CS533: Prefetching

Josep Torrellas
(Tom Shull Presenting)

University of Illinois in Urbana-Champaign

February 10, 2015

Using a User-Level Memory Thread for Correlation Prefetching

- Main Idea: add a processor to either the memory controller or DRAM module to prefetch data into the L2 cache
- By adding a processor to perform *Memory-side* prefetching, one can implement application-specific prefetching schemes

Using a User-Level Memory Thread for Correlation Prefetching

- Main Idea: add a processor to either the memory controller or DRAM module to prefetch data into the L2 cache
- By adding a processor to perform *Memory-side* prefetching, one can implement application-specific prefetching schemes
- Advantages of Memory-side prefetching:

Using a User-Level Memory Thread for Correlation Prefetching

- Main Idea: add a processor to either the memory controller or DRAM module to prefetch data into the L2 cache
- By adding a processor to perform *Memory-side* prefetching, one can implement application-specific prefetching schemes
- Advantages of Memory-side prefetching:
  - minimal changes need to be made to processor
  - off critical-path for cache hits
  - can store necessary state in memory

Design Considerations

- Why fetch to L2, not L1 cache?

Design Considerations

- Why fetch to L2, not L1 cache?
    - Requires less changes to the processor
    - L2 cache is bigger so evicting relevant data (cache conflicts) are not as big of a concern
    - prefetcher can be slower since L2 cache accesses are more rare, yet still provides important performance improvements
- Why embed processor to prefetch instead of custom hardware?

Correlation Prefetching

- Idea: “learn” sequences of misses and prevent them from happening in the future
- When a miss occurs, record subsequent misses in table
- The next time a miss to the same address occurs, prefetch misses recorded from previous executions
- Effective for handling irregular accesses to memory

Correlation Prefetching Example

Code Sequence: `A`

Tag	Succ_L1	
A		

Correlation Prefetching Example

Code Sequence: `A,B`

Tag	Succ_L1	
A	B	
B		

Correlation Prefetching Example

Code Sequence: `A,B,C`

Tag	Succ_L1	
A	B	
B	C	
C		

Correlation Prefetching Example

Code Sequence: `A,B,C,A`

Tag	Succ_L1	
A	B	
B	C	
C	A	

Correlation Prefetching Example

Code Sequence: `A,B,C,A,D`

Tag	Succ_L1	
A	B	D
B	C	
C	A	
D		

Correlation Prefetching Example

Code Sequence: `A,B,C,A,D,C`

Tag	Succ_L1	
A	B	D
B	C	
C	A	
D	C	

Correlation Prefetching Issues

This has been implemented in hardware before, but there are issues:

Correlation Prefetching Issues

This has been implemented in hardware before, but there are issues:

- Table is too small. Can easily have bad information
- Still have misses

Improvement: Chain together different Tag entries

- Still can have bad information
- Takes long time to determine what to prefetch

Advanced Correlation Prefetching

Insight: Since prefetching processor is storing its state is main memory, one can create large tables without significant overheads.

- Instead of chaining, store multiple levels for each tag
- Maintain pointers to lessen overhead of recording miss addresses to multiple locations in the table
- On miss, prefetch all addresses associated with a tag

Advanced Correlation Prefetching Example

Code Sequence: `A`

Tag	Succ_L1		Succ_L2	
A				

Advanced Correlation Prefetching Example

Code Sequence: `A,B`

Tag	Succ_L1		Succ_L2	
A	B			
B				

Advanced Correlation Prefetching Example

Code Sequence: `A,B,C`

Tag	Succ_L1		Succ_L2	
A	B		C	
B	C			
C				

Advanced Correlation Prefetching Example

Code Sequence: `A,B,C,A`

Tag	Succ_L1		Succ_L2	
A	B		C	
B	C		A	
C	A			

Advanced Correlation Prefetching Example

Code Sequence: `A,B,C,A,D`

Tag	Succ_L1		Succ_L2	
A	B	D	C	
B	C		A	
C	A		D	
D				

Advanced Correlation Prefetching Example

Code Sequence: `A,B,C,A,D,C`

Tag	Succ_L1		Succ_L2	
A	B	D	C	
B	C		A	
C	A		D	
D	C			

Configuration

Prefetching Algorithm	Implementation	Name	Parameter Values
Base	Software in memory as ULMT	Base	NumSucc = 4, Assoc=4
Chain		Chain	NumSucc = 2, Assoc=2, NumLevels=3
Replicated		Repl	NumSucc = 2, Assoc=2, NumLevels=3
Sequential 1-Streams		Seq1	NumSucc = 1, NumPref=6
Sequential 4-Streams		Seq4	NumSucc = 4, NumPref=6
Sequential 4-Streams	Hardware in L1 of main processor	Conven4	NumSucc = 4, NumPref=6

- *NumRows* is large enough such that less than 5% of insertions replace an existing entry

Performance Considerations

Issues to consider when evaluating performance

Performance Considerations

Issues to consider when evaluating performance

- Coverage
- Response Time
- Occupancy Time
- Cache Miss Frequency
- Comparison with Simplier Prefetching Strategies
- Main Memory Bus Utilization

Coverage

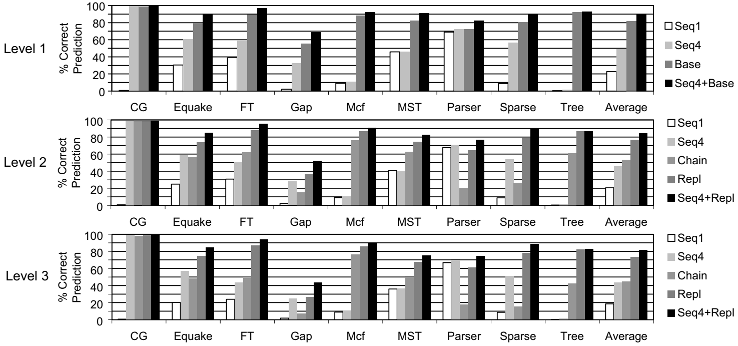
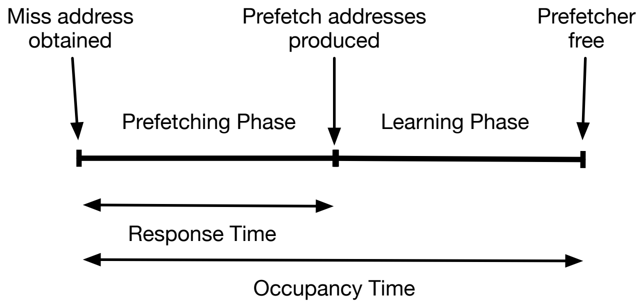


Figure 5. Fraction of L2 cache misses that are correctly predicted by different algorithms for different levels of successors.

Response/Occupancy Time Considerations



Ideal algorithm:

- lowest reponse time
- occupancy time $<$ time between misses

Cache Miss Frequency

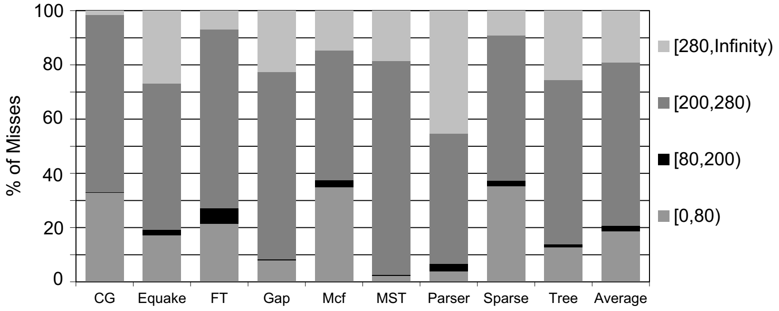


Figure 6. Characterizing the time between L2 misses.

Response/Occupancy Time

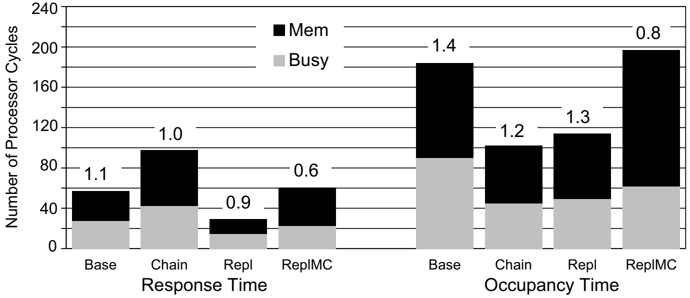


Figure 10. Average response and occupancy time of different ULMT algorithms in main-processor cycles.

Execution Time

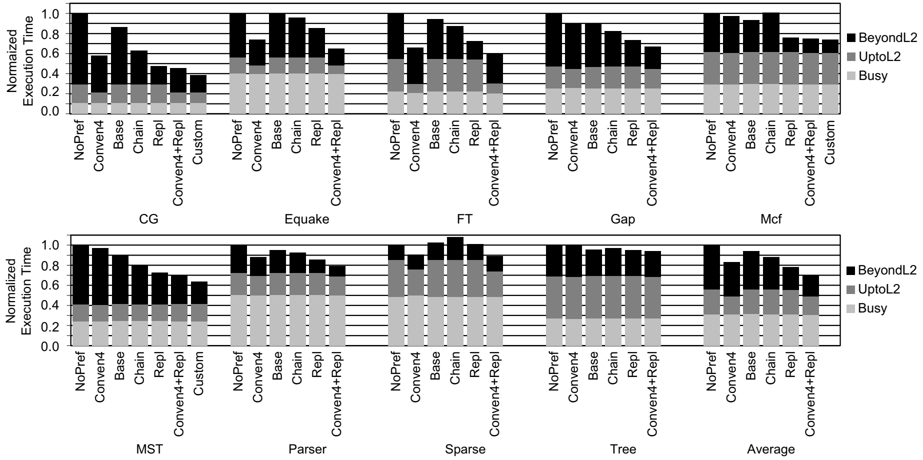


Figure 7. Execution time of the applications with different prefetching algorithms.

Breakdown of L2 Accesses

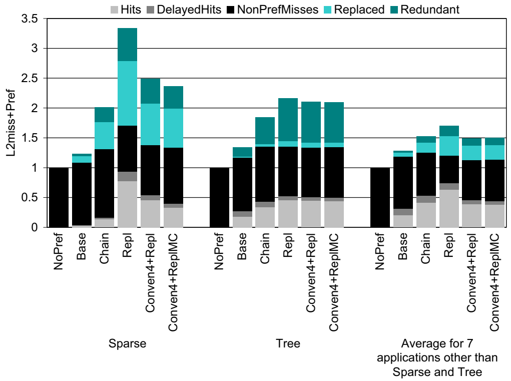


Figure 9. Breakdown of the L2 misses and lines prefetched by the ULMT (prefetches). The original misses are normalized to 1.

Processor Location

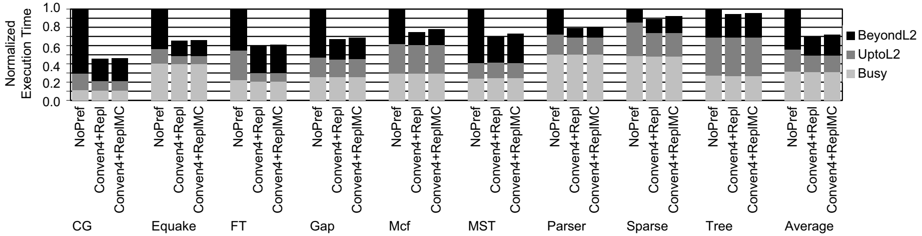


Figure 8. Execution time for different locations of the memory processor.

Implementation Considerations

- Does this idea sound practical? What are potential hurdles in implementing this prefetching scheme?

Implementation Considerations

- Does this idea sound practical? What are potential hurdles in implementing this prefetching scheme?
  - multiple programs/threads/cores
  - inaccurate information due to cache misses disappearing
  - hard to add processor to memory (different silicon technology)
  - on DRAM with multiple chips it is not clear where to put the processor

Summary

Strengths:

Weaknesses:

Summary

Strengths:

- considers a lot of different scenarios
- achieves very respectable speedups
- minimal changes necessary to the processor

Weaknesses:

- Only prefetches to L2
- Processor-in-Memory ideas have been floated around a lot, but never seem to happen
- Unclear what the interaction with the OS would be