基于RISC-V构建
领域专用芯片设计的通用基座



Our *ball* Team

buckyball



Zhe Jiang (江哲)

Young Chief Professor of
Southeast University



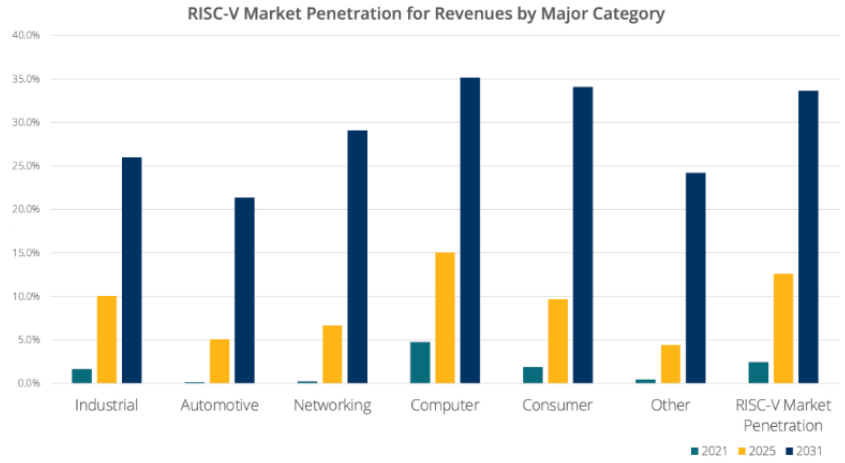
shirohasuki (王晖)

Buckyball & RuyiAI Maintainer



https://github.com/DangoSys/buckyball

RISC-V Drives Today's DSAs

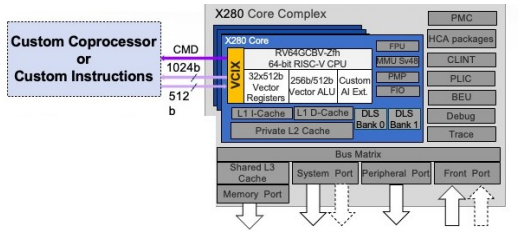


RISC-V revenue penetration, 2021 → 2031

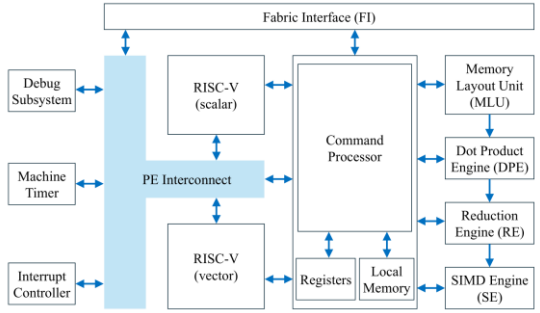
2.5% → 33.7%

Share of RISC-V chip revenue from AI accelerators, 2031

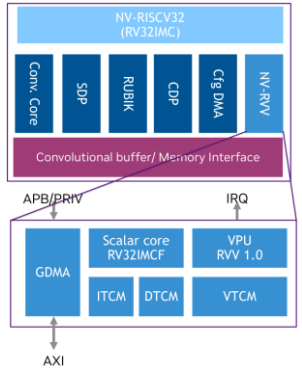
~70%



Google TPU
SiFive X280 drives the matrix unit



Meta MTIA
RISC-V cores in every PE



NVIDIA
~1B RISC-V cores shipped in 2024



Tenstorrent Blackhole
RISC-V CPU cluster on chip

Data: The SHD Group RISC-V market reports (forecasts) Images: SiFive • Meta, ISCA 2025 • Tenstorrent • NVIDIA, RISC-V Summit 2024

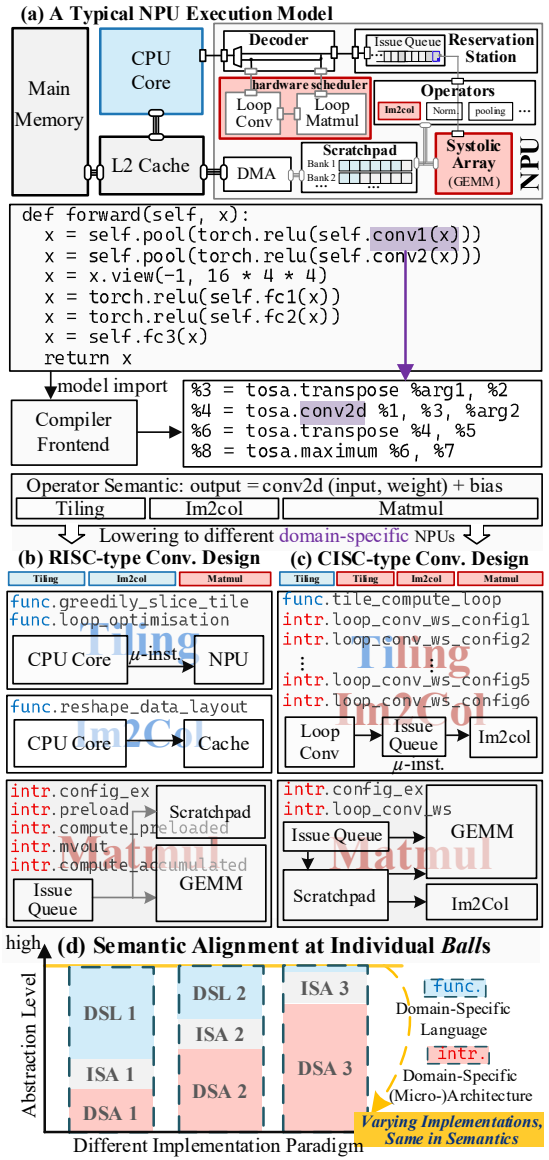
A DSA Is Far More Than an Accelerator

buckyball

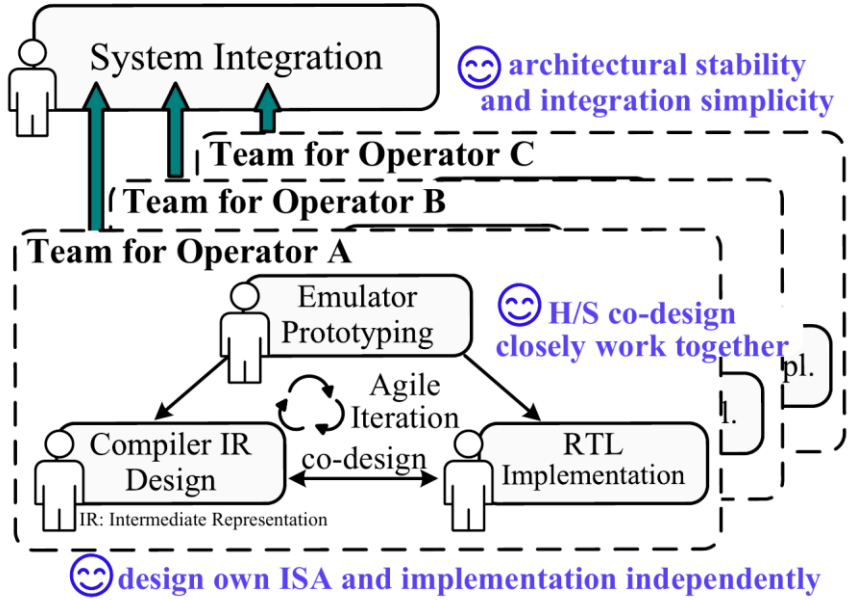
Part of Full Stack	Google	Meta	Tenstorrent	NVIDIA
Serving				
Runtime				
Simulation & Evaluation				
Verification				
Compiler	XLA	Triton-MTIA	TT-Forge	TensorRT
Your ISA + accelerator	MXU	MTIA PE	Tensix	DLA

A different ISA means a whole new stack, rebuilt from scratch.

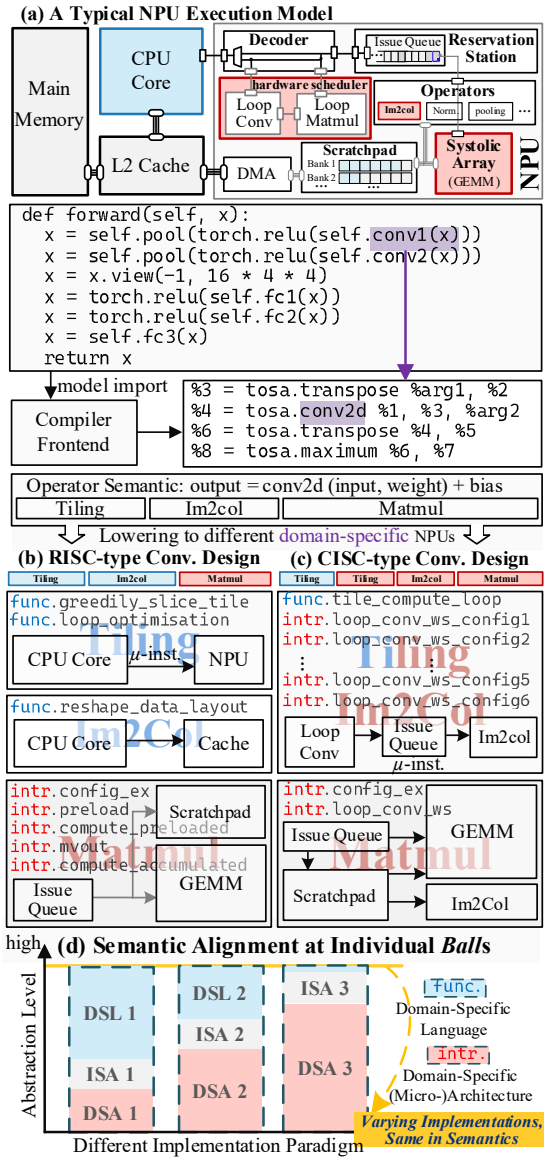
A Stable Architecture Abstraction



- **Implementations are scattered across HW & SW**
- **ISAs are fragmented, but semantics are stable**
- **Eliminate the HW/SW boundary, we get a stable architecture**



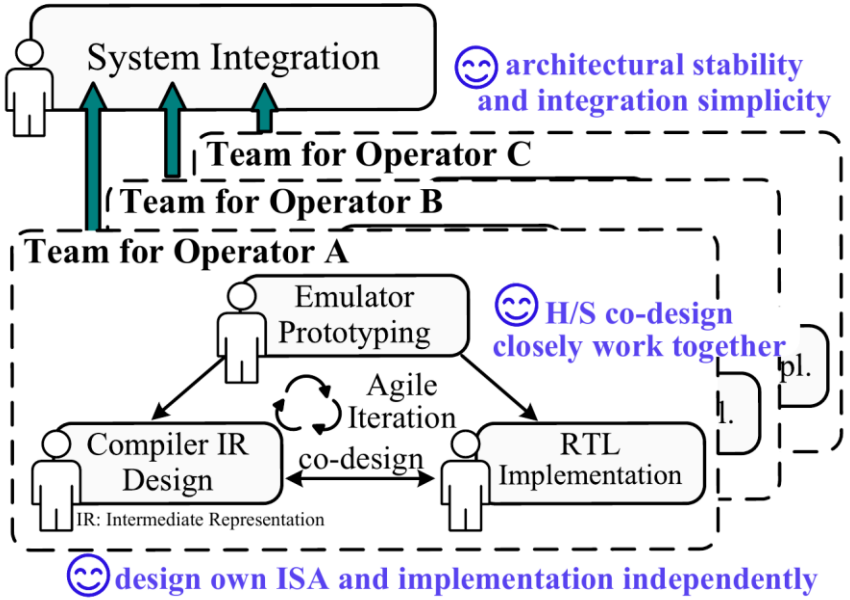
A Stable Architecture Abstraction

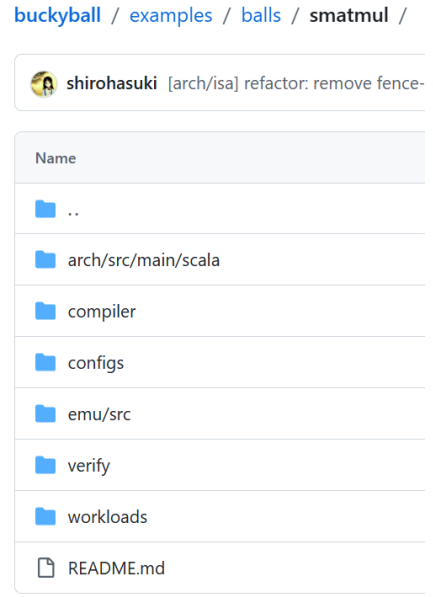


Implementations are scattered across HW & SW

ISAs are fragmented, but semantics are stable

Eliminate the HW/SW boundary, we get a stable architecture



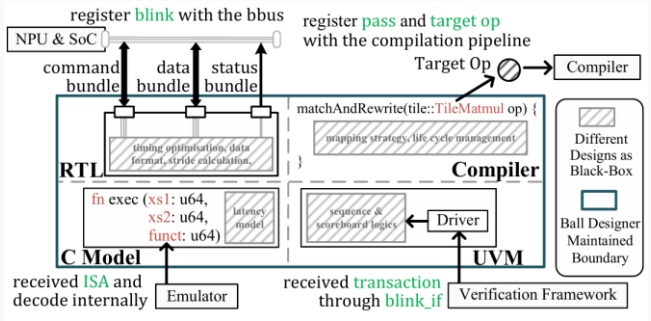


All in one Ball !!!

Micro-Architecture

buckyball

How can one interface serve every design?



Blink

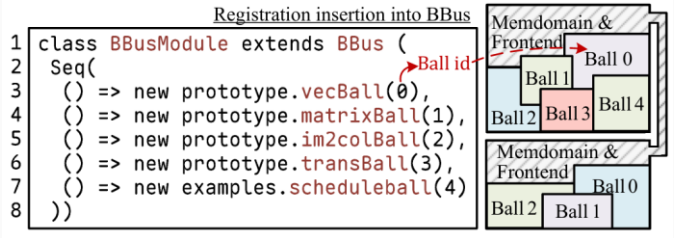
How to schedule without knowing the ISAs?

```
mset(alloc) bankid=0, row=2, col=1
mset(alloc) bankid=2, row=1, col=4
…
for 0 to n {
  mvin memory_address -> %bank0 ①
  mvin memory_address -> %bank1 ②
  transpose %bank0 -> %bank2    ②
  matmul %bank2, %bank1 -> %bank3 ③
  mvout %bank3 -> memory_address  ④

  mvin memory_address -> %bank4 ②
  mvin memory_address -> %bank5   ③
  transpose %bank5 -> %bank6      ③
  matmul %bank2, %bank5 -> %bank7  ④
  mvout %bank7 -> memory_address    ⑤
}
mset(release) bank0, …
```

Bank Abstraction

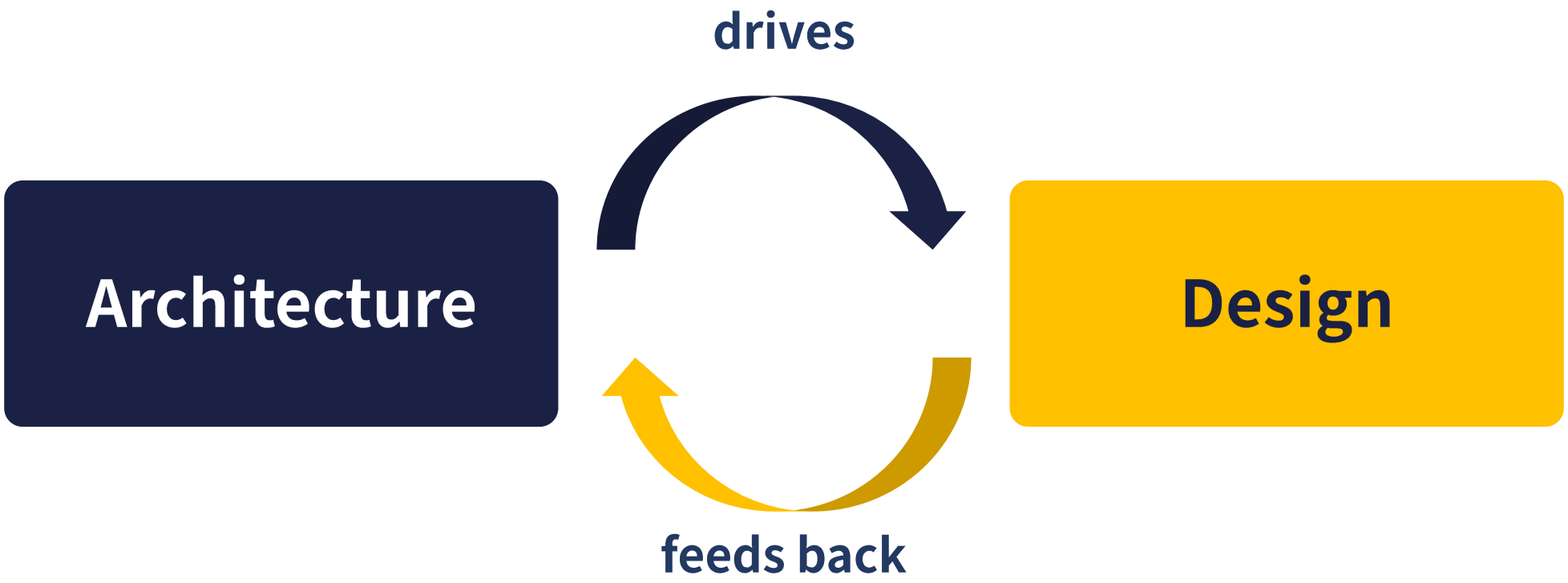
How to scale up without knowing each Ball's design?



Latency hiding

Design Philosophy

buckyball



The best framework, built together. The best chips, born from it.

DSA Shared Compiler Infrastructure

buckyball

Models

- Qwen3-8B
- DeepSeek-R1
- Llama 3.2
- ResNet18
- …

RuyiAI 如意社区

Full stack built on RuyiAI

github.com/buddy-compiler/buddy-examples

Frontend

- Subgraph partitioning
- Quantization framework
- Trace framework

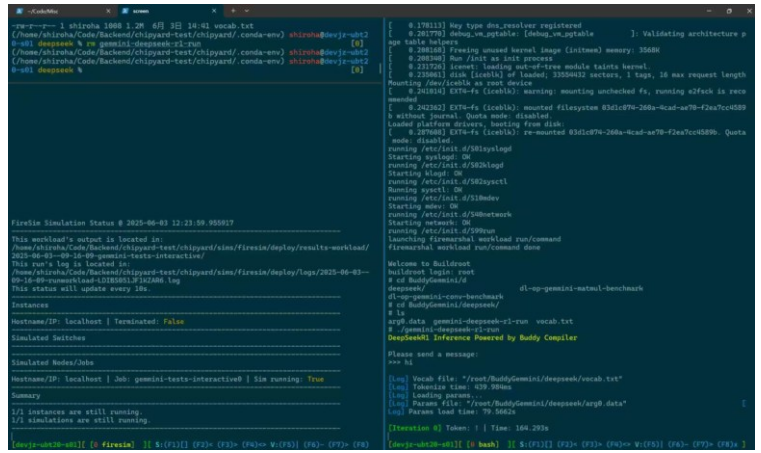
buckyball backend

linalg dialect → tile dialect → **buckyball dialect**

Your design plugs in here

Chips

- Toy
- **Pebble**
- Goban
- Poly
- omni



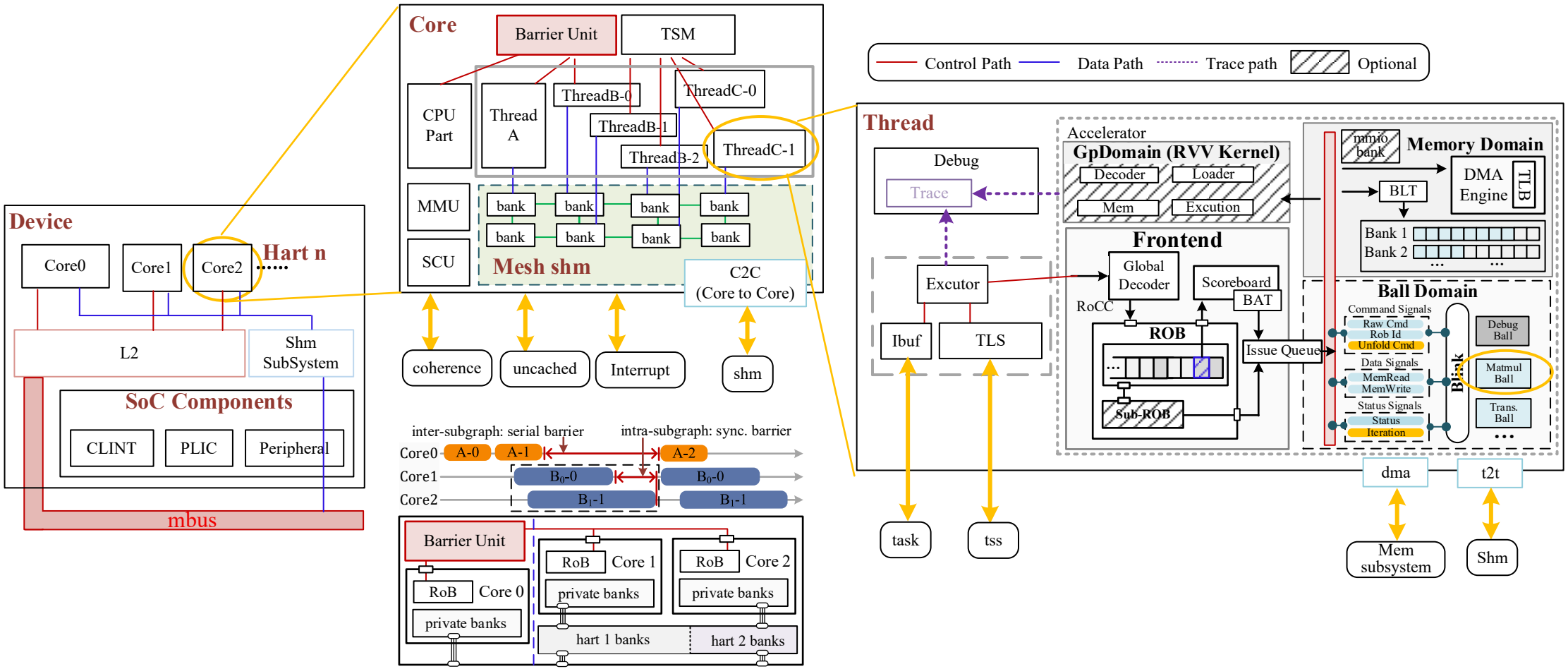
DeepSeek-R1 via RuyiAI

```
$ bbdev compiler --build '--chip pebble'
```

```
$ bbdev workload --build '--chip pebble --model qwen3-8b'
```

DSA Shared Architecture Infrastructure

buckyball

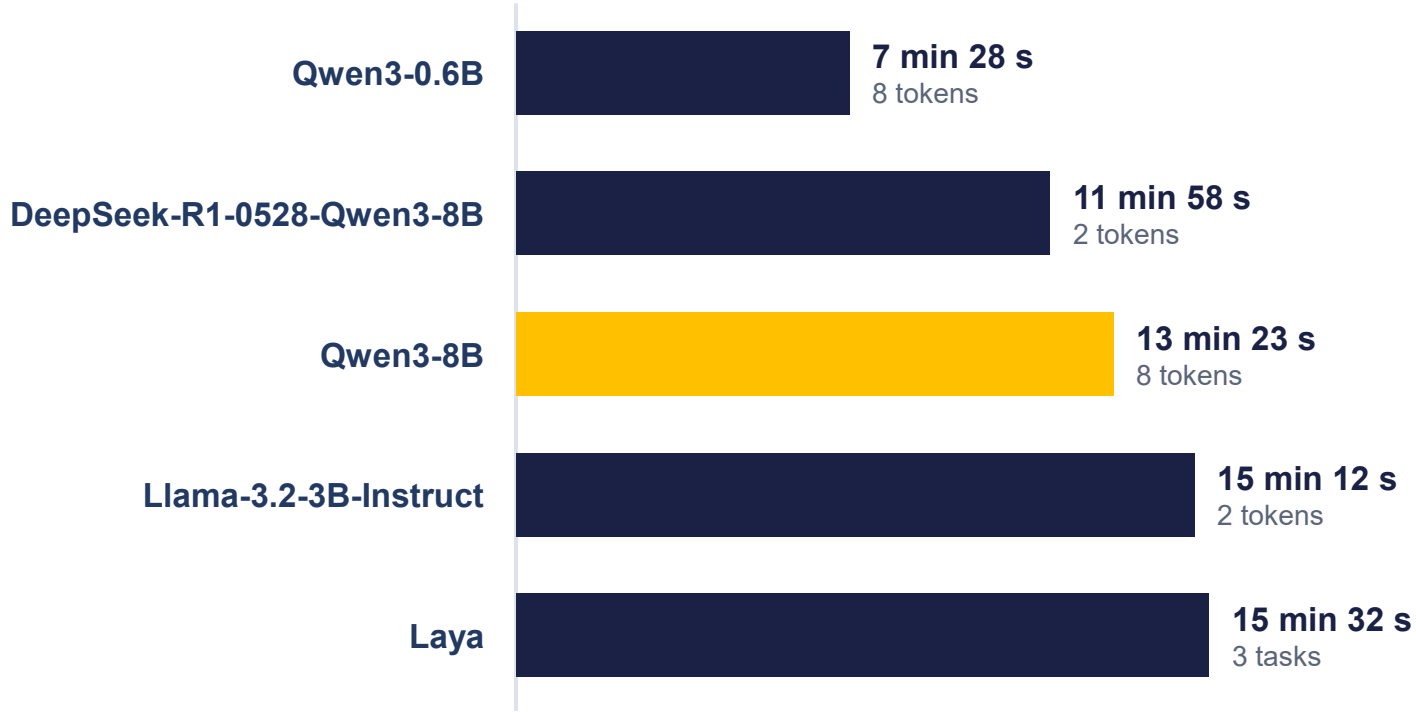


DSA Shared Emulator Infrastructure

buckyball

BEMU

Simulate workload end to end

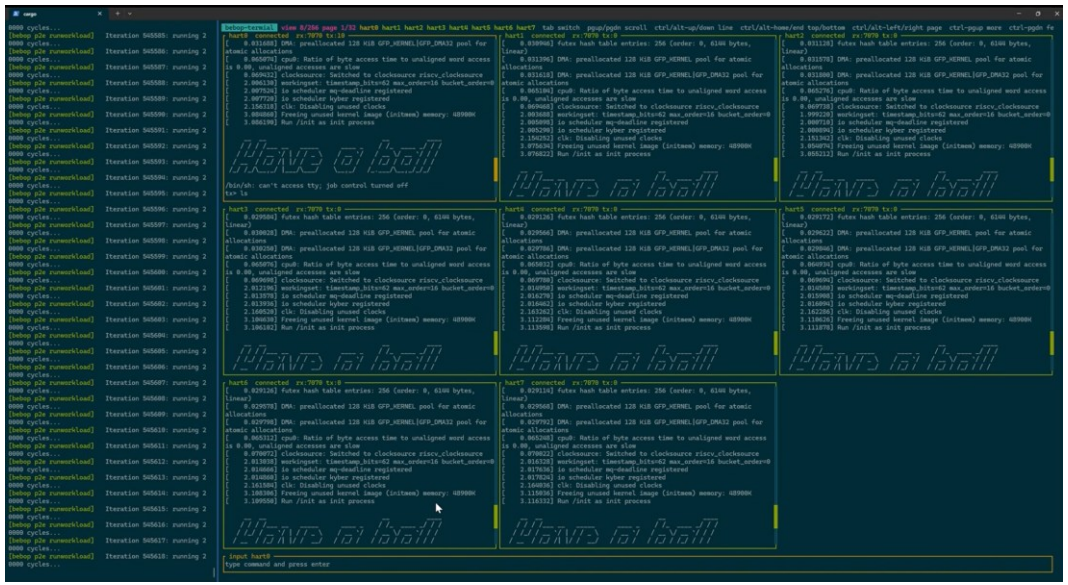



```
$ bbdev bemu --run '--chip toy --binary toy-toy-ctest-relu_i32_16x16_test-baremetal --itrace --mtrace --pmctrace --ctrace'
```

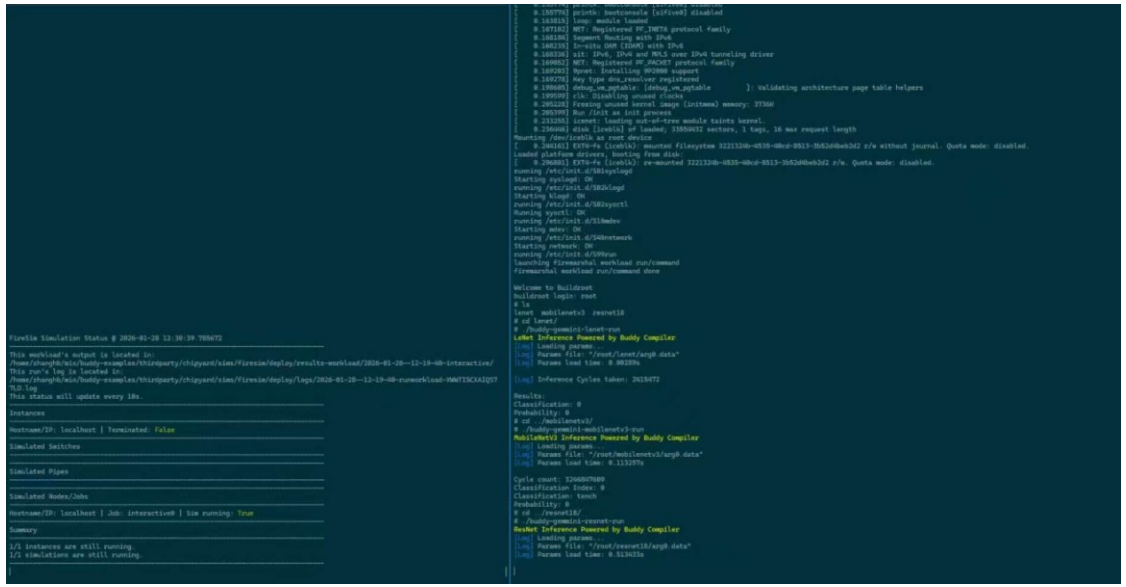
DSA Shared Prototyping Infrastructure

buckyball

Bebop



X-EPIC



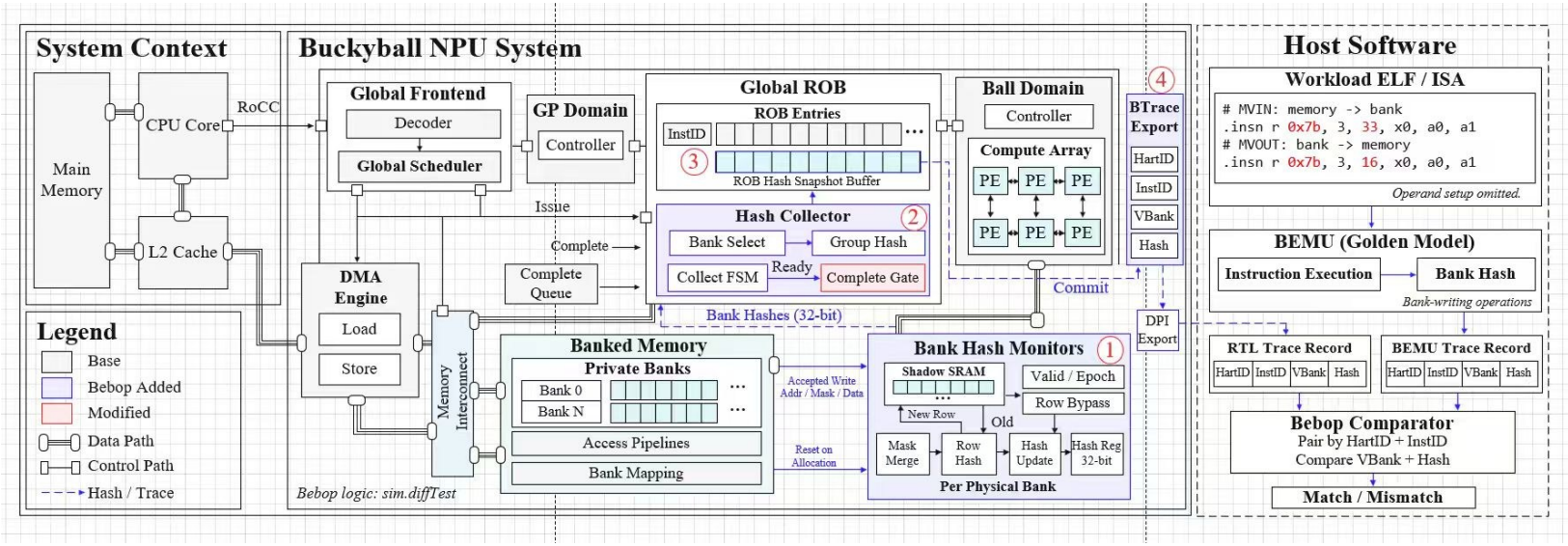
FireSim

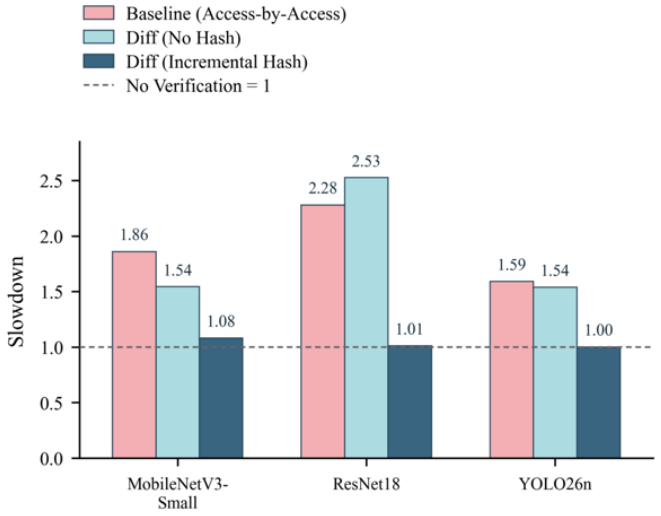
```
$ bbdev bebop-p2e --verilog '--chip pebble'
$ bbdev bebop-p2e --buildbitstream '--chip pebble'
$ bbdev bebop-p2e --runworkload '--chip pebble --image
vecunit_matmul --bitstream bitstream.bit -diff --itrace --mtrace'
```

```
$ bbdev firesim --verilog '--chip pebble'
$ bbdev firesim --buildbitstream '--chip pebble'
$ bbdev firesim --runworkload '--chip pebble --image
vecunit_matmul --bitstream bitstream.bit'
```

DSA Shared Debug Infrastructure

Difftest-N





① Bank Hash Monitors

Hash every bank write, row by row, as it happens

② Hash Collector

Gather each instruction's bank hashes before it completes

③ ROB Snapshot

Keep the hash with the instruction until commit

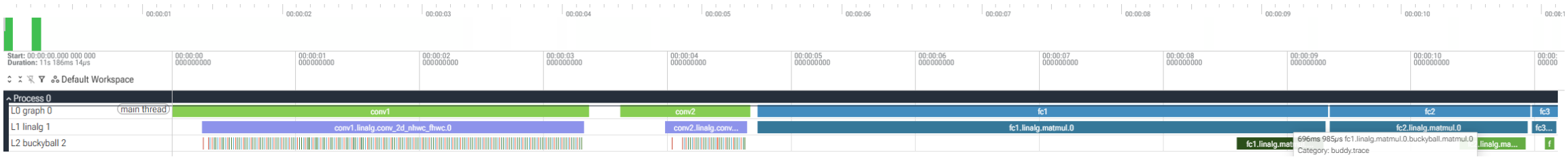
④ BTrace Export

Compare against BEMU, instruction by instruction

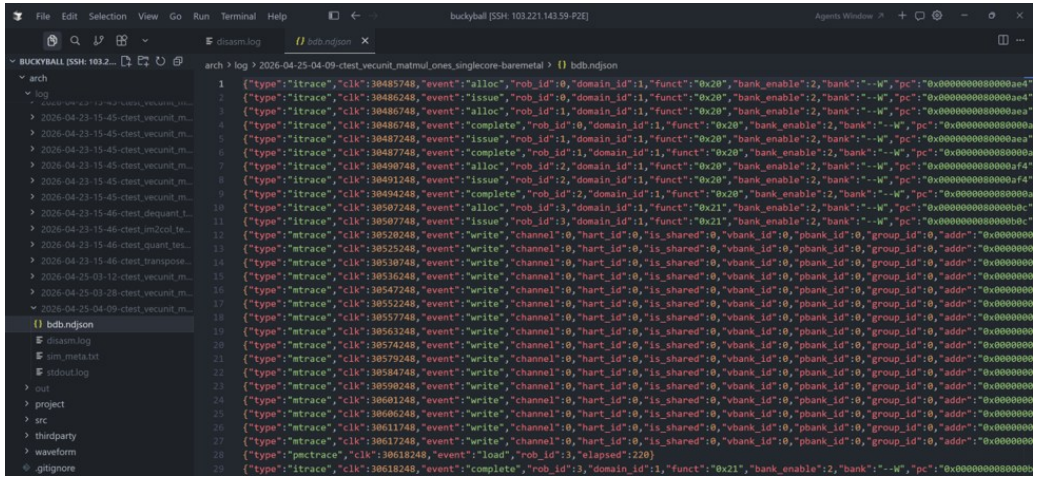
DSA Shared Profiling Infrastructure

buckyball

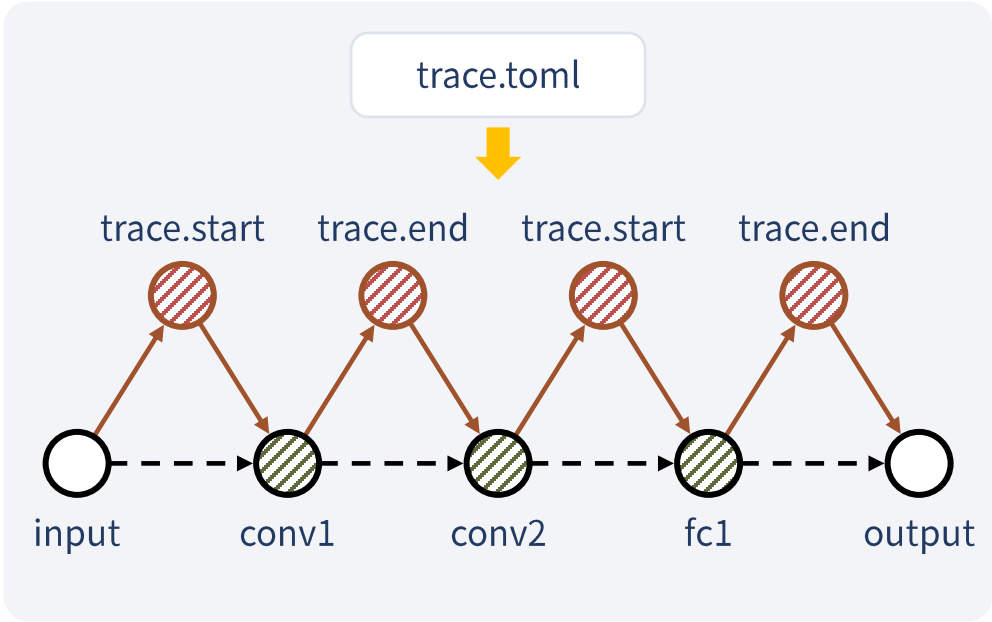
Software trace



Hardware trace

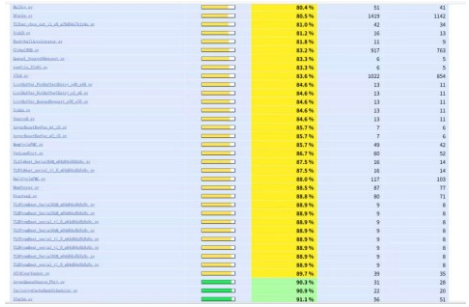


--itrace --mtrace --pmctrace –-ctrace --dtrace



Other DSA Shared Toolchain





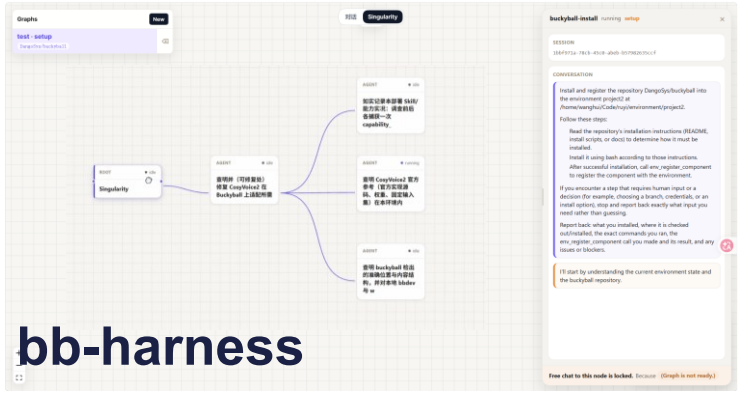
```
$ bbdev uvm --build '--chip pebble --ball --ip'
```

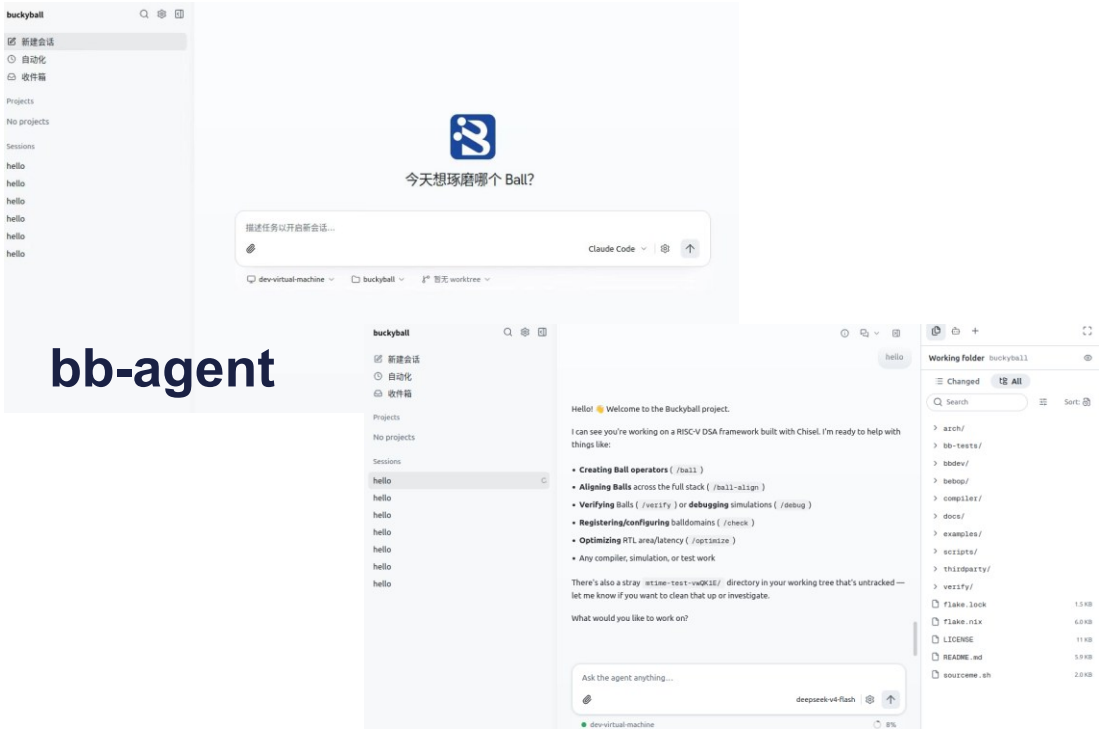
```
$ bbdev dc --area '--chip pebble --top ChipTop'
```

```
$ bbdev dc --power '--chip pebble --top ChipTop -workload
vecunit_matmul
```

```
$ bbdev ip --generate "--chip pebble --top ChipTop"
```

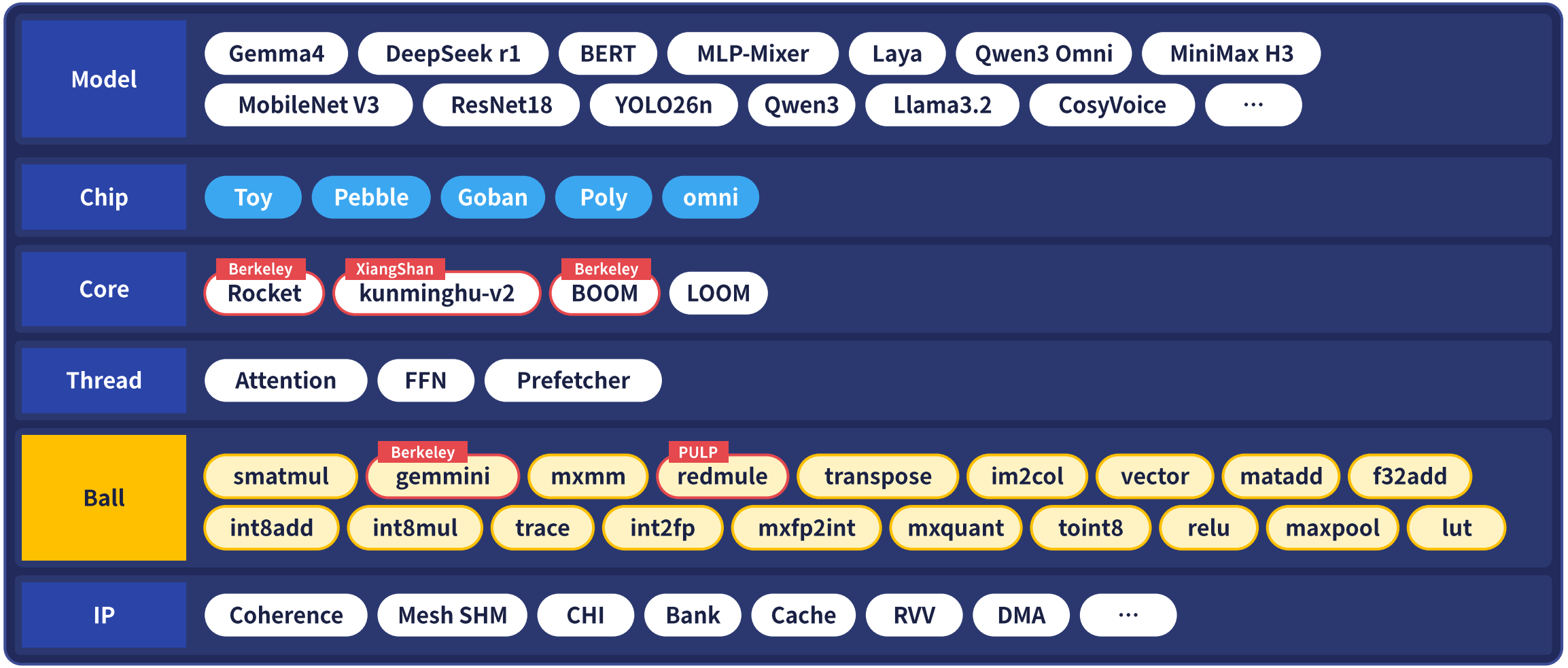
⋮





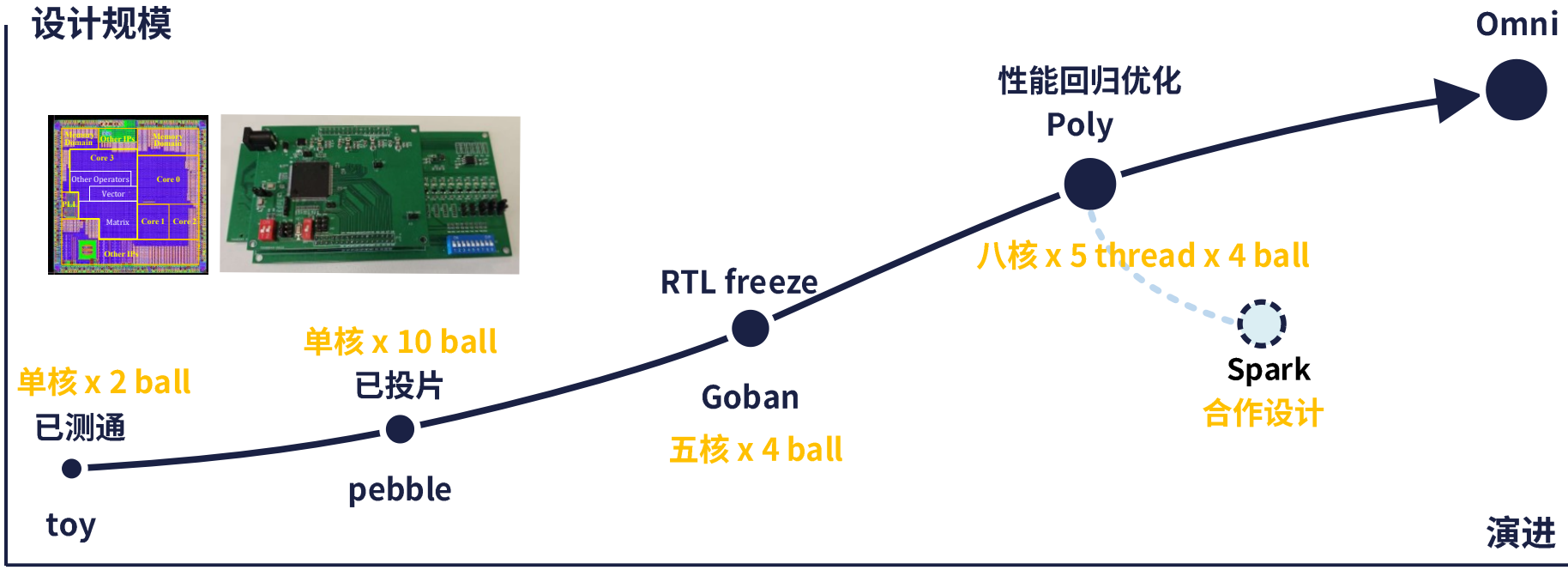
Designs on buckyball

buckyball



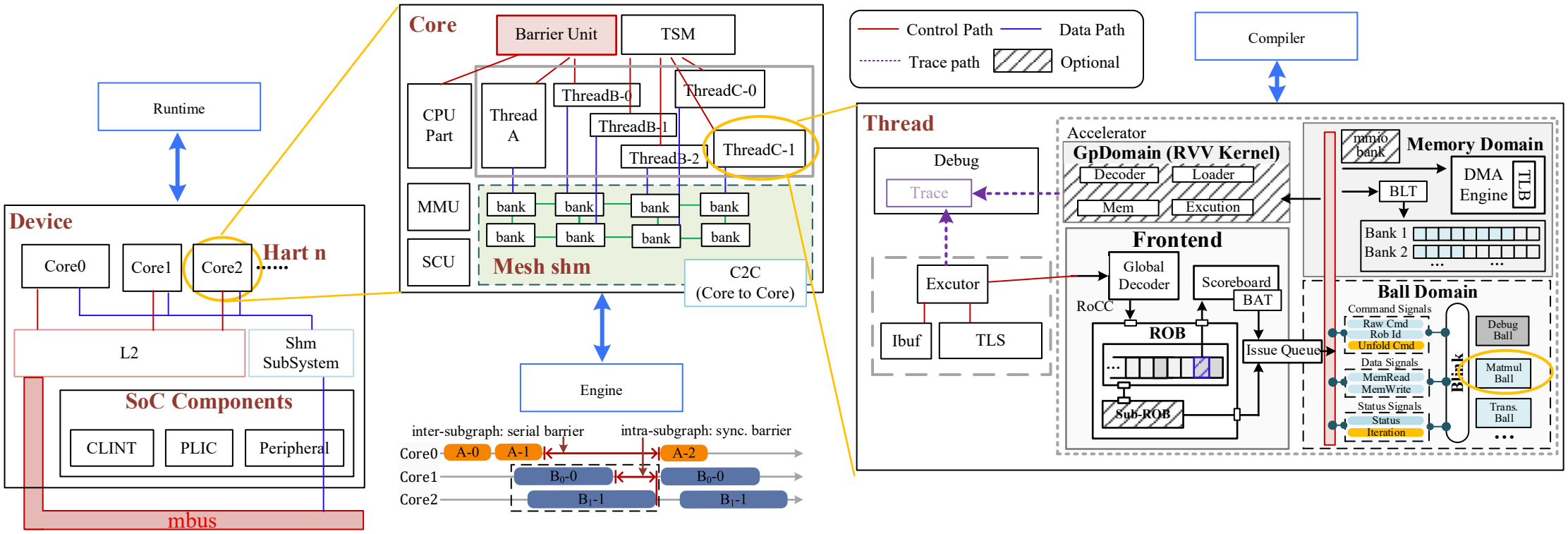
Designs on buckyball

buckyball



Customisation Is More Than Accelerators

buckyball



Hierarchical co-design

Customisation at different abstraction levels integrated together

NPU Prefetcher

buckyball

Background:

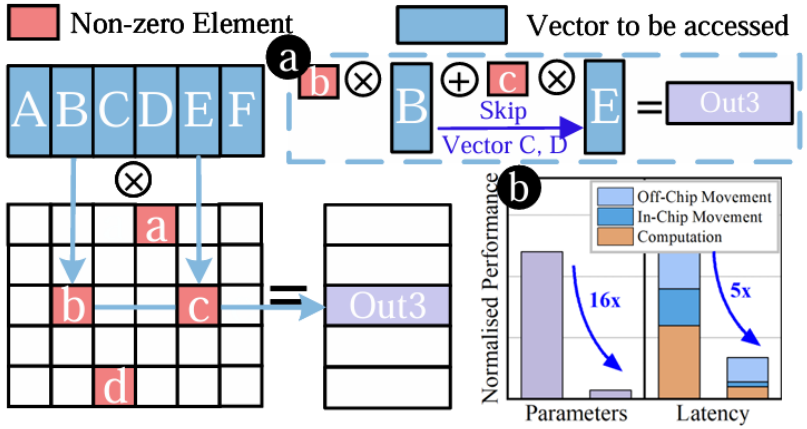
Sparsity is widely present across various workloads in DNNs, such as attention mechanisms, MoE, 3D point cloud.

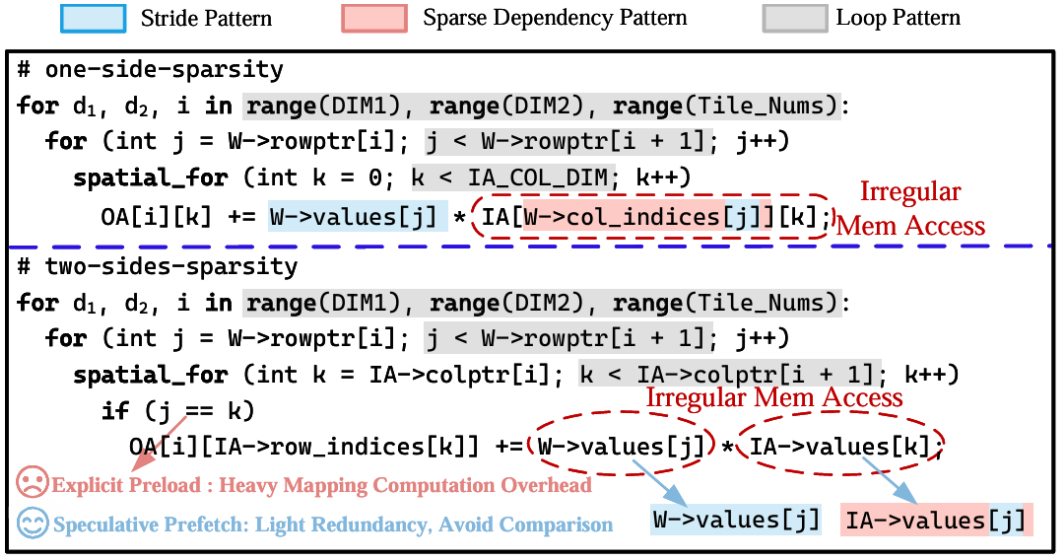
Hardware Unfriendly. In Double Sparsity[1], despite a 16× parameter reduction, frequent **cache misses** trigger extensive off-chip access, resulting in only 5× speedup.

Motivation:

Facing cache misses, NPUs suffer even more severe performance degradation than CPUs and GPUs.

Runahead is a prefetching technique that leverages processor idle resources to speculatively execute and prefetch ahead of cache misses.
It has been used in modern CPUs, particularly effective for irregular memory access patterns that traditional prefetchers struggle to predict.





Existing Sparse Approach: regularizing irregular memory access

Low Performance. Structural sparse pruning, e.g., Nvidia 2:4[2]

Lacking Generalizability. cuSPARSE's CSR compression overlooks specific sparsity patterns like channel and block-level sparsity, causing it to underperform cuBLAS even at 98% matrix sparsity.

Heavy Overhead. Extra scheduling introduce heavy overhead, e.g. SCNN[3]

In this work, instead of regularizing irregular memory access patterns, we leverage prefetching techniques directly to mitigate the impact of cache misses, rather than attempting to eliminate the irregularity itself.

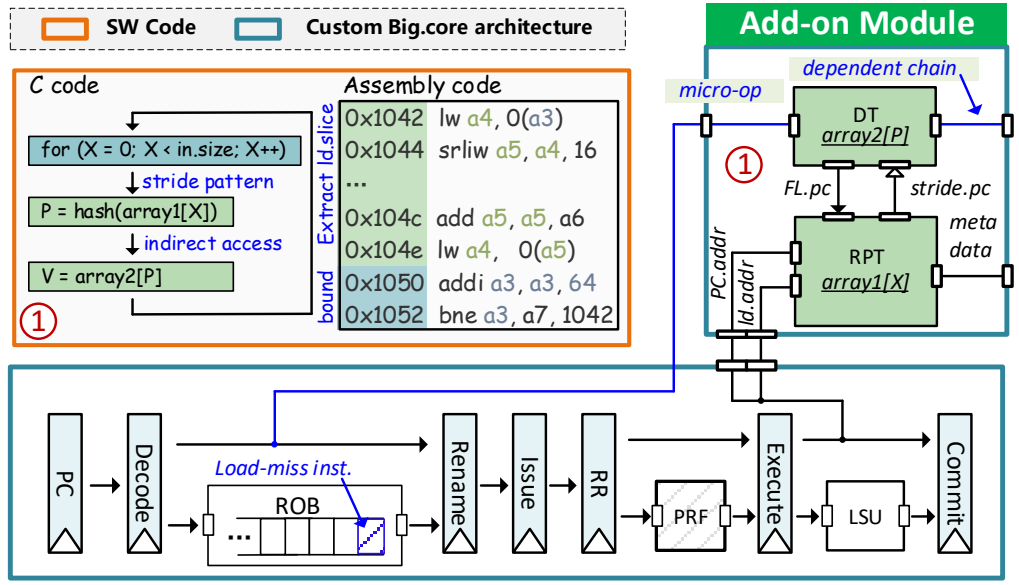
[1] Post-Training Sparse Attention with Double Sparsity.
[2] Accelerating Sparse Deep Neural Networks.
[3] SCNN: An accelerator for compressed-sparse convolutional neural networks.

Custom CPU Prefetcher for Specific Workloads

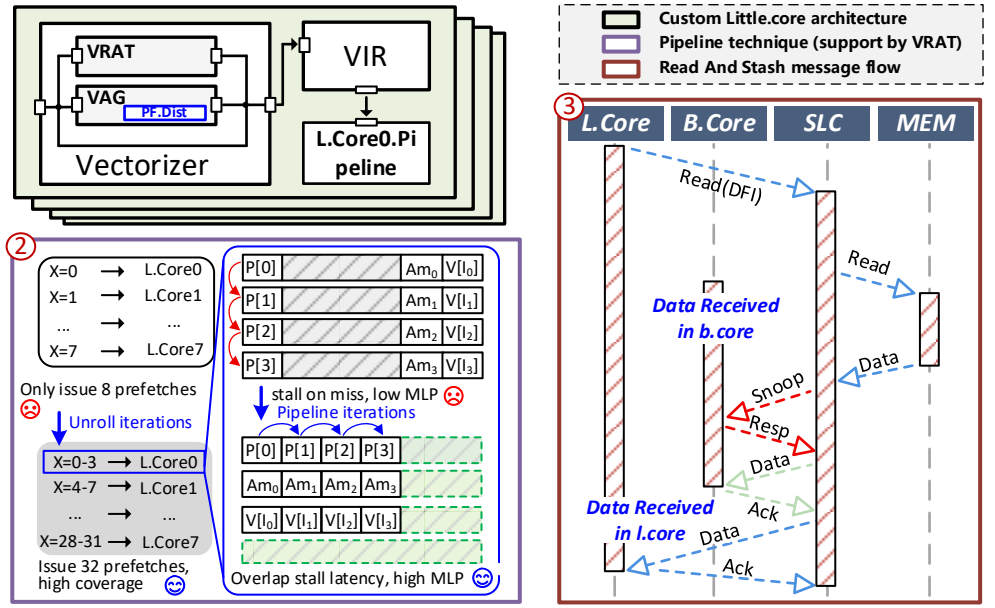
buckyball

- Workloads and Bottlenecks
  Graph retrieval (or traversal), and sparse inference in agentic workloads involve irregular, data-dependent memory accesses (Subsequent memory addresses depend on earlier load results, limiting MLP).

- Design Approach: Big-Core Detection, Little-Core Runahead, System-Level-Cache (SLC) Coordination
  ① The custom big core extracts critical memory-access dependence chains.
  ② The custom little core use pipeline technique to interleaved execution of multiple chains.
  ③ The custom SLC use "Read And Stash" to delivery the prefetched data to the big core and little core.



Custom big.core achitecture



Custom little.core architecture and SLC

Community Timeline







Aug 2025
First line of code

Apr 2026
First internship call on RedNote

Jun 2026
First tutorials published

Jul 2026
Joined OSPP 2026

Aug 2026
Selected for CCF ODTC incentive program

Sep 2026
First community-led tape-out

Oct 2026
Founding SEU Future Computing Association

Next
Join us, write the next one

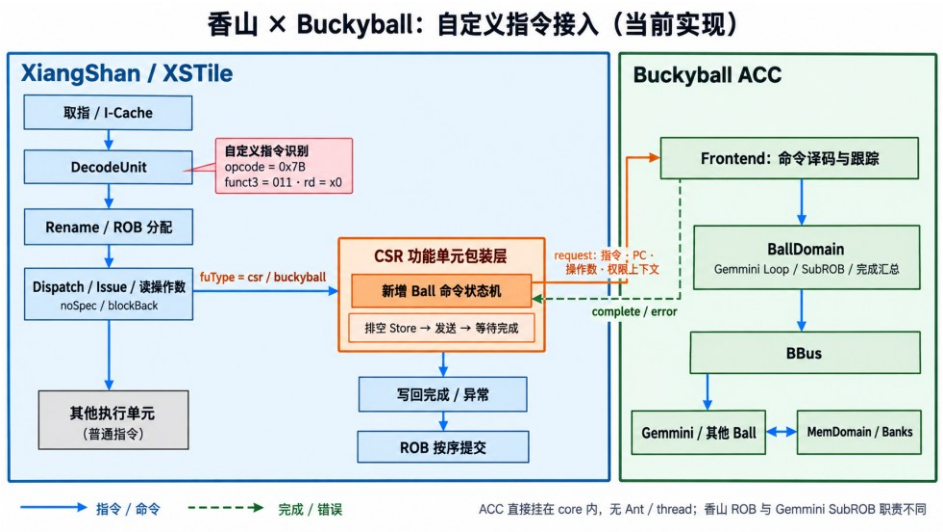
buckyball

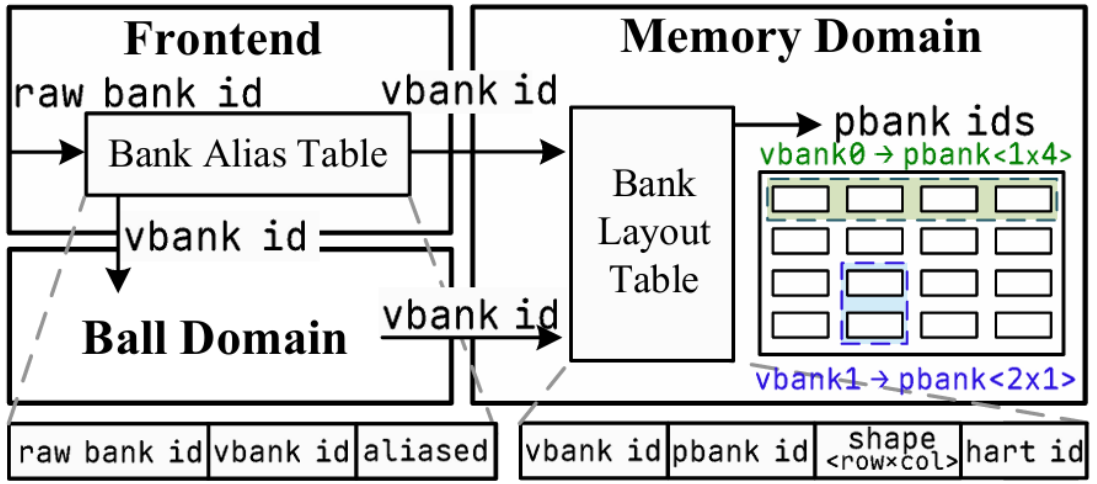
Make DSA development faster, more reusable, and easier to share.

GitHub

Contact Us

Q&A

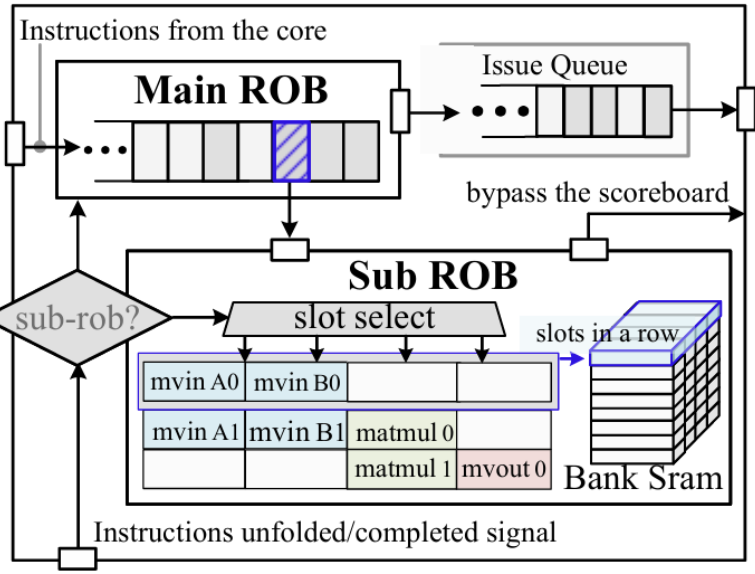




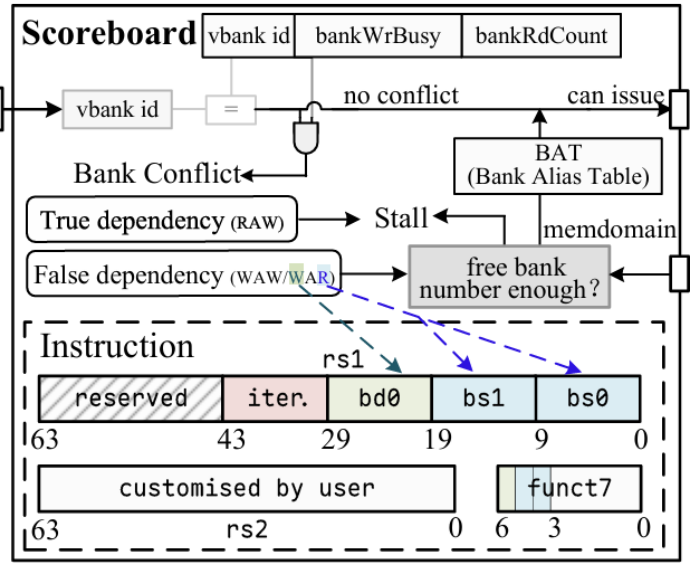

```
mset(alloc) bankid=0, row=2, col=1
mset(alloc) bankid=2, row=1, col=4
…
for 0 to n {
  mvin memory_address -> %bank0 ①
  mvin memory_address -> %bank1 ②
  transpose %bank0 -> %bank2    ②
  matmul %bank2, %bank1 -> %bank3 ③
  mvout %bank3 -> memory_address  ④

  mvin memory_address -> %bank4 ②
  mvin memory_address -> %bank5  ③
  transpose %bank5 -> %bank6     ③
  matmul %bank2, %bank5 -> %bank7  ④
  mvout %bank7 -> memory_address   ⑤
}
mset(release) bank0, …
```

(a) Bank based programming model.



(b) Two cascaded ROBs with parallel dispatch.



(c) vbank instruction dependency resolution.