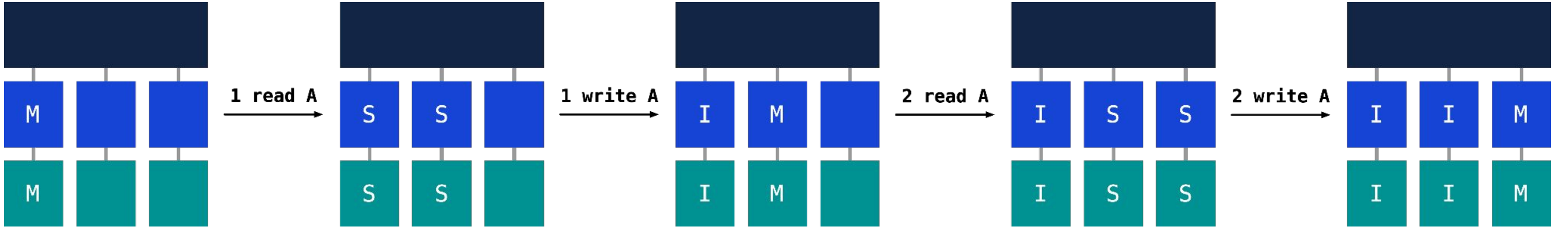
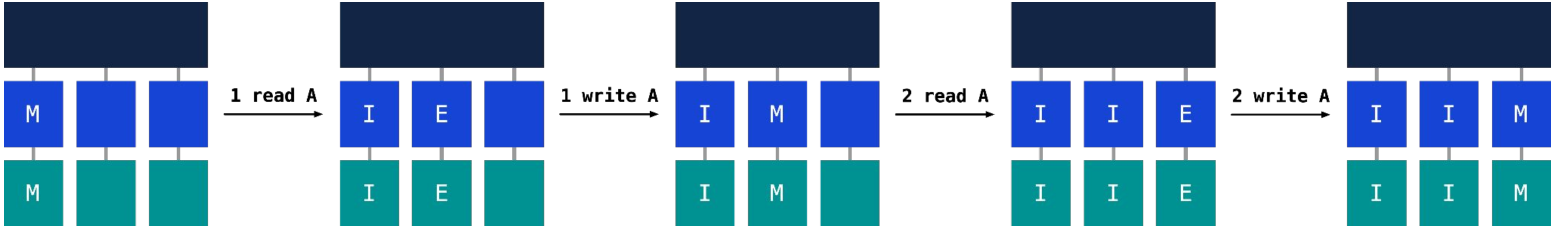
① Background

ZSim: An x86-64 multicore simulator, with fully directory-based cache system following a tree structure (LLC is root node at top, L1d and L1i are leaf nodes at bottom)

Migratory Sharing: Multiple processors take turns to read and modify shared resources one at a time, e.g. accessing shared data structures inside a critical section



Issue: If migratory sharing can be detected, the access pattern can be optimized!

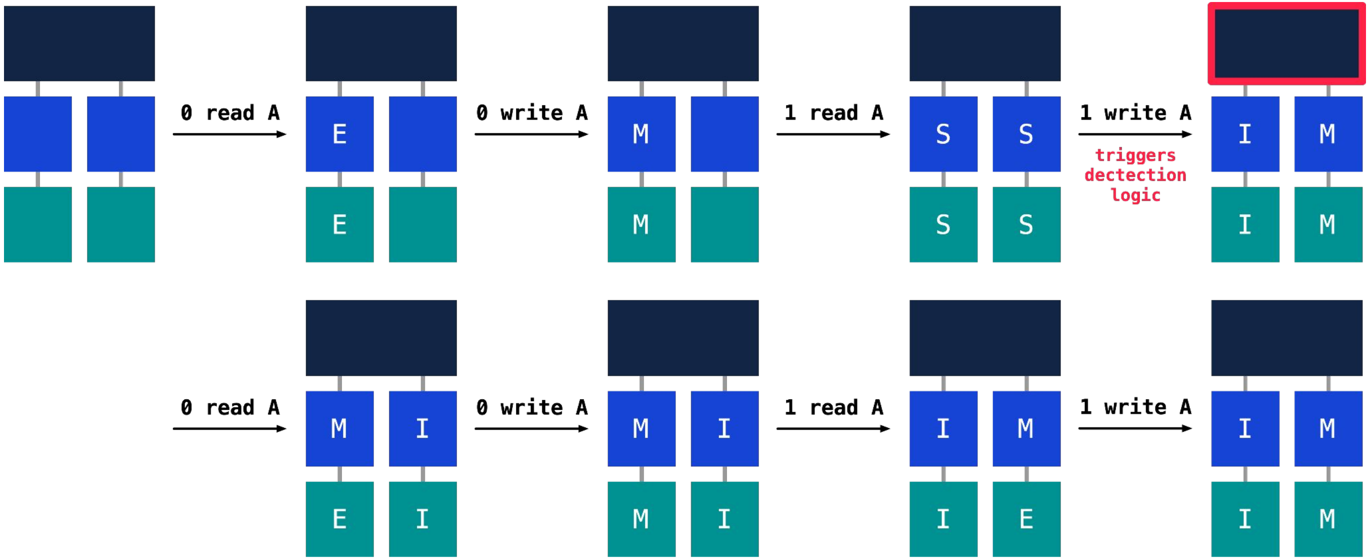


② Algorithm

Trigger detection when read-exclusive request on a non-migratory line

If the line has two sharers at this moment, and the last read-exclusive requester is NOT the current one → mark the line migratory

Migratory mark gone when the line is evicted or number of sharers > 2 at any time



Edge Cases: what if some processor changes behavior to only read the migratory object after the object is marked migratory? (Alteration of Access Pattern)

③ Experiment Setup

Micro-benchmark: Improvement more visible as tests have the access patterns we define; runnable in our VMs with limited computing capabilities

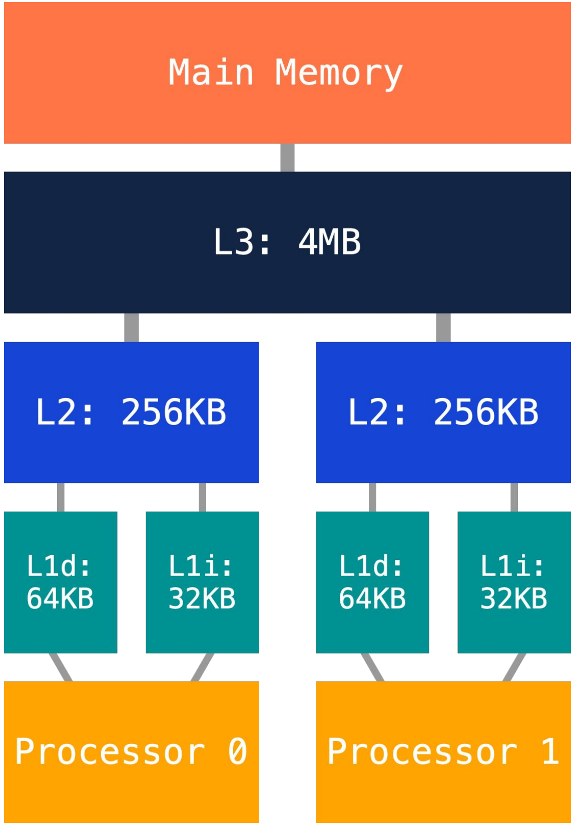
Machine: Ubuntu 16.04 VM with 2 vCPUs on a x86-64 PC

Interested Stats:

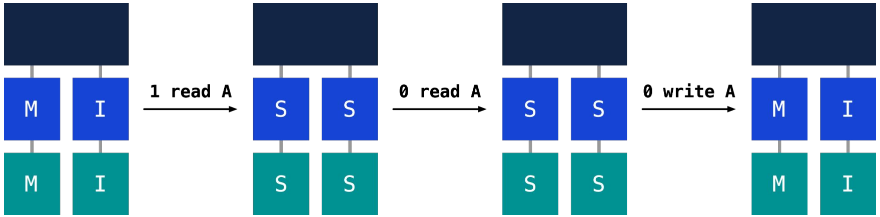
Instructions Simulated
Cycles Simulated
L1d/L2 Upgrade Miss Count
L1d/L2 True Invalidation (INV) Count
L1d/L2 Downgrade Invalidation (INVX) Count

* Each stats has a Core ID (0 or 1)
and ZSim version used (default or optimized)

* Upgrade Miss:	Read-Exclusive on S State
* True Invalidation:	Non-I State Change to I State
* Downgrade Invalidation:	M/E State Change to S State



④ Results: Test 0 & Test 1

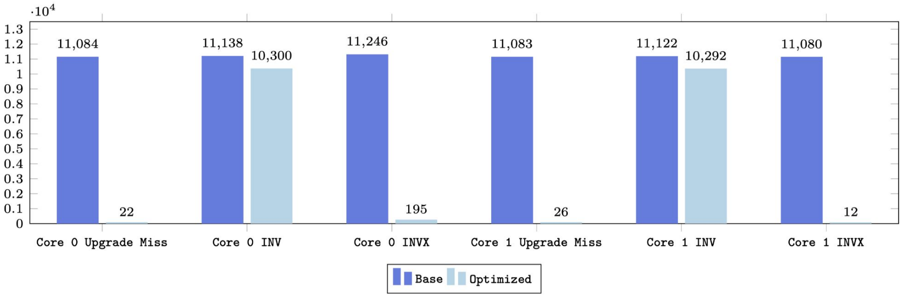


Test 0:

- ➔ Verify improvement on migratory sharing pattern
- ➔ Migratory object is an <u>integer array</u>

Table 1: Instruction and Cycle Count of Test 0

Implementation	Core 0 ins	Core 0 cycle	Core 0 IPC	Core 1 ins	Core 1 cycle	Core 1 IPC	Average IPC improvement (%)
Base	40959334	51574552	0.7972943342	40775651	51142532	0.7972943342	0.1513786423
Optimized	40960190	51497937	0.7985053393	40776932	51066574	0.7985053393	

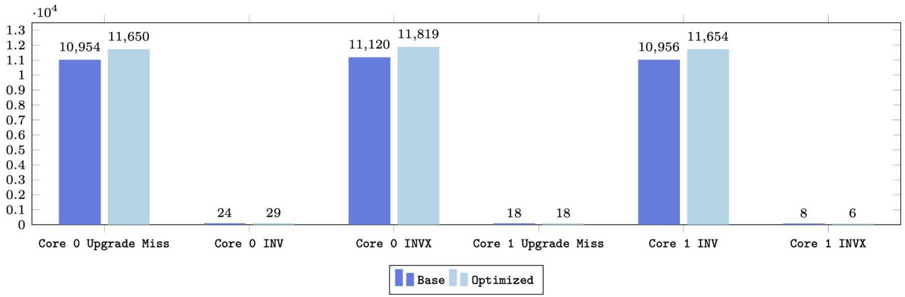


Test 1:

- ➔ Verify non-migratory object not detected as migratory object
- ➔ A naïve <u>producer-consumer pattern</u> (graph above)

Table 2: Instruction and Cycle Count of Test 1

Implementation	Core 0 ins	Core 0 cycle	Core 0 IPC	Core 1 ins	Core 1 cycle	Core 1 IPC	Average IPC improvement (%)
Base	40848255	51297351	0.7963033998	40478789	50444215	0.8024466036	-0.001147717365
Optimized	40848255	51297164	0.7963063026	40478789	50445551	0.8024253516	

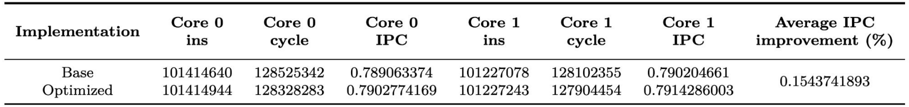


⑤ Results: Test 2 & Test 3

Test 2 & Test 3:

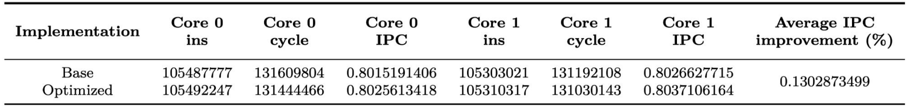
- ➔ Extensions of test 0 with different migratory objects and increased workload
- ➔ Verify improvement on migratory sharing pattern
- ➔ Migratory object is a linked list for test 2, a struct array for test 3

Table 3: Instruction and Cycle Count of Test 2

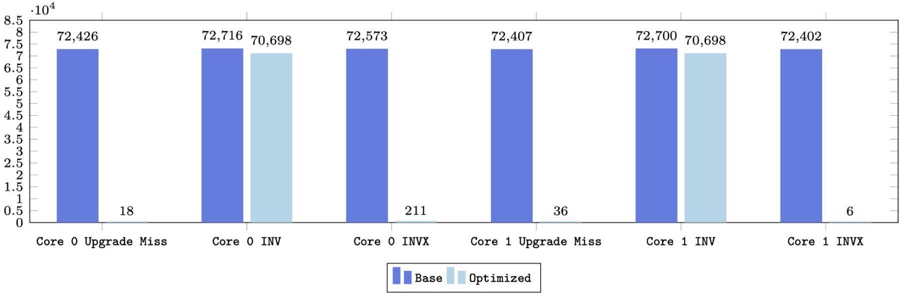


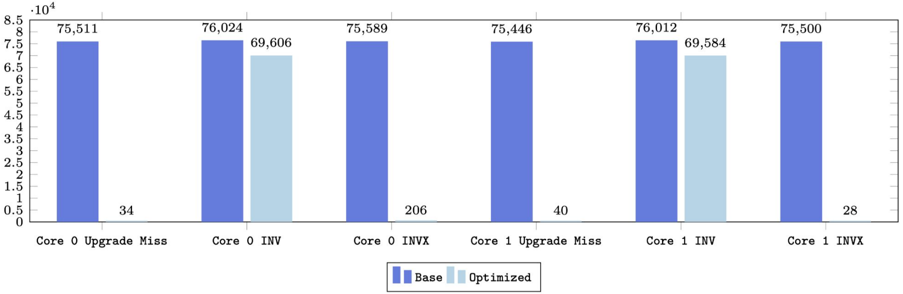
Implementation	Core 0 ins	Core 0 cycle	Core 0 IPC	Core 1 ins	Core 1 cycle	Core 1 IPC	Average IPC improvement (%)
Base Optimized	101414640 101414944	128525342 128328283	0.789063374 0.7902774169	101227078 101227243	128102355 127904454	0.790204661 0.7914286003	0.1543741893

Table 4: Instruction and Cycle Count of Test 3

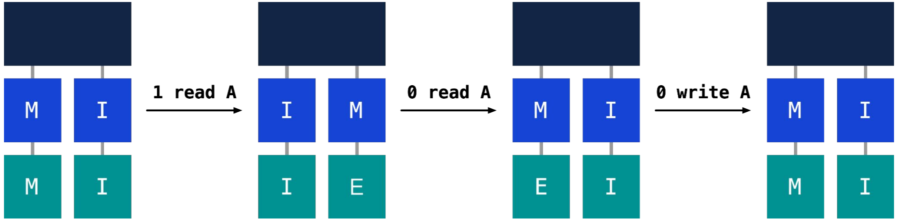


Implementation	Core 0 ins	Core 0 cycle	Core 0 IPC	Core 1 ins	Core 1 cycle	Core 1 IPC	Average IPC improvement (%)
Base Optimized	105487777 105492247	131609804 131444466	0.8015191406 0.8025613418	105303021 105310317	131192108 131030143	0.8026627715 0.8037106164	0.1302873499





⑥ Results: Test 4 & Test 5

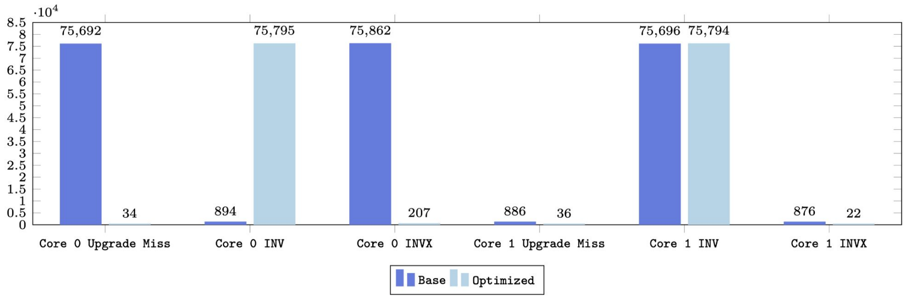


Test 4:

- ➔ Migratory sharing changes to naïve producer-consumer pattern
- ➔ Still treated as migratory, no degradation (graph above)

Table 5: Instruction and Cycle Count of Test 4

Implementation	Core 0 ins	Core 0 cycle	Core 0 IPC	Core 1 ins	Core 1 cycle	Core 1 IPC	Average IPC improvement (%)
Base	105473191	131292665	0.8033441244	104077090	128027300	0.8129288831	0.002278383362
Optimized	105473152	131290458	0.8033573316	104077044	128023524	0.8129525008	

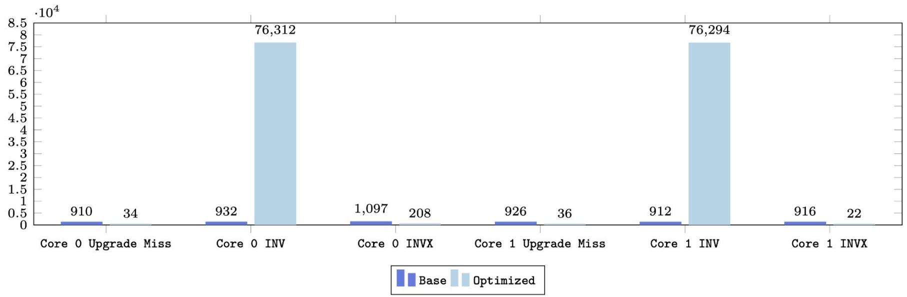


Test 5:

- ➔ Migratory sharing changes to read-only pattern
- ➔ Still treated as migratory, processors invalidate each other, degradation observed

Table 6: Instruction and Cycle Count of Test 5

Implementation	Core 0 ins	Core 0 cycle	Core 0 IPC	Core 1 ins	Core 1 cycle	Core 1 IPC	Average IPC improvement (%)
Base	104260413	127778163	0.8159485984	104077044	127345461	0.8172811436	-0.5407563013
Optimized	104260600	128481455	0.8114836495	104077217	128029753	0.8129142997	

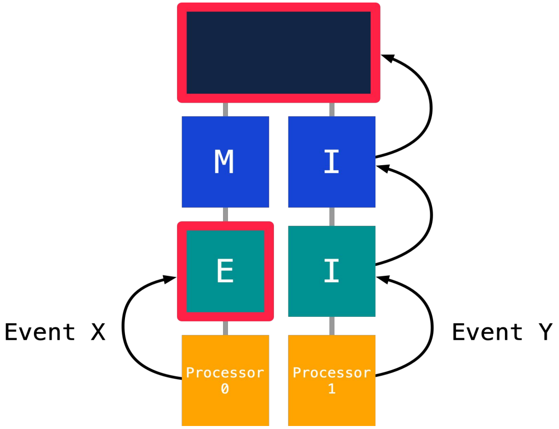


⑦ Fix Edge Cases

After migratory detection on A, a read from core 0 is treated as read-exclusive, granting 0 the exclusive copy of A

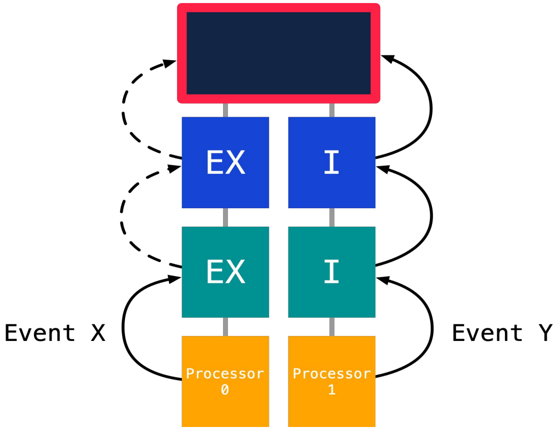
Define:
Event X: 0 modifies A, causing silent state transition at L1d
Event Y: 1 read A, request sent to and handled by L3

Issue:
Event X and *Y* detected by separate components in ZSim, there is no communication to exchange those detection information



Solution:

- ➔ Introduce new state EX, given to initial read request on migratory object
- ➔ If *Event X* happens first, EX -> M, send special message to L3, migratory object stays migratory
- ➔ If *Event Y* happens first, L3 sends downgrade invalidation, EX -> S, migratory object is no longer migratory



Now, L3 can detect both *Event X* and *Y*

⑧ Conclusion and Reflection

Migratory Sharing Optimization Tradeoffs

Benefits:
Small improvement only when accessing migratory objects

Costs:
If implemented in hardware, several additional bits required for each cache line

Extra States	**Costs in ZSim**
migratory flag	a boolean
last read-exclusive requester index	a 32-bit index number

Migratory Sharing Optimization may not be worth it in real cache architecture design!

Potential Improvements:

- ➔ Implement the alteration check to fix edge cases
- ➔ Run standard benchmarks like SPEC on a powerful testing machine
- ➔ Configure test environments to simulate existing architectures
- ➔ Add traffic contention simulation to ZSim, making improvement metrics more accurate