np-1

Neighborhood Prefetching

David M. Koppelman

Electrical & Computer Engineering
Louisiana State University

Formatted 13:38, 20 November 1999 from pren˙sem.
np-1

np-2

Outline

Motivation, Background, Prior Work

Computer Speed

Memory Slowness

Caches

Prefetching

Contribution: Neighborhood Prefetching

Operation

Analysis

Performance

Conclusions

Formatted 13:38, 20 November 1999 from pren˙sem.

np-2

np-3

Motivation and Background

Computer:
A device that executes instructions.

A *Good* Computer:
A device that executes instructions fast.

A *Better* Computer:
A device that executes instructions faster.

An *Adequate* Computer:
??? An open question.

np-4

Computer Programs

Programmers write in *high-level languages*:

```
double sum = 0, i = 1, pi;
while( i < 10000000000 ){ sum+=1/i; i+=2; sum-=1/i; i+=2; }
pi = sum * 4.0;
```

np-5

High-level language code is converted to *assembler*:

```
.LL9:
     fdivd %f10,%f6,%f2
     faddd %f6,%f12,%f6
     fdivd %f10,%f6,%f4
     faddd %f8,%f2,%f8
     faddd %f6,%f12,%f6
     fcmped %f6,%f14
     nop
     fbl .LL9
     fsubd %f8,%f4,%f8
     sethi %hi(.LLC10),%o3
     ldd [%o3+%lo(.LLC10)],%f2
     fmuld %f8,%f2,%f2
```


Formatted 13:38, 20 November 1999 from pren˙sem.
np-5

Which is assembled and linked into executable form:

Each integer corresponds to one instruction.

Only portion corresponding to code above shown.

In decimal:

```
385876033, -585440320, -1784676314, -1751121881, 385876033,
-652549192, -2052945466, -1918793652, -1985836602
```

In hexadecimal (radix 16):

```
0x17000041, 0xdd1ae3c0, 0x95a00026, 0x97a00027, 0x17000041,
0xd91ae3b8, 0x85a289c6, 0x8da1884c, 0x89a289c6
```

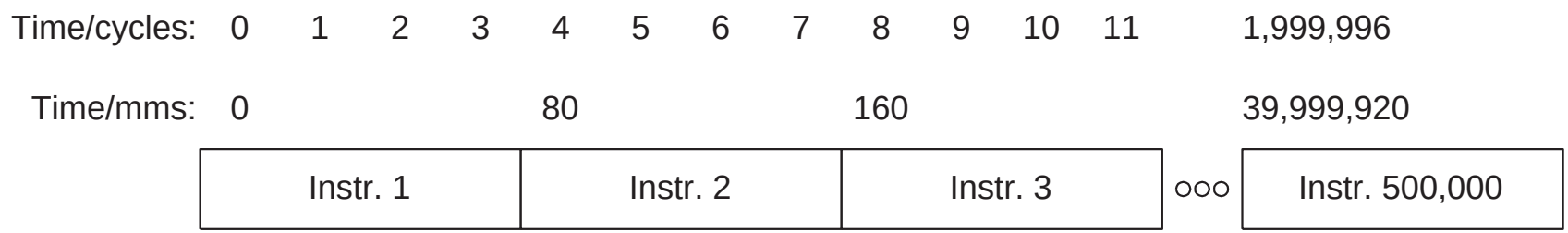
np-7

Program Execution

To Run: Execute Instructions

Possible Execution (Once upon a time.)

In program order . . .
. . . one at a time.



Execution time: # instructions × ticks per instr. × clock period.

Formatted 13:38, 20 November 1999 from pren˙sem.

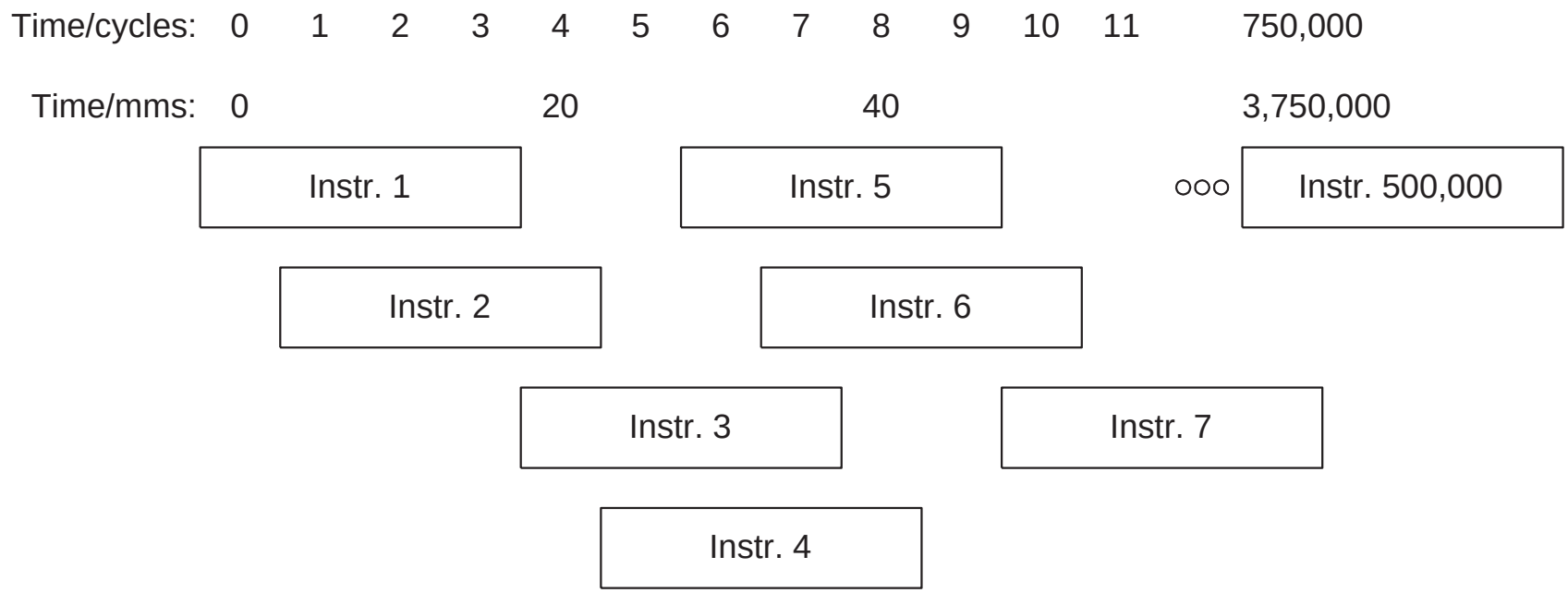
np-7

np-8

Execution: Pipelined, In Order

To Run Faster: Overlap Instructions *(Pipelined Execution)*

Result must be same as one-at-a-time execution . . .
. . . not too difficult to achieve.



Execution time at best: # instructions × clock period . . .
. . . assuming 1 cycle to start each instruction . . .
. . . instruction can start each cycle.

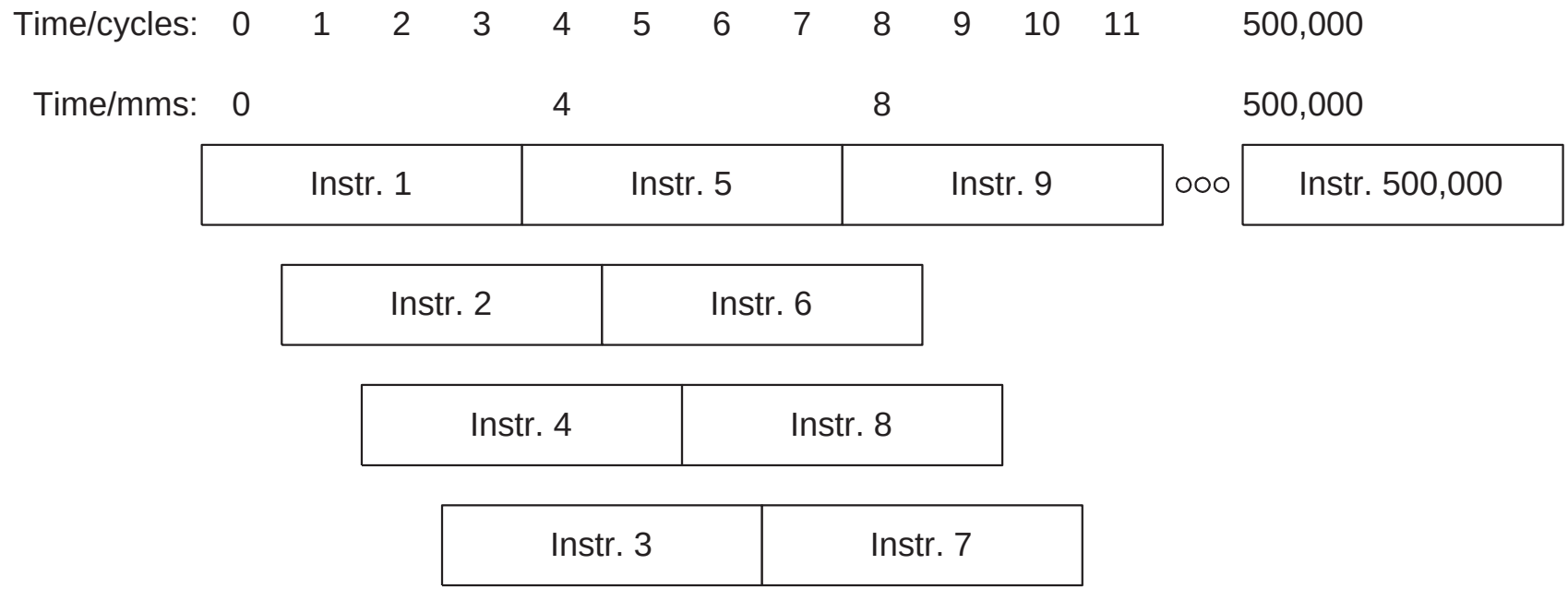
Formatted 13:38, 20 November 1999 from pren˙sem.
np-8

np-9

Execution: Pipelined, Out of Order

To Run Even Faster: Overlap Instructions and Start Out of Order

Sometimes skip an instruction and execute it later.



Execution time at best: # instructions × clock period . . .
. . . assuming 1 cycle to start each instruction . . .
. . . instruction can start each cycle.

Same ideal execution time as pipelined, In-Order, but better on average.

Formatted 13:38, 20 November 1999 from pren˙sem.

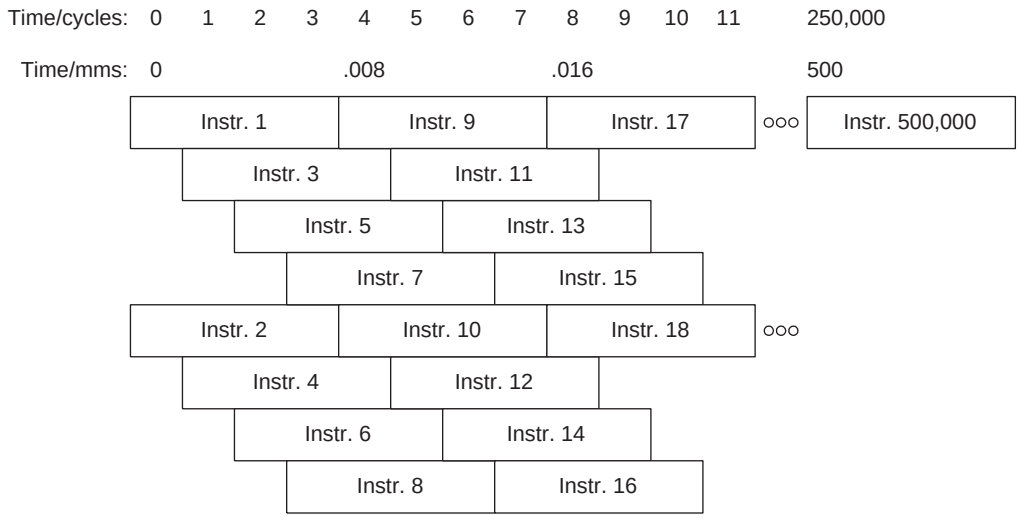
np-9

np-10

Execution: Pipelined, Out of Order, Superscalar

To Run Fastest[1]: Overlap, Out-of-Order, Start n per Tick (*n-Way Superscalar*).

Requires about n times as much hardware.



Execution time at best: $\frac{1}{n}\times$ # instructions $\times$ clock period ...

... assuming 1 cycle to start each instruction ...

... instruction can start each cycle.

[1] Using a conventional serial instruction set architecture.

Formatted 13:38, 20 November 1999 from pren˙sem.

np-10

np-11

Memory Speed v. CPU Speed

As technology advances . . .

. . . more gates available for processor design . . .

. . . clock frequencies increase . . .

. . . *but* memory access time nearly constant.

Growing number of gates allow improved performance via logic-heavy superscalar, out-of-order designs.

Increasing clock frequencies further improve performance . . .

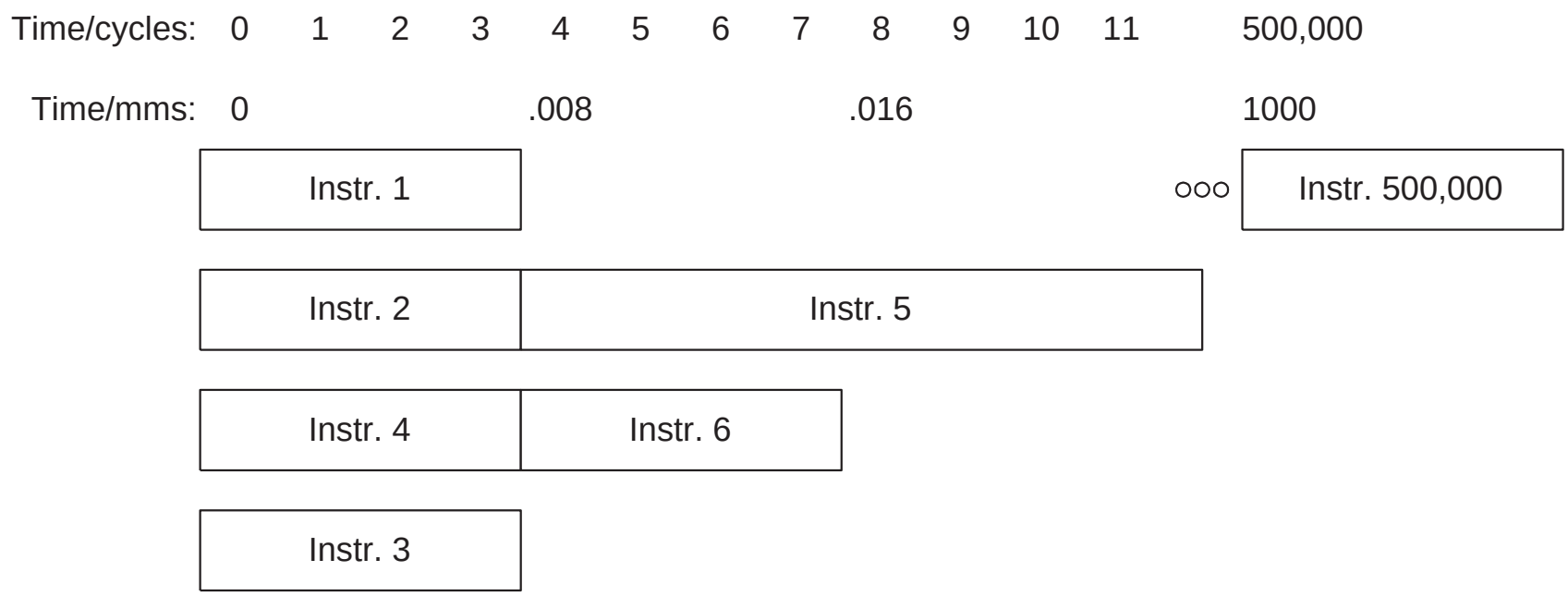
. . . though without employing computer engineers `:-)`.

Meanwhile memory access time falls far behind.

np-12

Execution: One More Thing . . .

Execution sometimes slowed by memory accesses that miss cache.



Miss can delay instruction for over 100 cycles . . .
. . . during which $\approx$ 200 instructions could be executed . . .
. . . if they didn't have to wait for missing instruction.

A large and growing obstacle to faster processors.

Problem dubbed "The Memory Wall."

Formatted 13:38, 20 November 1999 from pren˙sem.

np-12

Memory Access

np-13

Load and *Store* instructions access memory.

```
ldd [%o3+%lo(.LLC10)],%f2 ! Load double.
std %f2,[%fp-24]          ! Store double.
```

To execute these instructions CPU presents an *address* to *memory system*. . .

. . . and may present a data value (for stores) . . .

. . . or wait for a data value (for loads).

Address is an integer referring to a storage location.

Formatted 13:38, 20 November 1999 from pren˙sem.

np-13

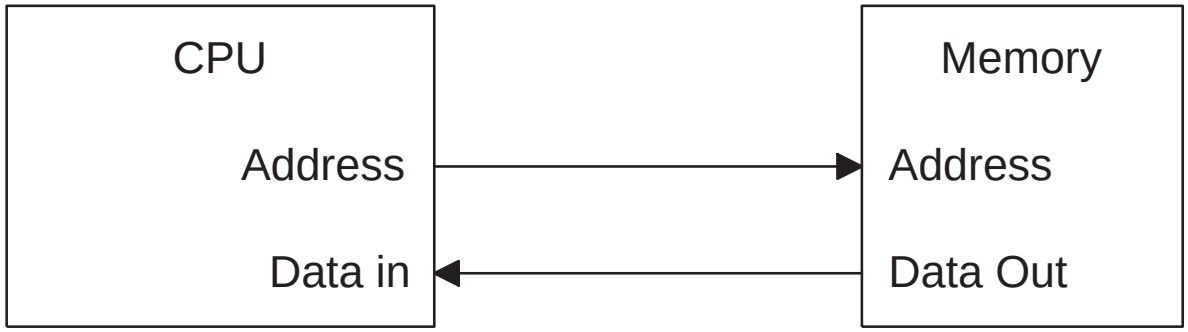
np-14

Memory System Construction (Oversimplified)

Consider only load instructions.

Memory system build using memory devices . . .
. . . having a single *address* input . . .
. . . and a single *data* output.

Just connect to CPU! (Remember, this is oversimplified.)



Formatted 13:38, 20 November 1999 from pren˙sem.
np-14

np-15

Types of Memory Devices

Two Types of Memory Devices:

Cheap, slow (20-cycle latency), high capacity (density).

Expensive, fast (1-cycle latency), low capacity (density).

Which ones should we use?

All cheap: too slow and getting slower.

All expensive: too expensive and maybe not as fast.

Both: close to speed of expensive at cost of cheap ...
... amazingly, it works.

Using Both

Bulk of storage provided by cheap memory.

Frequently used data *cached* in expensive memory.

Formatted 13:38, 20 November 1999 from pren˙sem.
np-15

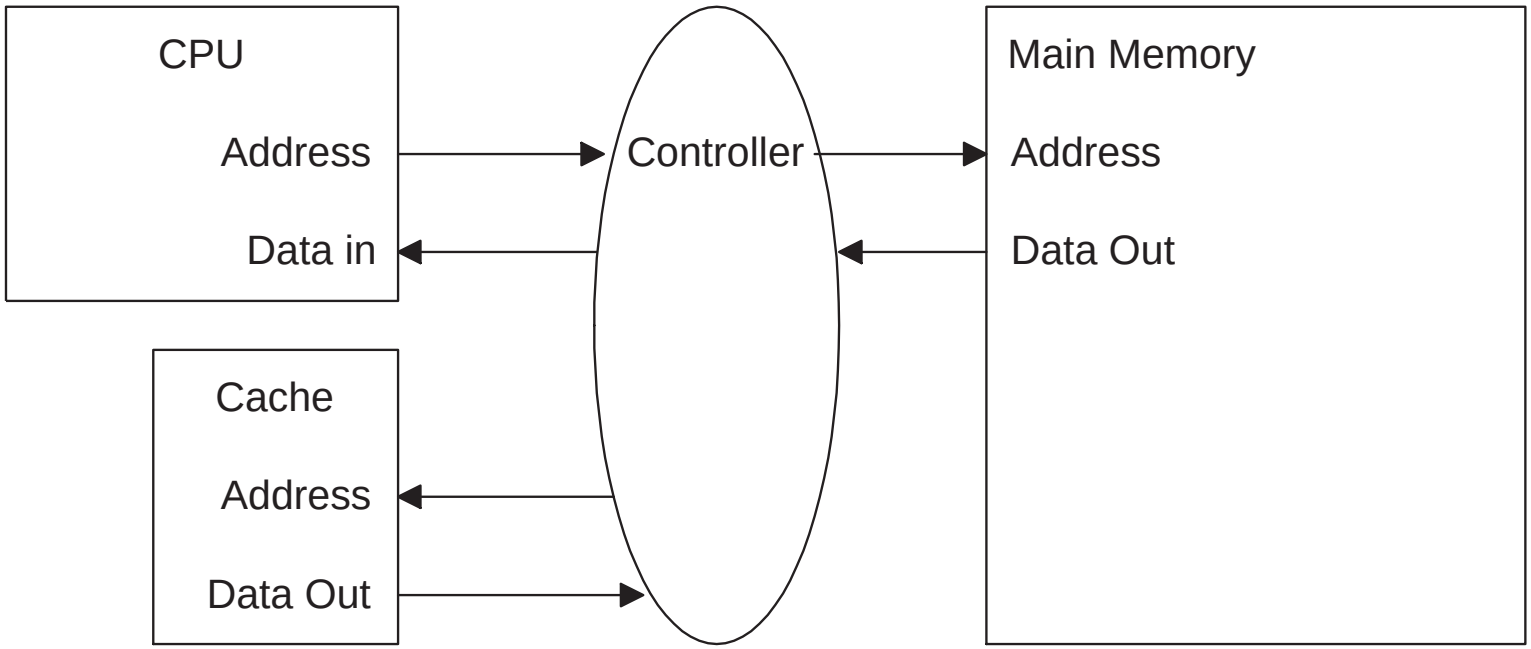
np-16

Cache Construction

Use both cheap (*main*) and expensive (*cache*) memory.

Cache checked for address, if not present main memory accessed.

We hope to find data in cache.



Formatted 13:38, 20 November 1999 from pren˙sem.

np-16

np-17

Cache Terminology

Hit:
A memory accesses in which data found in cache.

Miss:
A memory accesses in which data not found in cache.

Hit Ratio:
Fraction or percentage of memory accesses that hit.

Formatted 13:38, 20 November 1999 from pren˙sem.
np-17

np-18

Spatial and Temporal Locality

Address Sequence:
Sequence of address (in time) issued by CPU (for loads, stores, etc.)

Example: 100, 200, 2000, 500, 10

Vitally Important Characteristics of Address Sequences

- *Temporal Locality*:
  Tendency for addresses to recur soon.

  Example: **100**, 200, **100**, 205, 5000, 207, 9000.

- *Spatial Locality*:
  Tendency for addresses to soon be followed by nearby addresses.

  Example: 100, **200**, 100, **205**, 5000, **207**, 9000.

In each example, second **bold** address probably in cache.

Were it not for spatial and temporal locality ...
... caches would be useless ...
... and computers would be much slower.

np-19

Limiting Factors on Cache Performance

Capacity Misses
A miss to data that was in cache . . .
. . . but was removed to make room for other data.

Can be reduced by increasing size of cache.

Cold Misses
A miss to data that has never been in the cache.

Can be reduced only by predicting that data will be accessed.

True Sharing Misses
A miss to data that was written at another processor . . .
. . . and must be brought to missing processor.

Can be reduced only by predicting that data will be re-accessed.

Formatted 13:38, 20 November 1999 from pren˙sem.
np-19

np-20

Hardware Prefetching

Idea: use hardware to cache data *in advance* of need ...
... by predicting access addresses.

Existing Methods

- *Line Size*
- *Sequential*
- *Adaptive Sequential*
- *Stride*

np-21

Prefetching With Long Lines

Used in nearly all systems . . . though not usually called prefetching.

Line:
Unit of data moved into cache, . . .
. . . consists of several addresses . . .
. . . only one of those may have been requested . . .
. . . but access to others is likely.

Longer lines mean more addresses.

Spatial locality exploited with longer lines.

As line size is increased . . .
. . . percentage of line that is accessed decreases . . .
. . . but time to fetch entire line increases.

Long lines cause additional problems with data sharing.

Multiprocessor line sizes: 64 to 256 bytes.

Formatted 13:38, 20 November 1999 from pren˙sem.
np-21

np-22

Sequential Prefetching

Pattern: sequential.

Consider the following address sequence:

100, 3000, **101**, 2000, 3000, **102**, **103**, 6, ...

Addresses shown in bold form predictable sequence.

A *sequential prefetch* mechanism detects sequence ...
... and fetches several addresses ahead.

Hardware looks for misses to several consecutive addresses ...
... and then will prefetch several addresses ahead.

Formatted 13:38, 20 November 1999 from pren˙sem.

np-22

Sequential Prefetching Reported Work

np-23

Early Work:

B.T. Bennett and P.A. Franaczek, 1976 [1].

J.D. Gindele, 1977 [7].

A.J. Smith, 1978 [8]

Recent Work

J. Fu and J.H. Patel, 1991 [5].

Formatted 13:38, 20 November 1999 from pren˙sem.

np-23

np-24

Sequential Prefetching Tradeoffs

Advantages: Simple, can anticipate many accesses.

Disadvantage: Inefficient, can pollute cache with unused data.

Formatted 13:38, 20 November 1999 from pren˙sem.

np-24

Adaptive Sequential Prefetching

np-25

Pattern: sequential.

Fetch next d addresses, where d is *degree*.

Adjust degree based on success of past prefetches.

:

Consider:

100, 3000, **101**, 2000, 3000, **102**, **103**, 6, ...

If $d = 3$ then on access to 100 addresses 101, 102, and 103 prefetched.

If prefetches not used, d reduced, possibly to zero ...
... turning off prefetching.

Adaptive Sequential Prefetching

np-26

Published Work

Fredrik Dahlgren, Michel Dubois, and Per Stenström, 1995 [3].

Fredrik Dahlgren, and Per Stenström, 1996, [4].

Stride Prefetching

np-27

Pattern: constant difference, *stride*, between addresses.

Challenge find what strides are and when to use them.

Consider

100, 3000, **110**, 2000, 3000, **120**, **130**, 6, ...

Here stride is 10.

Consider

100, 500, **110**, 505, 510, 515, **120**, **130**, 520, ...

Here stride is 10 and 5.

Which stride do we use?

Method 1: base on address, *address-correlated*. (100's use stride 10 in example).

Method 2: base on instruction, *instruction-correlated*.

np-28

Stride Prefetching

Address-Correlated Work

J. Fu and J.H. Patel, 1991, 1992 [5,6].

Instruction-correlated Work

Tien-Fu Chen and Jean-Loup Baer, 1995 [2].

Fredrik Dahlgren and Per Stenström, 1996 [4].

Formatted 13:38, 20 November 1999 from pren˙sem.

np-28

Stride Prefetching

np-29

Advantage: covers more cases than sequential.

Disadvantage: stride must be learned for each instruction.

np-30

Neighborhood Prefetching

Pattern: a repeated set of address offsets, *neighborhood*..

Consider:

100, 500, **101**, 705, 515, **105**, **98**, 520, ...

Neighborhood: (offsets from 100): $\{0, 1, 5, -2\}$.

On miss to 100, neighborhood used to prefetch 101, 105, and 98.

Neighborhood associated with an access instruction.

Formatted 13:38, 20 November 1999 from pren˙sem.

np-30

np-31

Neighborhood Prefetching

Advantages:

Can detect sequential patterns: $\{0, 1, 2, 3, \ldots\}$.

Can detect stride patterns: $\{0, 10, 20, 30, \ldots\}$.

Can detect arbitrary patterns: $\{0, 10, 20, 50\}$.

Formatted 13:38, 20 November 1999 from pren˙sem.

np-31

np-32

Neighborhood Prefetching Implementation

Uses Two Tables: Recent Miss Table, Neighborhood History Table.

Recent Miss Table (RMT)

Entry for recently encountered neighborhoods.

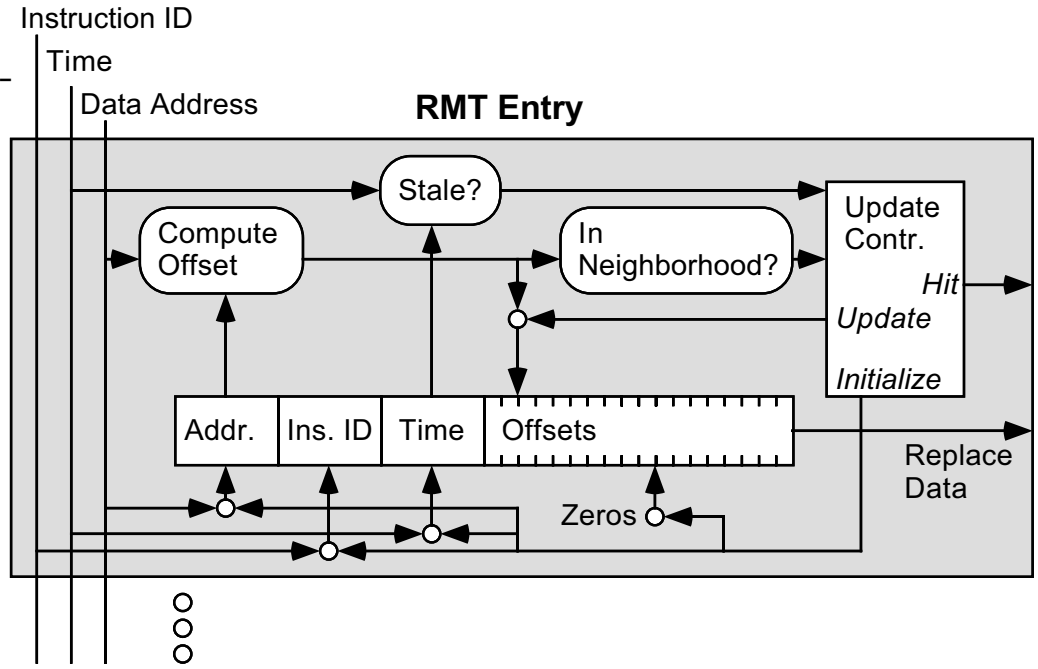
Entry holds offsets in neighborhood ...

... instruction creating entry (miss at offset zero) ...

... and other data.

On miss update existing entry or create new ones.

Entries occasionally moved to neighborhood history table.

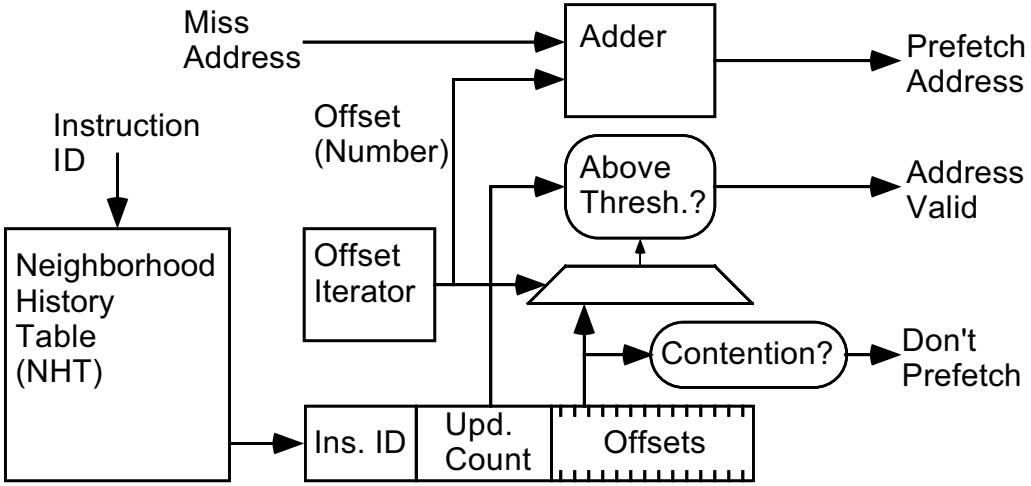


Formatted 13:38, 20 November 1999 from pren˙sem.

np-32

np-33

Neighborhood History Table



Neighborhood History Table

Entry for memory access instructions.

Entry retrieved on a miss using address of missing instruction.

Retrieved entry used to construct prefetch addresses.

Formatted 13:38, 20 November 1999 from pren˙sem.

np-33

np-34

Recent Miss Table Fields

Field Name	Abbr. Size / bits	Description
Initiator.	IN 20	Identity (hash of PC or index into instruction cache) of instruction creating this entry.
Base address.	BA 32	Effective address accessed by initiator. Offsets are relative to this address.
Initialization time.	IT 20	Time that entry created.
Last access time.	LA 20	Time that entry last accessed.
Saturated.	SA 1	Bit indicating whether a least one offset count is saturated.
Neighborhood.	NI 128	Field divided into offset count subfields for each possible offset (based on neighborhood size) for both reads and writes. An offset count is the number of times that an effective address was encountered at the corresponding offset. In the base configuration, subfield size is four bits and the neighborhood size is 16 offsets, including zero.

Formatted 13:38, 20 November 1999 from pren˙sem.
np-34

np-35

Neighborhood History Table Fields

Field Name	Abbr. Size / bits	Description
Instruction tag.	TG 2	Part of the instruction address; on entry retrieval is compared with the corresponding part of the instruction address used for the lookup. If they are different then the entry is for a different instruction, if they are the same the entry is or may be for the same instruction, depending on the size of the tag. (If an entry for one instruction is used by another instruction predictions will likely be wrong but program execution will still be correct.)
Tag matches.	MT 3	Number of times tag matched. (Initialized to 1, incremented on hit, decremented on miss, saturates at zero and 4.) An entry is not replaced when field is positive.
Prefetch control.	PC 6	A prefetching score. Incremented on each prefetch, decremented on an eviction of a prefetched line, and set to a minimum value if an unused prefetched line is invalidated. In simulated systems saturated at ± 16.
Neighborhood.	NB 128	Field divided into subfields, called *offset counts*, for each possible offset (based on neighborhood size) for both reads and writes. The count is related to the number of times that an effective address was encountered at the corresponding offset. In the base configuration, the subfield size is four bits and the neighborhood size is 16 offsets, including zero.

Formatted 13:38, 20 November 1999 from pren˙sem.
np-35

np-36

Evaluation

Evaluated with execution-driven simulation using LSU Proteus

Base system is a multiprocessor with 2-level cache.

Base system, neighborhood prefetch, and adaptive sequential prefetch compared.

Simulated systems ran SPLASH-2 benchmark suite.

Formatted 13:38, 20 November 1999 from pren˙sem.
np-36

np-37

Base System Parameters

Simulation Parameter	Value
System Size	16 processors
Network Topology	$4^2 = 4 \times 4$ mesh
VM Page Size	2^{12} bytes
TLB Capacity	64 entries
TLB Replacement	LRU, fully assoc.
Cache Size	2^8 sets
Cache Associativity	9, LRU Repl.
Cache Line Size	32 bytes
L1 Cache Hit Latency	1 cycle
L2 Cache Hit Latency	3 cycles
Directory Size	full map
Completion Buffer	5 stores
Memory Latency	10 cycles
Protocol Message Size	$n_{\mathrm{pr}} = 6$ bytes (plus data)
Network Interface Width	4 bytes
Network Link Width	4 bytes
Hop Latency	20 cycles (plus waiting)
Neighborhood Size	16 offsets
NHT Size	256 entries
RMT Size	8 entries

Formatted 13:38, 20 November 1999 from pren˙sem.
np-37

np-38

Experiments

General Performance: Base Configuration

Determine:

Execution Time

Cache Hit Ratio

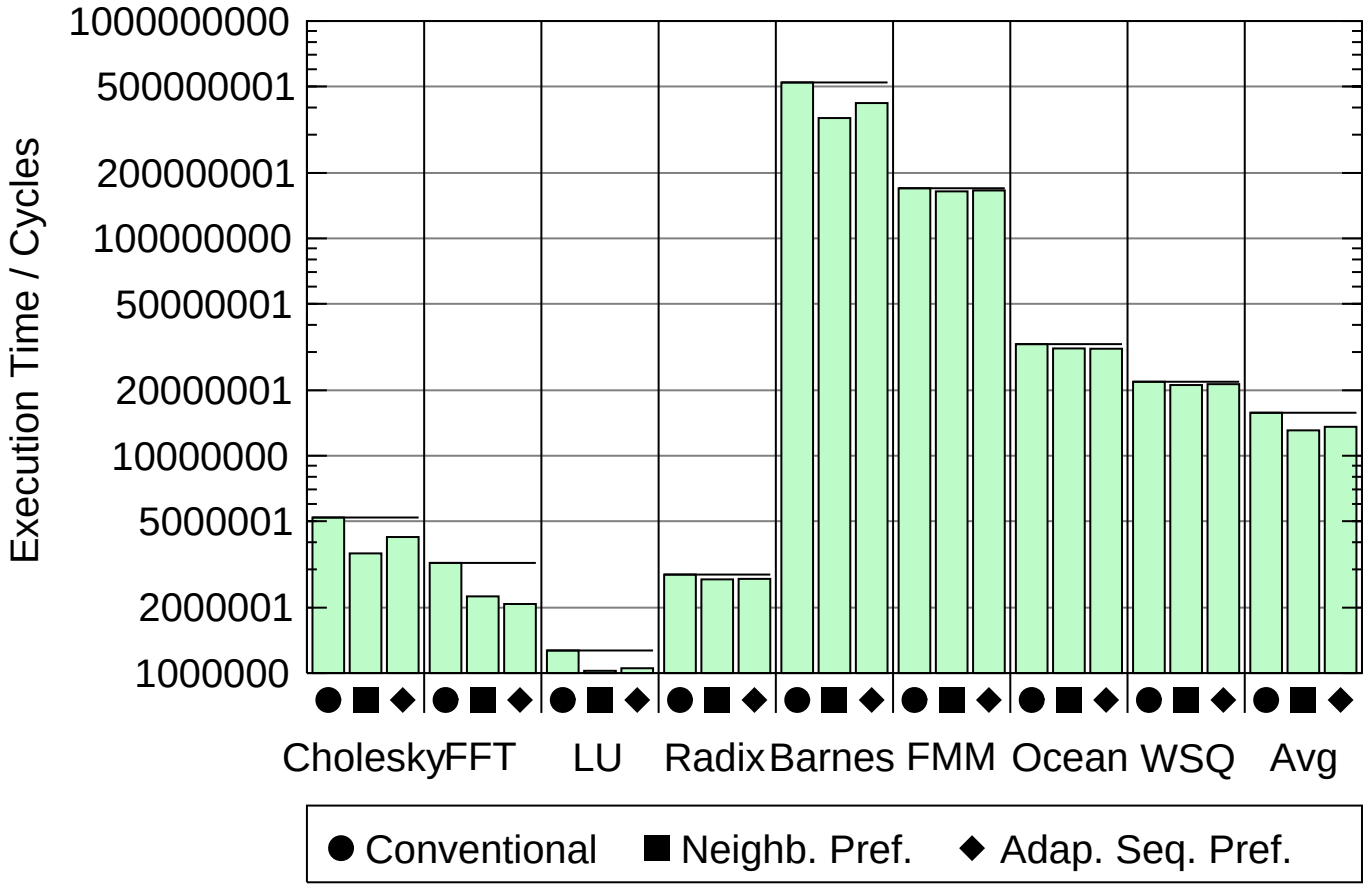
Avoided `:-)` and added `:-(` misses.

Access Latency

Formatted 13:38, 20 November 1999 from pren˙sem.
np-38

np-39

Execution Time

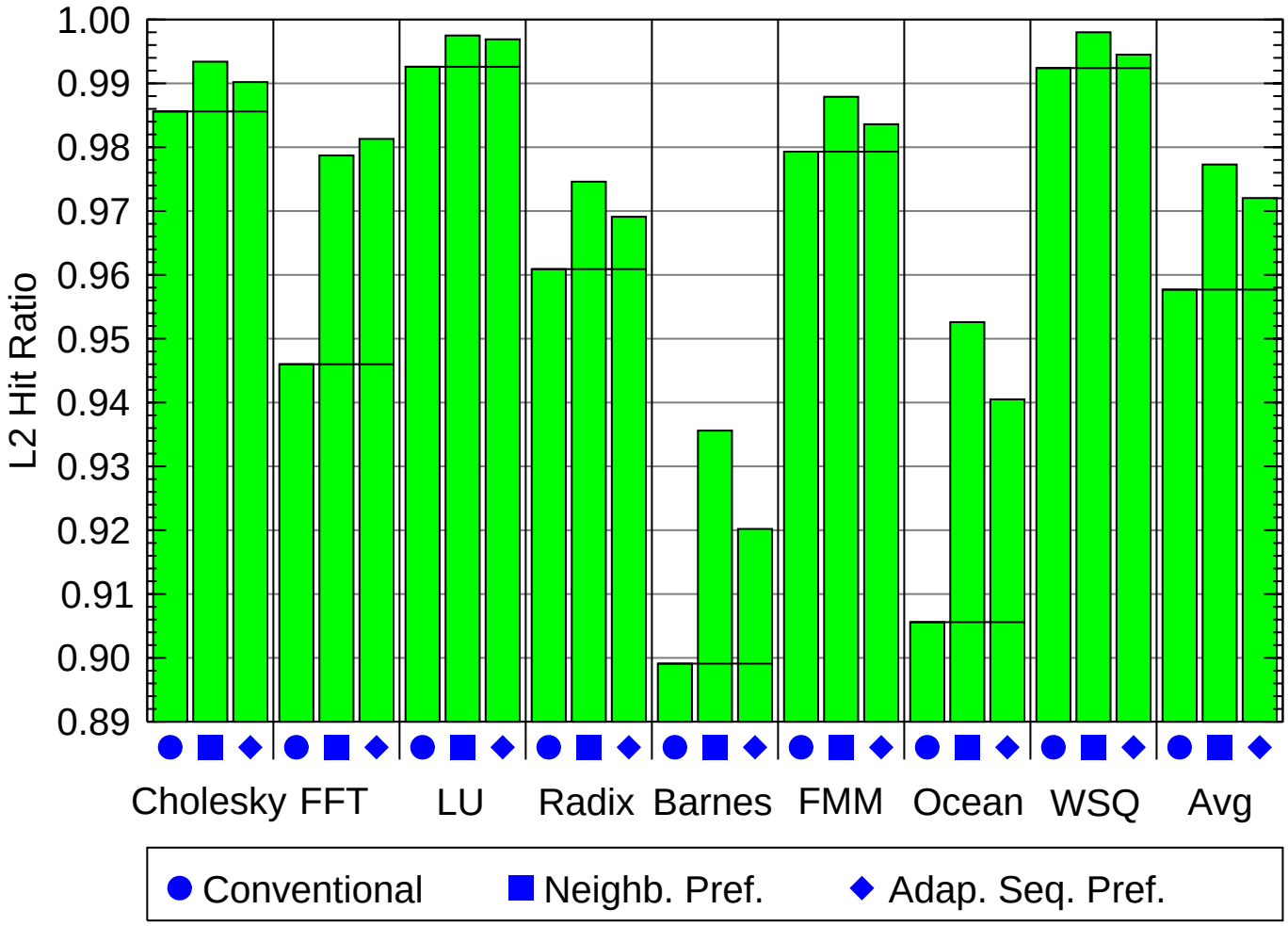


Formatted 13:38, 20 November 1999 from pren˙sem.

np-39

np-40

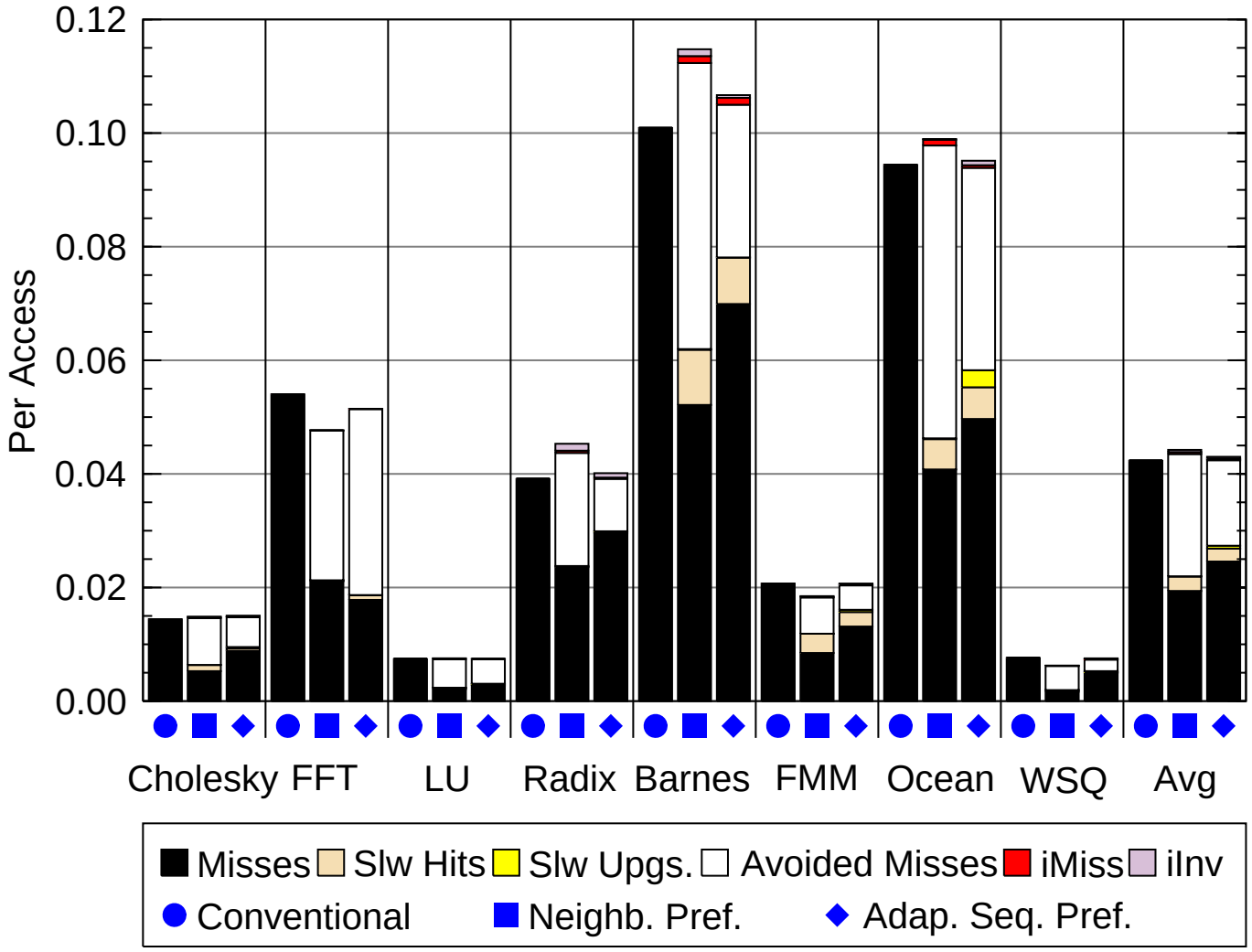
Hit Ratio



Formatted 13:38, 20 November 1999 from pren˙sem.
np-40

np-41

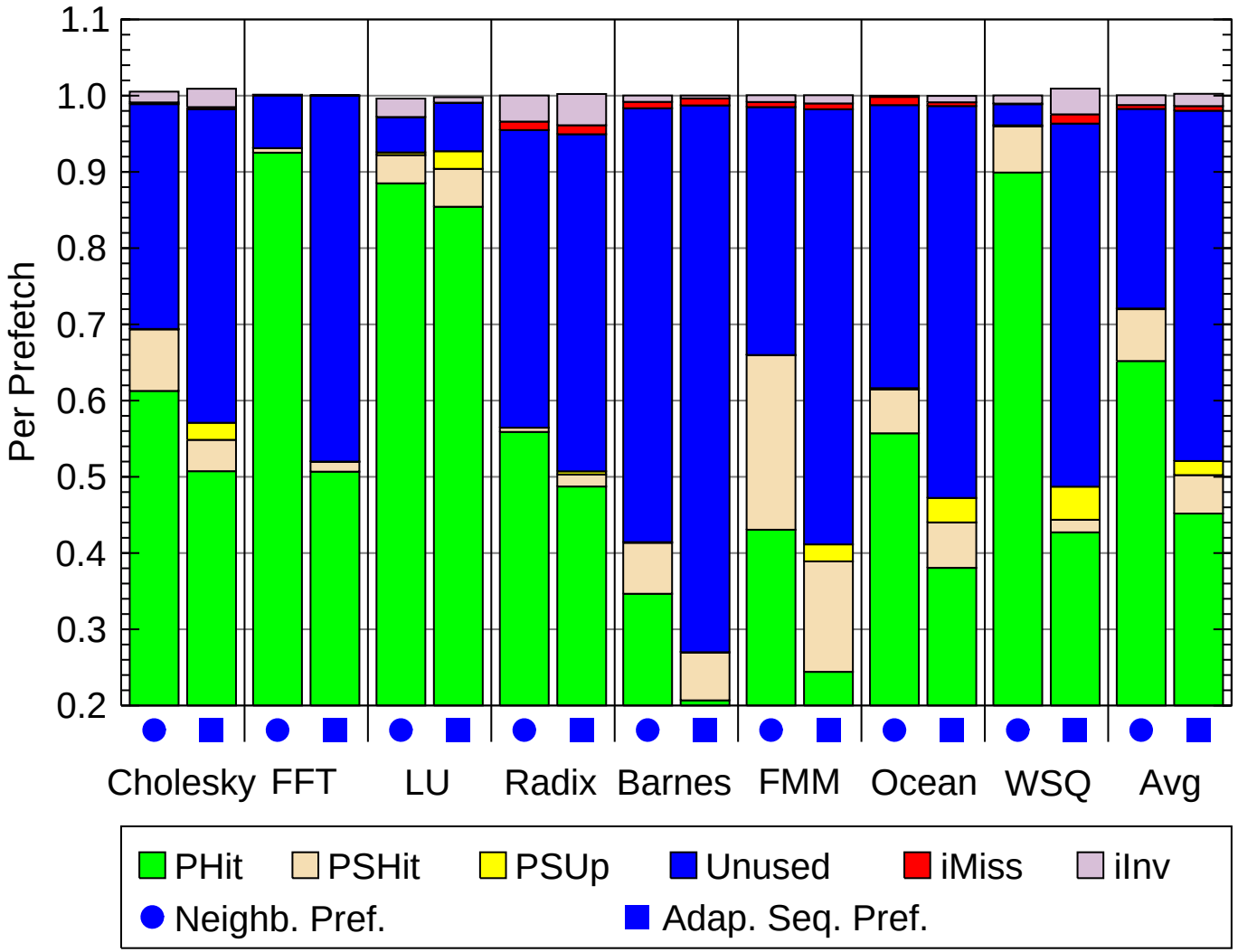
Avoided Misses



Formatted 13:38, 20 November 1999 from pren˙sem.
np-41

np-42

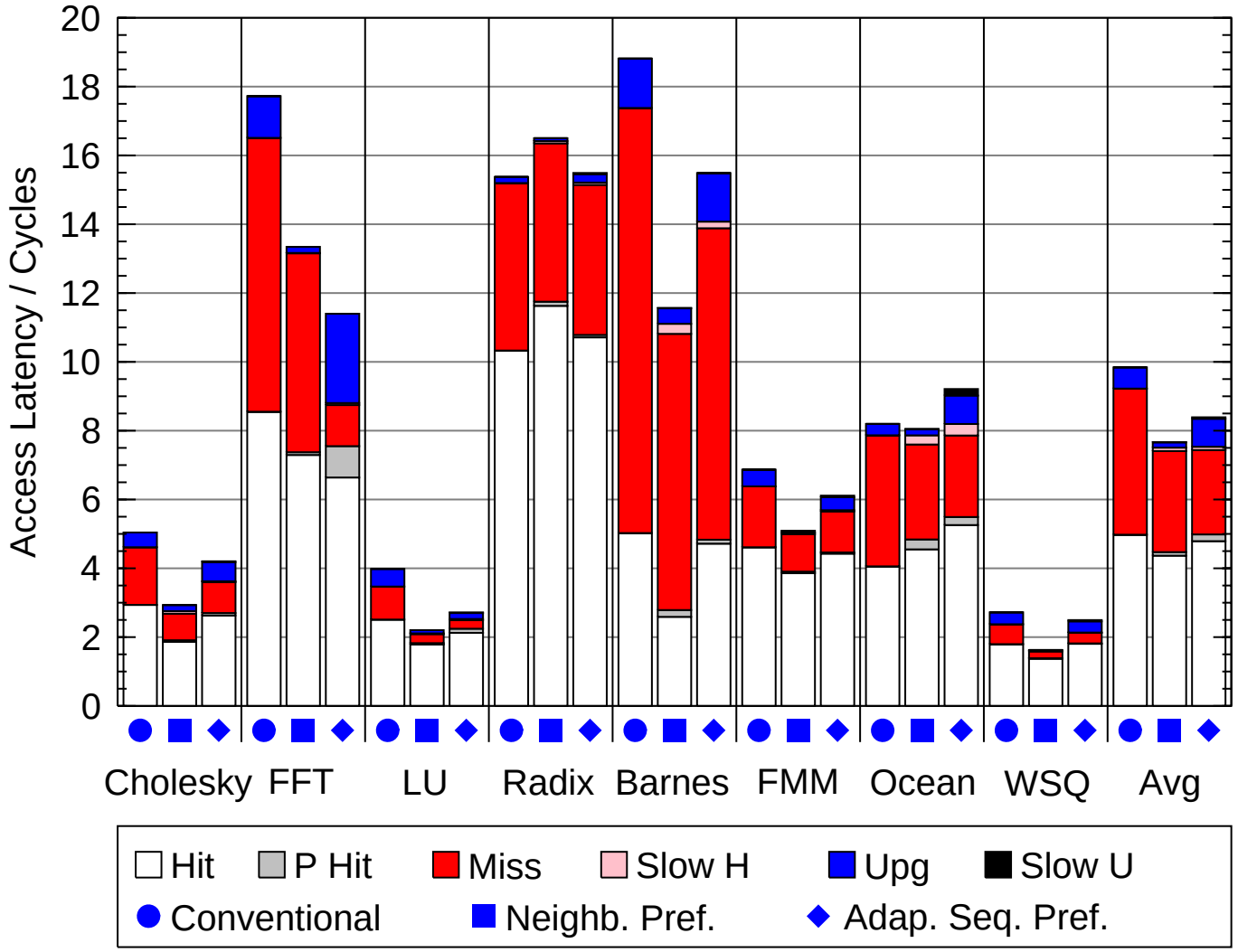
Prefetch Outcome



Formatted 13:38, 20 November 1999 from pren˙sem.
np-42

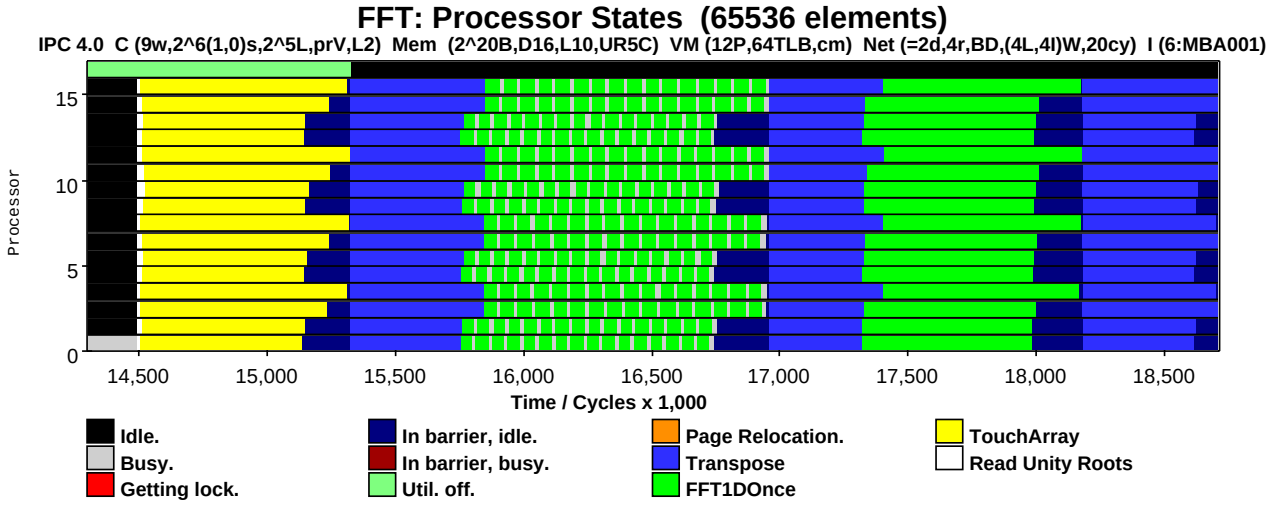
np-43

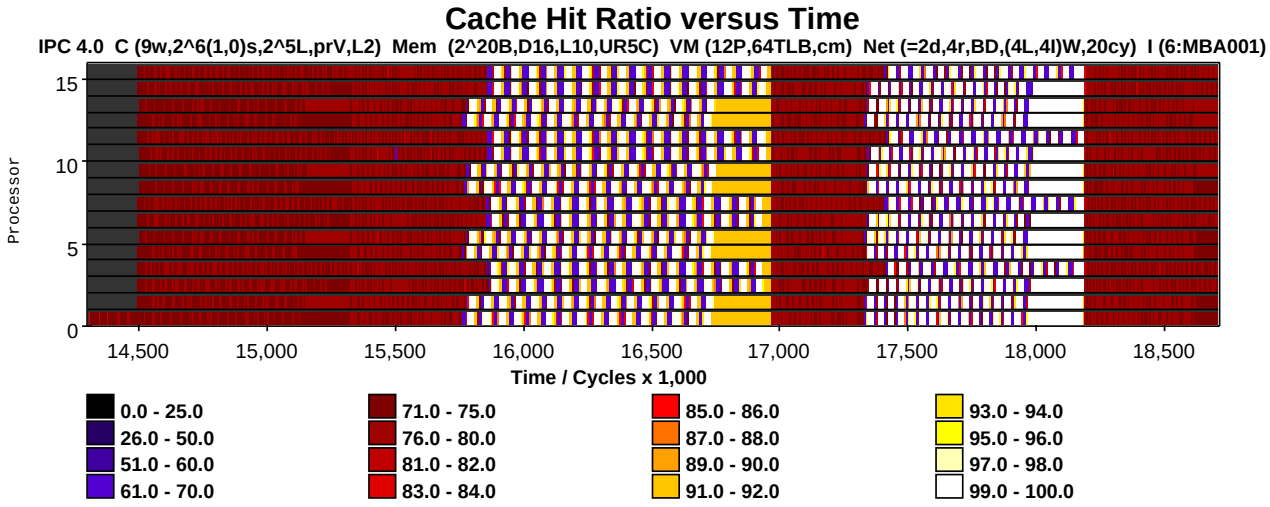
Access Latency

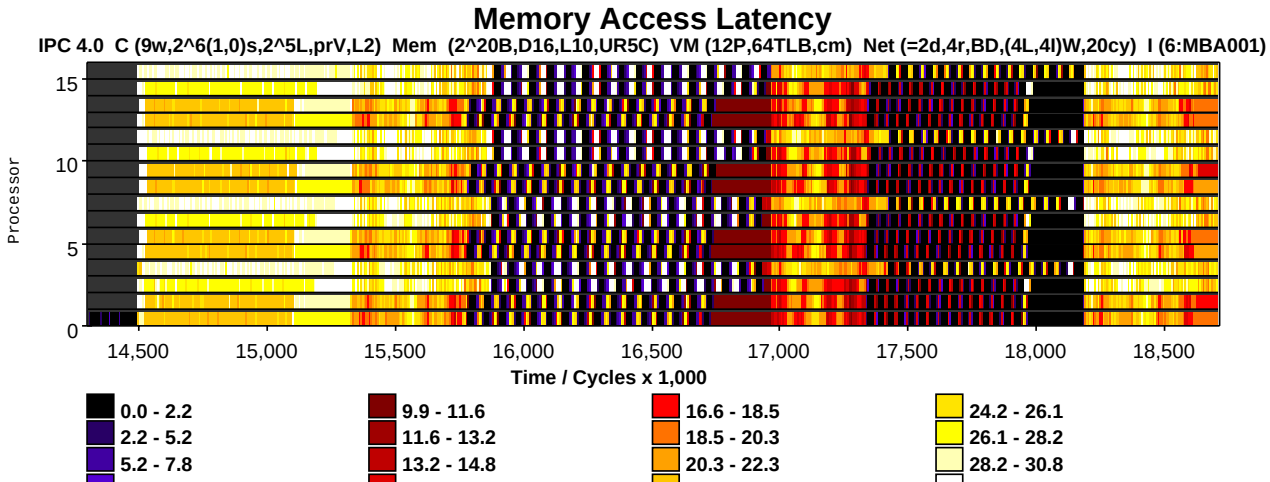


Formatted 13:38, 20 November 1999 from pren˙sem.
np-43

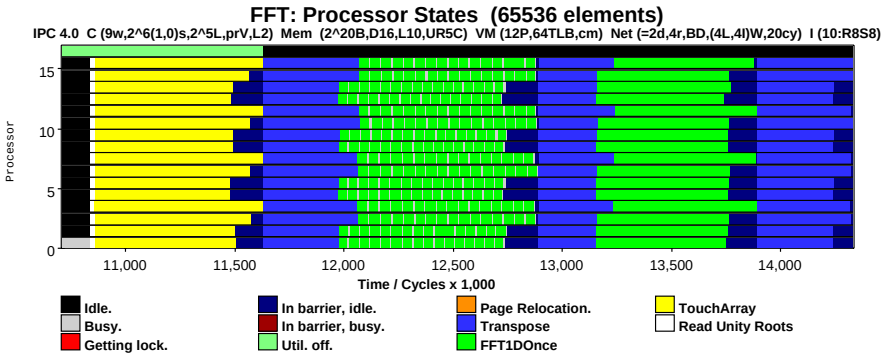
FFT: Behavior Over Time (Normal Cache)

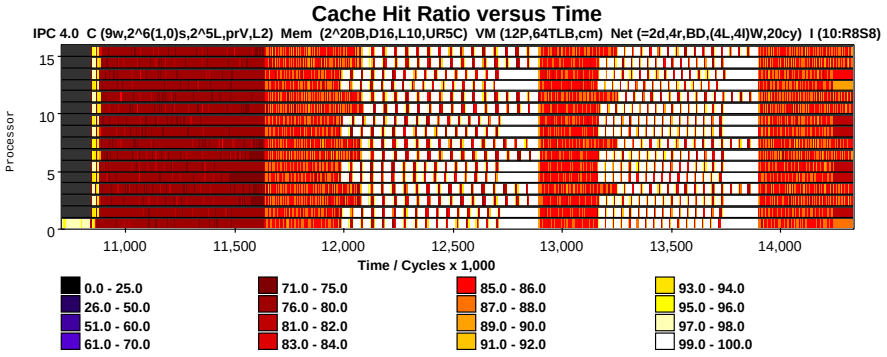


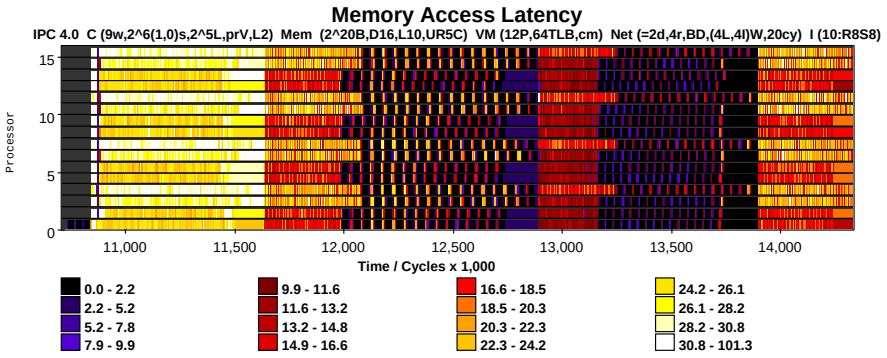


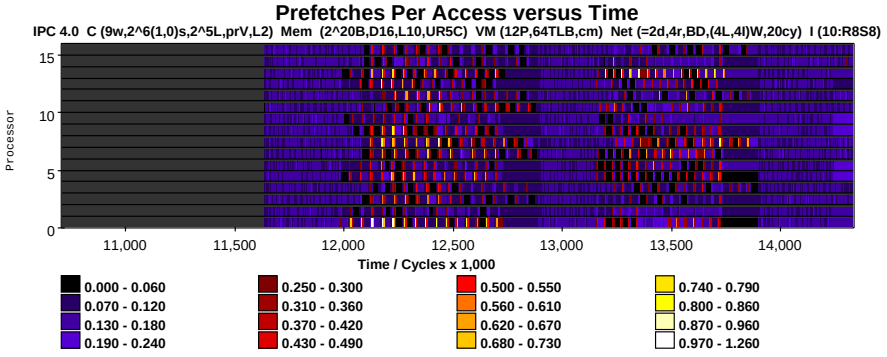


FFT: Behavior Over Time (Neighborhood Prefetch)



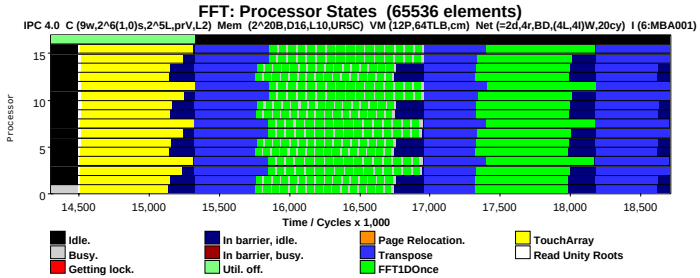


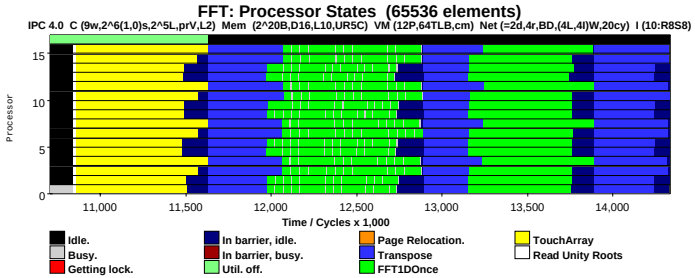


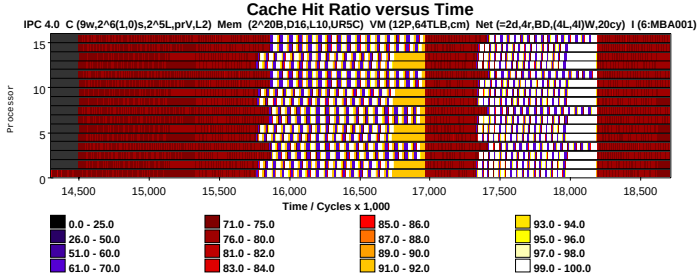


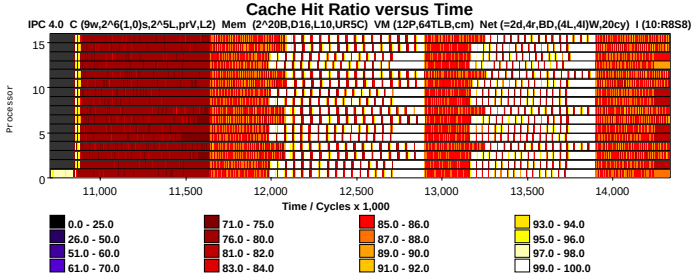
np-46

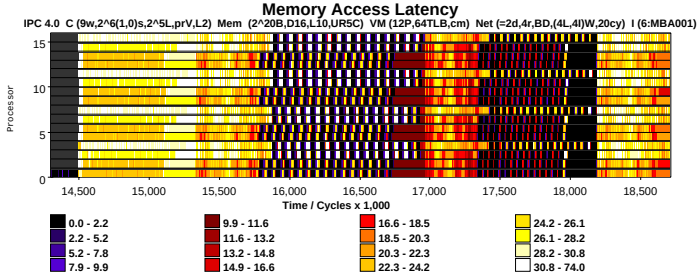
FFT: Behavior Over Time

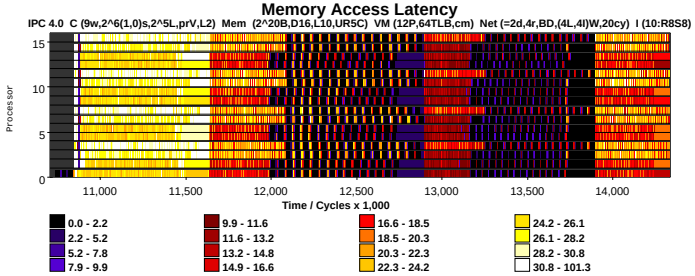


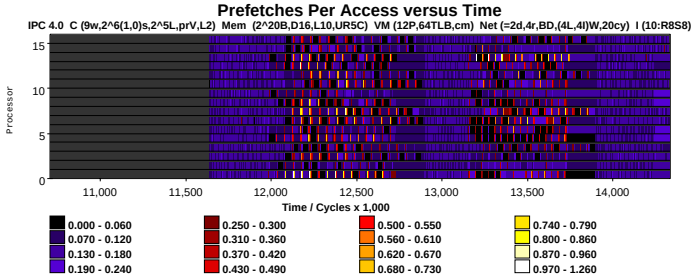








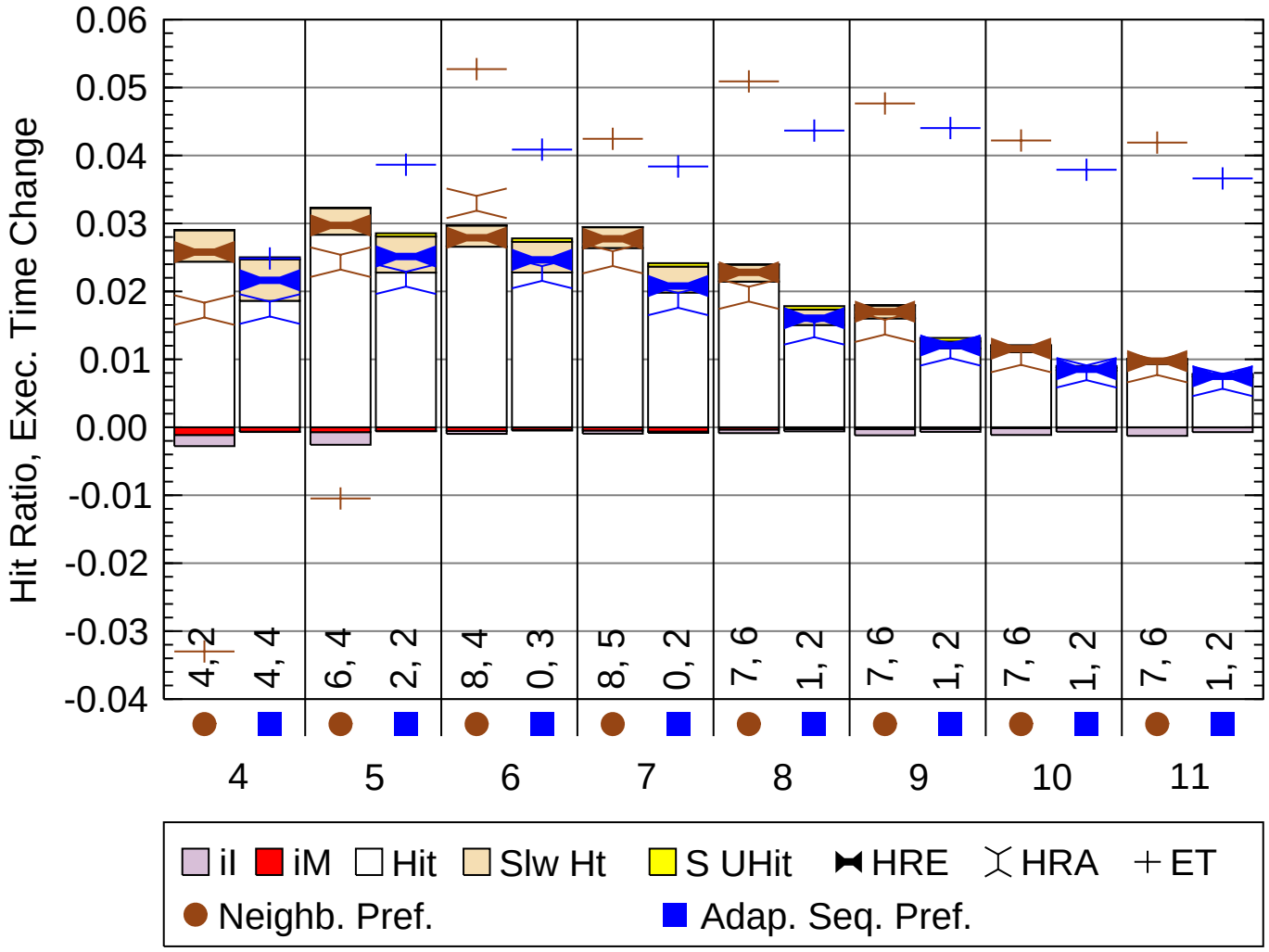




np-46

np-47

Performance v. Cache Size

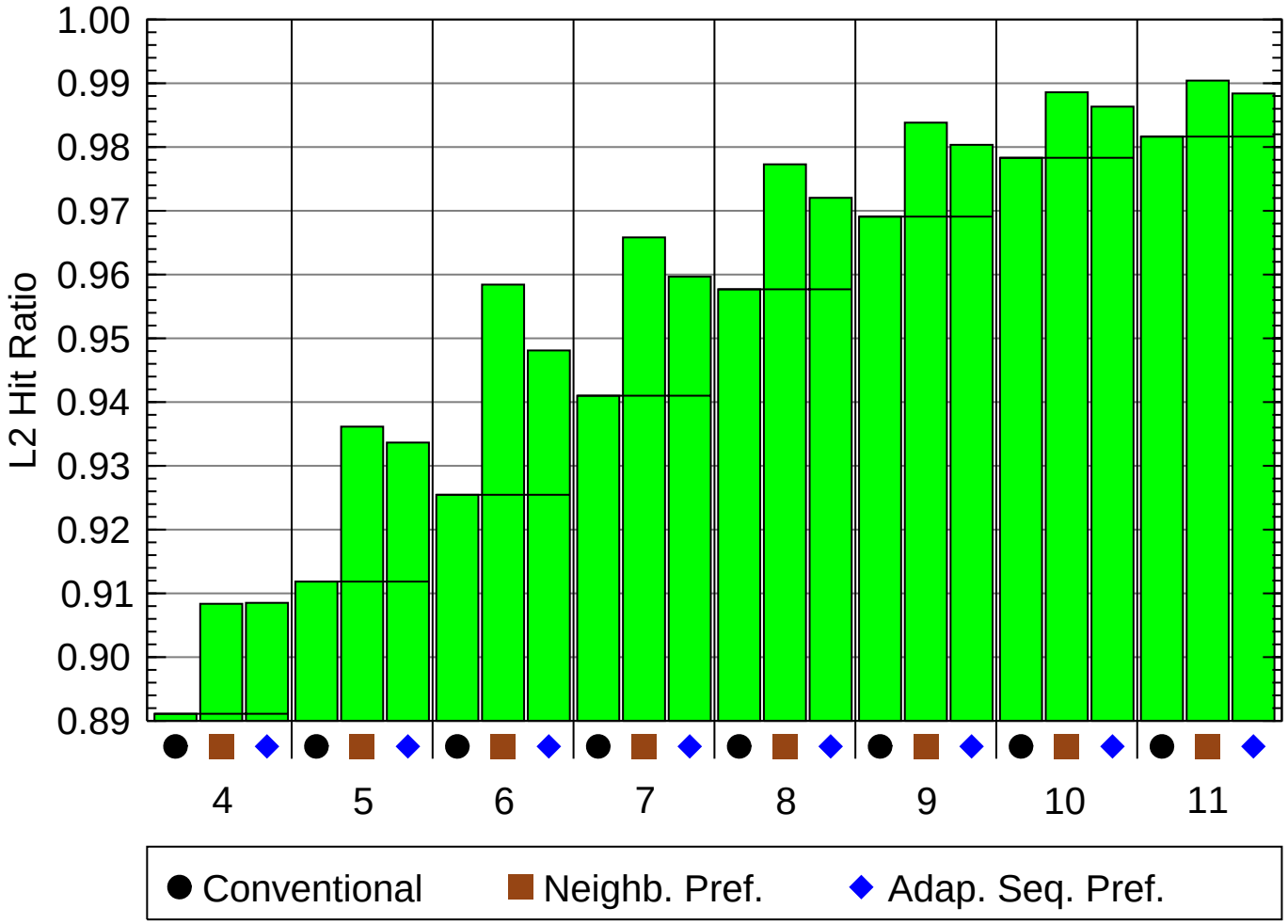


Formatted 13:38, 20 November 1999 from pren˙sem.

np-47

np-48

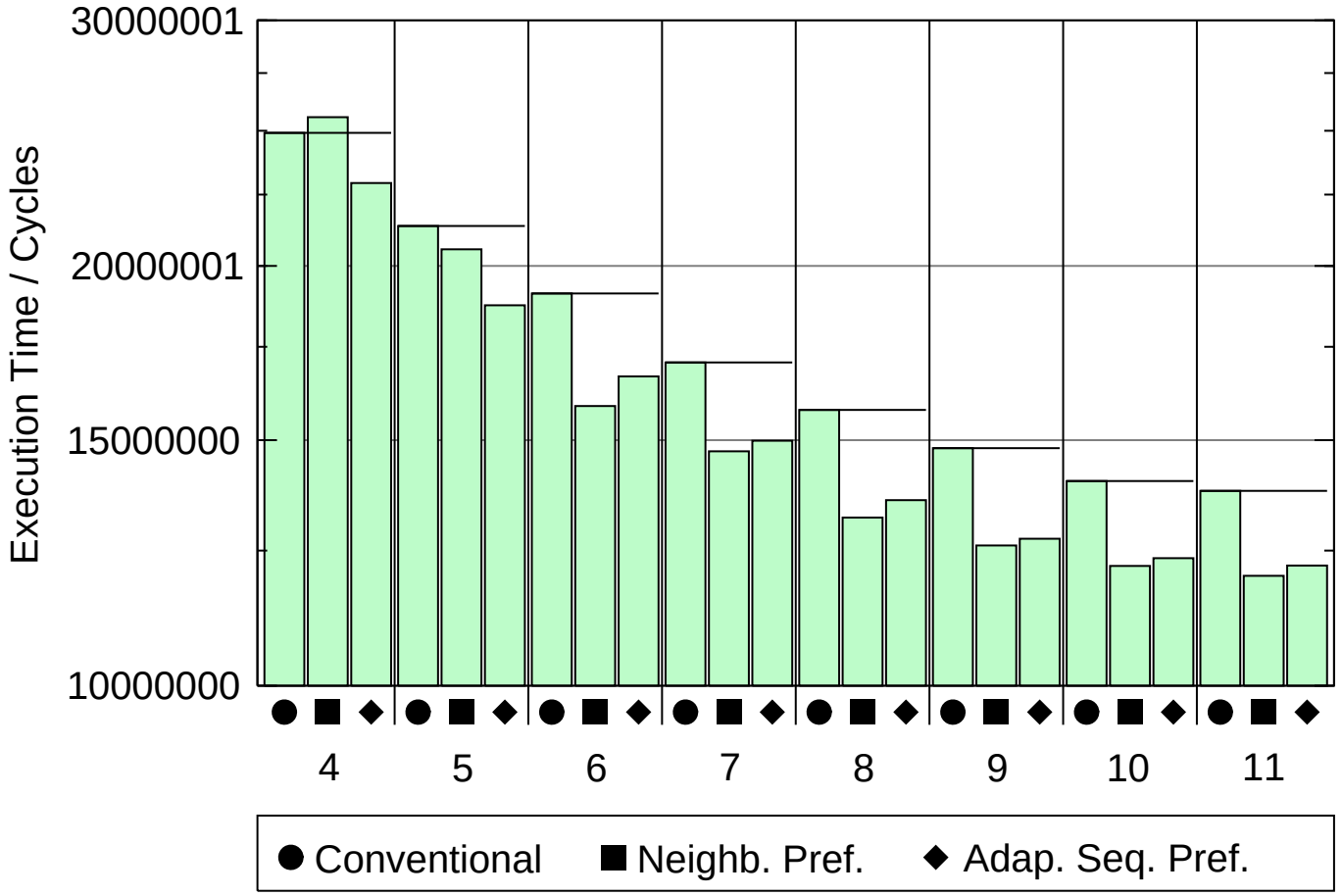
Hit Ratio v. Cache Size



Formatted 13:38, 20 November 1999 from pren˙sem.
np-48

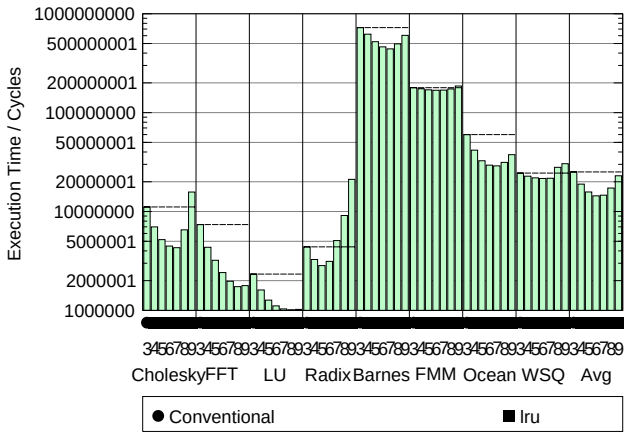
np-49

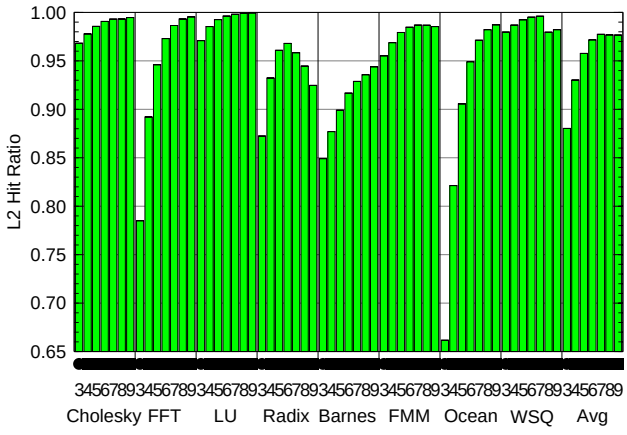
Execution Time v. Cache Size

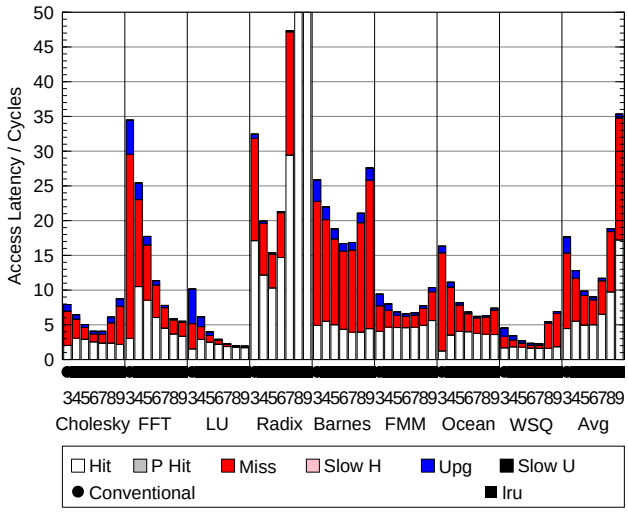


Formatted 13:38, 20 November 1999 from pren˙sem.
np-49

Effect of Line Size

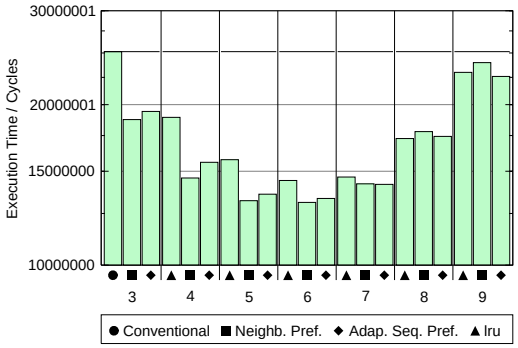


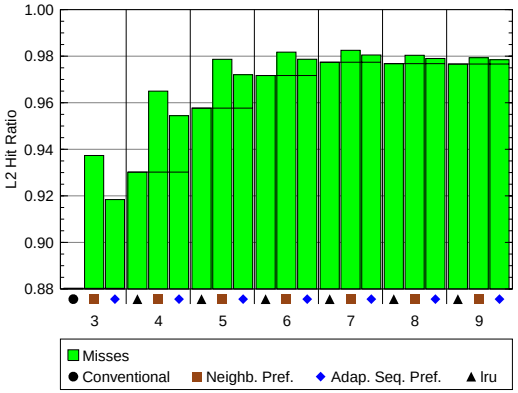


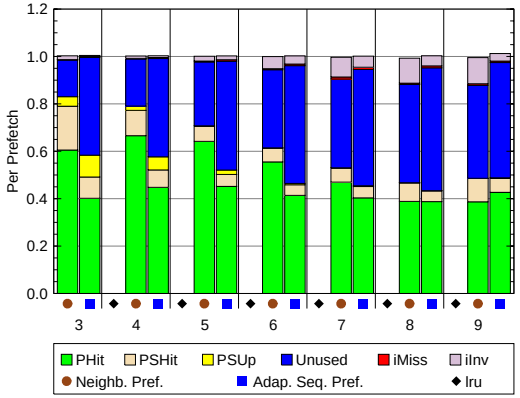


np-51

Line Size Effect On Prefetching







np-51

Formatted 13:38, 20 November 1999 from pren˙sem.

np-52

Conclusions

Usable performance improvement.

Better than adaptive sequential in equal-cost comparisons.

In future might incorporate stride and prefetch-on-hit techniques.

Formatted 13:38, 20 November 1999 from pren˙sem.
np-52

np-53

References

[1] B.T. Bennett and P.A. Franaczek, "Cache memory with prefetching of data by priority," *IBM Technical Disclosure Bulletin,* vol. 18, pp 4231-4232, May 1976.

[2] Tien-Fu Chen and Jean-Loup Baer, "Effective hardware-based data prefetching for high-performance processors," *IEEE Trans. on Computers,* vol. 44, no. 5, pp. 609-623, May 1995.

[3] Fredrik Dahlgren, Michel Dubois, and Per Stenström, "Sequential hardware prefetching in shared-memory multiprocessors," *IEEE Trans. on Parallel and Distributed Systems,* vol. 6, no. 7, pp. 733-746, July 1995.

[4] Fredrik Dahlgren and Per Stenström, "Evaluation of hardware-based stride and sequential prefetching in shared-memory multiprocessors," *IEEE Trans. on Parallel and Distributed Systems,* vol. 7, no. 4, pp. 385-398 April 1996.

[5] J. Fu and J.H. Patel, "Data prefetching in multiprocessor vector memories," in *Proc. of the 18th Annual International Symposium on Computer Architecture*, pp. 54-63, May 1991.

[6] J. Fu, J.H. Patel, and B.L. Janssens, "Stride directed prefetching in scalar processors," in *Proc. of the 25th Annual International Symposium on Microarchitecture*, pp. 102-110, 1992

[7] J.D. Gindele, "Buffer block prefetching method," *IBM Technical Disclosure Bulletin,* vol. 20, pp. 696-697, July 1977.

np-54

[8] A.J. Smith, "Sequential program prefetching in memory hierarchies," *IEEE Computer*, vol. 11, no. 12, pp.7-21, Dec. 1978

Formatted 13:38, 20 November 1999 from pren˙sem.
np-54