Rebasing and Optimizing Cache Miss Handling: Fetching Beyond MSHR Capacity

Simranjit Singh[1] Agustín Navarro-Torres[2]
Biswabandan Panda[3] Alberto Ros[1]

[1]Computer Engineering Department
University of Murcia, Spain

[2]Dept. Informática e Ingeniería de Sistemas, I3A
Universidad de Zaragoza, Spain

[3]Dept. of Computer Science and Engineering
IIT Bombay, India

Overview

- L1D Miss Status Handling Registers (MSHRs) track outstanding cache misses
  - Their capacity limits memory level parallelism (MLP) and performance
  - L1D MSHRs are hard to scale

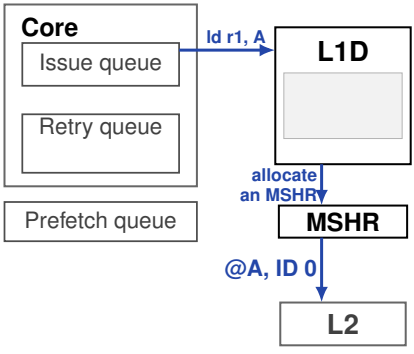
Overview

- L1D Miss Status Handling Registers (MSHRs) track outstanding cache misses
  - Their capacity limits memory level parallelism (MLP) and performance
  - L1D MSHRs are hard to scale
- Gap between academia and industry
  - Academic simulators stop handling misses when L1D MSHRs are full
  - This is not the industry standard: Industry designs prefetch data to L2
  - This has a large performance impact, so academic simulators should model them

Overview

- L1D Miss Status Handling Registers (MSHRs) track outstanding cache misses
  - Their capacity limits memory level parallelism (MLP) and performance
  - L1D MSHRs are hard to scale
- Gap between academia and industry
  - Academic simulators stop handling misses when L1D MSHRs are full
  - This is not the industry standard: Industry designs prefetch data to L2
  - This has a large performance impact, so academic simulators should model them
- Contributions
  - We are the first to evaluate advanced industry MSHR designs
  - NOMAD: Improves industry designs by filling into L1D when MSHR is full

Background: How MSHRs work

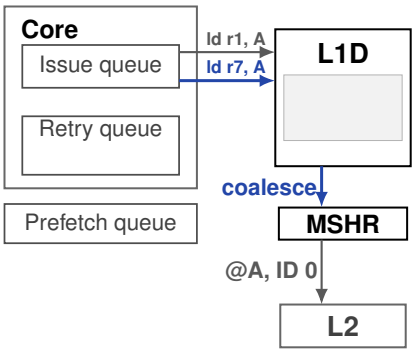


L1D MSHR

V	ADDR	ID	Depending
1	**A**	0	r1
0	–	–	–
0	–	–	–
0	–	–	–

(1) Track outstanding misses: record the address and depending instructions

Background: How MSHRs work



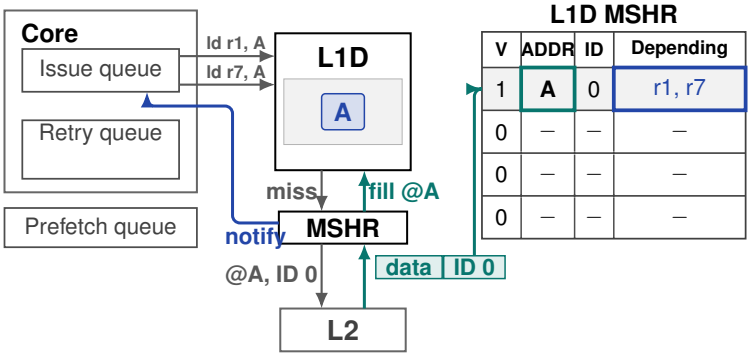
L1D MSHR

V	ADDR	ID	Depending
1	**A**	0	r1, r7
0	–	–	–
0	–	–	–
0	–	–	–

(1) Track outstanding misses: record the address and depending instructions

(2) Coalesce requests to the same line into the existing MSHR entry

Background: How MSHRs work

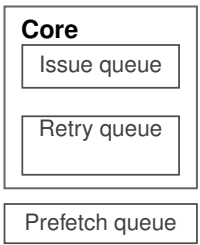


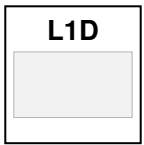
(1) Track outstanding misses: record the address and depending instructions

(2) Coalesce requests to the same line into the existing MSHR entry

(3) Handle the response: fill the L1D and notify depending instructions

What happens when the MSHR is full?





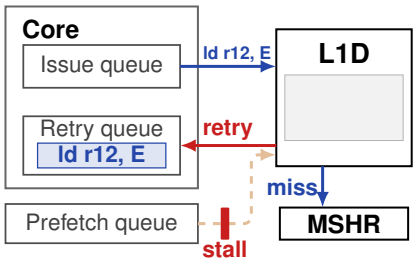
L1D MSHR full

V	ADDR	ID	Depending
1	B	0	r2, r5, r9
1	C	1	r3, r8
1	D	2	r4, r6, r11
1	F	3	r7, r10

(1) Cache misses cannot be handled without an MSHR entry

Academic model

What happens when the MSHR is full?



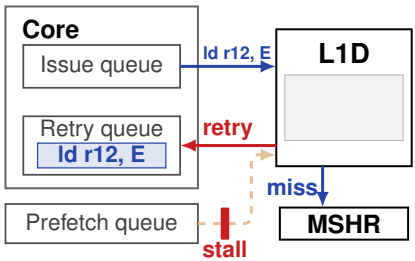
L1D MSHR full

V	ADDR	ID	Depending
1	B	0	r2, r5, r9
1	C	1	r3, r8
1	D	2	r4, r6, r11
1	F	3	r7, r10

(1) Cache misses cannot be handled without an MSHR entry

- Prefetch queue stalls
- Demand misses must retry

What happens when the MSHR is full?



L1D MSHR full

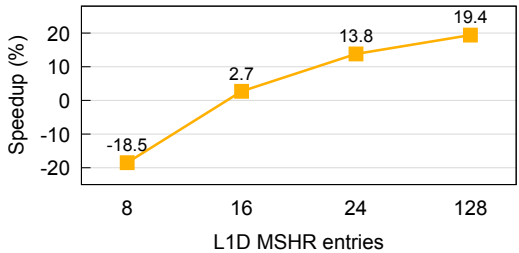
V	ADDR	ID	Depending
1	B	0	r2, r5, r9
1	C	1	r3, r8
1	D	2	r4, r6, r11
1	F	3	r7, r10

(1) Cache misses cannot be handled without an MSHR entry

- Prefetch queue stalls
- Demand misses must retry

(2) The L1D MSHR is full 50% of the time on AI/ML workloads

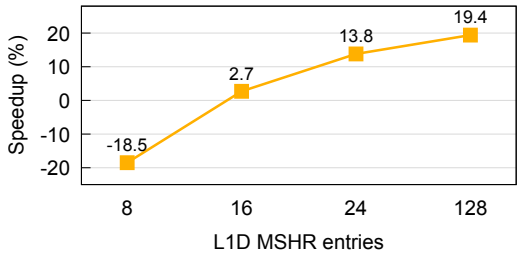
WHY NOT ADD MORE MSHR ENTRIES?



More MSHR entries bring more performance

AI/ML workloads using our methodology

WHY NOT ADD MORE MSHR ENTRIES?

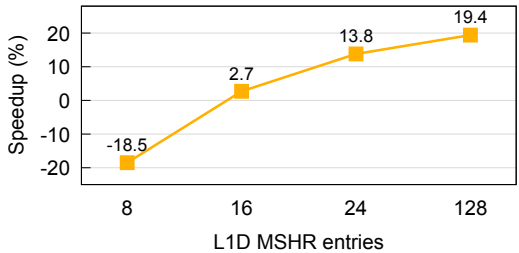


More MSHR entries bring more performance

Larger **associative searches** cost more area and power.
MSHR searches must meet L1D's **latency constraints**.

AI/ML workloads using our methodology

WHY NOT ADD MORE MSHR ENTRIES?



More MSHR entries bring more performance

Larger **associative searches** cost more area and power.
MSHR searches must meet L1D's **latency constraints**.

We don't have to!

AI/ML workloads using our methodology

How does industry handle a full MSHR?

INDUSTRY DESIGNS



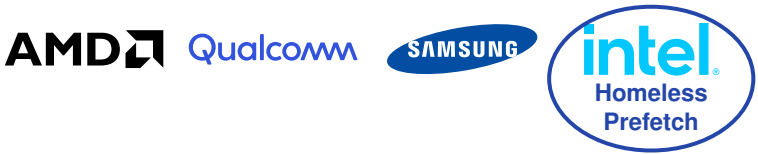
- Public information is limited

INDUSTRY DESIGNS



- Public information is limited
- We gather information from patents and vendor guides

INDUSTRY DESIGNS



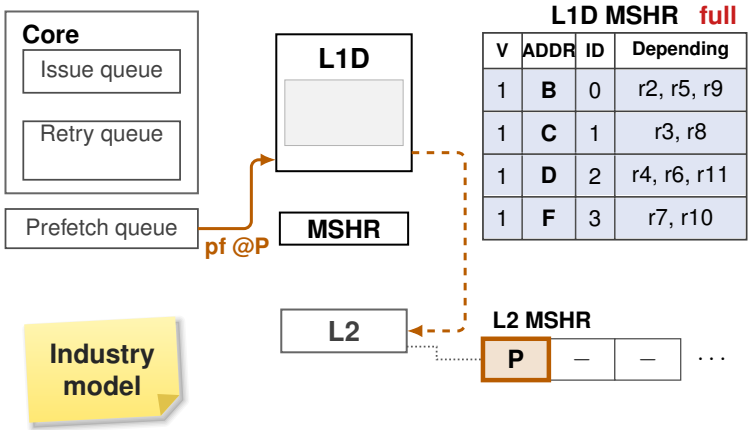
- Public information is limited
- We gather information from patents and vendor guides

We model **Homeless**

It is **representative** of the other designs
and is **deployed** on Intel server platforms[1]

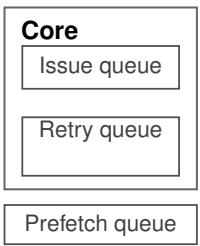
[1] Dell, Fujitsu and Lenovo BIOS guides; SPEC CPU2017 flag descriptions by Nettrix, Compal and Fujitsu.

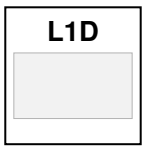
INDUSTRY MSHR HANDLING (HOMELESS)



(1) Homeless demotes prefetches to L2

INDUSTRY MSHR HANDLING (HOMELESS)





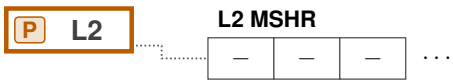
L1D MSHR full

V	ADDR	ID	Depending
1	B	0	r2, r5, r9
1	C	1	r3, r8
1	D	2	r4, r6, r11
1	F	3	r7, r10

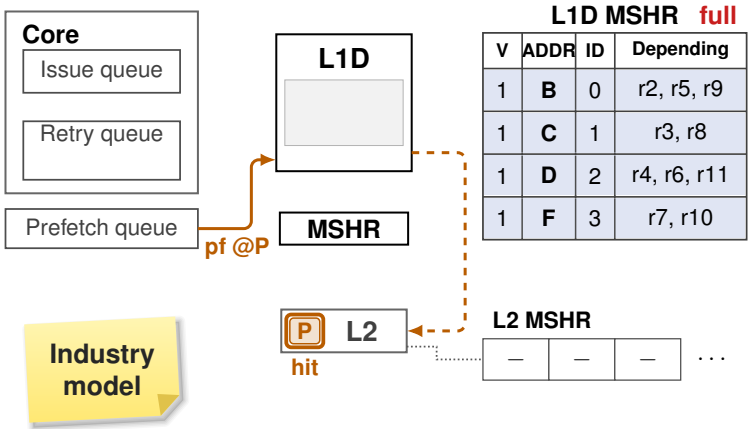
(1) Homeless demotes prefetches to L2

- It fills L2 but not L1D

Industry model



INDUSTRY MSHR HANDLING (HOMELESS)

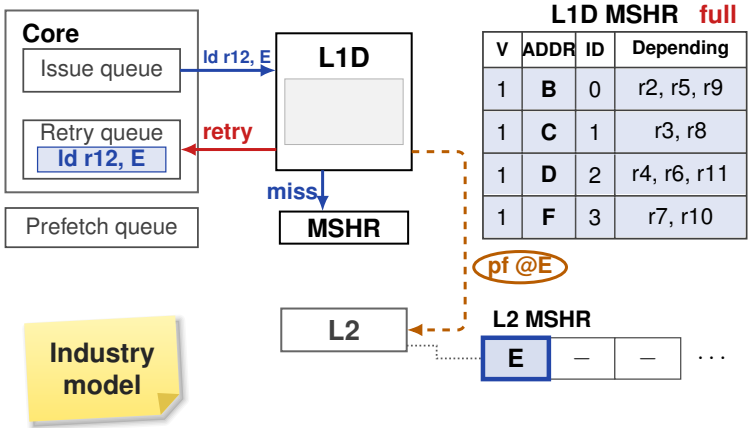


(1) Homeless demotes prefetches to L2

- It fills L2 but not L1D

(2) Coalescing is handled via L2 MSHRs or L2 hits

INDUSTRY MSHR HANDLING (HOMELESS)



(1) Homeless demotes prefetches to L2

- It fills L2 but not L1D

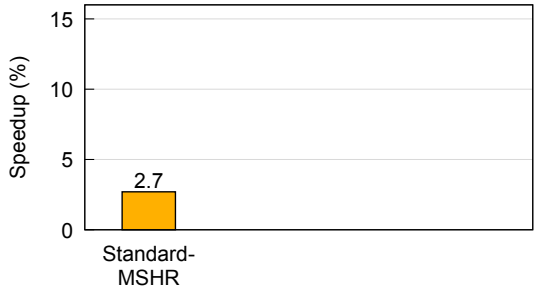
(2) Coalescing is handled via L2 MSHRs or L2 hits

(3) Homeless Demand to Prefetch (D2P): a demand miss to E also triggers a prefetch for E

METHODOLOGY

- Detailed cycle-level core (in-house) + SLICC + GARNET
  - Out-of-order, Golden Cove parameters
  - L1D: 48 KB, 16 MSHR entries, Berti L1D prefetcher
  - L2: 1.25 MB, 32 MSHR entries, Pythia L2 prefetcher
  - LLC: 3 MB per core, 64 MSHR entries
  - DRAM: Mess simulator, DDR4-2666
- DPC4 traces: GAP, Ligra, SPEC17, GMS, Google, AI/ML
  - 50M instructions of warm-up, 200M measured

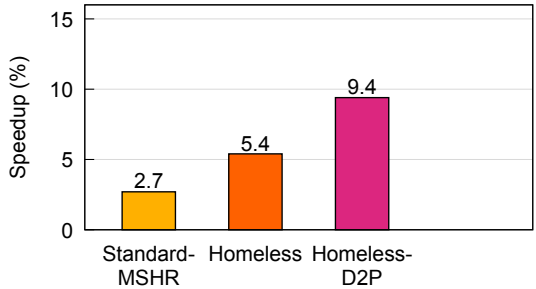
RE-EVALUATING THE BASELINE



Berti provides 2.7% speedup with the academic MSHR model

AI/ML workloads using our methodology

RE-EVALUATING THE BASELINE

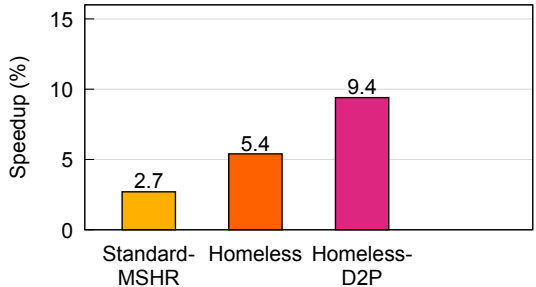


Berti provides 2.7% speedup with the academic MSHR model

Industry designs improve performance

AI/ML workloads using our methodology

RE-EVALUATING THE BASELINE



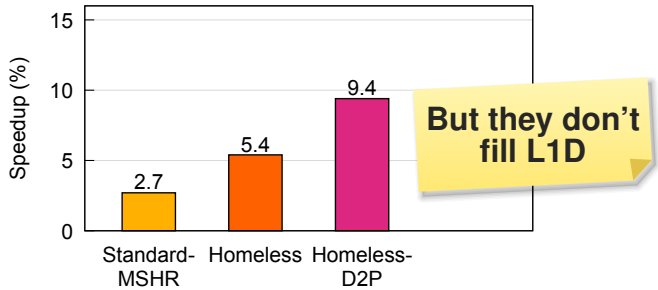
Berti provides 2.7% speedup with the academic MSHR model

Industry designs improve performance

Prefetchers must be evaluated with realistic MSHR designs

AI/ML workloads using our methodology

RE-EVALUATING THE BASELINE



Berti provides 2.7% speedup with the academic MSHR model

Industry designs improve performance

Prefetchers must be evaluated with realistic MSHR designs

AI/ML workloads using our methodology

CAN WE PUSH IT FURTHER?

CAN WE PUSH IT FURTHER?

Yes, we can bring prefetches to L1D
without allocating an MSHR entry

How?

LETS REVISIT L1D MSHR FUNCTIONS

(1) **Notify depending instructions**

LETS REVISIT L1D MSHR FUNCTIONS

(1) **Notify depending instructions**

- **Prefetches do not need to notify depending instructions**

LETS REVISIT L1D MSHR FUNCTIONS

(1) **Notify depending instructions**
- **Prefetches do not need to notify depending instructions**

(2) **Coalesce requests**

LETS REVISIT L1D MSHR FUNCTIONS

(1) **Notify depending instructions**
- **Prefetches do not need to notify depending instructions**

(2) **Coalesce requests**
- Delegate to L2 MSHRs

LETS REVISIT L1D MSHR FUNCTIONS

(1) **Notify depending instructions**
- **Prefetches do not need to notify depending instructions**

(2) **Coalesce requests**
- Delegate to L2 MSHRs

(3) **Identify returning data**

LETS REVISIT L1D MSHR FUNCTIONS

(1) **Notify depending instructions**
- **Prefetches do not need to notify depending instructions**

(2) **Coalesce requests**
- Delegate to L2 MSHRs

(3) **Identify returning data**
- Extend the L2-to-L1D response with the **line address**

LETS REVISIT L1D MSHR FUNCTIONS

(1) **Notify depending instructions**
- **Prefetches do not need to notify depending instructions**

(2) **Coalesce requests**
- Delegate to L2 MSHRs

(3) **Identify returning data**
- Extend the L2-to-L1D response with the **line address**

Prefetches can fill L1D without an L1D MSHR entry

Introducing Nomad

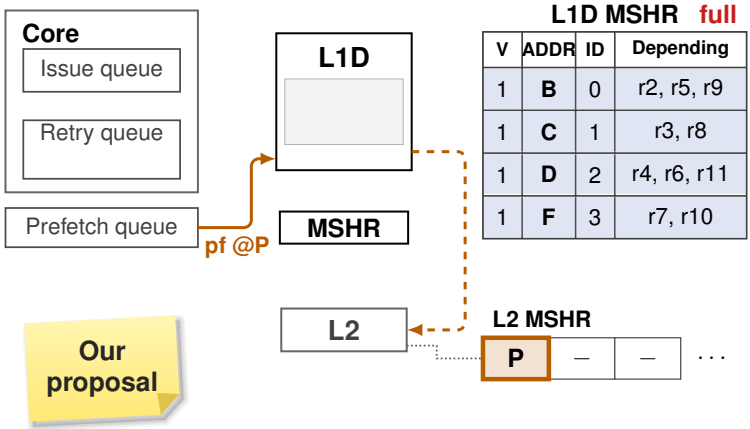
Nomad combines three techniques to fetch beyond L1D MSHR capacity.

(1) **MSHR bypass**

(2) **MSHR eviction**

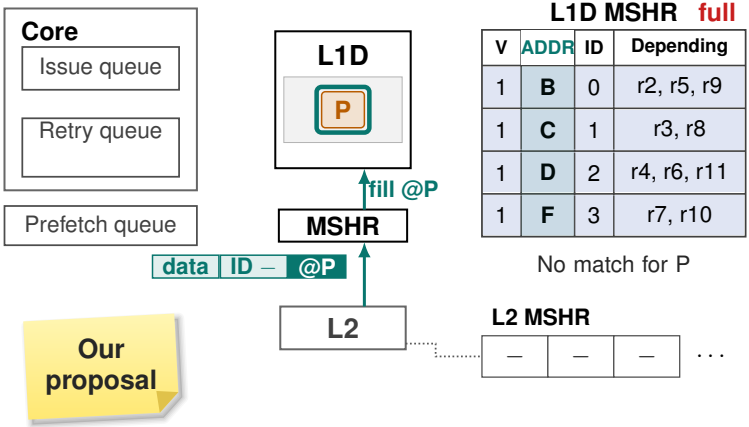
(3) **Demand-to-prefetch (L1D)**

NOMAD (1): MSHR BYPASS



(1) Nomad sends the prefetch without an entry

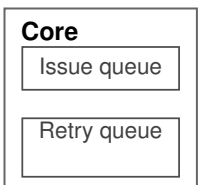
NOMAD (1): MSHR BYPASS

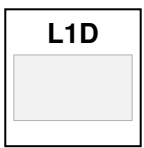


(1) Nomad sends the prefetch without an entry

(2) Nomad uses the address in the prefetch response to search the MSHR and fill L1D without requiring a matching entry

NOMAD (2): MSHR EVICTION

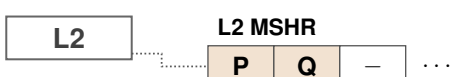




L1D MSHR **full**

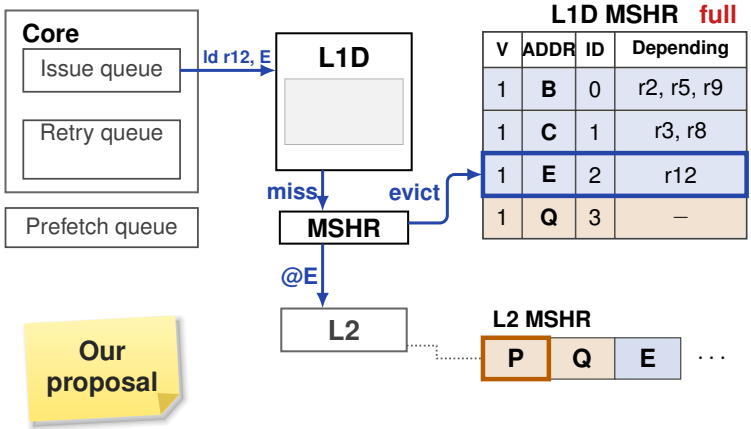
V	ADDR	ID	Depending
1	B	0	r2, r5, r9
1	C	1	r3, r8
1	P	2	–
1	Q	3	–

(1) Prefetches use free MSHR entries to help coalescing



Our proposal

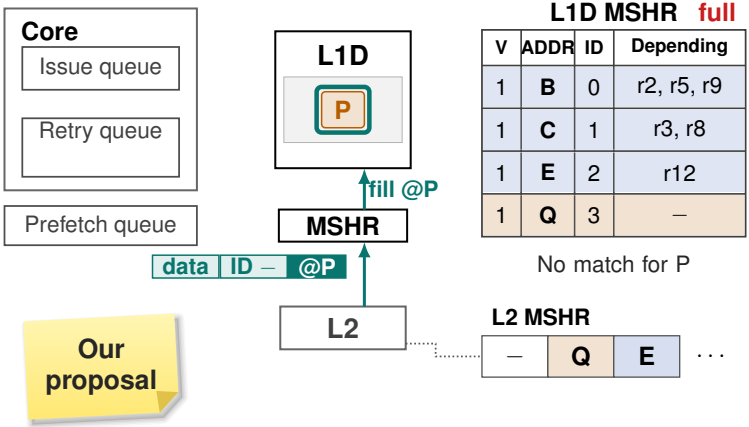
NOMAD (2): MSHR EVICTION



(1) Prefetches use free MSHR entries to help coalescing

(2) However, if the full MSHR contains a prefetch entry, a demand miss reclaims it while the prefetch remains in flight

NOMAD (2): MSHR EVICTION

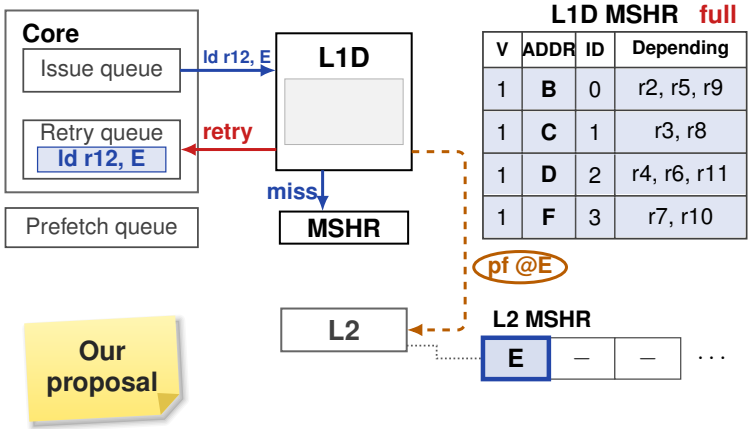


(1) Prefetches use free MSHR entries to help coalescing

(2) However, if the full MSHR contains a prefetch entry, a demand miss reclaims it while the prefetch remains in flight

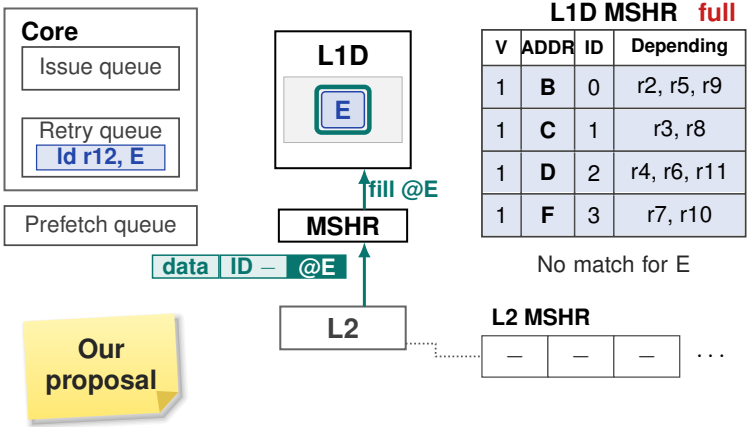
(3) The prefetch still fills L1D

NOMAD (3): DEMAND-TO-PREFETCH



(1) When all MSHR entries hold demands, a miss to E triggers a prefetch for E

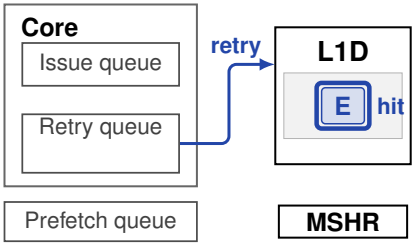
NOMAD (3): DEMAND-TO-PREFETCH



(1) When all MSHR entries hold demands, a miss to E triggers a prefetch for E

(2) The prefetch fills L1D

NOMAD (3): DEMAND-TO-PREFETCH



L1D MSHR full

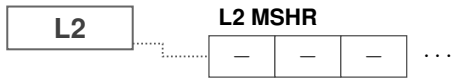
V	ADDR	ID	Depending
1	B	0	r2, r5, r9
1	C	1	r3, r8
1	D	2	r4, r6, r11
1	F	3	r7, r10

(1) When all MSHR entries hold demands, a miss to E triggers a prefetch for E

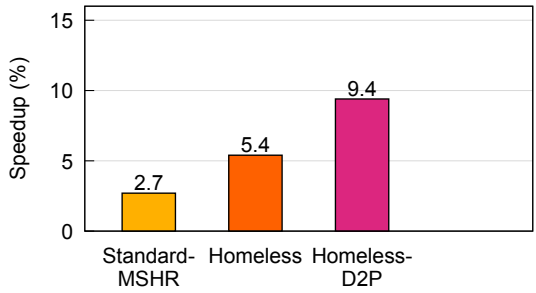
(2) The prefetch fills L1D

(3) When the demand retries, it hits in L1D

Our proposal

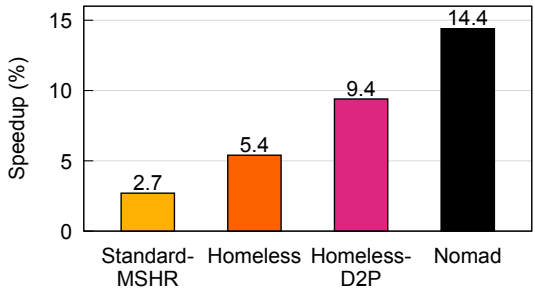


RESULTS: NOMAD VERSUS HOMELESS-D2P



AI/ML workloads using our methodology

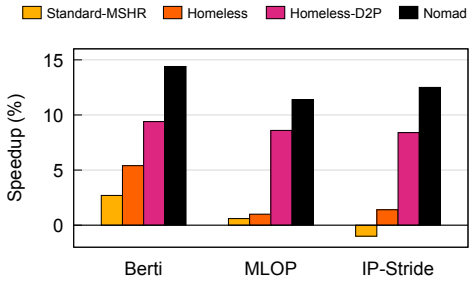
RESULTS: NOMAD VERSUS HOMELESS-D2P



Nomad outperforms Homeless-D2P

AI/ML workloads using our methodology

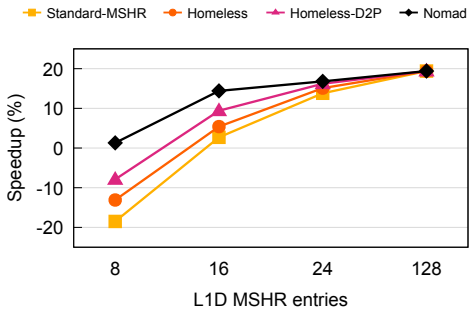
RESULTS: L1D PREFETCHERS



Nomad leads with all three L1D prefetchers

AI/ML workloads using our methodology

RESULTS: MSHR SIZE



Nomad does not need a larger MSHR

AI/ML workloads using our methodology

Conclusions

- L1D MSHRs limit MLP and are hard to scale
  - Full 50% of the time on AI/ML workloads
- Academic simulators do not model realistic MSHRs
  - Berti's speedup changes from 2.7% to 9.4% with Homeless-D2P
  - Prefetchers should be evaluated with realistic MSHRs
- Nomad: Can fill L1D even when the MSHR is full

Rebasing and Optimizing Cache Miss Handling: Fetching Beyond MSHR Capacity

Simranjit Singh[1] Agustín Navarro-Torres[2]
Biswabandan Panda[3] Alberto Ros[1]

simranjit.singh@um.es

Thank you for your attention!



ERC ECHO (819134) · Fundación Séneca SCALE (23039/GERM/25) · Government of Aragon (T58_23R)
MICIU/AEI/10.13039/501100011033 and ERDF, EU: DAMAS (PID2022-136315OB-I00)
MICIU/AEI/10.13039/501100011033: TAPAS (PDC2025-166157-I00), APCE III (PID2022-136454NB-C22)

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

Additional results

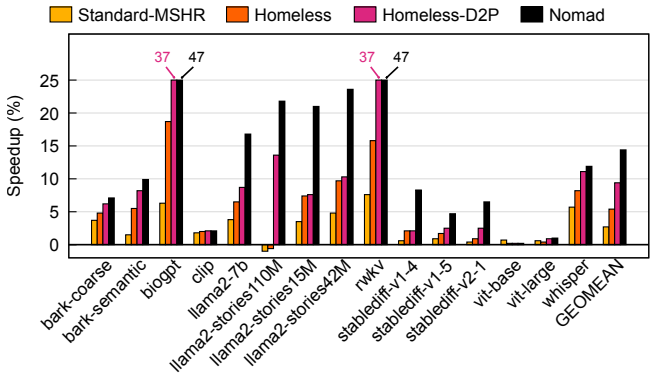
- All suites at a glance
- Per application
  AI/ML · GAP · Ligra
  SPEC · GMS · Google
- Without an L1D prefetcher
- Ablation
- Other L1D prefetchers
- Limited DRAM bandwidth
- Traffic
- L2 MSHR-full cycles
- Dynamic energy
- Hardware cost
- Multicore
- A later load to P

All suites at a glance

	MSHR-full	Speedup (%)			
	cycles (%)	Standard	Hml.	H-D2P	Nomad
AI/ML	50.3	2.7	5.4	9.4	**14.4**
GAP	26.5	0.0	0.3	10.0	**11.3**
Ligra	12.4	7.2	8.6	**10.2**	10.0
SPEC17	4.6	5.0	5.4	**5.6**	5.5
Google	4.5	1.1	1.5	2.2	**2.6**
GMS	0.8	−0.1	0.1	0.5	**0.6**

All suites, Berti at L1D, 16 entries. MSHR-full cycles measured with Standard-MSHR.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

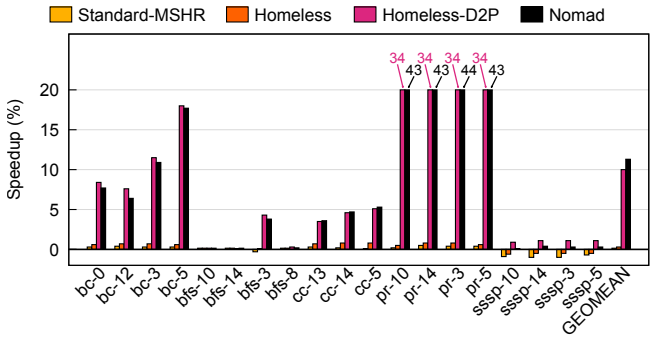
Per application: AI/ML



AI/ML, Berti at L1D, 16 entries.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

Index

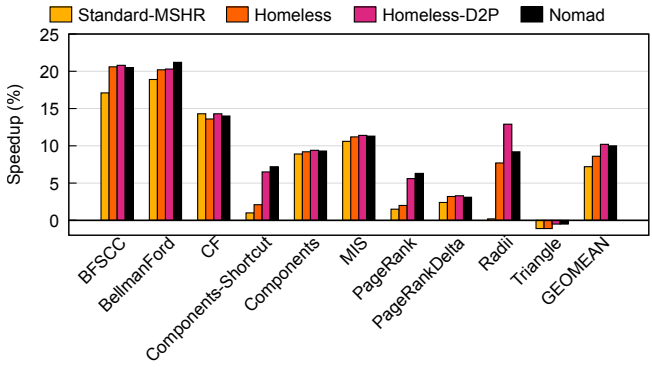
Per application: GAP



GAP, Berti at L1D, 16 entries.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

Index

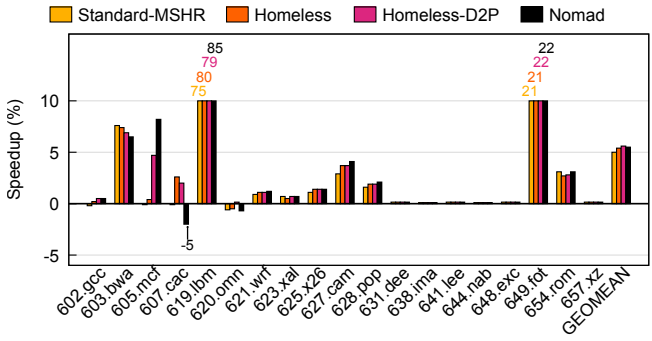
Per application: Ligra



Ligra, Berti at L1D, 16 entries.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

Index

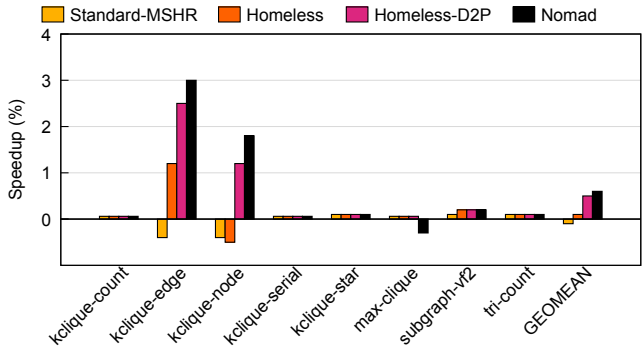
PER APPLICATION: SPEC CPU2017



SPEC CPU2017, Berti at L1D, 16 entries.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

Index

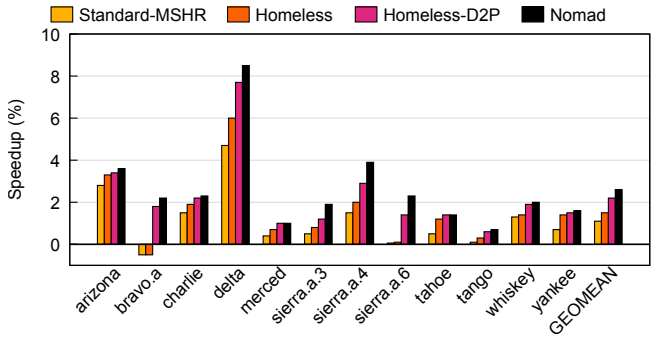
Per application: GMS



GMS, Berti at L1D, 16 entries.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

Index

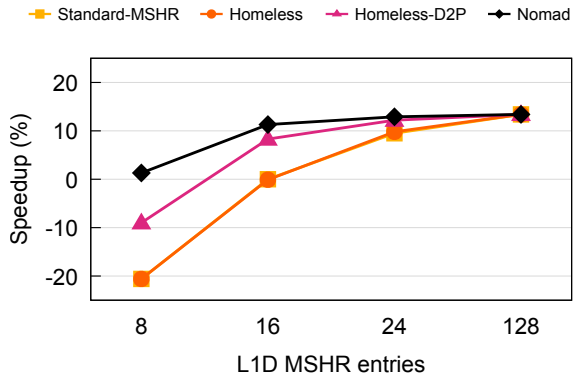
Per application: Google



Google, Berti at L1D, 16 entries.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

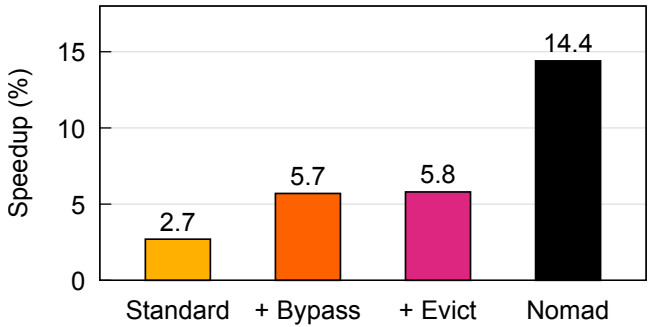
Index

WITHOUT AN L1D PREFETCHER



AI/ML workloads, no L1D prefetcher.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

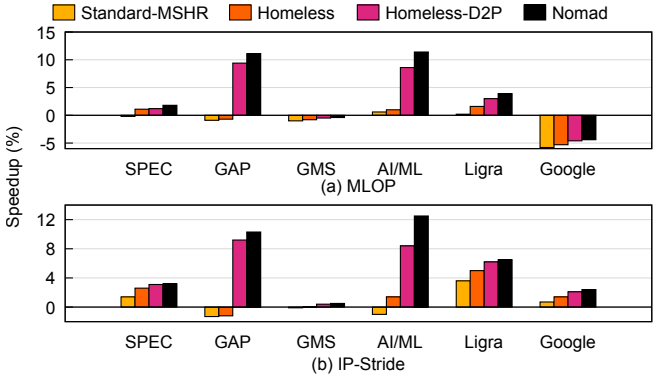
Ablation



AI/ML workloads, Berti at L1D, 16 entries.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

Index

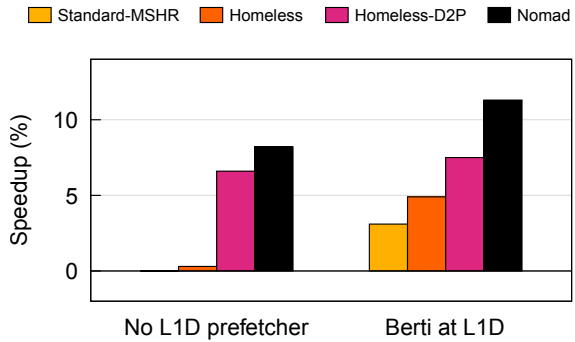
Other L1D prefetchers



All suites, MLOP and IP-Stride at L1D, 16 entries.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

Index

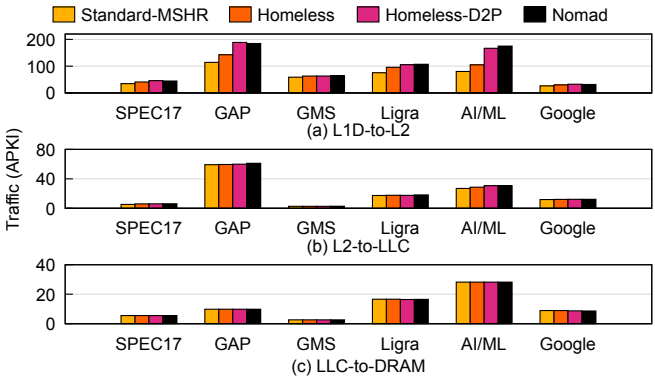
LIMITED DRAM BANDWIDTH



AI/ML workloads, 16 entries, 1/8 DRAM bandwidth.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, same bandwidth.

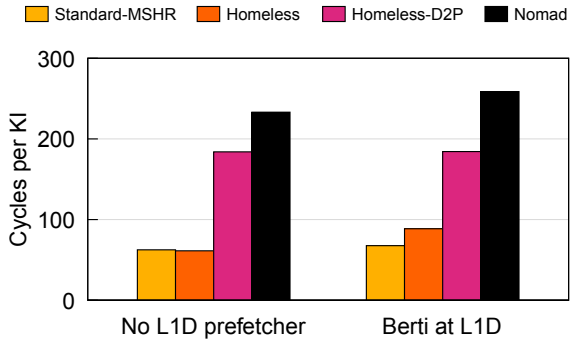
Index

TRAFFIC



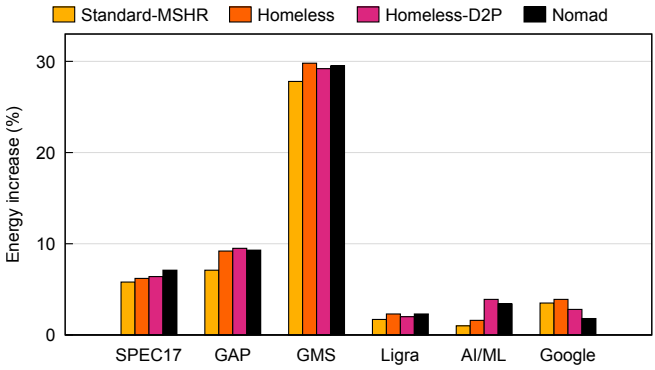
All suites, Berti at L1D, 16 entries.

L2 MSHR-FULL CYCLES



AI/ML workloads, 16 L1D and 32 L2 MSHR entries.

DYNAMIC ENERGY



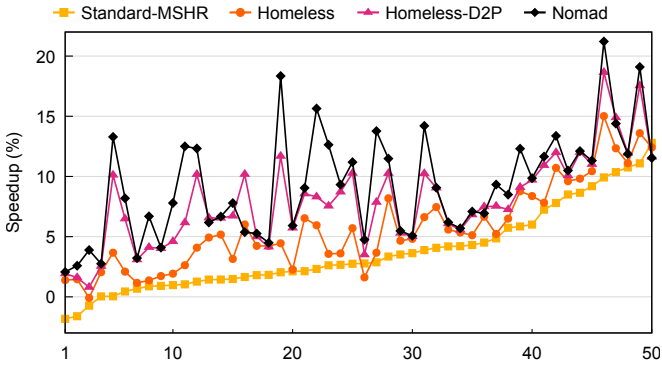
All suites, Berti at L1D, 16 entries.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

Index

HARDWARE COST

Channel	Direction	Purpose	Base	Nomad
TXREQ	L1D → interconnect	Send requests	178	178
TXDAT	L1D → interconnect	Send data	374	374
TXRSP	L1D → interconnect	Send responses	77	77
RXDAT	interconnect → L1D	Receive data	374	**438**
RXRSP	interconnect → L1D	Receive responses	77	77
RXSNP	interconnect → L1D	Receive snoops	114	114
Total	–	–	**1194**	**1258**

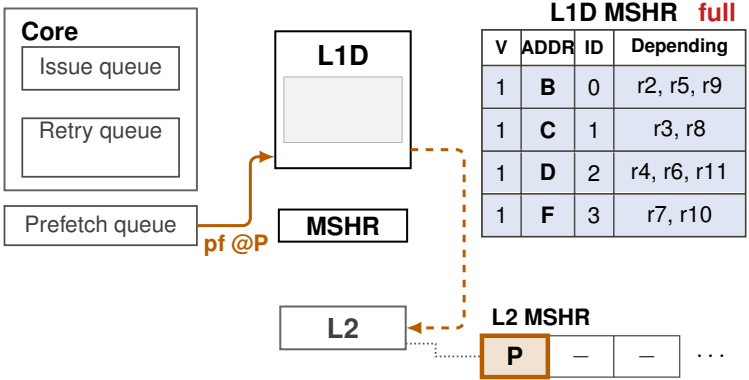
MULTICORE



50 four-core mixes, Berti at L1D, 16 entries.
Baseline: 16-entry Standard-MSHR, no L1D prefetcher, Pythia at L2.

Index

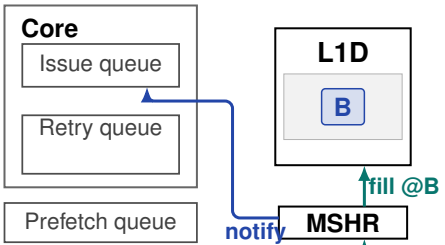
A later load to P



(1) Prefetch P bypasses the full MSHR

Index

A LATER LOAD TO P



L1D MSHR full

V	ADDR	ID	Depending
1	B	0	r2, r5, r9
1	C	1	r3, r8
1	D	2	r4, r6, r11
1	F	3	r7, r10

L2 MSHR

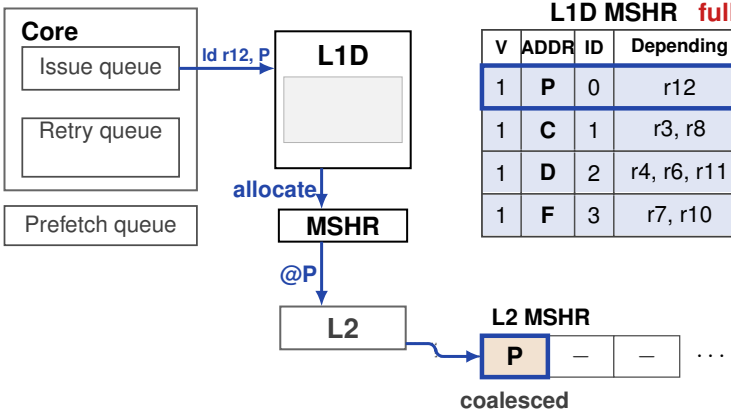
P	–	–	...

(1) Prefetch P bypasses the full MSHR

(2) B returns and frees an entry

Index

A LATER LOAD TO P



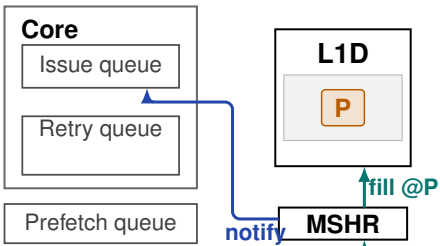
(1) Prefetch P bypasses the full MSHR

(2) B returns and frees an entry

(3) A later load to P allocates that entry

Index

A LATER LOAD TO P



L1D MSHR full

V	ADDR	ID	Depending
1	P	0	r12
1	C	1	r3, r8
1	D	2	r4, r6, r11
1	F	3	r7, r10

P matches entry 0

L2 MSHR

–	–	–	...

(1) Prefetch P bypasses the full MSHR

(2) B returns and frees an entry

(3) A later load to P allocates that entry

(4) The response matches P: fill L1D and notify r12

Index