中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

鹏城实验室
Peng Cheng Laboratory

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

XiangShan: an Open-Source High-Performance RISC-V Processor

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

Dec 6, 2021@RISC-V Summit

Outline

- **Introduction of XiangShan**
- Agile Performance Modelling with Software-based RTL-simulation
- Micro-architecture Design and Optimizations of XiangShan

Open-source Processor Ecosystem

① ISA + ② Microarch. design/impl. + ③ Workflow/Tools

Open Framework

microarch.

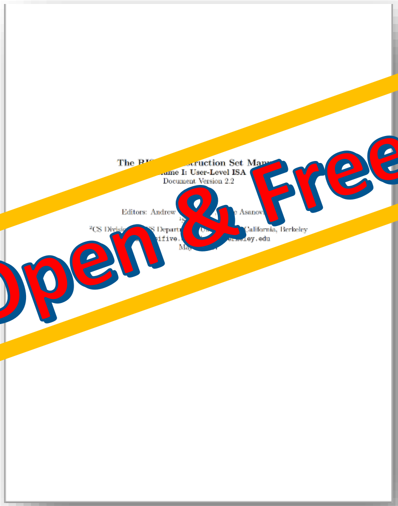
Open Flow

coding

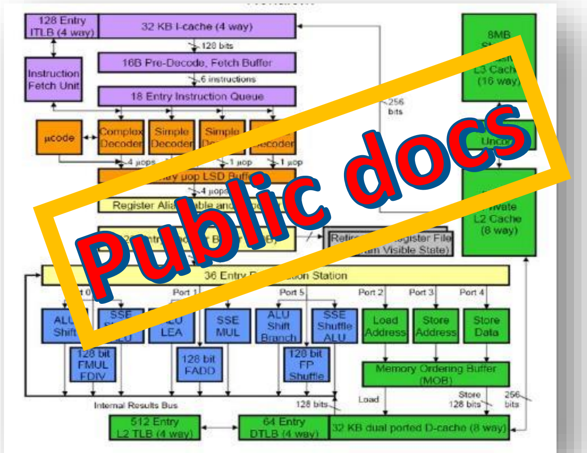
Open Tools

EDA Tools

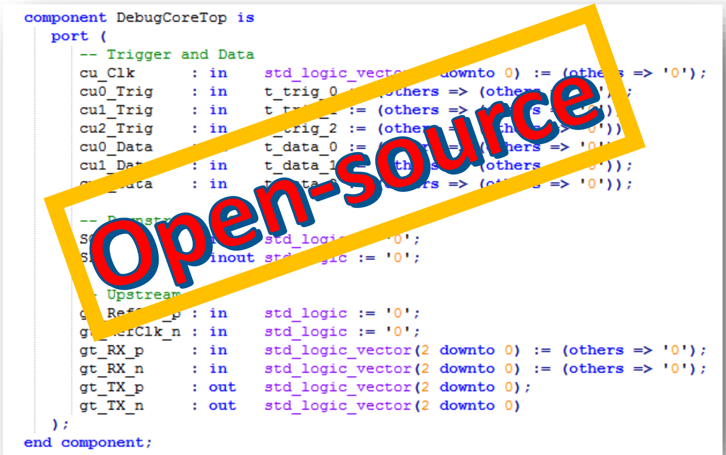
ISA Spec. | Docs | RTL | Layout



Open & Free

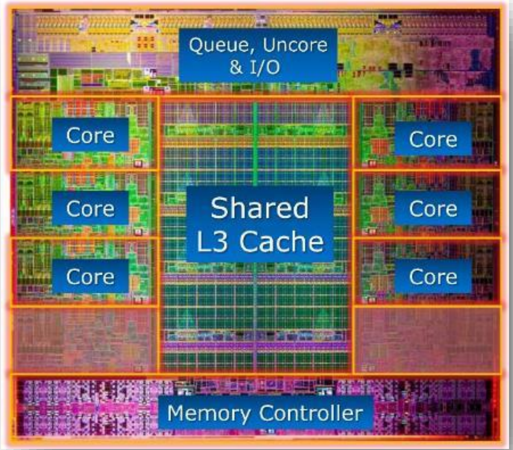


Public docs



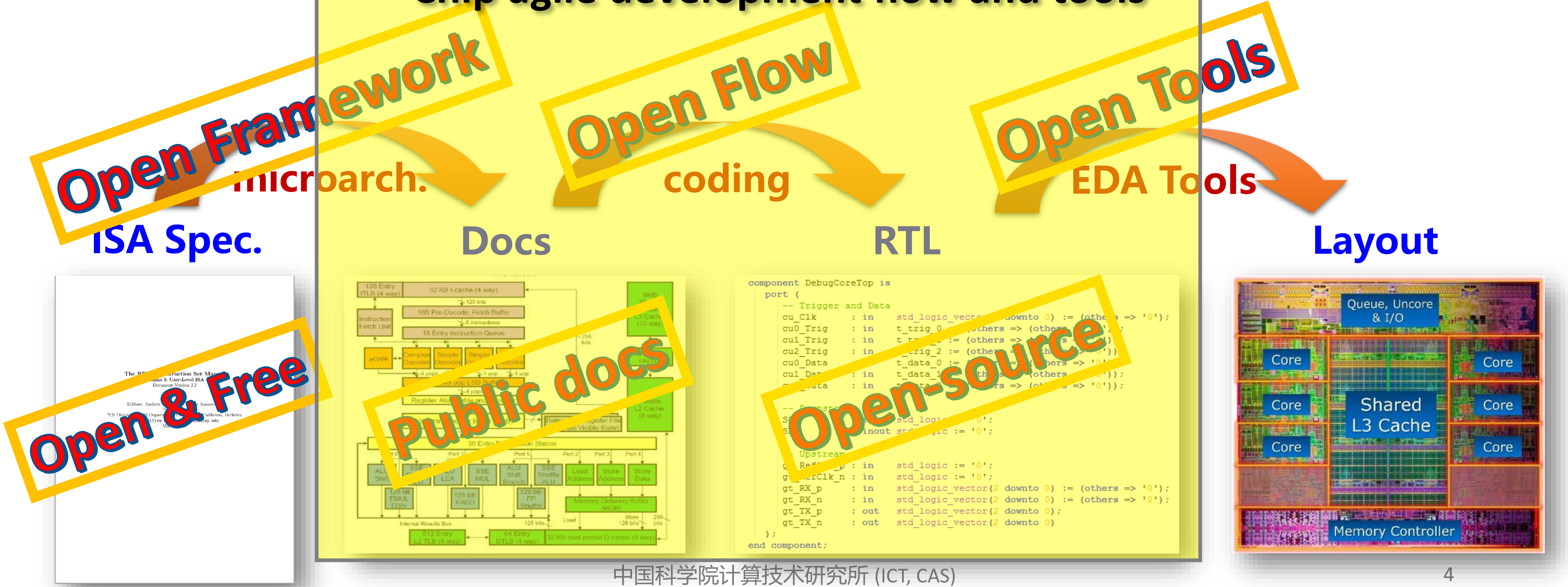
```
component DebugCoreTop is
  port (
    -- Trigger and Data
    cu_Clk      : in    std_logic_vect[illegible] downto 0) := (othe[illegible]s => '0');
    cu0_Trig    : in    t_trig_0 [illegible]thers => (othe[illegible]'[illegible];
    cu1_Trig    : in    t_t[illegible] := (others => [illegible]s [illegible]))
    cu2_Trig    : in    [illegible]rig_2 := (othe[illegible] [illegible]0'))
    cu0_Data    [illegible]      t_data_0 := [illegible]hers => '0'[illegible]
    cu1_Da[illegible]  : in    t_data_1[illegible] (others [illegible]'));
    [illegible]ata      : in    [illegible]rs => (o[illegible]s => '0'));

    -- D[illegible]
    S[illegible]        std_logi[illegible] '0';
    S[illegible]        inout st[illegible]ic := '0';

    [illegible] Upstream
    g[illegible]Ref[illegible]p : in    std_logic := '0';
    g[illegible]erClk_n : in    std_logic := '0';
    gt_RX_p     : in    std_logic_vector(2 downto 0) := (others => '0');
    gt_RX_n     : in    std_logic_vector(2 downto 0) := (others => '0');
    gt_TX_p     : out   std_logic_vector(2 downto 0);
    gt_TX_n     : out   std_logic_vector(2 downto 0)
  );
end component;
```

Open-source



Open-source RISC-V processor

- Open-source RISC-V processor
- Chip agile development flow and tools

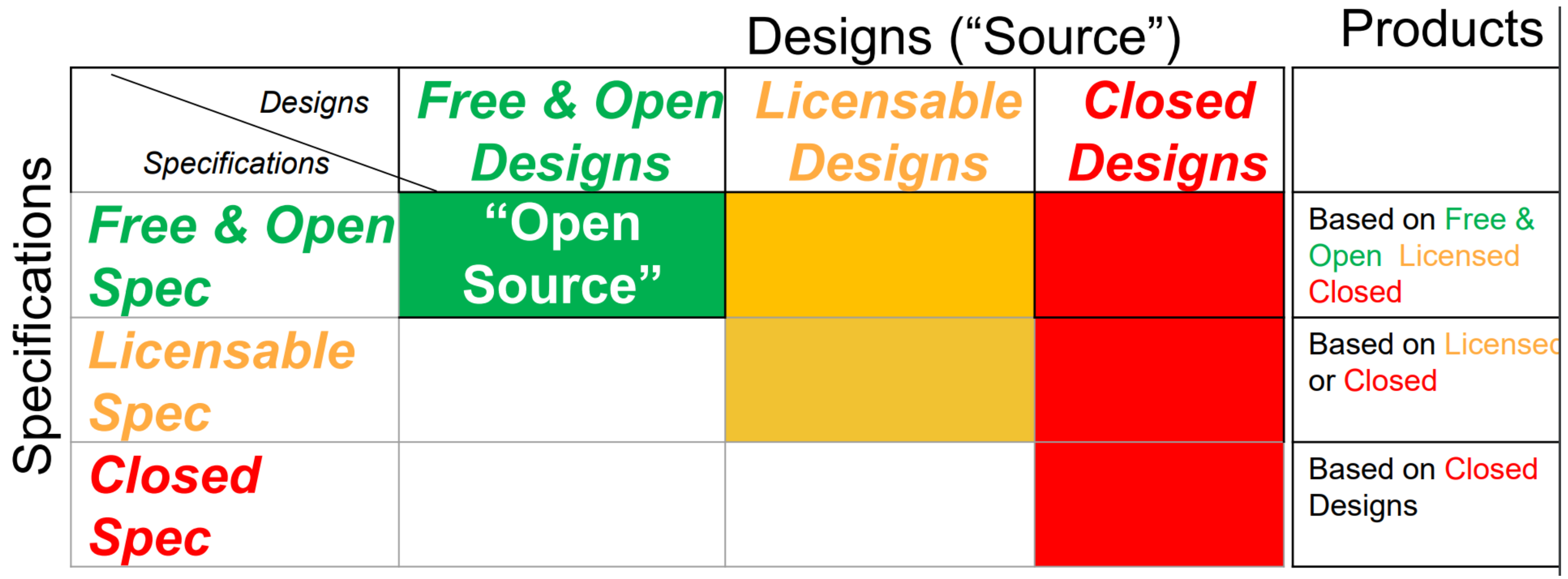


```
component DebugCoreTop is
  port (
    -- Trigger and Data
    cu_Clk       : in    std_logic_vect[illegible] downto 0) := (othe[illegible]s => '0');
    cu0_Trig     : in    t_trig_0 [illegible]others => (othe[illegible]
    cu1_Trig     : in    t_t[illegible] := (others => [illegible]
    cu2_Trig     : in    [illegible]rig_2 := (other[illegible]
    cu0_Data     [illegible] t_data_0 := [illegible]hers => '0'[illegible]
    cu1_Da[illegible]  : in    t_data_1[illegible] (others [illegible]'));
    [illegible]ata   : in    [illegible]rs => (o[illegible] => '0'));

    -- D[illegible]
    S[illegible]      std_logi[illegible] '0';
    S[illegible] inout st[illegible]ic := '0';

    [illegible] Upstream
    g[illegible]Ref[illegible]_p : in    std_logic := '0';
    g[illegible]fClk_n : in    std_logic := '0';
    gt_RX_p      : in    std_logic_vector(2 downto 0) := (others => '0');
    gt_RX_n      : in    std_logic_vector(2 downto 0) := (others => '0');
    gt_TX_p      : out   std_logic_vector(2 downto 0);
    gt_TX_n      : out   std_logic_vector(2 downto 0)
  );
end component;
```

Why open-source high-perf. RISC-V processor?

- **Why RISC-V**: Free and open ISA
- **Why high-perf** : Most RISC-V processors are for IoT/AI, but both academic and industrial community need high-performance RISC-V processors
- **Why open-source**: An open and innovative hardware platform, "hardware version of Linux"

- **Build a leading platform with chip development method, flow and tools**



	Designs ("Source")			Products
Designs / *Specifications*	***Free & Open Designs***	***Licensable Designs***	***Closed Designs***	
Free & Open Spec	**"Open Source"** 🟩	🟧	🟥	Based on Free & Open Licensed Closed
Licensable Spec		🟨	🟥	Based on Licensed or Closed
Closed Spec			🟥	Based on Closed Designs

Specifications

Source: David Patterson, Keynote @ CRVF 2019, https://crvf2019.github.io/pdf/keynote1.pdf

XiangShan: an open-source high-performance processor

- **XiangShan**: an industrial-level processor written in Chisel
  - Named after a mountain in Beijing, China

Fragrant Hills in Beijing

- **Timeline**

July 15, 2021
First tape-out (Yanqihu)
Estimated SPEC CPU2006 9@1.3GHz

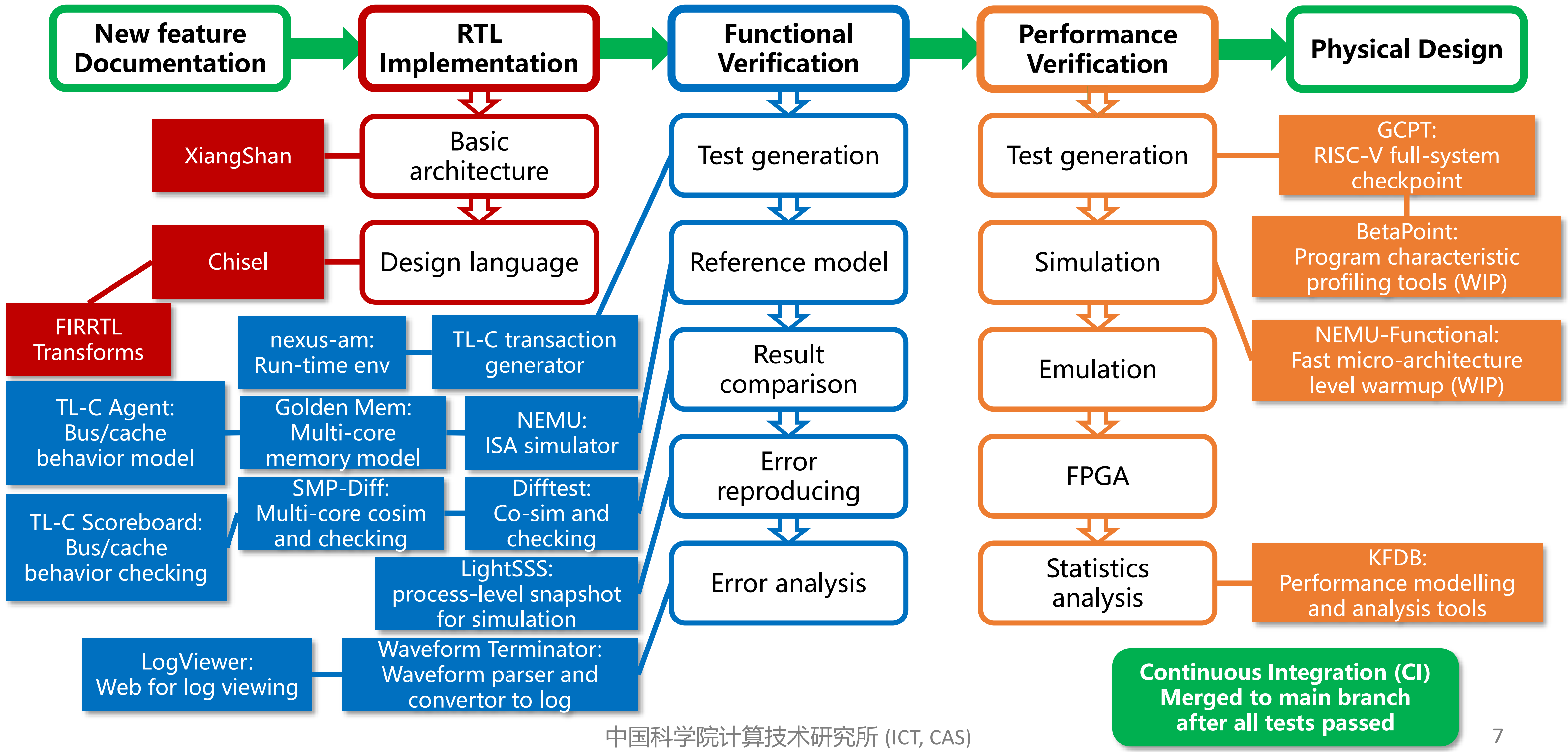
June 11, 2020
First commit

Early 2022
To be the 2nd tape-out (Nanhu)
Estimated SPEC06 20@2GHz

- **Open-sourced at** https://github.com/OpenXiangShan/XiangShan

Scan to follow us on GitHub

Agile development infrastructures

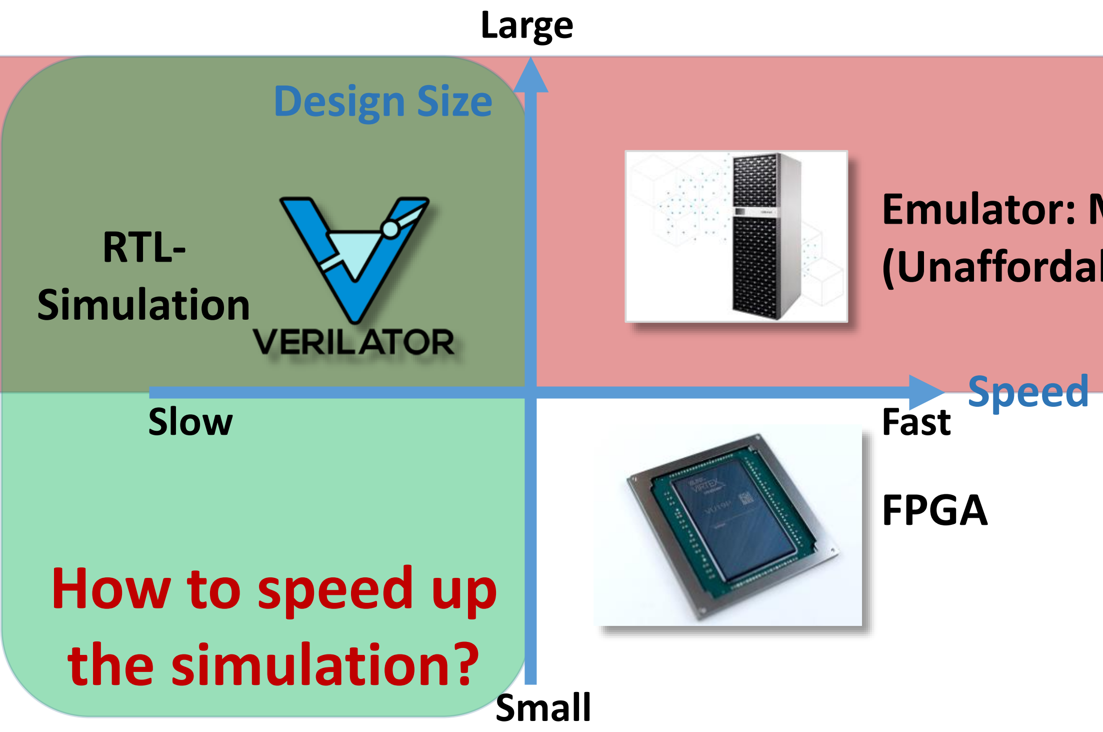

New feature Documentation
RTL Implementation
Functional Verification
Performance Verification
Physical Design
XiangShan
Basic architecture
Chisel
Design language
FIRRTL Transforms
Test generation
Reference model
Result comparison
Error reproducing
Error analysis
nexus-am: Run-time env
TL-C transaction generator
TL-C Agent: Bus/cache behavior model
Golden Mem: Multi-core memory model
NEMU: ISA simulator
TL-C Scoreboard: Bus/cache behavior checking
SMP-Diff: Multi-core cosim and checking
Difftest: Co-sim and checking
LightSSS: process-level snapshot for simulation
LogViewer: Web for log viewing
Waveform Terminator: Waveform parser and convertor to log
Test generation
Simulation
Emulation
FPGA
Statistics analysis
GCPT: RISC-V full-system checkpoint
BetaPoint: Program characteristic profiling tools (WIP)
NEMU-Functional: Fast micro-architecture level warmup (WIP)
KFDB: Performance modelling and analysis tools
Continuous Integration (CI) Merged to main branch after all tests passed

Outline

- Introduction of XiangShan
- **Agile Performance Modelling with Software-based RTL-simulation**
- Micro-architecture Design and Optimizations of XiangShan

Performance Modelling with Software-based RTL-simulation

High-performance designs are too complex to develop agilely on FPGA



How to speed up the simulation?

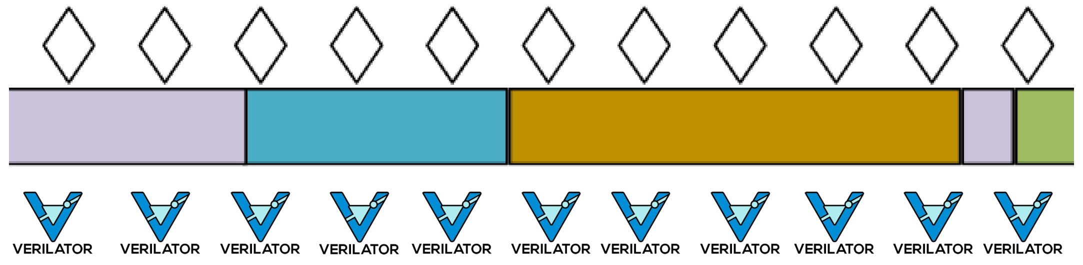
Time for performance modelling of single-core XiangShan on SPEC CPU2006

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

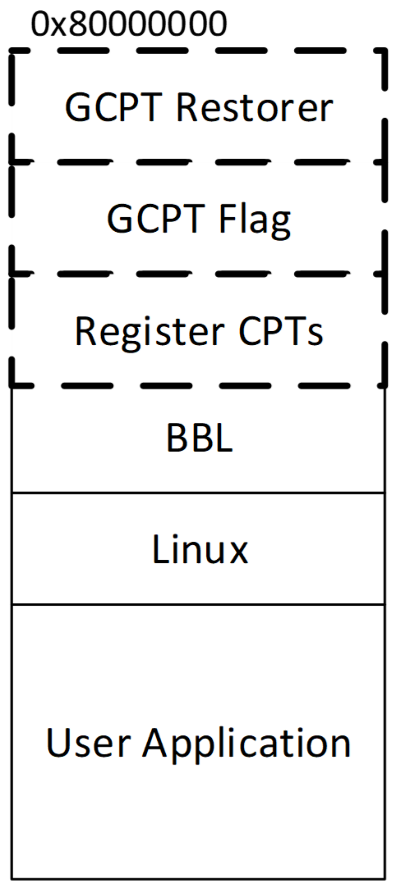
Our Approach

Parallelism via RISC-V Checkpoint

- **Cut a program into small segments, and simulate the segments simultaneously**



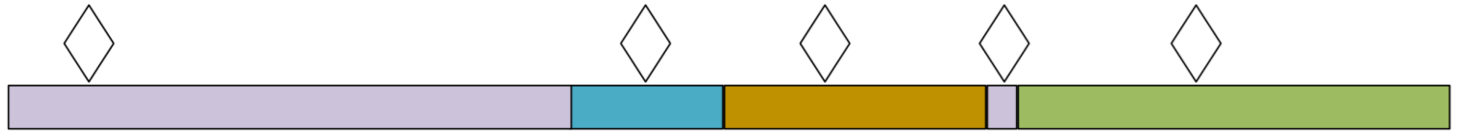
- **Challenge ①: create checkpoints on a certain position of the program**
  - Based on NEMU, an instruction set simulator (ISS) running at ~300MIPS
  - Required information: program counter, architecture registers, memory, etc
- **Challenge ②: restore checkpoints when simulating XiangShan**
  - Key problem: the design is changing, and we need a generic approach for RISC-V
  - **Solution: use privilege instructions to initialize the registers**
  - Dromajo [MICRO'21] uses debug mode, which does not exist on some processors



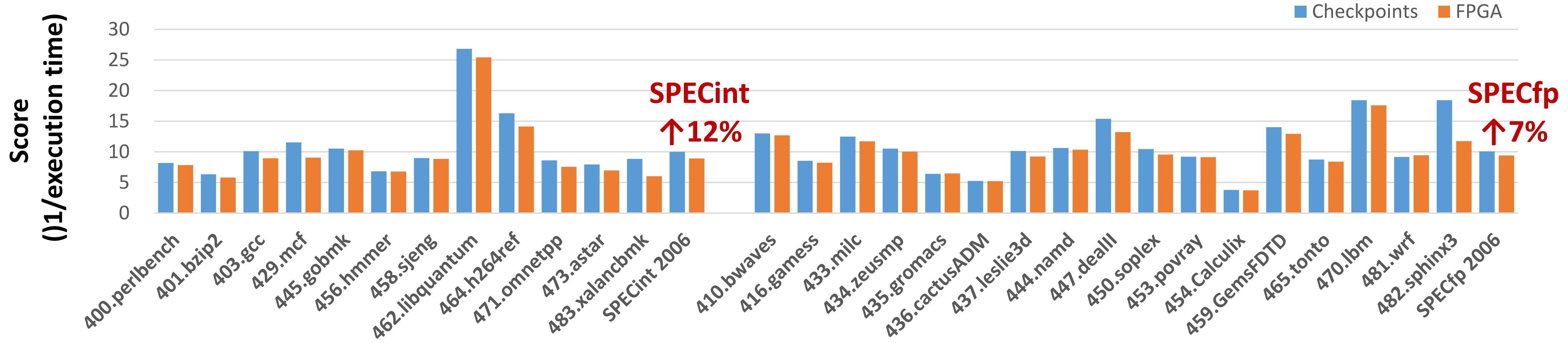
Format of RISC-V Checkpoint

Feature Clustering via Simpoint

- Simpoint[1]: finding and exploiting program phase behavior



- Generating representative RISC-V checkpoints from SPEC CPU2006
- Example: execution time estimation[2] based on clustering weights and simulated IPC



Estimated Yanqihu SPEC CPU2006 results@1.3GHz

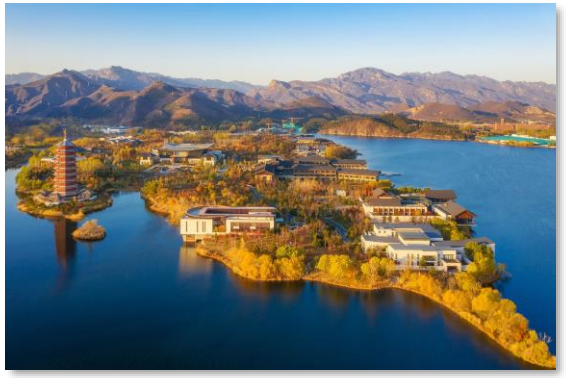
[1] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder, SimPoint 3.0: Faster and More Flexible Program Analysis , Journal of Instruction Level Parallel, September 2005.
[2] 90-cycle fixed latency. Simpoints with 80% coverage and maxK=30.

Outline

- Introduction of XiangShan
- Agile Performance Modelling with Software-based RTL-simulation
- **Micro-architecture Design and Optimizations of XiangShan**

Yanqihu: 1st generations of XiangShan

- **Yanqihu**: named after a lake in Beijing, China
  - RV64GC, 11-stage, superscalar, out-of-order
  - 5.3 CoreMark/MHz (gcc-9.3.0 –O2)
  - Estimated SPEC CPU2006 ~9@1.3GHz


Yanqi Lake in Beijing

- **Tape-out**: single XiangShan core (commit hash ccbca07) with 1MB L2 Cache

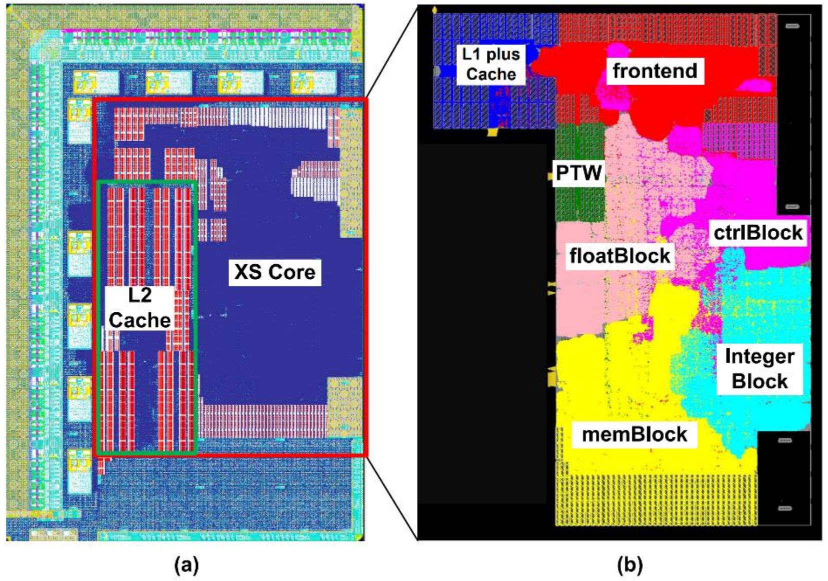
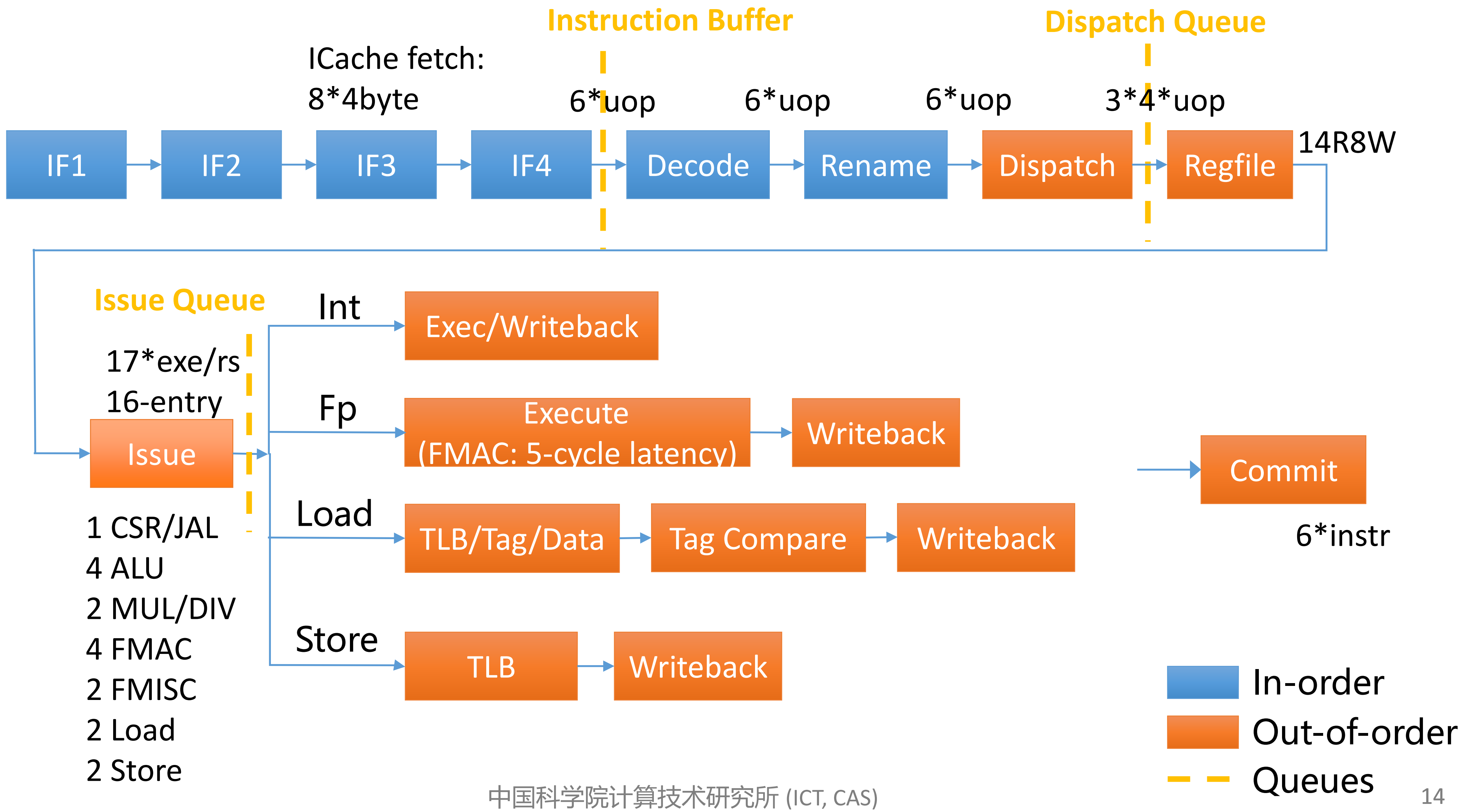


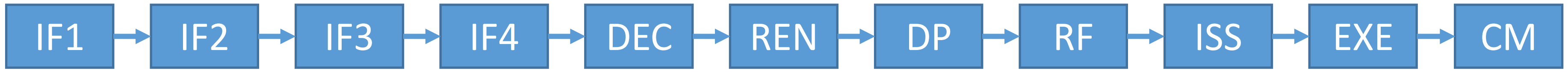
Figure. Layout of (a) the entire chip; (b) the core

Tape-out information for the processor core	
Process Node	28nm
Die Size	8.6 mm^2
Std Cell	5.05M, 4.27 mm^2
Mem	261, 1.7mm^2
Density	66%
Cell	ULVT 1.04%, LVT 19.32%, SVT 25.19%, HVT 53.67%
Estimated Power	5W
Frequency	1.3GHz, TT85C

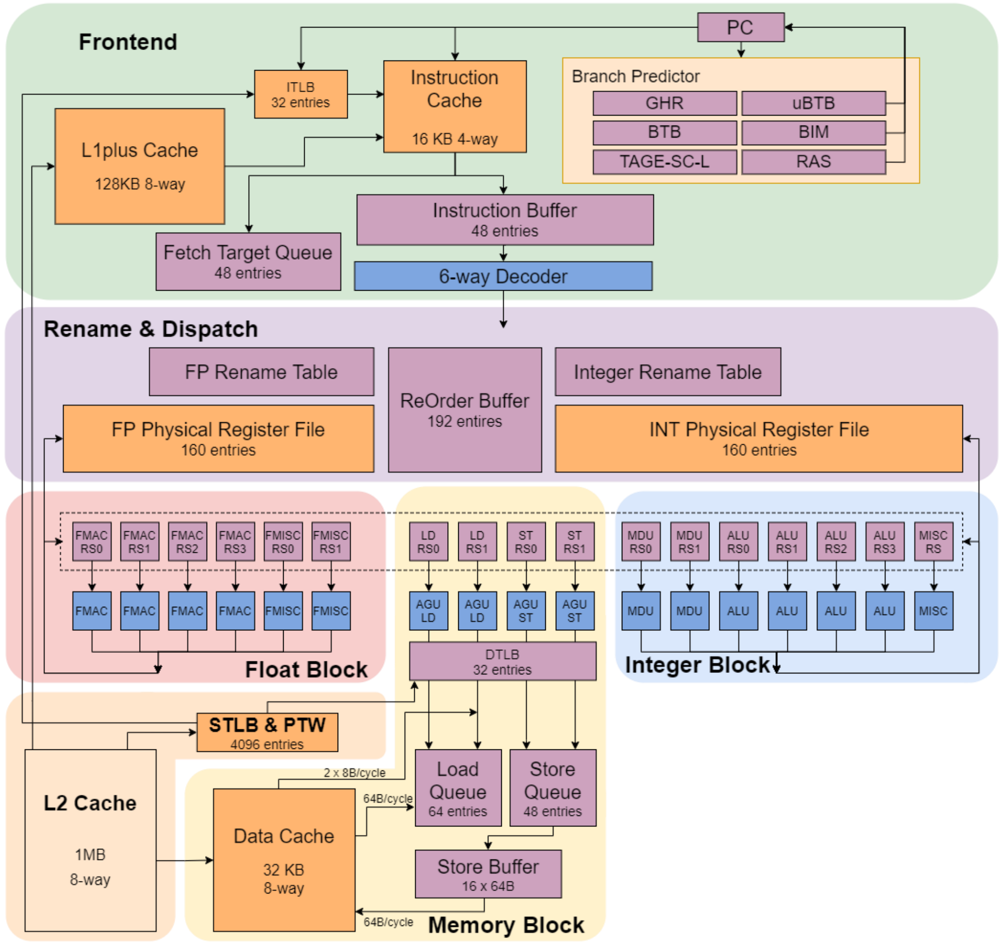
Yanqihu Pipelines



XiangShan microarchitecture (Yanqihu)

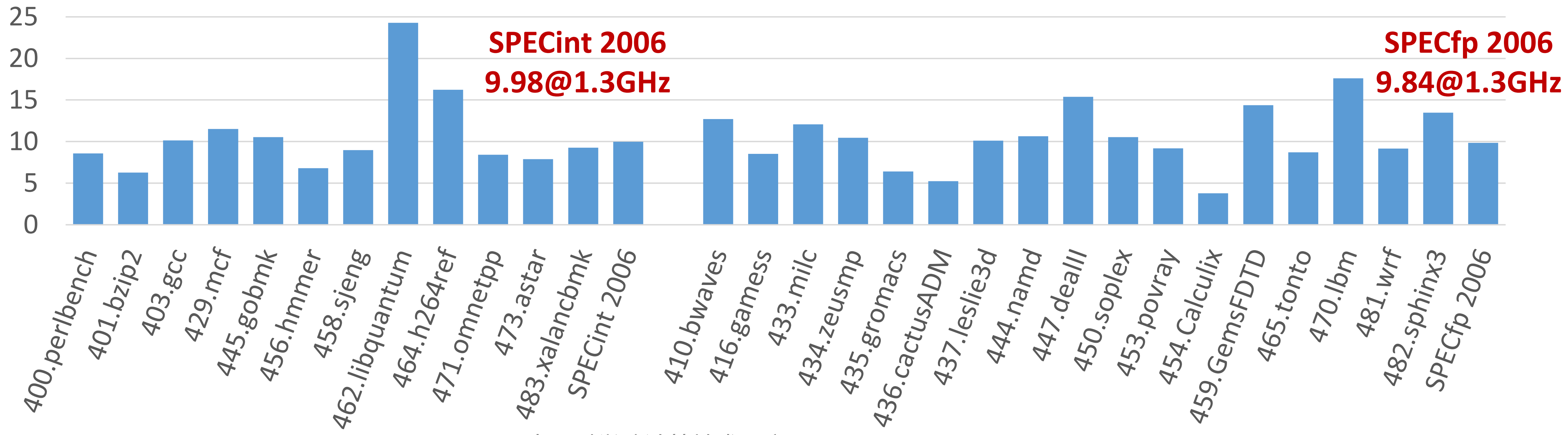


- **11**-stage, **6**-wide decode/rename
- **TAGE-SC-L** branch prediction
- **160** Int PRF + **160** FP PRF
- **192**-entry ROB, **64**-entry LQ, **48**-entry SQ
- **16**-entry RS for each FU
- **16KB** L1 Cache, **128KB** L1plus Cache for instruction
- **32KB** L1 Data Cache
- **32**-entry ITLB/DTLB, **4K**-entry STLB
- **1MB** inclusive L2 Cache



Estimated SPEC CPU2006 results of Yanqihu

- **RTL simulation** (commit hash adf9fcb, yanqihu)
  - DDR4-2400 under DRAMsim3
  - Sampling with SimPoint simulation points of SPEC CPU2006
    - maxK = 30, benchmarks with coverage > 80%
  - Each point with 50M warmup + 50M sampling
  - Customized RISC-V checkpoint format



* Pre-silicon estimation results. Updated May 1, 2021.

Nanhu: 2nd generation microarchitecture

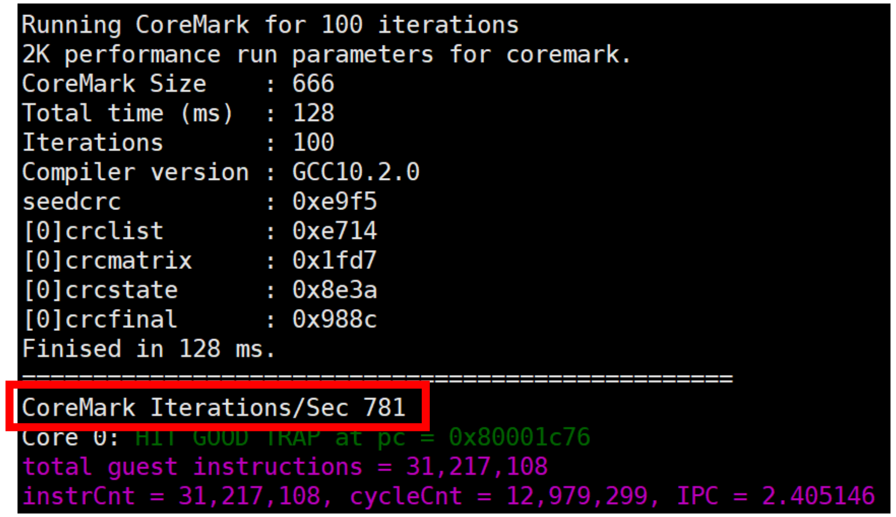
- Named after a lake in Jiaxing, Zhejiang, China
- Target: 2GHz@14nm, SPEC CPU2006 20 marks

- **Major changes**
  - New frontend design: decoupled BP and instruction fetch
  - Improved backend: better scheduler, instruction fusions, and more
  - New L2 cache: designed for high frequency and high performance
  - Tape-out with dual cores (RV64GCBK), more devices support (PCIe, USB, …)

- **Continuously optimize the performance, frequency, ..., in an agile way**

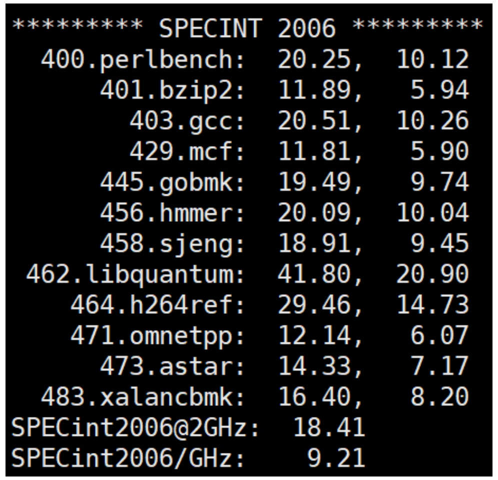
Estimated performance of Nanhu

- **Estimated SPECint 2006 18.41, SPECfp 2006 20.94@2GHz**
  - RTL simulation, DDR4-2400 under DRAMsim3
  - SimPoint simulation points of SPEC CPU2006 (RV64GCB, O2)
    - maxK = 30, each point with 20M warmup + 20M sampling

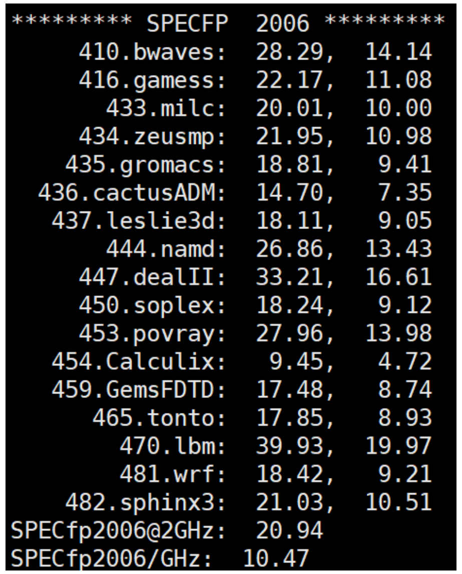


```
Running CoreMark for 100 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total time (ms)  : 128
Iterations       : 100
Compiler version : GCC10.2.0
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x988c
Finised in 128 ms.
==================================================
CoreMark Iterations/Sec 781
Core 0: HIT GOOD TRAP at pc = 0x80001c76
total guest instructions = 31,217,108
instrCnt = 31,217,108, cycleCnt = 12,979,299, IPC = 2.405146
```

7.81 CoreMark/MHz @ (RV64GCB, O2)



```
********* SPECINT 2006 *********
  400.perlbench:  20.25,  10.12
      401.bzip2:  11.89,   5.94
        403.gcc:  20.51,  10.26
        429.mcf:  11.81,   5.90
      445.gobmk:  19.49,   9.74
      456.hmmer:  20.09,  10.04
      458.sjeng:  18.91,   9.45
 462.libquantum:  41.80,  20.90
    464.h264ref:  29.46,  14.73
    471.omnetpp:  12.14,   6.07
      473.astar:  14.33,   7.17
  483.xalancbmk:  16.40,   8.20
SPECint2006@2GHz:  18.41
SPECint2006/GHz:    9.21
```

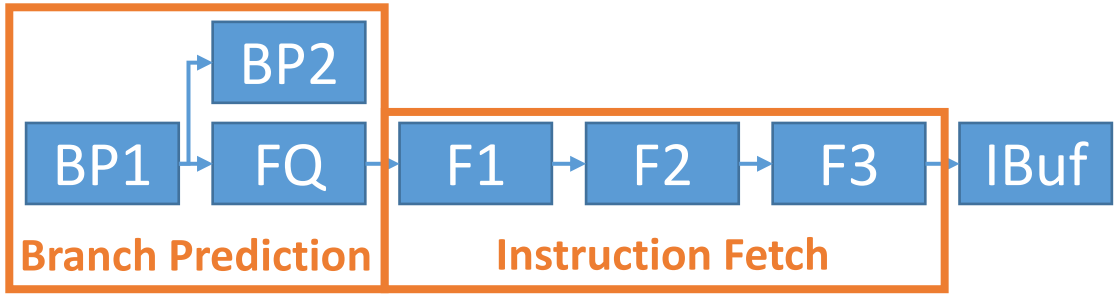


```
********* SPECFP  2006 *********
     410.bwaves:  28.29,  14.14
     416.gamess:  22.17,  11.08
       433.milc:  20.01,  10.00
     434.zeusmp:  21.95,  10.98
    435.gromacs:  18.81,   9.41
  436.cactusADM:  14.70,   7.35
   437.leslie3d:  18.11,   9.05
       444.namd:  26.86,  13.43
    447.dealII:  33.21,  16.61
     450.soplex:  18.24,   9.12
     453.povray:  27.96,  13.98
   454.Calculix:   9.45,   4.72
   459.GemsFDTD:  17.48,   8.74
      465.tonto:  17.85,   8.93
        470.lbm:  39.93,  19.97
        481.wrf:  18.42,   9.21
    482.sphinx3:  21.03,  10.51
SPECfp2006@2GHz:  20.94
SPECfp2006/GHz:  10.47
```

Estimated SPECint 2006 18.41, SPECfp 2006 20.94@2GHz

*** Updated Nov. 27, 2021**

Opt. ①: Instruction Fetch and Branch Prediction



- **Decoupled IF and BP**
- **Higher Branch Prediction Accuracy**
  - More accurate branch history
  - One-cycle earlier prediction latency of TAGE
  - ITTAGE indirect branch prediction
  - Larger RAS for the call-return instruction pair
- **Higher Fetch Throughput**
  - Hide the instruction fetch bubbles
  - BP continues when IF is blocked

Opt. ②: Execution Units

- **More instructions supported**
  - RISC-V bit-manipulation extension (RVB)
  - RISC-V scalar crypto extension (RBK)
  - Fine-Grained Address-Translation Cache Invalidation (Svinval)
- **Instruction fusion** for common cases

- **IEEE754-compatible high-performance FPU**
  - Open-source at https://github.com/OpenXiangShan/fudian
  - Dual-path floating-point adder
  - Cascade FMA (5-cycle latency)
  - Improved Int/Fp Division with SRT-16

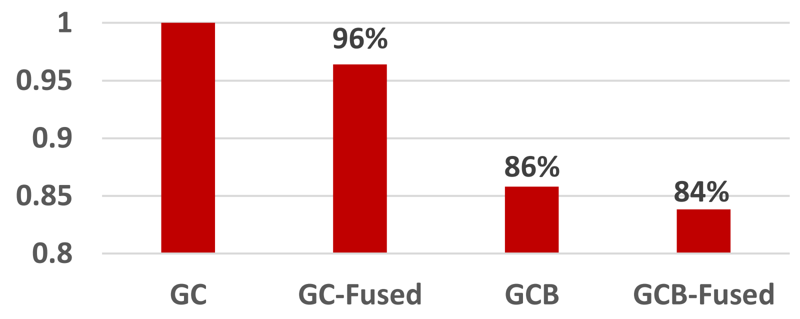


Figure. Normalized Dynamic Instruction Count for CoreMark (-O2)

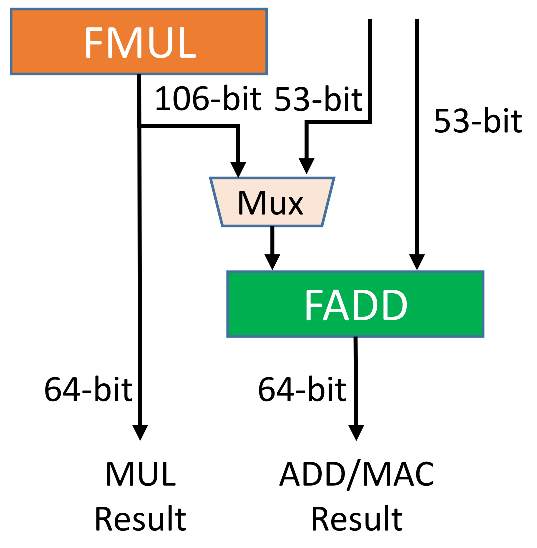


Figure. Cascade FMA

Opt. ③: Load Store Unit

- **New Features Supported**
  - Customized Configurable Physical Memory Attributes (PMA)
  - RISC-V Physical Memory Protection (PMP)
  - ECC for L1/L2/L3
  - Cache coherency between L1I and L1D (eliminated overhead of `fence.i`)
- **L1 TLB/STLB**
  - Larger size (136 entries for DTLB) and better replacement (PLRU)
  - Max. 8 in-flight PTW and memory requests supported
  - Next-line prefetcher for STLB
- **Load/Store Pipelines**
  - Virtual-address based fast data bypassing
  - Memory dependence prediction using store sets
  - Separated store address and data

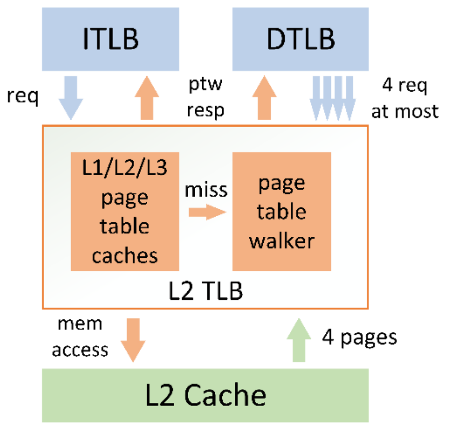


Figure. TLB Hierarchy

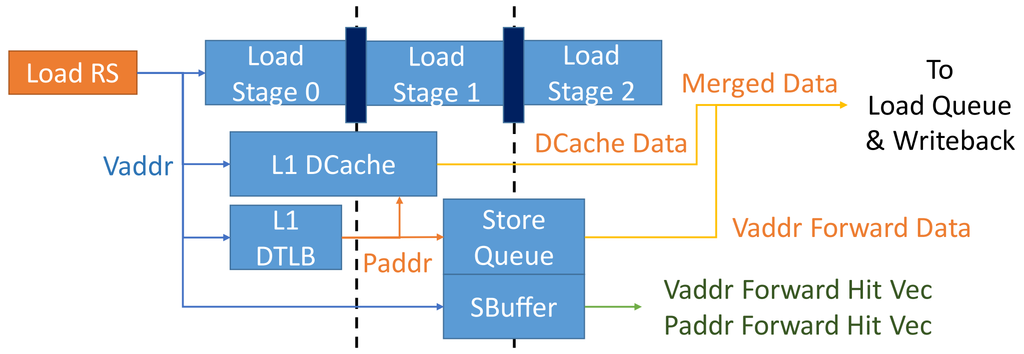
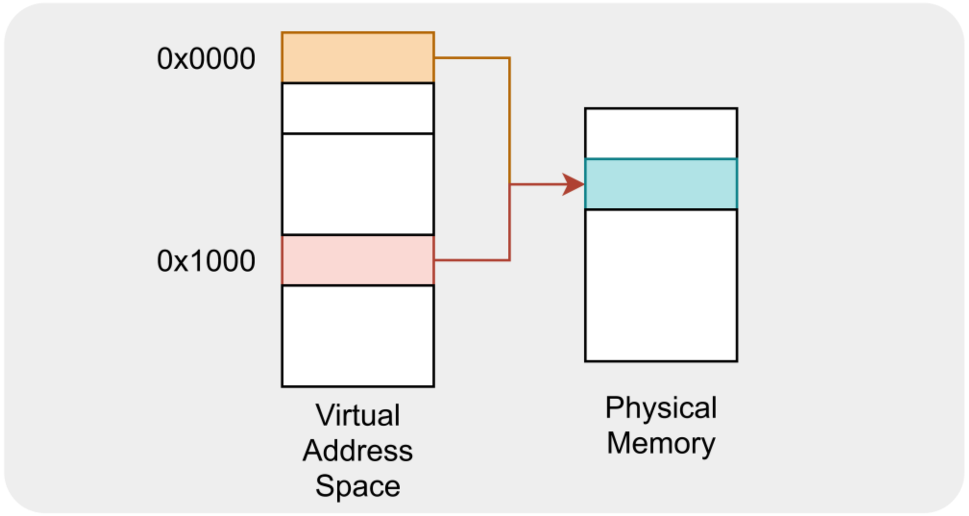
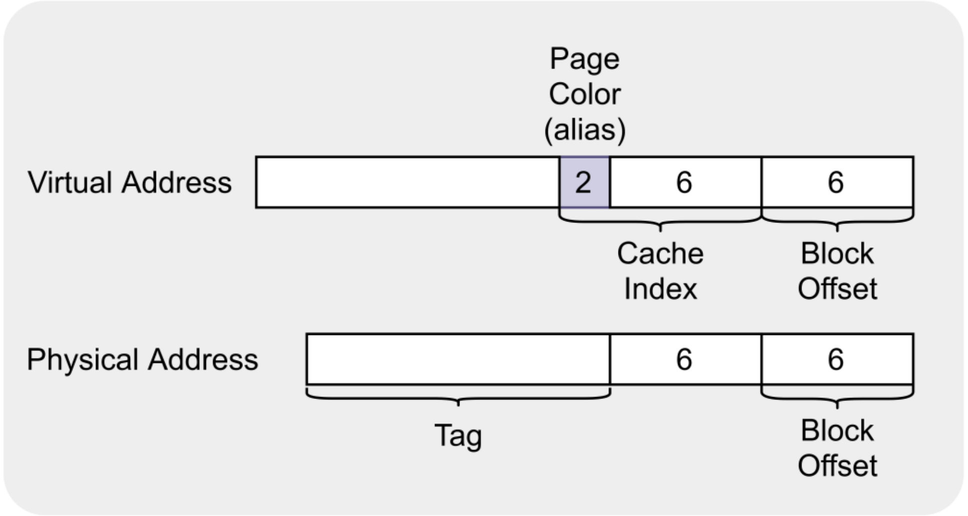


Figure. 3-cycle Load Latency

Opt. ④: Data Cache

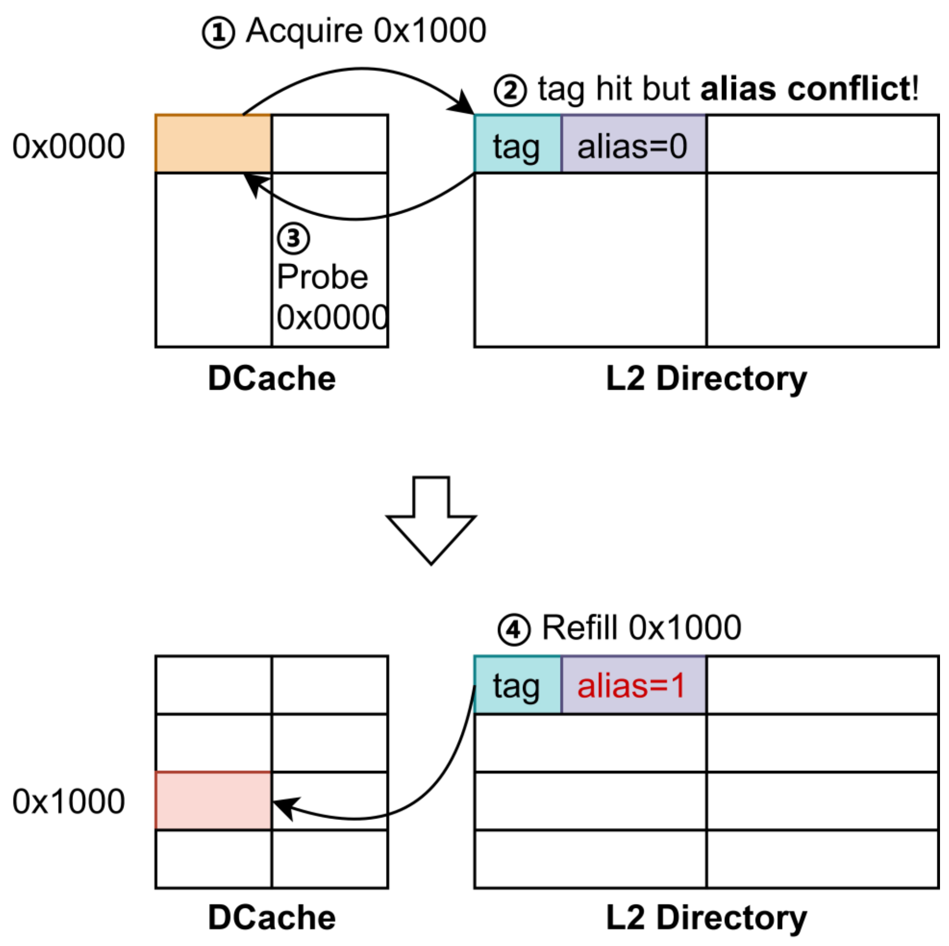
- 128KB DCache with hardware-based solution to cache alias
  - Key idea: disallow aliases (different page colors) to co-exist in a VIPT cache





Opt. ④: Data Cache

- **Hardware solution for cache alias**
  - Virtual-indexed DCache
  - Physical-indexed L2
  - L2 directory saves alias bits (page color) of the block in DCache
  - L2 will ensure only one alias exists in L1



Opt. ⑤: L2/L3 Cache

- **Non-blocking inclusive/non-inclusive cache**
  - Open-source at https://github.com/OpenXiangShan/HuanCun
  - Some design choices inspired by SiFive Block Inclusive Cache
- **Configurable parameters**
  - Size, #ways, #MSHRs
  - Inclusion policy, prefetch policy, replacement policy
- **Optimized design and improved performance**
  - Better timing and higher frequency
  - Up to 30% IPC increase on sensitive benchmarks

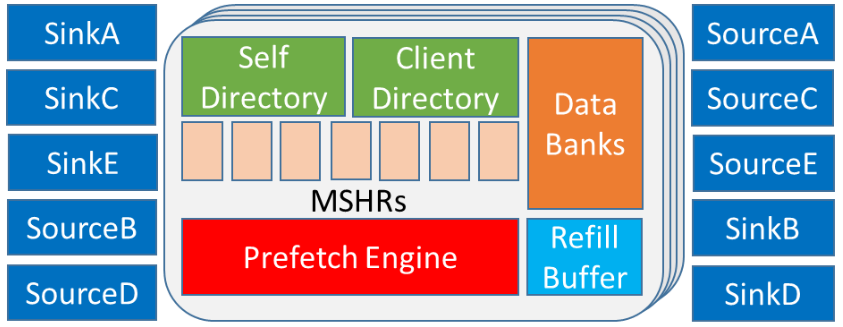


Figure. Overview of HuanCun

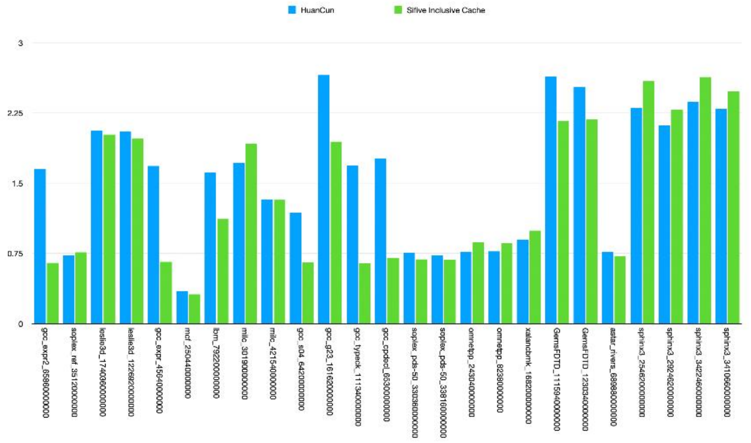


Figure. Performance improvements on some memory-sensitive checkpoints

Open-source plan

- XiangShan has been open-sourced on GitHub since June 2020
  - MulanPSLv2 License (compatible with Apache v2.0)
  - https://github.com/OpenXiangShan/XiangShan
  - Mailing list: xiangshan-all@ict.ac.cn

Scan to follow us on GitHub

- Every commit, every generation, every tool, …, **ALL on GitHub**
- Any discussions, features, patches, …, **ALL are welcome**
  - After the verification and review processes, PRs will be merged

- **Contribute to XiangShan and realize your ideas on real chips!**
  - The open-source XiangShan will be **taped-out every ~6 months**

Thanks!