

REGIONAL OUT-OF-ORDER WRITES IN TOTAL STORE ORDER

Sawan Singh

Alexandra Jimborean

Alberto Ros



UNIVERSIDAD DE
MURCIA



Department Computer Engineering and Technology
University of Murcia

October 6, 2020

- This talk is about a new **store buffer (SB)** design
 - Hide **store-miss** latency

- This talk is about a new **store buffer (SB)** design
 - Hide **store-miss** latency
- We relax **TSO** by performing selective stores **Out-of-Order (OoO)**
 - And study it's affect on **performance improvemnt** and **energy consumption**

- This talk is about a new **store buffer (SB)** design
 - Hide **store-miss** latency
- We relax **TSO** by performing selective stores **Out-of-Order (OoO)**
 - And study it's affect on **performance improvemnt** and **energy consumption**
- The result is
 - **ROOW**, **relaxed** yet as strong as **TSO**
 - **16 entries** SB with **5.64%** performance improvement compared to **TSO** with **56 entries** SB
 - Almost **0** hardware overhead (**$N+1$ bits**, **N : Entries in SB**)

OUTLINE

1 MOTIVATION

2 ROOW

3 RESULTS

4 CONCLUSION

OUTLINE

1 MOTIVATION

2 ROOW

3 RESULTS

4 CONCLUSION

MOTIVATION

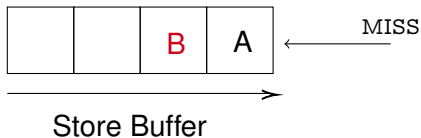
```
store A
store B
store C
store D
store E
.
.
.
```



TSO

MOTIVATION

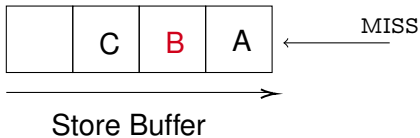
```
store A
store B
store C
store D
store E
.
.
.
```



TSO

MOTIVATION

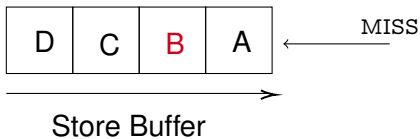
```
store A
store B
store C
store D
store E
.
.
.
```



TSO

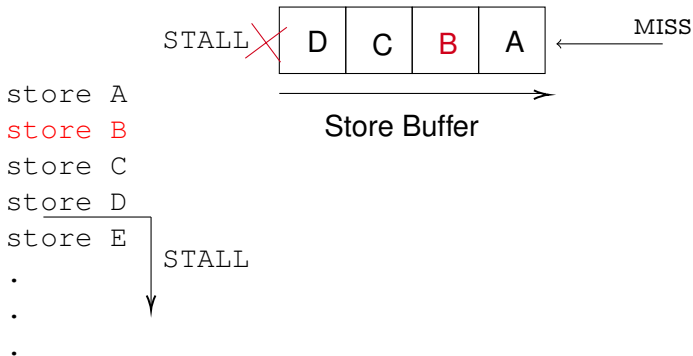
MOTIVATION

```
store A
store B
store C
store D
store E
.
.
.
```



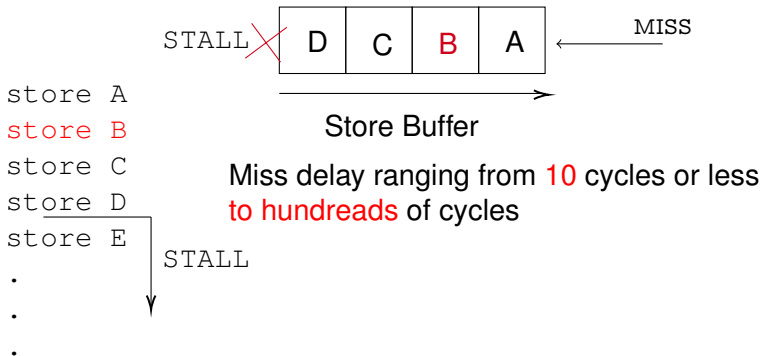
TSO

MOTIVATION



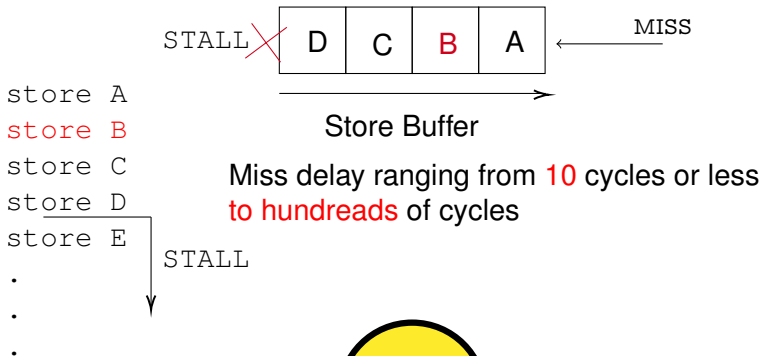
TSO

MOTIVATION



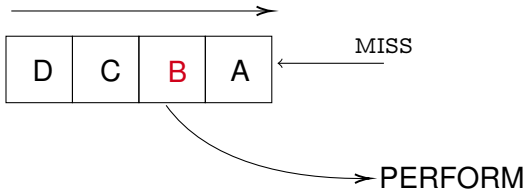
TSO

MOTIVATION



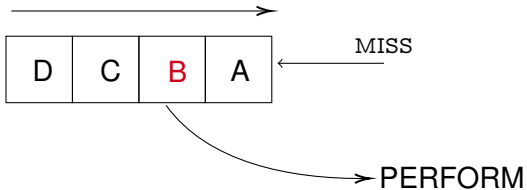
MOTIVATION

```
store A
store B
store C
store D
store E
.
.
.
```

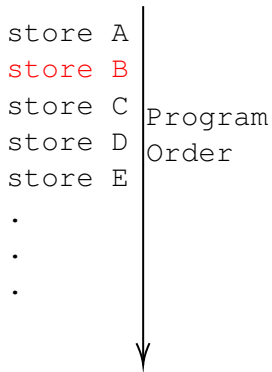


MOTIVATION

```
store A
store B
store C
store D
store E
.
.
.
```



GUARANTEE NEEDED




GUARANTEE NEEDED

```

store A
store B
store C
store D
store E
.
.
.

```

Program
Order




```

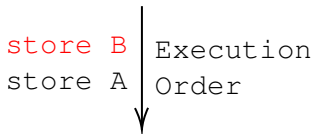
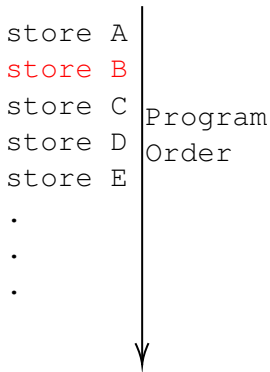
store B
store A

```

Execution
Order

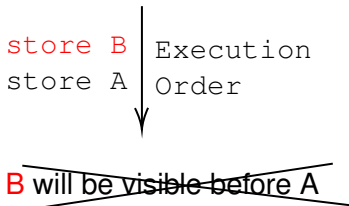
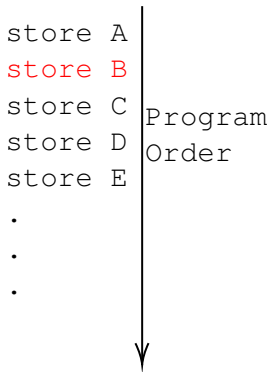


GUARANTEE NEEDED

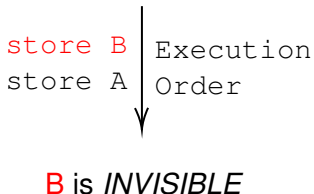
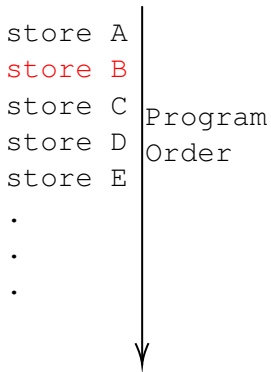


B will be visible before A

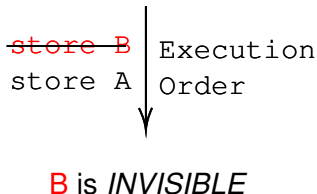
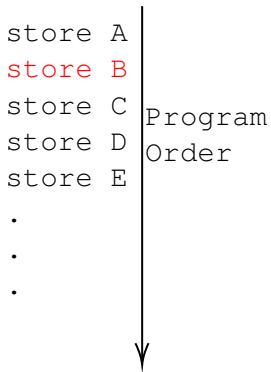
GUARANTEE NEEDED



GUARANTEE NEEDED



GUARANTEE NEEDED



IDENTIFYING STORES

No concurrent accesses by other thread/core

No **concurrent accesses** by other **thread/core**



Access becomes **INVISIBLE** until the next **synchronization operation**

IDENTIFYING STORES

No **concurrent accesses** by other **thread/core**

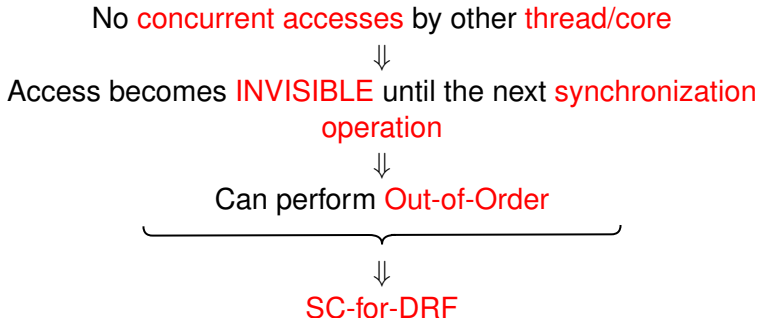


Access becomes **INVISIBLE** until the next **synchronization operation**



Can perform **Out-of-Order**

IDENTIFYING STORES



No **concurrent accesses** by other **thread/core**



Access becomes **INVISIBLE** until the next **synchronization operation**



Can perform **Out-of-Order**



SC-for-DRF

(Supported by most of the programming languages such as Java, C, C++)

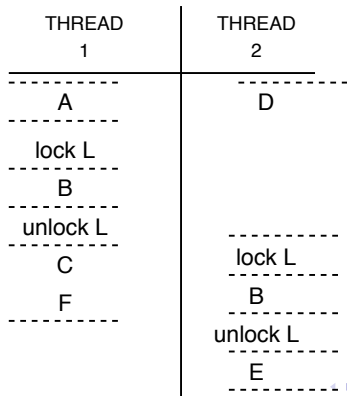
- Sequential consistency guarantee only for **DRF programs**

SC-FOR-DRF

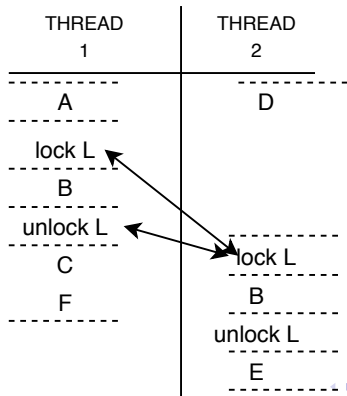
- Sequential consistency guarantee only for **DRF programs**
- **SC-for-DRF** requires **racy accesses** to be confined within **synchronization operations**

SC-FOR-DRF

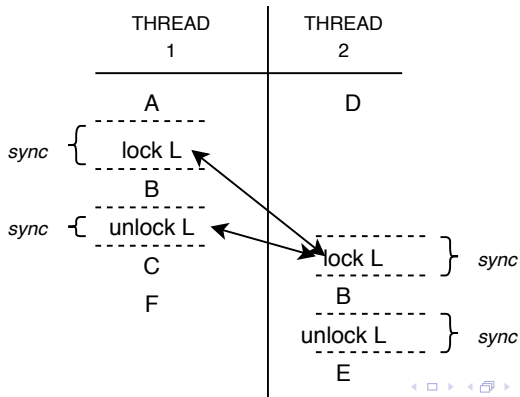
- Sequential consistency guarantee only for **DRF programs**
 - **SC-for-DRF** requires **racy accesses** to be confine within **synchronization operations**
- ⇒ If program does not have **data races**, the **compiler** will insert all the **necessary fences** to preserve the **SC**



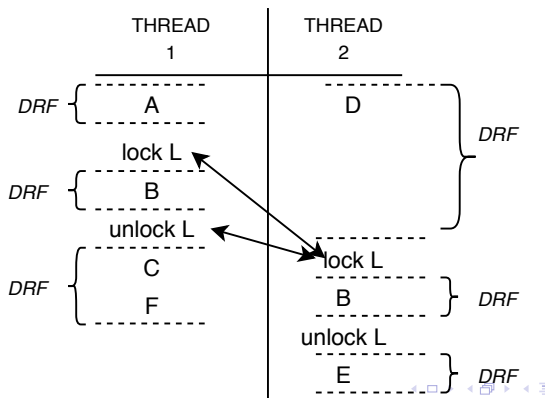
SC-FOR-DRF



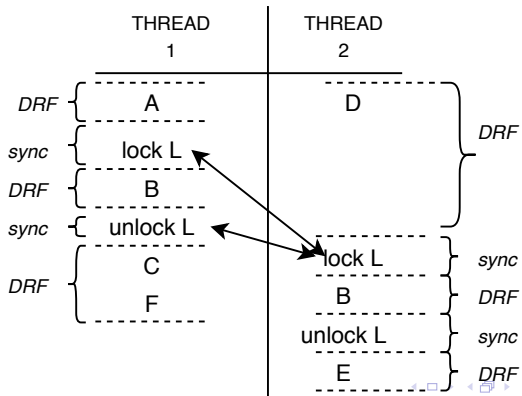
SC-FOR-DRF



SC-FOR-DRF



SC-FOR-DRF



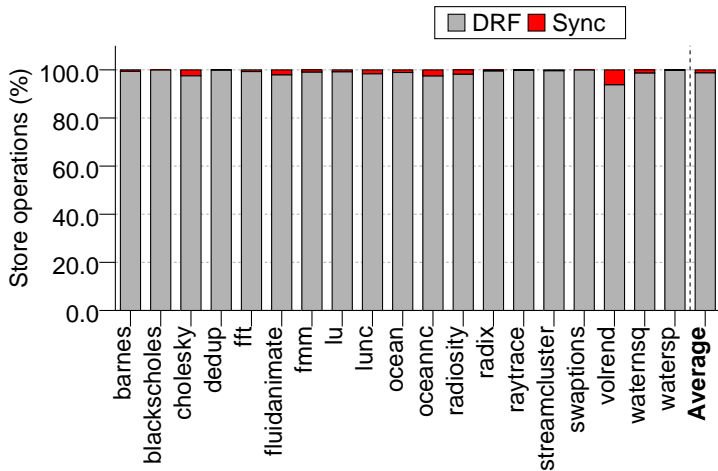
IMPORTANT POINTS

- Stores in **DRF region** can perform **OoO** as they are **INVISIBLE** until the end of the region

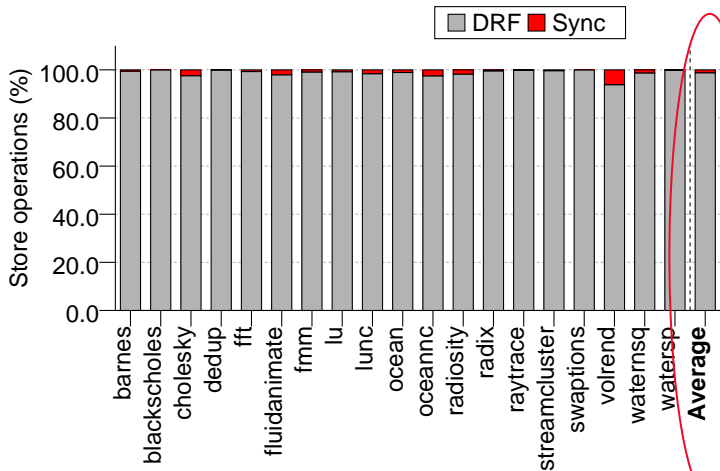
IMPORTANT POINTS

- Stores in **DRF region** can perform **OoO** as they are **INVISIBLE** until the end of the region
- All stores belonging to **sync region** will follow **TSO**

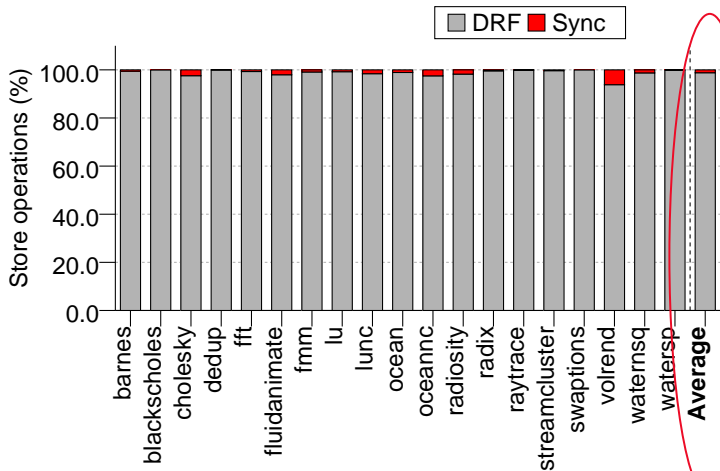
POTENTIAL



POTENTIAL



POTENTIAL



98.8%
stores
are
DRF

OUTLINE

1 MOTIVATION

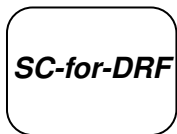
2 ROOW

3 RESULTS

4 CONCLUSION

SC-for-DRF

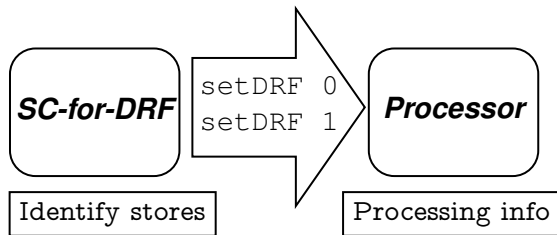
Identify stores

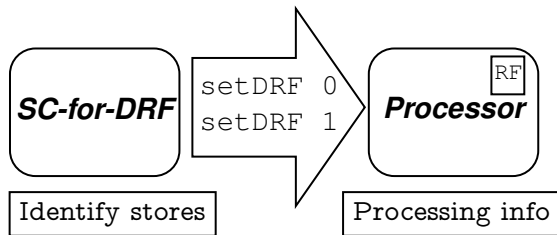


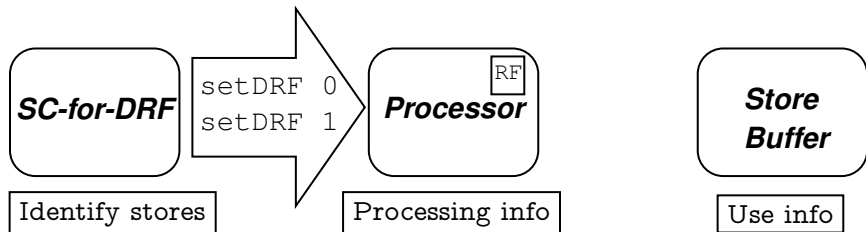
Identify stores

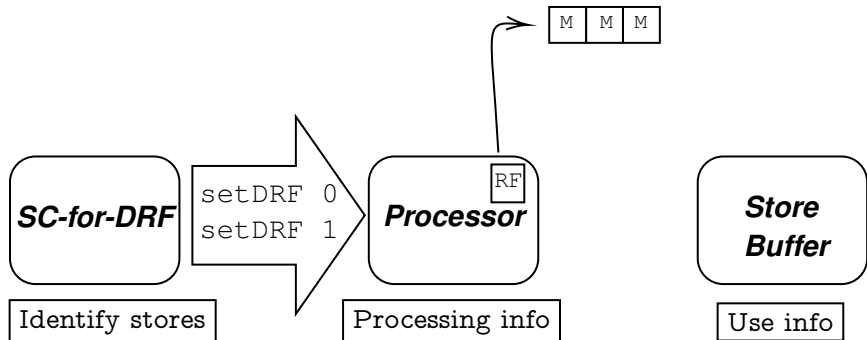


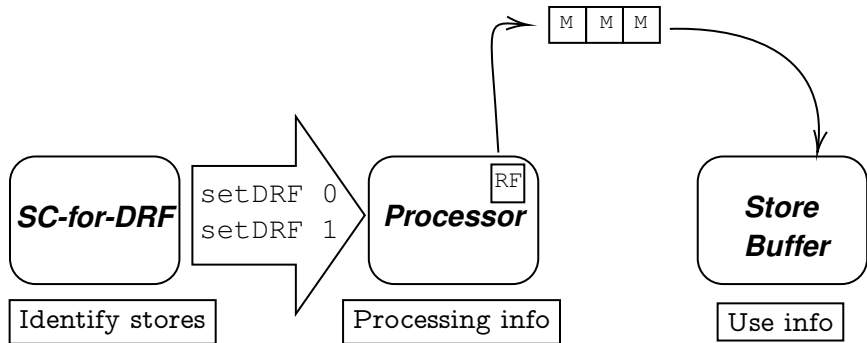
Processing info











EXECUTION OVERVIEW

RoB commit

| |
|---------------------------|
| <i>setDRF</i> <i>0</i> |
|---------------------------|

| |
|----------------------------|
| <i>Setting sync region</i> |
|----------------------------|

EXECUTION OVERVIEW

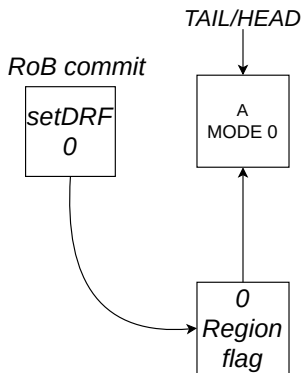
RoB commit

setDRF
0

0
Region
flag

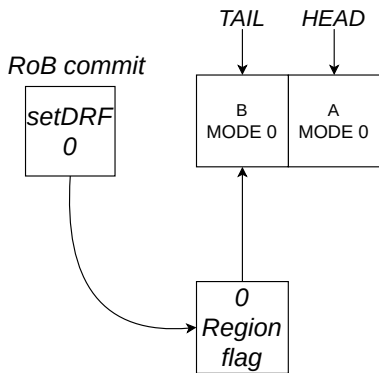
Reset region flag

EXECUTION OVERVIEW



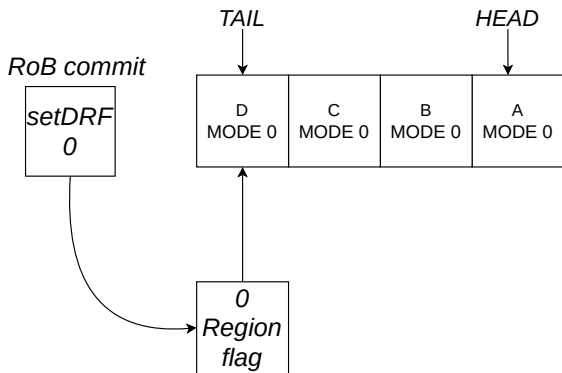
Store copy region flag to their mode bit

EXECUTION OVERVIEW



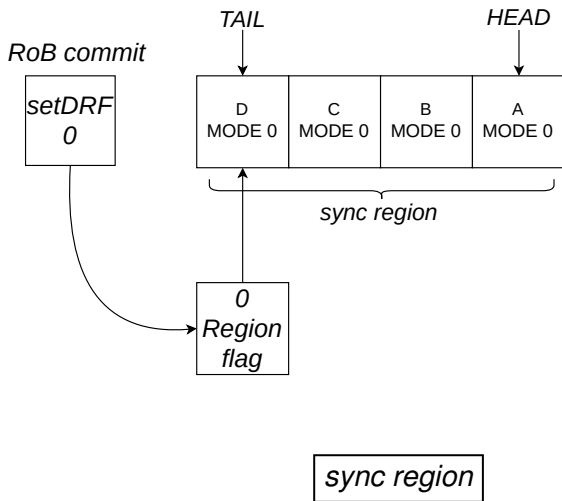
Store copy region flag to their mode bit

EXECUTION OVERVIEW

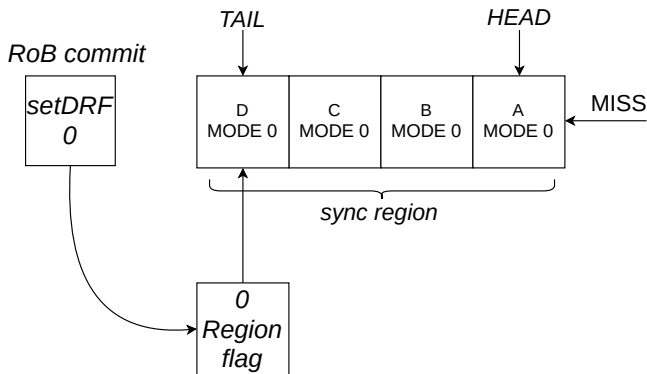


Store copy region flag to their mode bit

EXECUTION OVERVIEW

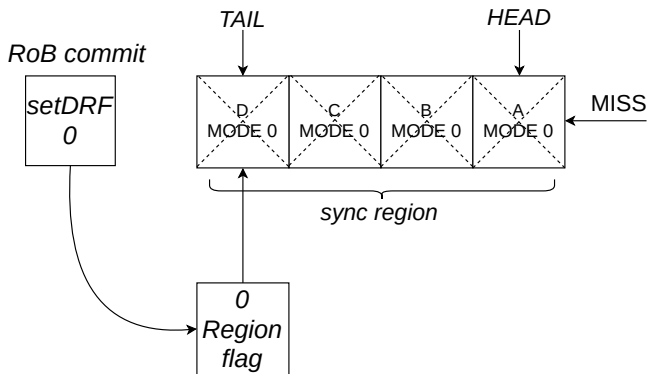


EXECUTION OVERVIEW



On miss

EXECUTION OVERVIEW



On miss all stores wait till the miss is resolved

EXECUTION OVERVIEW

RoB commit

setDRF
1

Setting DRF region

EXECUTION OVERVIEW

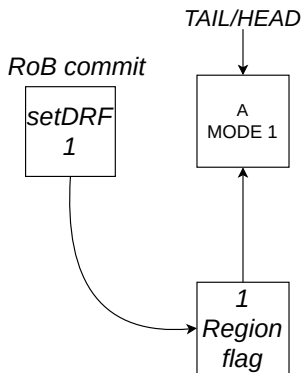
RoB commit

setDRF
1

1
Region
flag

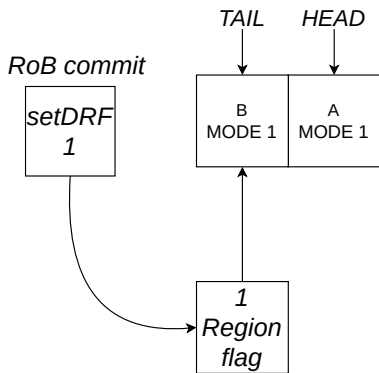
Set region flag

EXECUTION OVERVIEW



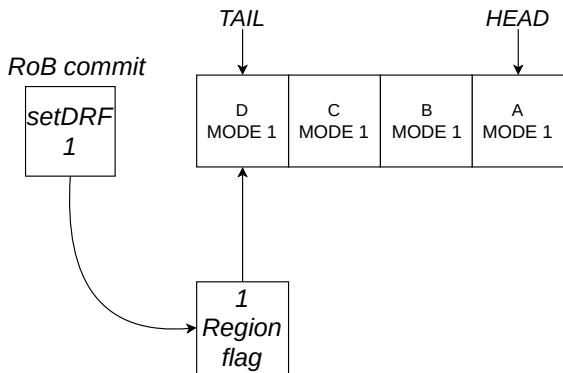
Store copy region flag to their mode bit

EXECUTION OVERVIEW



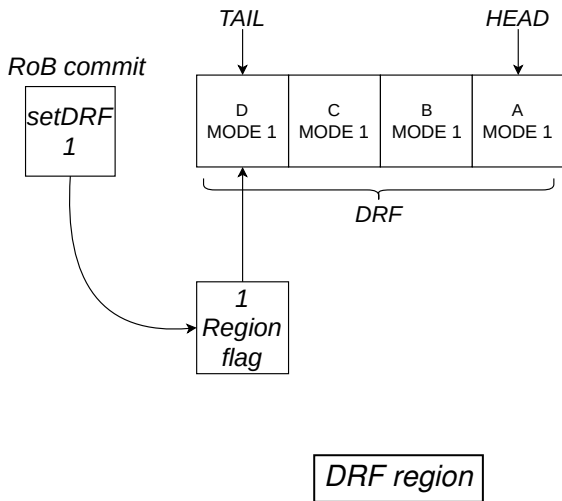
Store copy region flag to their mode bit

EXECUTION OVERVIEW

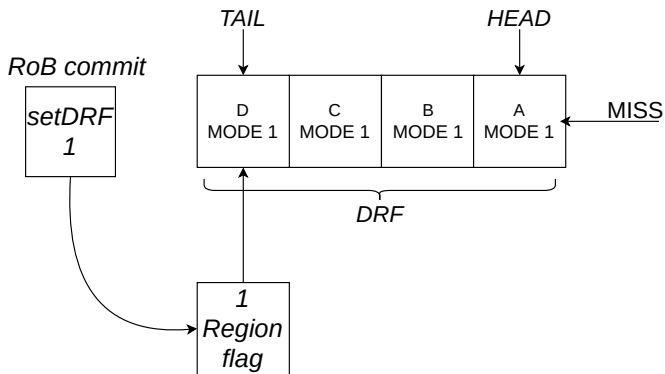


Store copy region flag to their mode bit

EXECUTION OVERVIEW

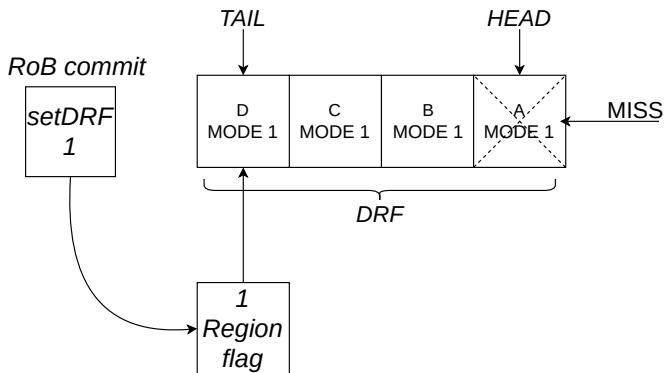


EXECUTION OVERVIEW



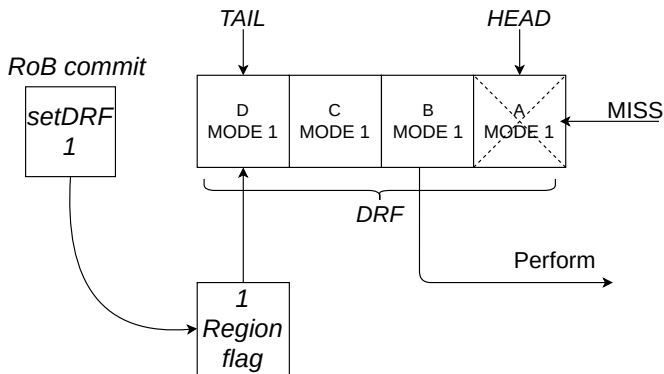
On miss

EXECUTION OVERVIEW



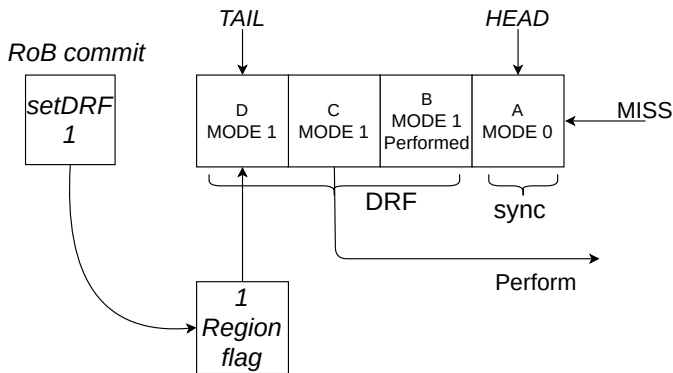
Store A will wait for the miss to be resolved

EXECUTION OVERVIEW



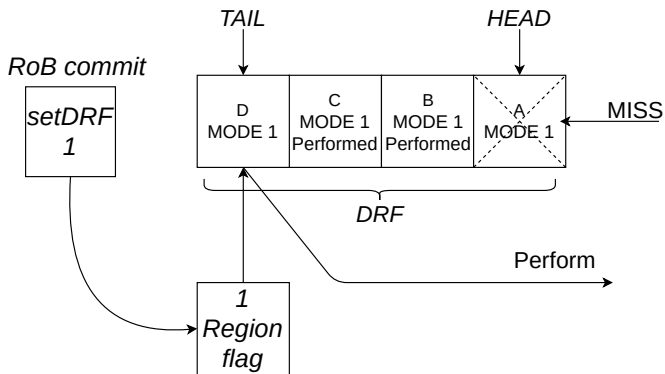
Next store will be allowed to perform

EXECUTION OVERVIEW



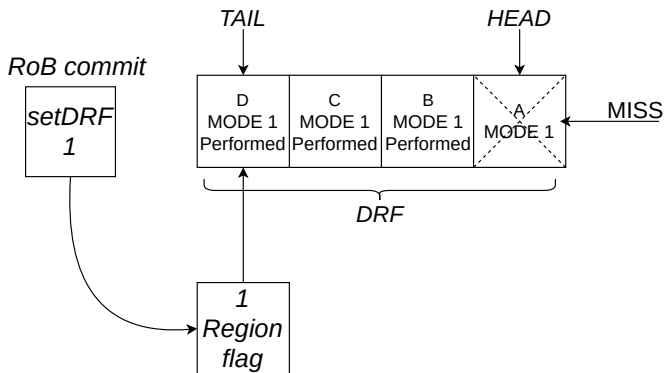
Similarly the next store will perform

EXECUTION OVERVIEW



Next store

EXECUTION OVERVIEW



All stores except A already performed the memory operation

Store to load forwarding

GUARANTEEING SEQUENTIAL SEMANTICS

Store to load forwarding
↓
Loads takes the **recent value**

GUARANTEEING SEQUENTIAL SEMANTICS

Store to load forwarding



Loads takes the **recent value**



Store enter and exit SB only in program order

GUARANTEEING SEQUENTIAL SEMANTICS

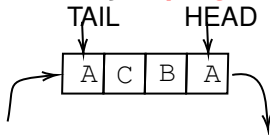
Store to load forwarding



Loads takes the **recent value**



Store enter and exit SB only in program order



GUARANTEEING SEQUENTIAL SEMANTICS

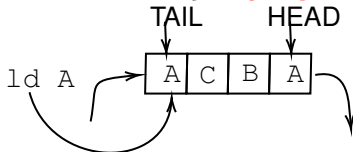
Store to load forwarding



Loads takes the **recent value**



Store enter and exit SB only in program order



GUARANTEEING SEQUENTIAL SEMANTICS

Alias stores should always go in-order

GUARANTEEING SEQUENTIAL SEMANTICS

Alias stores should always go in-order



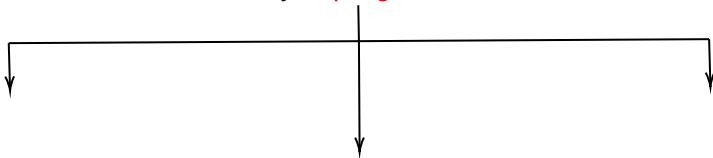
Memory operations should update
memory in program order

GUARANTEEING SEQUENTIAL SEMANTICS

Alias stores should always go in-order



Memory operations should update
memory in program order

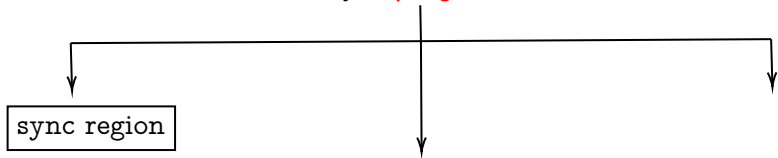


GUARANTEEING SEQUENTIAL SEMANTICS

Alias stores should always go in-order



Memory operations should update
memory in program order

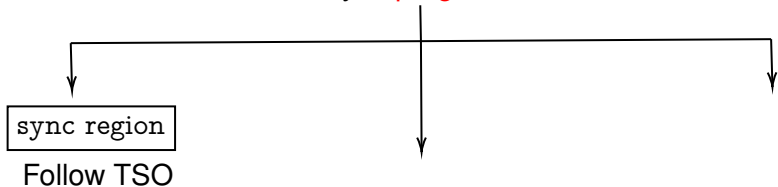


GUARANTEEING SEQUENTIAL SEMANTICS

Alias stores should always go in-order



Memory operations should update
memory in program order

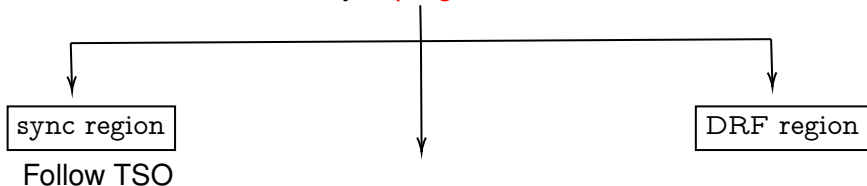


GUARANTEEING SEQUENTIAL SEMANTICS

Alias stores should always go in-order



Memory operations should update
memory in program order

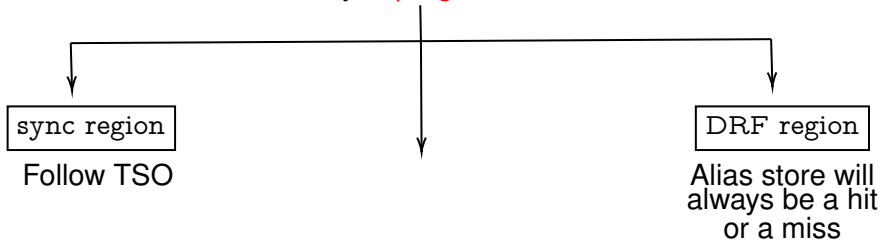


GUARANTEEING SEQUENTIAL SEMANTICS

Alias stores should always go in-order



Memory operations should update
memory in program order

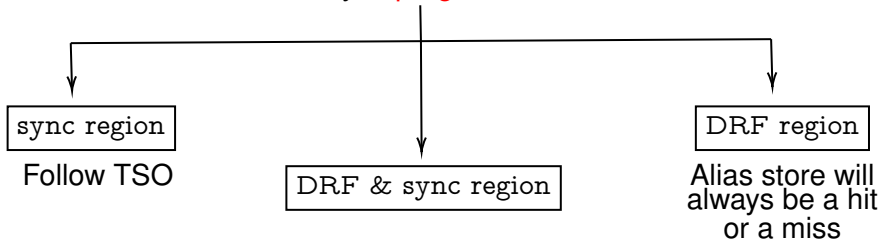


GUARANTEEING SEQUENTIAL SEMANTICS

Alias stores should always go in-order



Memory operations should update
memory in program order

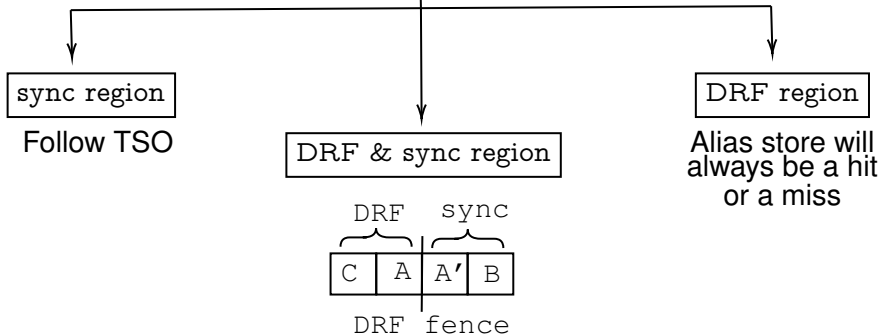


GUARANTEEING SEQUENTIAL SEMANTICS

Alias stores should always go in-order



Memory operations should update memory in program order



Exclusive access to memory during a DRF region, thanks to SC-for-DRF

Exclusive access to memory during a DRF region, thanks to
SC-for-DRF



No coherence problems

STORE BUFFER AS CACHE

Exclusive access to memory during a DRF region, thanks to
SC-for-DRF



No coherence problems



Can stay in SB until it gets full

STORE BUFFER AS CACHE

Exclusive access to memory during a DRF region, thanks to
SC-for-DRF



No coherence problems



Can stay in SB until it gets full



Increase store forwarding

Exclusive access to memory during a DRF region, thanks to
SC-for-DRF



No coherence problems



Can stay in SB until it gets full



Increase store forwarding



AT 0 COST!!

OUTLINE

1 MOTIVATION

2 ROOW

3 RESULTS

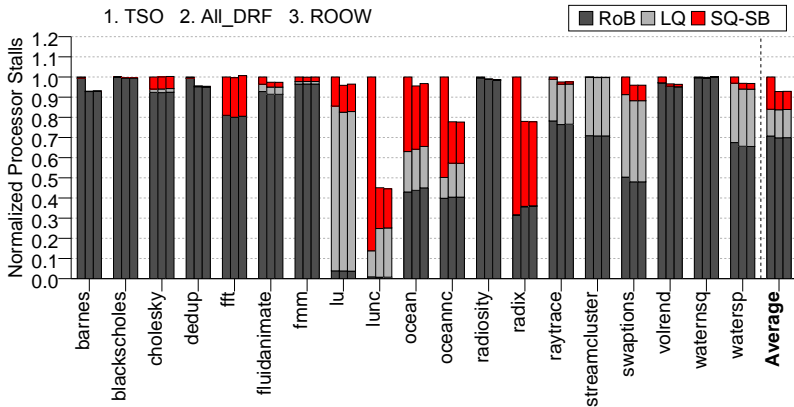
4 CONCLUSION

- Sniper + in-house processor model + GEMS
 - 8 out-of-order **Skylake**-like cores
 - Load queue: 72 entries
 - Store queue + store buffer: 56-16 entries
 - Re-order Buffer: 224 entries

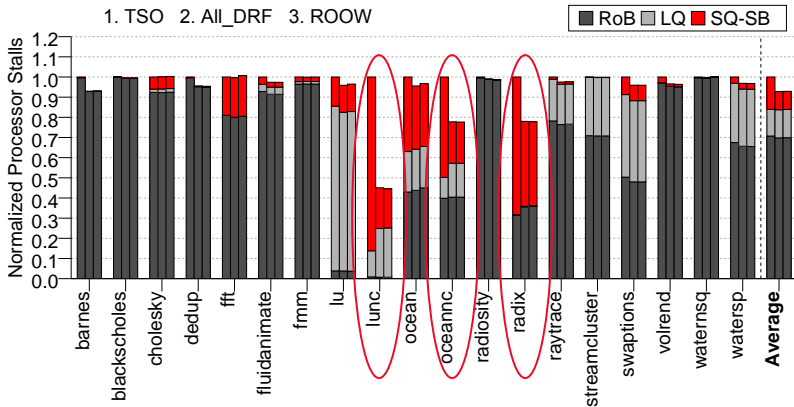
- Sniper + in-house processor model + GEMS
 - 8 out-of-order **Skylake**-like cores
 - Load queue: 72 entries
 - Store queue + store buffer: 56-16 entries
 - Re-order Buffer: 224 entries

- Parallel benchmarks
 - Splash-3
 - Parsec-3.0

Processor Stalls



Processor Stalls



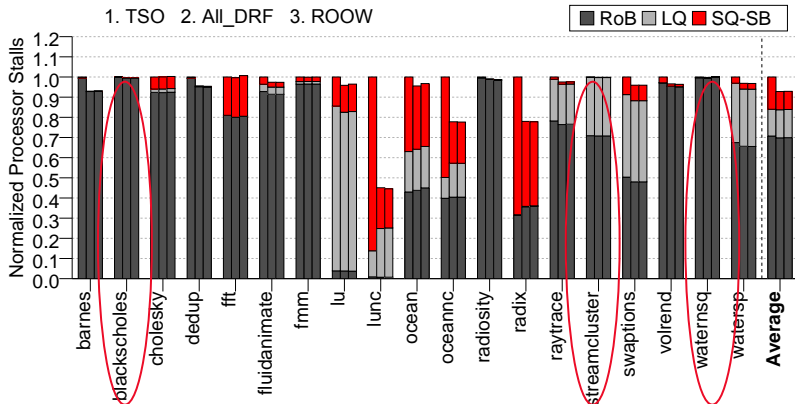
SB stalls reduces significantly

leads to improvement in execution time

(63.36 %, 19.16% & 20.40%)

RESULTS

Processor Stalls



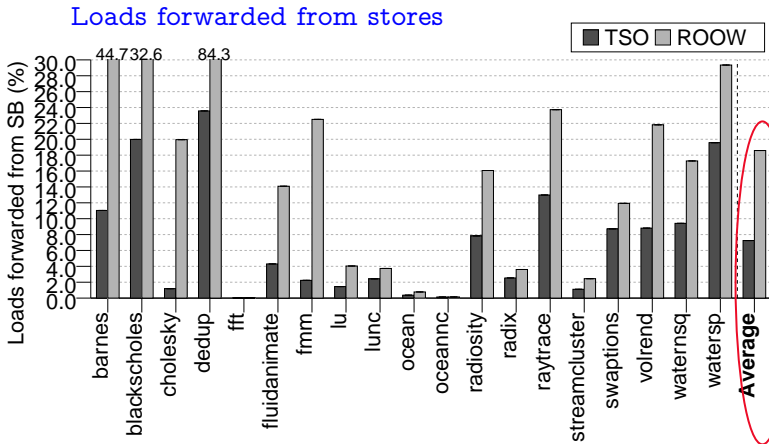
Few applications encounter very few SB stalls in TSO(baseline)
thus ROOW is not very effective

RESULTS

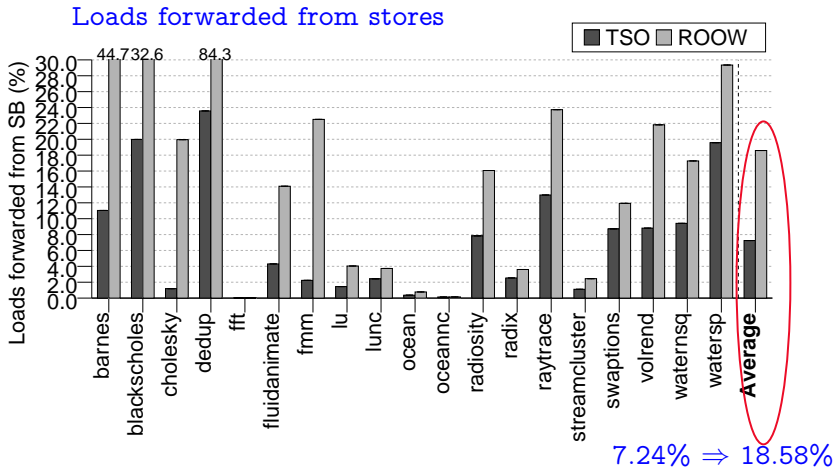
Loads forwarded from stores



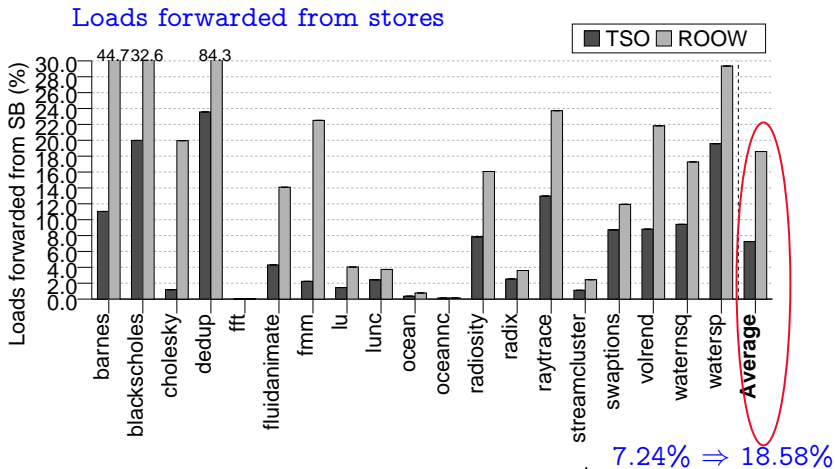
RESULTS



RESULTS



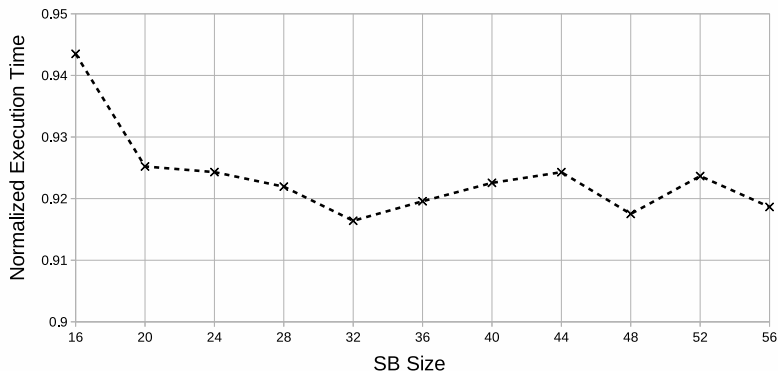
RESULTS



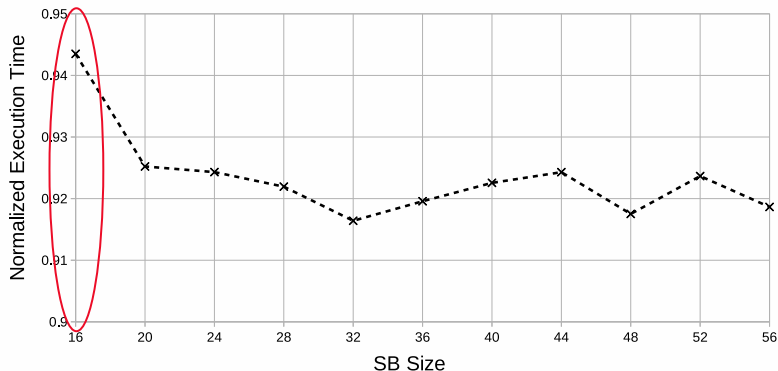
Can reduce **energy consumption** even more¹

¹Alves *et al.* Filter caching for free: The untapped potential of the store-buffer, ISCA'46 2019

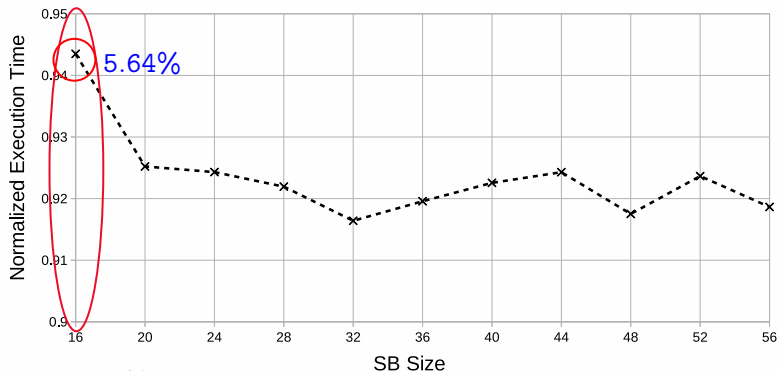
Sensitivity analysis



Sensitivity analysis

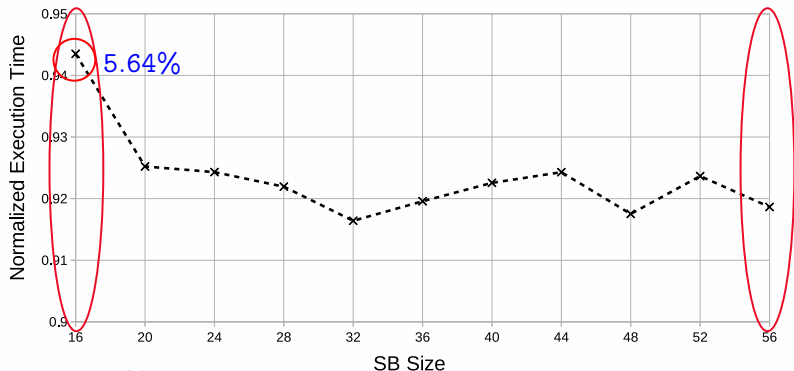


Sensitivity analysis



45.16%
Energy savings

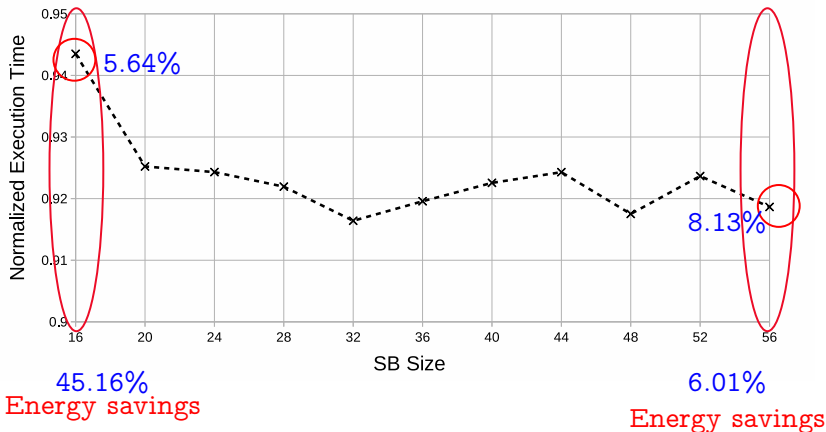
Sensitivity analysis



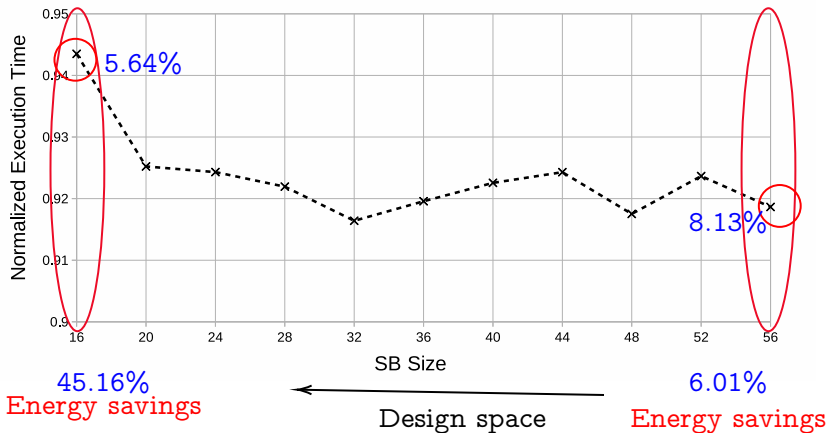
45.16%
Energy savings

RESULTS

Sensitivity analysis



Sensitivity analysis



OUTLINE

1 MOTIVATION

2 ROOW

3 RESULTS

4 CONCLUSION

CONCLUSIONS

- New **SB** design for non speculative **store-store** reordering
→ using **SC-for-DRF**

CONCLUSIONS

- New **SB** design for non speculative **store-store** reordering
 - using **SC-for-DRF**
- Analysis of
 - which stores can be **reordered**
 - how to **reorder stores**

CONCLUSIONS

- New **SB** design for non speculative **store-store** reordering
 - using **SC-for-DRF**
- Analysis of
 - which stores can be **reordered**
 - how to **reorder stores**
- Efficient implementation with just **$N+1$** bits²

² N : Number of entries in SB

CONCLUSIONS

- New **SB** design for non speculative **store-store** reordering
 - using **SC-for-DRF**
- Analysis of
 - which stores can be **reordered**
 - how to **reorder stores**
- Efficient implementation with just **N+1** bits²
- Results:
 - **56 ENTRIES SB**: **performance** and **energy consumption**
(+8.13%/-6.01%)
 - **16 ENTRIES SB**: **performance** and **energy consumption**
(+5.64%/-45.16%)

² N : Number of entries in SB

CONCLUSIONS

- New **SB** design for non speculative **store-store** reordering
 - using **SC-for-DRF**
- Analysis of
 - which stores can be **reordered**
 - how to **reorder stores**
- Efficient implementation with just **N+1** bits²
- Results:
 - **56 ENTRIES SB**: **performance** and **energy consumption**
(+8.13%/-6.01%)
 - **16 ENTRIES SB**: **performance** and **energy consumption**
(+5.64%/-45.16%)
- Can we get better performance?

² N : Number of entries in SB

CONCLUSIONS

- New **SB** design for non speculative **store-store** reordering
 - using **SC-for-DRF**
- Analysis of
 - which stores can be **reordered**
 - how to **reorder stores**
- Efficient implementation with just **N+1** bits²
- Results:
 - **56 ENTRIES SB**: **performance** and **energy consumption** (+8.13%/-6.01%)
 - **16 ENTRIES SB**: **performance** and **energy consumption** (+5.64%/-45.16%)
- Can we get better performance?
 - Yes, by using **xDRF compiler**³ (+1%)

² N : Number of entries in SB

³ Jimborean *et al.* Automatic Detection of Extended Data-Race-Free Regions, CGO 2017

REGIONAL OUT-OF-ORDER WRITES IN TOTAL STORE ORDER

Sawan Singh Alexandra Jimborean Alberto Ros

sawan.singh@um.es
CAPS, DITEC University of Murcia

Thank you for your attention!



ECHO, ERC Consolidator Grant (No 819134)

This presentation and recording belong to the authors. No distribution is allowed without the authors' permission.

