CONDOR
An Andes Company

Cuzco: From Open-Source to High-Performance RISC-V CPU IP

International Workshop on RISC-V for HPC (RISCVHPC)
SC25, St. Louis, November 17, 2025

Shashank Nemawarkar, Senior Director of Architecture

ANDES™
Taking RISC-V® to New Heights

Intro to Condor Computing

- Condor Computing, a wholly owned USA subsidiary of Andes Technology, was founded in 2023 with the goal of creating the highest performance, licensable RISC-V CPU IP in the industry

- We are a tight knit team (~50 engineers), with very light management and overhead, entirely focused on bringing an innovative new micro-architecture to the RISC-V CPU market

- We intend to demonstrate that RISC-V can be competitive in any high-performance computing application, from datacenters to handsets, and up to automotive

Taking RISC-V® to New Heights

Cuzco Processor Summary

- Licensable O-O-O CPU core IP designed for highest end performance application processors
- Innovative, time-based scheduling to eliminate Tomasulo algorithm to achieve similar performance much less power than comparable cores
- Support for up to 8 cores with private L2$s in a coherent cluster with a shared L3$
- Latest RISC-V profile support (RVA23) for maximum software compatibility
- Full support for ISA customization

Taking RISC-V® to New Heights

CONDOR
An Andes Company

RISC-V Ecosystem: SIG Performance Modeling

- Charter:

"…address our community's need for cycle-accurate simulation and aims to establish a common workflow and toolscape to use through the product lifecycle…"

- Interface with other SIGs for common topics

- Some participants: Condor, MIPS, Ventana, SiFive, TensTorrent, Imagination, and individuals

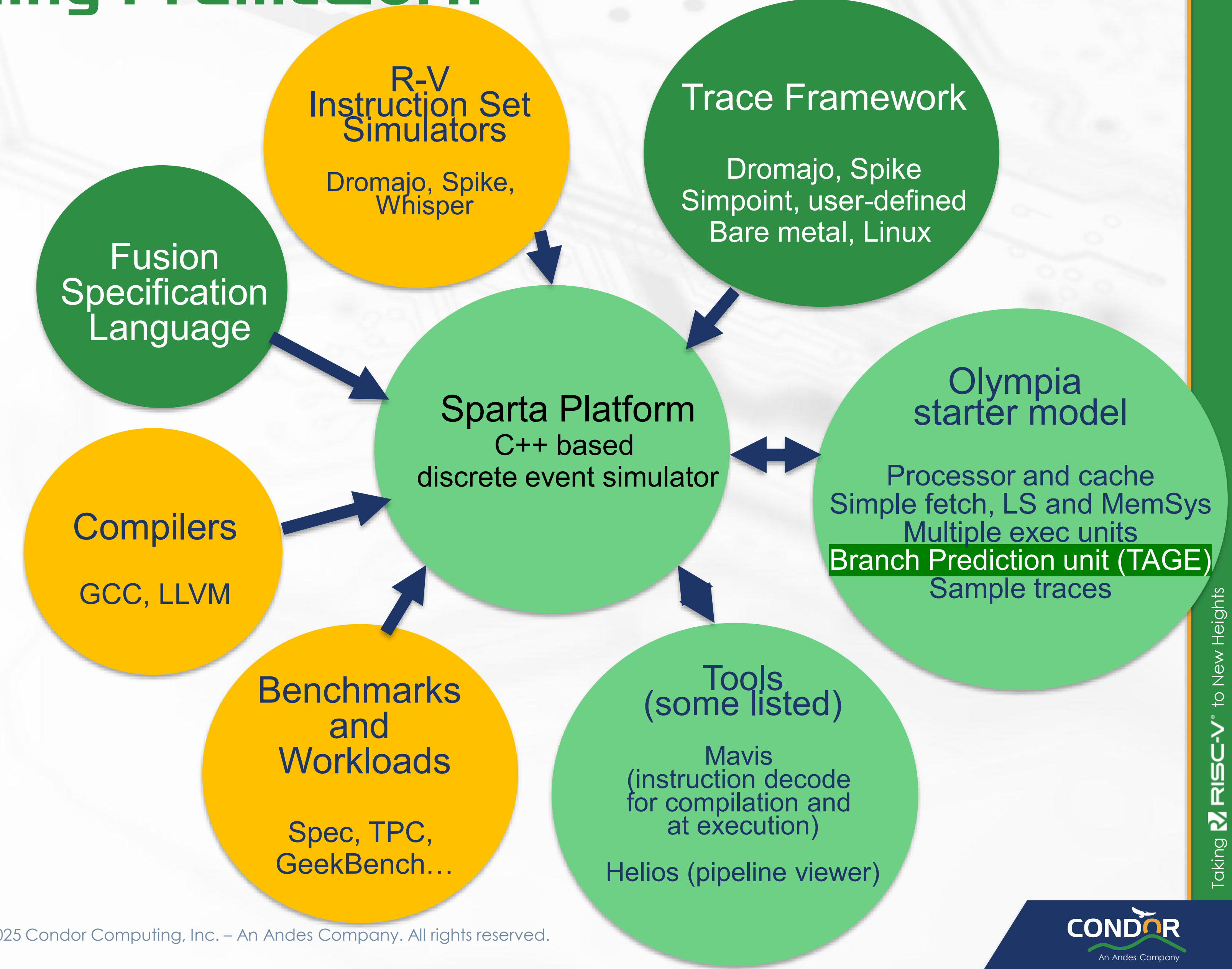
- Confluence: https://riscv.atlassian.net/browse/RVG-9
- Github: https://github.com/riscv-software-src/riscv-perf-model
- Webpage: https://lists.riscv.org/g/sig-perf-modeling

Platform and Modeling Framework

- Provides a basic processor model
  - Enhance per needs
  - Sample traces
  - Statistics* and Visualization (Helios)

Condor team contributions:

  - Sparta (multiple "original" members)
  - Mavis ("original" members)
  - Trace framework enhancements
  - Fusion Specification Language
  - Branch prediction (*TAGE adaptation)
- Creates democratization of high-performance compute research and product design with high quality



RISC-V in HPC and AI/ML: Ecosystem

RV Accelerators

- Clusters: large number of cores
  - Meta (MTIA), Nvidia,
  - Startups: InspireSemi, Calligo Tech
- RISC-V CPUs and/or GPUs
- RISC-V cores as command and control processors

Advantages

- Custom area, power efficient cores
- Standardized ISA extensions
- Custom functions/instructions
- Ecosystem for software stack (RISE)

SIG-Perf-Modeling Covers Single core systems

- Detailed modeling of single core, its memory system and finer components
- Stats generation of finer components, workload analysis (stf, stalls, visuals)

SIG-Perf-Modeling extensions needed for HPC

- Extend open source support from CPU level modeling and analysis to the system
- Performance and power modeling
- Memory system, on-chip/off-chip network design trade-offs
- Plug-and-play modules for the same
- Coherent and non-coherent protocol based analysis

Thanks: Nick Brown, RISC-V Summit North America, October 2025

FSL: Fusion/Fracture Specification Language

- FSL expressions define instruction transforms
  - FRACTURE A **one-to-many** instruction transformation
  - FUSION A **many-to-one** instruction transformation
  - BINARY General binary translation

- *FSL* provides a language, transform toolchain and C++ API
  - A Python interface is built from the shared library

- FSL based tools can generate RTL, perf model methods and compiler MDL
  - RTL (System Verilog): decoders, trace/uop structures, unit computation sites, etc.
  - Perf model methods (C++): documented uses are shown through Sparta/Olympia
  - Compiler output (LLVM): TableGen fusion patterns, predicates, scheduling hooks, etc.

- FSL is ISA agnostic
  - Current examples and documentation focus on RISC-V

Taking RISC-V® to New Heights

CONDOR
An Andes Company

FSL: Transform Syntax Example

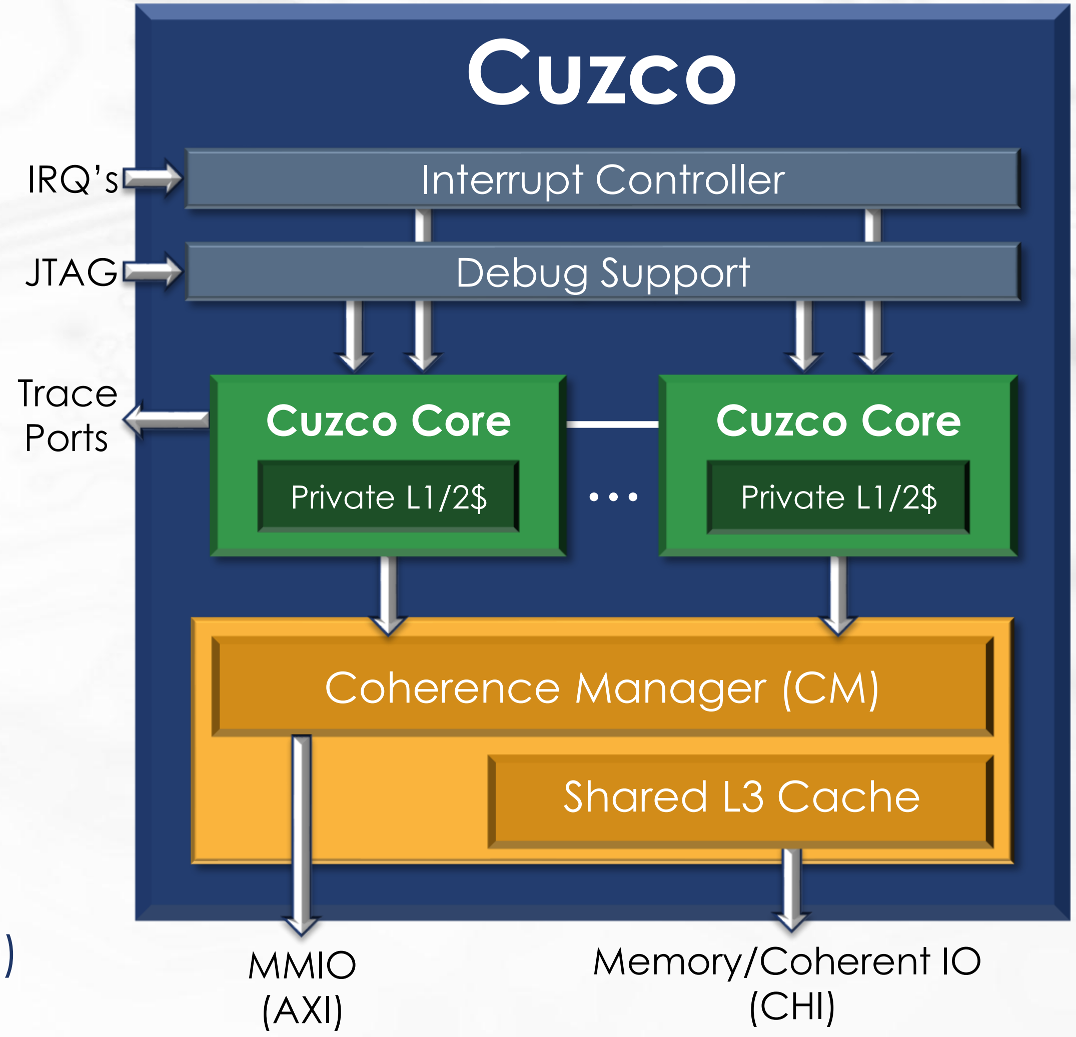
FSL processes instructions windows in three *phases*: each phase has a syntactical expression in the grammar, a clause
1. Identify instruction **sequence**; 2. apply **constraints** to filter tuples; and 3. perform **transformation** to map to new instructions

```
transform uf10
{
  // Prolog elements
  isa    rv64g     // ISA definition object
  uarch oly1       // uArch definition object
  ioput iop1       // API interface object

  // Variables at transform scope
  gpr  g1,g2,g3,g4
  u5   c1
  s12  c2

  // Abstracted instruction sequence
  sequence seq_uf10 {
    sd  g1,c1(g2)
    sd  g3,c2(g4)
  }

 // continued ->
```

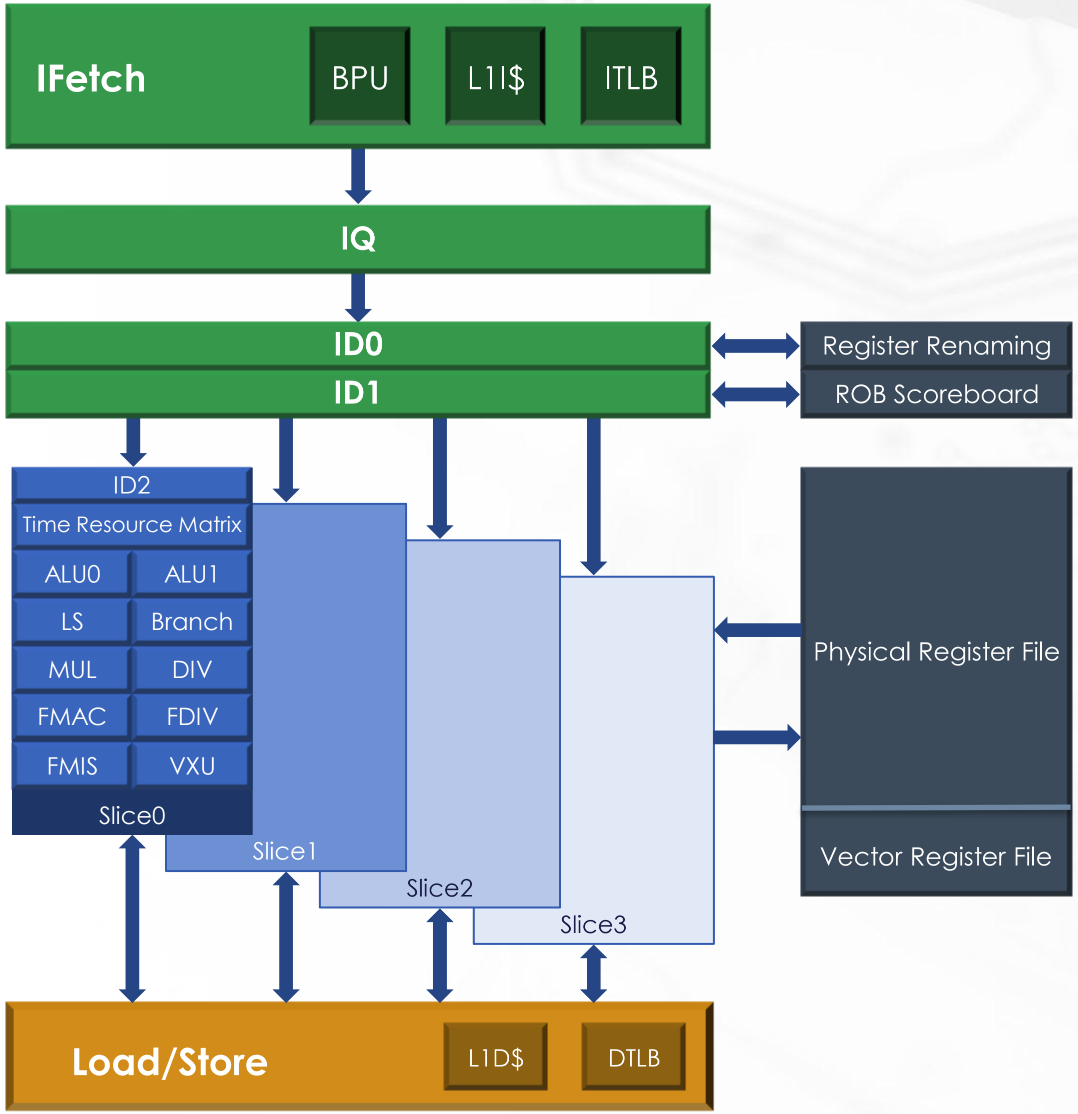
```
  // -> continued

  // Example of a constraint specification
  constraints cns_uf10 {
    g1 != g2
    g3 != g4
    g1 != g4
  }

  // Conversion clause using abstract morphing
  conversion cnv_uf10 {

    // merge sequence into 1 object
    instr instr_uf10.morph(seq_uf1)

    // insert transform into pipeline
    iop1.input.replace(seq_uf10,instr_uf10)
  }
}
```

Cuzco Feature Overview

- RVA23 profile compliant, with Hypervisor
  - 64-bit, RV64GCBKV + CMO
- Innovative time-based microarchitecture
- 12-Stage Pipeline
- 8-Wide Frontend Decode
- 256-Entry Reorder Buffer (ROB)
- 8 Execution Pipelines
- RISC-V Vector 1.0 + Crypto, 256/512b VLEN
- Branch Target Buffer-way predicted (BTB)
- TAGE-SC-L & Tournament Branch Predictor
- 1K/2K/4K 4-Way L2 TLBs
- 64 KB, 8-Way Private I/D Caches
- Up to 8 MB, Private L2$
- SECDED ECC error protection
- 8-Core Multiprocessor w/ Shared L3$ (up to 256MB)
- 256/512-bit CHI and 64b/512b MMIO Buses



Cuzco CPU Core Block Diagram



Taking RISC-V® to New Heights

CONDOR An Andes Company

Cuzco: Vector Everywhere

O-O-O Scalar/Vector:

- Compilers making more use of RVV:
  - vector/scalar intermingled code
- Gains from "vector everywhere":
  - frequent, tight lower latency interactions and power-conscious resource sharing
- Unlike traditional wide-vector, decoupled designs:
  - no need for dense, pure vector library code to be performant and cost effective

Excels at:

- Capturing enhanced data bandwidth opportunities general purpose applications
- AI transformer inferencing (quantization, dot product and matmul)
- Bursts of parallel element computation with scalar/vector ILP

Spec2K17, XZ benchmark:

```
...
   1ed78: sub   x9, x31, x8
   1ed7c: add   x22, x21, x8
   1ed80: add   x23, x12, x8
   1ed84: vsetvli   x24, x9, e8, m2, ta, ma
   1ed88: vle8.v    v8, (x22)
   1ed8c: vle8.v    v10, (x23)
   1ed90: vmsne.vv  v12, v8, v10
   1ed94: vfirst.m  x23, v12
   1ed98: c.mv  x22, x24
   1ed9a: blt   x23, x0, 0x1eda0 <bt_find_func+0xd4>
...
```

bt_find_func(): a byte-by-byte string comparison to find matches in a binary tree

Taking RISC-V® to New Heights

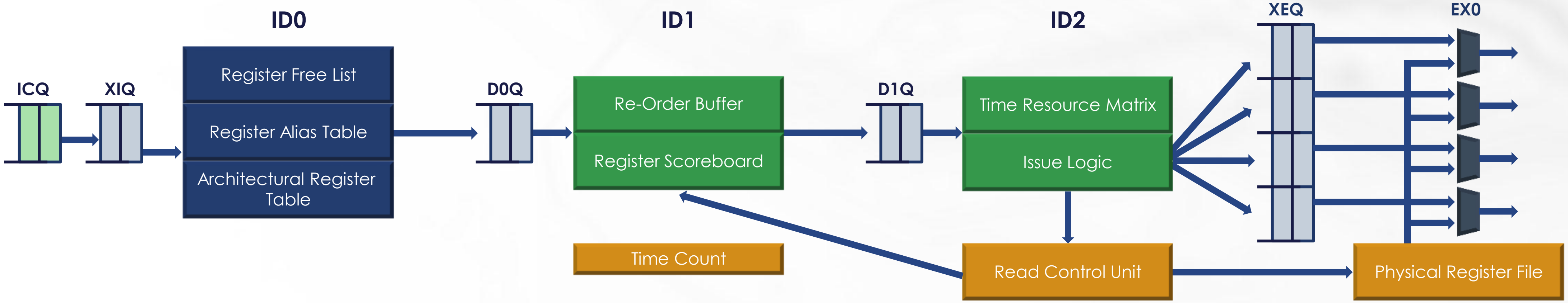
CONDOR An Andes Company

Cuzco Core Pipeline

IF0	IF1	IF2	IF3	ID0	ID1	ID2	EX0	EX1/DC1	DC2	DC3	DC4
Calculate next address & access iTLB	Access IC tag array, IC(hit/miss) BTB, and GHT, TAGE-SC-L	Access IC data array access, BTB hit/miss and taken/non-taken	Write IC cache line to ICQ & bypass/ read N instructions to XIQ	Read N instrs from XIQ, 1st instr decode, & access RFL & RAT	Access RSB to calculate execution times, for each instruction, and create dependen ce chains	Access TRM to issue instruction, secondary instruction decode, write issued instructions to XEQ	Read RF/fwd data, check RSB, send instr from XEQ to functional unit	Execute instrs, write result to PRF. AGU and access dTLB	Access DC tag array (DC hit/miss)	Access DC data array	Align and send load data to write back to PRF

Cuzco Time-Based Microarchitecture

- 1st CPU designed with hardware compilation for optimal instruction sequencing
- Schedule instruction execution with a perfect view of all past instructions
  - Register Scoreboard: record write time of an instruction to a register which becomes the read time of dependent instruction
  - Time Resource Matrix (TRM): busy indicators for read & write buses, and other resources
  - Issues instructions with precise predicted future times for execution
  - Reschedule/replay: Account for dynamic latencies, resource conflicts
- Reduces scheduling complexity, timing, area and power of typical OOO cpu mid-cores



Time Resource Matrix (TRM): Performance Model

Time	Scheduled		Read ports				Write ports		Br	ALU		LSU
83	O	P	A	A	J							
84	Q	R	L				J		A	J		
85	S	T	N				M		M			L
86	U	V	H	I	T		N			N		
87	W	X	K	K			H	I		H	I	T
88	Y	[illegible]	B	B			K			K		
89	a	b	C	C	D	D	B			B		
90	c	d	E	E	F	F	D		C	D		
91	e	f	G	G	P		F		E	F		
92	g	h	U						G			P
93	i	j	O	O	V	c						U
94	k	l	X				O	c		O	c	V

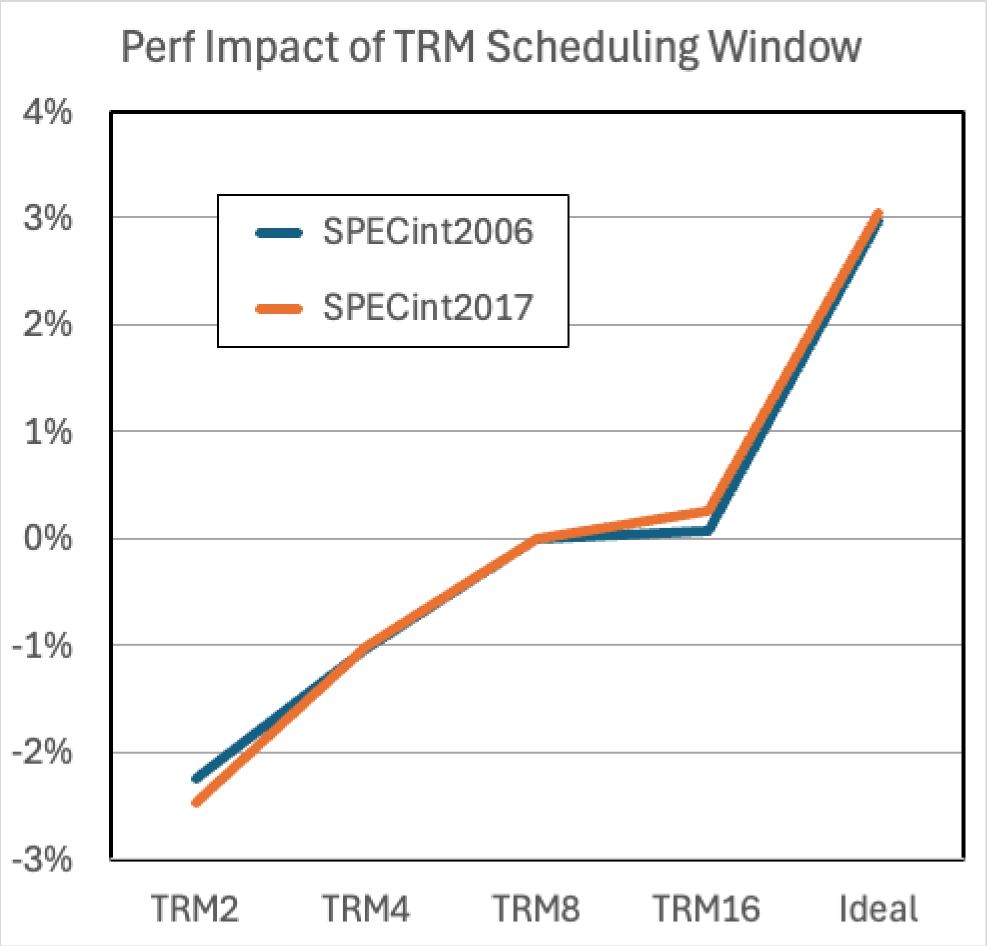
- Instructions are efficiently scheduled in the TRM to take advantage of available resources
- In this example snippet from Dhrystone, instruction O is scheduled in cycle 83, issued in cycle 93, and executed in cycle 94

Taking RISC-V® to New Heights

CONDOR
An Andes Company

Time-Based Microarchitecture: Design Selection

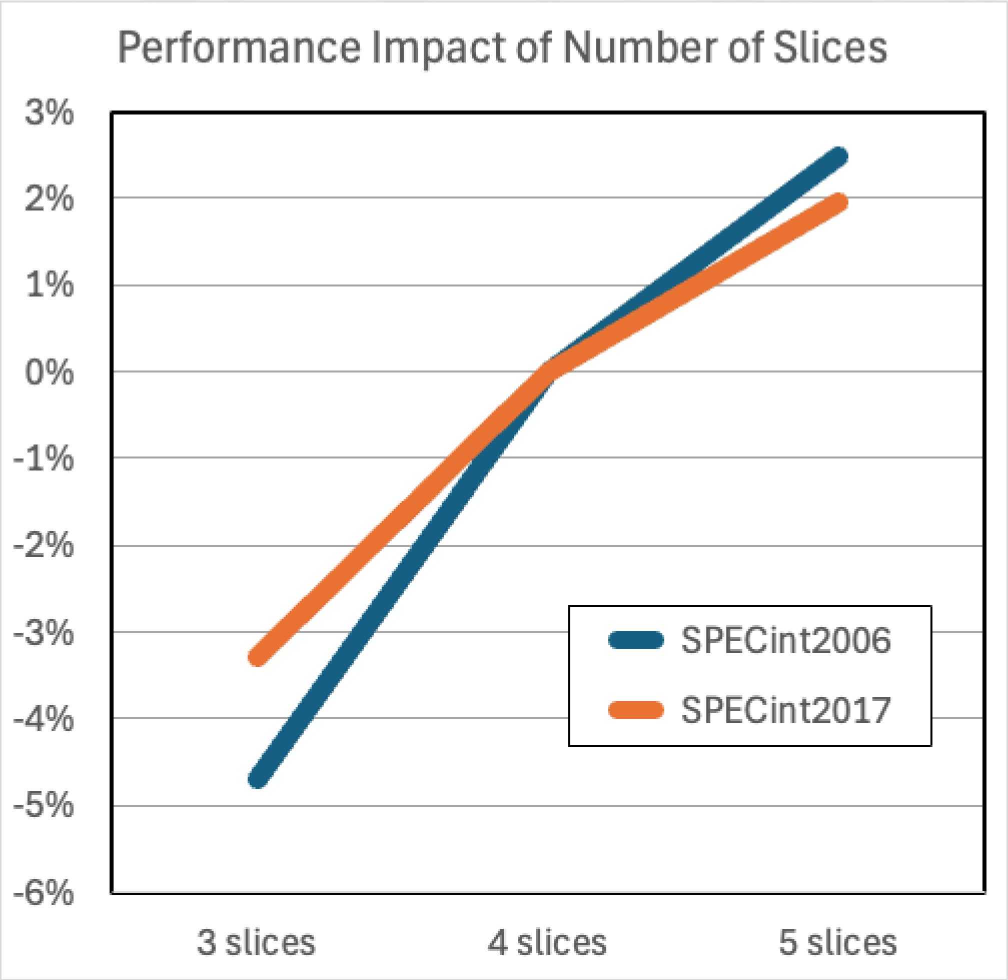
- Build execution schedule with known & projected dependencies and latencies
  - Activate operands, resources, and execution units only at scheduled times
  - No need to search or prioritize selection at reservation stations
  - Reduced scheduler complexity (logic & area)
  - Dynamic power reduction vs conventional scheduler
- Two instructions scheduled/slice/cycle, s calable with slices
  - TRM2: Identify functional unit slots within 2 cycles of operand availability
  - TRM8: Within the next 8 cycles
  - Ideal: Idealized scheduling



Taking RISC-V® to New Heights

Slice-based Microarchitecture: Config Selection

- Baseline: 4 slices, 2 instruction pipelines/slice
- Uniformly scalable design IP
  - Each slice implements a fully compatible RISC-V CPU
  - Each slice adds symmetric set of resources to the machine
- Static PPA control through IP configuration
- Dynamic power vs. throughput control with slice-based enables and scheduling

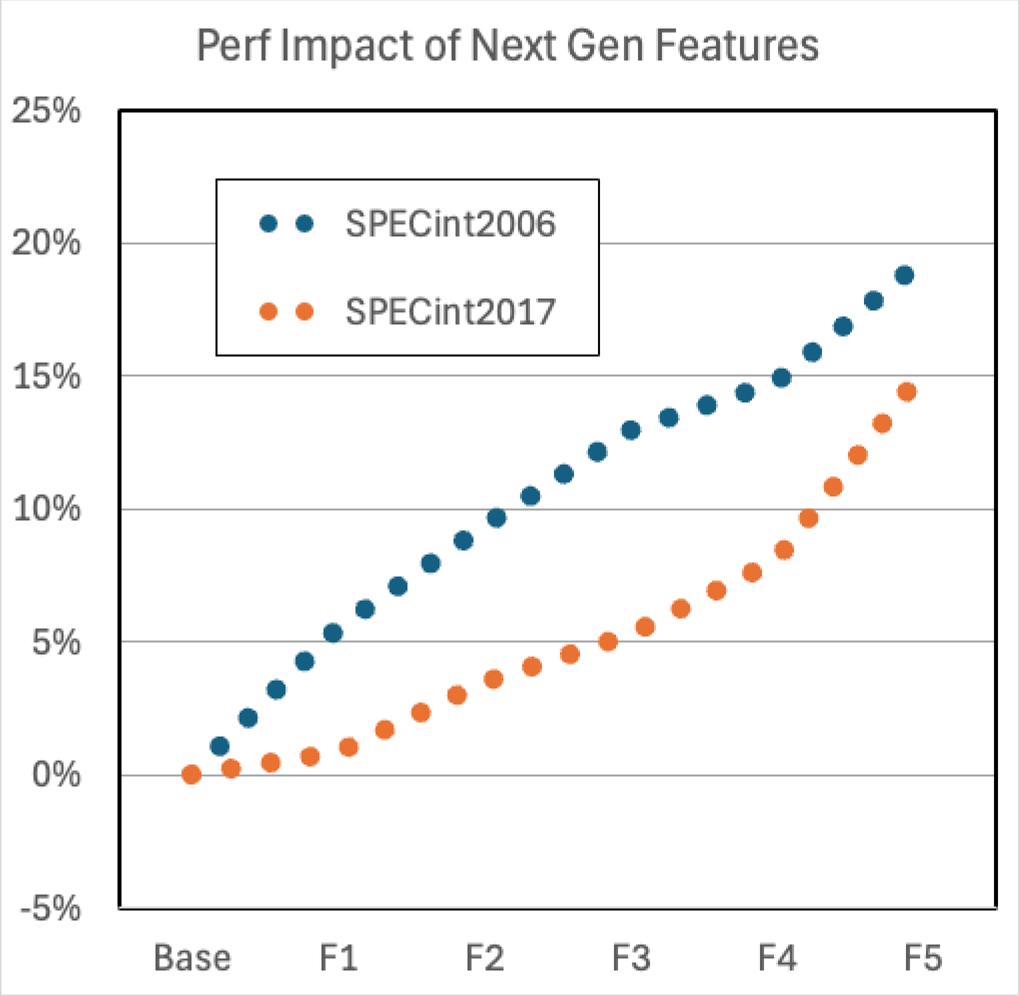


Taking RISC-V® to New Heights

CONDOR
An Andes Company

Future Improvements

- Cycle-accurate performance model allows projecting impact of features implementable in the next generation
- This chart shows the impact of successively applying the first 5 of these features



Comparison with Andes O-O-O Processor

CPU Cores	Andes AX65	Cuzco
Pipeline stages	13	12
Issue width	4	8
ROB	128	256
Branch prediction	TAGE-L	TAGE-SC-L-IT
Int ALU & FPU Load/Store Unit	4 & 2 2 L/S	8 & 4 4 L/S
L1 Caches	I$: 64KB D$: 64KB	I$: 64KB D$: 64KB
L2 Cache	8MB Shared	Up to 8MB Private
L3 Cache (Shared)	None	Up to 256MB
DMIPS/MHz	4.82	8.5
SPECint2k6/GHz	8.78*	>17.5*

with 8MB L2 and no vector ISA usage

Taking RISC-V® to New Heights

CONDOR
An Andes Company

Cuzco Memory System Microarchitecture

Private L1/L2 Caches

- L1: 64KB I$/D$, 8-way
- I$/D$ L1/L2 prefetch and D$ writearound
- L2: Up to 8MB, up to 16-way
- L2: Configurable multi-cycle SRAM accesses
- Configurable up to 64 outstanding requests

Shared L3 Cache, Coherent Cluster

- Up to 256MB, up to 16-way,
- Up to 8 cores+L2$s in a cluster

Bus Interfaces (synchronous and asynchronous)

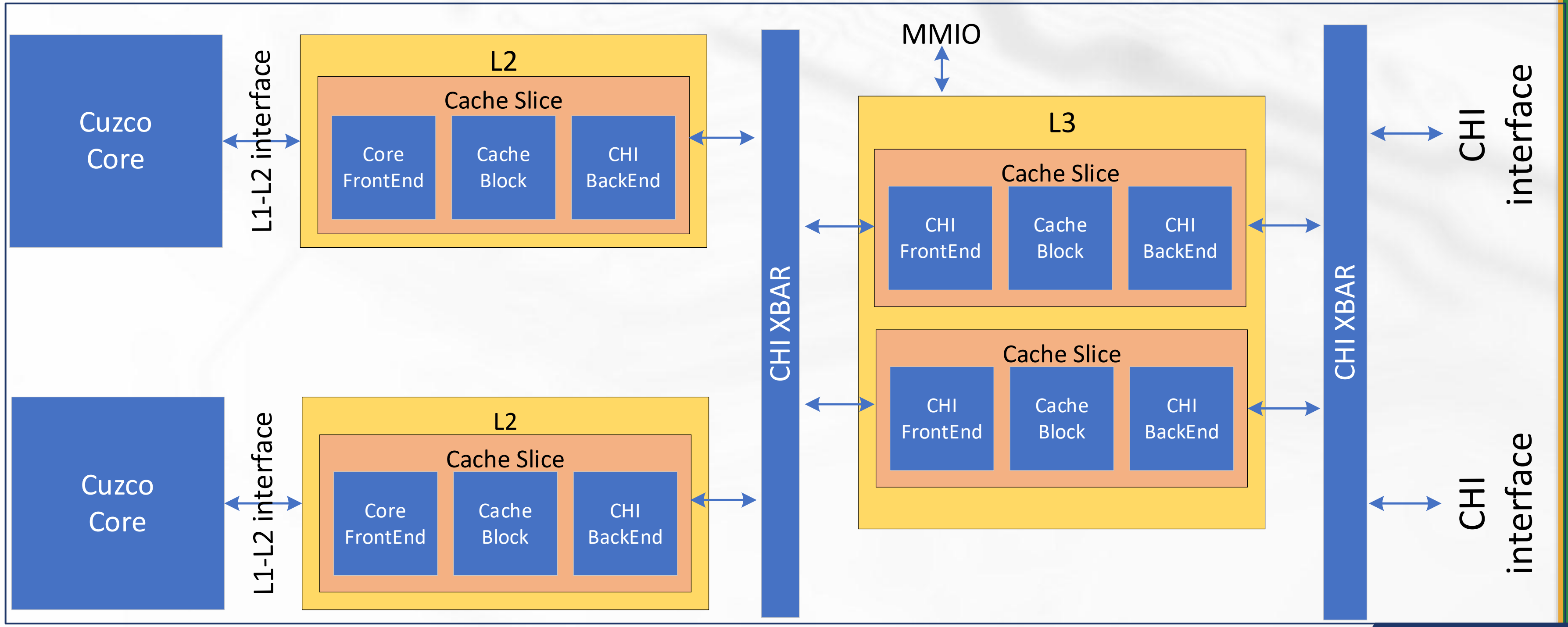
- 512-bit main memory CHI bus interface
- 256-bit memory mapped I/O (MMIO) interface

Privilege Modes

- Machine (M), hypervisor (H), supervisor (S) and user (U)

Memory Management Unit (MMU)

- Bare, Sv39, Sv48, and Sv57 VA



Cuzco in RISC-V Ecosystem

- **With RISC-V SIG Perf Modeling: open source tools/infra**
  - From scratch to full fledged cycle-accurate model and tools significantly impacting design within 2 years
  - Contributions back to open-source tools/infra
- **Cuzco: Industry standard RISC-V ISA Compliant Design**
  - Better performance / $
  - Better performance / µW of power
  - Fully MP-ready: up to 8 HPC CPU cores per cluster
  - RVA23 profile
  - Extensible
- **Approach also used by multiple RISC-V teams**

Acknowledgments

The team at Condor Computing wishes to thank:

- Andes Technology Corporation, for their support:
  - Frankwell Lin, Chairman & CEO
  - Dr. Charlie Su, CTO & President
  - Emerson Hsiao, President Andes Technology USA

- Dr. Thang Tran, for his fundamental contributions to the invention of the time-based scheduling microarchitecture

Thank you!

Shashank Nemawarkar

shashank.nemawarkar@condorcomputing.com
www.condorcomputing.com

Taking RISC-V® to New Heights

FSL: Processing and Syntax

- FSL processes instructions windows in three *phases*
  - Sequence, constraints and transform

- Each phase has a syntactical expression in the grammar, a clause
  - **Sequence clause**
    - Action: Identifies instruction sequences matching specified attributes
    - Eg: ADD/ADD or ADD/ADD/SUB or LD/ST/BR, etc., wild cards are supported
  - **Constraints clause**
    - Action: filters sequences for transformable tuples, returns sequences
    - Eg: Operand limits (PRF ports), machine state constraints, e.g. busy/available computation sites
  - **Transform clause**
    - Action: map instructions to new instruction(s), adjust operands, insert transform into pipe
    - Eg: Fuse ADD/ADD/SUB into MAGIC_AAS, adjust operands, and dispatch to the correct execution queue

FSL: Transform Syntax Example

```
transform uf10
{
  // Prolog elements
  isa    rv64g     // ISA definition object
  uarch oly1       // uArch definition object
  ioput iop1       // API interface object

  // Variables at transform scope
  gpr  g1,g2,g3,g4
  u5   c1
  s12  c2

  // Abstracted instruction sequence
  sequence seq_uf10 {
    sd  g1,c1(g2)
    sd  g3,c2(g4)
  }

 // continued ->
```

```
 // -> continued

 // Example of a constraint specification
 constraints cns_uf10 {
   g1 != g2
   g3 != g4
   g1 != g4
 }

 // Conversion clause using abstract morphing
 conversion cnv_uf10 {

   // merge sequence into 1 object
   instr instr_uf10.morph(seq_uf1)

   // insert transform into pipeline
   iop1.input.replace(seq_uf10,instr_uf10)
 }
}
```

Taking RISC-V® to New Heights

CONDOR An Andes Company

FSL: More Information

- **We continue to develop FSL and utilities for analysis and generation**
  - STF and Simpoint based instruction stream mining utilities
  - Automation for performance model stats to fusion/fracture candidates expressed in FSL
  - Application examples for RISC-V vector ISA
  - Compiler build automation and output comparison utilities
  - Interaction/integration with other open-source analysis tools

- **FSL available at the Condor Computing git repo**
  - https://github.com/condorcomputing/fsl

- **FSL in RISC-V ecosystem**
  - Olympia usage example https://github.com/riscv-software-src/riscv-perf-model
  - FSL vs Python: https://github.com/riscv-software-src/riscv-perf-model/discussions/121

Taking RISC-V® to New Heights

CONDOR
An Andes Company

Cuzco Memory System Microarchitecture

Private L1 Caches

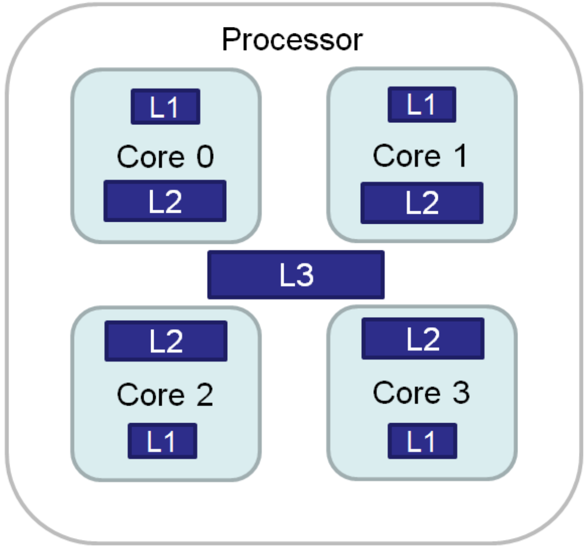
- 64KB I$/D$, 64B line size, 8-way, pLRU
- PIPT (Physical Index and Physical Tag)
- 64-byte cache line size
- I$/D$ prefetch and D$ writearound
- SECDED ECC error protection
- Up to 64 pending miss requests
- 4 cycle load->use penalty

Private L2 Cache

- Up to 8MB, 64B line size, up to 16-way, pseudo-random replacement
- I/D prefetch
    - Preset and configurable prefetch policies
- Configurable multi-cycle SRAM accesses
- SECDED ECC error protection
- +14 cycle delay on L2 hit

Privilege and Memory Management

- Machine (M), hypervisor (H), supervisor (S) and user (U) privilege modes
- Memory management unit (MMU)
    - Bare, Sv39, Sv48, and Sv57 VA translations
    - Svnapot, Svpbmt, Svinval VM extensions
    - L1 I/D TLBs: 64-entry, fully associative
    - L2 TLB: up to 4K-entry, 4-way
    - PMP and ePMP support with 16 PMP entries
- 16 PMA regions



Cuzco: Cluster Microarchitecture

Shared L3 Cache

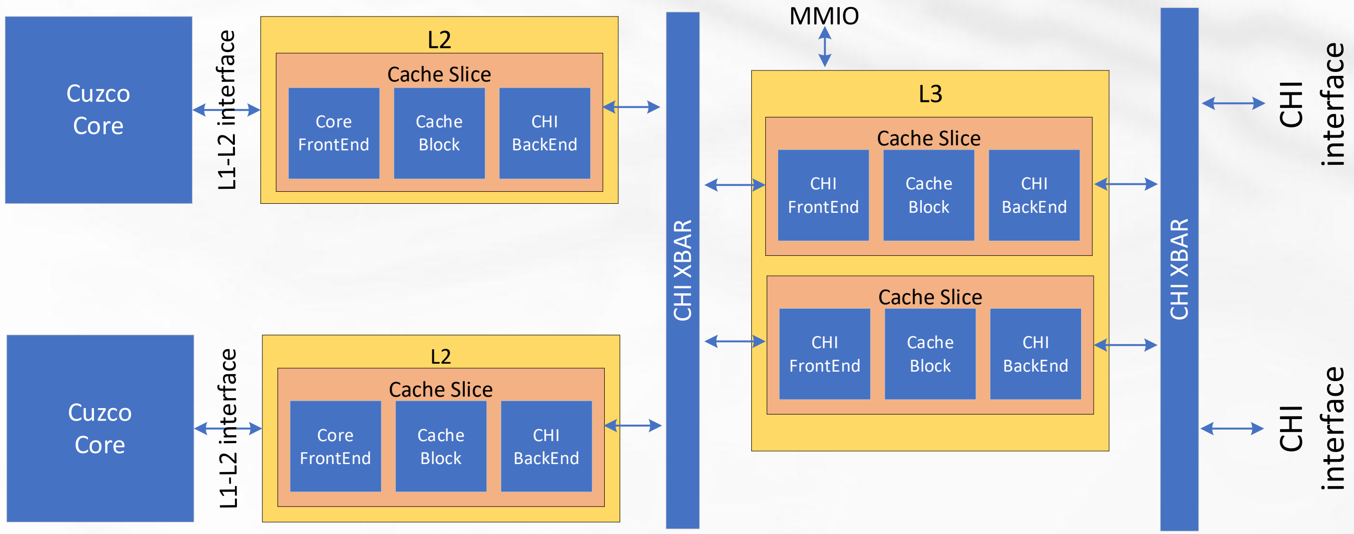
- Up to 256MB, 64B line size, up to 16-way, pseudo-random replacement
- I & D prefetch, configurable policies
- Max outstanding reads/writes (cacheable and uncacheable): 32-128
- Max 64 outstanding snoop transactions
- Configurable multi-cycle SRAM accesses
- SECDED ECC error protection

Cluster with Multicore Cache Coherency

- Up to 8 cores+L2$s in a cluster
- Coherence Manager and L3$

Bus Interfaces

- 512-bit main memory CHI bus interface
- 256-bit memory mapped I/O (MMIO) interface
- Core+L2 vs. external-bus clock
- Asynchronous, and Synchronous N:1 clock ratios



Analysis Example: Identifying the Workload

Identify the workload: feature **ir1:**

xalancbmk (SpecInt2K6) and **x264_s (SpecInt2K17)**

Benchmark	ir1
si06	-0.22%
astar	-0.00%
bzip2	-0.22%
gcc	0.20%
gobmk	-0.16%
h264ref	-0.05%
hmmer	-0.14%
libquantum	-0.33%
mcf	0.01%
omnetpp	-0.02%
perlbench	0.04%
sjeng	-0.02%
xalancbmk	-1.96%

Benchmark	ir1
si17	-0.26%
deepsjeng_s	-0.00%
exchange2_s	0.02%
gcc_s	-0.13%
leela_s	-0.02%
mcf_s	-0.00%
omnetpp_s	-0.03%
perlbench_s	-0.02%
x264_s	-2.29%
xalancbmk_s	-0.14%
xz_s	0.02%

Benchmark	ir1
xalancbmk	-1.96%
0001558	-8.48%
0001559	-0.03%
0001560	-18.78%
0001561	0.01%
0001562	-0.88%
0001563	-0.56%
0001564	7.93%
0001565	-3.76%
0001566	0.02%
0001567	-0.04%
0001568	-1.03%
0001569	-0.15%
0001570	-0.11%
0001571	0.05%
0001572	-2.12%
0001573	-0.07%
0001574	-0.24%

One workload each in SpecInt2K6 and SpecInt2K17

Single outlier trace **1560** in xalancbmk

Look for stalls at interfaces:

Front end, mid-core, LS, MemSys

stats	% diff
rob.ipc	-18.78%
fetch.fet_stall_pct_not_stalled	-8.00%
fetch.fet_stall_pct_no_xiq_credits	-11.39%
fetch.fet_stall_pct_fg_br_mispred	19.12%
fetch.pred_ind_correct	-37.86%
fetch.pred_ind_incorrect	17701300.00%

- Front end stall analysis
- ipc -19% drop,
- indirect br mispred +177K times!

- Feature “ir1” is midcore design alternative is unrelated to branches or predictions
- A perfect indirect predictor recovered the ipc loss completely

Taking RISC-V to New Heights

CONDOR An Andes Company

Analysis Example Part I: Identifying the Workload

- We see that feature **ir1** has small negative impact overall (**si06** and **si17** lines), but this is almost entirely due to two benchmarks, **xalancbmk** and **x264_s**
- We examine the SimPointed traces that make up **xalancbmk** and see that there is a single trace, **1560**, which is a large outlier.
  - When we examine **x264_s** traces (not shown here), we see that the performance impact is more broad-based and affects all traces in this benchmark
- We therefore start by examining trace **1560**

Benchmark	ir1
si06	-0.22%
astar	-0.00%
bzip2	-0.22%
gcc	0.20%
gobmk	-0.16%
h264ref	-0.05%
hmmer	-0.14%
libquantum	-0.33%
mcf	0.01%
omnetpp	-0.02%
perlbench	0.04%
sjeng	-0.02%
xalancbmk	-1.96%

Benchmark	ir1
si17	-0.26%
deepsjeng_s	-0.00%
exchange2_s	0.02%
gcc_s	-0.13%
leela_s	-0.02%
mcf_s	-0.00%
omnetpp_s	-0.03%
perlbench_s	-0.02%
x264_s	-2.29%
xalancbmk_s	-0.14%
xz_s	0.02%

Benchmark	ir1
xalancbmk	-1.96%
0001558	-8.48%
0001559	-0.03%
0001560	-18.78%
0001561	0.01%
0001562	-0.88%
0001563	-0.56%
0001564	7.93%
0001565	-3.76%
0001566	0.02%
0001567	-0.04%
0001568	-1.03%
0001569	-0.15%
0001570	-0.11%
0001571	0.05%
0001572	-2.12%
0001573	-0.07%
0001574	-0.24%

Analysis Example Part 2: Looking at Stalls

- For trace 1560, we compare the stats between the baseline and the feature in question
- We confirm that the IPC has gone down nearly 19% for this trace.
- We first look at the fetch stall statistics, which shows how the fetch slots are distributed
  - **not_stalled** – this is the number of fetch slots that are not stalled (used for sending instructions to the midcore)
    - The higher this number, the better
  - **no_xiq_credits** – this is the number of fetch slots that are stalled due backpressure from the midcore
    - Note that midcore backpressure has gone down 11%
  - **fg_br_mispred** – this is the number of fetch slots that are stalled due to branch mispredicts
    - The stalls due to branch mispredicts has gone up 19%, which is surprising because our feature should not affect branch prediction rates
- We then examine our prediction rates. We see that the number of indirect target predictions has gone through the roof. We have almost no indirect target mispredicts in the baseline, but a significant number when the feature is enabled
- Conclusions:
  - We must investigate further to determine the source of instability in the modeled indirect target predictor, which is unrelated to our feature
  - In the meantime, we experiment by setting indirect target prediction to **perfect** in the model. We see that the **xalancbmk** performance difference has disappeared, although **x264_s** is unaffected and must still be analyzed

```
stats                               % diff
-----                               ------
rob.ipc                             -18.78%

fetch.fet_stall_pct_not_stalled     -8.00%
fetch.fet_stall_pct_no_xiq_credits  -11.39%
fetch.fet_stall_pct_fg_br_mispred   19.12%

fetch.pred_ind_correct              -37.86%
fetch.pred_ind_incorrect            17701300.00%
```

Analysis Example Part 3: Other Stalls

- In a different example, we analyze a 67% drop in IPC
  - Used fetch slots has gone down, but this is almost entirely due to downstream backpressure
    - `fet_stall_pct_no_xiq_credits +23%`
  - We see that used rename slots has gone down, almost entirely due to backpressure from the LSU
    - `ren_stall_pct_no_lsu_ldb_credits +22%`
  - Downstream of rename, we see that the used dispatch slots has gone down, but it's primarily due to upstream stalls (from rename)
    - `dp_stall_pct_d1q_empty +32%`
- Our problem is that we are mis-scheduling instructions, filling up the execution units with poorly-scheduled instructions that must be rescheduled
  - This is confirmed by the number of TRM reschedules
  - `num_trm_rescheduled +1745%`
  - Our baseline configuration avoids this problem

stats	% diff
rob.ipc	-67.22%
fetch.fet_stall_pct_not_stalled	-21.67%
fetch.fet_stall_pct_no_xiq_credits	22.90%
rename.ren_stall_pct_not_stalled	-21.67%
rename.ren_stall_pct_no_lsu_ldb_credits	22.35%
sl_mgr.dp_stall_pct_not_stalled	-21.67%
sl_mgr.dp_stall_pct_trmc_busy_overflow	-23.17%
sl_mgr.dp_stall_pct_alu_xeq_full	10.98%
sl_mgr.dp_stall_pct_br_xeq_full	10.75%
sl_mgr.dp_stall_pct_d1q_empty	32.15%
sl_mgr.slice0.num_trm_rescheduled	1745.00%

Taking RISC-V® to New Heights

CONDOR
An Andes Company