

Exploring Instruction Fusion Opportunities in General Purpose Processors

Sawan Singh, Arthur Perais, Alexandra Jimborean, Alberto Ros

2022 55th IEEE/ACM International Symposium on
Microarchitecture (MICRO)

Review by Dong Hyun Kim, Yeungnam University, Department
of Computer Science and Engineering

Background

- Instruction Cracking
 - Divide architectural instruction into multiple microarchitectural instruction (u-ops).
 - Some architectural instructions are too complicated to execute on hardware.
- Instruction Fusion
 - Fuse instruction to maximize resource utilization
 - Some architectural instructions are too simple, so could waste resource.

Problem of RISC Instructions

- Many RISC-V architectural instructions do not express enough work given modern hardware capabilities.
- Nevertheless, ISAs such as RISC-V want to keep architecture instructions simple.
- Thus, keep architectural instructions simple but perform fusions at microarchitectural level.

Fusion at μ -arch Level

- Microarchitectural fusion allows for more aggressive optimizations.
 - For instance, load pair in ARMv8 requires that the data be exactly **contiguous in memory**.
 - But, imagine that two non-contiguous memory accesses as long as they fall within a certain memory region.
- > This paper focuses on **microarchitectural fusion**.

Prior Fusion Techniques

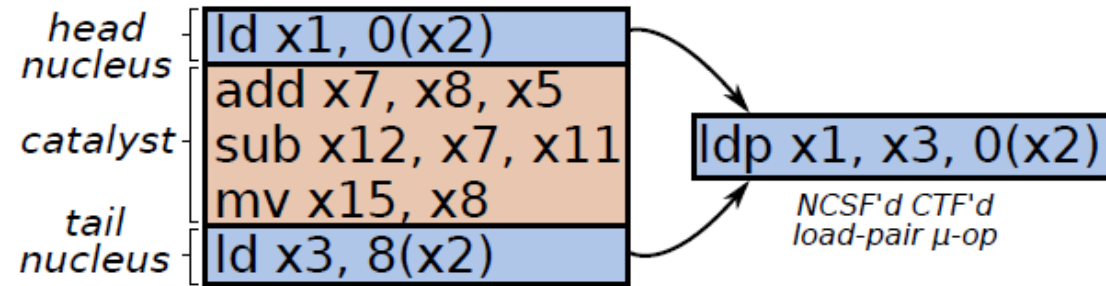
- Commercially available fusion proposals and implementations focus on consecutive and contiguous fusion.
- ConSecutive Fusion (CSF): operation of fusing two (or more) μ -ops that are **consecutive** execution stream of the program.
- ConTiguous Fusion (CTF): operation of fusing two (or more) memory μ -ops access **contiguous**.

This Work

- Non-ConSecutive Fusion (NCSF): operation of fusing two (or more) μ -ops that are **not consecutive** execution stream of the program.
- Non-ConTiguous Fusion (NCTF): operation of fusing two (or more) memory μ -ops that access **non-contiguous** memory bytes.

Think Instruction Fusion as Nuclear Fusion

- Head nucleus: the oldest μ -op (in program order).
- Tail nucleus: the youngest μ -op.
- Catalyst: μ -ops that are “in between” (in program order) the head nucleus and the tail nucleus



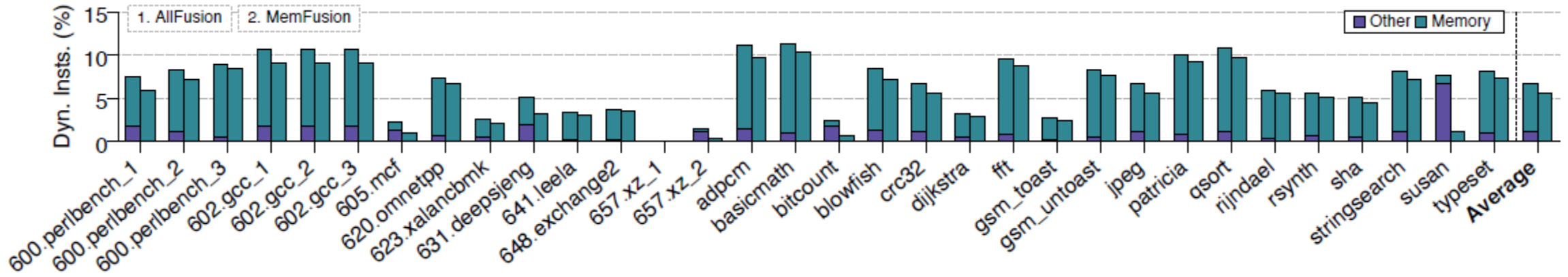
Catalyst facilitates the fusion process since there is no dependency to nucleii.

head and tail nucleii can be fused in a single load pair.

Motivation

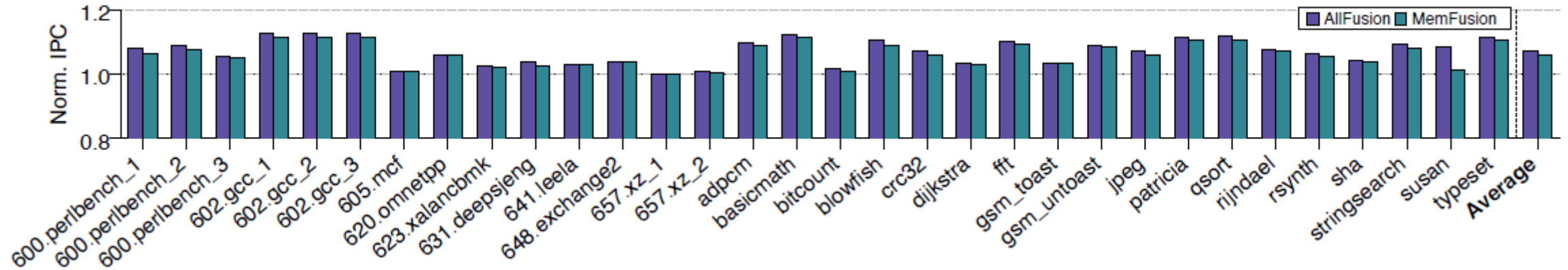
- Exploring additional microarchitectural fusion opportunities is motivated by several observations
 1. RISC-V ISA features very simple and therefore fuseable instructions.
 2. Architectural fusion is limited in what it can fuse.
 3. Current implemented microarchitectural fusions are too conservative.

Observation 1



- Figure shows the percentage of fused pairs divided into Memory and Others.
- On average 5.6% of the dynamic μ -ops belong to the Memory category while 1.1% belong to Others

Observation 2



- Figure shows the corresponding IPC normalized to a baseline without fusion.
- Differences between fusing all μ -ops and just fusing memory μ -ops are minimal.

Observation 1 and 2

- This work focuses on memory μ -ops and that's enough.
- Memory μ -ops fusion can also reduce IQ, ROB, LQ or SQ requirements.
 - Why? Memory instructions spend a lot of resources.
 - Thus, more performance benefit.

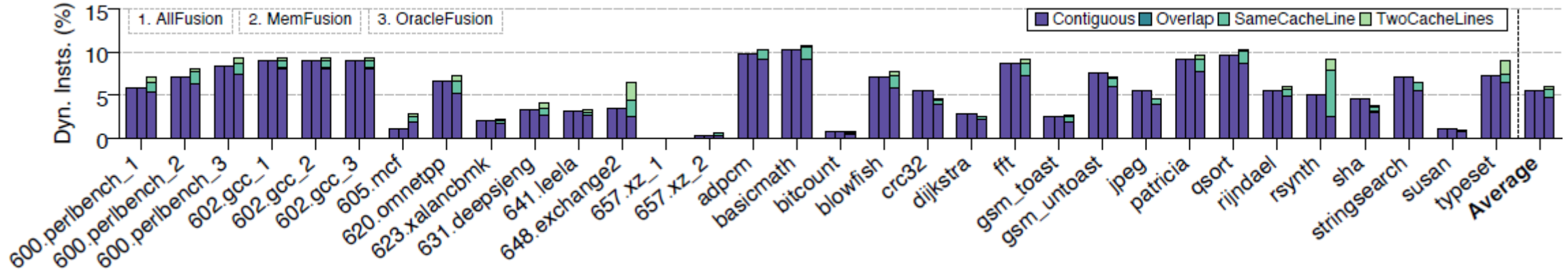
Observation 3

- Pairing at architectural level is limited.
 - Requires both accesses to be exactly in contiguous in memory.
 - Requires both accesses to have the same size.
- Hence, architectural fusion **misses** on accesses that have
 - Overlapping bytes
 - Asymmetric
 - Close accesses but do not have overlapping bytes.
- In summary, architectural fusion wants μ -ops to be well-aligned.

Observation 3

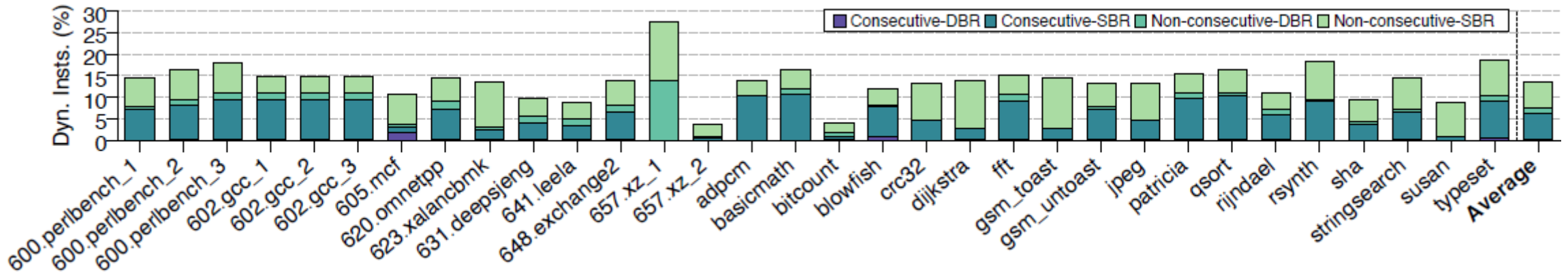
- μ -arch level fusion is different. At the μ -arch level, cache can access larger chunk of memory at once.
- Memory μ -ops may be fused if they access data that falls within a cache access granularity region.
- This requirement is satisfied by accesses with
 - Overlapping bytes
 - Asymmetric
 - Close accesses but do not have overlapping bytes.

Observation 3



- Figure shows the contribution of different consecutive memory fusion categories
 - Contiguous
 - Overlapping
 - Same cacheline
 - Two contiguous cachelines
- Architectural fusion would still leave potential on the table since 1% additional memory μ -ops could be fused.

Observation 4: Consecutivity

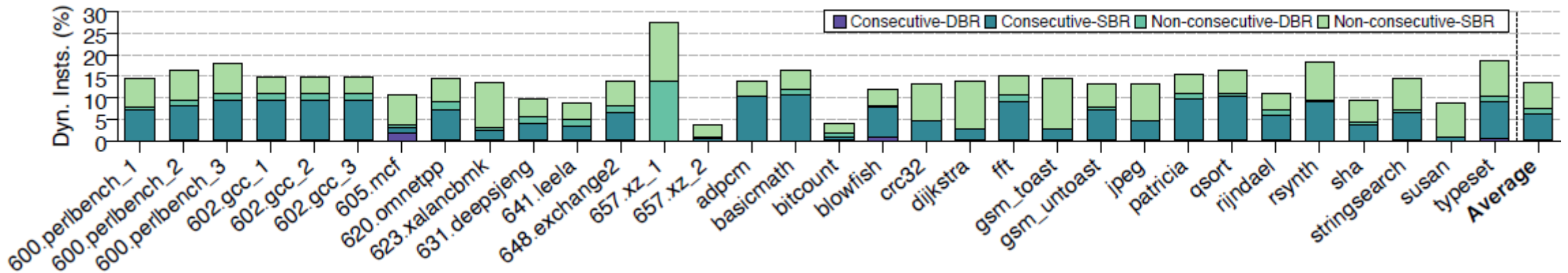


- Usually, fusion technique is fuse two or more μ -ops that are consecutive in the instruction stream.
- But if this constraint were to be relaxed, non-negligible amount of additional μ -ops could be fused.
- Figure shows additional fusion potential by non-consecutive fusion for memory μ -ops.
- The number of asymmetric access is quite high, 12.1% of NCSF pairs. If fusion of those μ -ops are possible, we can get more performance benefit.

Observation 4: Static Information Only

- In the context of RISC-V, memory μ -ops may be fused if
 - 1) They are all loads or all stores.
 - 2) They share the same physical base register.
 - 3) They access data that resides within a cacheline size region.
- Problems
 - Not all fuseable μ -op pairs meeting the condition 3).
 - There exist pairs of both consecutive and non-consecutive memory μ -ops that has **different base register (DBR)** but access the same cacheline sized memory region.
- It is not easy or even possible to identify fuseability **using only static information.**

Observation 4: Static Information Only



- Figure shows fusion potential brought by fusing instructions that do not share the same physical base register: 1.5% of dynamic μ -ops.
- If we can fuse those instructions, we can also get performance benefit.

Helios Microarchitecture

- Helios supports Non-ConSecutive (NCS) and Non-ConTiguous (NCT) fusion of memory μ -ops that might use DBR.
- Three key mechanisms
 1. To perform NCSF, each μ -op has to find all older μ -ops. This is too exhaustive, so Helios uses **predictive approach**.
 2. Fusion may suffer from dependencies between nuclei and their catalyst. Helios identifies dependencies and addresses them if it is possible.
 3. Helios handles incorrectly fused instructions as well as other mispredictions within a catalyst.

Unified Predictor

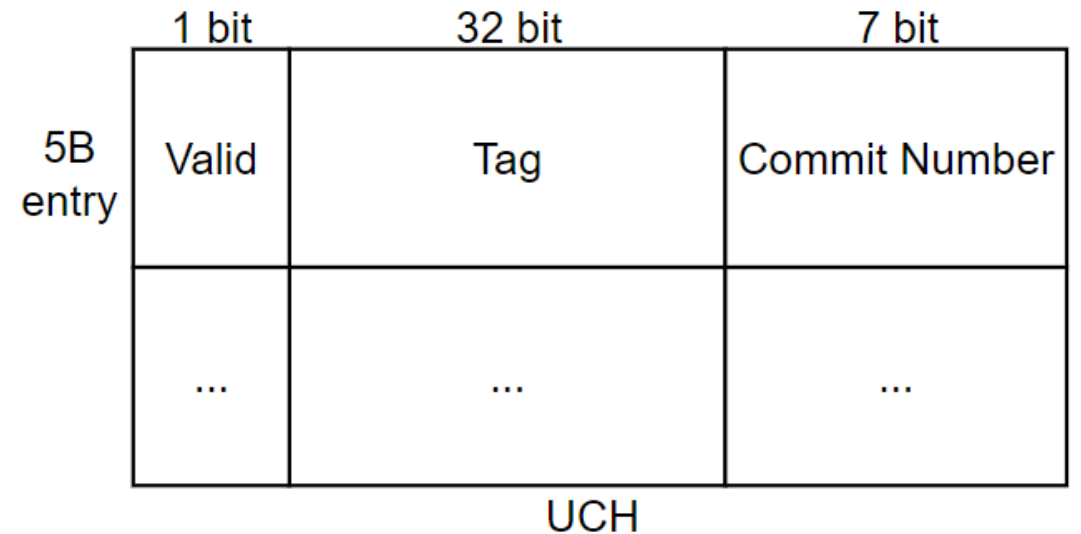
- Helios implements a unified hardware predictor for NCSF, NCTF, and DBR.
- This provides the distance to the head nucleus to fuse with.
- Predictor consists of 2 structures.
 1. UCH: lives in Commit stage, used to find potential fusion pairs to train FP.
 2. FP: placed in frontend and predicts which μ -ops should be speculatively fused.

Unfused Committed History

- UCH is organized as a cache.
- UCH keeps the cache lines accessed by the last committed memory μ -ops eligible for fusion.

UCH Structure

- 1-bit, valid bit
 - 32-bit, tag (partial cache line address)
 - 7-bit, commit number (CN)
- + LRU or another replacement policy
- > Total : 5-bytes entry

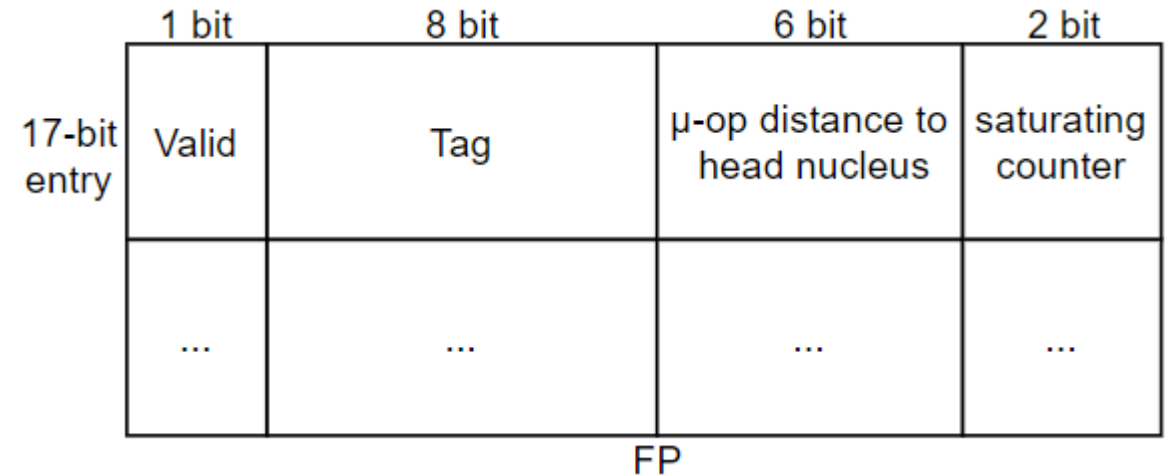


Fusion Predictor

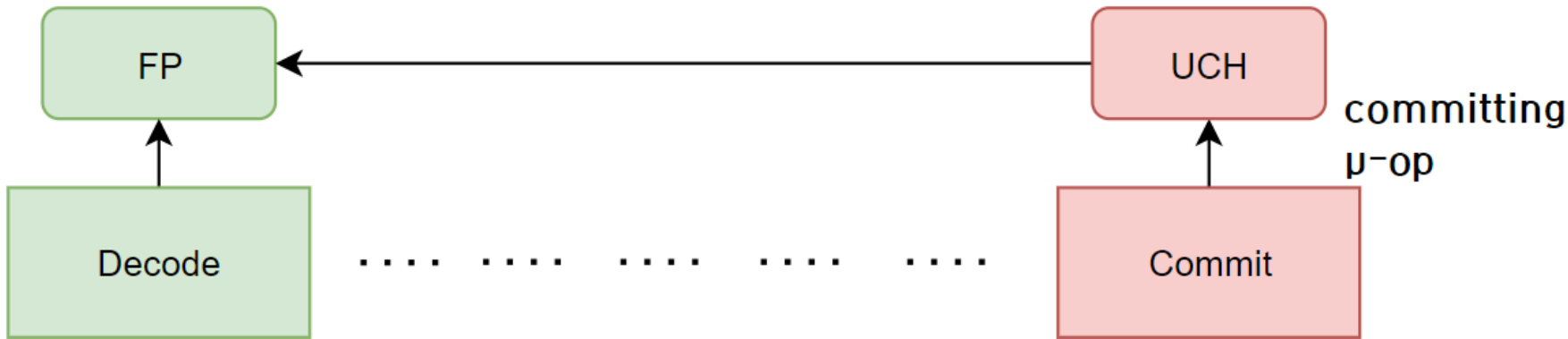
- FP contains information about potential tail nuclei.
- 8-bit, tag
- 6-bit, μ -op distance to head nucleus
- 2-bit, saturating counter
- 1-bit, valid bit

+ LRU or another replacement policy

-> Total : 17-bit entry



Update UCH for Fusion



- Retiring μ -op accesses to UCH at the Commit stage
 - Tag match
 - Compute distance by subtracting CN
 - Invalidate UCH entry
 - Allocate FP entry with distance
 - Tag miss
 - Insert μ -op to UCH
 - Previous match evicted

Update FP for Fusion



- μ -op accesses to at the Decode stage
 - Tag match
 - Distance match: saturating counter +1
 - Distance miss: insert new distance & saturating counter reset
 - Tag miss
 - Selected for eviction

Fusion Process

- Fusion is attempted in the Allocation Queue only when the **three conditions are met**:
 - 1) Saturating counter has the **maximum value**
 - 2) Two μ -ops form a **valid fusion idiom**
 - Both μ -ops are both loads or both stores.
 - The head nucleus is not already a fused μ -op.
 - 3) Head nucleus still **resides in the Allocation Queue** or is in the **same Decode Group** as the tail nucleus.
- Head nucleus is finally replaced by the fused μ -op.

Problems about Fusion

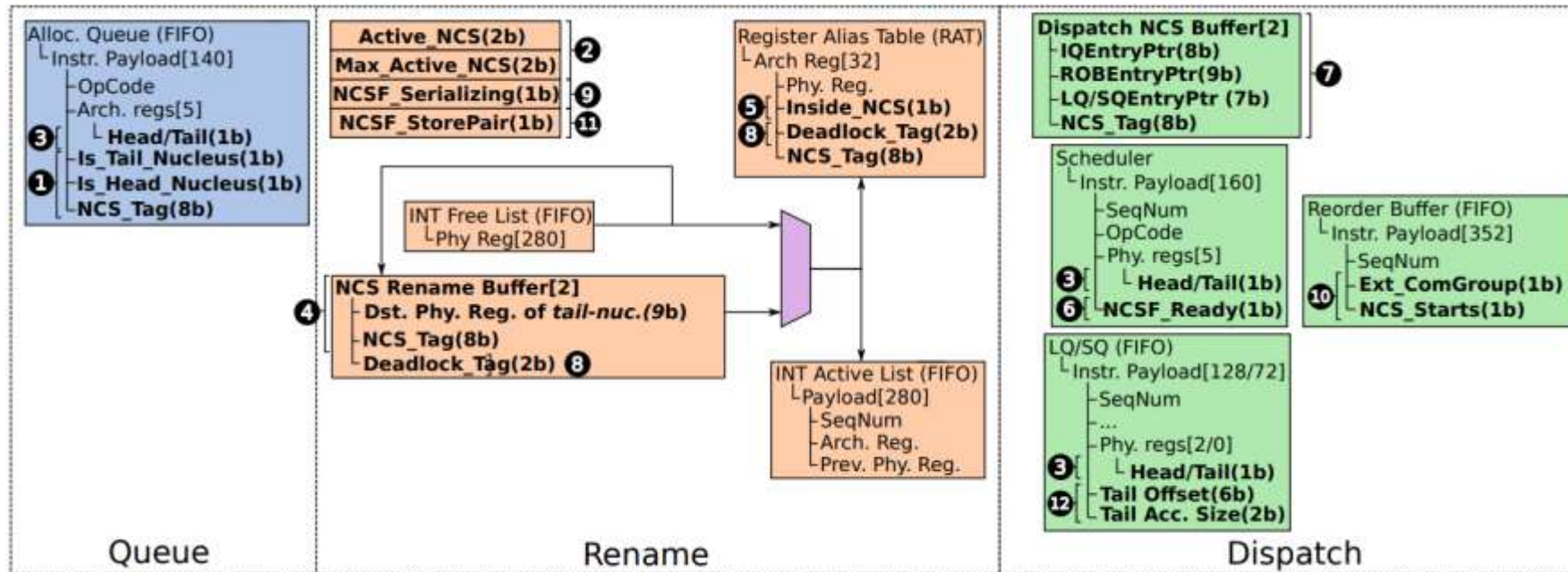
- Register dependency
 - Write after Read
 - Read after Write
- Dependency-based Deadlocks
- Preserving Program Correctness
- Memory Consistency

=> Helios implements hardware to correct these issues.

For Correct Fusion,

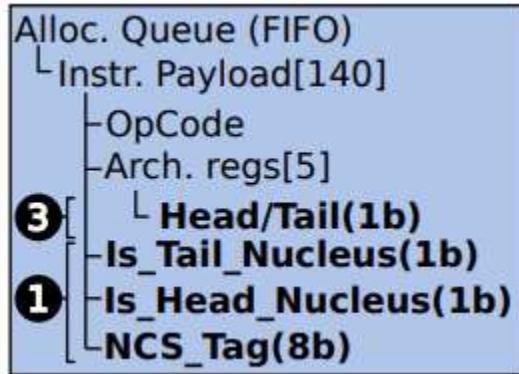
- NCSF does not remove tail nucleus from AQ since those instructions might not a valid fusion pair.
- A fused μ -op has to be validated for correct fusion.
- Validated fused μ -op
 - Possess its correct src. phy. reg. identifiers (**dependency problem**).
 - Not produce a deadlock (**deadlock by data dependency**).
 - Not have store in its catalyst if it's a store pair μ -op (**memory consistency problem**)
- Pending fused μ -op
 - Fused μ -op that is not yet validated.

Pipeline Structure Changes



- Helios added a lot of hardwares for correct fusion (1 ~ 12).
- These are going to solve the issues said before.

Allocation Queue



- *Is_Head_Nucleus* & *Is_Tail_Nucleus*
 - Track which μ -op is head or tail.
- *NCS_Tag*
 - Pointing the other nucleus μ -op in the AQ.
- *Head/Tail*
 - Inform whether that physical register belongs to head or tail.

- Head nucleus carry their own AQ entry number until they are dispatched to the ROB/IQ/LQ/SQ.
- Tail nucleus carry their respective *NCS_Tag* until they are dispatched

Rename

Active_NCS(2b)	②
Max_Active_NCS(2b)	
NCSF_Serializing(1b)	⑨
NCSF_StorePair(1b)	⑪

- *Max_Active_NCS*
 - Tracks the number of head nuclei in the current Rename.
 - *Active_NCS*
 - Determines Rename is processing the NCSF group or not.
-
- Both counters are
 - incremented when a head nucleus enters Rename.
 - reset on pipeline flush.
 - When tail leaves Rename, *Active_NCS* decremented.
 - If *Active_NCS* == 0, Rename is not processing NCSF. Thus, *Max_Active_NCS* is also reset.

Problem: Register Dependency

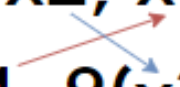
- Non-consecutive fusion is problematic when there exists register dependencies:
 - Between the catalyst and tail nucleus
 - Between the head and tail nuclei

Problem: Register Dependency

[1] ld x1, 0(x2)

[2] add x2, x4, 1

[3] ld x4, 8(x2)



- [1] and [3] are trying to fuse.
- [3] has Write-after-Read dependency with 2 through x4.
- [3] has also Read-after-Write dependency through x2.

Write-after-Read (x4)

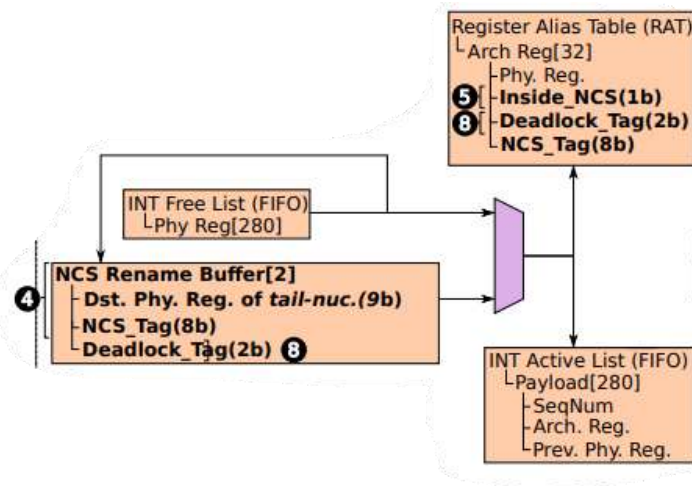
[1] ld x1, 0(x2) + ld x4, 8(x2)

[2] add x2, x4, 1

[3] ld x4, 8(x2)

- [1] and [3] are fused and renamed together.
- [2] will be able to see the version of x4 produced by [3] in RAT and use it as a source.
- Therefore, Helios needs to prevent [2] from observing new name of x4.

Write-after-Read (x4): Solution



- *NCS Rename Buffer*
 - Stores *dst. phy. reg. identifier*.
- *Active List*
 - Contains the in-flight register mappings.

- Helios prevents NCSF'd μ -op from updating the RAT for destination register of tail nucleus.
- Those physical registers are stored in *NCS Rename Buffer*.
- When tail nucleus goes Rename, destination renaming is performed by reading physical register identifiers from buffer to update RAT.

Read-after-Write (x2)

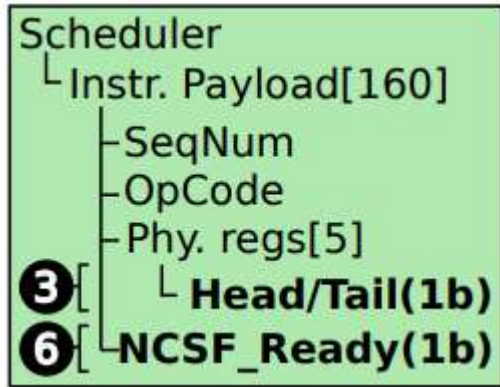
[1] ld x1, 0(x2) + ld x4, 8(x2)

[2] add x2, x4, 1

[3] ld x4, 8(x2)

- Fused μ -op will incorrectly rename the source operand of [3] because of RaW dependency.
- Helios prevents pending NCSF'd μ -op from issuing until RaW dependency is evaluated.
- This allows the correct source register name to be provided to the IQ entry of the NCSF'd μ -op as it becomes validated.

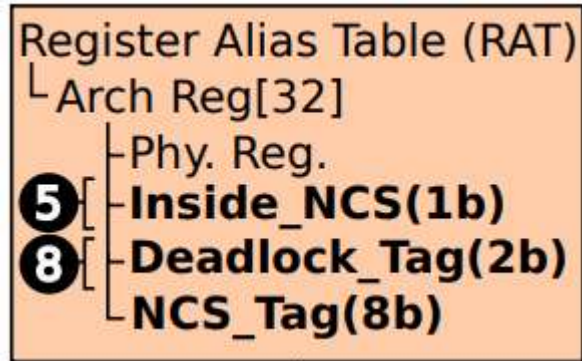
Read-after-Write (x2): Solution



- *NCS_Ready*
 - Denotes whether an NCSF'd μ -op is pending or validated.

- All μ -ops in flight between Rename and the IQ carry this bit.
- NCSF'd μ -op leaves Rename, its *NCS_Ready* is reset.
- But *NCS_Ready* of all other μ -ops (not NCSF'd μ -op) make the bit set.

Read-after-Write (x2): Solution



- *Inside_NCS*
 - Indicates that Rename is currently processing NCSF'd μ -op or not.

- *Inside_NCS* is set in the RAT entry,
 - when a μ -op renames its dest. reg. and
 - Rename is currently processing an NCSF's'd μ -op (*Active_NCS* \neq 0).
- When tail nucleus enters Rename, it looks its src. regs. in RAT.
 - If one of src has *Inside_NCS* bit set, there is RaW dependency between tail nucleus and catalyst.

Read-after-Write (x2): Solution

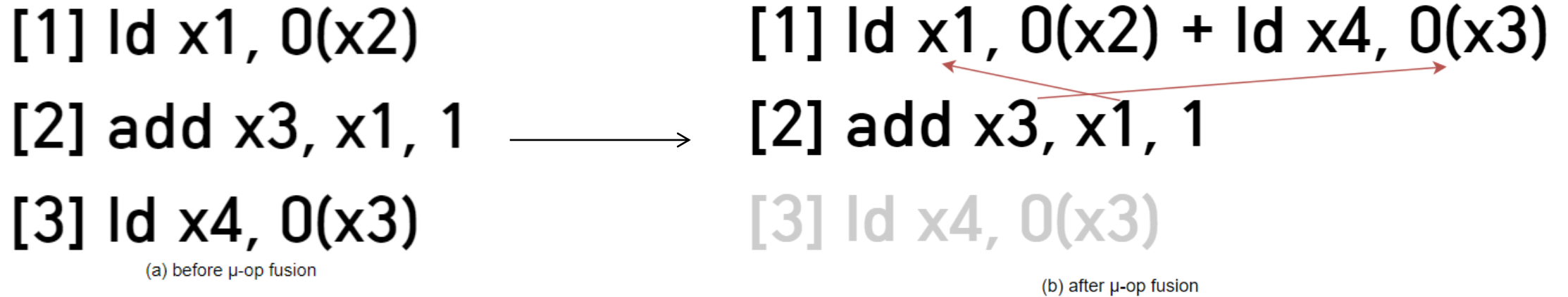
- If *NCSF_Ready* bit is not set, NCSF'd μ -op cannot issue in the IQ.
- Thus, corresponding tail nucleus flows to Dispatch,
 - unfuse the NCSF'd μ -op or
 - correct the source register name and set the *NCSF_Ready* bit.
- It uses a buffer in Dispatch stage to write the IQ entry.

Read-after-Write (x2): Solution



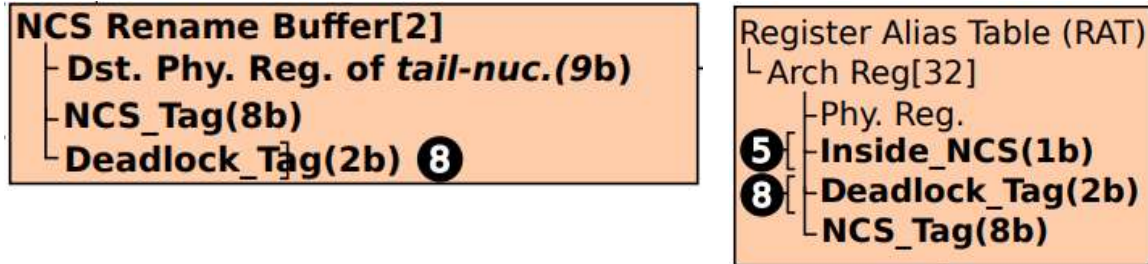
- *Dispatch NCS Buffer*
 - A buffer to correct pending NCSF'd μ -op in the IQ.
 - Stores to pointers to ROB and LQ/SQ for unfusing a pending NCSF'd μ -op.
- Entries are tagged with AQ entry number of head nucleus.
- It is allocated when a pending NCSF'd μ -op dispatches.
- Tail nucleus validates head nucleus IQ entry -> entries are reclaimed.

Problem: Deadlock



- Tail nucleus might depends on the head nucleus directly or indirectly.
- In the figure above, x3 cannot be produced until x1 is produced by load pair μ -op, which cannot happen until x3 is produced by 2.
- It means that the execution is deadlocked.

Deadlock: Solution

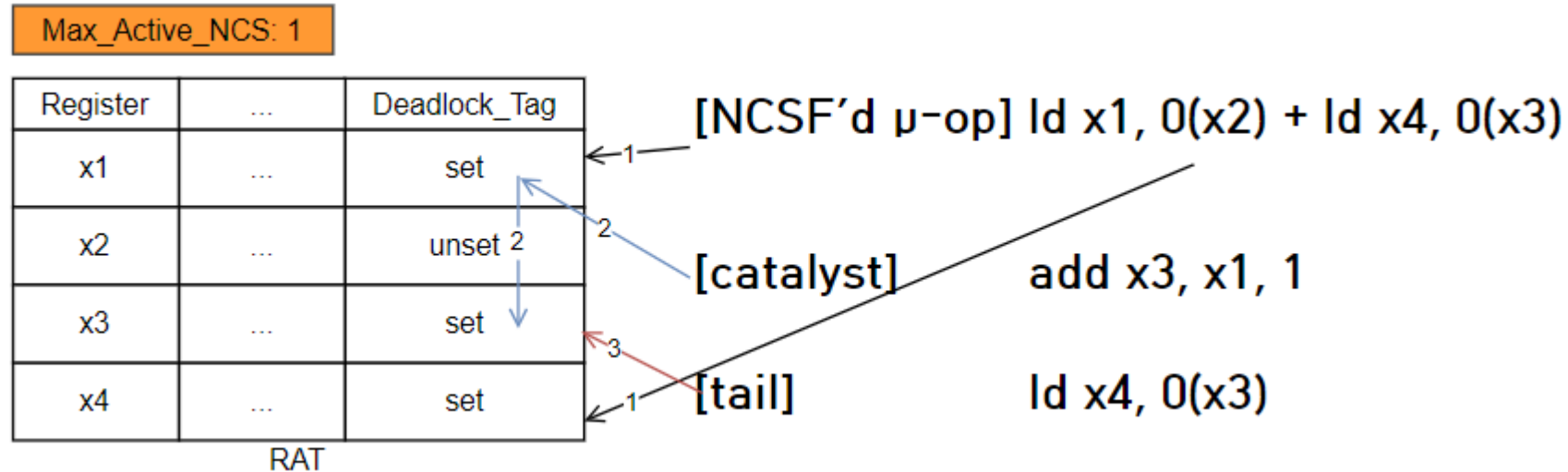


- Helios needs to detect whether tail nucleus directly or indirectly depends on its head nucleus or not.
- Therefore, *Deadlock_Tag* is added and it denotes the current value of *Max_Active_NCS* (current head nucleus).

Deadlock: Solution

1. When NCSF'd μ -op is renamed, it writes *Deadlock_Tag* field in RAT entries of its destination register.
2. *Deadlock_Tag* is propagated from source(s) to destination as younger μ -ops are renamed.
3. If any of the source registers of a tail nucleus has a *Deadlock_Tag* with the bit set, there is dependency-based deadlock.

Deadlock: Solution

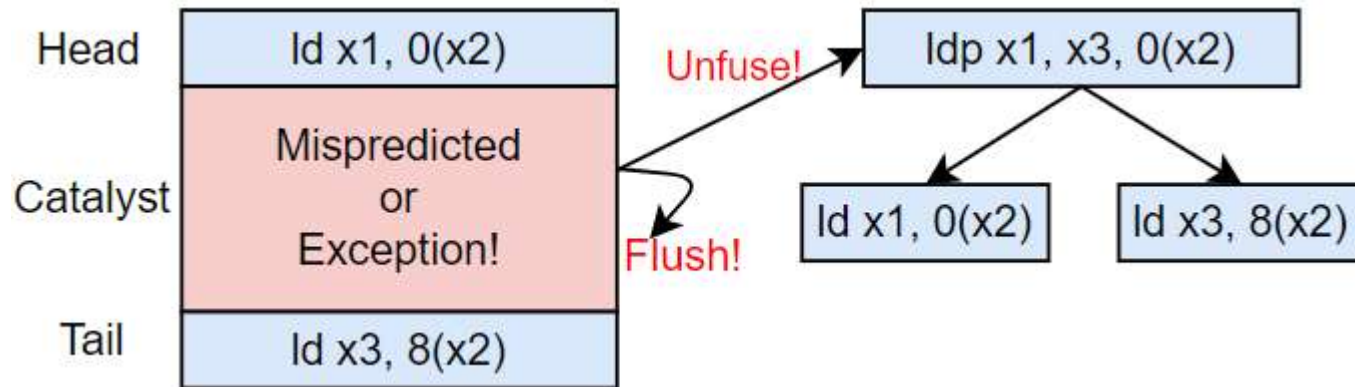


1. NCSF'd μ -op sets its Deadlock_Tag of destination register, x1 and x4.
2. Younger μ -op (catalyst) sets Deadlock_Tag from source to destination register (x1 \rightarrow x3).
3. Next instruction (tail) see its Deadlock_Tag of destination register (x3) is set and then realizes dependency-based deadlock occurred.

Deadlock: Recovery

- If there is deadlock, NCSF should not take place.
- Recovery is done by letting the tail nucleus flow through Dispatch and unfuse NCSF'd μ -op.

Preserving Program Correctness



- If a misprediction or an exception is detected in the catalyst, then the pipeline flush or unfuse is needed.
- Thus, **unfuse** NCSF'd μ -op whose catalyst contains the mispredicted instruction and **flush** from the mispredicted instruction.

Memory Consistency

- Helios is speculative, so it cannot guarantee that:
 1. There is no store μ -op in catalyst of a store pair NCSF'd μ -op.
 2. If there is such a store μ -op, it cannot guarantee that this does not overlap with tail nucleus.
- As a result, NCSF'd μ -op could violate memory consistency.

Memory Consistency: Solution

Active_NCS(2b)
Max_Active_NCS(2b)
NCSF_Serializing(1b)
NCSF_StorePair(1b)

- *NCSF_StorePair*
 - Denotes that there is store μ -op in the catalyst
- Helios prevents NCS store pair fusion when finding a store μ -op in the catalyst.
- It adds *NCSF_StorePair* bit to Rename.
- This bit is set when any store μ -op other than the first head nucleus of the NCSF nest is renamed.
- Any store tail nucleus seeing this bit set will unfuse NCSF'd store pair μ -op waiting in the IQ.

Different Base Register Problem

- Non-negligible amount of pairs of loads access data that fit within a cacheline through **different base register**.
- Fortunately, DBR fusion is captured by the predictive scheme employed by Helios.

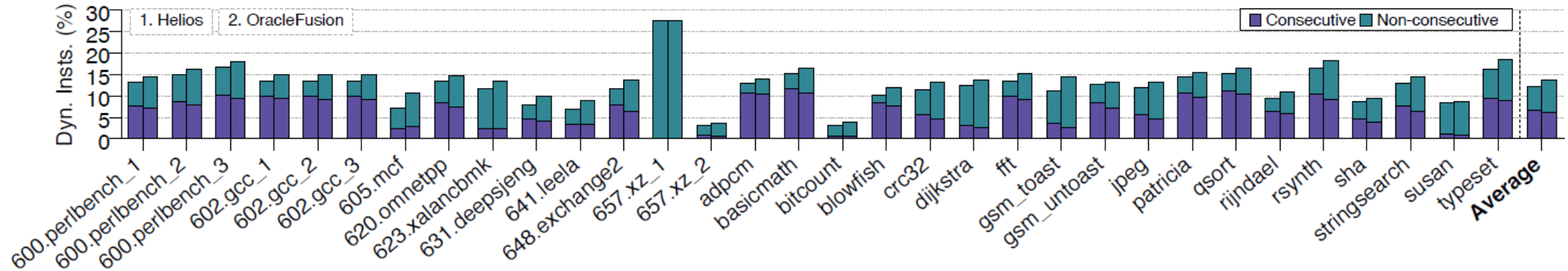
Storage Requirement

- Supporting NCSF in the different stages of the pipeline requires a total of 4.77Kbits, or 0.60KB.
- Adding the Fusion predictor yields a grand total of 76.77Kbits (9.60KB).

Performance Evaluation

- Simulation infrastructure
 - RISC-V Spike simulator
 - 8-stage out-of-order processor model
- Benchmark suites
 - SPEC CPU 2017
 - MiBench
- Configurations
 - RISCVFusion: fuses μ -ops non memory pairs
 - CSF-SBR: fuses only consecutive memory instruction with same base register
 - RISCVFusion++: fuses all instructions
 - OracleFusion: fuses all eligible memory pairs (upper bound).

Result: Fused Pairs



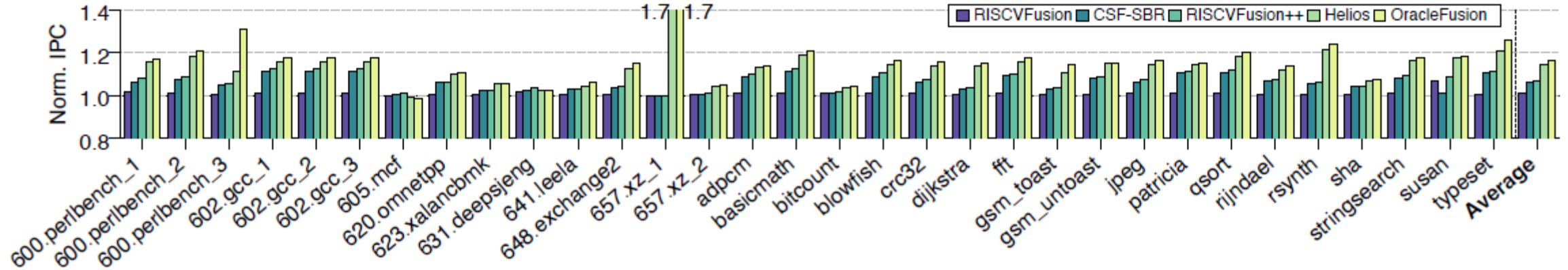
- Total number of fused pairs in Helios is close to upper limit of OracleFusion.
- Helios delivers 6.7% CSF and 5.5% NCSF pairs.

Result: Fusion Prediction

- Coverage
 - Helios predictor fuses 68.2% of the memory μ -ops.
 - Helios fuses 12.2% of dynamic μ -ops, approaching to 13.6% of OracleFusion.
- Accuracy
 - Helios provides high accuracy, 99.7% on average.
 - Average mispredictions per kilo instructions (MPKI): 0.1416

Benchmark	Coverage	Accuracy	MPKI
600.perlbench_s_1	76.56	99.96	0.0183
600.perlbench_s_2	75.30	99.90	0.0592
600.perlbench_s_3	71.99	99.97	0.0170
602.gcc_s_1	63.32	99.51	0.1776
602.gcc_s_2	62.73	99.52	0.1709
602.gcc_s_3	63.43	99.52	0.1722
605.mcf_s	62.24	98.60	0.8350
620.omnetpp_s	67.86	99.40	0.3018
623.xalancbmk_s	82.94	99.97	0.0269
631.deepsjeng_s	58.82	98.68	0.4602
641.leela_s	62.15	97.74	0.8172
648.exchange2_s	50.04	99.56	0.1595
657.xz_s_1	99.99	99.99	0.0000
657.xz_s_2	73.90	99.90	0.0207
adpcm	58.90	99.99	0.0005
basicmath	61.68	99.99	0.0013
bitcount	74.54	99.56	0.1083
blowfish	48.00	99.89	0.0254
crc32	66.49	99.99	0.0046
dijkstra	85.64	99.99	0.0058
fft	57.05	99.93	0.0210
gsm_toast	65.34	99.51	0.3765
gsm_untoast	67.89	99.99	0.0010
jpeg	72.74	99.99	0.0061
patricia	62.80	99.99	0.0036
qsort	66.97	99.77	0.0965
rijndael	62.22	99.85	0.0471
rsynth	64.23	99.99	0.0047
sha	69.22	99.99	0.0023
stringsearch	67.76	99.97	0.0115
susan	91.36	99.99	0.0010
typeset	69.26	99.17	0.5758
Average	68.23	99.68	0.1416

Result: IPC Improvement



- On average, Helios provide an IPC improvement of 14.2% over a baseline.
- Also, Helios achieves most of the benefits of OracleFusion which stands at 16.3% improvement.

Summary

- Instruction fusion might improve resource utilization.
- RISC-V wants to keep their architectural instructions simple thus, Helios fuses instructions at microarchitectural level.
- Fusing instruction at microarchitectural level has less constraints,
 - Don't need to be consecutive in program order
 - Don't need to be contiguous in memory
 - Helios fuses memory uops that is even non-consecutive and non-contiguous.
- Helios uses predictive approach but handles a lot of issues such as dependency, deadlock.
- Performance
 - 68.2% coverage
 - 99.7% accuracy
 - 14.2% IPC improvement