МЦСТ ЭЛЬБРУС

SYRCoSE 2020

Implementation of Memory Subsystem of Cycle-Accurate Application-Level Simulator of the Elbrus Microprocessors

Pavel Poroshin, Dmitriy Znamenskiy, Alexey Meshkov

Problem Statement

- Cycle-accurate simulators (CAS) are useful tools
  - For debugging of performance problems of applications
  - For exploration of (micro)architecture design space
- Performance of modern microprocessors are largely dependent on their memory subsystem
- Quality of software model of memory subsystem plays vital role in overall accuracy of CAS
- Goal: Implement memory subsystem model as part of CAS and evaluate it accuracy

“Elbrus” Microprocessors

- Very Long Instruction Word (VLIW)
  - Each wide instruction (WI) consists of several operations, including arithmetic, memory access, control flow and other types of operations
- In-Order pipeline
- Two types of pipeline stalls
  - Ordinary stalls, that immediately block pipeline progress
  - “Deferred” stalls, that “activate” after several cycles and on activation rollback instructions from later (specific) stage to earlier (specific) one for several cycles
- Non-blocking caches
  - Cache miss on memory load potentially stalls operation that uses load result, but not load operation itself

“Elbrus” Microprocessors
Memory Subsystem

- Memory subsystem consists of
  - Cache hierarchy (L1D, L2, L3)
  - Instruction Buffer (IB, plays role of L1I)
  - Address translation structures (DTLB, ITLB, etc.)
- Besides ordinary memory access operations there are
  - Array Access Unit (AAU) – programmable device for asynchronous array prefetch
  - Automatically inserted load/store operations for filling and spilling register file contents
  - CLW device for automatic cleanup of stack
  - And others

“Elbrus” Application-Level CAS

- Based on Application-Level (AL) Instruction Set Simulator (ISS)
  - No operating system
  - No proper memory management
  - Trivial virtual to physical address translation
- Functional simulation (inherited from ISS) directly “feeds” CAS model
- Faithfull simulation of instruction pipeline state
  - Including both types of stalls and proper rollback
- “Speculative” execution of some effects with correction on miss-speculation
- Proper ordering of effects and data transfers are mostly achieved by ordering of simulation of pipeline stage effects

Memory Subsystem Model
Original Version

- Originally developed as part of System-Level Memory-Only CAS (SLM CAS), based on existing System-Level ISS
- Comprehensive support of memory subsystem of “Elbrus” microprocessors, including
  - Cache hierarchy
  - Instruction buffer
  - Address translation structures
- Basic support of instruction fetch pipeline and scoreboard
- Get input directly from SL ISS, packed in single chunk per executed instruction
- Do not provides any output to SL ISS

Memory Subsystem Model
Integration into AL CAS

- All address translation structures are disabled or function in physically addressed mode
  - AL CAS has trivial address translation
- Scoreboard and later instruction fetch pipeline stages are removed
  - They are implemented as part of pipeline model of AL CAS
- Inputs are separated into independent inputs for instruction fetch and for memory access
  - Initiation of related actions are controlled by pipeline model of AL CAS and happens at the different pipeline stages
- Simulation step function called during pipeline simulation
  - Early enough to affect possible consumer at the same tick
  - Late enough to take into account possible inputs
- Output to AL CAS implemented with callbacks
- Both implementations (for AL CAS and for SLM CAS) share most of the source code

Memory Access Operations

- Information about memory access operations is saved during functional simulation of instruction
- When instruction reaches specific pipeline stage, this information transferred to memory subsystem model
  - Necessary to be certain that there is no related miss-speculation or future unroll due to “deferred” stall, as memory subsystem model do not support “speculative” features of AL CAS or unroll of state
- Result of memory access communicated with callback

Instruction Buffer

- IB is responsible not only for ordinary instruction fetch, but also for execution of control transfer (CT) operations
  - Besides information about new instruction (its address and size), memory subsystem model is also given information about CT operations
- IB has its own pipeline, that corresponds to earliest pipeline stages of instruction
  - Also has 3 separate and parallel pipelines for control transfer preparation (CTP) operations, but we do not “faithfully” simulate them for simplicity
- In pipeline model this part of pipeline is “compressed” and represented by one “pseudo” stage in the beginning
  - Each new instruction is starts its life on the pipeline model on this stage
  - Instruction is stalled on this stage until memory subsystem model signals about successful fetch

Hardware Generated Memory Access Operations

- Some memory access operations are automatically generated by hardware and inserted into pipeline
  - For example, operations for spilling/filling register file to/from memory
- Functional part of AL CAS already simulates such operations
  - It happens during execution of ordinary instruction
- To support them in AL CAS:
  - During functional execution all information about HW generated operations is saved in intermediate storage
  - When this instruction transferred to pipeline model, all this saved information is placed alongside with other data of the instruction
  - When instruction reaches certain pipeline stage, information of HW generated operations is used to populate pipeline (instead of next instruction)

Debug Facilities

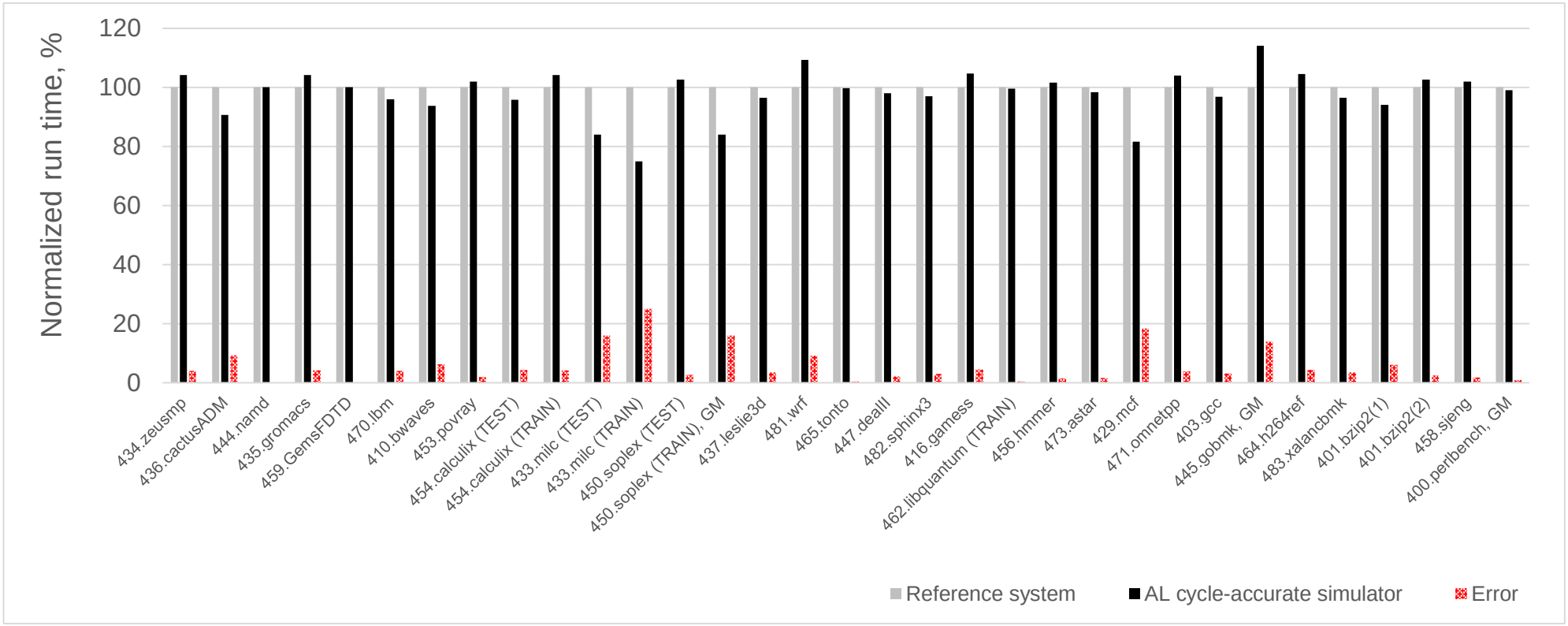
- Execution Log
  - Due to “speculative” features of CAS, logging events accumulated in tick-indexed buffer, which is filtered on miss-speculation
- Execution Statistic
  - Accumulates counters for each WI
  - RAW output generated at the end of the simulation
  - Separated tools for transforming into more human readable form
    - Annotated disassembler
    - Summary for functions
    - Call-graph, compatible with Kcachegrind for visualization
- Both use universal mechanism for identifying / marking events

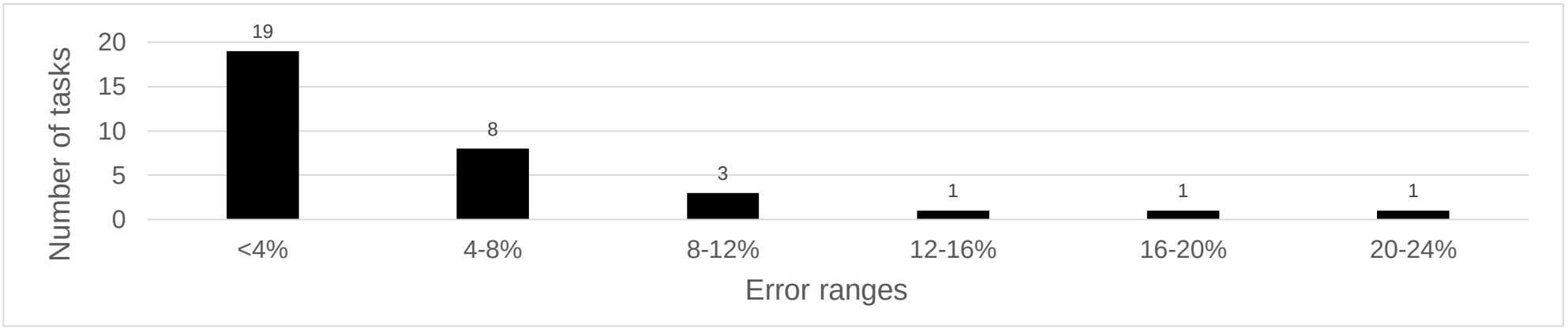
Accuracy Evaluation

- Test cases from SPEC CPU2006 benchmarks
  - Used with test data sets due to long times of simulation
- Compiled by optimizing compiler developed by MCST
  - Highest optimization level (O3)
  - No use of AAU (due to its incomplete implementation in CAS)
- Real 8-core Elbrus CPU as reference

CPU	Elbrus, 8 cores, 1 node
Clock frequency	1500 MHz
L1 instruction cache (IB)	128 KBytes, 256-byte cache line, 4-way set-associative, virtually addressed
L1D cache	64 Kbytes, 32-byte cache line, 4-way set-associative, virtually addressed
L2 cache	512 Kbytes, 64-byte cache line, 4-way set-associative, 4-banks interleaved, physically addressed
Shared L3 cache	16 Mbytes, 64-byte cache line, 16-way set-associative, 8-banks interleaved, physically addressed

Accuracy Evaluation





Sources of Errors
433.milc

- Error of 16% for test data and 25% for train data
- Unusually high DTLB miss rate
  - 1% of all memory accesses are due to DTLB misses (on reference system)
  - Orders of magnitude more than for most other tasks
- Application-Level CAS in question does not simulate DTLB
- Unaccounted by CAS DTLB misses adequately explain difference in total cycles
- Possible solutions (?):
  - Simplified heuristic for DTLB misses
  - Implement memory management as part of AL CAS

Sources of Errors
429.mcf & 450.soplex

- Error of 18% for 429.mcf and 13% for 450.soplex (on train data)
- Higher L1D hit-rate on CAS than reference system
  - Likely logical inaccuracy in CAS
- Effects of memory management by OS
  - COW pages, zero pages, shared pages etc.
  - Different physical addresses patterns
  - Possible more physical addresses conflicts
- AL CAS in question has trivial virtual address mapping
- Possible solution (?):
  - Implement memory management as part of AL CAS

Sources of Errors
445.gobmk

- Error of 14%
- Workload consists of many short subtasks
- High runtime variation of reference system on short subtasks

Performance Evaluation

Workload		CFP			CINT		
		453.povray	434.zeusmp	410.bwaves	471.omnetpp	462.libquantum	429.mcf
Clocks per Wide Instruction		1.6	3.1	6.6	1.7	3.5	4.6
Machine		Intel Core i7-2600 CPU @ 3.40GHz, 16GB RAM		Intel Xeon E3-12xx v2 (Ivy Bridge), 32GB RAM			
Simulator speed, MHz	FUNC	5.081	4.152	9.442	3.579	3.361	10.565
	AL	0.796	0.670	0.817	0.462	0.609	0.795
	SLM	0.734	0.699	0.965	0.577	0.733	1.066
Speed (FUNC) / speed (AL)		8.37	8.43	12.67	6.30	5.43	13.77
Speed (AL) / speed (SLM)		1.09	0.96	0.85	0.80	0.83	0.75

- AL CAS is 5 to 14 times slower than AL ISS
- Highest slowdown is on high CPWI task slowdown
- AL CAS slower by 13% on average than SLM CAS
  - Overhead of whole cycle accurate pipeline of AL CAS is moderately greater than simulation of MMU and other system level components

Conclusion

- Described Application-Level CAS achieves good runtime accuracy on average (average error 5%)
- Simulator achieves expected performance (order of magnitude slowdown on average relative to ISS)
- Debug tools greatly helped in finding and fixing mistakes in simulation

But

- There is still inaccuracies
  - Current accuracy is achieved on feature incomplete model (most notably without use of AAU)
  - Memory management effects are not accounted for
- Many tasks are still too long to run

Future Work

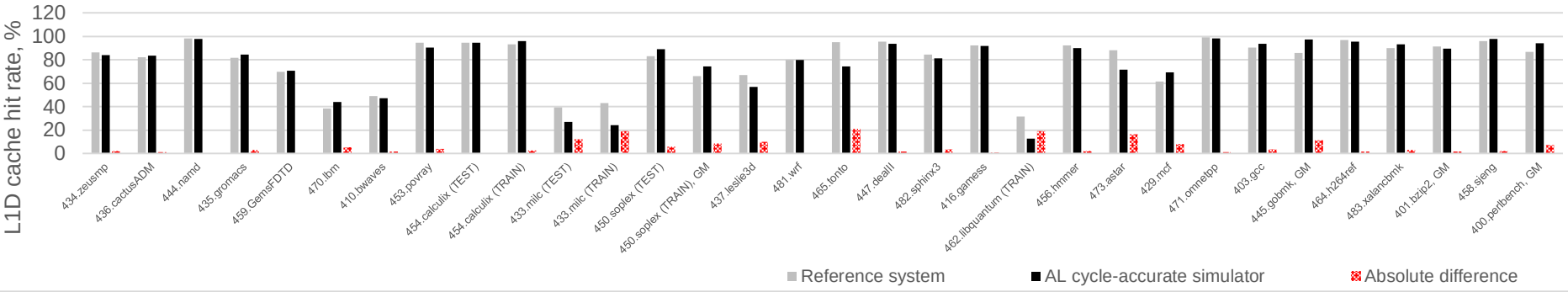
- Further improvement of accuracy
  - Support of missing components (first and foremost, AAU)
  - Investigating of ways to account for memory management effects
  - General polish
- Improvement simulation speed
  - Code optimization
  - Removing unnecessary details
  - Investigating of “smart” ways to achieve speedup
- Further development of debug tools
- Support of next versions of “Elbrus” ISA

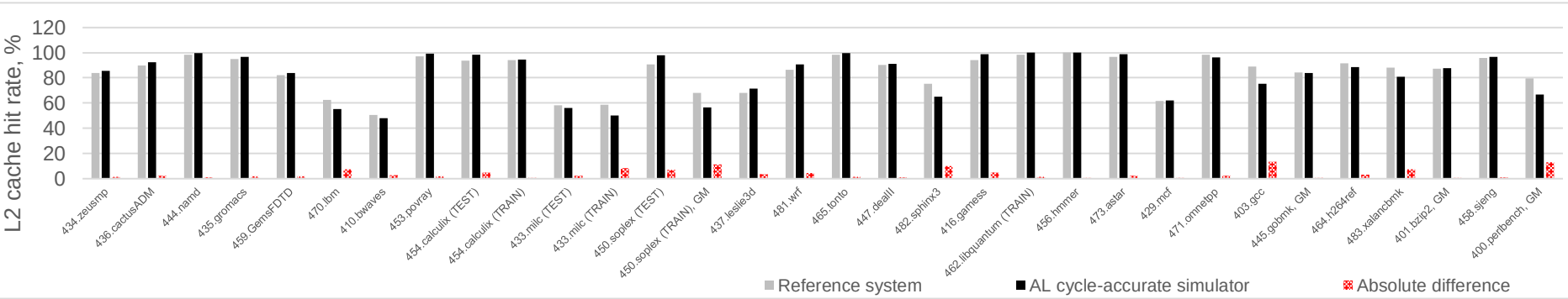
Previous Work

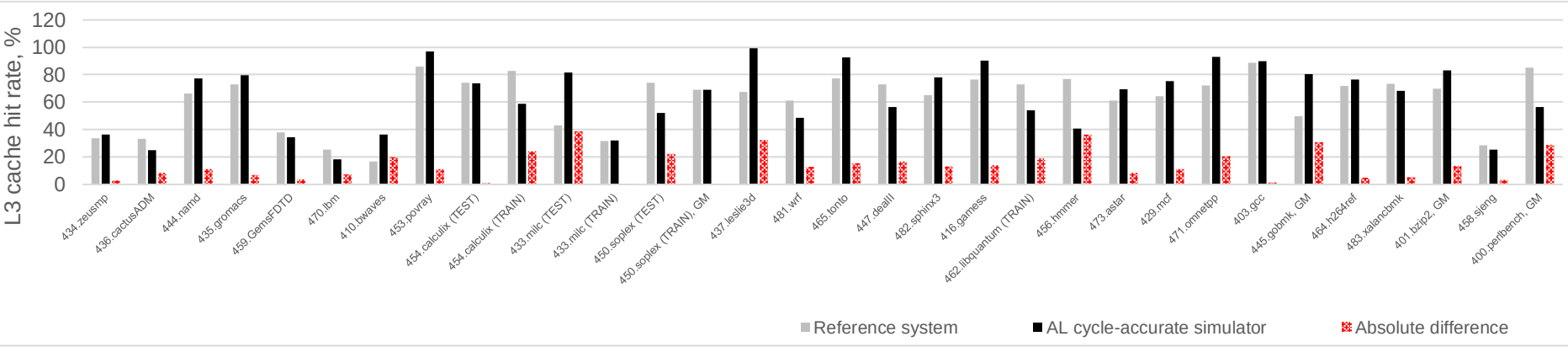
- Memory subsystem model of SLM CAS
  - Znamenskiy D.V., Kutsevol V.N. Development of a cycle-accurate simulator of the Elbrus processor core memory subsystem. Radio industry (Russia), 2019, vol. 29, no. 2, pp. 17-27
- General design of pipeline model of AL CAS
  - Poroshin P.A., Meshkov A.N. An exploration of approaches to instruction pipeline implementation for cycle-accurate simulators of «Elbrus» Microprocessors. Proc. ISP RAS, 2019, vol. 31, no. 3, pp. 47-58

Thank you for listening!

Accuracy Evaluation







Reference System Configuration

CPU	Elbrus, 8 cores, 1 node
Clock frequency	1500 MHz
L1 instruction cache (IB)	128 KBytes, 256-byte cache line, 4-way set-associative, virtually addressed
L1D cache	64 Kbytes, 32-byte cache line, 4-way set-associative, virtually addressed
L2 cache	512 Kbytes, 64-byte cache line, 4-way set-associative, 4-banks interleaved, physically addressed
Shared L3 cache	16 Mbytes, 64-byte cache line, 16-way set-associative, 8-banks interleaved, physically addressed
RAM channels	4 channels DDR4
RAM size	DDR4-2400, 128 GBytes
Operating system	OS Elbrus based on the Linux kernel
Linux kernel version	4.9.0-4.1-e8c2

Overall AL CAS Design