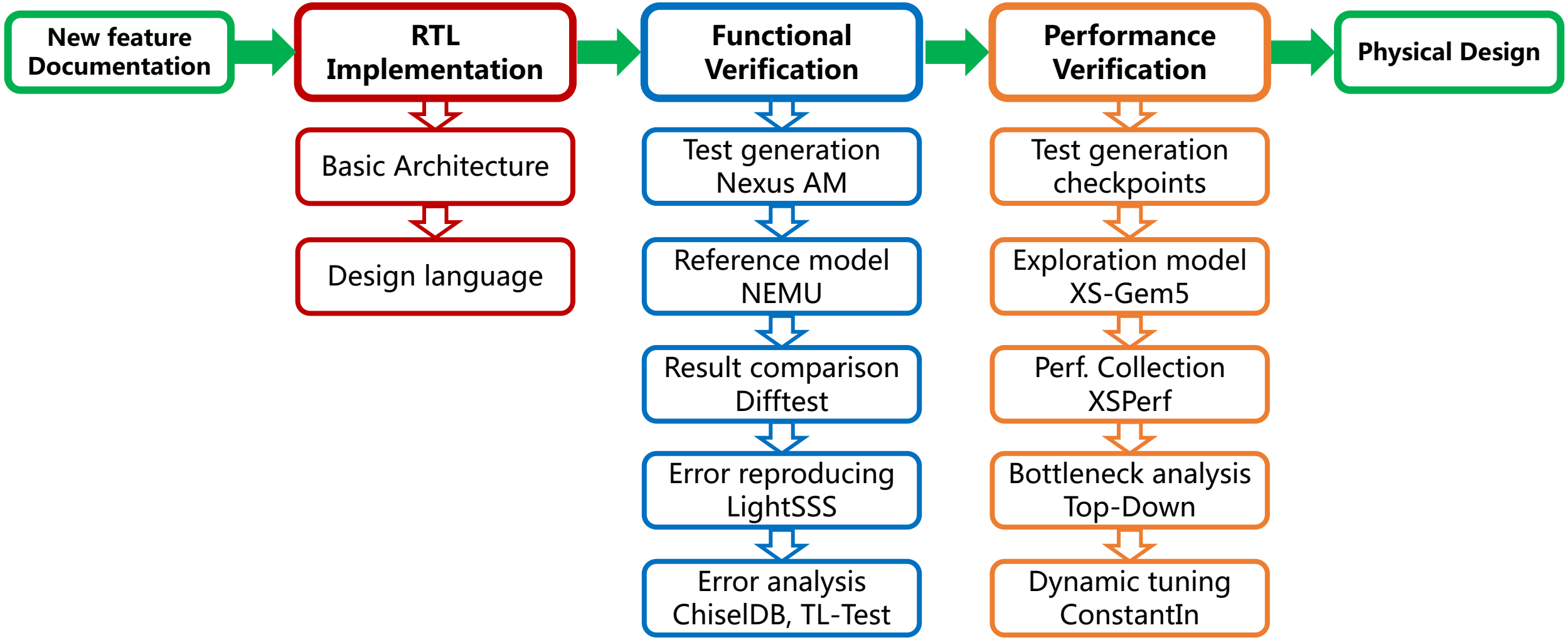
- Return to the first terminal where you compiled the emu
- Run simulation with your own emulator

```
$ 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 the tab key for command completion

```
$ ./build/emu -i ../tutorial/p1-basic-func/hello.bin --diff ./ready-to-
run/riscv64-nemu-interpreter-so 2> perf.err
```

Summary: MinJie Development Flows and Tools



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

中国开放指令生态（RISC-V）联盟
China RISC-V Alliance

香山开源处理器社区
XiangShan Open-Source Processor Community

Thanks!

This tutorial is organized by Haojin Tang, Luoshan Cai, Yanqin Li and Zefan Xu.