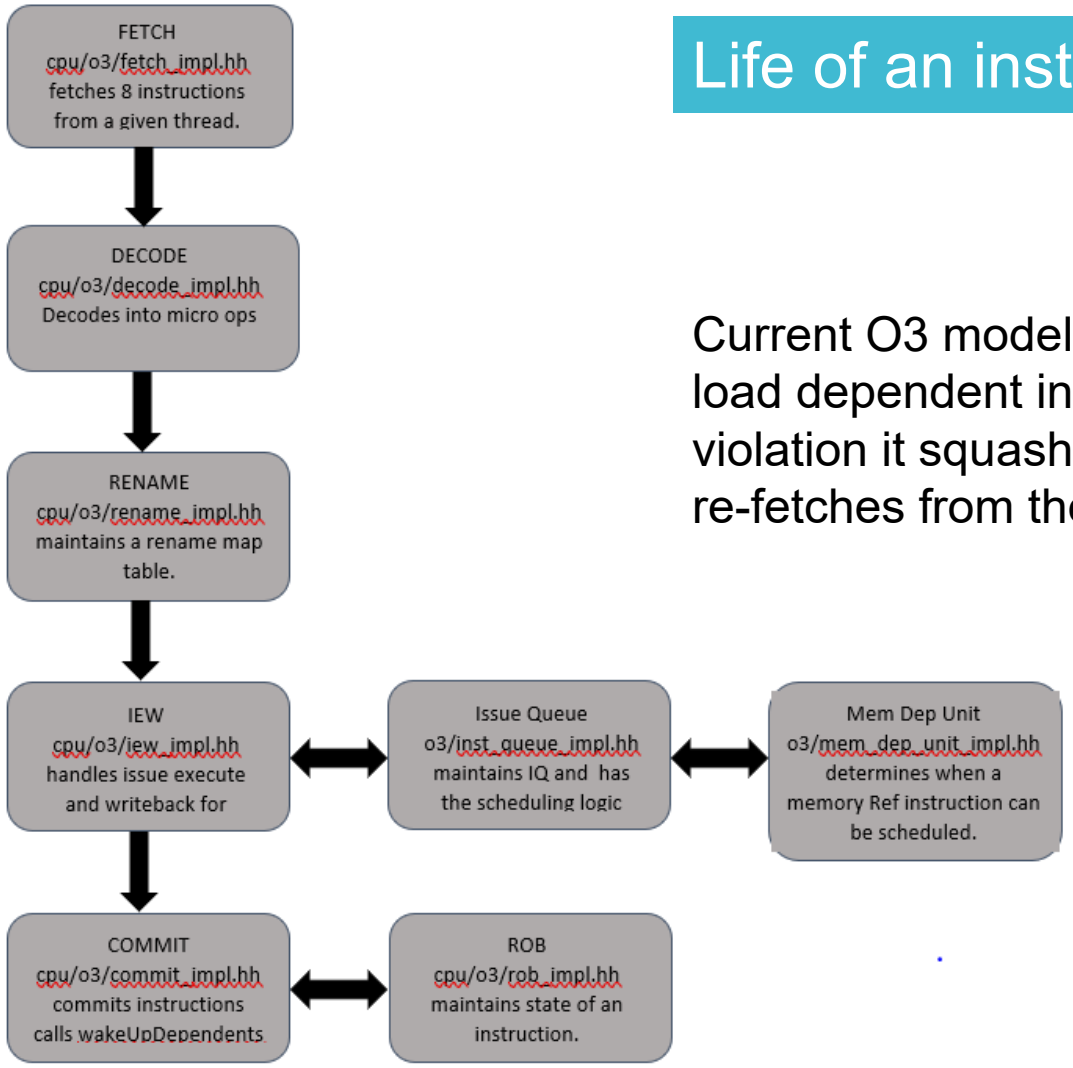
Adding Selective Replay and Bank Conflict Prediction to the Out-of-Order CPU model in Gem5

Pradeep Kumar

Nikhita Kunati

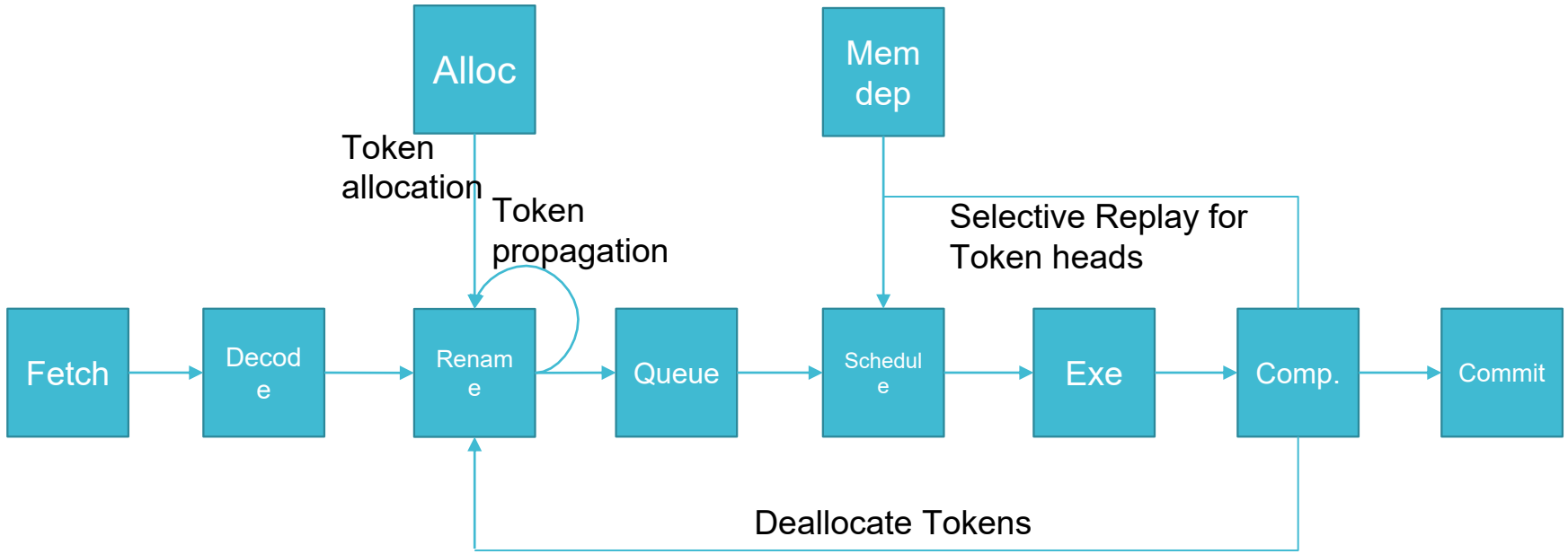
Life of an instruction in the O3 CPU



Current O3 model doesn't allow speculative scheduling of load dependent instructions and on a memory order violation it squashes all the instructions in the pipeline and re-fetches from the violating PC

SELECTIVE TOKEN BASED REPLAY

IDEA – Allocate a token to every load and every dependent instruction inherits the token id in a dependence vector



Architected Register	Physical Register	Dependence vector
R2	p2	00000000
R1	p9	00000001
R4	p7	00000001
R3	p4	00000011
R5	P11	00000011

LOAD R1, [R2] → Token ID 1
ADD R4,R1,R2
LOAD R3,[R4] → Token ID 2
SUB R5,R3,R4

While renaming dest regs once mapping is done

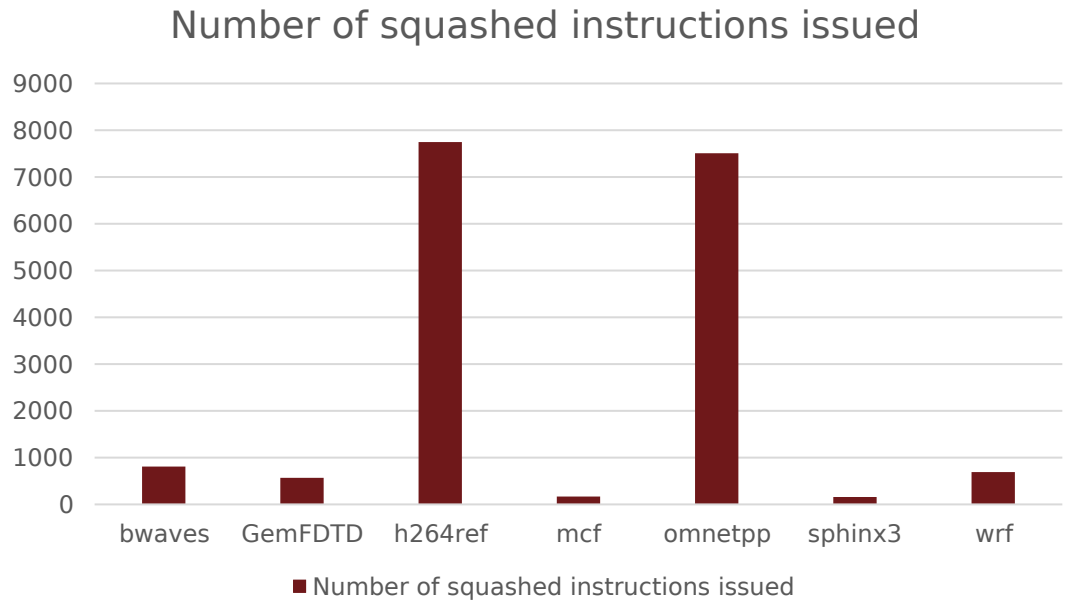
```
If (inst->isLoad) {
  depVector[renamedDestReg] = depVector[renamedDestReg]  | ( 1 << (tokenID -1)
}
Iterate(renamedSrcRegs) {
  depVector[renamedDestReg] = dep[renamedDestReg] |
depVector[renamedSrcReg]
}
```

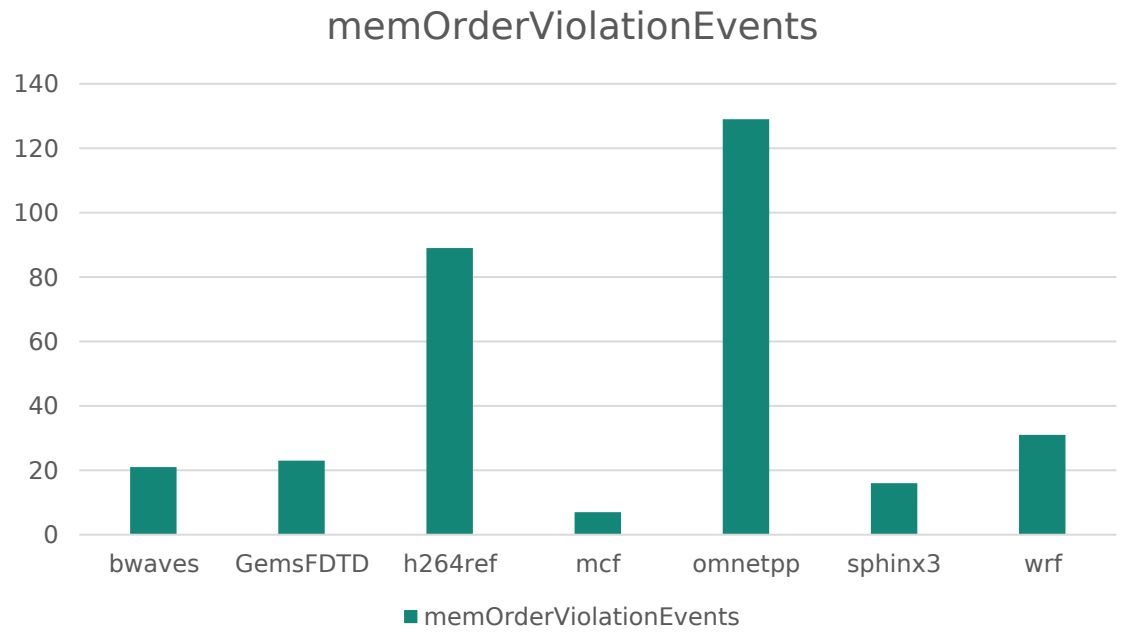
Changes made to O3 source code

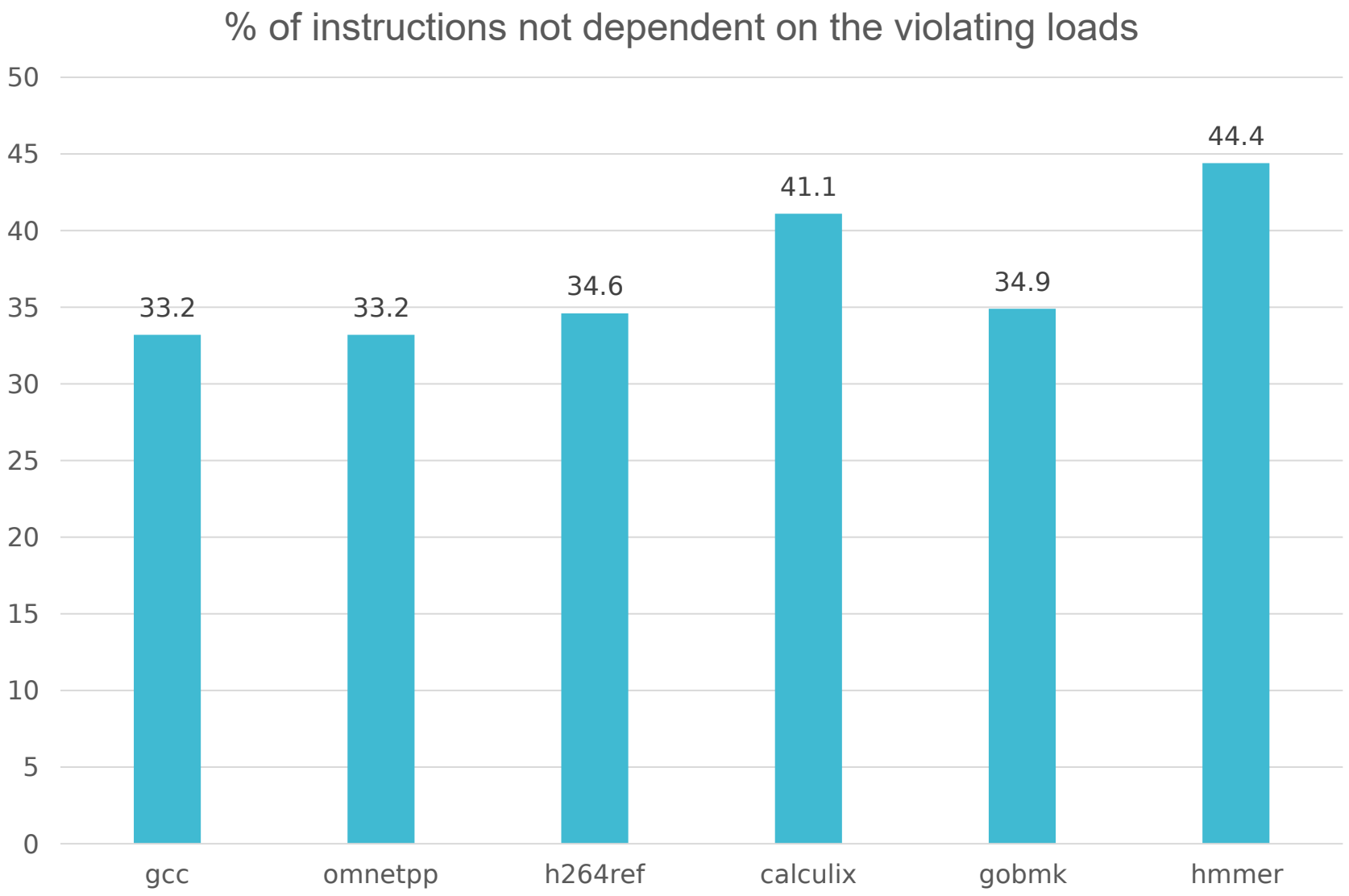
- rename_impl.hh, rename.hh -> tokenID allocation/deallocation, dependence vector calculation, clearing dependence vector
- In the inst_queue_impl.hh instead of popping entries from the instList during commit we added a logic to only pop entries with dependence vector as all zeroes. We maintain a replayQueue which has instructions with non-zero Dependence vectors and remove insts when the commit.
- In lsq_unit_impl.hh when we detect a memory order violation we replay->notify(tokenID). And in the next cycle in issue stage we reissue instructions that match the tokenID from replayQueue.
- Changes made to Fetch and decode and ROB to not squash instructions.

EVALUATIO

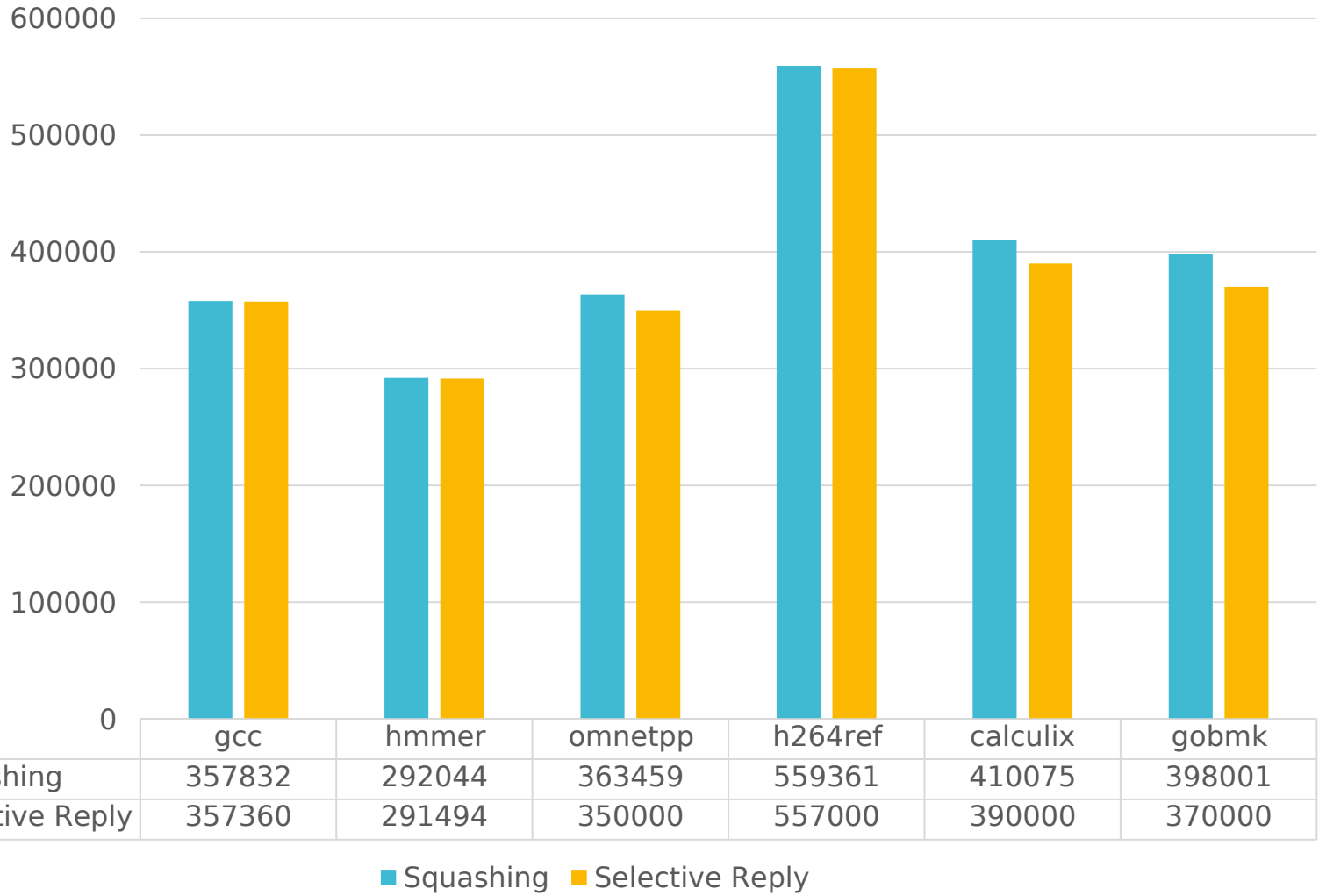
OoO Execution	**8-wide fetch/issue/commit, 256 ROB, 128 LSQ, 128 issue queue entries**
Functional Units(Latency)	8 INT ALUs(1), 4 FP ALUs(2), 4 INT MULT/DIV(3/20), 4 FP MULT/DIV(4/24), 4 general memory ports
Branch Prediction	Tournament Predictor
Memory System(Latency)	32KB 2-way 64B line L1I(2), 32KB 4-way 64B line L1D(2), 512KB 4-way 64B line L2(8), 8193MB main memory(1ns)






% of instructions not dependent on the violating loads
50
45
40
35
30
25
20
15
10
5
0
33.2
33.2
34.6
41.1
34.9
44.4
gcc
omnetpp
h264ref
calculix
gobmk
hmmer

Number of squashed instructions that are issued



	gcc	hmmer	omnetpp	h264ref	calculix	gobmk
Squashing	357832	292044	363459	559361	410075	398001
Selective Reply	357360	291494	350000	557000	390000	370000

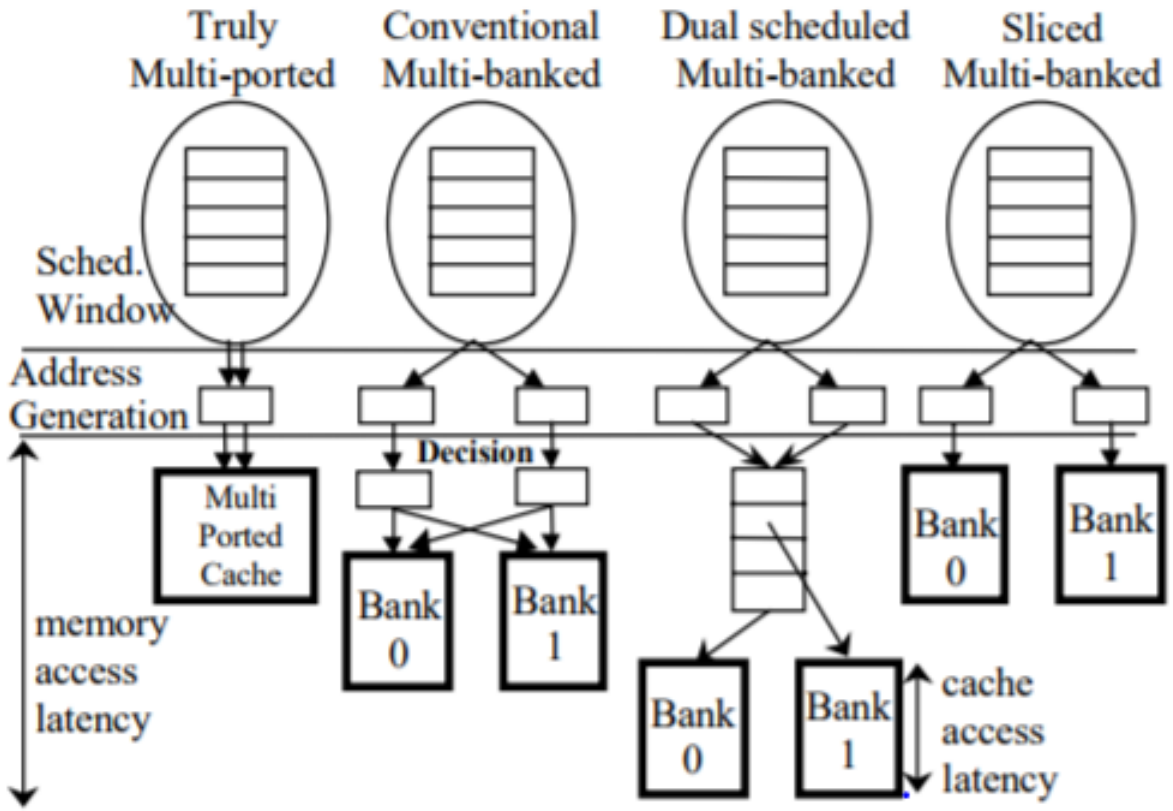
BANK CONFLICT PREDICTION

MOTIVATION – To allow more than one concurrent memory reference Multi-Banked caches were proposed as a less expensive option compared to multiported cache however possible bank conflicts may pose as a limitation

IDEA – Use Bank conflict prediction to intelligently schedule load instructions that have less chance of conflicting.

Previous Work on Bank Prediction

Previously proposed technique involves predicting a bank for each load which will improve scheduling of loads as well as simplify the memory execution pipeline

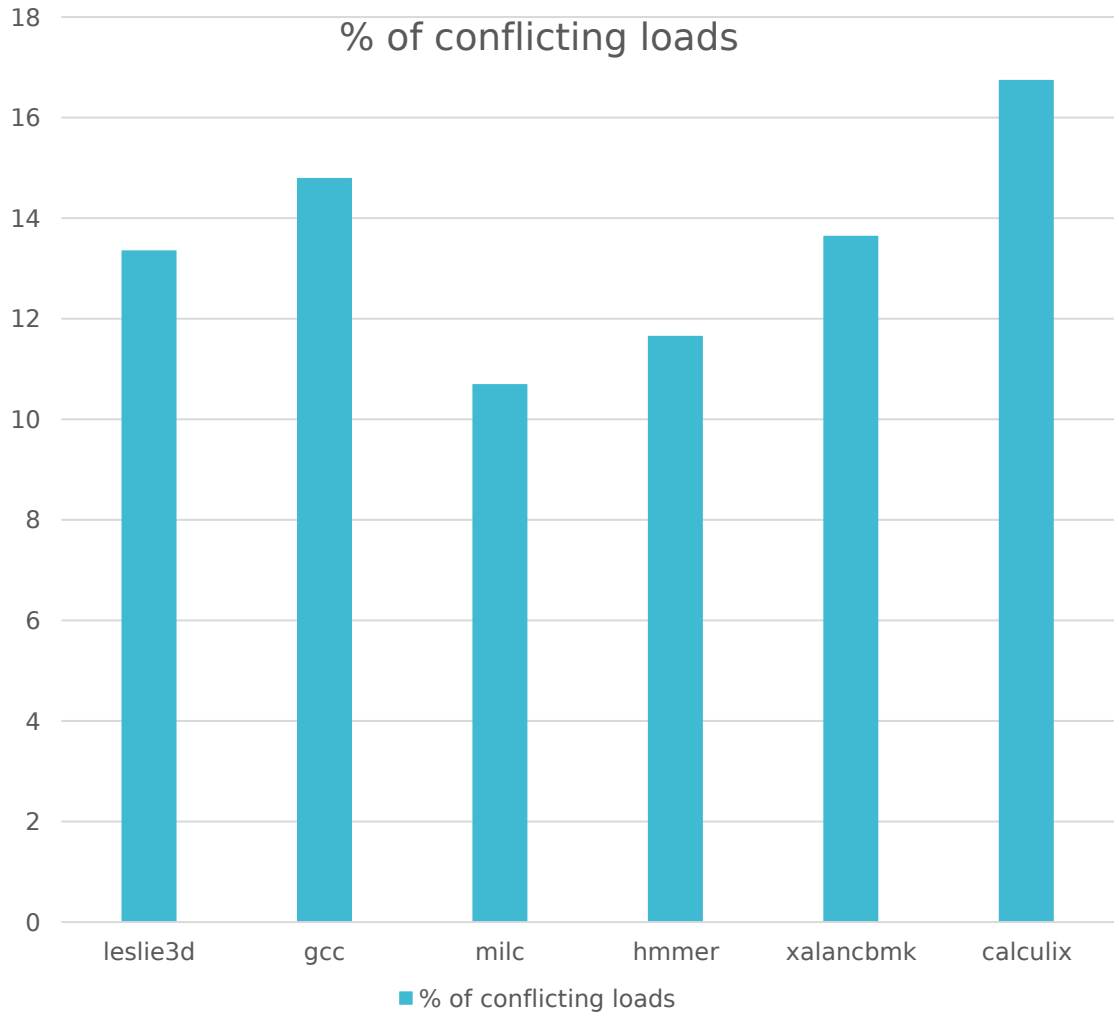


Load	Load	
CONFLICT	CONFLICT	✗
CONFLICT	NOCONFLICT	✓
NOCONFLICT	CONFLICT	✓
NOCONFLICT	NOCONFLICT	✓

Ready Queue
Load
Sub
Add
Store
Load
Add
Add
Mov

OoO Execution	8-wide fetch/issue/commit, 256 ROB, 128 LSQ, 128 issue queue entries
Memory System(Latency)	32KB 2-way 64B line L1I(2), 32KB 4-way 64B line L1D(2), 512KB 4-way 64B line L2(8), 8193MB main memory(1ns)

Benchmark	Total number of loads	Total number of bank conflicting Loads
Leslie3d	32534	4348
Gcc	25648	3816
Milc	23012	2470
hmmer	29049	3389
calculix	35507	5949
Xalancbmk	27936	3814



Changes in Gem5 Source code

Made a record of all the load instructions that got scheduled at the same cycle.

Prediction for each load was made based on their PC Address.

If there are more than one conflicting predictions then Scheduling logic pushes the load instruction back to the ready queue for the next cycle.

In the IEW Unit,

When the address for all the same cycle dispatched load instructions was calculated, we can easily find out the number of actually conflicted loads and thus update the conflict predictor table along with the hashed PC address.(This is how we train the predictor).

Changes in IQ::scheduleReadyInsts()

Changes in IEW::**ExecuteLoad()**

Difficulty faced while implementing in Gem5

No L1D cache banks option available in Gem5.
L2 cache banking is implemented in Ruby cache simulator.(Part of Gem5)
But very hard to replicate the same thing for L1D.
We tried but failed to do so.

A common Data structure was required for Scheduling logic to read while predicting and executeLoad() function to update it.

Both were happening in different cycle for the same load.
We spent a lot time in figuring out and then moved on to generate Trace.

We generated traces(PC address + effective mem address) of all the load instructions that got issued in the same cycle by making some changes in the execute stage of gem5.
Used These traces for an offline predictor that we wrote in Perl.

FUTURE WORK

- Improvise token-ID allocation to loads with confidence estimator
- Add speculative scheduling of dependent instructions to see more benfits of selective replay.
- Handle inflight instructions in the O3 cpu model cleanely on the event of a memory order violation

References

[1] I. Kim and M. Lipasti. “Understanding Scheduling Replay Schemes.” In Proc. 10th International Symposium on High Performance Computer Architecture, Feb. 2004.

[2] A. Yoaz, E. Mattan, R. Ronen, and S. Jourdan.: Speculation Techniques for Improving Load Related Instruction Scheduling. Proc. of 26th ISCA (1999) 42–53

[3] The gem5 Simulator. Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. May 2011, ACM SIGARCH Computer Architecture News.