Using a User-Level Memory Thread for Correlation Prefetching

Yan Solihin[1] **Jaejin Lee[2]** **Josep Torrellas[1]**

[1] University of Illinois at Urbana-Champaign

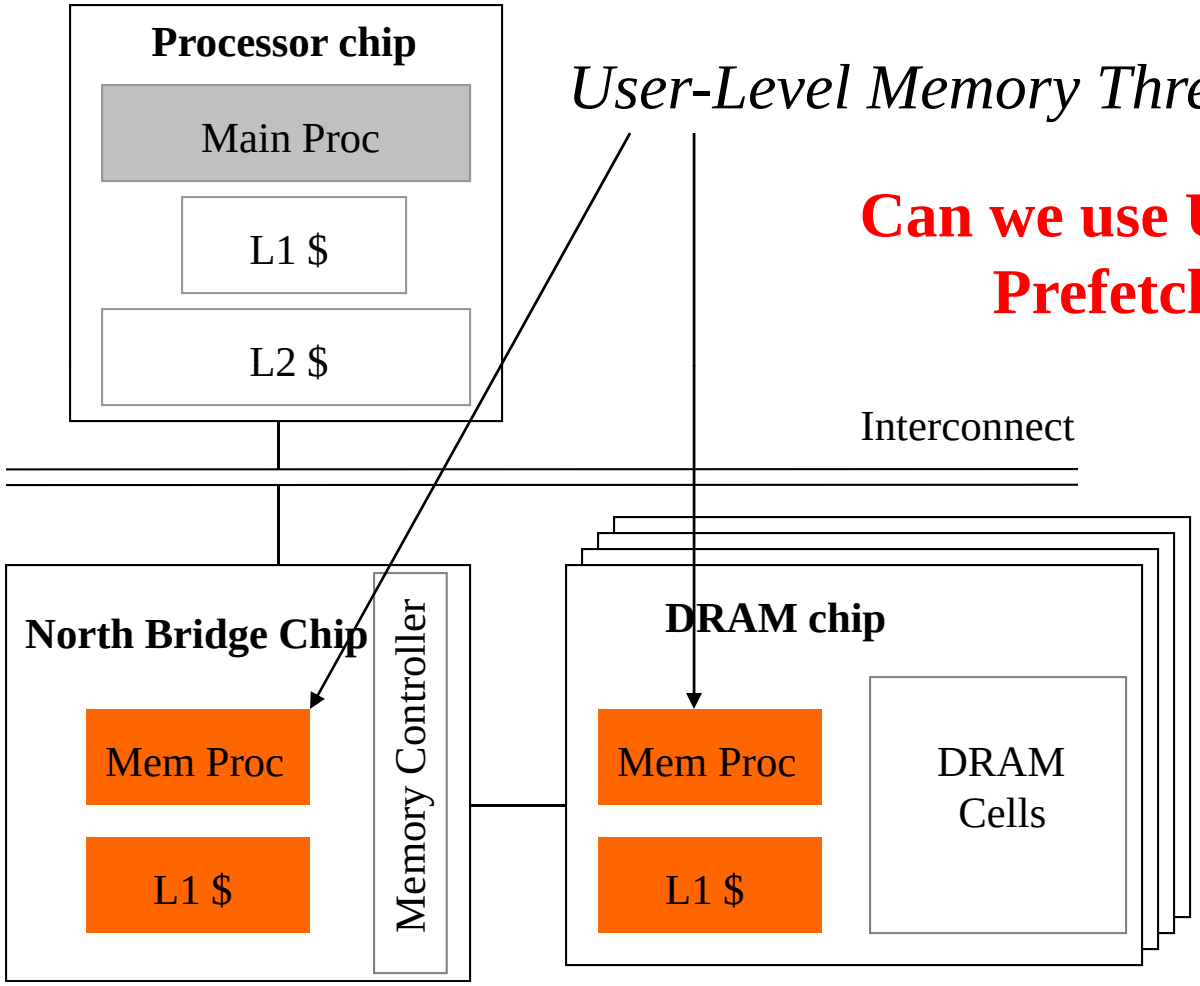
[2] Michigan State University

{solihin,torrellas}@cs.uiuc.edu jlee@cse.msu.edu

Motivation

- Processor-memory speed gap growing
- Memory wall: memory becoming the bottleneck
- Challenges in prefetching irregular apps
- Want to have prefetching:
  - Effective for both irregular and regular apps
  - Customizable to applications
  - No major hardware cost
  - No compiler support

Intelligent Memory

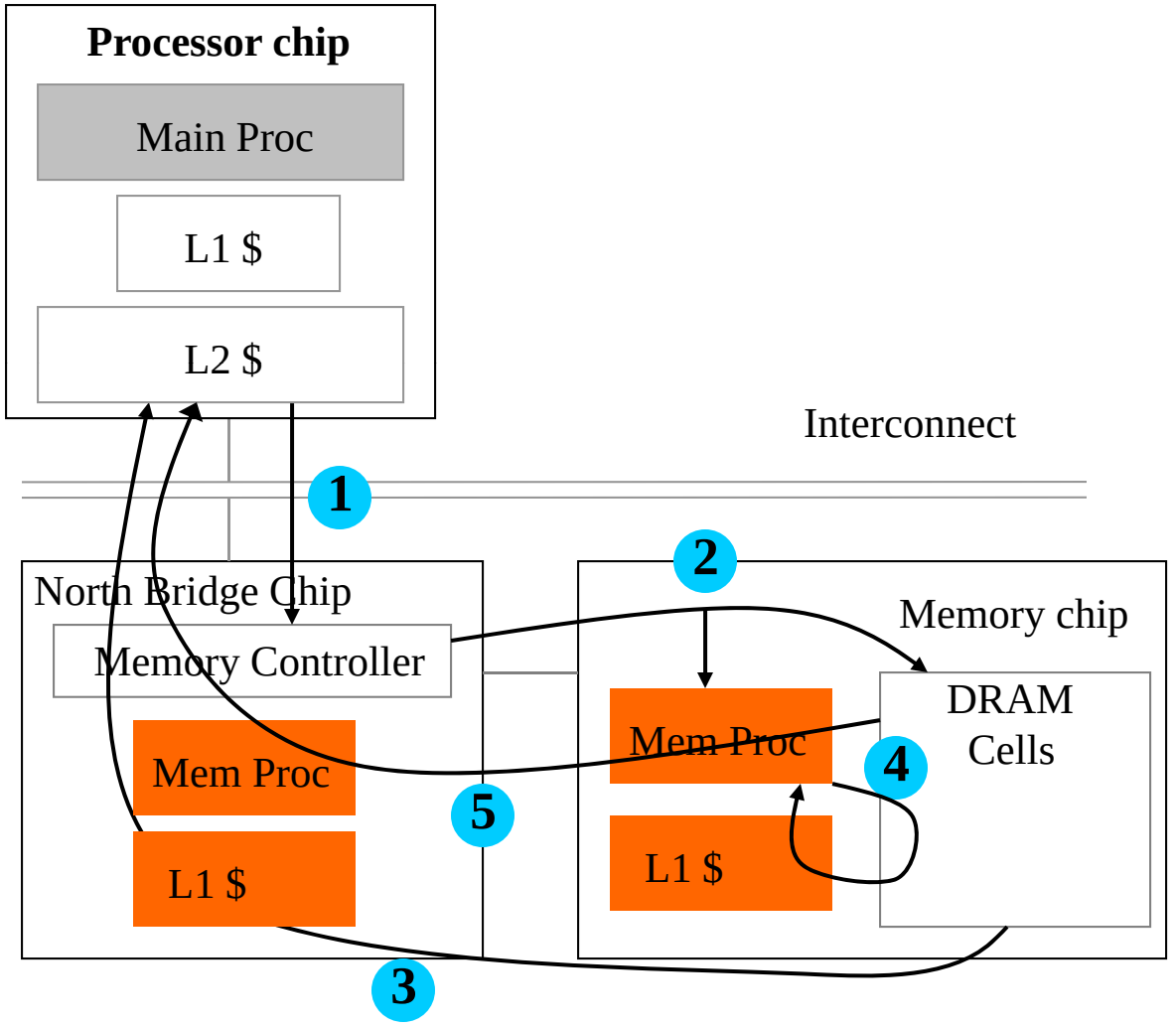


Contribution

Correlation prefetching in software as a ULMT in memory

- **Widely applicable**: prefetches irregular apps
- **Flexible**: prefetching can be customized for applications
- **Inexpensive**: few hardware changes, no compiler support needed
- **Effective**:
  - Conventional prefetching only: **1.22** speedup
  - ULMT only: **1.32** speedup
  - Conventional + ULMT: **1.46** speedup
  - Conventional + ULMT + Customization: **1.53** speedup

Proposed Scheme



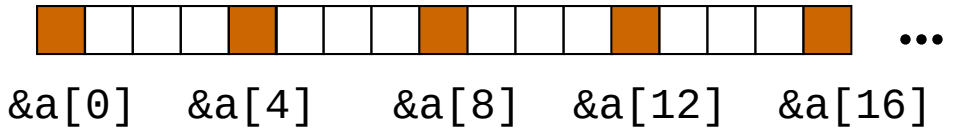
Outline

- Overview of correlation prefetching
- ULMT
- Prefetching algorithms
- Results

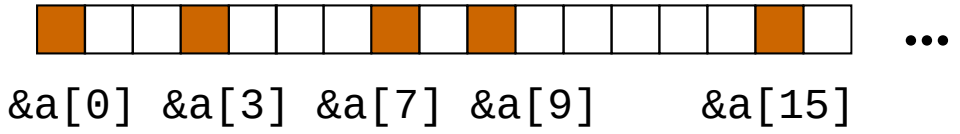
Correlation Prefetching [Joseph&Grunwald,97]

- Records sequences of miss addresses in a **correlation table**
- When the head of a sequence is seen, prefetch the rest
- Effective: any miss patterns that repeat

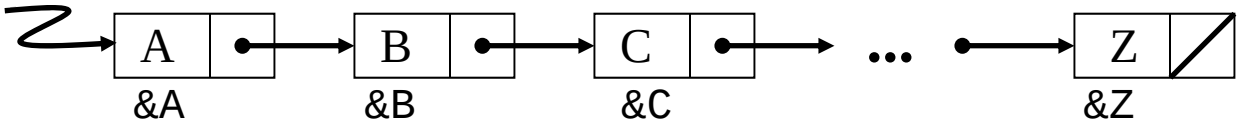
(1) `a[4*(i++)]`



(2) `a[foo(i)]`
`a[b[i]]`



(3)

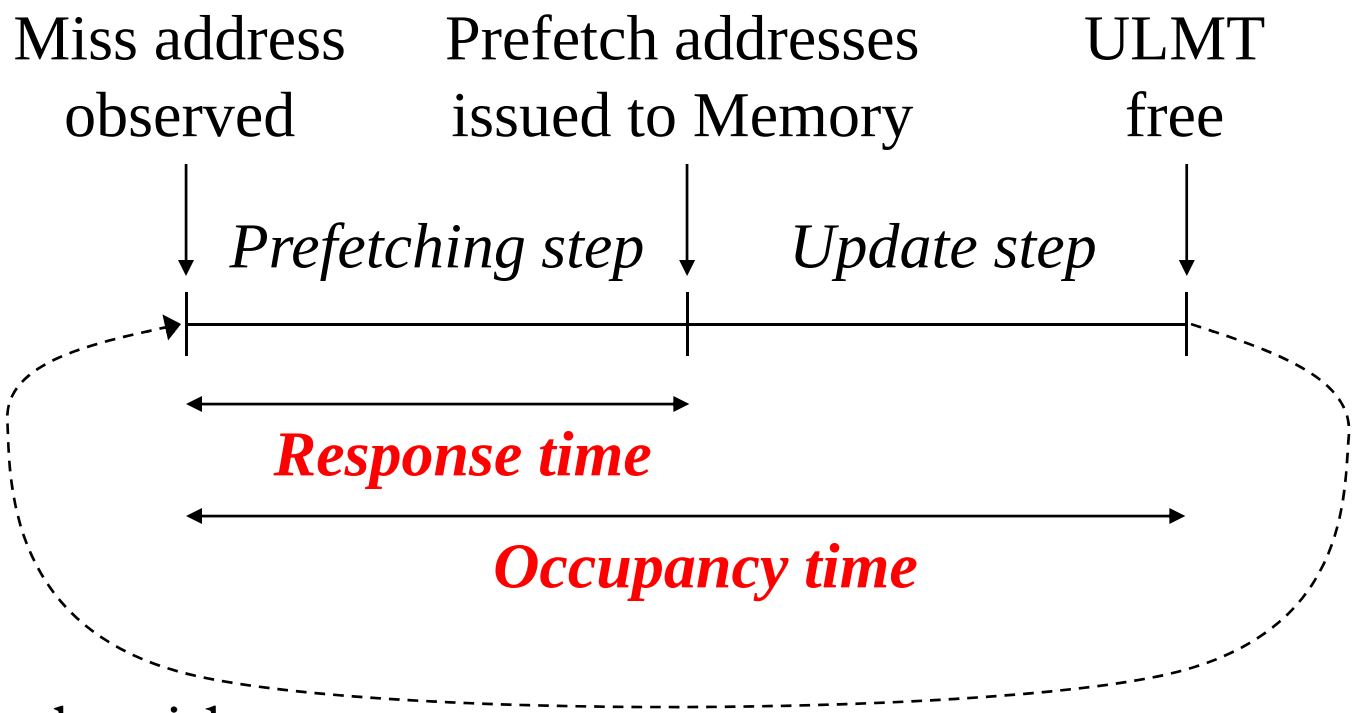


Proposal: ULMT + Correlation Pref

Aspects	Past Correlation Pref	ULMT
Prefetcher	Custom hardware[1-4]	General purpose core
Location	On-chip at L1[1,2,4] or in Memory (buffering [3])	Memory (prefetching to L2)
Table cost	High (1 – 7.6 MB SRAM) [1-4]	Low (DRAM), dynamically allocated
Multiple apps	Cross-pollution [1-4]	One instance per app
Custom & control	No [1-4]	Yes, by application/OS

[1] Joseph&Grunwald, ISCA 97
[2] Charney&Reeves, TR 95
[3] Alexander&Kedem, HPCA 96
[4] Lai, Fide, & Falsafi, ISCA 01

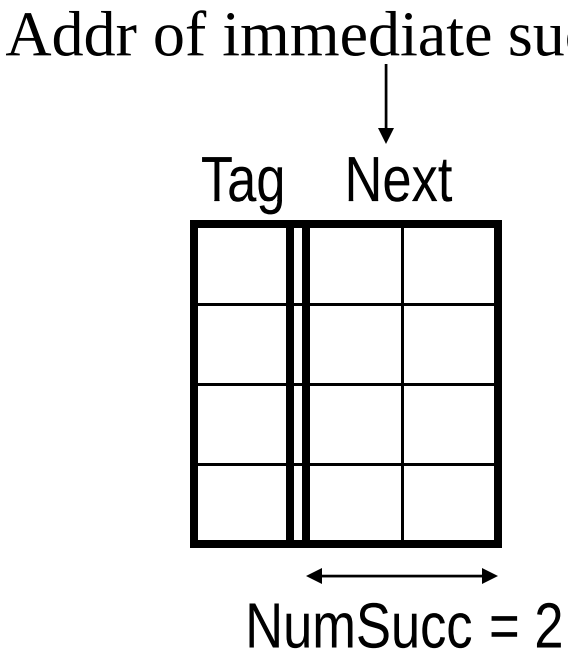
Timeline of ULMT Prefetching



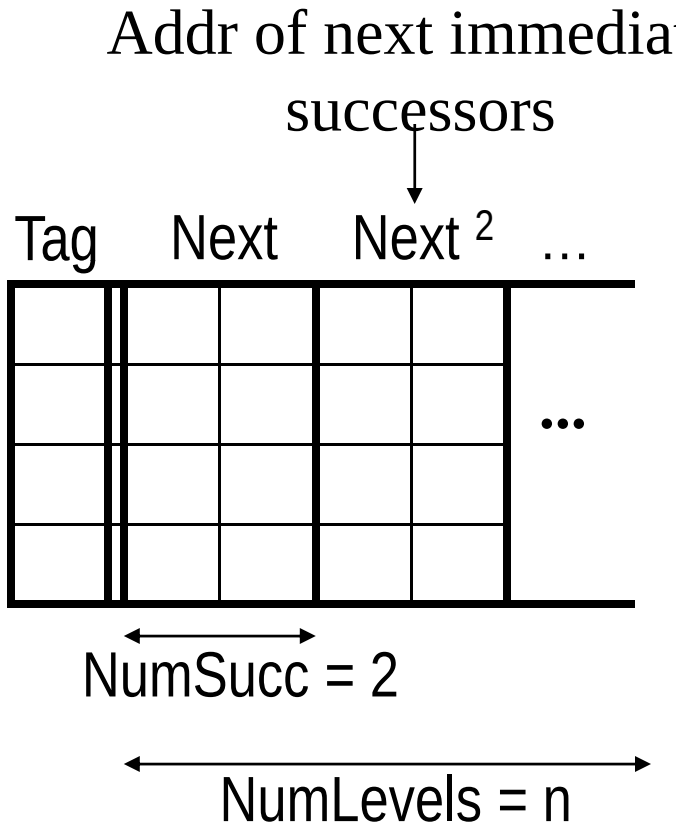
- Ideal algorithm:
  - lowest response time
  - occupancy time < time between misses

Correlation Table

Basic Organization [JG97]



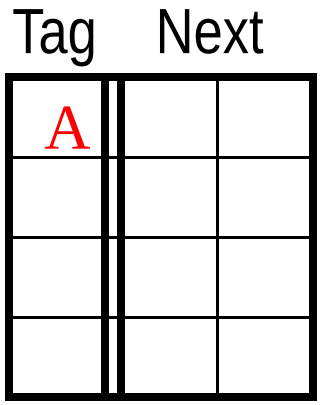
Our Advanced Organization



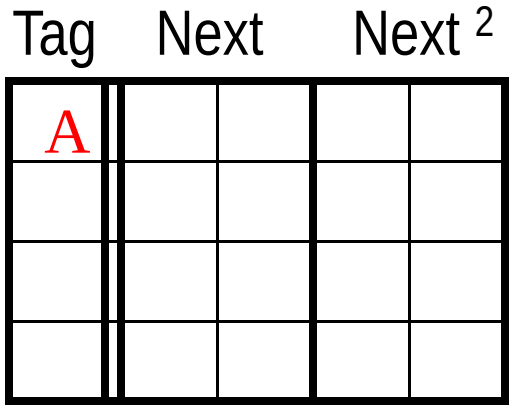
[JG97] Joseph&Grunwald, ISCA 97

Update Step

Basic Organization[JG97]



Our Advanced Organization

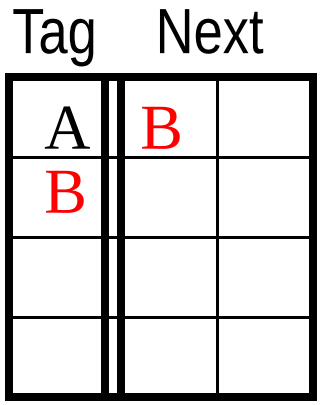


Current miss

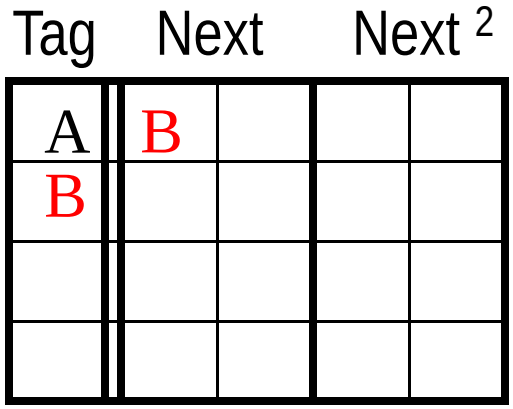
A, B, C, A, D, C, ...

Update Step

Basic Organization[JG97]



Our Advanced Organization



Current miss

A, B, C, A, D, C, ...

Update Step

Basic Organization[JG97]

Tag	Next	
A	B	
B	C	
C		

Our Advanced Organization

Tag	Next		Next 2	
A	B		C	
B	C			
C				

Current miss

A, B, C, A, D, C, ...

Update Step

Basic Organization[JG97]

Tag	Next	
A	B	
B	C	
C	A	

Our Advanced Organization

Tag	Next		Next 2	
A	B		C	
B	C		A	
C	A			

Current miss

A, B, C, A, D, C, ...

Update Step

Basic Organization[JG97]

Tag	Next	
A	B	D
B	C	
C	A	
D	C	

Our Advanced Organization

Tag	Next		Next ^{2}	
A	B	D	C	
B	C		A	
C	A		D	
D	C			

Current miss

A, B, C, A, D, C, ...

Update Step

Basic Organization[JG97]

Tag	Next	
A	B	D
B	C	
C	A	
D	C	

Our Advanced Organization

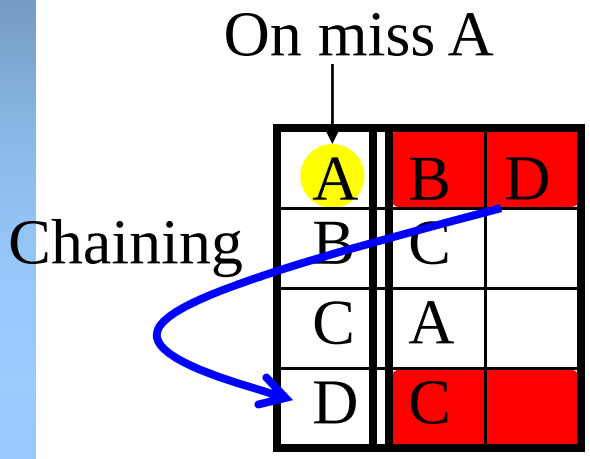
Tag	Next		Next ^{2}	
A	B	D	C	
B	C		A	
C	A		D	
D	C			

Current miss

A, B, C, A, D, C, ...

Prefetching Step

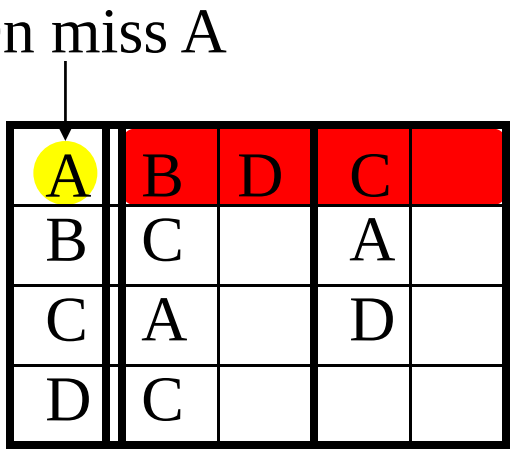
Basic Organization[JG97]



Basic: 1 miss ⇒ next misses
- no far ahead prefetching
- low coverage and late prefetches

Basic + Chaining:
- high response time

Our Advanced Organization



Advanced: 1 miss ⇒ next, next^2, …
- far ahead prefetching
- high coverage
- more timely prefetches
- low response time
- more accurate

Advanced Improves Accuracy

Basic Organization[JG97]

A	B	
B	C	E
C	D	
D		
E	F	

Our Advanced Organization

A	B		C		D	
B	C	E	D	F		
C	D					
D						
E	F					

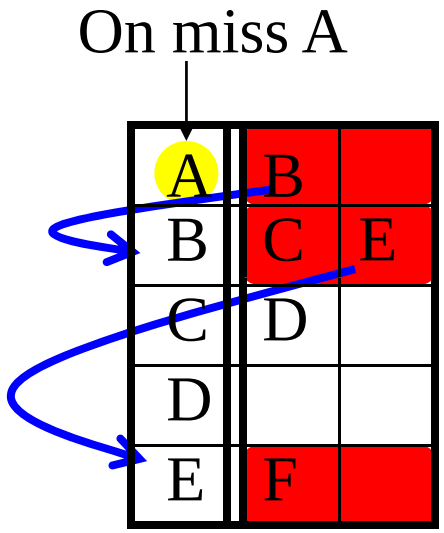
On miss

A, B, C, D, …, A, B, C, D, …, A, B, C, D, …, B, E, F, …, A

⇒ ideally prefetch B,C,D

Advanced Improves Accuracy

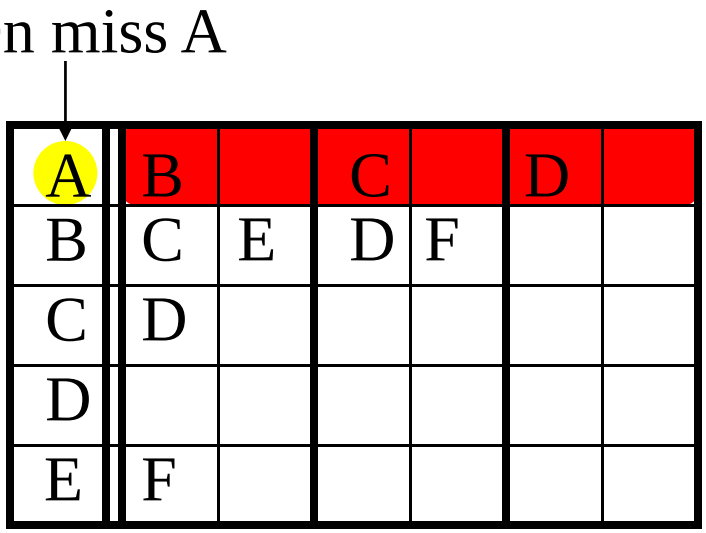
Basic Organization[JG97]



B prefetched (Basic)

B, C, E, F prefetched (Basic+Chaining)

Our Advanced Organization



B, C, D prefetched

On miss

A, B, C, D, …, A, B, C, D, …, A, B, C, D, …, B, E, F, …, A

⇒ ideally prefetch B,C,D

Simulation Environment

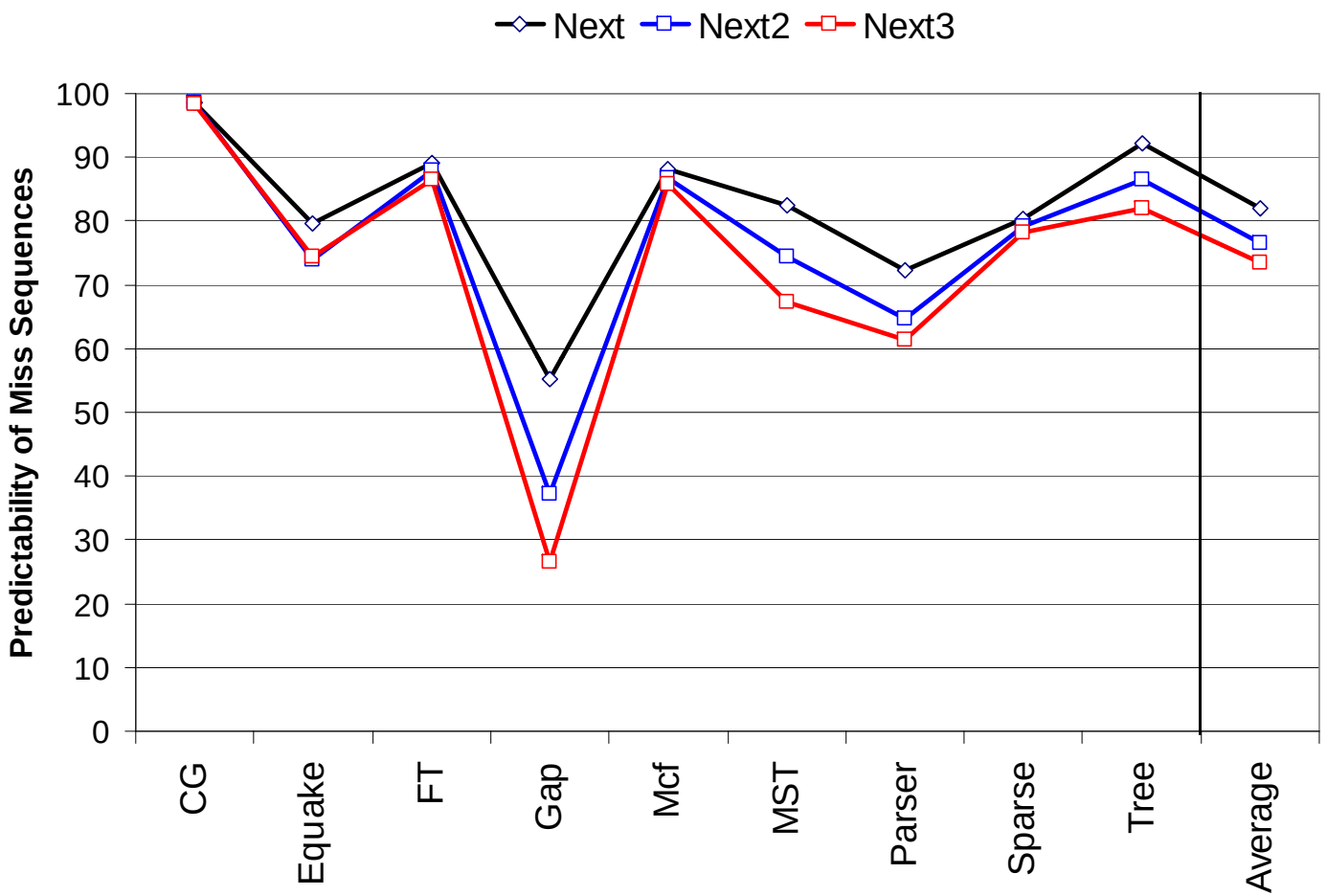
- Main processor:
  - 1.6 GHz, 6-issue out-of-order
  - L1: 2-way 16 KB; L2: 4-way 512 KB
  - Mem: 152 ns round-trip (RT)
- Memory processor:
  - 800 MHz, 2-issue out-of-order, integer only
  - L1: 2-way 32 KB
  - Mem: 63 ns RT (in NorthBridge), 35 ns cycle RT (in DRAM)
- Bus bandwidth: 3.2 GB/sec
- Correlation table: app-specific (64K entries, 3 levels, 2 successors)
- Applications: memory intensive, mostly irregular
  - Specint2000, Specfp2000, NAS, Olden

Predictability of Miss Sequences

Next Next^3

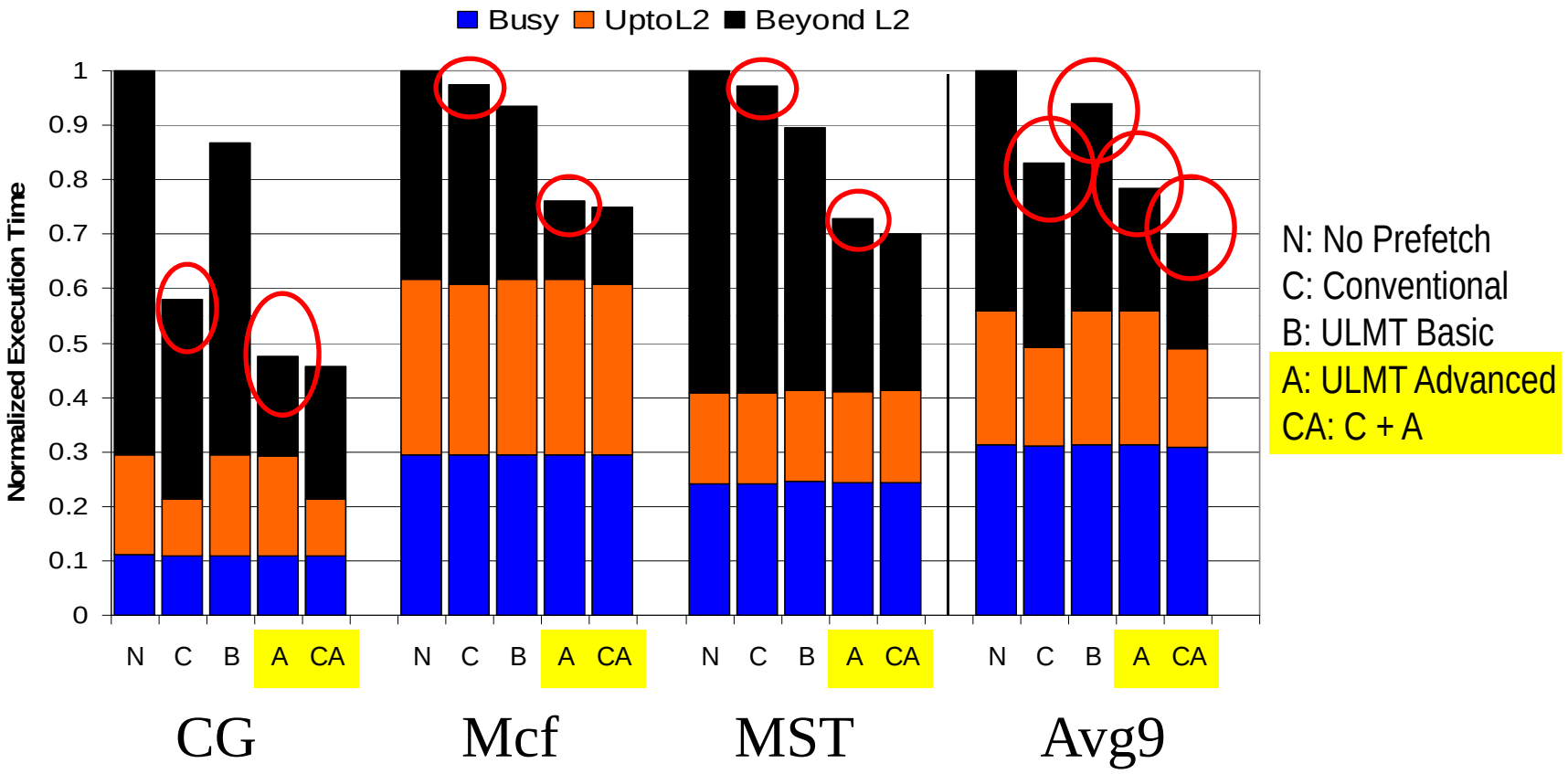
A, **B**, C, D

Next^2



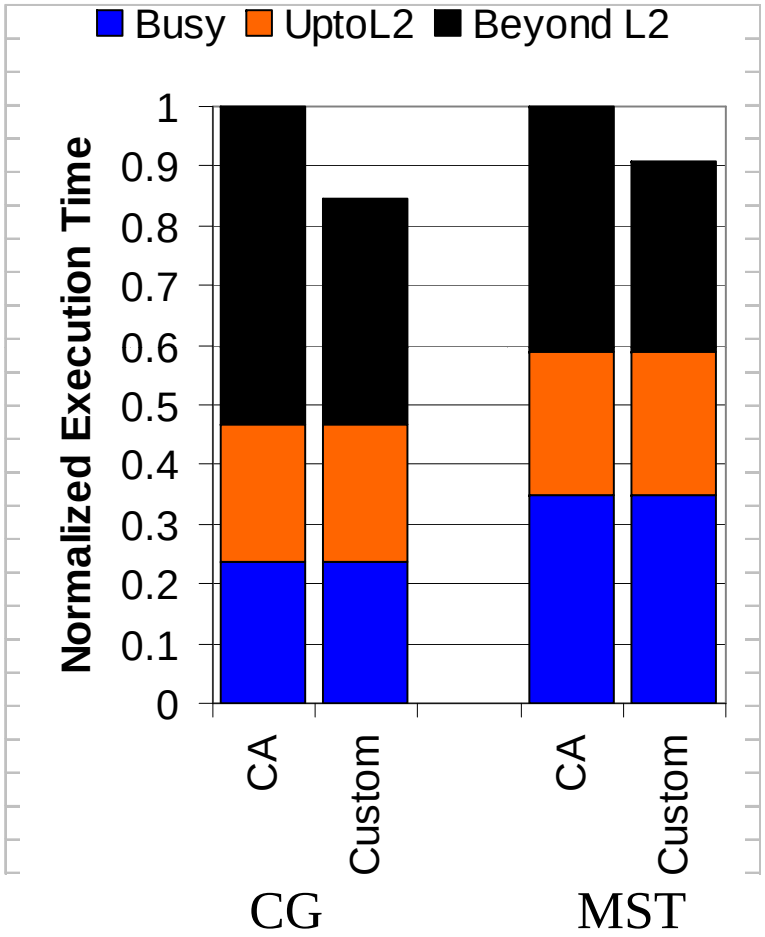
- Miss sequences are predictable

Execution Time (Mem Proc in DRAM)



- Advanced memory prefetching works

Customization Delivers Further Gains



Types of Customization:

- CG: ULMT includes n-stream stride prefetcher
- MST: ULMT prefetches up to 4 levels of successors (high predictability)

Simple customization successful in some apps

Other Results

- More on predictability
- Timing analysis
- Prefetching effectiveness
- Bus utilization
- Processor in North Bridge vs. DRAM
  - Comparable speedups
- OS issues

Conclusions

- Simple Intelligent Memory: performance ↑ significantly
- Correlation prefetching in ULMT is effective
  - Conventional only: 1.22 speedup
  - ULMT only: 1.32 speedup
  - ULMT + Conventional: 1.46 speedup
  - ULMT + Conventional + Customization: 1.53 speedup
- Flexible systems: customizable
- Few hardware changes
- No compiler support needed

Using a User-Level Memory Thread for Correlation Prefetching

Yan Solihin[1] **Jaejin Lee[2]** **Josep Torrellas[1]**

[1] University of Illinois at Urbana-Champaign

[2] Michigan State University

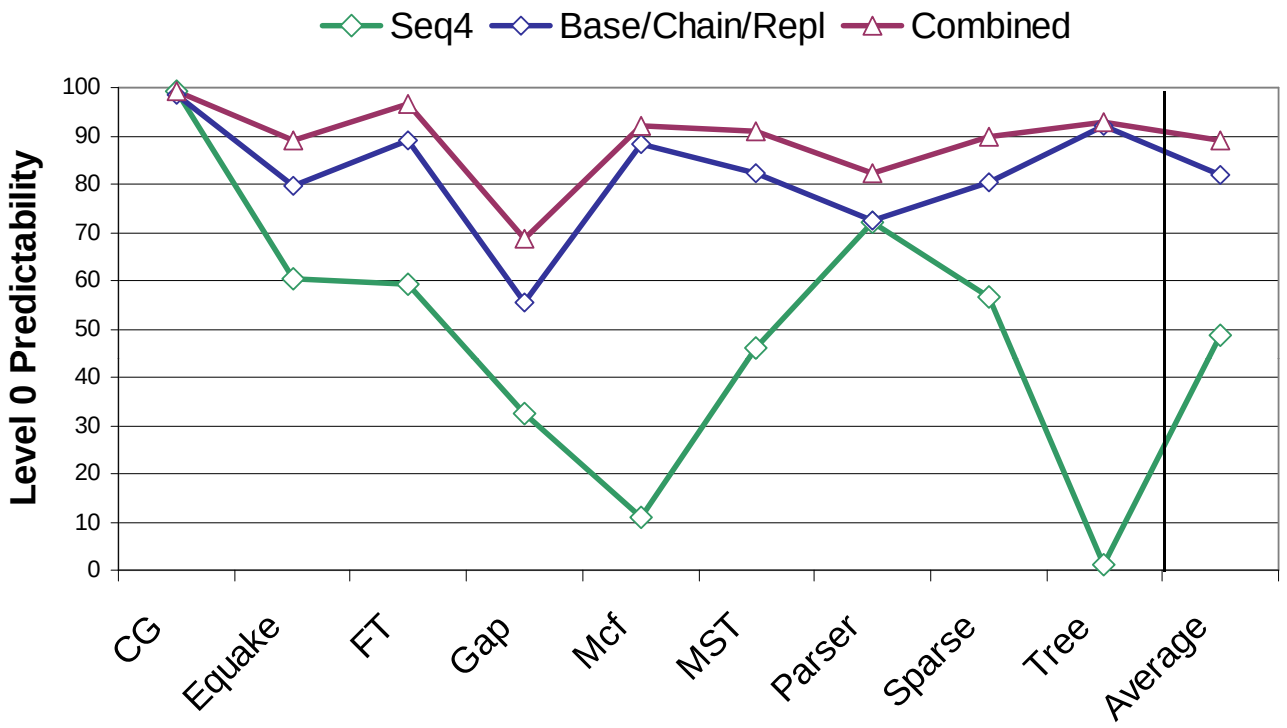
Slide Map

- (example)(vs. past work)
- (architecture and mechanism)
- (timeline)
- (update and prefetching steps)
- Accuracy of Chaining vs. Advanced
- Results (params) (predictability) (exectime) (custom)
- Size of correlation table
- Additional Results (L0pred) (Chain vs. Adv) (time between misses) (response time) (MC) (Effectiveness) (Bus)
- OS Issues (multitasking) (page remap) (sharing)
- Multichip DRAM
- (Related work)

Correlation Table Size

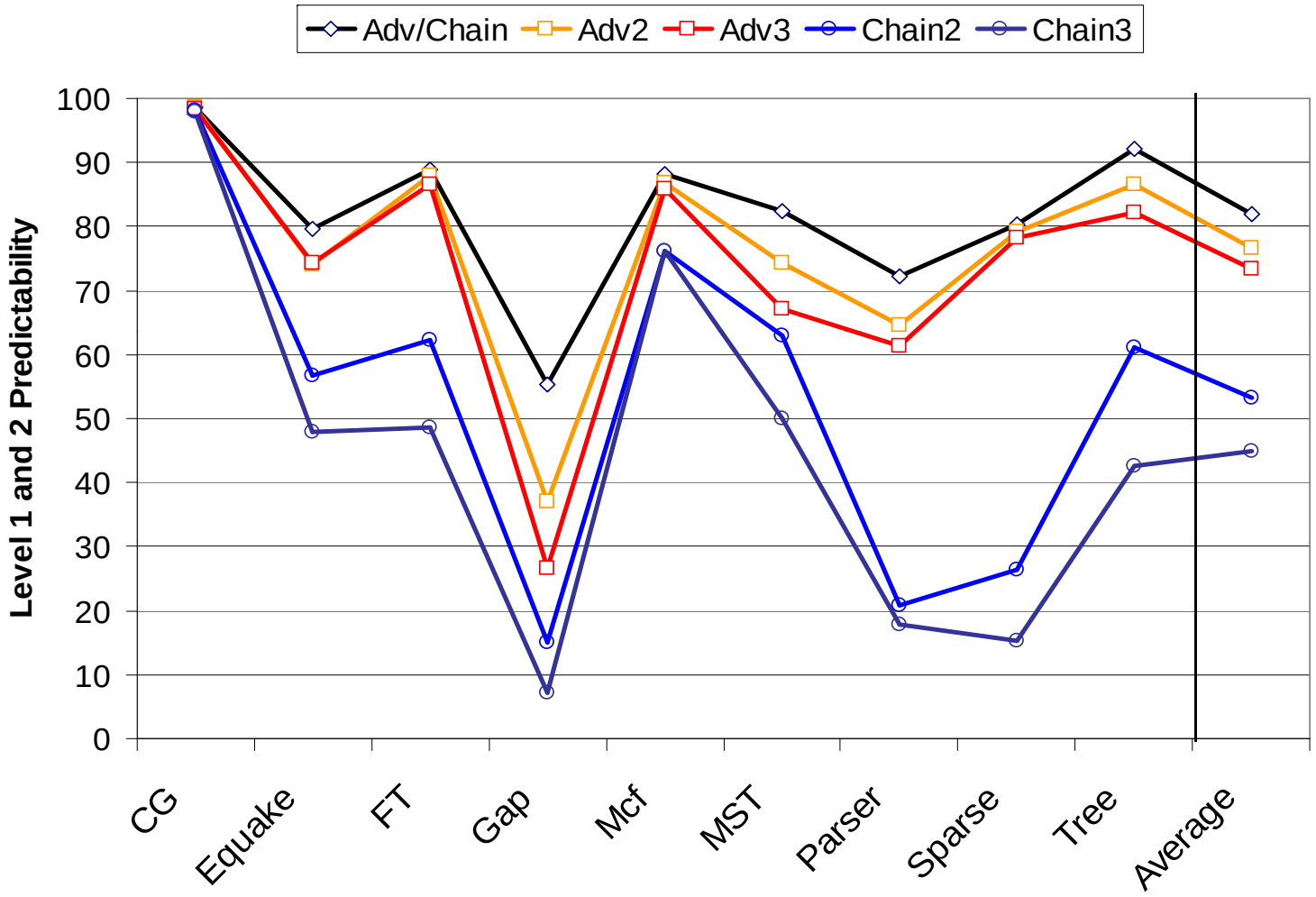
App	Basic (MB)	Advanced (MB)
CG	1.3	1.8
Equake	2.5	3.5
FT	5.0	7.0
Gap	2.5	3.5
Mcf	0.6	0.9
MST	5.0	7.0
Parser	2.5	3.5
Sparse	5.0	7.0
Tree	0.2	0.2
Average	**2.7**	**3.8**

Predictability of Miss Addresses



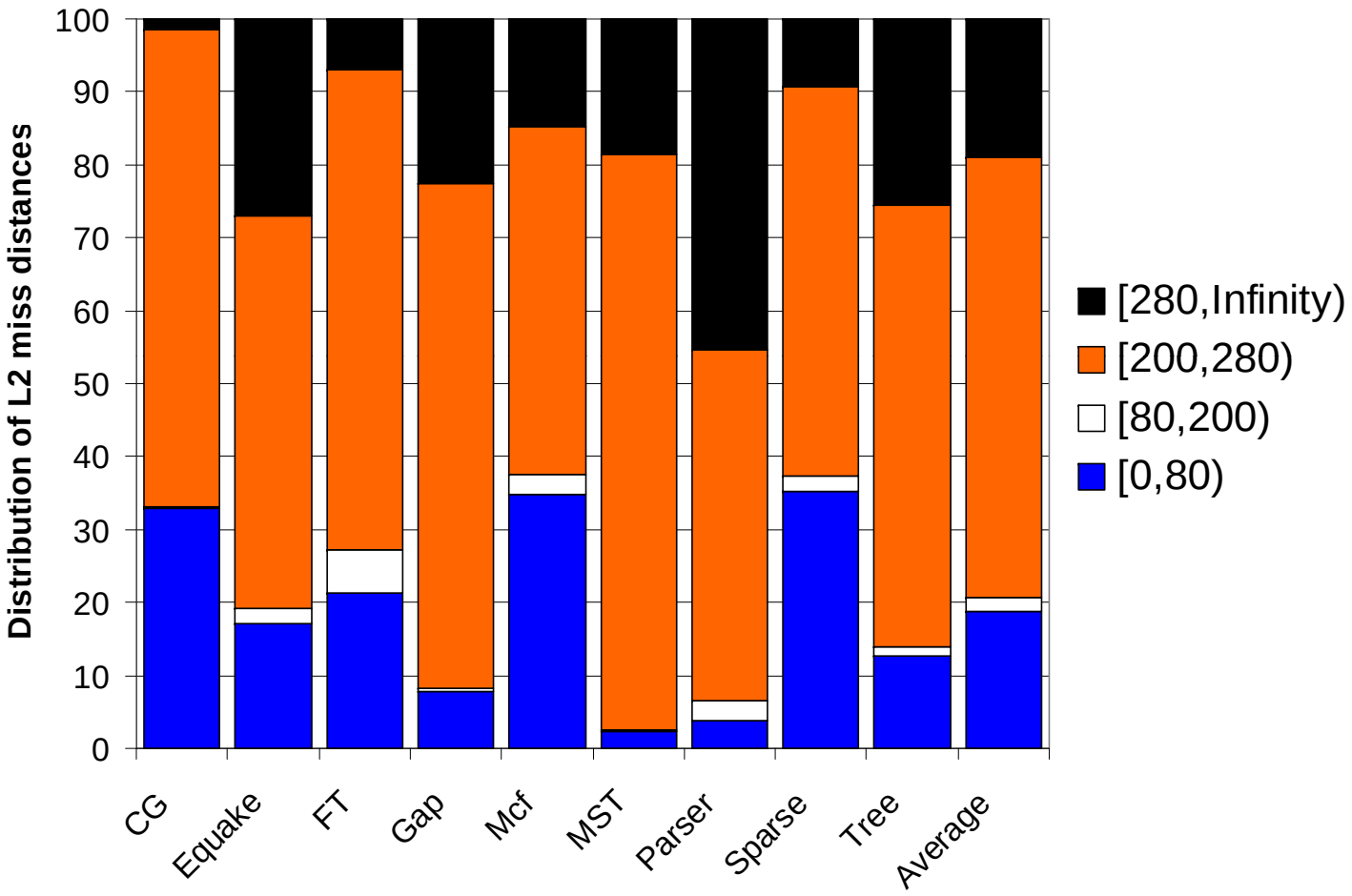
- Miss patterns are application-specific
- Base is able to capture most of the patterns

Advanced vs. Basic+Chaining



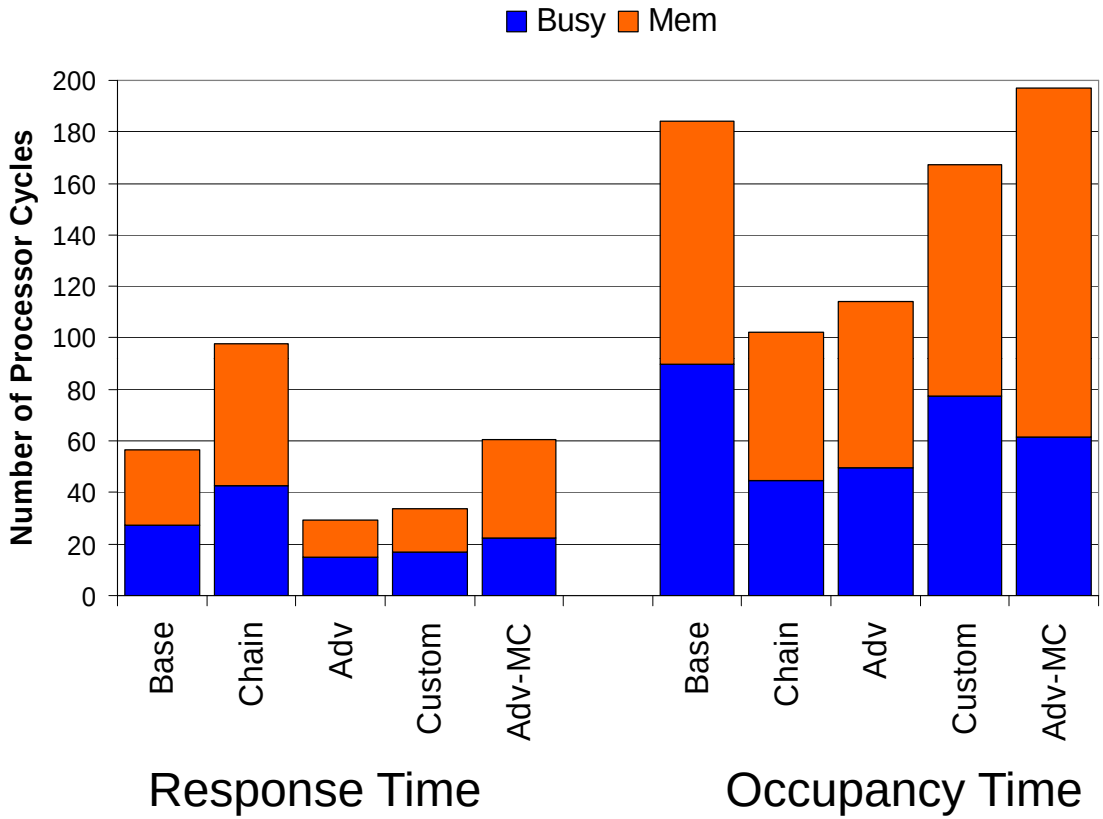
advanced is superior in all cases

Cycles Between Misses



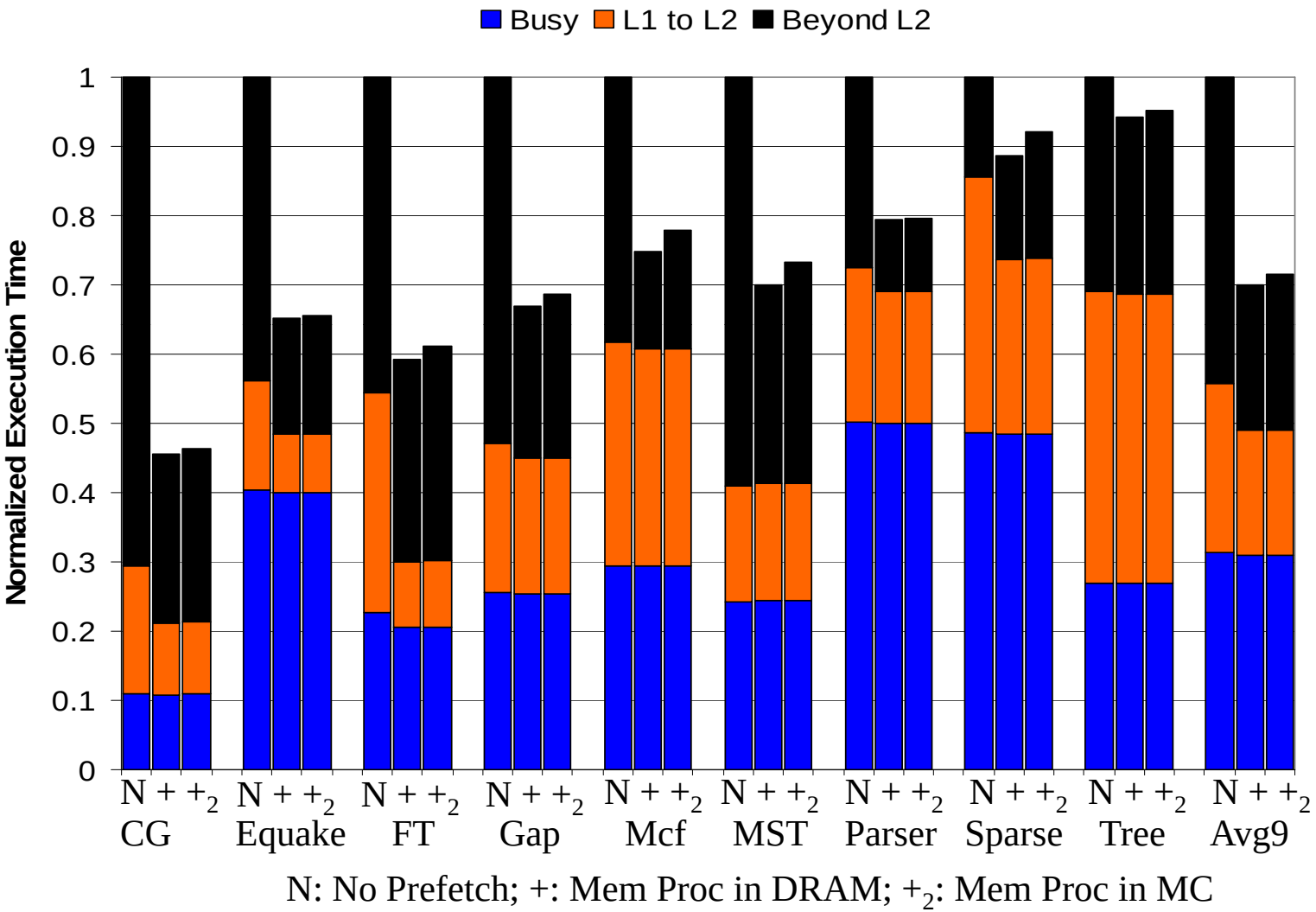
- Occupancy time of prefetch thread must be < 200 cycles

ULMT Response and Occupancy Time



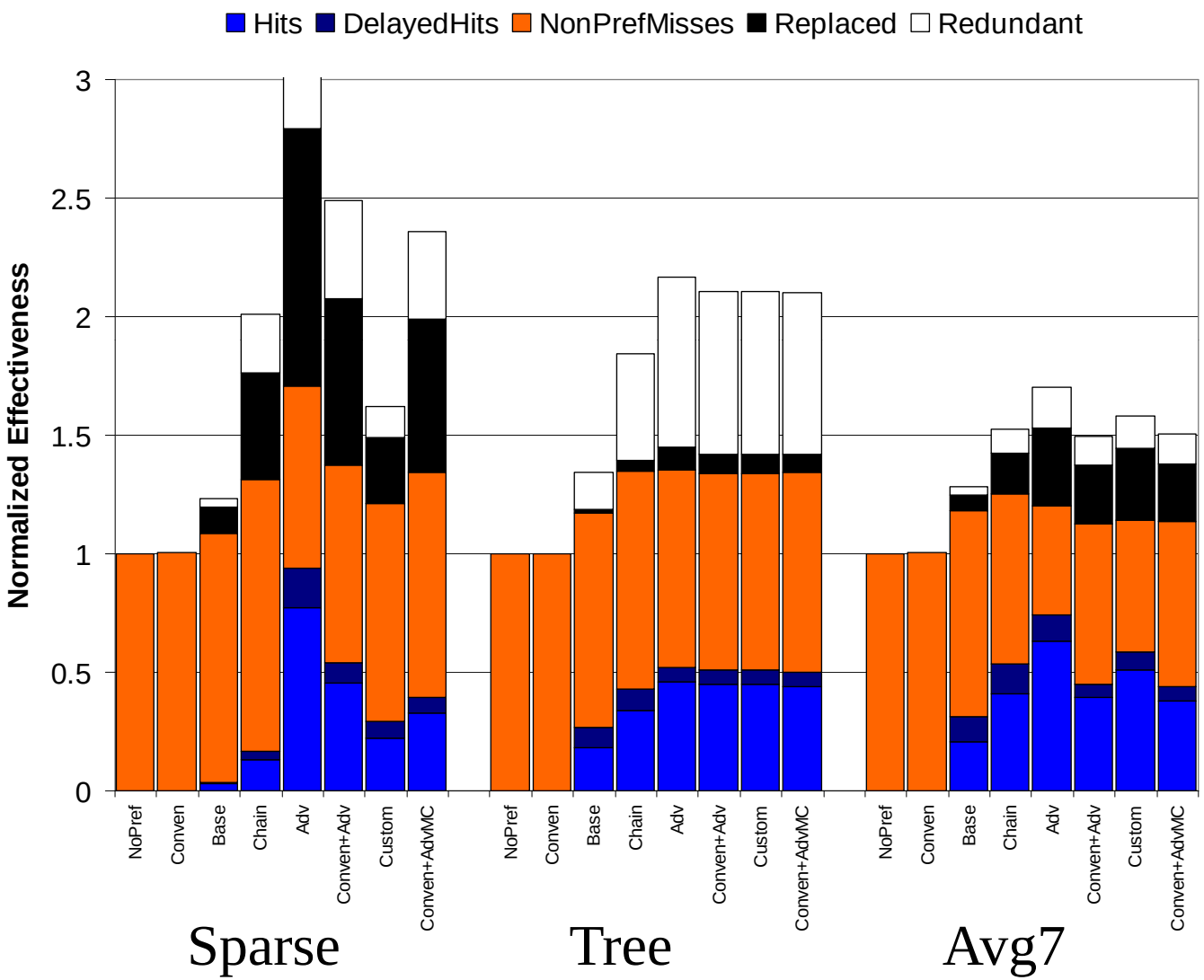
- All algo feasible: Occupancy Time < 200 cycles
- Advanced (≡ Repl) has the best Response Time

DRAM vs. Mem Controller Chip

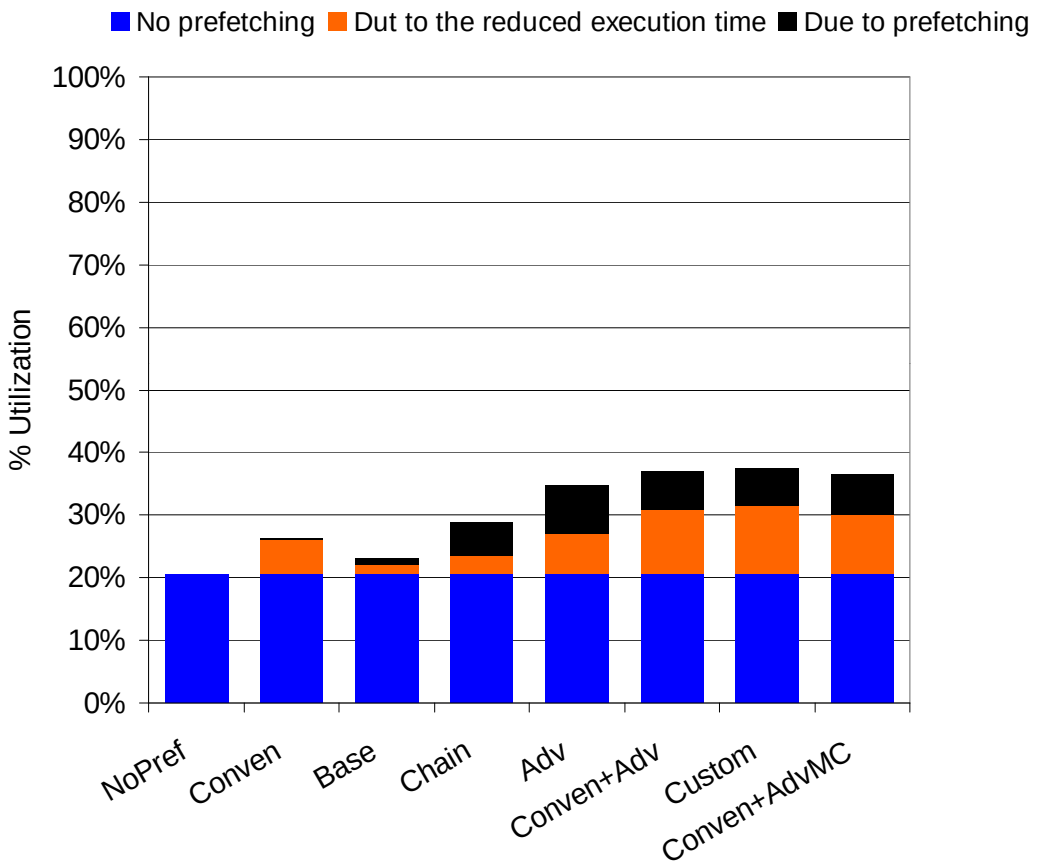


N: No Prefetch; +: Mem Proc in DRAM; $+_2$: Mem Proc in MC

Prefetching Effectiveness



Bus Utilization



Advanced: avg increase of 8%

Multitasking Environment

- ULMT is flexible (vs. hardware correlation prefetching)
- Shared correlation table:
  - \+ Smaller table
  - \- Interference or
  - \- Cold start at every context switch
- Private correlation table: one instance per application
  - \+ No interference
  - \- Larger space requirement, but solutions available:
    - Still a small portion of main memory
    - OS imposes quota on total table size through `malloc( )`
    - Dynamic resizing eliminates table fragmentation

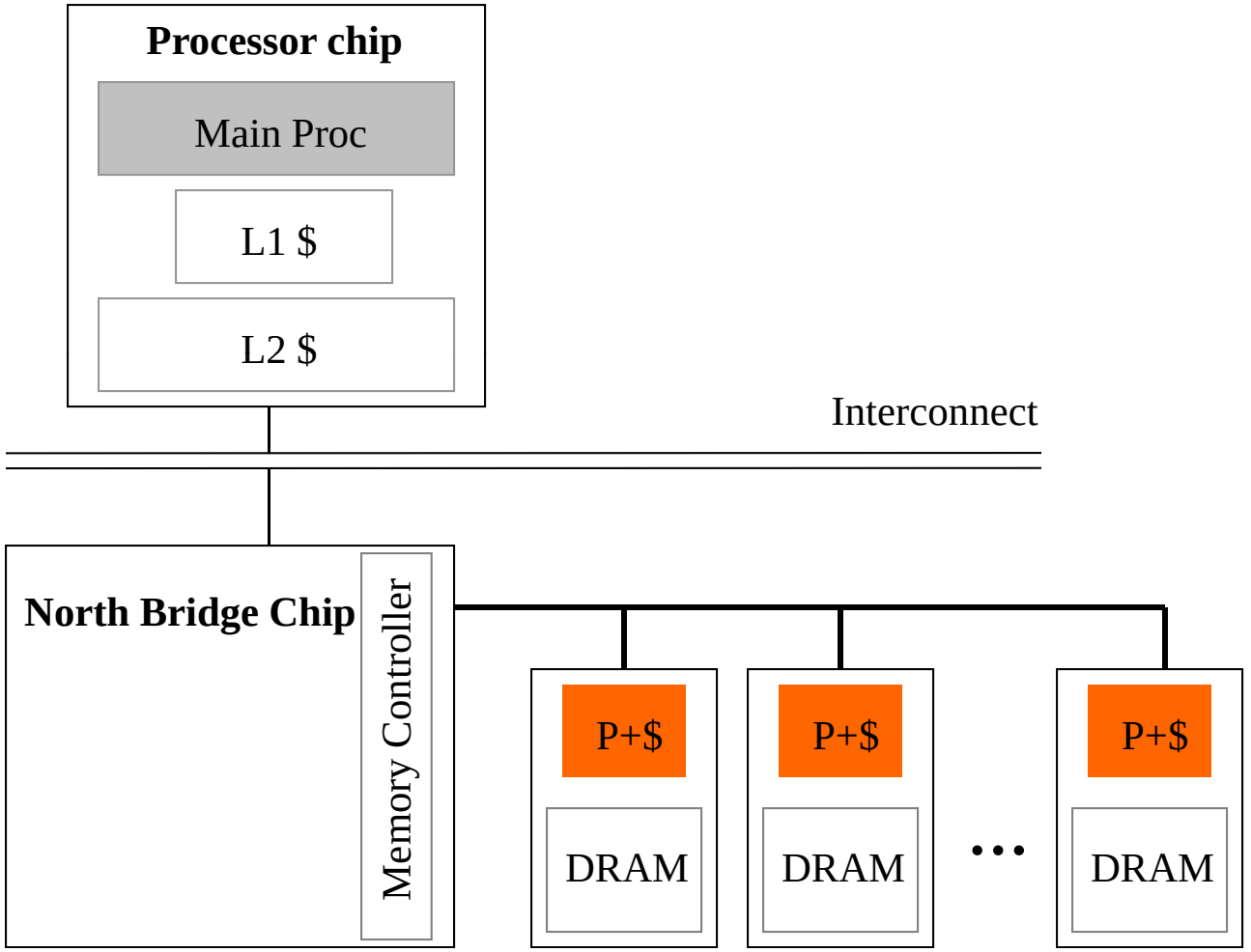
Handling Page Mapping Changes

- ULMT is flexible (vs. hardware correlation prefetching):
- Software solution:
  - OS informs page mapping changes of the main program
  - ULMT remaps or invalidates entries of the old page and (for 8KB-page: overhead ≈ 4 µs - 8 µs)
  - Only tags are changed, successors may be incorrect
- Hardware solution:
  - A small cache in MC stores invalid pages for each process
  - Prefetches are filtered by this cache
  - If cache overflows, prefetches may be incorrect
- Lazy solution: ignore page mapping changes

OS Issues

- Scheduling:
  - An instance of ULMT is attached to the main program
  - Scheduled and context switched together
  - States are equivalent
- Sharing and protection
  - ULMT and main program do not share data and instruction
  - Main program is protected: ULMT only **reads** physical addresses
  - ULMT is protected: it has its own address space
  - OS provides sys calls for main program to control life of ULMT

Multi-chip DRAM



Related Work

- Memory-side prefetching
  - Yang&Lebeck (ICS'00), Hughes (MS Thesis'00), Cooksey et.al. (Content-based prefetcher, WIMS'00)
- Correlation-based prefetching
  - Baer ('96), Pomerene (Patent '89), Joseph&Grunwald (ISCA'97), Charney&Reeves (TR'95), Alexander&Kedem (HPCA'96), Sherwood et.al. (Micro'00), Lai et.al. (ISCA'01)
- Helper thread
  - Chappel et.al. (SSMT, ISCA'99), Intel's SP (ISCA'01, HPCA'02), Collins et.al. (MICRO'01), Roth et.al.'s DDMT (HCPA'01),
- Prefetching for LDS
  - Luk&Mowry (ASPLOS'96), Roth et.al. (ASPLOS'98, ISCA'99,), Choi et.al. (PACT'01), Hughes (MS Thesis'00), Yang&Lebeck (ICS'00),
- Other
  - Carter et.al. (HPCA'99), Burger et.al. (DataScalar, ISCA'97).