

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

---

*Kaifan Wang, Yinan Xu, Kan Shi, Yungang Bao*

*The XiangShan Team*

Institute of Computing Technology (ICT)  
Chinese Academy of Sciences (CAS)

ASPLOS'23@Vancouver, Canada

March 25, 2023



# Schedule

Time	Topic
9:00-9:25	Introduction of the XiangShan Project
9:25-9:35	Tutorial Overview and Highlights
9:35-10:20	Microarchitecture Design and Implementation
	Coffee Break
<b>10:40-12:00</b>	<b><i>Hands-on Development</i></b>

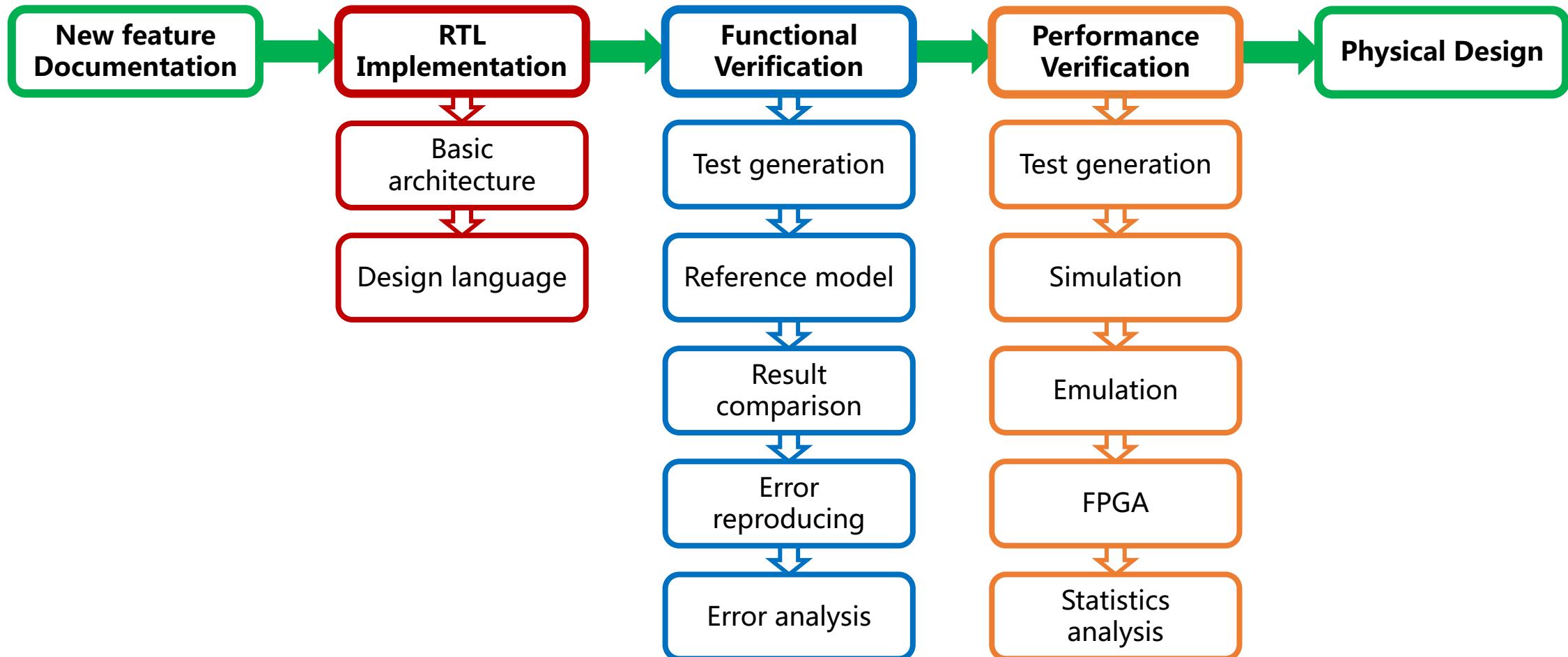


# What we will cover in this tutorial

- Highlights XiangShan on Chips, FPGAs, and FireSim
- CPU Microarchitecture (45 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 (80 minutes, 10:40AM after coffee break)
  - Introduction of their usages – How to develop on XiangShan with MinJie
  - Simulation and Debugging
  - Research Demo

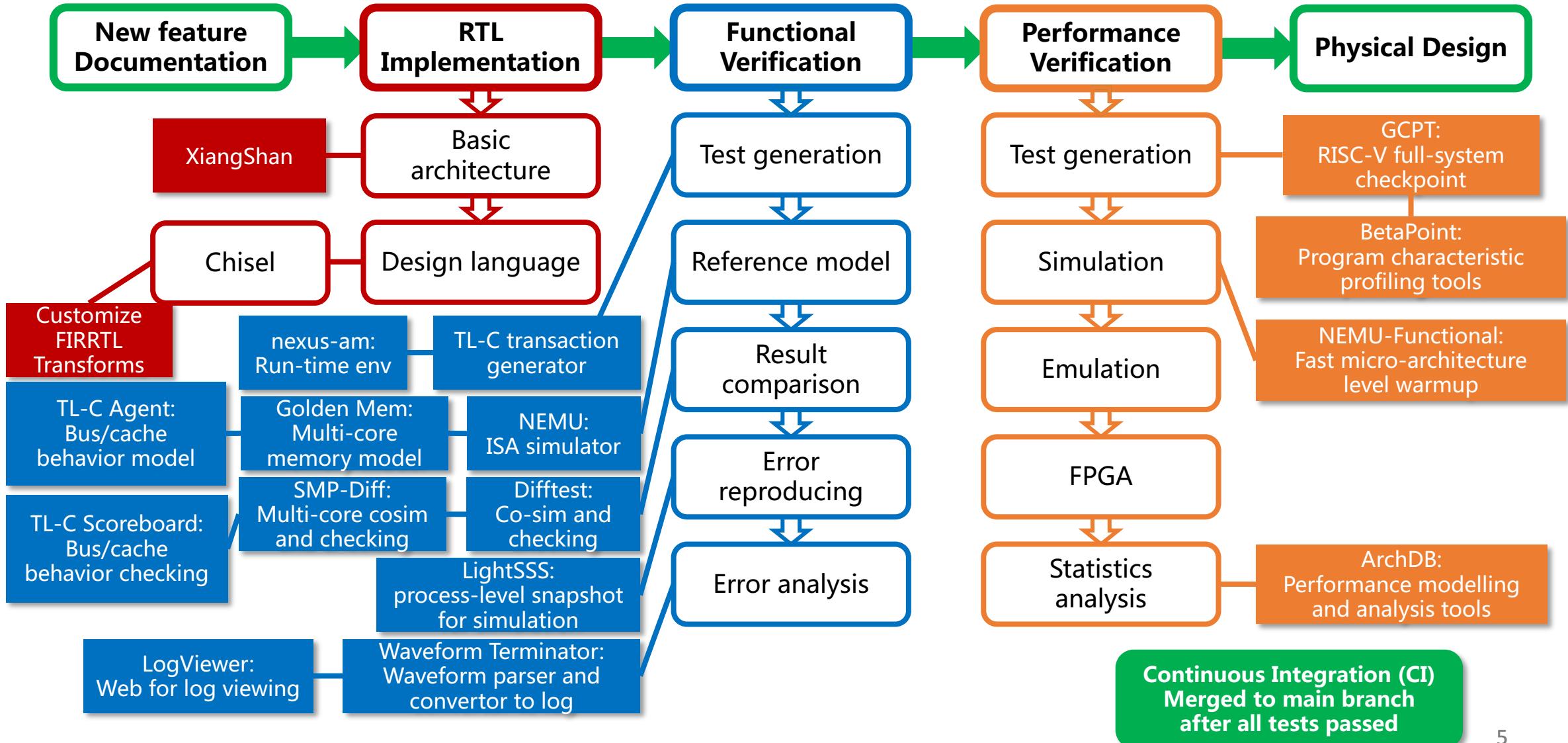


# Roadmap: MinJie Development Flows and Tools





# Roadmap: MinJie Development Infrastructures





# In this tutorial, we will

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

## Simulation

- Environment
- RTL Generation
- RTL Simulation
- .....

## Func. Verification

- Nexus-AM
- NEMU
- Difftest
- .....

## Perf. Verification

- Perf. counter
- Constin
- Checkpoint
- .....

## Research Demo

- Oracle BP
- Top-Down
- .....

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



# Demo Instructions

- **Interactive 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
- Copy tutorial environment `xs-env-asplos2023` to your path
- Set up environment variables



# Prerequisites

```
# Login  
$ wget https://openxiangshan.github.io/tutorial.pem  
$ chmod 400 tutorial.pem  
$ ssh -i "tutorial.pem" guest@47.88.5.191
```

```
# copy xs-env to your path  
$ cp -r /opt/xs-env-asplos2023 /home/guest/YOUR_NAME
```



# Prerequisites

```
# make sure to execute it under xs-env-asplos2023

$ pwd
#/home/guest/YOUR_NAME

# set up environment variables

$ 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
```



# Prerequisites

# Project Structure

```
$ tree -d -L 1  
.  
├── NEMU          NEMU  
├── NEMU-ahead    NEMU ahead version  
├── XiangShan     XiangShan processor  
├── XiangShan-Oracle XiangShan processor (oracle BP version)  
├── nexus-am      Abstract machine  
└── tutorial      Scripts and patches for tutorial
```

# Enter XiangShan directory

```
$ cd XiangShan
```



# Chisel Compilation

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



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

*Options:*

*CONFIG=TutorialConfig*

*Configuration of XiangShan*

*EMU\_THREADS=2*

*Simulation threads*

*EMU\_TRACE=1*

*Enable waveform dumping*

*// WITH\_DRAMSIM=1*

*Enable DRAMSim3 for DRAM simulation*

*// WITH\_CHISELDB = 1*

*Enable ChiselDB feature*

*// WITH\_CONSTANTIN = 1*

*Enable Constantin feature*



# Run RTL Simulation with Verilator

- Compilation might take 30 mins~, let it run in background
- Meanwhile we switch to a new shell and use pre-built emu

```
$ cd /home/guest/YOUR_NAME && source env.sh && cd tutorial  
$ ./emu --help  
# Some key 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  
  
$ ./emu -i hello.bin --no-diff 2>hello.err
```

**Great!**

**We have learned the basic simulation process of Xiangshan.**

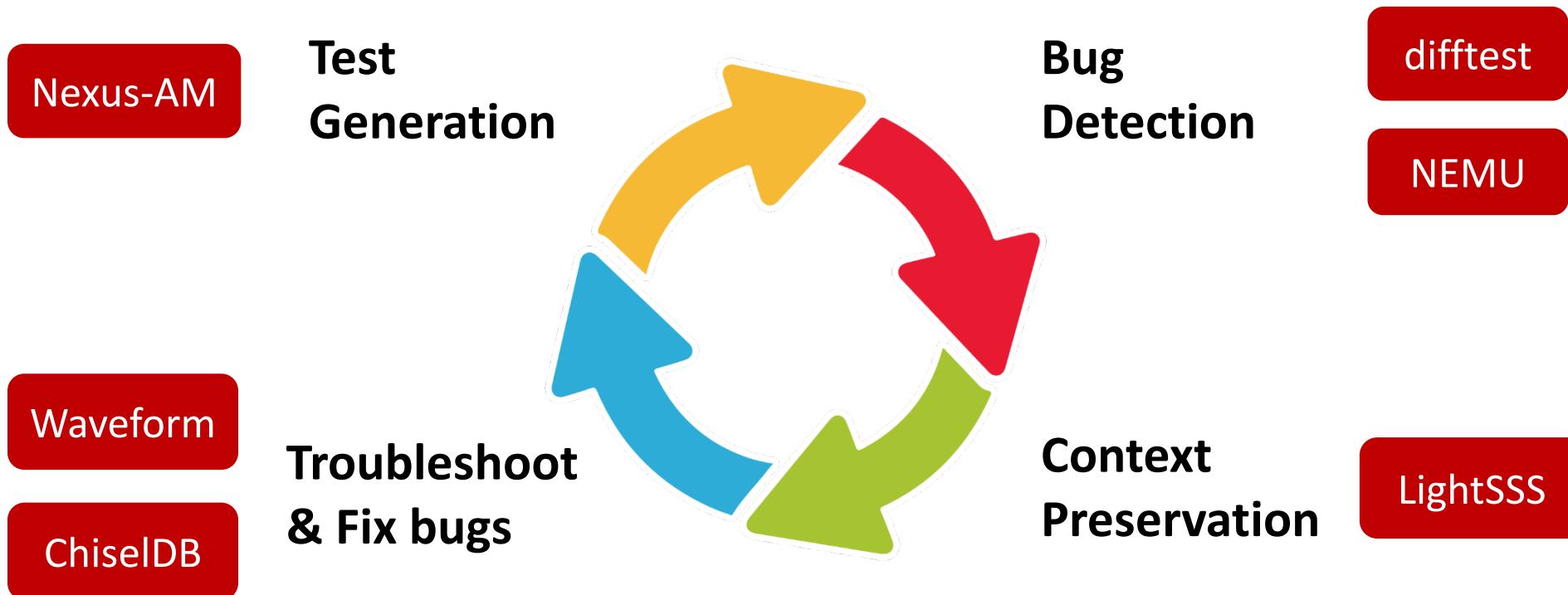
Once we get the RTL simulator ready, what's next?

How can we

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



# MinJie Functional Verification





# Bare Metal Runtime: Nexus-AM

- 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**

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

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

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

- **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**

- 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          build NEMU as the reference for XiangShan
```

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

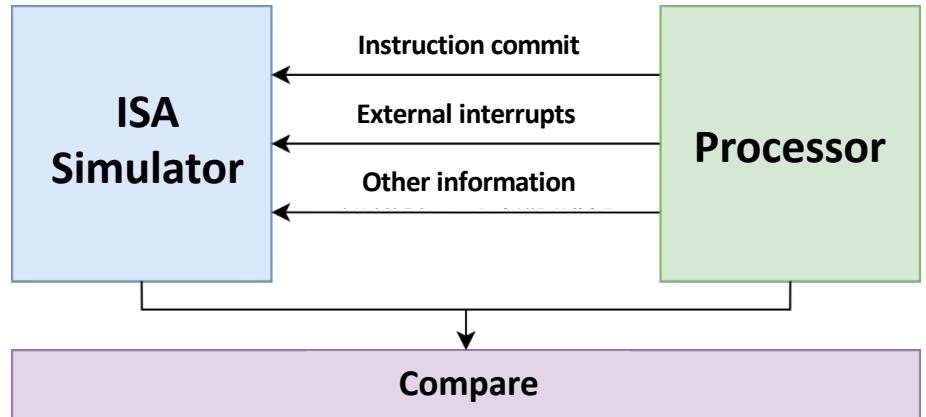
```
$ bash run-nemu.sh
```

```
# 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
```



# ISA co-simulation using DiffTest

- **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 a 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

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

```
# Running Coremark takes about 2 minutes
```

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

```
# ./emu \      if the simulator is successfully compiled, you can use $NOOP_HOME/build/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
Emu compiled at Mar 24 2023, 20:33:03
Using simulated 32768B flash
[warning]no valid flash bin path, use preset flash instead
The image is /home/guest/xs-env-asplos2023/nexus-am/apps/coremark/build/coremark-riscv64-xs.bin
Using simulated 8192MB RAM
NemuProxy using /home/guest/xs-env-asplos2023/NEMU/build/riscv64-nemu-interpreter-so
The first instruction of core 0 has committed. Difftest enabled.
[NEMU] Can not find flash image: (null)
[NEMU] Use built-in image instead
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'flash' at [0x0000000010000000, 0x00000000100fffff]
Running CoreMark for 1 iterations
2K performance run parameters for coremark.
CoreMark Size      : 666
Total time (ms)   : 4288
Iterations        : 1
Compiler version  : GCC9.4.0
seedcrc           : 0xe9f5
[0]crclist         : 0xe714
[0]crcmatrix       : 0x1fd7
[0]crcstate         : 0x8e3a
[0]crcfinal         : 0xe714
Finised in 4288 ms.
=====
CoreMark Iterations/Sec 233
Core 0: HIT GOOD TRAP at pc = 0x80001c52
total guest instructions = 398,372
instrCnt = 398,372, cycleCnt = 517,099, IPC = 0.770398
Seed=0 Guest cycle spent: 517,103 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 77,559ms
```

- **Hands-on:**

- We injected a bug in a pre-built XiangShan processor
- Now, let's trigger it on the pre-built buggy XiangShan 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 will report the error once failed
  - Dump info like PC, GPRs, etc.

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

```
===== REF Regs =====
$0: 0x0000000000000000    ra: 0x0000000080001620    sp: 0x000000008000cef0    gp: 0x0000000000000000
tp: 0x0000000000000000    t0: 0x000000000000001f    t1: 0x0000000080003a08    t2: 0x0000000000000000
s0: 0x0000000000007fff    s1: 0x0000000000000000    a0: 0x00000000800039e8    a1: 0x0000000000000000
a2: 0x0000000000000000    a3: 0x0000000080003a18    a4: 0x0000000000000001    a5: 0x0000000000000009
a6: 0x0000000000000001    a7: 0x0000000080003be4    s2: 0x0000000000000000    s3: 0x0000000000000000
s4: 0x0000000000000000    s5: 0x0000000000000000    s6: 0x0000000000000000    s7: 0x0000000000000000
s8: 0x0000000000000000    s9: 0x0000000000000000    s10: 0x0000000000000000   s11: 0x0000000000000000
t3: 0x0000000080003be4   t4: 0x0000000080003bd8   t5: 0x0000000080003c54   t6: 0x000000000000001f
ft0: 0xfffffff00000000   ft1: 0xfffffff00000000   ft2: 0xfffffff00000000   ft3: 0xfffffff00000000
ft4: 0xfffffff00000000   ft5: 0xfffffff00000000   ft6: 0xfffffff00000000   ft7: 0xfffffff00000000
fs0: 0xfffffff00000000   fs1: 0xfffffff00000000   fa0: 0xfffffff00000000   fa1: 0xfffffff00000000
fa2: 0xfffffff00000000   fa3: 0xfffffff00000000   fa4: 0xfffffff00000000   fa5: 0xfffffff00000000
fa6: 0xfffffff00000000   fa7: 0xfffffff00000000   fs2: 0xfffffff00000000   fs3: 0xfffffff00000000
fs4: 0xfffffff00000000   fs5: 0xfffffff00000000   fs6: 0xfffffff00000000   fs7: 0xfffffff00000000
fs8: 0xfffffff00000000   fs9: 0xfffffff00000000   fs10: 0xfffffff00000000  fs11: 0xfffffff00000000
ft8: 0xfffffff00000000   ft9: 0xfffffff00000000   ft10: 0xfffffff00000000  ft11: 0xfffffff00000000
pc: 0x0000000080000cc4 mstatus: 0x8000000a00006000 mcause: 0x0000000000000000 mepc: 0x4ede2b1aa1cb2ef6
                           sstatus: 0x8000000200006000 scause: 0x0000000000000000 sepc: 0x0000000000000000
satp: 0x0000000000000000
mip: 0x0000000000000000 mie: 0x0000000000000000 mscratch: 0x0000000000000000 sscratch: 0x0000000000000000
mideleg: 0x0000000000000000 medeleg: 0x0000000000000000
mtval: 0x0000000000000000 stval: 0x91da2105e8dfee5b mtvec: 0x0000000000000000 stvec: 0x0000000000000000
privilege mode:3 pmp: below
  0: cfg:0x00 addr:0x0000000000000000| 1: cfg:0x00 addr:0x0000000000000000
  2: cfg:0x00 addr:0x0000000000000000| 3: cfg:0x00 addr:0x0000000000000000
  4: cfg:0x00 addr:0x0000000000000000| 5: cfg:0x00 addr:0x0000000000000000
  6: cfg:0x00 addr:0x0000000000000000| 7: cfg:0x00 addr:0x0000000000000000
  8: cfg:0x00 addr:0x0000000000000000| 9: cfg:0x00 addr:0x0000000000000000
 10: cfg:0x00 addr:0x0000000000000000|11: cfg:0x00 addr:0x0000000000000000
 12: cfg:0x00 addr:0x0000000000000000|13: cfg:0x00 addr:0x0000000000000000
 14: cfg:0x00 addr:0x0000000000000000|15: cfg:0x00 addr:0x0000000000000000
priviledgeMode: 3
      a5 different at pc = 0x0080000cc2, right= 0x0000000000000009, wrong = 0x0000000000000000
Core 0: ABORT at pc = 0x80000cd2
total guest instructions = 1639
instrCnt = 1639, cycleCnt = 8726, IPC = 0.187829
Seed=0 Guest cycle spent: 8729 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 1814ms
```

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

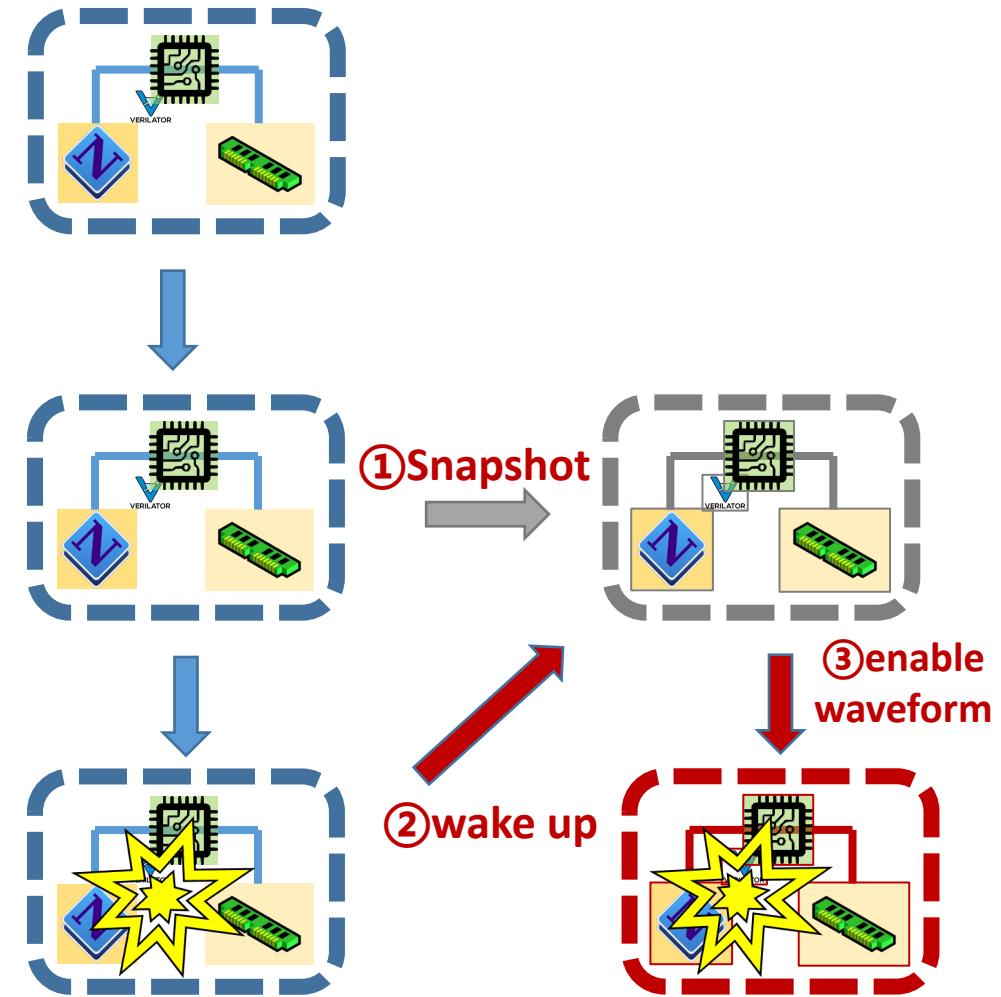
```
$ bash run-emu-dumpwave.sh  
$ ls $NOOP_HOME/build | grep "vcd" # There should be a .vcd waveform
```

```
# ./emu-bug \  
-i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \  
--diff $NEMU_HOME/build/riscv64-nemu-interpreter-so \  
-b 1200 \      the begin cycle of the wave, you may find XiangShan trap at 1600s cycle in previous step  
-e 1700 \      the end cycle of the wave  
--dump-wave \  set this option to dump wave, you will find *.vcd on $NOOP_HOME/build
```



# Light-weight Simulation SnapShot: LightSSS

- 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 for on-demand debugging (waveform)

- **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 \      No longer need to use complex parameters
2 > lightsss.err
```



# LightSSS for on-demand debugging (waveform)

- **Hands-on:** use lightSSS to dump waveform
- LightSSS is working if you see:

```
[FORK_INFO pid(NUMBER)] the oldest checkpoint start to dump wave and dump nemu Log...
[FORK_INFO pid(NUMBER)] dump wave to /SOME/PATH/XXX.vcd...
```

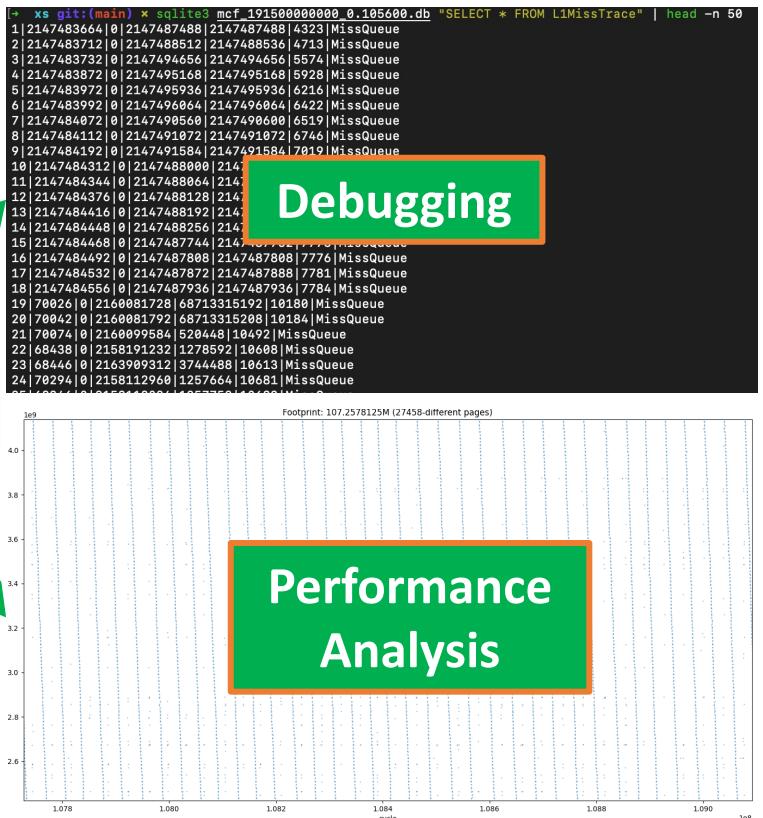
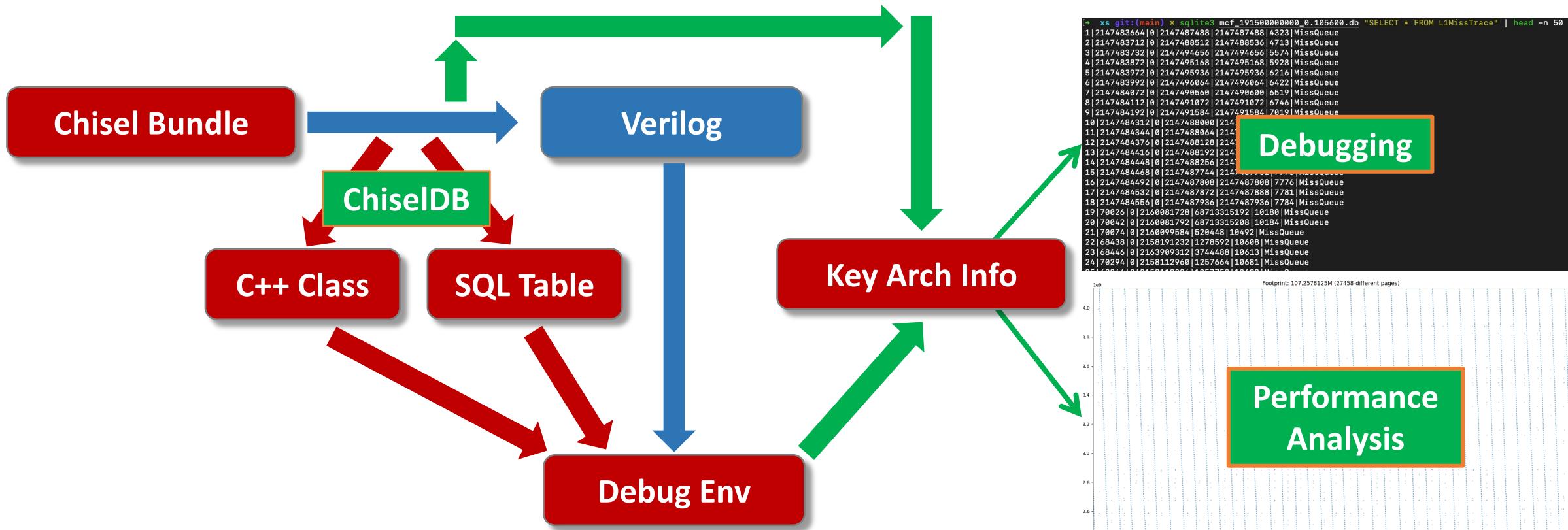
- After that, simulation will start again from the latest snapshot



# Debug-friendly Structured Database: ChiselDB

- **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

- Automated workflow for structured data



- **Design source code:** *XiangShan/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)
```

- **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
)
```

- 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)
```

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

- Add TLLOG
- Inject bug
- Compile and run
- Analyze TLLOG

- Hands-on: Analysis of Cache Coherence Violation

```
# Inject bug – We set all Release Data as a constant value
```

```
$ cd $NOOP_HOME/huancun
```

```
$ cat $XS_PROJECT_ROOT/tutorial/cc_err.patch
```

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

- Hands-on: Analysis of Cache Coherence Violation

```
# Compile and run (time consuming, use pre-built emu instead)  
# normally, `make emu -j4 EMU_THREADS=4 WITH_CHISELDB=1`  
  
$ cd $NOOP_HOME  
  
$ ./tutorial/emu-cc-err -i ready-to-run/linux.bin \  
  --diff ready-to-run/riscv64-nemu-interpreter-so \  
  --dump-db 2>linux.err
```

- Hands-on: Analysis of Cache Coherence Violation

```
# Analyze  
  
# 1. convert failed addr from Hex to Decimal  
  
# 2. xargs passes failed addr as Eaddr for 3.  
  
# 3. sqlite query for all transactions on Eaddr  
  
# 4. use script to parse TLLog  
  
$ printf "%d\n" 0x80000dc0 | xargs -I Eaddr \  
sqlite3 build/<TIME>.db \  
"select * from TLLOG where ADDRESS=Eaddr" | \  
sh scripts/utils/parseTLLog.sh
```

**one-click script at**  
**[..../tutorial/analyze\\_db.sh](#)**



# ChiselDB:

# Result: [Time | To\_From | Channel | Opcode | Permission | Address | Data(0)]

# Data successfully transferred from L1I to L2

1364 L2\_L1I\_0 C ReleaseData Shrink TtoN 80000dc0 689cf9796ec050ef

# Success from L2 to L3

1365 L3\_L2\_0 C ReleaseData Shrink TtoN 80000dc0 689cf9796ec050ef

# But when L1I acquires Eaddr again, data loaded from L3 is wrong

1796 L2\_L1I\_0 A AcquireBlock Grow NtoB 80000dc0

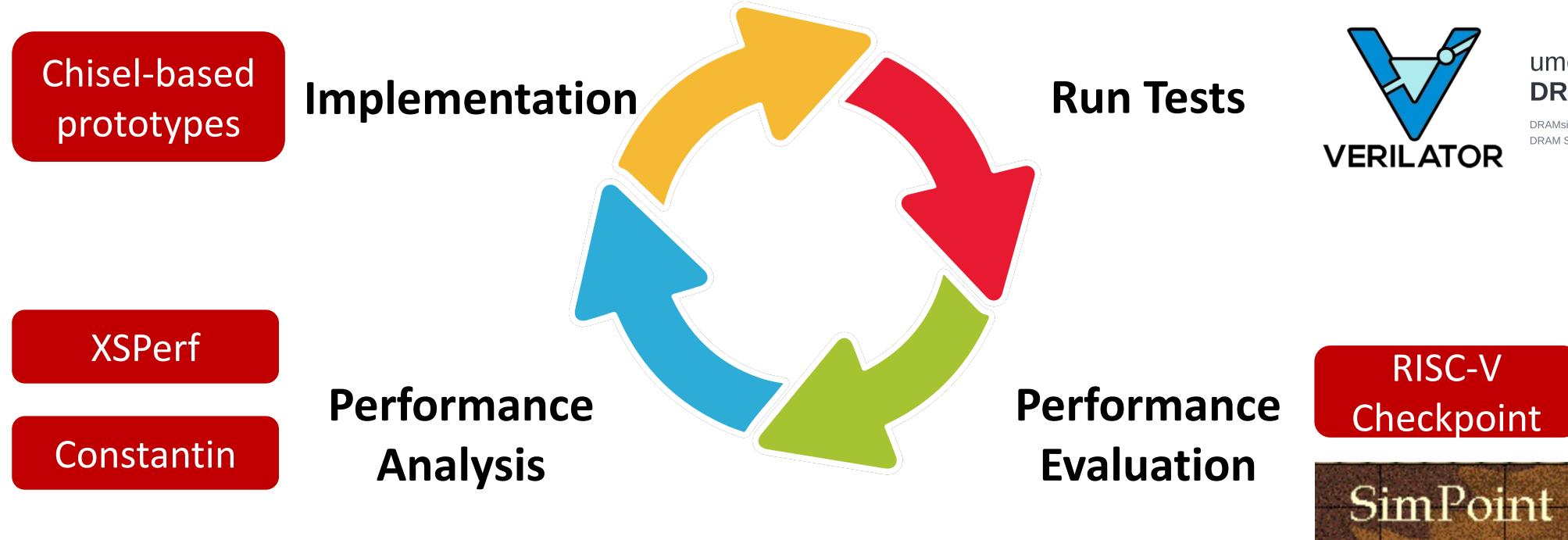
1797 L3\_L2\_0 A AcquireBlock Grow NtoB 80000dc0

1808 L3\_L2\_0 D GrantData Cap toT 80000dc0 0000000000abcdef

1809 L2\_L1I\_0 D GrantData Cap toT 80000dc0 0000000000abcdef

# So there must be something wrong when L3 records Release Data

# MinJie Performance Verification



umd-memsys/  
DRAMsim3

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



RISC-V  
Checkpoint





# Simulation Performance Counter

- Two common types:
  - **Accumulation** : basic accumulator counter
  - **Histogram** : count the distribution

## 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



# Simulation Performance Counter

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

```
$ cd $XS_PROJECT_ROOT/tutorial  
$ ./emu -i hello.bin \  
  --diff $NEMU_HOME/build/riscv64-nemu-interpreter-so 2>hello.err  
  
$ vim hello.err
```



# Simulation Performance Counter

```
[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>

- **Motivation**
  - Wanna test the performance under different parameters
  - Bug re-compilation is **time consuming**
  - Can we change parameters(constants) at runtime?
- **We present Constantin** (Constant - in)
- **Design/Highlights:**
  - DPI-C : Using C++ function in Chisel code (through Verilog using BlackBox)
  - Signal values configured at runtime rather than compile time

- Constants keep constant only during runtime

```

+ XiangShan_another git:(constantinople) x cat build/constantin.txt
lowerbound 10
presel0
presel1

+ XiangShan_another git:(constantinople) x ./build/emu -i ~oracle/XiangShan_oracle/ready-to-run/microbench.bin -I 10000 -e 0 2> /dev/null
Using simulated 32768B flash
No valid flash bin path, use preset 'flash' instead
The image is /nfs/home/chengukai/oracle/XiangShan_oracle/ready-to-run/microbench.bin
Using simulated 8192MB RAM
--diff is not given, try to use $NEMU_HOME/build/riscv64-nemu-interpreter-so by default
NemuProxy using /nfs/home/chengukai/NEMU_backup/build/riscv64-nemu-interpreter-so
The file /nfs/home/chengukai/NEMU/core.d has committed. Difftest enabled.
[NEMU] Use built-in image
[sysdev] Using /sysdev
[nemu] Running MicroBench
[insert] Quick sort: Core
to guest instructions = 10000
instructions = 10,981, cycle0
Seed#0 Guest cycle spent:
Host time spent: 5,93ms
+ XiangShan_another git:
lowerbound 10
presel0
presel1
+ XiangShan_another git:
Emu compiled at Feb 6 2022, 11:24:27
Using simulated 32768B flash
No valid flash bin path, use preset 'flash' instead
The image is /nfs/home/chengukai/oracle/XiangShan_oracle/ready-to-run/microbench.bin
Using simulated 8192MB RAM
--diff is not given, try to use $NEMU_HOME/build/riscv64-nemu-interpreter-so by default
NemuProxy using /nfs/home/chengukai/NEMU_backup/build/riscv64-nemu-interpreter-so
Assertion failed at line 72285!
The assertion statement might be some assertion failed.
Core #0: ABORT at pc = 0xffffffd19e3731e4
total guest instructions = 0
Instructions per second = IPC = 0.000000
Seed#0 Guest cycle spent: 6 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 4ms
+ XiangShan git:(master) x export NEMU_HOME=/nfs/home/chengukai/NEMU_backup
+ XiangShan git:(master) x export NOOP_HOME=/nfs/home/chengukai/tutorial_demo/Constantin/XiangShan
+ XiangShan git:(master) x python3 scripts/constantHelper.py constant.json
iteration 0 begins

[('block_cycles_cache_0', 5), ('block_cycles_cache_1', 42), ('block_cycles_cache_2', 19), ('block_cycles_cache_3', 51)
[('block_cycles_cache_0', 12), ('block_cycles_cache_1', 48), ('block_cycles_cache_2', 29), ('block_cycles_cache_3', 41)
[('block_cycles_cache_0', 13), ('block_cycles_cache_1', 19), ('block_cycles_cache_2', 31), ('block_cycles_cache_3', 21)
[('block_cycles_cache_0', 21), ('block_cycles_cache_1', 17), ('block_cycles_cache_2', 41), ('block_cycles_cache_3', 55)
[('block_cycles_cache_0', 1), ('block_cycles_cache_1', 1), ('block_cycles_cache_2', 1), ('block_cycles_cache_3', 1)
[('block_cycles_cache_0', 3), ('block_cycles_cache_1', 1), ('block_cycles_cache_2', 1), ('block_cycles_cache_3', 1)
[('block_cycles_cache_0', 1), ('block_cycles_cache_1', 1), ('block_cycles_cache_2', 1), ('block_cycles_cache_3', 1)

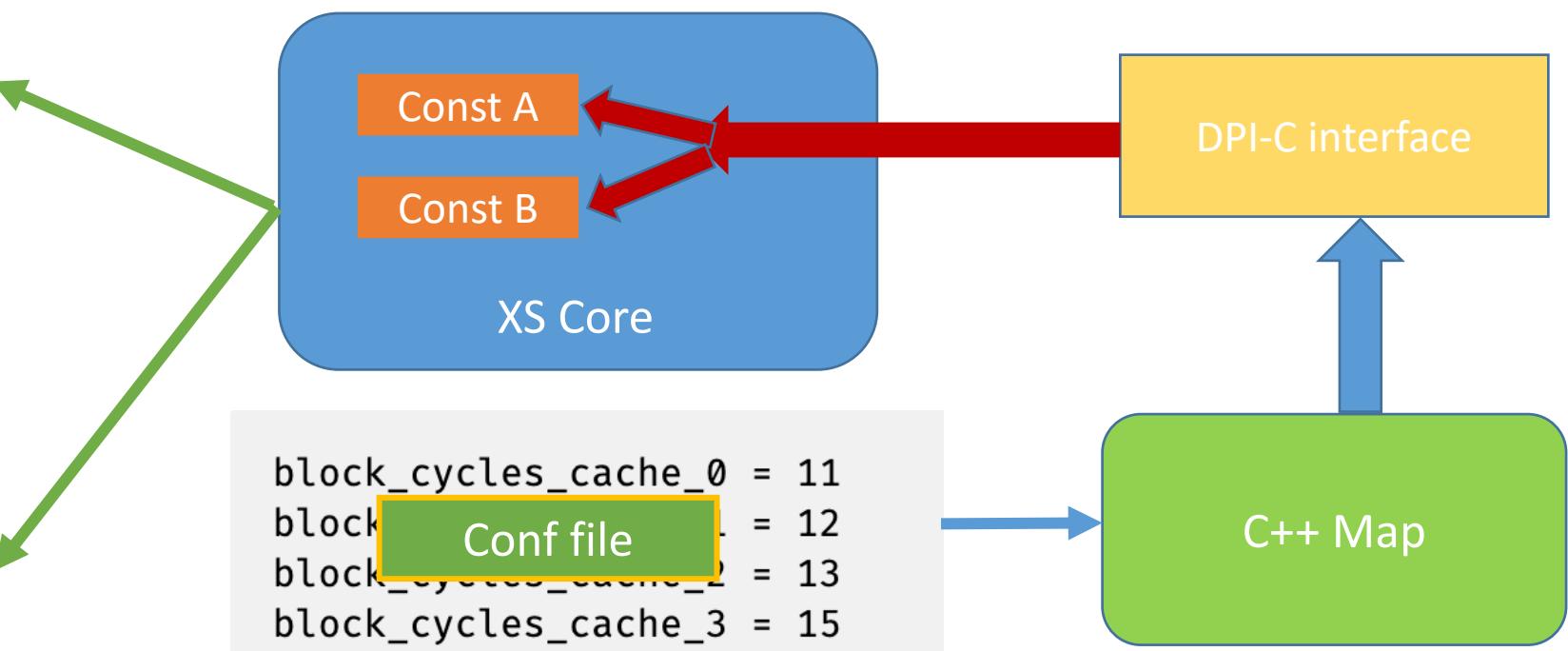
opt constant in this round
fitness is -4075

iteration 1 begins

[('block_cycles_cache_0', 1), ('block_cycles_cache_1', 18), ('block_cycles_cache_2', 9), ('block_cycles_cache_3', 17)
[('block_cycles_cache_0', 2), ('block_cycles_cache_1', 17), ('block_cycles_cache_2', 12), ('block_cycles_cache_3', 19)
[('block_cycles_cache_0', 21), ('block_cycles_cache_1', 17), ('block_cycles_cache_2', 45), ('block_cycles_cache_3', 55)
[('block_cycles_cache_0', 13), ('block_cycles_cache_1', 43), ('block_cycles_cache_2', 39), ('block_cycles_cache_3', 51)
[('block_cycles_cache_0', 22), ('block_cycles_cache_1', 59), ('block_cycles_cache_2', 21), ('block_cycles_cache_3', 11)
[('block_cycles_cache_0', 14), ('block_cycles_cache_1', 58), ('block_cycles_cache_2', 29), ('block_cycles_cache_3', 41)
[('block_cycles_cache_0', 7), ('block_cycles_cache_1', 59), ('block_cycles_cache_2', 39), ('block_cycles_cache_3', 19)
[('block_cycles_cache_0', 13), ('block_cycles_cache_1', 28), ('block_cycles_cache_2', 6), ('block_cycles_cache_3', 43)

opt constant in this round is [('block_cycles_cache_0', 11), ('block_cycles_cache_1', 18), ('block_cycles_cache_2', 127), ('block_cycles_cache_3', 17)]
fitness is -4075
opt constant for gene algorithm is [('block_cycles_cache_0', 11), ('block_cycles_cache_1', 60), ('block_cycles_cache_2', 127), ('block_cycles_cache_3', 17)]
fitness is -4075
+ XiangShan git:(master) x

```



- **Usage:** Create signal

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

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

- Usage: configuration file

```
// format: signal_name value
// default path: ${NOOP_HOME}/build/constantin.txt
// no comments allowed in real configuration file

block_cycles_cache_0 11
block_cycles_cache_1 18
block_cycles_cache_2 127
block_cycles_cache_3 17
```

- **Hands-on: Auto Solving of better constant values**
- **Process**
  - Enable AutoSolving & prepare signals
  - Compile
  - Run basic demonstration
  - Set parameters for AutoSolving
  - Auto solve
  - Cleanup

- **Hands-on: Auto Solving of better constant values**

```
# Enable AutoSolving
```

```
$ cat $XS_PROJECT_ROOT/tutorial/utility.patch
```

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

```
# Signal preparation
```

```
$ cat $XS_PROJECT_ROOT/tutorial/xs.patch
```

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

- **Hands-on: Auto Solving of better constant values**

```
# Since compiling is timing consuming, so we only present commands here

# Apply these two patches
# cd $XS_PROJECT_ROOT/XiangShan && git apply ..../tutorial/xs.patch
# cd utility && git apply ../../tutorial/utility.patch

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

- **Hands-on: Auto Solving of better constant values**

```
# Basic demonstration of manually setup of constants  
$ sh $XS_PROJECT_ROOT/tutorial/constantin-basic.sh
```

```
# cat constantin.txt | ${XS_PROJECT_ROOT}/tutorial/emu-constantin \  
-i ./ready-to-run/linux.bin \  
--diff ./ready-to-run/riscv64-nemu-interpreter-so 2>constantin.err  
  
# Contents read from configuration file to emu  
4  
block_cycles_cache_0 11  
block_cycles_cache_1 12  
block_cycles_cache_2 13  
block_cycles_cache_3 15
```

- **Hands-on: Auto Solving of better constant values**

```
# Automatical parameter solver configuration file  
$ cat ${XS_PROJECT_ROOT}/tutorial/constantin.json
```

- **Parameters defined**

- properties of signals
- target of Auto Solving
- parameters of genetic algorithm
- parameters of XS simulator

- **Hands-on: Auto Solving of better constant values**

```
# Run auto solver, output will be the optimal constant currently found  
$ sh $XS_PROJECT_ROOT/tutorial/constantin-solve.sh
```

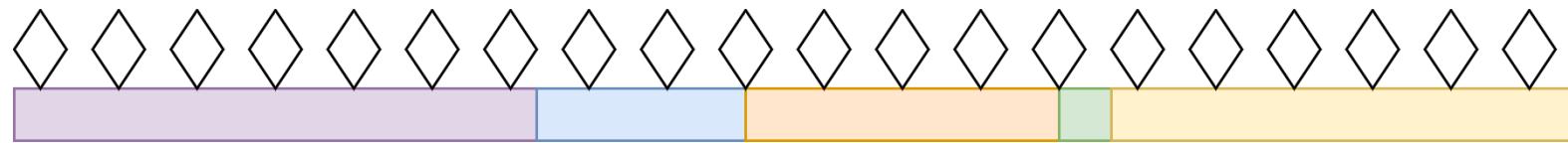
```
# The solver generates optimal parameters  
opt constant for gene algrithom is  
[['block_cycles_cache_0', XXX], ['block_cycles_cache_1', XXX],  
 ['block_cycles_cache_2', XXX], ['block_cycles_cache_3', XXX]] fitness 0
```

```
# Remember to cleanup if you have manually compiled  
$ sh $XS_PROJECT_ROOT/tutorial/constantin-clean.sh
```



# Checkpoints

- Uniform Checkpoint
- Divide a program to small parts which can be simulated in parallel



- Simpoint Checkpoint
- Select representative parts from a program with cluster analysis



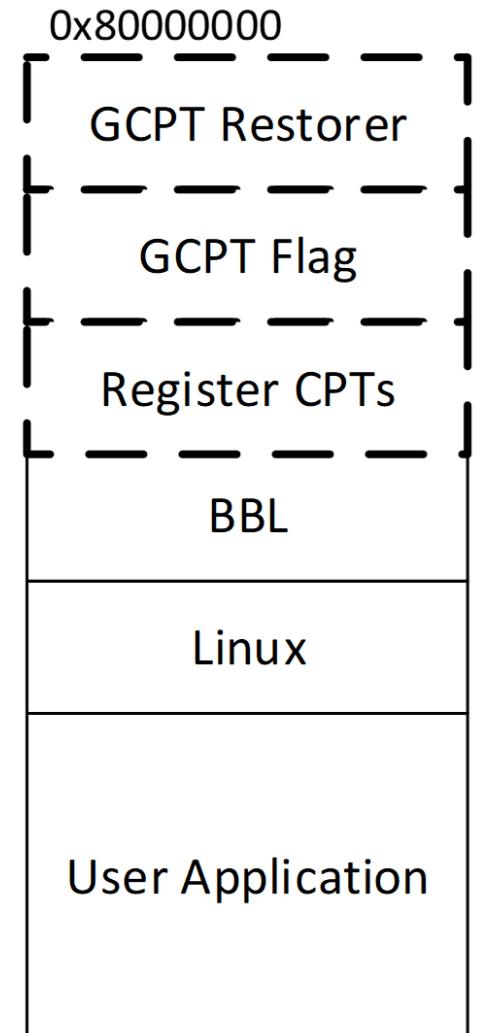


# Checkpoints: Uniform Checkpoint

- **step0: Prepare NEMU environment for checkpoints**

```
$ cd $XS_PROJECT_ROOT/tutorial/checkpoint  
$ sh uniform_step0_prepare.sh
```

```
# uniform_step0_prepare.sh  
  
# cd $NEMU_HOME  
  
# git checkout asplos2023-tutorial  
# git submodule update --init  
# make clean  
# make tutorial_defconfig  
# make -j4  
  
// generate gcpt restorer binary  
# cd resource/gcpt_restore && make -j4
```



Basic format of RISC-V Checkpoint



# Checkpoints: Uniform Checkpoint

- **step1: Generate uniform checkpoints in NEMU**

```
$ sh uniform_step1_gen.sh
```

```
# uniform_step1_gen.sh

# cd $NEMU_HOME
# rm -rf tutorial_uniform
# ./build/riscv64-nemu-interpreter
#   --cpt-interval 2000000  the profiling period for simpoint
#   -u                      uniformly take cpt with fixed interval
#   -b                      run with batch mode
#   -D tutorial_uniform     directory where the checkpoint is generated
#   -C try                  name of this task
#   -w linux                name of workload
#   -r ./resource/gcpt_restore/build/gcpt.bin binary of gcpt restorer
#   --dont-skip-boot        checkpoint immediately after boot
#   -I 3000000               max number of instructions executed
#   ./ready-to-run/Linux-0xa0000.bin
```



# Checkpoints: Uniform Checkpoint

```
[ 0.000000] Dentry cache hash table entries: 4096 (order: 3, 32768 bytes)
[ 0.000000] Inode cache hash table entries: 2048 (order: 2, 16384 bytes)
[ 0.000000] Sorting __ex_table...
[src/checkpoint/serializer.cpp:255,shortTakecpt] Should take cpt now: 2000002
[src/checkpoint/path_manager.cpp:77,setOutputDir] Created tutorial_uniform/try/linux/0/
[src/checkpoint/serializer.cpp:127,serializeRegs] Writing int registers to checkpoint memory @[0x80001000, 0x80001100) [0x1000, 0x1100]
[src/checkpoint/serializer.cpp:137,serializeRegs] Writing float registers to checkpoint memory @[0x80001100, 0x80001200) [0x1100, 0x1200)
[src/checkpoint/serializer.cpp:145,serializeRegs] Writing PC: 0xffffffffffff003e4c at addr 0x80001200
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x105: 0xffffffffffff8039075c
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x106: 0xfffffffffffffff
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x141: 0x80200098
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x142: 0xc
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x143: 0x80200098
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x180: 0x8000000000008066f
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x300: 0xa00000180
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x301: 0x8000000000014112d
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x302: 0xb109
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x303: 0x222
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x304: 0x22a
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x305: 0x800a0004
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x306: 0xffffffffffffffffff
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x340: 0x800abdc0
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x341: 0xffffffffffff80390ed4
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x342: 0x9
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x3a0: 0x1f0800
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x3b0: 0x200000000
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x3b1: 0x2002d000
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x3b2: 0x3fffffff
[src/checkpoint/serializer.cpp:168,serializeRegs] CSR 0x5c4: 0x3
[src/checkpoint/serializer.cpp:171,serializeRegs] Writing CSRs to checkpoint memory @[0x80001300, 0x80009300) [0x1300, 0x9300)
[src/checkpoint/serializer.cpp:179,serializeRegs] Touching Flag: 0xbccf at addr 0x80000f00
[src/checkpoint/serializer.cpp:183,serializeRegs] Record mode flag: 0x1 at addr 0x80000f08
[src/checkpoint/serializer.cpp:188,serializeRegs] Record time: 0x1 at addr 0x80000f10
[src/checkpoint/serializer.cpp:192,serializeRegs] Record time: 0x1 at addr 0x80000f18
[src/checkpoint/serializer.cpp:81,serializePMem] Put gcpt restorer ./resource/gcpt_restore/build/gcpt.bin to start of pmem
Opening tutorial_uniform/try/linux/0/_2000002.gz as checkpoint output file
[src/checkpoint/serializer.cpp:110,serializePMem] Written 0x7fffffff bytes
[src/checkpoint/serializer.cpp:110,serializePMem] Written 0x7fffffff bytes
[src/checkpoint/serializer.cpp:110,serializePMem] Written 0x7fffffff bytes
[src/checkpoint/serializer.cpp:110,serializePMem] Written 0x4 bytes
[src/checkpoint/serializer.cpp:116,serializePMem] Checkpoint done!
[src/checkpoint/serializer.cpp:263,notify_taken] Taking checkpoint @ instruction count 2000002
[src/cpu/cpu-exec.c:203,per_bb_profile] Should take checkpoint on pc 0xffffffffffff80003e4c
[ 0.000000] Memory: 25548K/30720K available (699K kernel code, 78K rwdta, 102K rodata, 3648K init, 98K bss, 5172K reserved, 0K cma-reserved)
[ 0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[ 0.000000] NR_IRQS: 0, nr_irqs: 0, preallocated irqs: 0

```

Take a checkpoint from here

Save int and float registers

Save CSR registers

Save other information and gcpt restorer

Write checkpoints to gz file



# Checkpoints: Uniform Checkpoint

- **step2: Run the uniform checkpoint in NEMU or XiangShan**
- Here we take NEMU as an example

```
$ sh uniform_step2_run_nemu.sh
```

```
# uniform_step2_run_nemu.sh

# cd $NEMU_HOME

# ./build/riscv64-nemu-interpreter
# -b                               run with batch mode
# --restore                         restoring from CPT FILE
# -I 1000000                         max number of instructions executed
# `find tutorial_uniform/try/Linux/0/ -type f -name "*.gz" | tail -1` path of the checkpoint files just generated
```



# Checkpoints: Uniform Checkpoint

```
[src/monitor/monitor.c:103,parse_args] Restoring from checkpoint
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'clint' at [0x000000038000000, 0x00000003800ffff]
[src/isa/riscv64/init.c:100,init_isa] NEMU will start from pc 0x80000000
[src/monitor/image_loader.c:69,load_img] Loading Gcpt file form cmdline: tutorial_uniform/try/linux/0/_200002_.gz
[src/monitor/image_loader.c:77,load_img] Loading GZ image tutorial_uniform/try/linux/0/_200002_.gz
[src/monitor/image_loader.c:69,load_img] Loading Gcpt restorer form cmdline: (null) → Restore from checkpoint and load gz files

[src/monitor/image_loader.c:71,load_img] No image is given. Use the default built-in image/restorer.
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'uartlite' at [0x00000000000003f8, 0x000000000000404]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'uartlite' at [0x000000040600000, 0x00000004060000c]
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'rtc' at [0x000000000000048, 0x00000000000004f]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'rtc' at [0x0000000a1000048, 0x0000000a100004f]
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'screen' at [0x000000000000100, 0x000000000000107]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'screen' at [0x000000040001000, 0x000000040001007]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'vmem' at [0x000000050000000, 0x0000000500752ff]
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'keyboard' at [0x000000000000060, 0x000000000000063]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'keyboard' at [0x0000000a1000060, 0x0000000a1000063]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'sdhci' at [0x000000040002000, 0x00000004000207f]
[src/device/sdcard.c:137,init_sdcard] Can not find sdcard image:
[src/monitor/monitor.c:44,welcome] Debug: OFF
[src/monitor/monitor.c:49,welcome] Build time: 19:35:10, Mar 25 2023
Welcome to riscv64-NEMU! → NEMU will start execution from the moment of that checkpoint
For help, type "help"
[ 0.00000] Memory: 25548K/30720K available (699K kernel code, 78K rwdta, 102K rodata, 3648K init, 98K bss, 5172K reserved, 0K cma-reserved)
[ 0.00000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[ 0.00000] NR_IRQS: 0, nr_irqs: 0, preallocated irqs: 0
[ 0.00000] clocksource: riscv_clocksource: mask: 0xfffffffffffffff max_cycles: 0x1d854df40, max_idle_ns: 3526361616960 ns
[ 0.00000] console [hvc0] enabled
[ 0.00000] console [hvc0] enabled
[ 0.00000] bootconsole [early0] disabled
[ 0.00000] bootconsole [early0] disabled
[ 0.00000] Calibrating delay loop (skipped), value calculated using timer frequency.. 2.00 BogoMIPS (lpj=10000)
[ 0.00000] pid_max: default: 4096 minimum: 301
[ 0.00000] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
[ 0.00000] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)
[ 0.01000] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[ 0.02000] clocksource: Switched to clocksource riscv_clocksource
```



# Checkpoints: Simpoint Checkpoint

- **step0: Prepare NEMU environment for Simpoint checkpoint**

```
$ sh simpoint_step0_prepare.sh
```

```
# simpoint_step0_prepare.sh

# cd $NEMU_HOME

# git checkout cpt-bk
# git submodule update --init
# cd resource/simpoint/simpoint_repo/analysiscode && make simpoint -j4
#                                     generate simpoint generator binary
# cd resource/gcpt_restore && make -j4    generate gcpt restorer binary

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



# Checkpoints: Simpoint Checkpoint

- step1: Execute a round of workload and collect program behavior

```
$ sh simpoint_step1_profiling.sh
```

```
# simpoint_step1_profiling.sh

# cd $NEMU_HOME
# rm -rf tutorial_simpoint

# ./build/riscv64-nemu-interpreter
#   -b                               run with batch mode
#   -D $NEMU_HOME/tutorial_simpoint  directory where the checkpoint is generated
#   -C profiling                      name of this task
#   -w stream                         name of workload
#   --simpoint-profile                is simpoint profiling
#   --interval 50000000               simpoint interval instructions
#   $XS_PROJECT_ROOT/tutorial/bin/stream-0xa0000.bin
```



# Checkpoints: Simpoint Checkpoint

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

```
-----  
STREAM version $Revision: 5.10 $
```

```
This system uses 4 bytes per array element.
```

```
-----  
Array size = 2000000 (elements), Offset = 0 (elements)
```

```
Memory per array = 7.6 MiB (= 0.0 GiB).
```

```
Total memory required = 22.9 MiB (= 0.0 GiB).
```

```
Each kernel will be executed 10 times.
```

```
The *best* time for each kernel (excluding the first iteration)  
will be used to compute the reported bandwidth.
```

```
-----  
Your clock granularity/precision appears to be 2047 microseconds.
```

```
Each test below will take on the order of 102400 microseconds.
```

```
(= 50 clock ticks)
```

```
Increase the size of the arrays if this shows that  
you are not getting at least 20 clock ticks per test.
```

```
-----  
WARNING -- The above is only a rough guideline.
```

```
For best results, please be sure you know the  
precision of your system timer.
```

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	710.2	0.023438	0.022528	0.024576
Scale:	181.7	0.089884	0.088064	0.092160
Add:	217.0	0.112868	0.110592	0.114688
Triad:	172.3	0.140402	0.139264	0.141312

```
-----  
Solution Validates: avg error less than 1.000000e-06 on all three arrays
```

```
-----  
[src/monitor/cpu-exec.c,110,cpu_exec] nemu: HIT GOOD TRAP at pc = 0x0000002aaaaaaaa5f6
```

NEMU will do profiling while  
executing workload STREAM



# Checkpoints: Simpoint Checkpoint

- step2: Obtain multiple program parts with the highest weights

```
$ sh simpoint_step2_cluster.sh
```

```
# simpoint_step2_cluster.sh

# cd $NEMU_HOME

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

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



# Checkpoints: Simpoint Checkpoint

```
Run number 3 of at most 4, k = 2
-----
Initialization seed trial #1 of 2; initialization seed = 123460
-----
Initialized k-means centers using random sampling: 2 centers
Number of k-means iterations performed: 1
BIC score: 123.793
Distortion: 1.61246
Distortions/cluster: 1.54841 0.0640483
Variance: 0.161246
Variances/cluster: 1.54841 0.00711647
-----
Initialization seed trial #2 of 2; initialization seed = 123461
-----
Initialized k-means centers using random sampling: 2 centers
Number of k-means iterations performed: 1
BIC score: 123.793
Distortion: 1.61246
Distortions/cluster: 0.0640483 1.54841
Variance: 0.161246
Variances/cluster: 0.00711647 1.54841
The best initialization seed trial was #1
-----
Post-processing runs
-----
For the BIC threshold, the best clustering was run 3 (k = 2)
Post-processing run 3 (k = 2)
```

Compute the information  
of K-means clustering

Obtain multiple program checkpoints  
with the highest weights



# Checkpoints: Simpoint Checkpoint

- step3: Generate corresponding Checkpoints based on clustering results

```
$ sh simpoint_step3_take_cpt.sh      # It may take about 3 minutes
```

```
# simpoint_step3_take_cpt.sh

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

# ./build/riscv64-nemu-interpreter
#   -b                                run with batch mode
#   -D tutorial_simpoint                directory where the checkpoint is generated
#   -C take_cpt                         name of this task
#   -w stream                           name of workload
#   -S ./tutorial_simpoint/cluster     simpoints dir
#   --checkpoint-interval 50000000    simpoint interval instructions
#   $XS_PROJECT_ROOT/tutorial/bin/stream-0xa0000.bin
```



# Checkpoints: Simpoint Checkpoint

- **step4: Run the simpoint checkpoint in NEMU or XiangShan**
- Here we take XiangShan as an example

```
$ sh simpoint_step4_run_xs.sh
```

```
#simpoint_step4_run_xs.sh

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



# Research on BPU: Oracle Branch Predictor

- **Motivation**

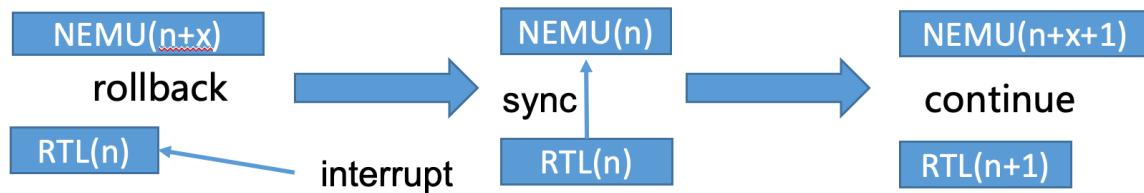
- Branch predictor precision has a great impact on the overall performance
- Current precision may **hide bottleneck** of other units
- We want a parallelizable design space exploration process

- **We present Oracle Branch Predictor**

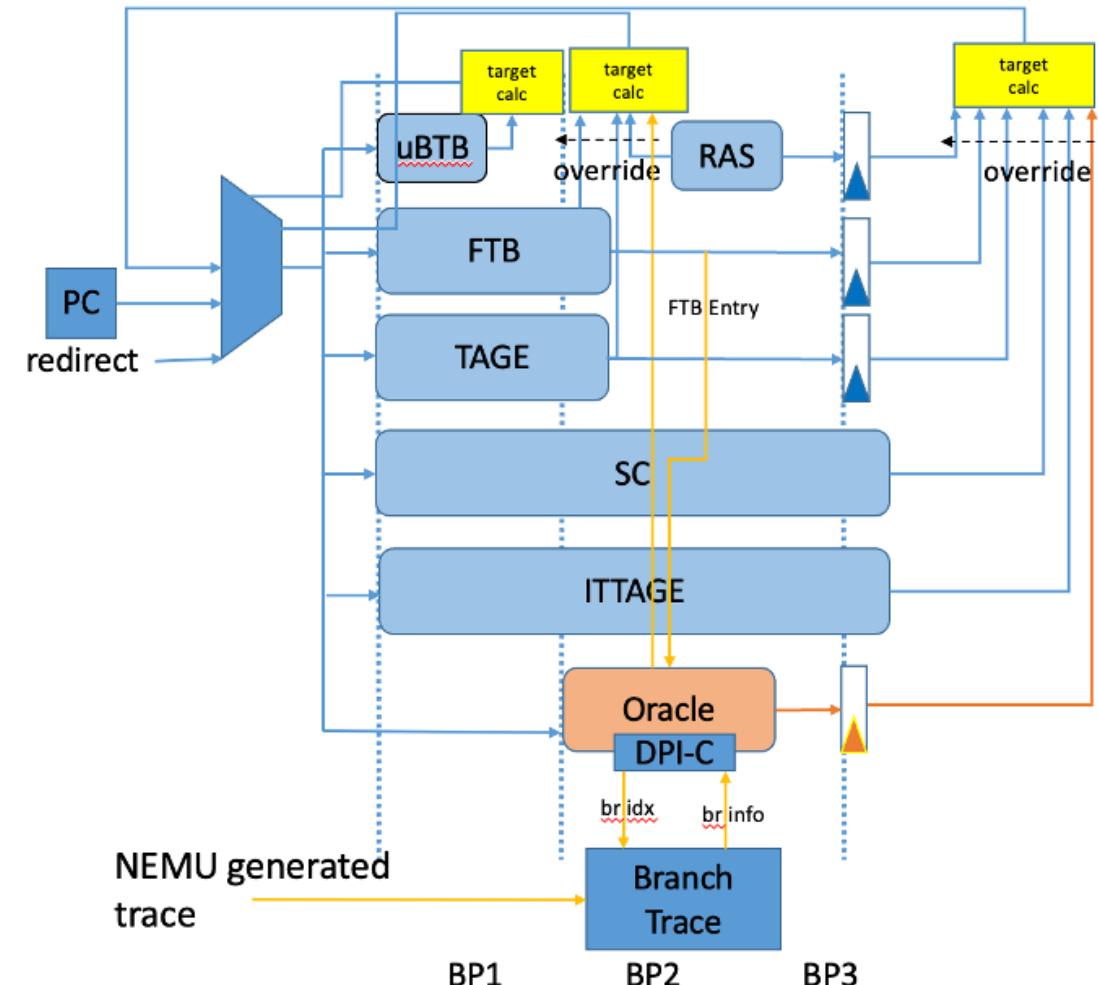
- Generate branch trace information with NEMU emulator
- Correct branch target and taken/not-taken results by real predictors

# Oracle Branch Predictor

- Design/Highlights:
  - **Runahead Trace Generation:** dynamic generation of high precision prediction from mostly synchronized NEMU emulator
  - **LightS<sup>4</sup>:** lightweight snapshot & recovery for runahead



Runahead recovery with LightS<sup>4</sup>



BPU with Oracle Branch Predictor



# Oracle Branch Predictor

- **Hands-on: enable Oracle Branch to achieve higher performance**
- **Process**
  - Configure and compile NEMU Runahead version
  - Compile XiangShan with Oracle Branch Predictor enabled
  - Run SPEC CPU checkpoints
  - Compare performance differences with non-oracle version



# Oracle Branch Predictor

- Hands-on: Enable Oracle Branch to achieve higher performance

```
# Build NEMU with run-ahead feature  
  
$ cd $XS_PROJECT_ROOT && source env-oracle.sh  
  
$ sh $XS_PROJECT_ROOT/tutorial/oracle-nemu-compile.sh
```

```
# export NEMU_HOME=${XS_PROJECT_ROOT}/NEMU-ahead  
  
# cd ${NEMU_HOME}  
  
# make riscv64-xs-ahead-ref_defconfig # Use ahead configuration file  
  
# make -j8 # Generate a dynamic library at build directory  
  
# export NEMU_HOME=${XS_PROJECT_ROOT}/NEMU # Reset NEMU_HOME for difftest
```



# Oracle Branch Predictor

- **Hands-on: Enable Oracle Branch to achieve higher performance**

```
# Time consuming, so we only present commands here  
# Compile XiangShan with Oracle BP enabled  
# make emu EMU_THREADS=8 EMU_TRACE=1 WITH_LIGHTQS=1 CONFIG=MinimalConfig -j8
```

```
# Run spec checkpoints  
$ sh $XS_PROJECT_ROOT/tutorial/oracle-run-and-stat.sh
```

```
# ${XS_PROJECT_ROOT}/tutorial/emu-oracle -i ${XS_PROJECT_ROOT}/tutorial/_492740000000.gz \  
-l 100000 --force-dump-result 2>debug.log  
# Takes about 1 minute
```



# Oracle Branch Predictor

- **Hands-on: Enable Oracle Branch to achieve higher performance**

```
# Automatically executed in the last script  
  
# Compare performance differences  
  
# cat debug.log| grep Bp  
  
# Bp related performance counters reflects the misprediction rate  
  
# cat debug.log| grep clock_cycle  
  
# clock_cycle records the overall cycles spent during simulation  
  
# For significant differences, more instructions need to be executed (at least 1M)
```

# Oracle Branch Predictor

```

Core 0: EXCEEDING CYCLE/INSTR LIMIT at pc = 0x472d0
total guest instructions = 100,000
instrCnt = 100,000, cycleCnt = 255,835, IPC = 0.390877
Seed=0 Guest cycle spent: 255,839 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 47,924ms
Bp Oracle
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpRWrong,           17
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpRRight,          2805
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpInstr,          23024
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpIRight,          3358
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpIWrong,          48
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpCRight,          2794
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpBRight,          14875
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpBWrong,          222
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpRight,          22754
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpIWrong,          0
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpWrong,          270
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpJRight,          4521
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpCWrong,          24
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpBInstr,         15097
Bp Base
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpRWrong,           64
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpRRight,          2753
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpInstr,          23024
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpIRight,          3249
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpIWrong,          157
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpCRight,          2157
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpBRight,          14027
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpBWrong,          1070
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpRight,          21797
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpJWrong,          0
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpWrong,          1227
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpJRight,          4521
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpCWrong,          61
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.frontend.ftq: BpBInstr,         15097
Cycle Oracle
[PERF ][time=          255838] TOP.SimTop.l_soc.core_with_l2.core.ctrlBlock.rob: clock_cycle,      255837
Cycle Base
[PERF ][time=          274386] TOP.SimTop.l_soc.core_with_l2.core.ctrlBlock.rob: clock_cycle,      274385

```

- BpWrong: Total branch misprediction
  - Clock\_cycle: Cycle count since reset
  - Both has been **effectively decreased** with OracleBP
  - BpWrong cannot reach zero because **OracleBP does not correct basic BTB information like start PC, branch PC and branch type**



# Top-Down: Method for Performance Analysis

- Organize scattered performance events in a **hierarchical manner**
- **Accurately** calculate one event's impact on processor performance
- We apply the Top-Down approach to 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
- Due to the limited time, we will not arrange a hands-on section here
  - Please refer to <https://github.com/OpenXiangShan/XiangShan/tree/master/scripts/top-down>



# Top-Down

- Use Top and Backend as examples

Top

Each Instruction Dispatch Slot from Frontend to Backend

Uop Has Been Assigned

Uop Is Finally submitted

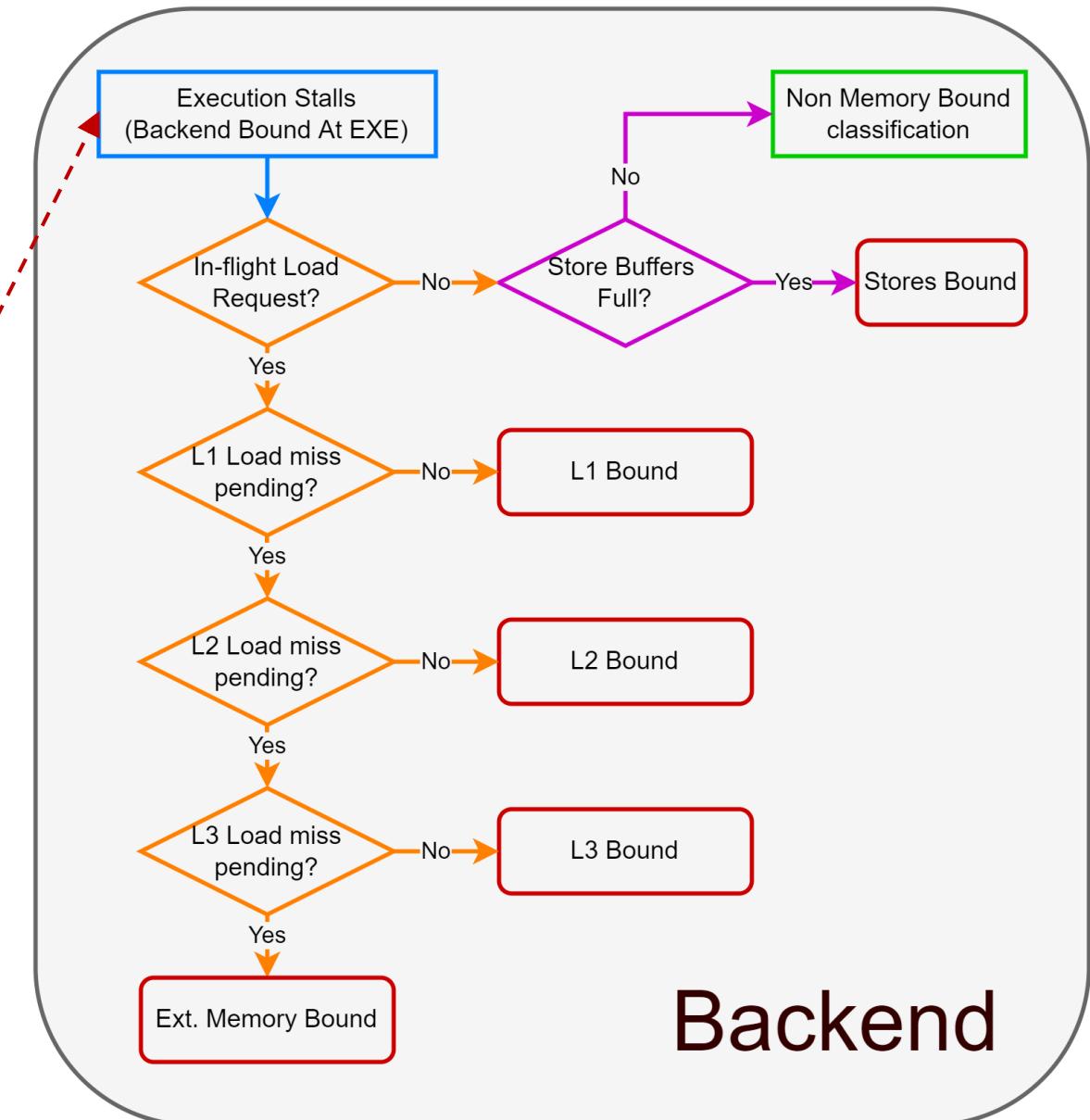
Retiring

Bad Speculation

Backend Is blocking

Backend Bound

Frontend Bound





# Top-Down

- Setting Performance Counters in RTL Code

*ICacheMainPipe.scala*

```
val tlb_miss_vec = VecInit((0 until PortNumber).map(i => toITLB(i).valid && s0_can_go
&& fromITLB(i).bits.miss))
val tlb_has_miss = tlb_miss_vec.reduce(_ || _)
XSPerfAccumulate("icache_bubble_s0_tlb_miss", s0_valid && tlb_has_miss)
XSPerfAccumulate("icache_bubble_s2_miss", s2_valid && !s2_fetch_finish)
```

- Simulation and collecting performance counter data



# Top-Down

- Analyze through Top-Down method

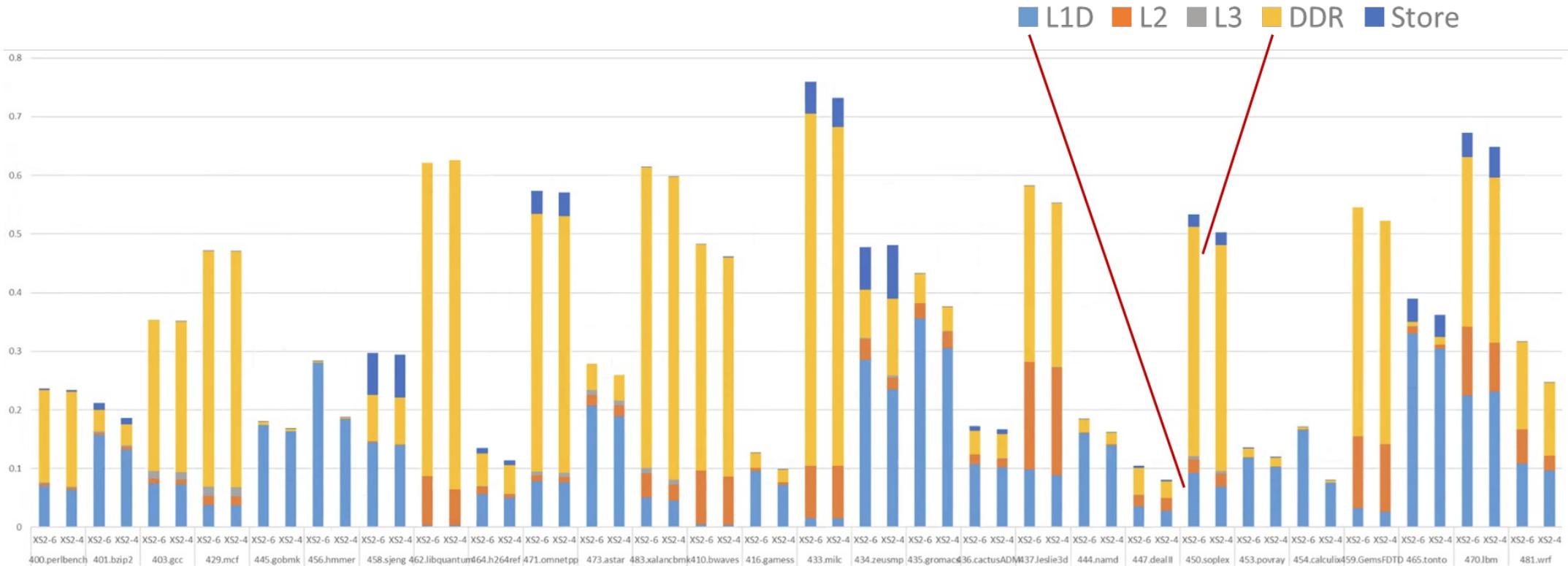
```
top_down.py
```

```
frontend_bound = top.add_down("Frontend Bound", use('decode_bubbles') / use('total_slots'))
fetch_latency = frontend_bound.add_down("Fetch Latency", use('fetch_bubbles') / use('total_slots'))
itlb_miss = fetch_latency.add_down("iTLB Miss", use('itlb_miss_cycles') / use('total_cycles'))
icache_miss = fetch_latency.add_down("iCache Miss", use('icache_miss_cycles') / use('total_cycles'))
```

- Obtain analysis results and make targeted optimizations

# Top-Down

- Example: Top-Down decomposition of memory access factors
- **L1D cache hit read latency** and **DDR read latency** are important bottlenecks





# Cont. Run Simulation

- Switch back to the shell where you compiled the emu
- Run Simulation with the emulator you just compiled

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

---

*Example : (use tab for command completion)*

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

Thanks!