Multiple Branch Prediction on Dynamically Scheduled Processors

David M. Koppelman

Electrical & Computer Engineering Department

Louisiana State University

Baton Rouge, LA U.S.A.

Notes on These Slides

2

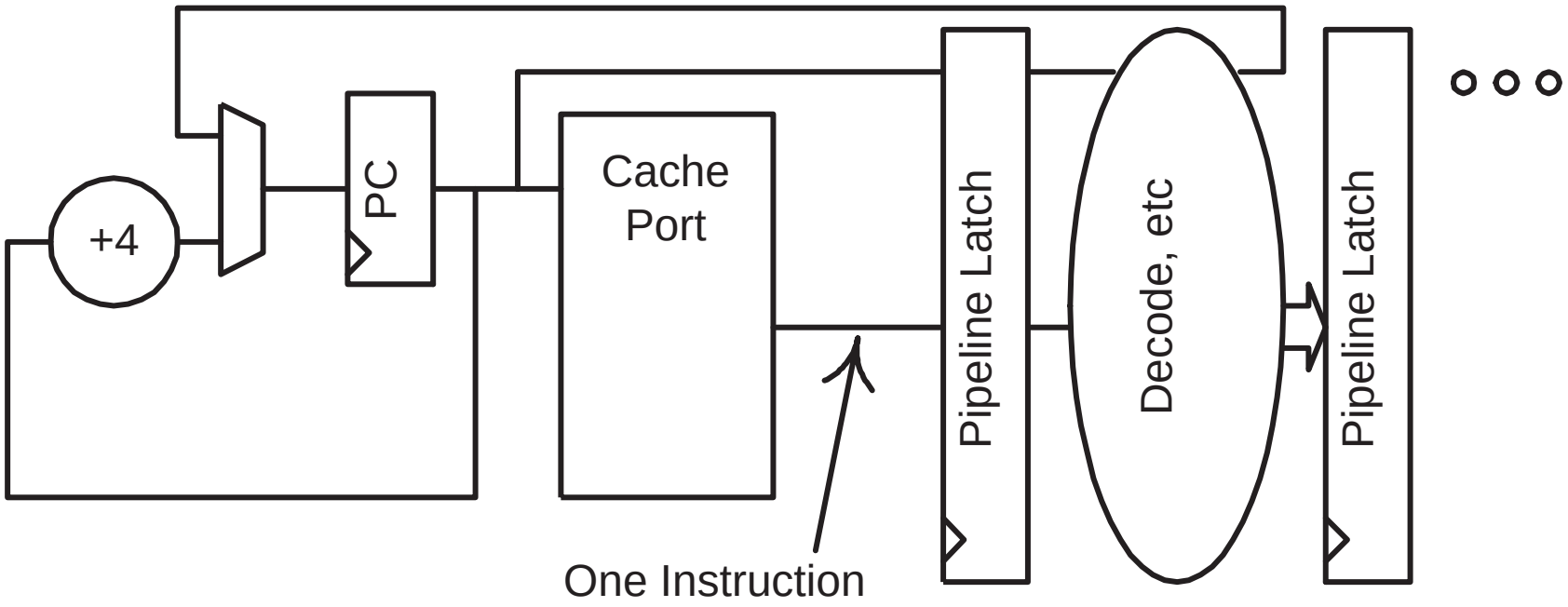
Slides for presentation made at the Workshop on Duplicating, Deconstructing, and Debunking, at the 29th International Symposium on Computer Architecture, 26 March 2002

Pages Added Since Presentation

- Plot key abbreviations.

Paper available via `http://www.ece.lsu.edu/koppel/pubs/bac_wddd.pdf`

Early RISC Processor Front End



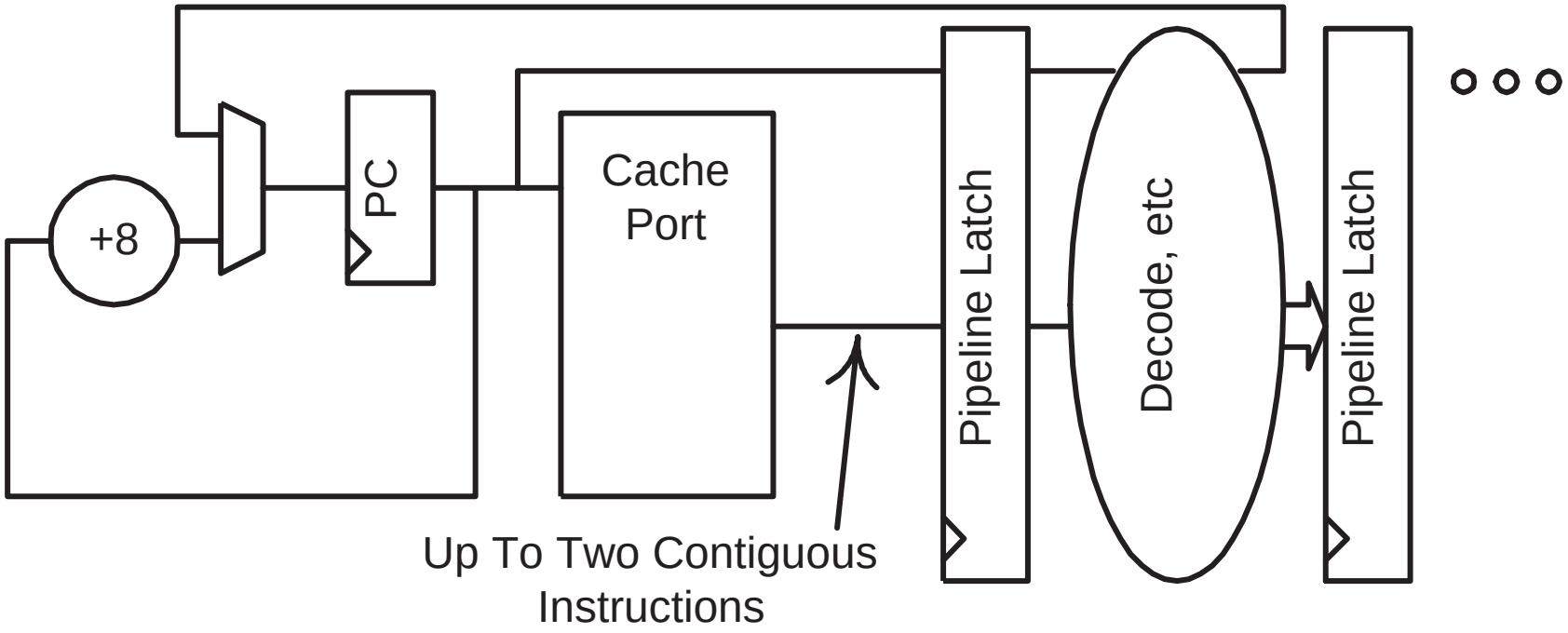
Fetches one instruction per cycle.

Branch targets resolved when needed.

Maximum execution rate $1\,\mathrm{IPC}$.

Two-Way Superscalar Processor Front End

4



Fetches two instructions per cycle.

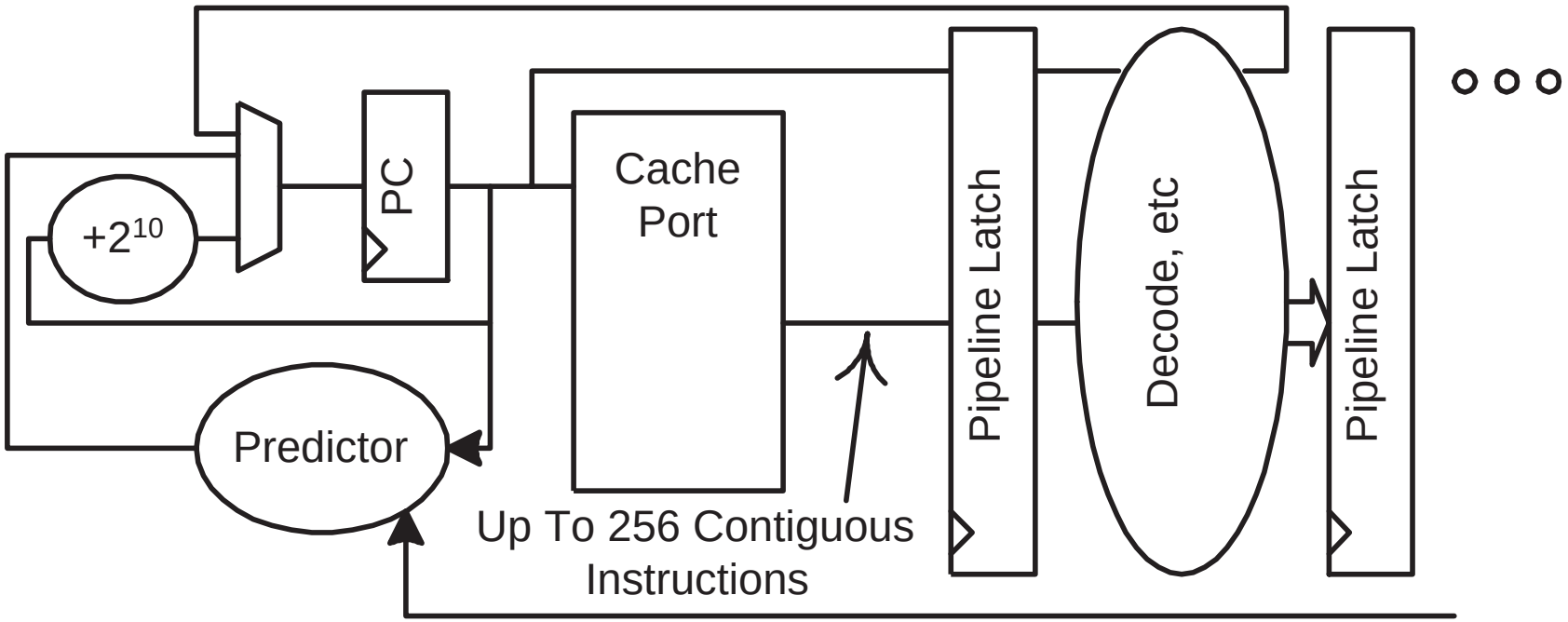
Will not get two useful instructions after taken branch in second slot.

Maximum execution rate $< 2\,\mathrm{IPC}$.

Branch resolution tricky.

Skipping ahead a bit . . .

Two-Hundred-Fifty-Six–Way Superscalar Front End (Imagine!)

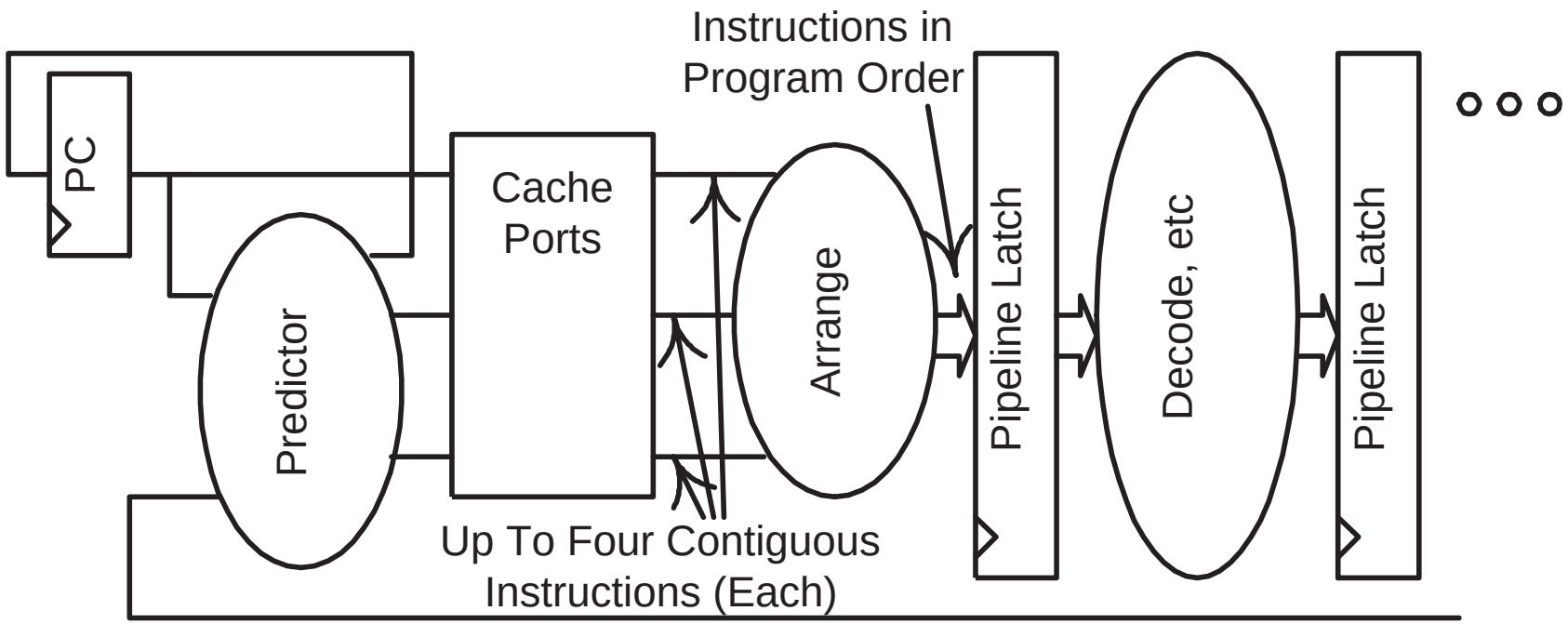


Fetches 256 *contiguous* instructions per cycle.

Number of instructions fetched limited by taken control transfers.

Maximum execution rate: $\ll 256\,\mathrm{IPC}$.

Generic MBP System



Example predicts three branches per cycle. . .

. . . and uses 3 port by 4 word instruction cache.

Multiple Branch Prediction

7

Multiple Branch Prediction:

Prediction of multiple branches per cycle.

In system using MBP, at rate of once per cycle:

- Predict direction and target of multiple branches.
- Retrieve instructions from non-contiguous addresses . . .
  . . . using multi-ported instruction cache.
- Arrange instructions in dynamic program order.

Need for Multiple Branch Prediction

8

With MBP execution rate not limited by taken control transfers!

Two-hundred-fifty-six way machines now possible!!

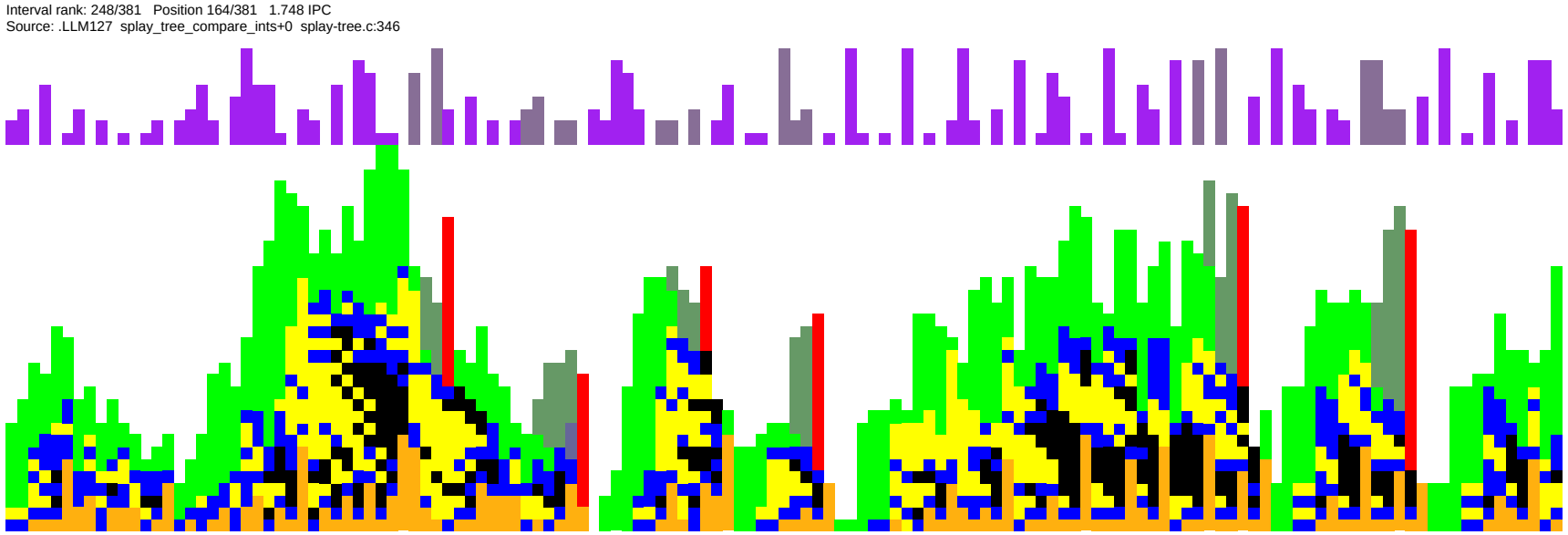
The **only thing** preventing feasible 256-way machines!!!:

- Available instruction level parallelism.
- Difficulty of exploiting available ILP.
- Memory system performance.

Reality Check: current machines' execution rate $<$ basic block size. . .
. . . so is additional fetch bandwidth really needed?

Fetch on a Conventional Eight-Way Superscalar Processor

Plot shows instructions in reorder buffer over time running gcc.



Key to plot on next page.

Reorder Buffer Visualization Key

Plot shows state/location of instructions in reorder buffer.

The x-axis shows time, y-axis shows reorder buffer position.

Purple: Instruction in first stage of decode.

Green: Too recently arrived to execute. (Decode, rename, schedule, dispatch, etc.)

Yellow: Waiting for an operand.

Blue: Executing or waiting for a functional unit.

Black: Completed execution (awaiting commitment).

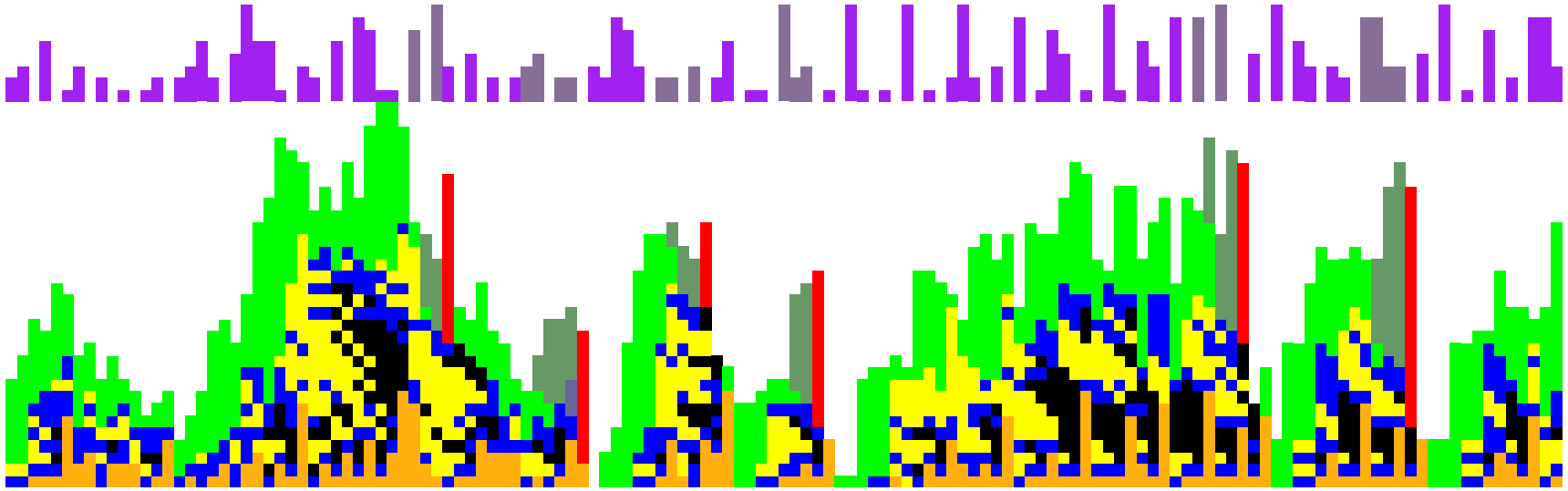
Orange: Committing

Red: Being squashed due to a misprediction or exception.

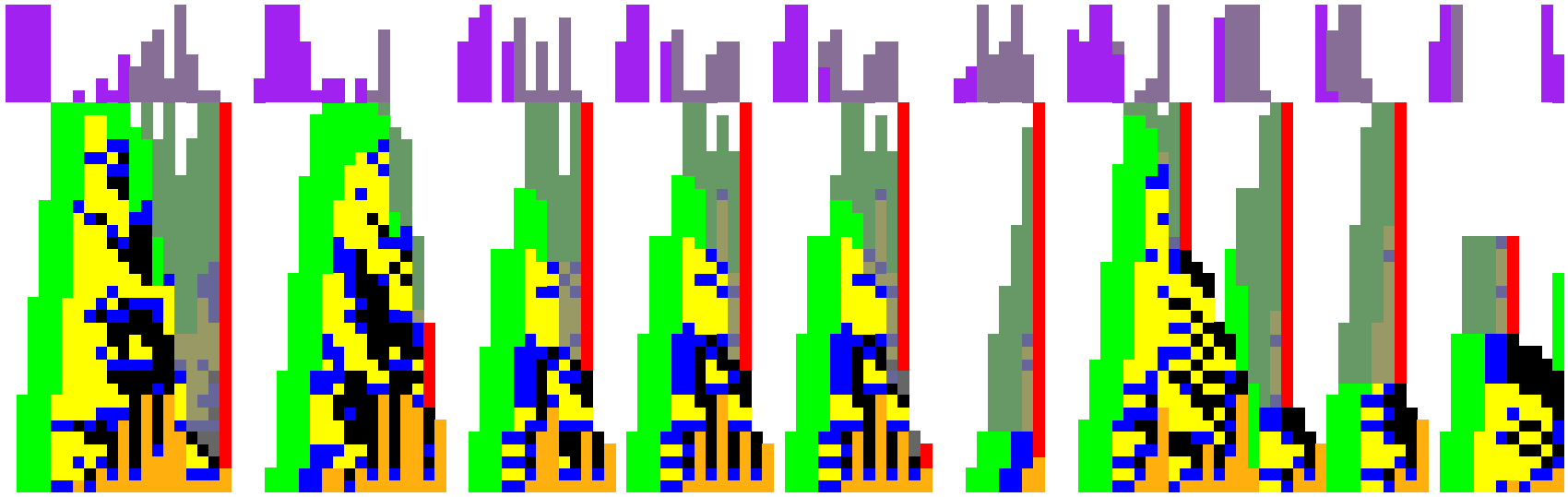
Grayed version of colors above: will be squashed.

Fetch on a MBP and Conventional Eight-Way Superscalar Processor

Interval rank: 248/381 Position 164/381 1.748 IPC
Source: .LLM127 splay_tree_compare_ints+0 splay-tree.c:346

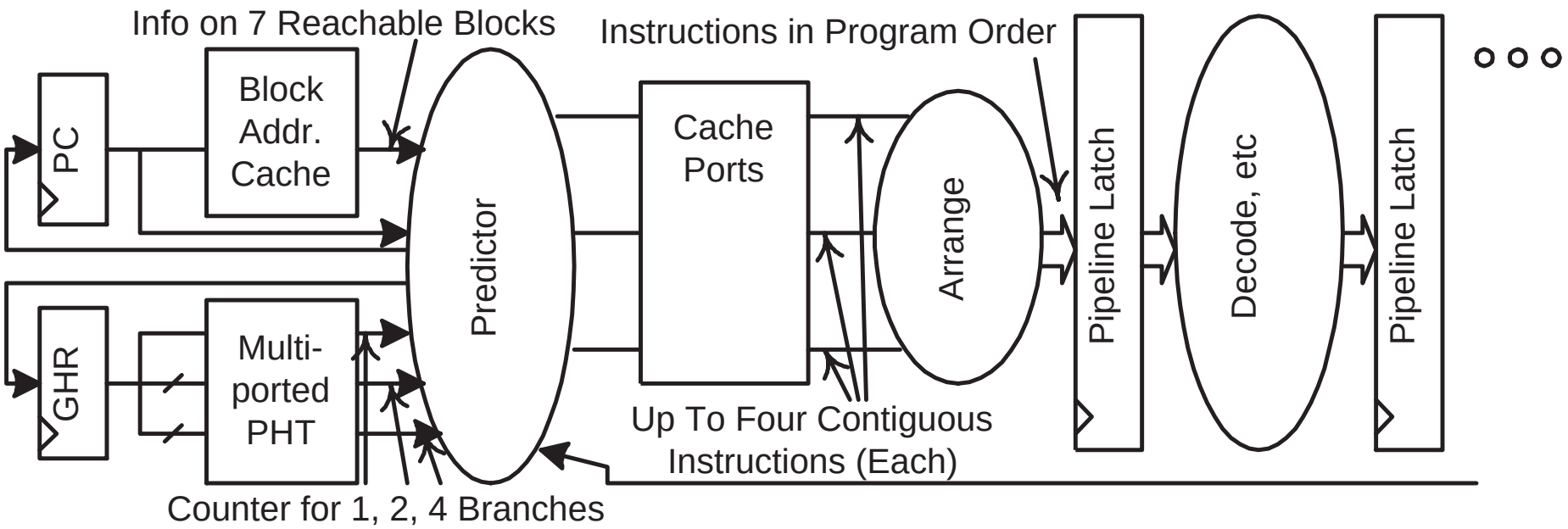


Interval rank: 196/361 Position 144/361 1.669 IPC
Source: .LLM127 splay_tree_compare_ints+0 splay-tree.c:346



MBP of Yeh, Marr, and Patt (ICS 1993)

12



Store block information in *block address cache* (BAC).

PC-indexed BAC entry holds information on blocks reachable from PC.

Block information includes length, CTI type, and target.

Order is height of “tree” stored in BAC.

Predict direction using modified correlating predictor.

BAC and PHT are read using addresses available at beginning of cycle.

Several other predictors described by Yeh et al.

12

Evaluation of MBP

13

Yeh, et al evaluated ability to supply instructions. . .
. . . but not ability of system to use them.

Unanswered Questions:

- Additional performance gained by faster fetch.
- Second order effects, such as realistic predictor table updates.
- Equal-size comparison of different orders.

Interestingly, much follow-on work by others also ignored rest of system.

Special Cases

14

Order-1 MBP (MBP-1).

Predicts just one branch per cycle.

Does not need a multi-ported cache.

Described by Yeh and Patt (ISMA 1992).

Superblock Predictor

Superblock ends with any control transfer. . .
. . . and may contain highly biased not-taken branches.

Like MBP-1 but BAC entry describes superblock.

Extends maximum fetch rate beyond basic block size.

15

Reinman, Calder, Austin Superblock Predictor (ISCA 1999).

Enqueues predicted fetch addresses:

Can use multiple-ported cache (when predictor gets ahead of cache).

Queued predictions hide BAC (FTB) misses.

Can be used to prefetch cache, etc.

Simulated entire system, tested against order-1 MBP.

Looked at size/latency tradeoffs.

Found better performance when BAC storage limited.

16

⋆⋆⋆ Duplication ⋆⋆⋆

Compare equal-cost (very roughly) conventional, superblock and MBP systems.

Common Features

- Aggressive Dynamically Scheduled Eight-Way Superscalar System
- Large YAGS branch predictor on all systems. (Minor differences.)
- Indirect jump and return address predictors.

Updated version of MBP

- Queuing to hide some misses.
- Added stages to minimize critical path impact.

Superblock Predictor

- Timing very similar to MBP system.
- Use confidence estimator to extend superblock.

Configuration Parameters

17

Common Parameters	Value
Decode Width	8-way Superscalar
Reorder Buffer	256 instructions
Return-Address Stack	8 entries
L1 ICache	256-B Line
L1 DCache Hit Latency	1 cycle
L2 DCache	8-way,64-B Line, 256 KiB
L2 Hit Latency	10 cycles
L2 DCache Miss Latency	$\approx$ 100 cycles
L1 ICache Ports	4.
ID to EX	9 cycles.
Global History	16 branches
Integer Units	8
Floating-Point Units	4
Memory Units	4
Base Configuration	**Value**
L1 ICache (MBP,super)	4-way, 64 kiB
L1 ICache (Conv)	7-way, 112 kiB
FTB,BAC	2^{13} nodes
L1 DCache (Conv)	4-way, 64 kiB
Large Configuration	**Value**
L1 ICache (MBP and super)	4-way, 256 kiB
L1 ICache (Conv)	7-way, 448 kiB
FTB,BAC	2^{15} nodes
L1 DCache (Conv)	4-way, 256 kiB

18

YAGS (Eden and Mudge, ISMA 1998).

PC-indexed table makes primary prediction using 2-bit counter.

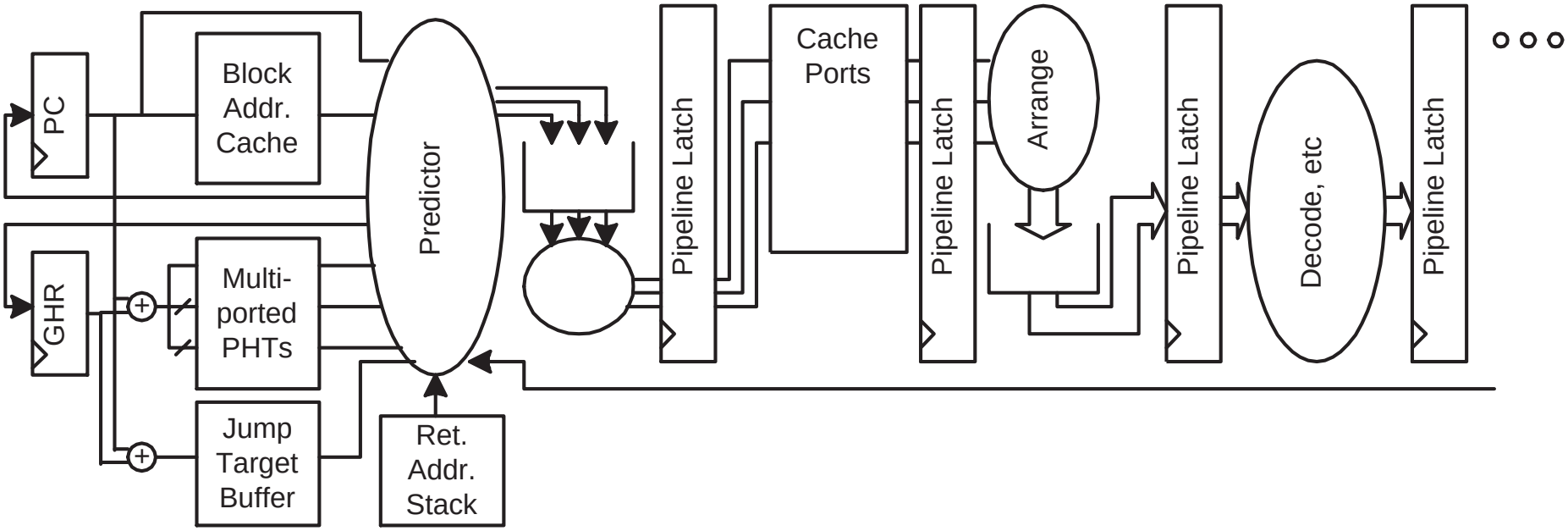
Primary can be overridden by one of two tagged GHR$\oplus$PC-indexed PHTs.

Primary warms up quickly.

Tagging limits use (and size) of PHTs.

Simulated MBP

19



Cannot use BAC and JTB in same cycle.

Single block can span multiple fetch ports and fetch groups.

Use YAGS predictor using single PC for all blocks.

19

Simulator

20

Simulated on RSIM (Pai 1997).

Simulates a dynamically scheduled SPARC V8 implementation.

Ran integer benchmarks from SPECcpu suite, including SPEC2000.

Miscellaneous Terminology

ROB:

Reorder buffer, queue holding instructions between decode and commit.

Fetch Cycle:

Cycle in which system can accept instructions for decoding. (ROB not full, etc.)

Abbreviations Used in Execution Rate Plots

21

In/Pred:

Number of predicted instructions divided by number of cycles in which a prediction made. (For conventional system, instructions per fetch.)

In/Fetch:

Number of instructions fetched divided by number of instruction cache accesses.

Dcd-In/Ftch-Cyc:

Fetch rate: number of decoded instructions divided by number of fetch cycles (cycles during which system can accept instructions because ROB not full, etc.)

Cmt-In/Fetch-Cyc:

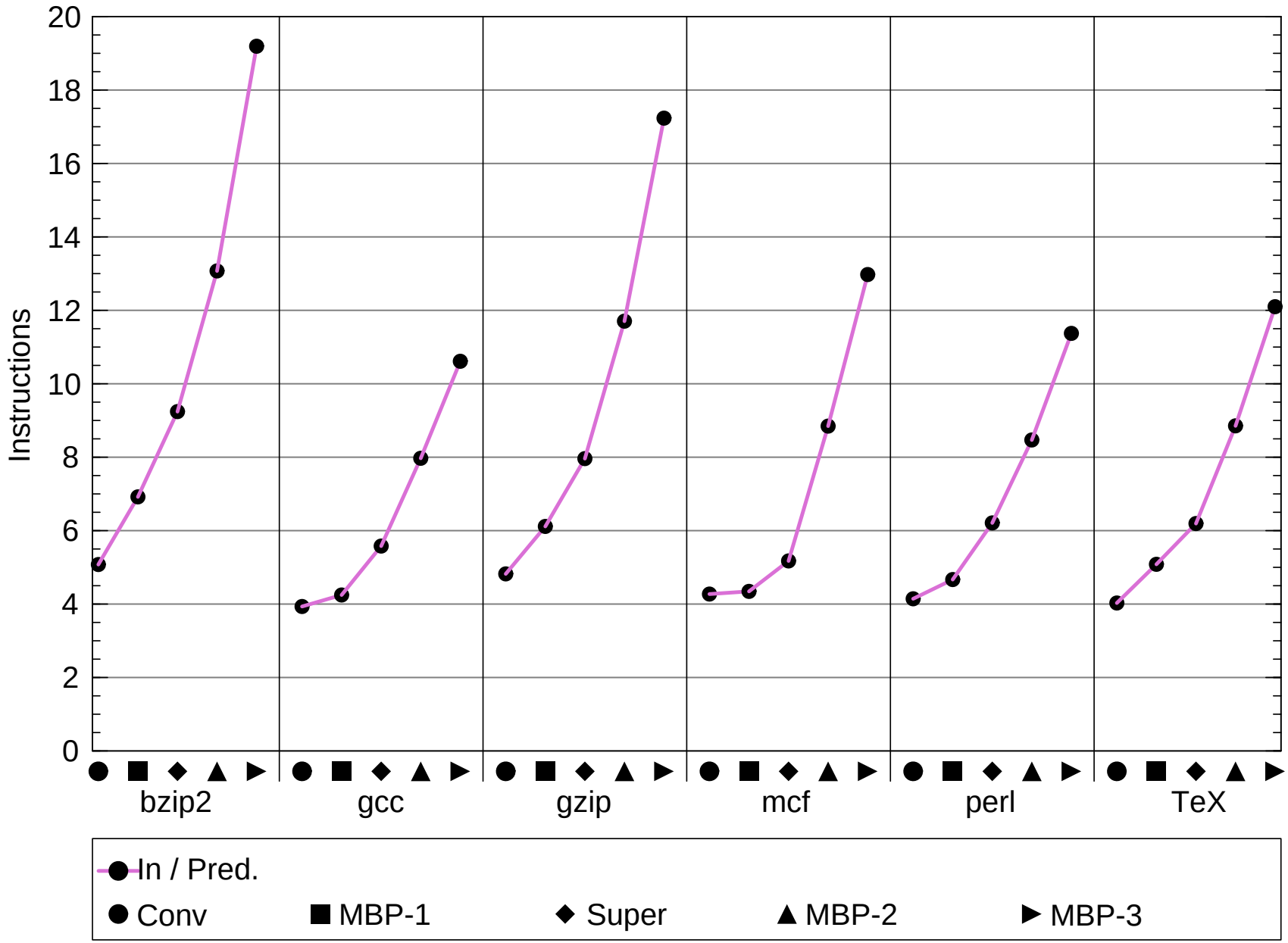
Effective fetch rate: number of committed instructions divided by number of fetch cycles (cycles during which system can accept instructions because ROB not full, etc.)

Exec:

Execution rate, IPC: number of committed instructions divided by number of cycles taken to run program.

Instructions per Prediction

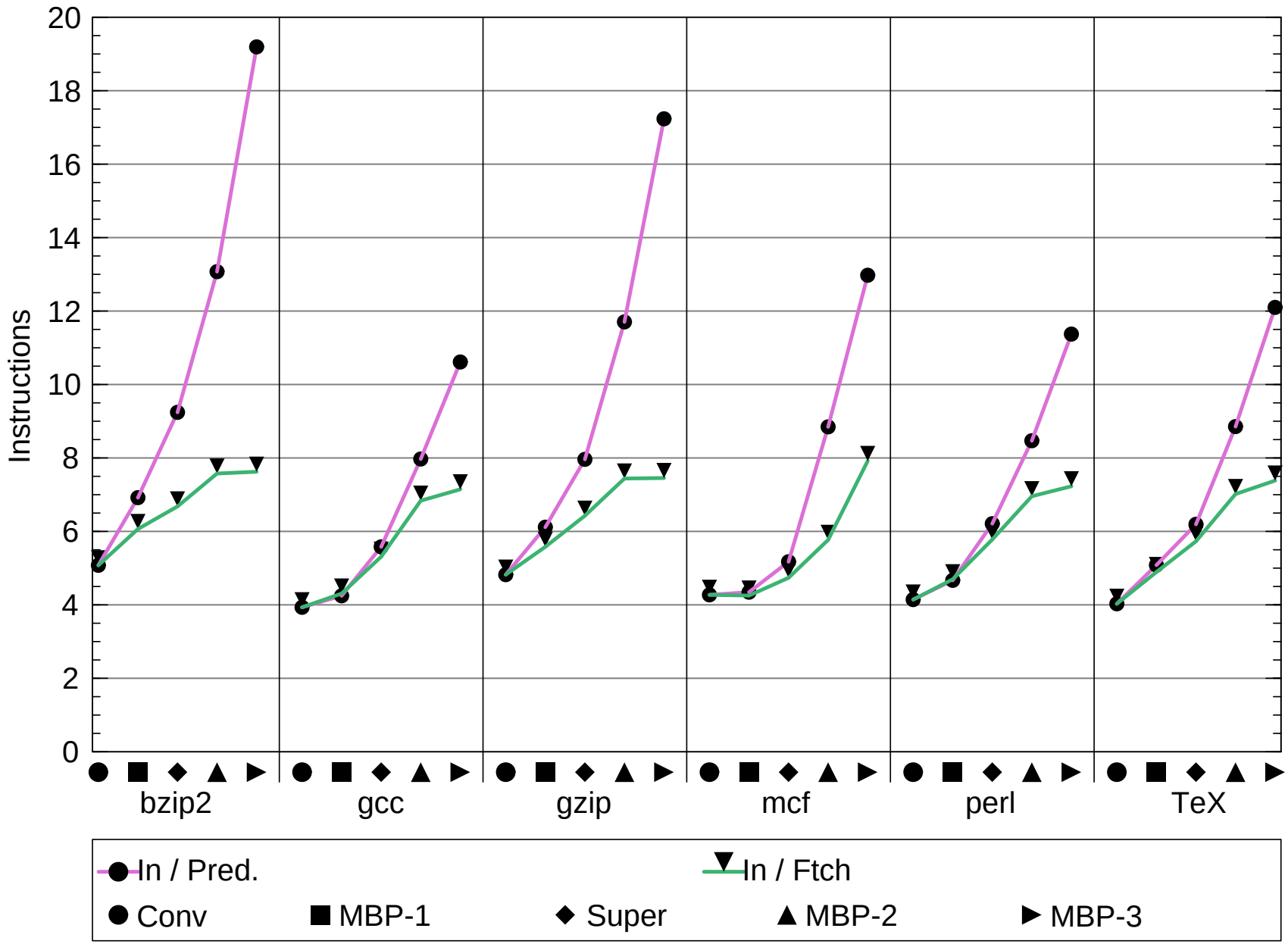
22



22

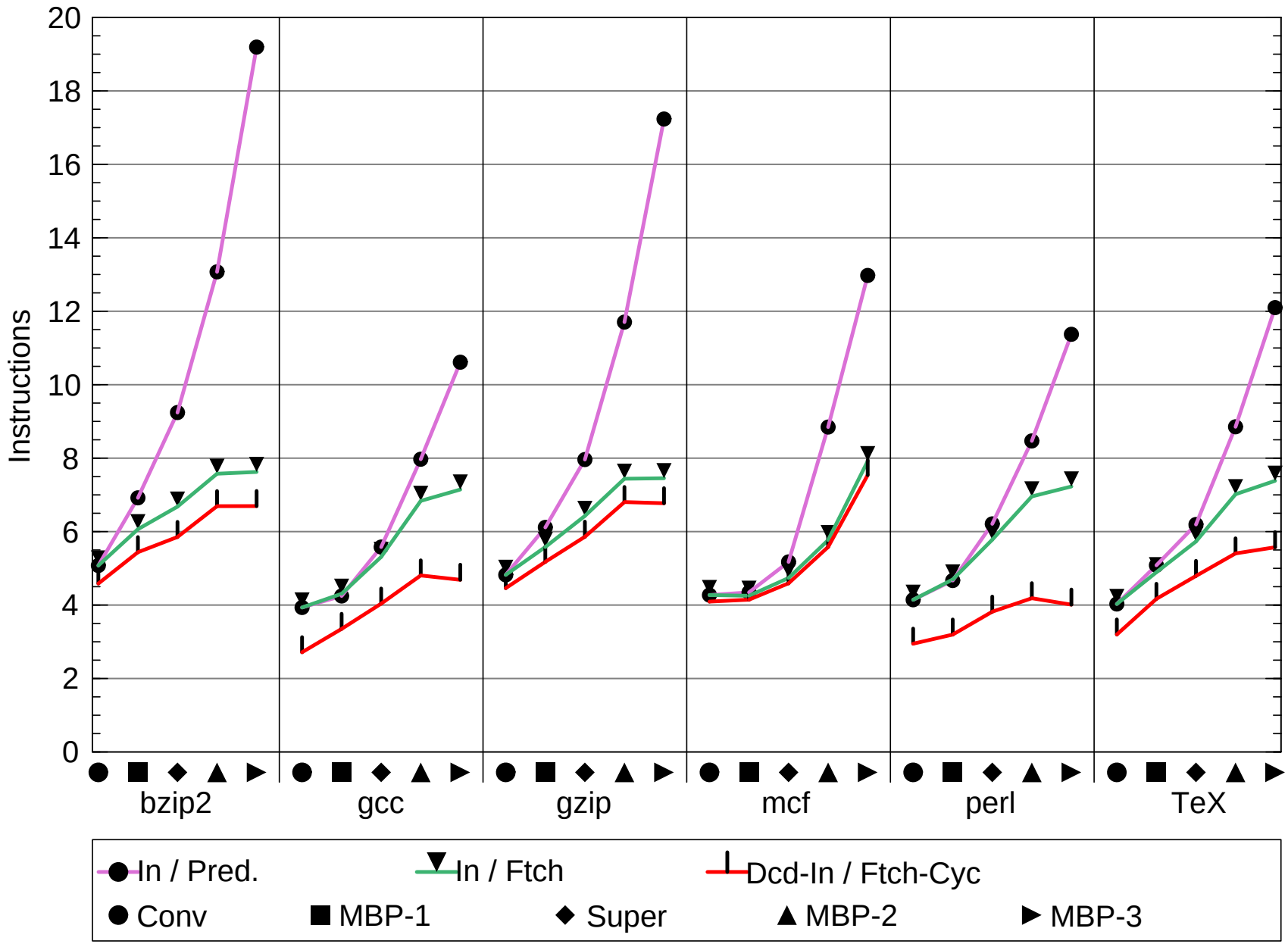
Instructions per Fetch (↓Cache Bandwidth, ↓Port Limit)

23

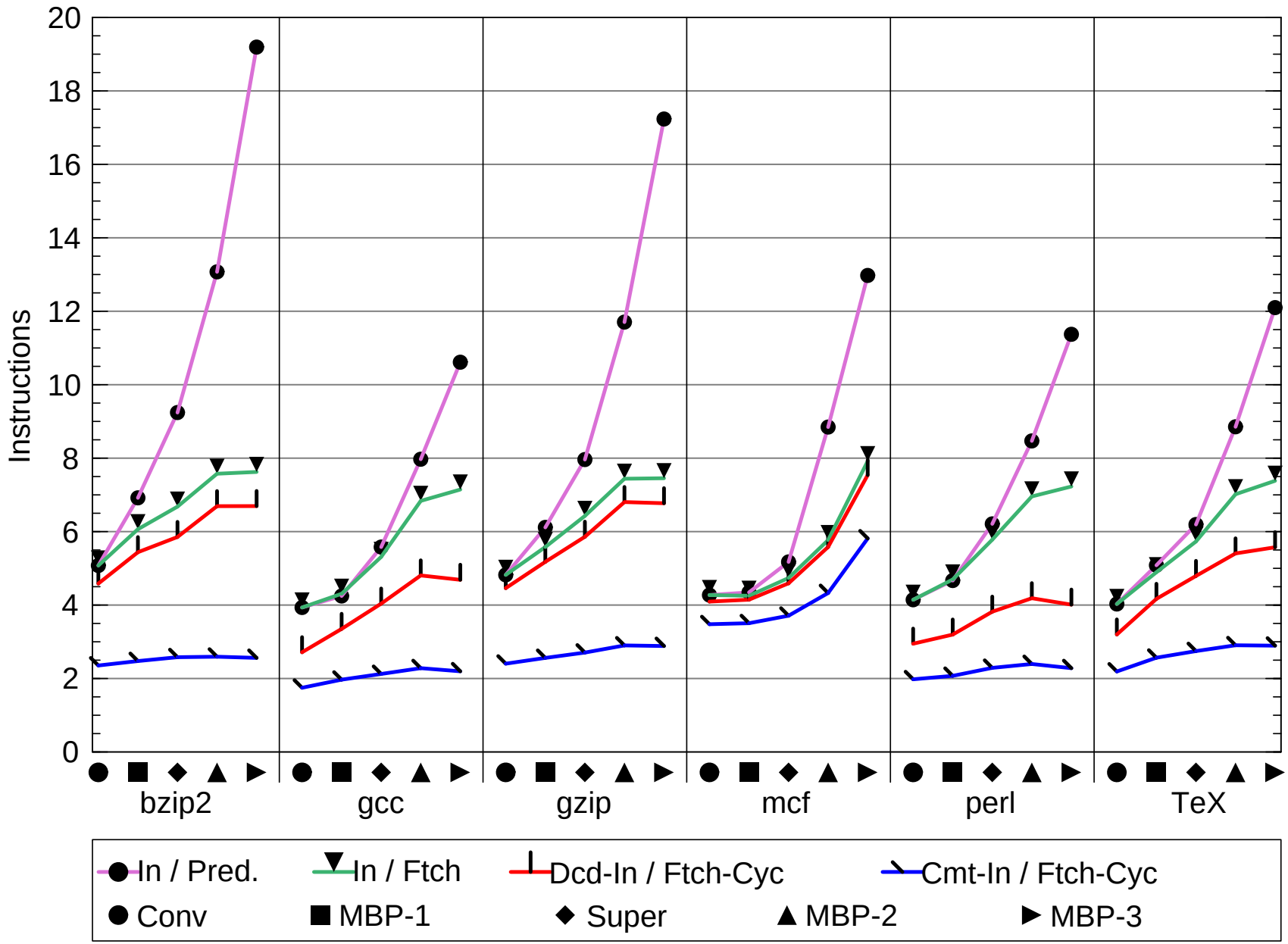


23

Instructions Decoded Per Fetch Cycle (↓ICache Misses)

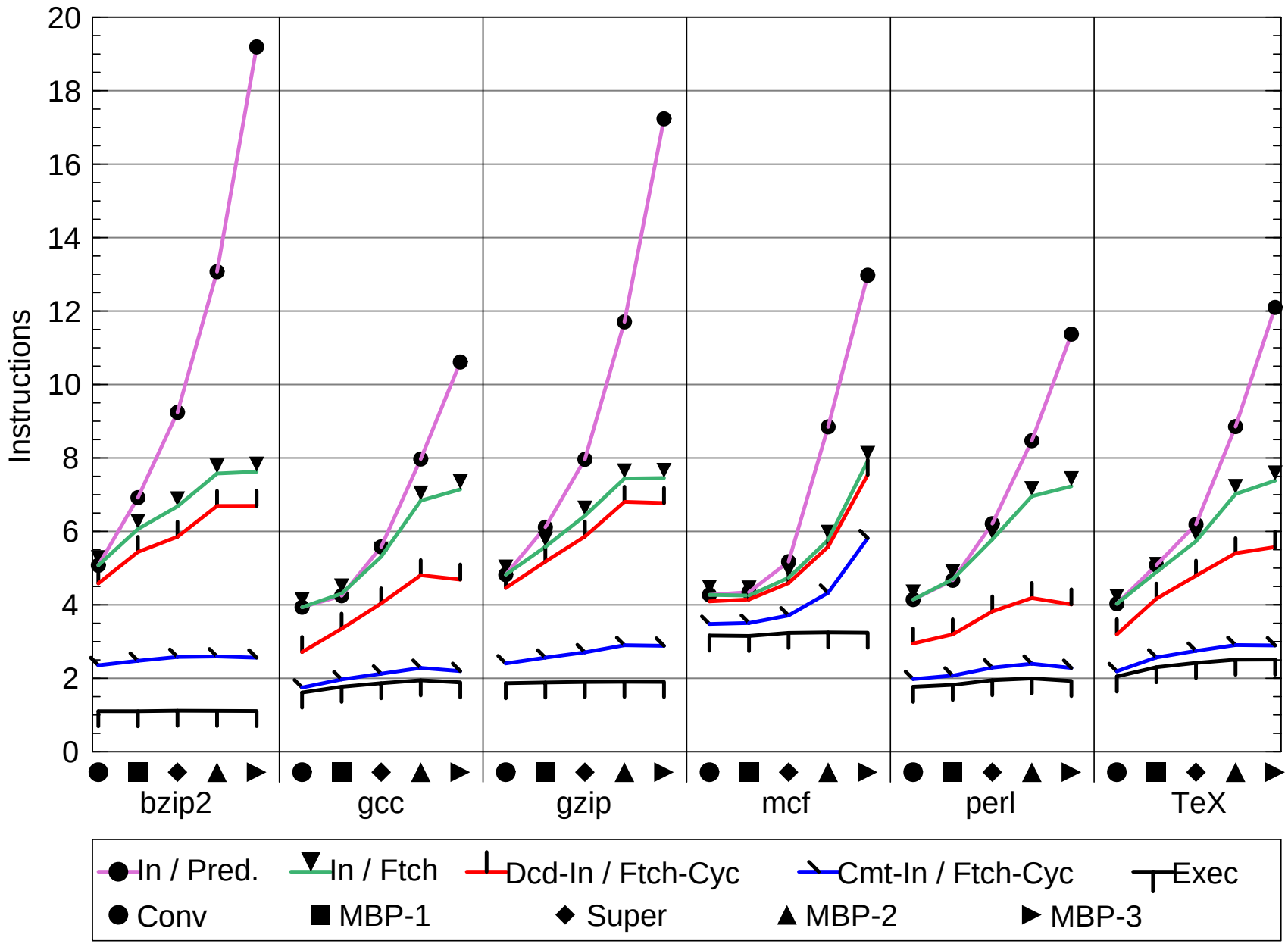


Committed Instructions Per Fetch Cycle (↓Mispred., ↓Exceptions)



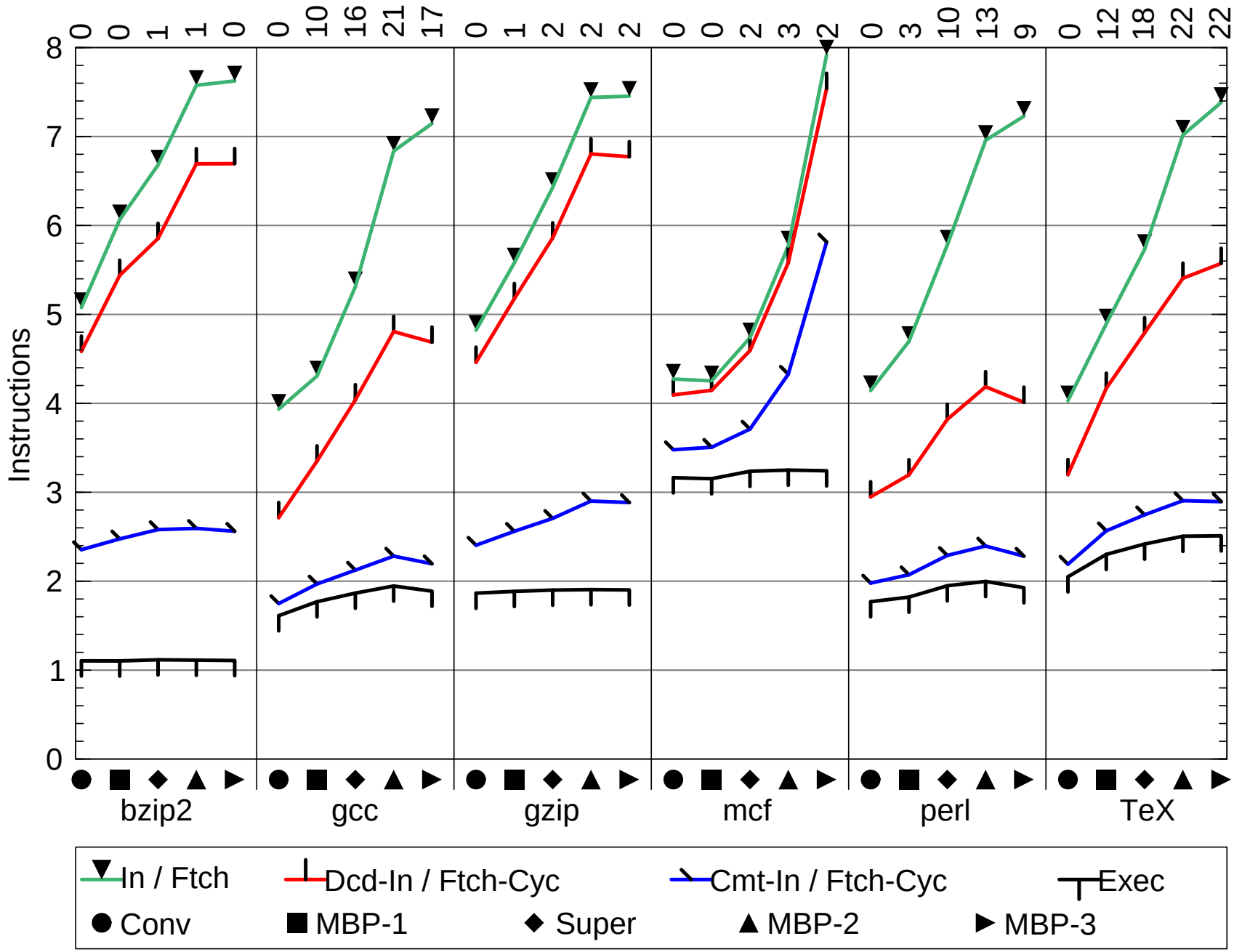
Bottom Line: Execution Rate (↓ILP [ROB Full])

26



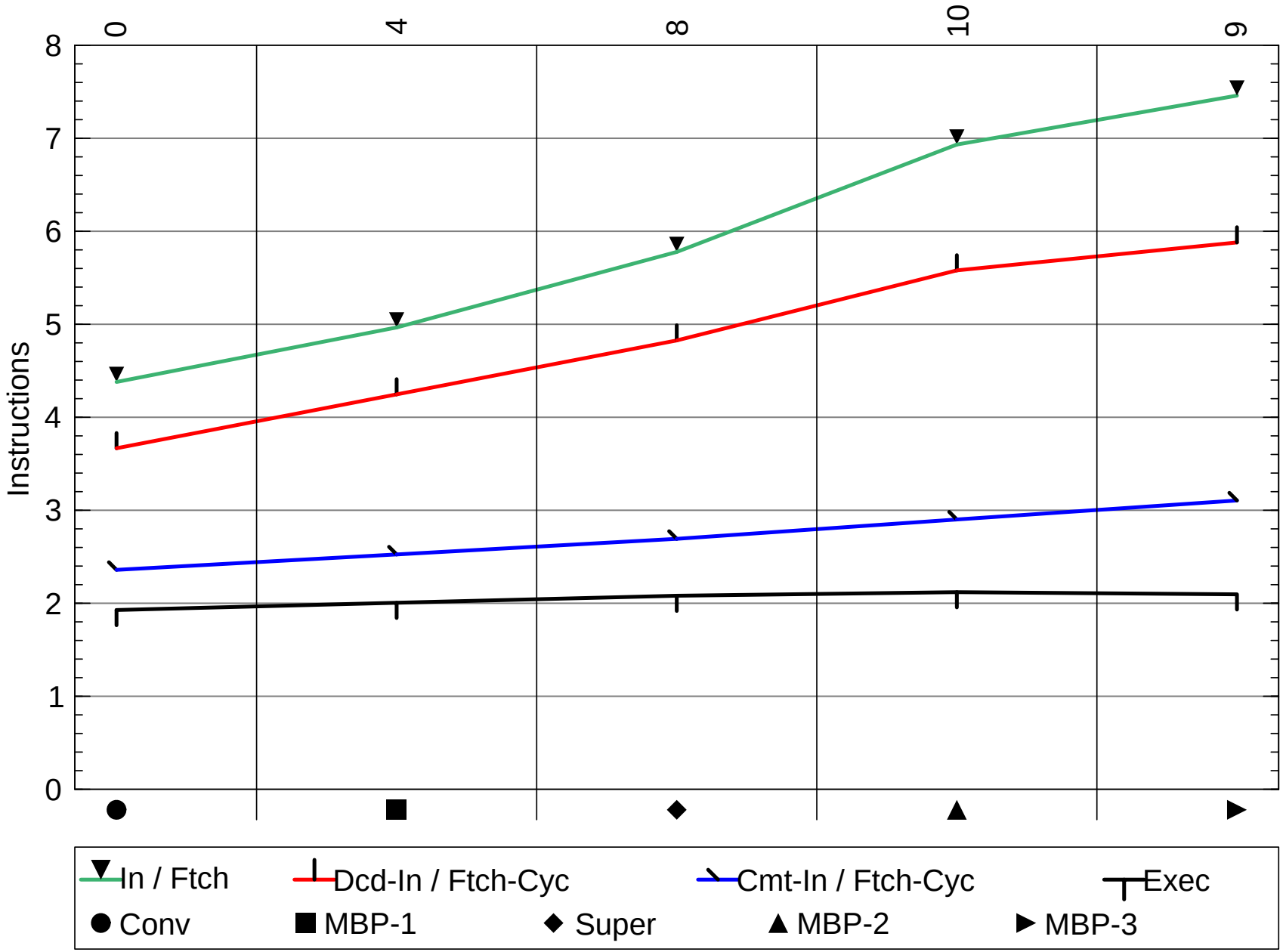
26

Speedup (without instructions per prediction)



Speedup average.

28



28

Abbreviations Used in Fetch Slot Usage Graphs

29

An n-way system has n decode slots. . .

. . . plot shows what they contained or why they are empty.

Top of plot shows branch misprediction rate, full segment is 10%

Cmt:

Slot holds instruction that will commit. Height of this segment is execution rate (IPC).

Unfilled:

Slots that are unfilled because the fetch mechanism returned less than eight (in this case) but more than zero instructions that later ripen.

Squash:

Slots holding wrong-path instructions that ripen (and are later squashed).

IC Miss:

Slots unfilled due to instruction cache misses.

ROB Fl:

Slots whos contents can not advance due to a full reorder buffer or some other resource limit. (This counts empty slots.)

Bbl:

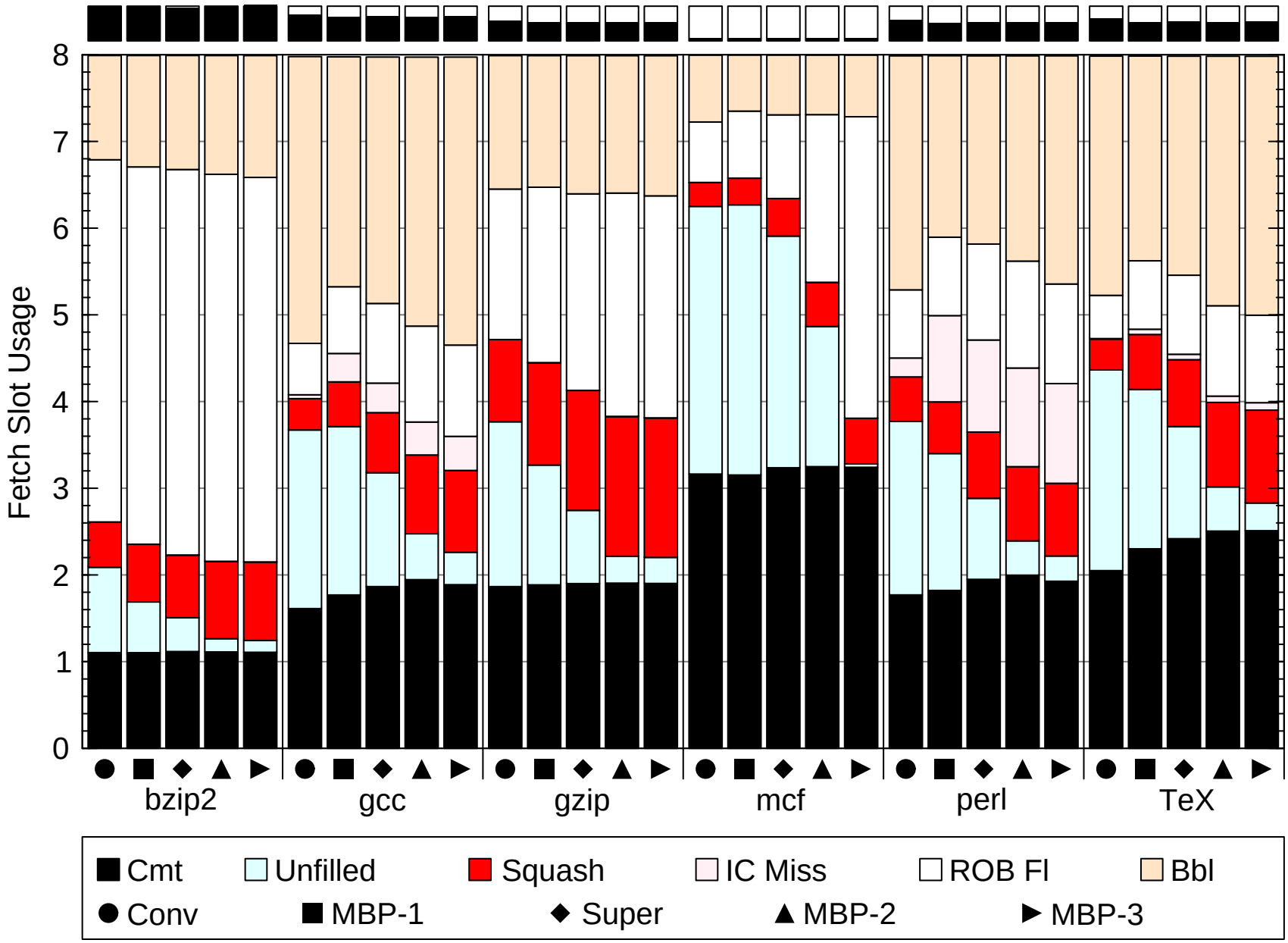
Slots holding wrong-path instructions that are squashed before they ripen. Also under

30

this category are unfilled slots fetched with the wrong-path instructions and delays due to unpredicted jumps and next-line-predictor mispredictions.

30

Fetch Slot Usage

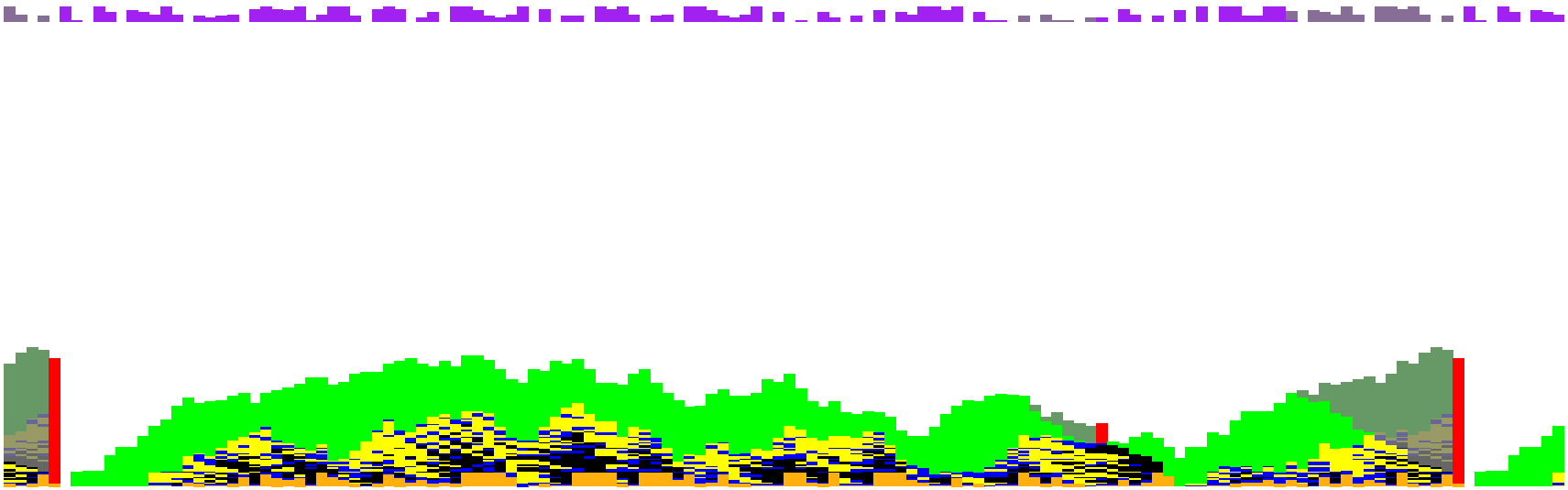


Execution Samples, Perl

32

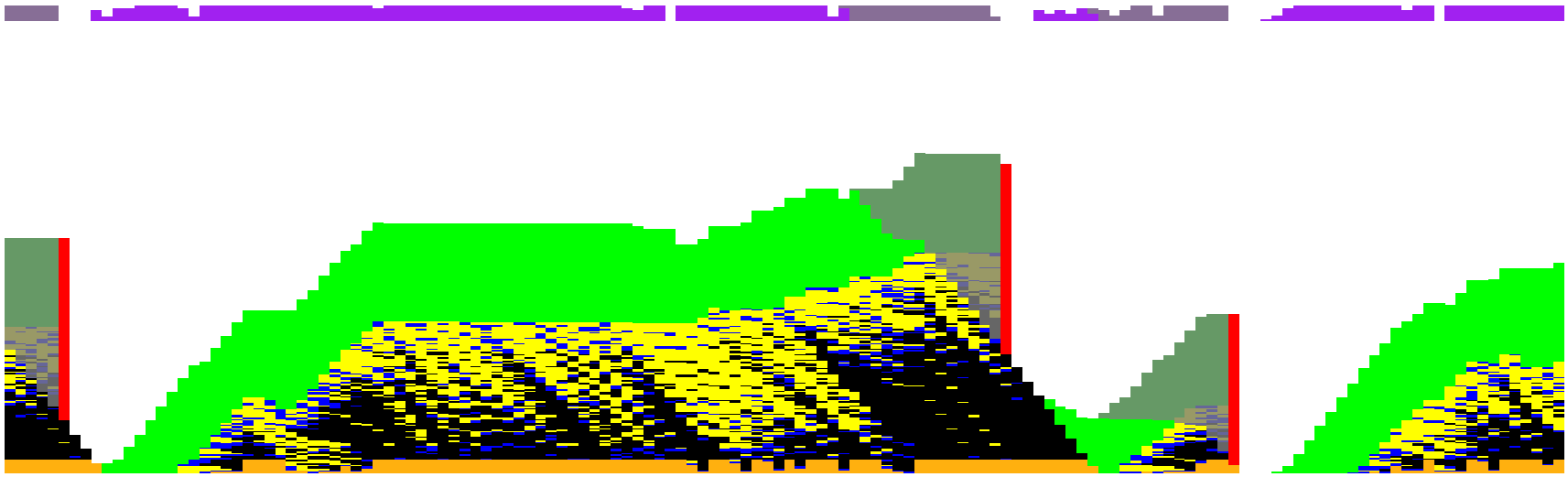
Conventional

Interval rank: 106/111 Position 83/111 3.682 IPC
Source: S_regmatch+3670



MBP-2

Interval rank: 91/98 Position 63/98 6.274 IPC
Source: S_regmatch+125



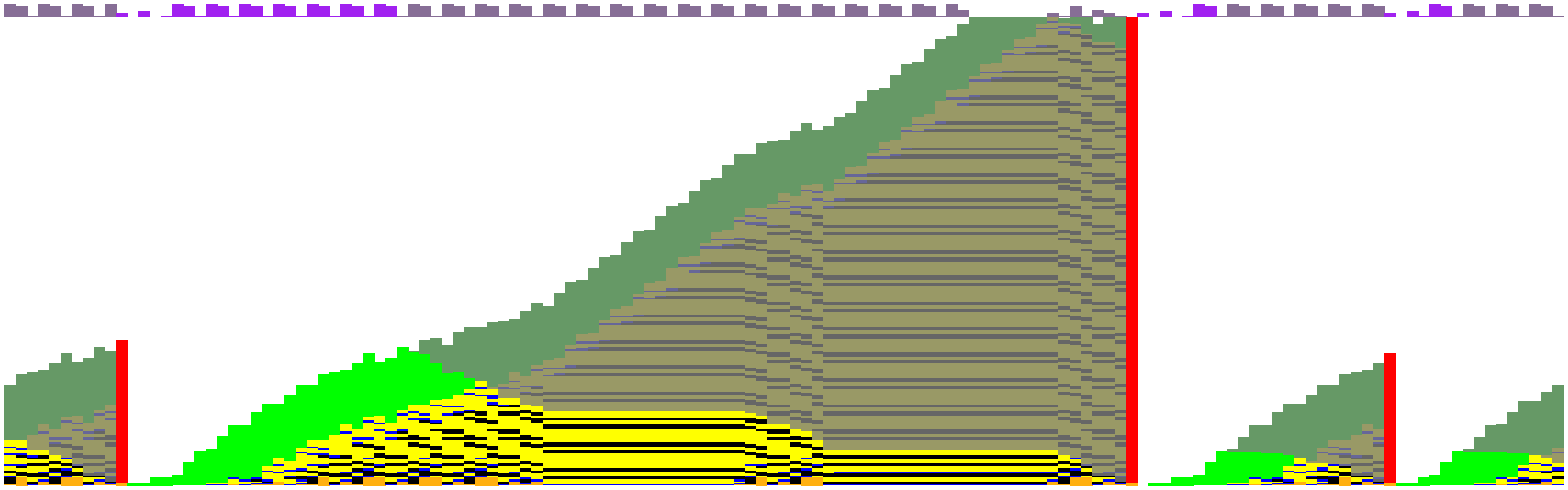
32

Execution Samples, gzip

33

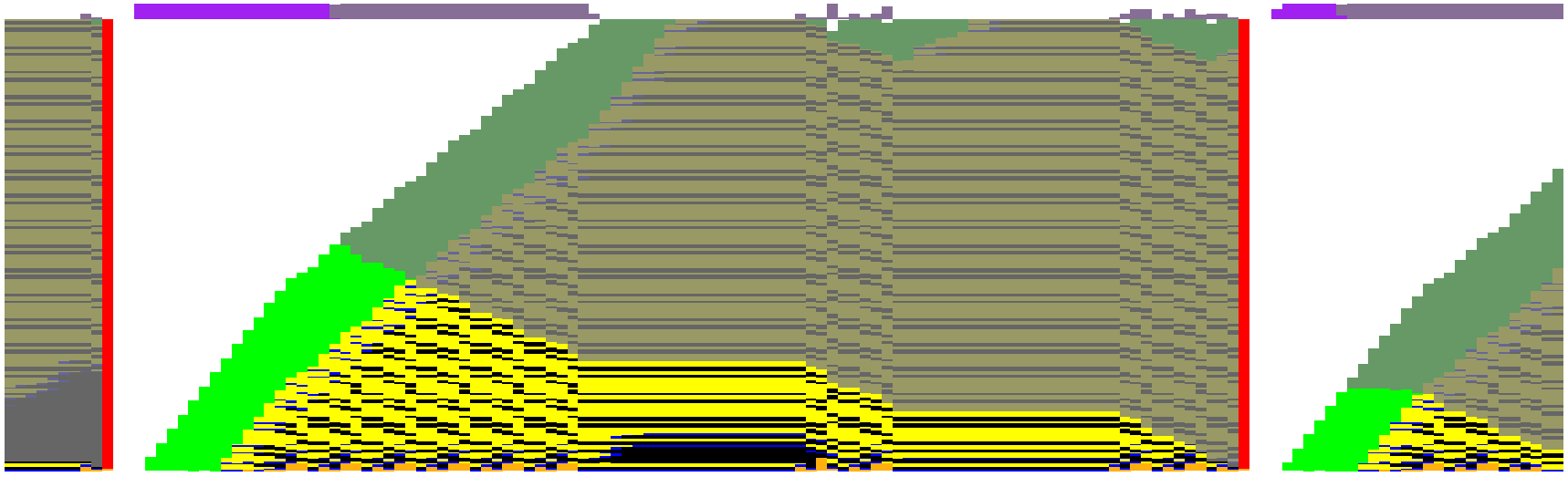
Conventional

Interval rank: 70/161 Position 46/161 1.786 IPC
Source: longest_match+125



MBP-2

Interval rank: 25/157 Position 122/157 1.086 IPC
Source: longest_match+130



33

Fetch Effectiveness

34

Ability to predict basic blocks: Very good.

Small BAC sufficient even for large programs.

Intermediate performance from superblock predictor.

34

35

⋆⋆⋆ Deconstruction ⋆⋆⋆

Yes, we can fetch faster. . .
. . . but why does faster fetching mean faster committing?

Performance Bound

Determine execution rate using. . .
. . . observed CTI mispredict rate and resolution time. . .
. . . and unlimited ILP.

Commit Rate Sensitivity

How much does larger window size (ROB occupancy) help?

Look at instructions blocking head of ROB.

Also look at different configurations. (If I talk fast enough.)

Performance of Fetch Mechanisms with Ideal Commit

36

Let i_{c} denote the number of committed instructions.

Let $i_{\mathrm{m.b}}$ denote the number of committed mispredicted branches.

Let $i_{\mathrm{m.j}}$ and $i_{\mathrm{m.r}}$ denote the number of committed mispredicted jumps and returns.

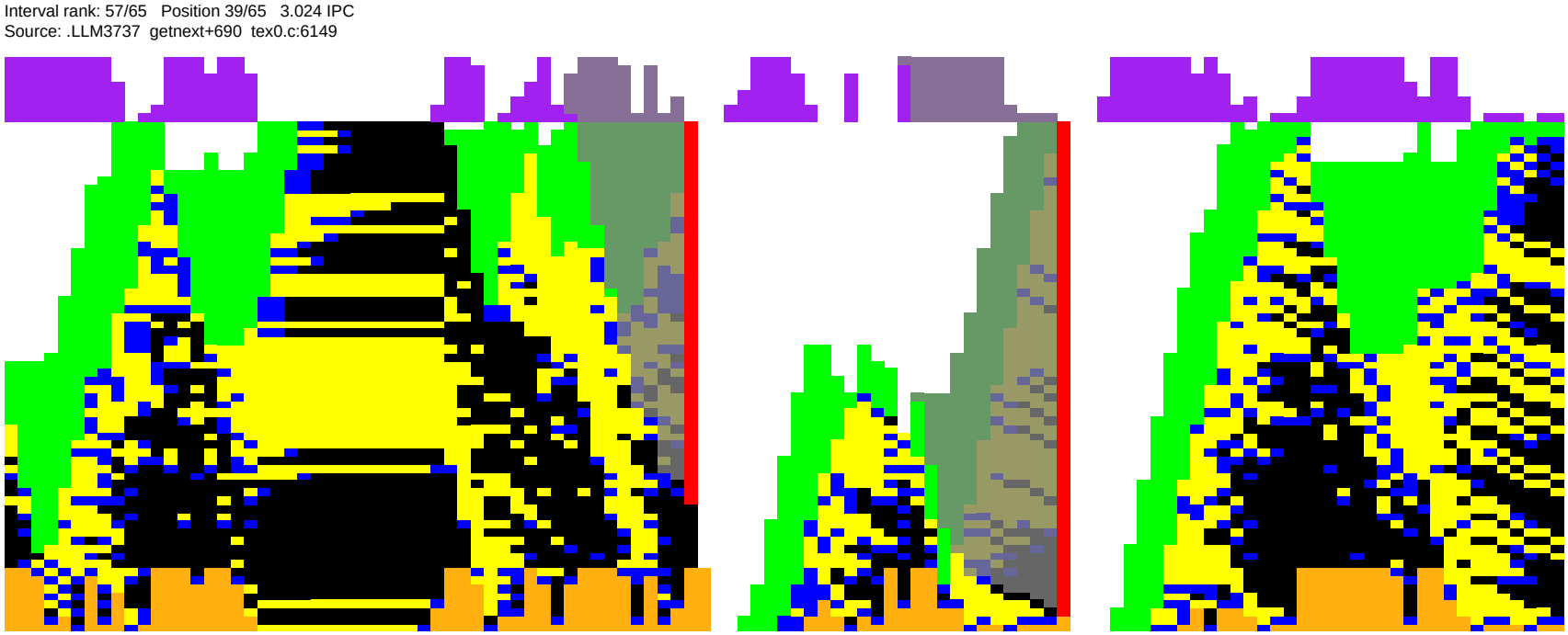
Let $t_{\mathrm{r.b}}$, $t_{\mathrm{r.j}}$, $t_{\mathrm{r.r}}$ denote the average observed branch, jump, or return resolution time.

Let r_{f} denote the observed fetch rate.

Ideal execution rate: $$r_{\mathrm{x.ideal}} = \frac{i_{\mathrm{c}}}{\frac{i_{\mathrm{c}}}{r_{\mathrm{f}}} + i_{\mathrm{m.b}} t_{\mathrm{r.b}} + i_{\mathrm{m.j}} t_{\mathrm{r.j}} + i_{\mathrm{m.r}} t_{\mathrm{r.r}}}$$

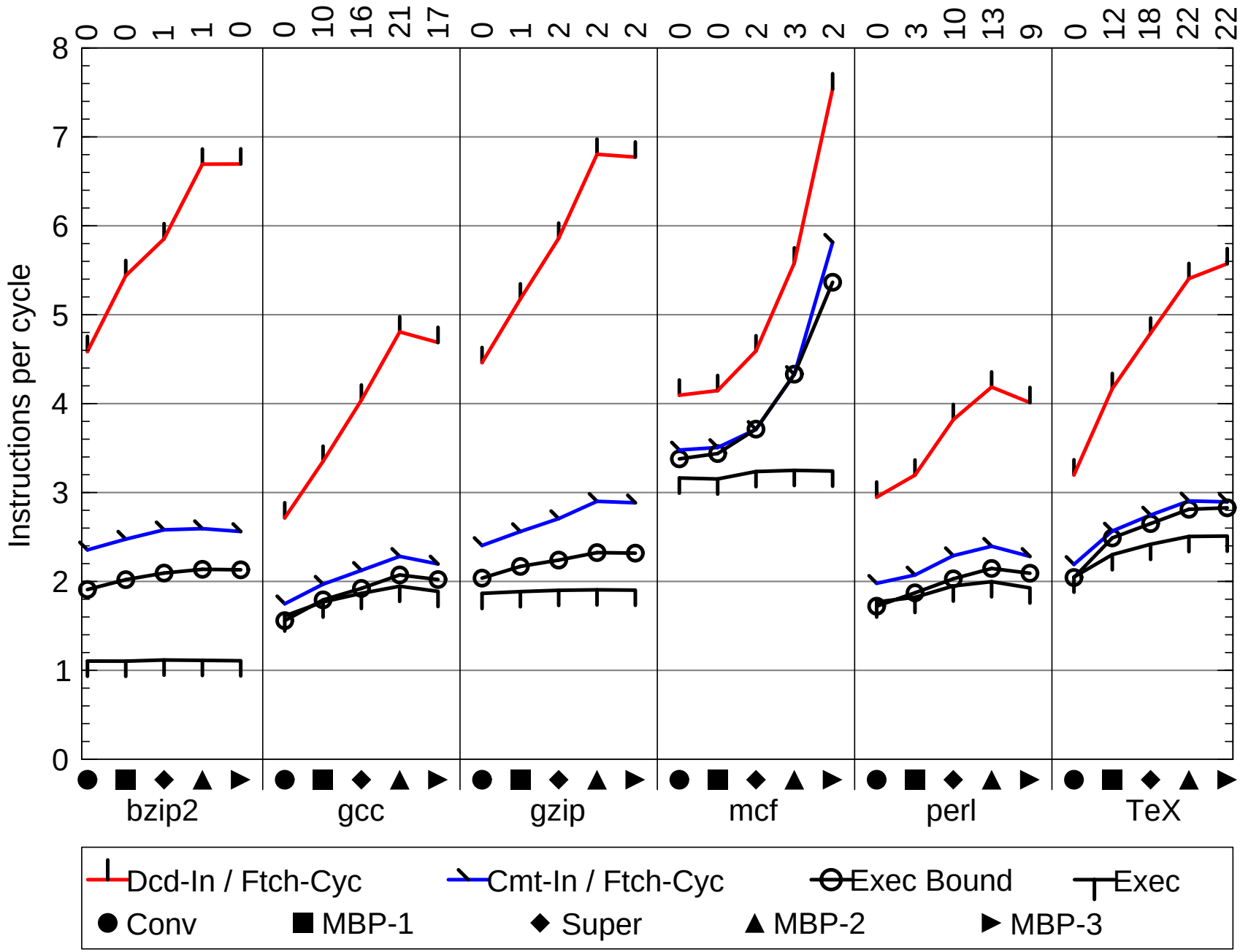
For code in which $r_{\mathrm{x}} = r_{\mathrm{x.ideal}} \ldots$
. . . reorder buffer never fills *with correct path instructions*.

Illustration for Explaining Performance Bound



Execution Rate Assuming Ideal Commit

38



38

39

Based on ideal execution rate:

Benchmarks gcc and perl:

Achieve ideal execution rate. (ROB never fills *with correct-path instructions*.)

Benchmark mcf

Far from ideal: rob fills.

Commit Rate

40

Commit Rate:

Number of committed instruction divided by ripe cycles.

Ripe:

An instruction that's old enough to execute. Decoded and been in the scheduler long enough.

$r_{\mathrm{c}} = \frac{i_{\mathrm{c}}}{t_{\mathrm{r}}}$.

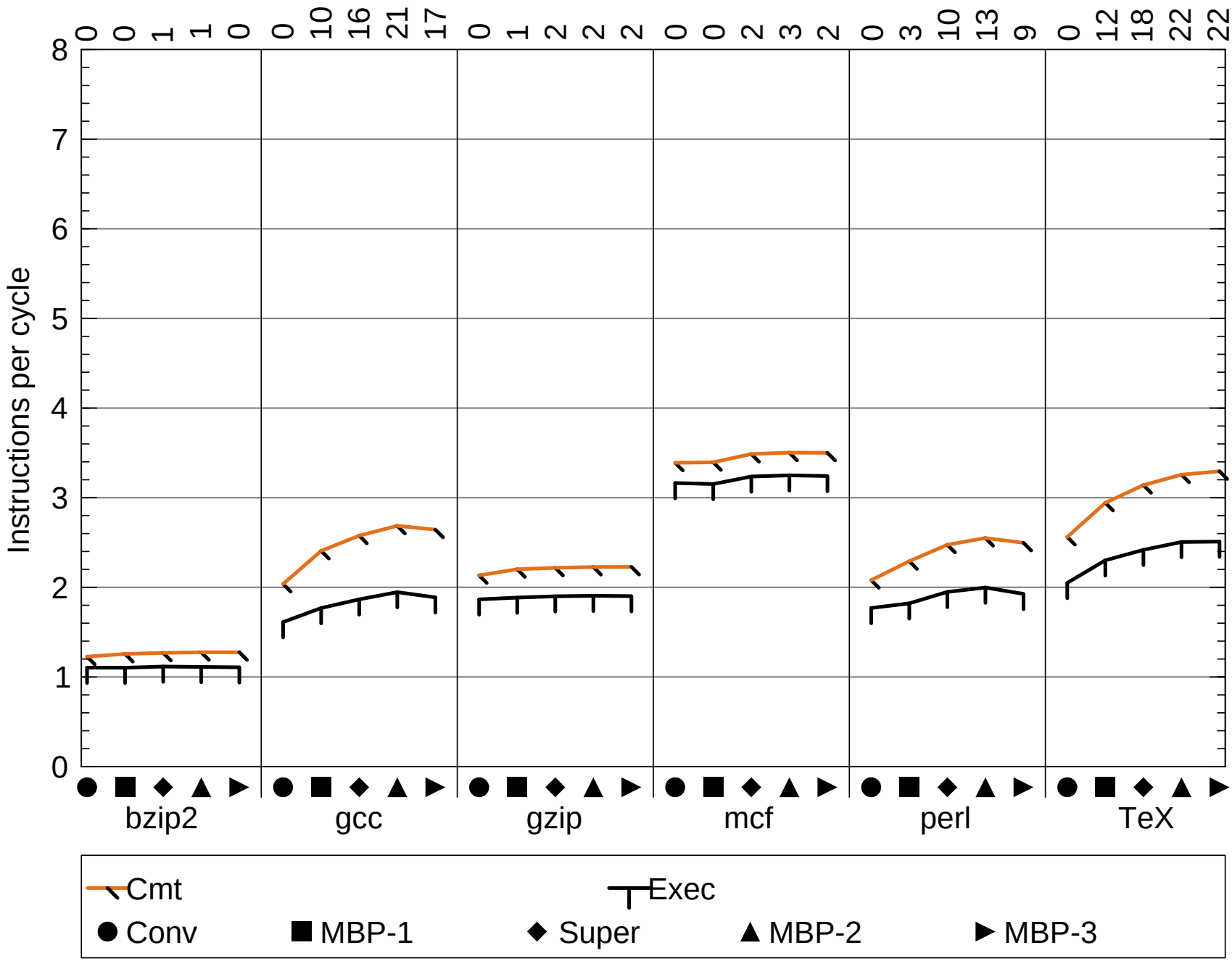
Commit rate usually higher than execution rate . . .

. . . difference determined by:

- Mispredict rate
- Branch resolution altitude. (Position in reorder buffer wrt x latency.)

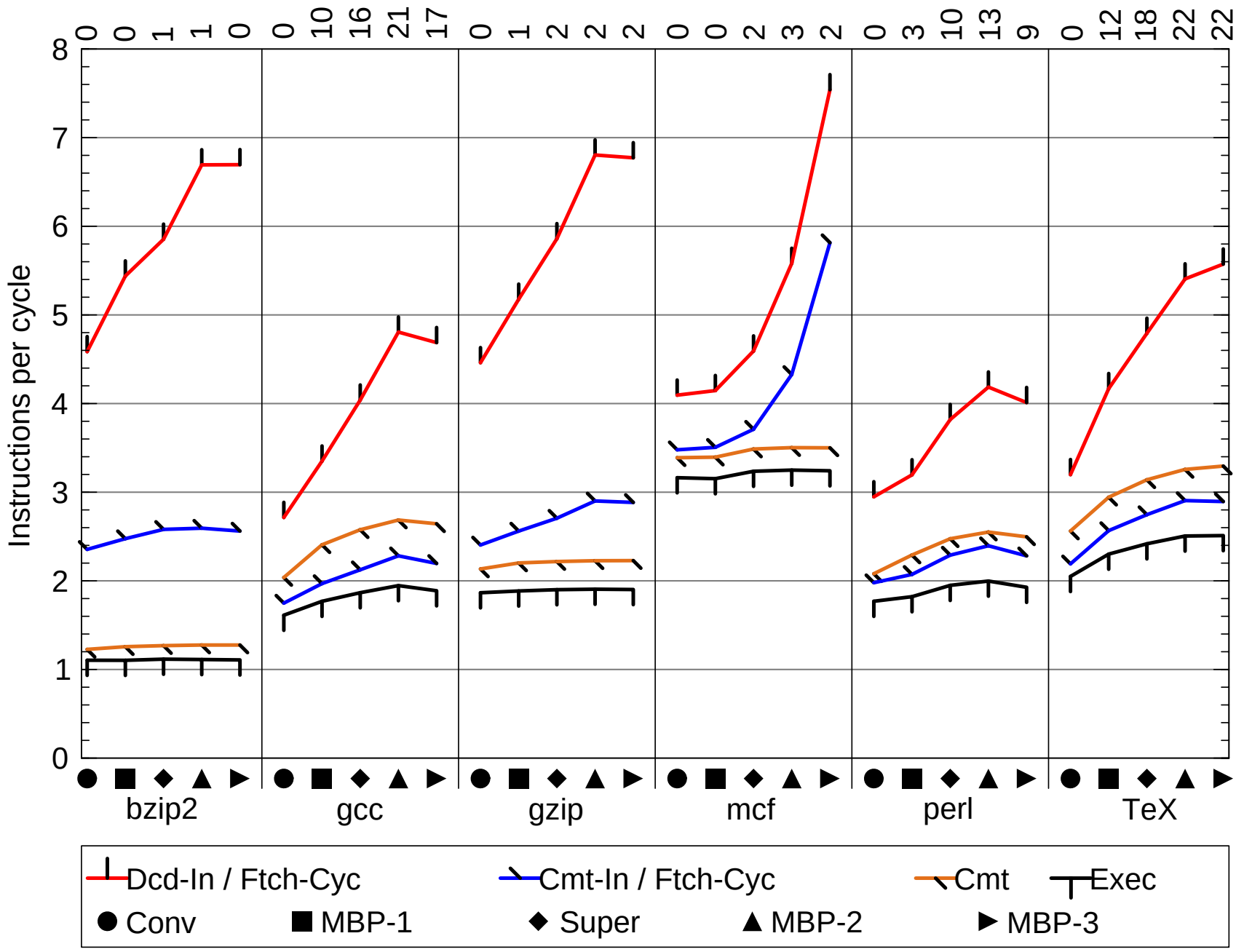
40

Commit Rate and Execution Rate



Commit Rate and Others

42



42

43

Commit Rate Factors

- Program characteristics.
- Execution resources and memory system performance.
- Window size sensitivity.

What hinders bzip2, gzip, and mcf?

Abbreviations in Commit Slot Graph

Segments above plot area show ROB occupancy on correct-path instruction arrival.

Segments in plot area show state of n (for an n-way system) instructions at ROB head.

Cmt:

Slots holding instructions that are committing. In per-cycle normalization shows execution rate, in per group normalization shows reciprocal of decode width.

Mem:

Slots holding instructions using a FU for more than one cycle, and instructions they block. Most such slots are load misses.

Depend:

Slots holding instructions that depend on an instruction that completed in the same cycle, and instructions they block.

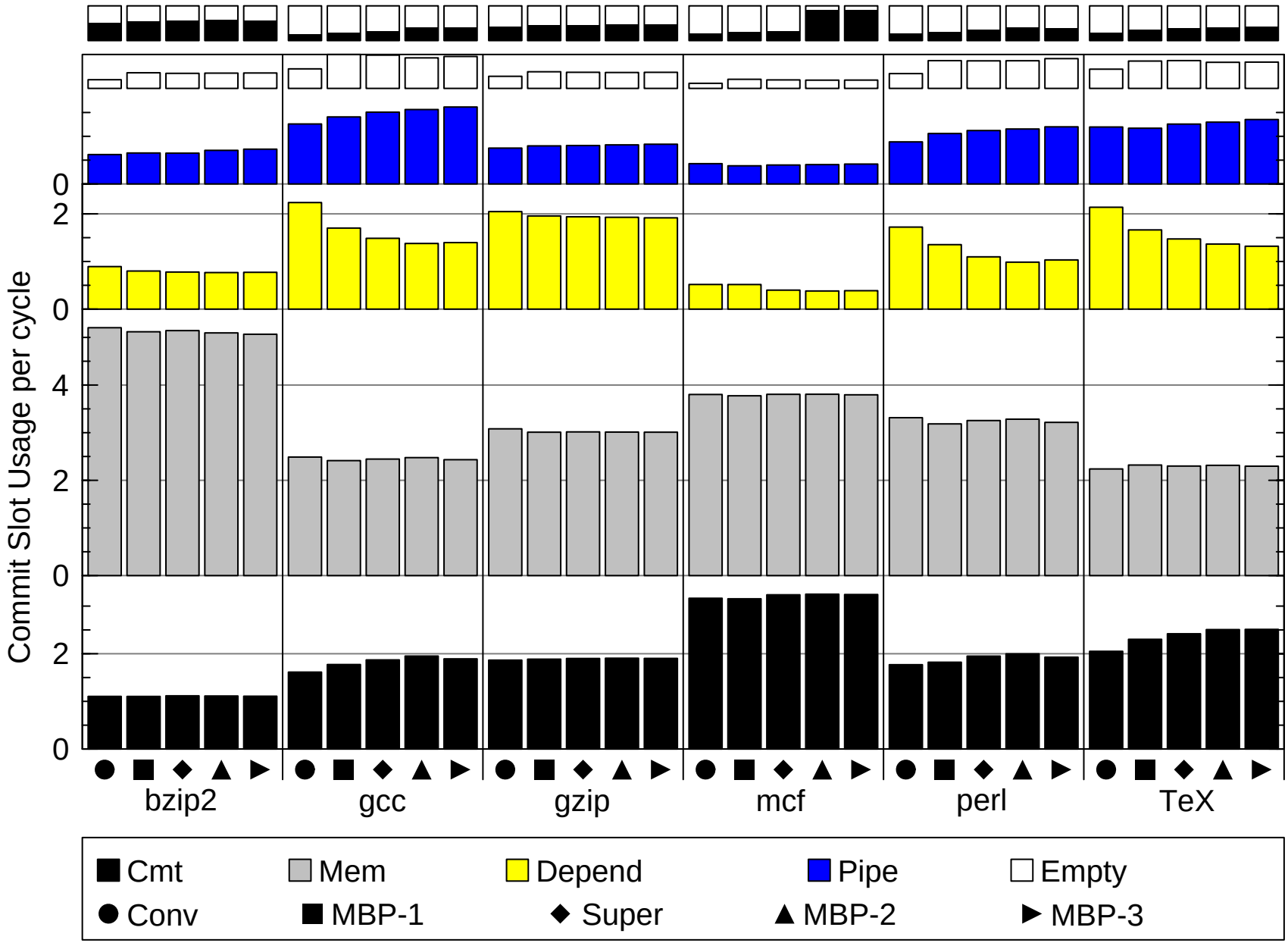
Pipe:

Slots holding instructions decoded too recently to execute.

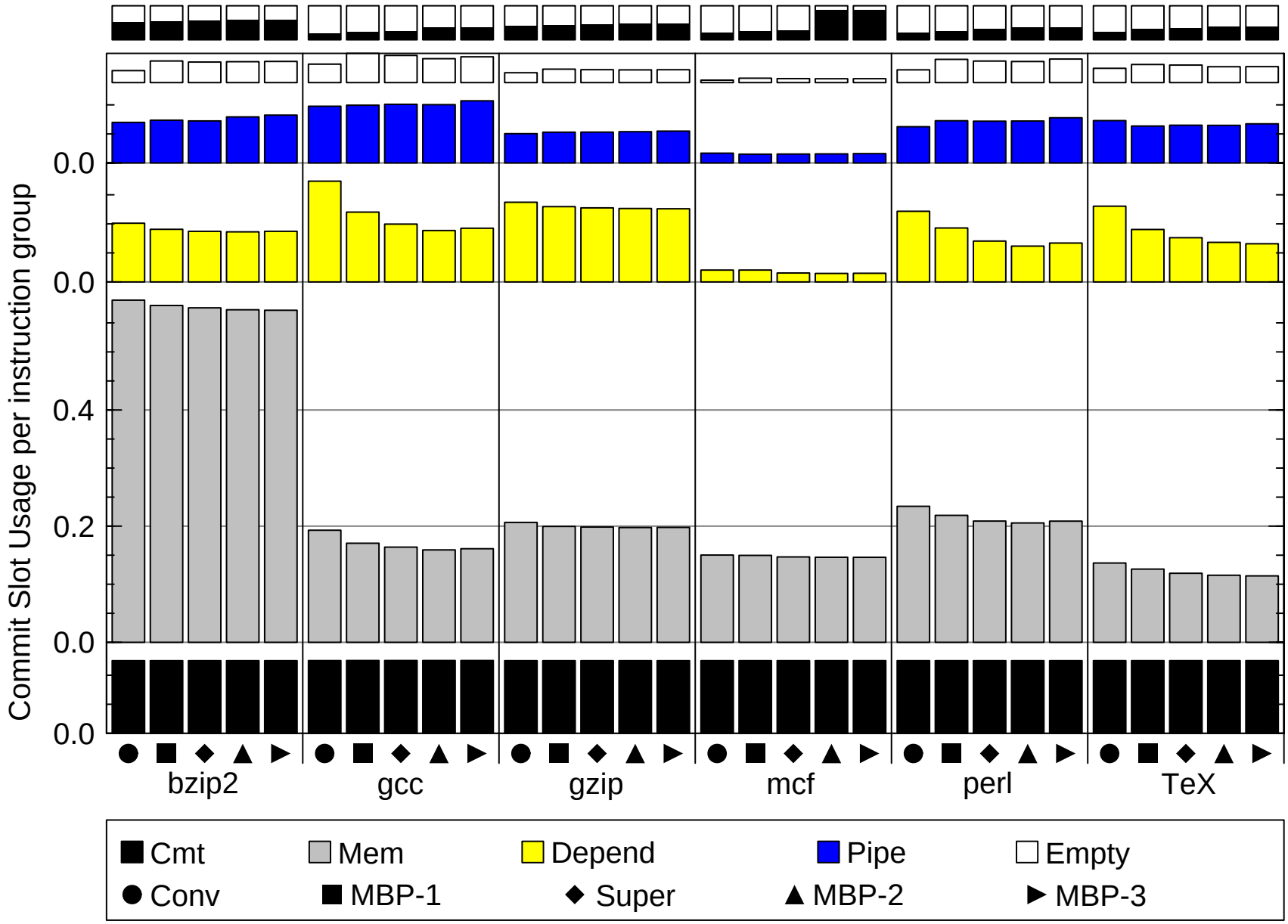
Empty:

Empty slots.

Commit Slot Usage (Per Cycle)



Commit Slot Usage (Per Committed Group)



47

Commit Rate Analysis Conclusions

Benchmarks gzip, perl, and TEX limited by branch prediction.

Greater ROB occupancy not sufficient for higher performance (bzip2).

Improvement due to increased sojourn times. (More time to finish.)

Some benchmarks insensitive due to program characteristics.

Load latency limits some benchmarks.

Configuration Effects

48

Branch Address Cache (BAC) Size

Tree Height on 16-Way Systems

Decode Width (Superscalaredness)

Level 2 Cache Size

BAC Size

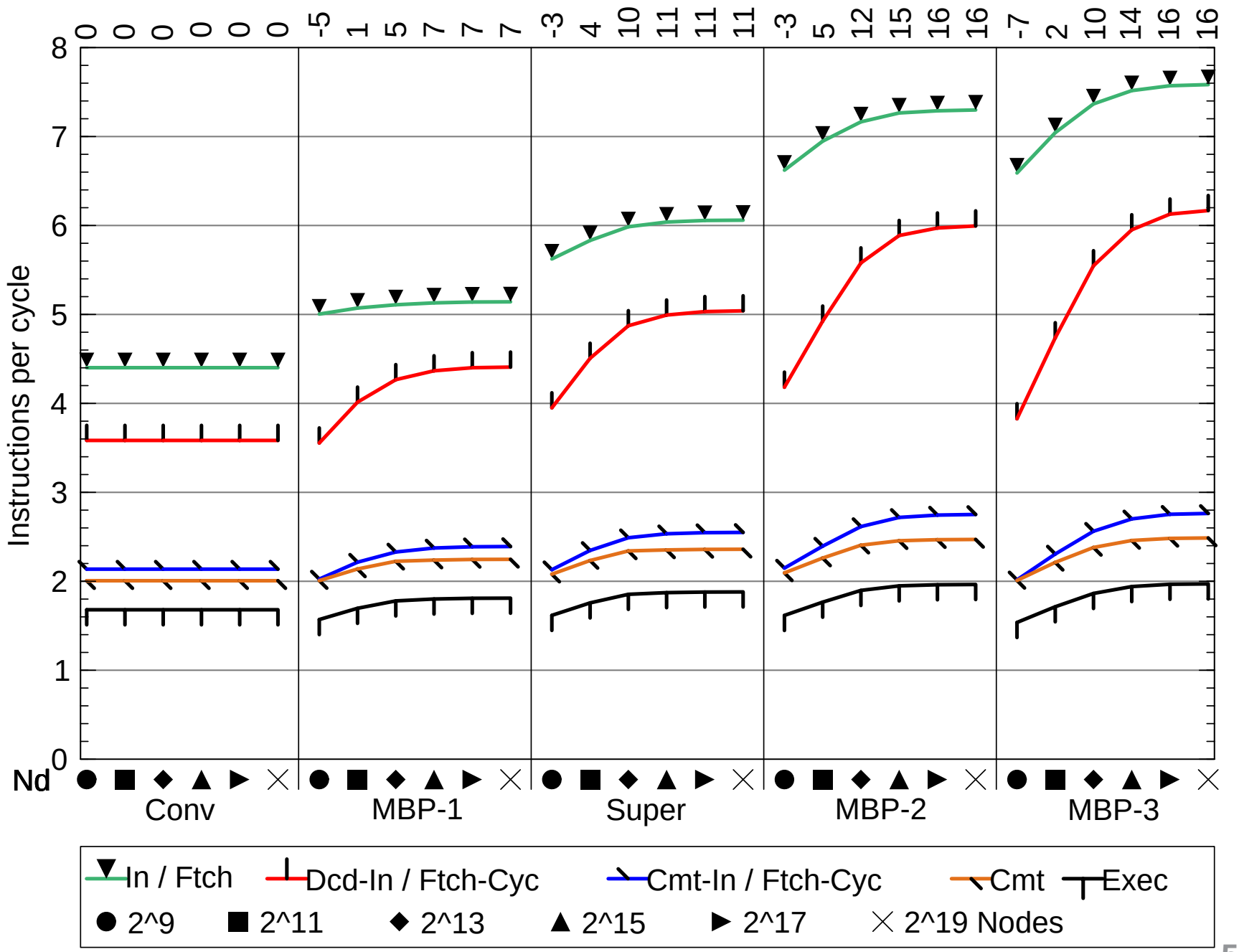
49

BAC size is substantial.

How small can it be made?

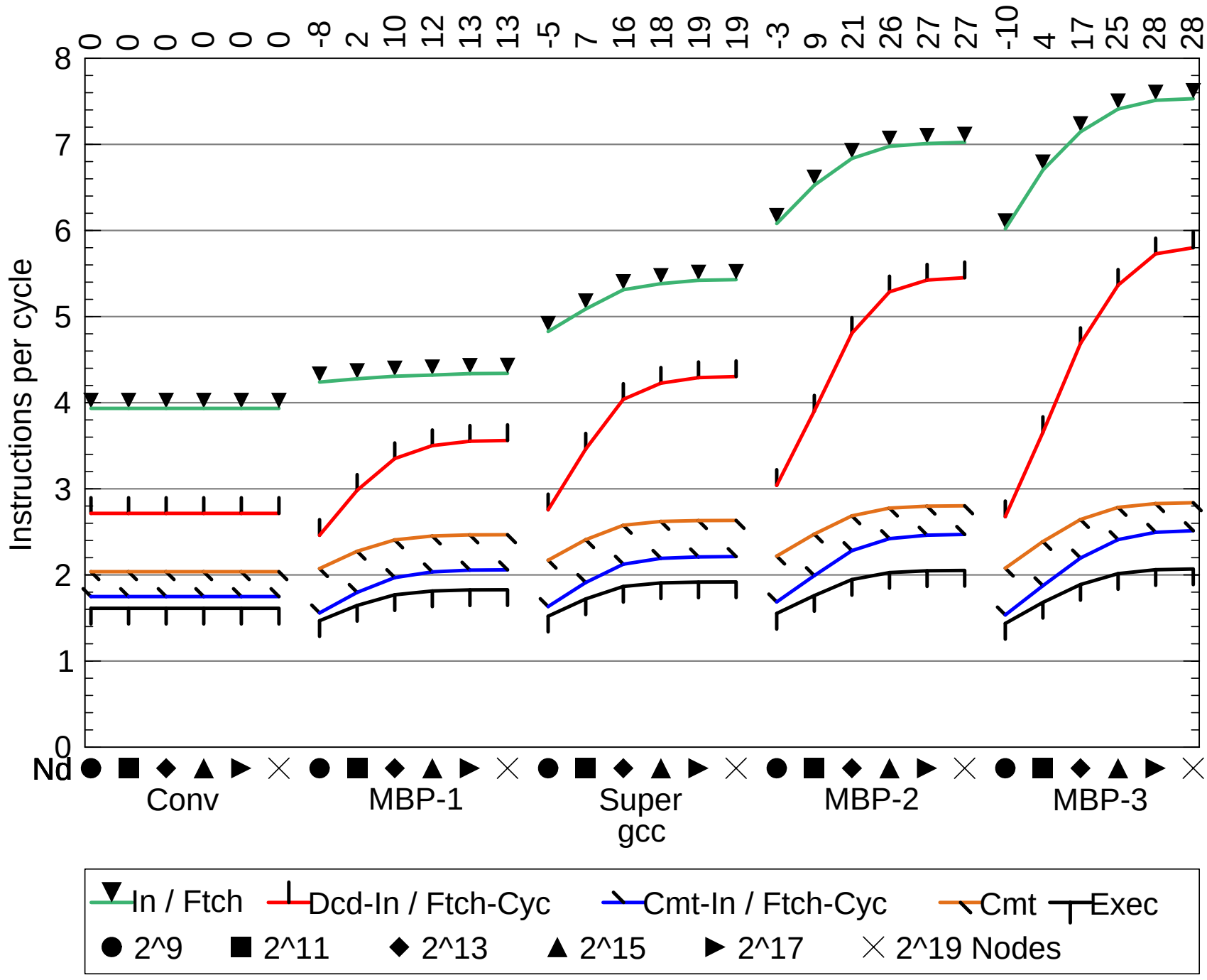
49

BAC Size (Average of Benchmarks)



BAC Size (Just gcc)

51



51
51

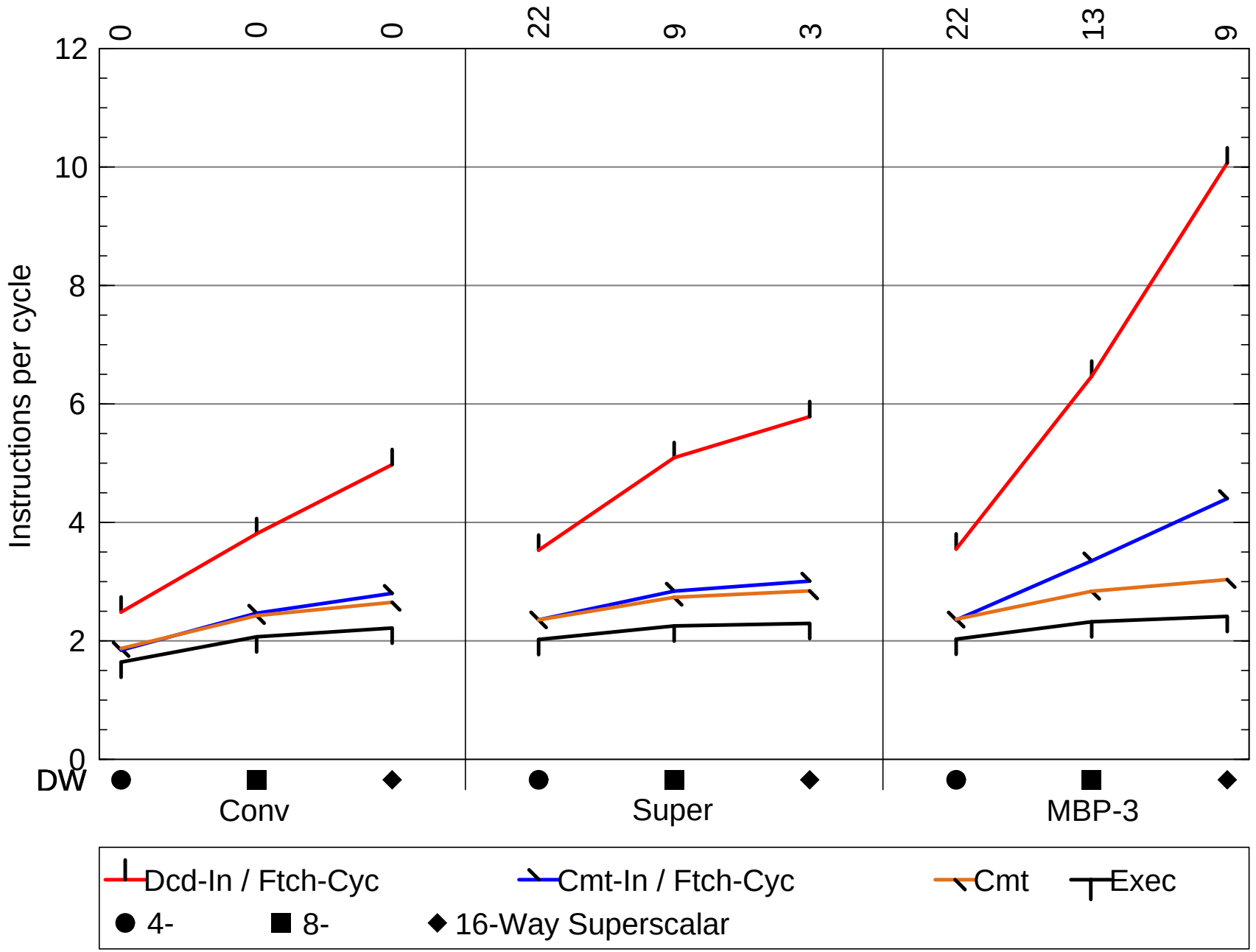
Decode Width

52

Simulated 4-, 8-, and 16-way machines.

Decode Width

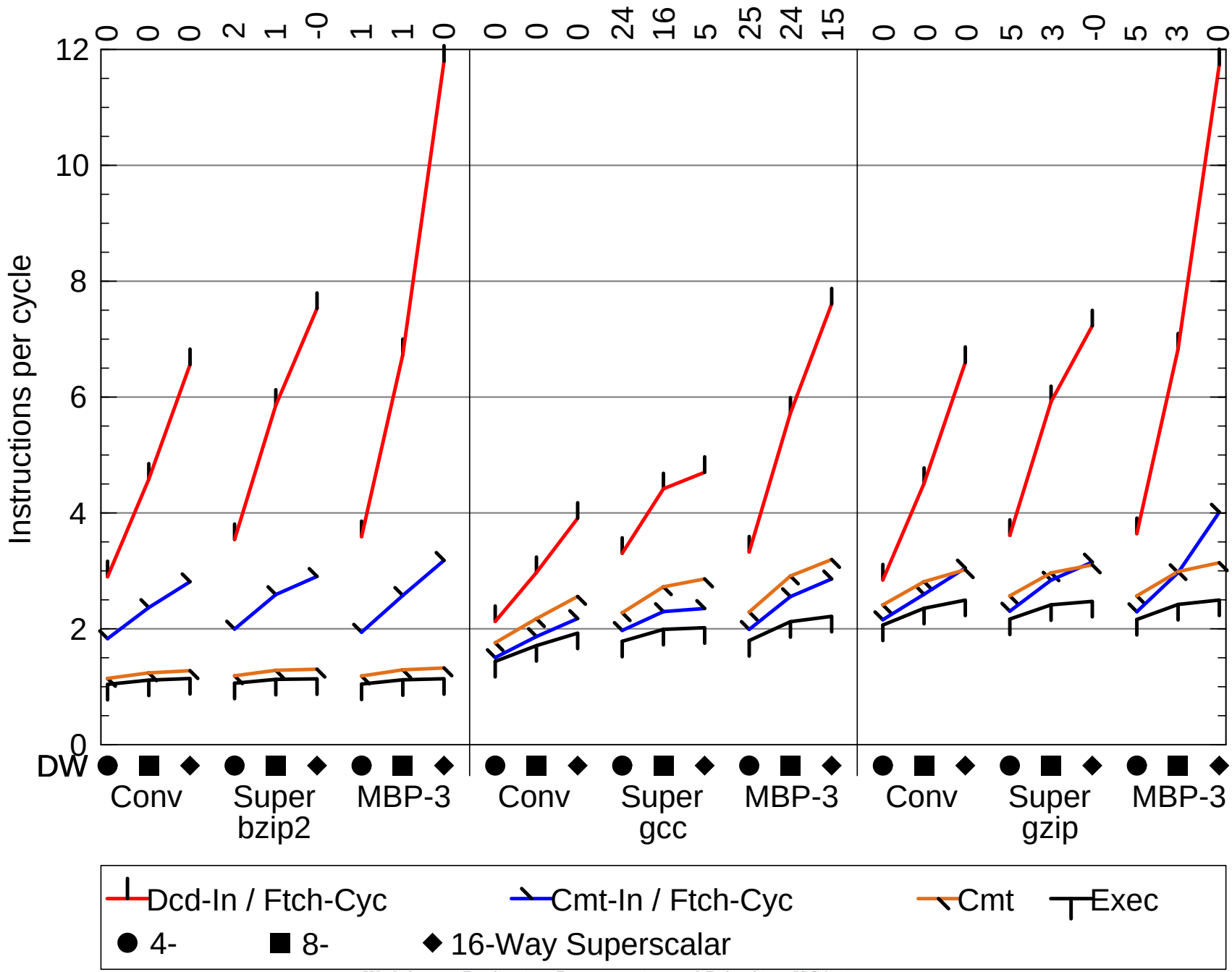
53



53

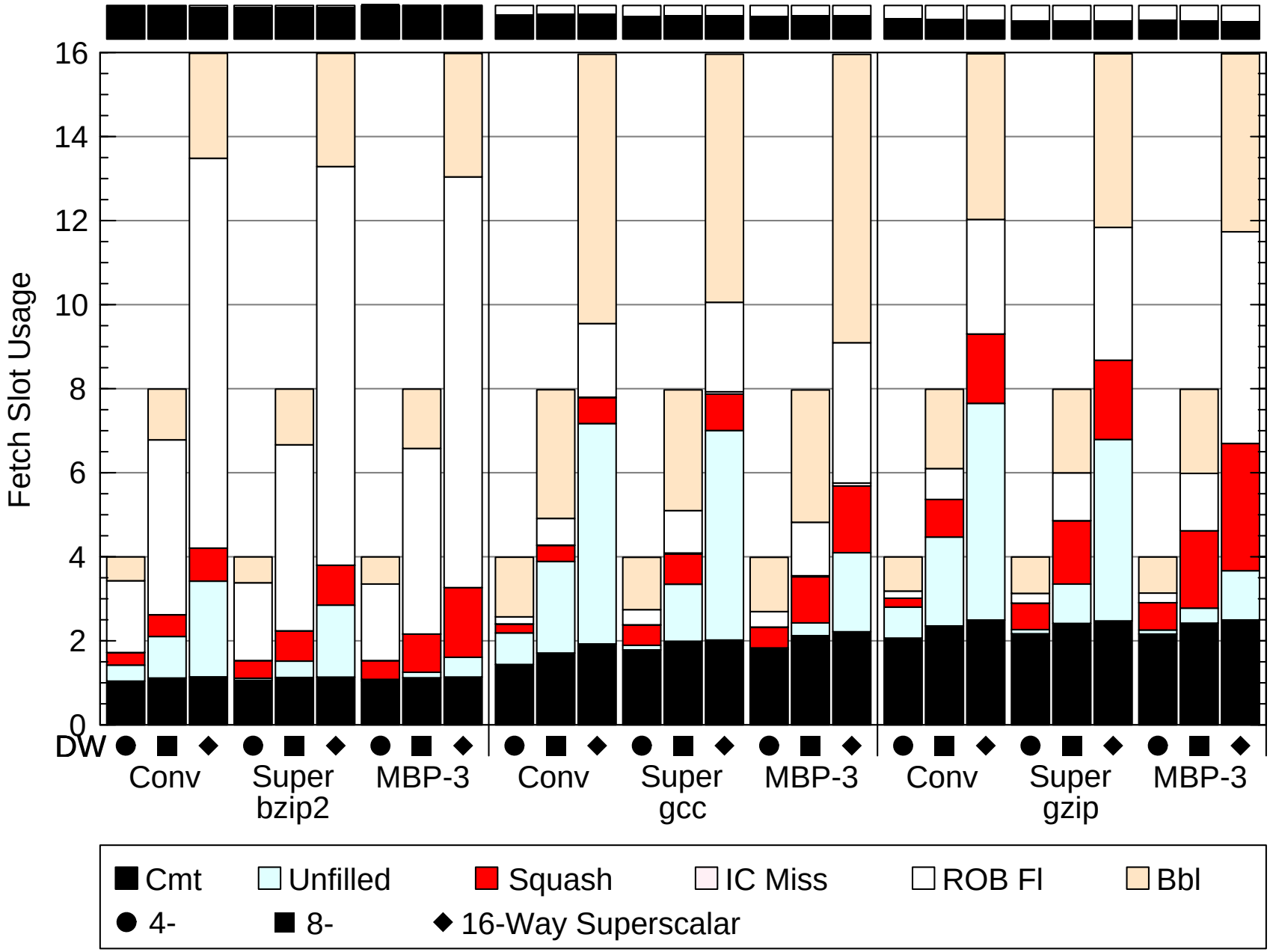
Decode Width

54



54

Decode Width



56

Decode Width Observations

MBP more effective for (relatively) narrow machines. . .
. . . because they need to make efficient use of fetch bandwidth.

Conventional wide machines already fetch almost enough instructions. . .
. . . so MBP helps less.

MBP would be more helpful on wider machines if other problems solved.

Sixteen-way superscalar barely faster than eight-way.

56

Impact of Level 2 Cache Size

57

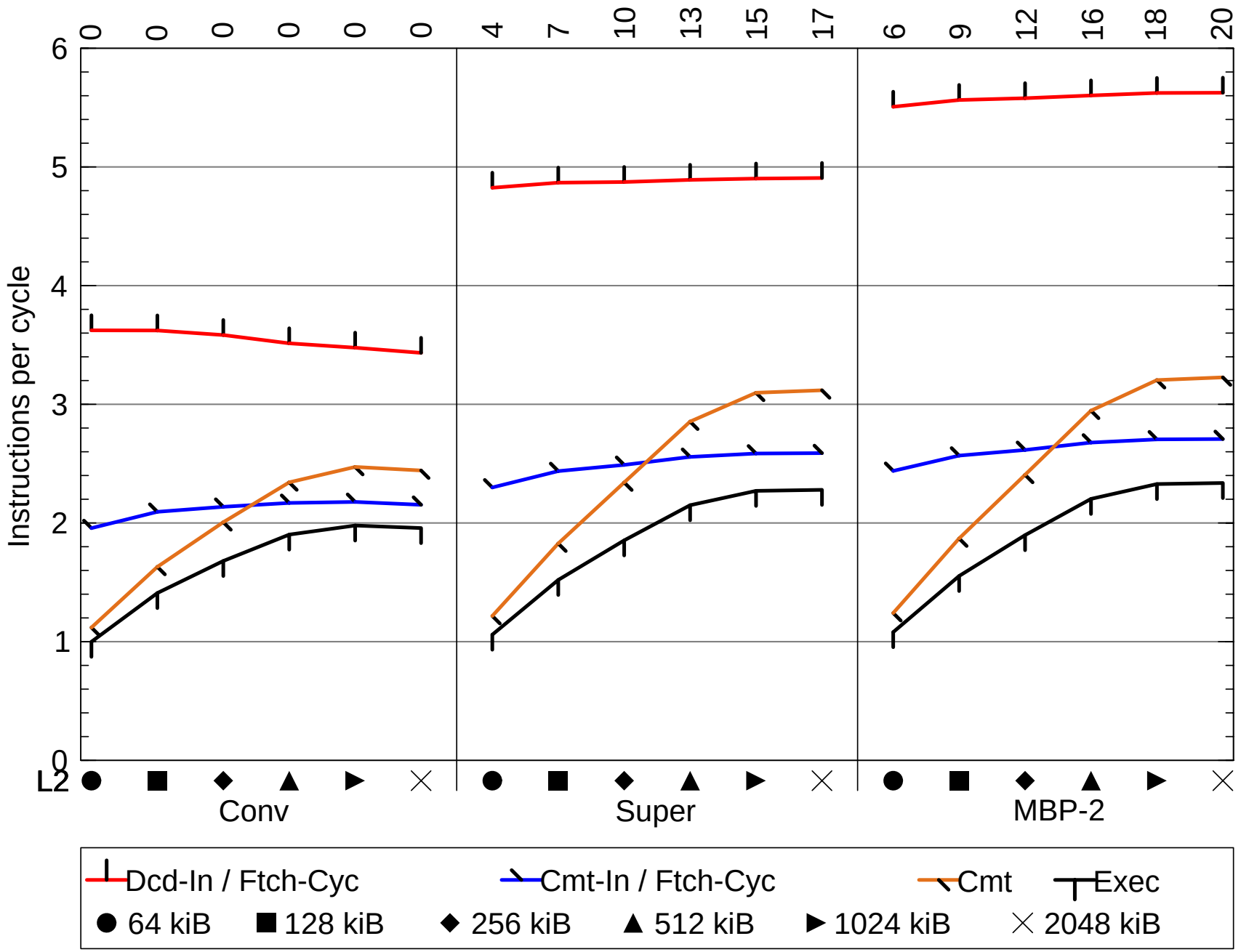
Does load latency make MBP pointless?

Vary level 2 size.

57

L2 Cache Size (Average)

58



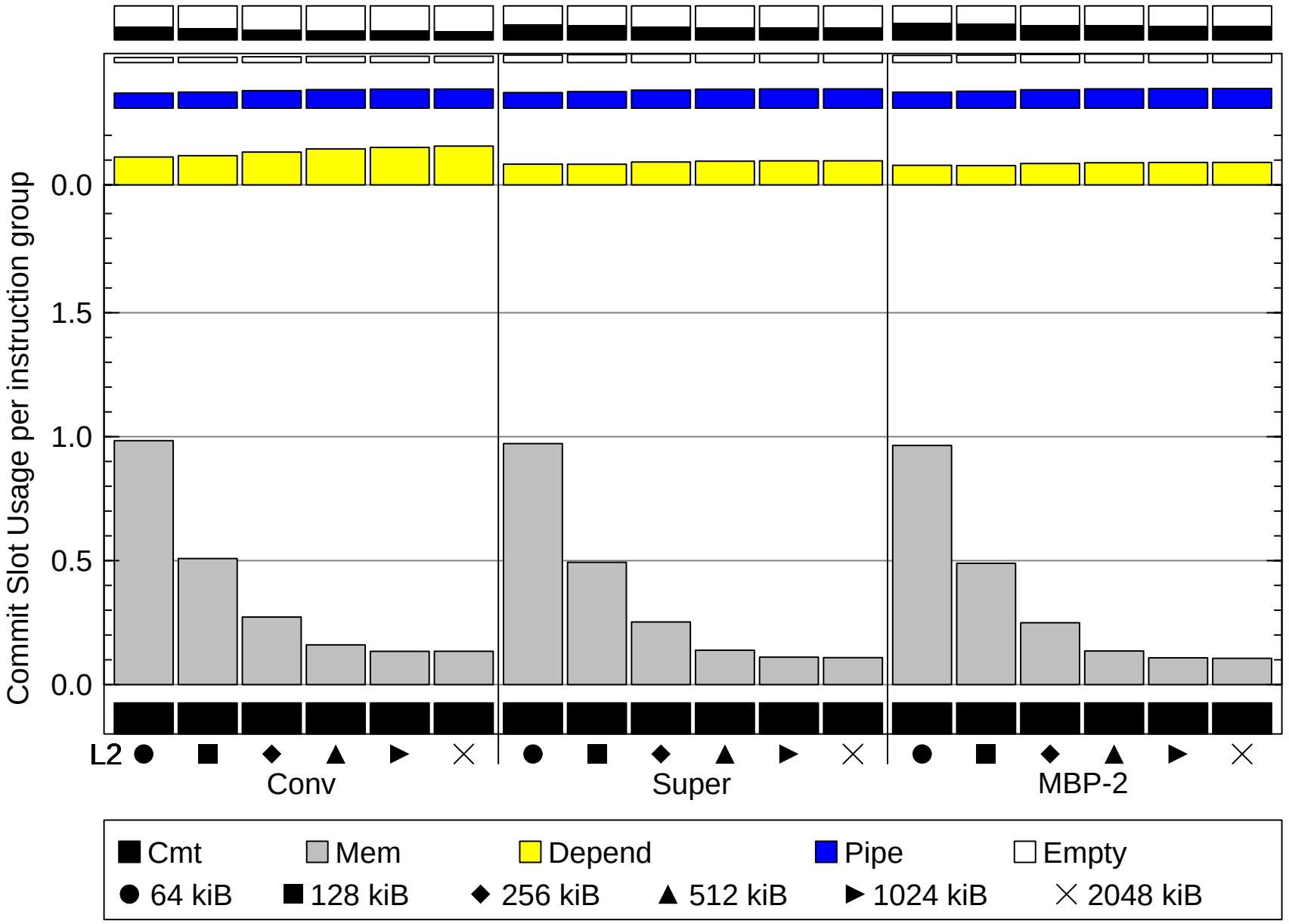
58

59

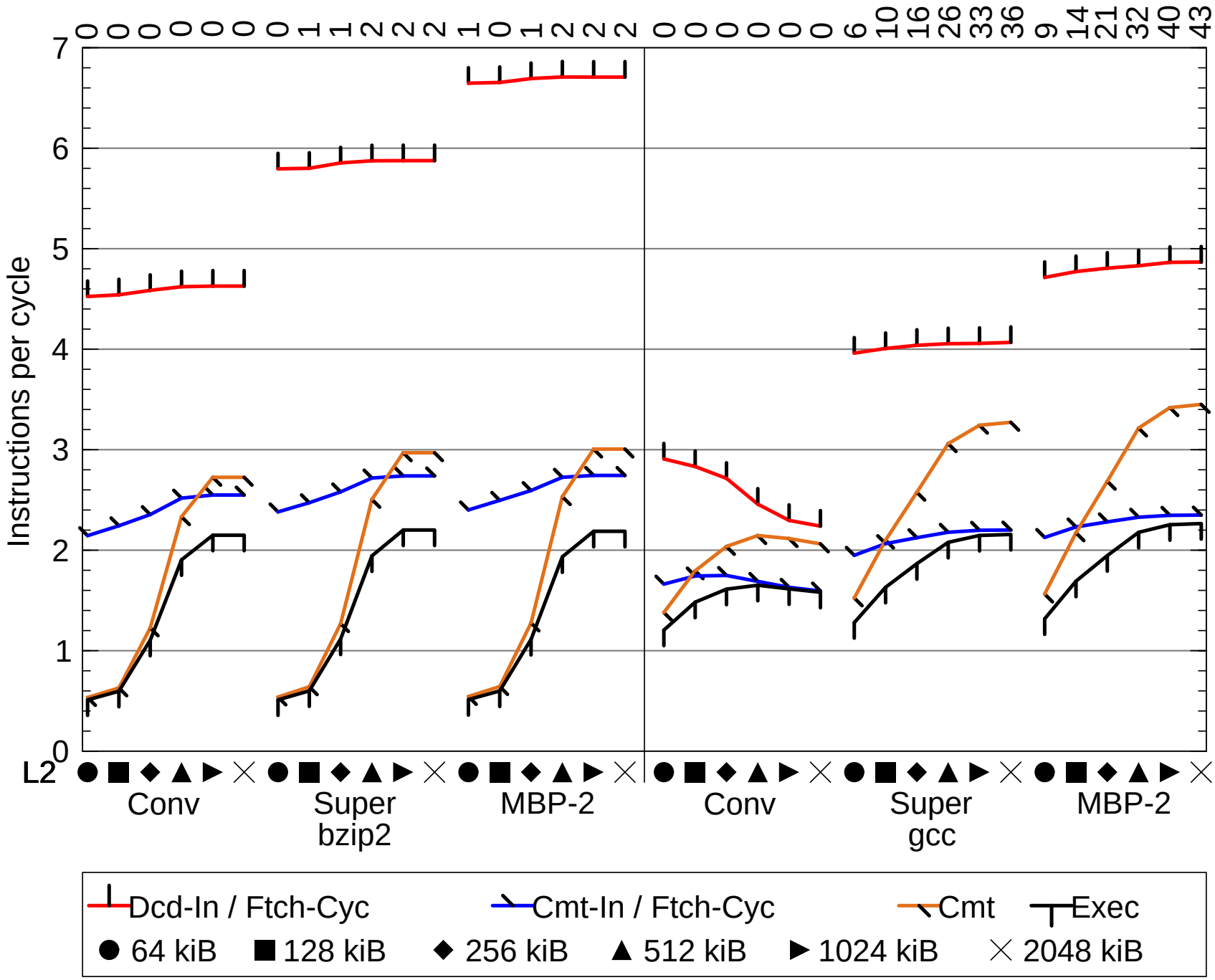
No, speedup still possible at smaller cache sizes.

59

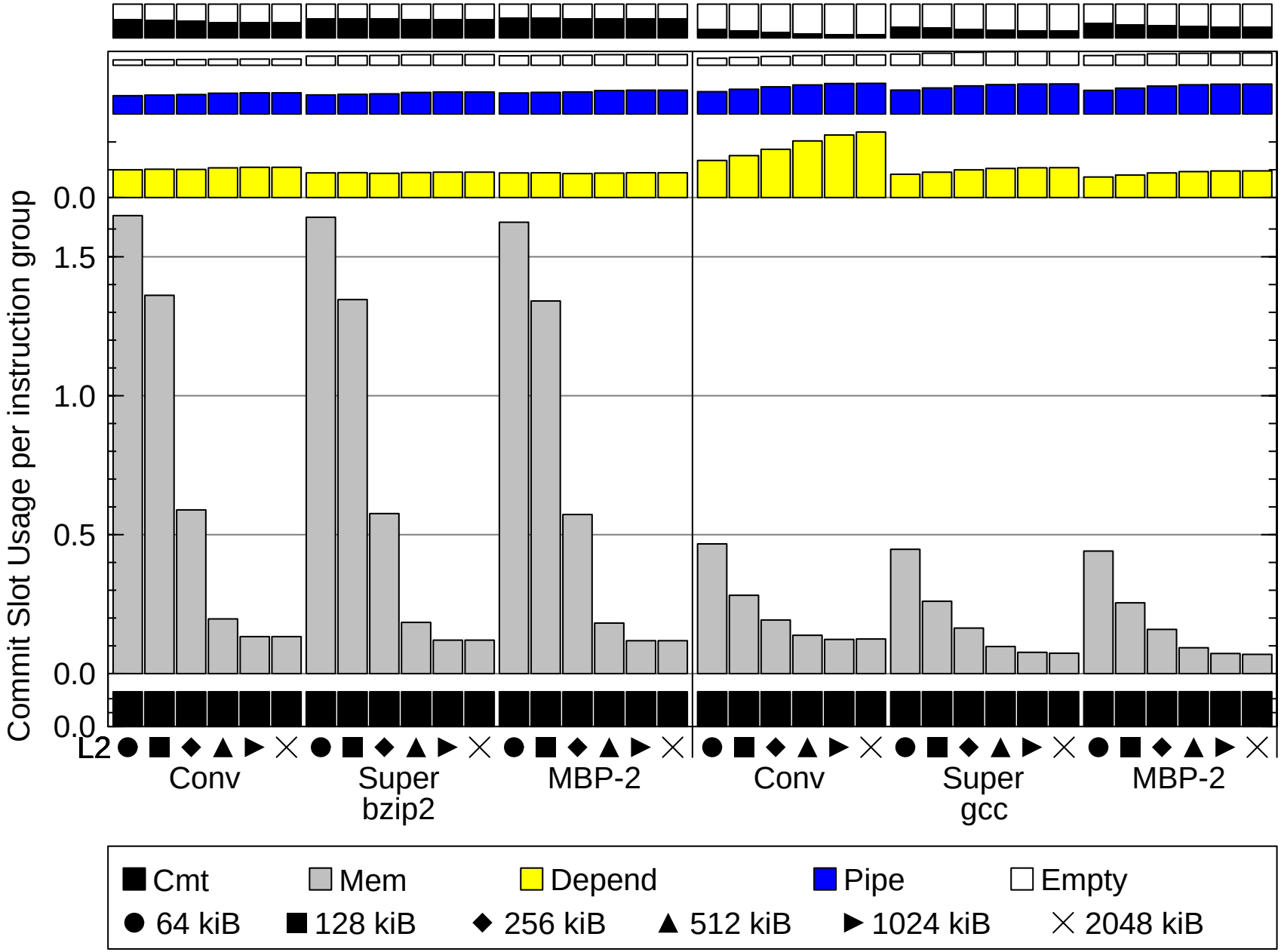
L2 Cache Size (Average)



L2 Cache Size (Selected Benchmarks)



L2 Cache Size (Selected Benchmarks)



MBP Order Tradeoffs

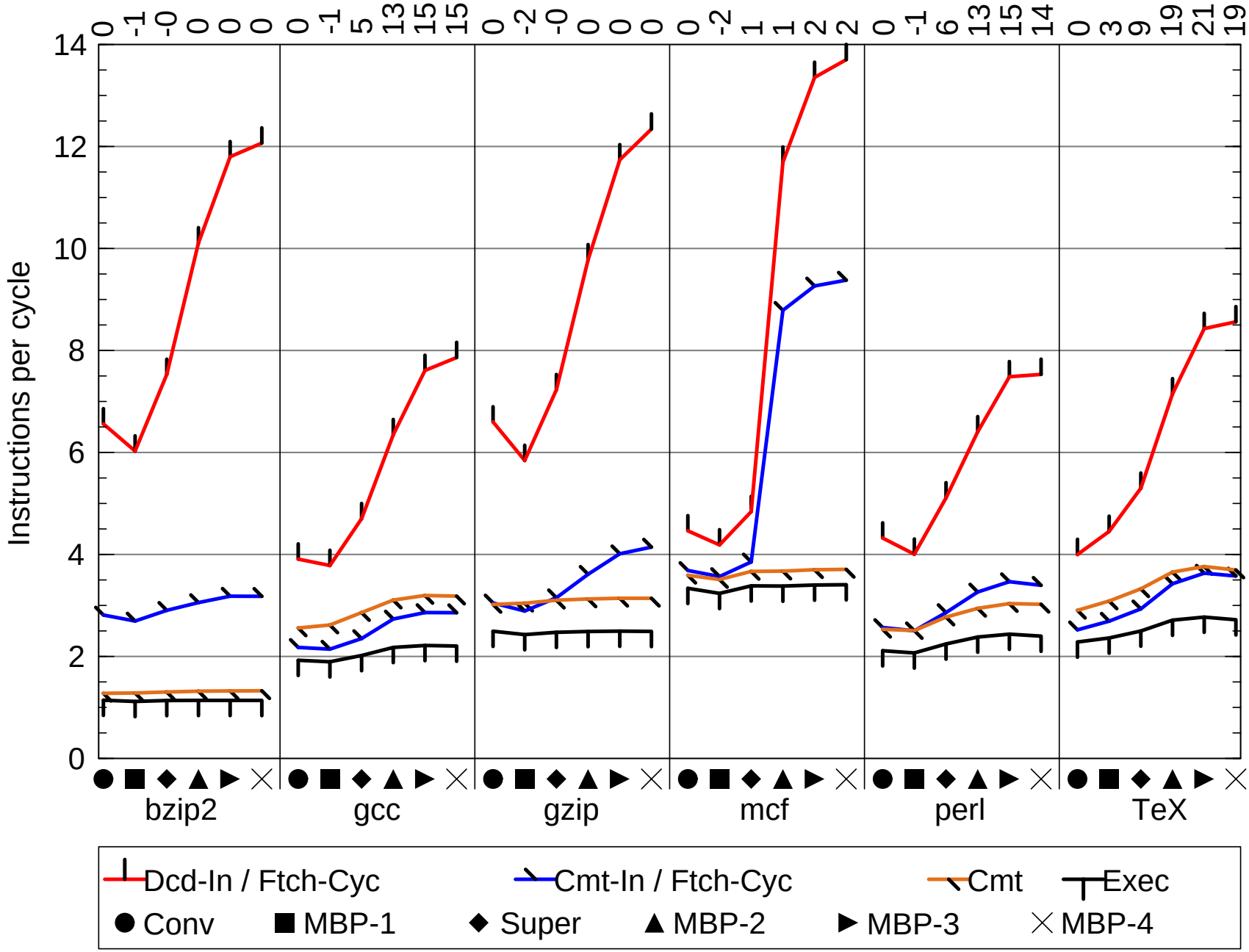
63

Different MBP Orders on 16-Way Systems

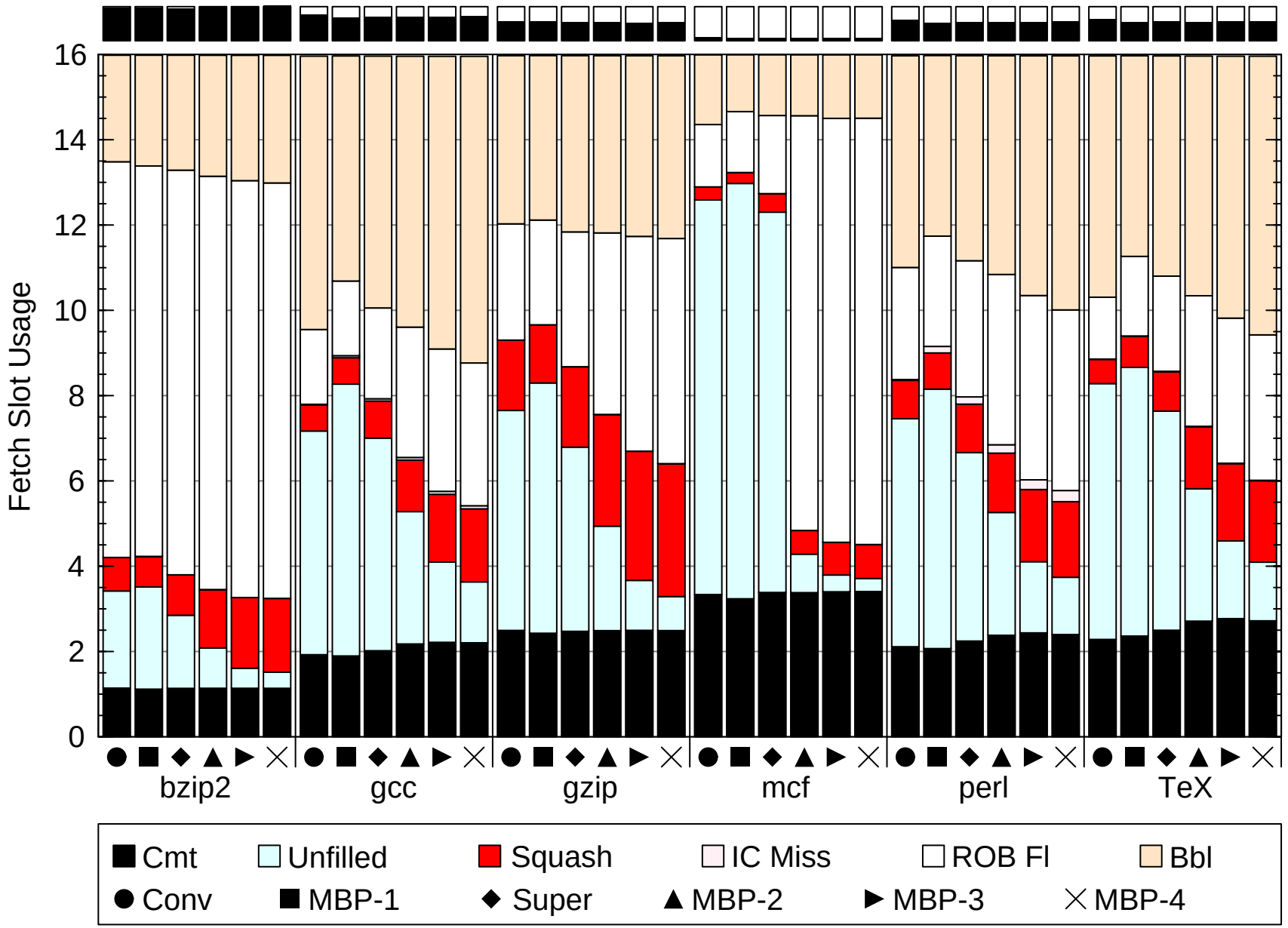
MBP-3 slightly better than MBP-4.

63

Sixteen-Way Larger Configuration

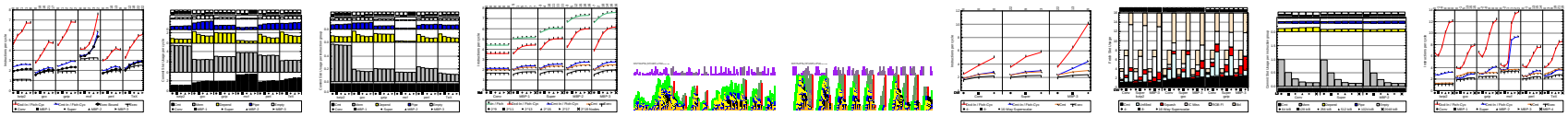


Sixteen-Way Larger Configuration



Conclusions

66



MBP systems are faster: order 2 sufficient.

Superblock between MBP-1 and MBP-2.

Speedup limited by branch prediction accuracy, load latency.

Interesting Questions

- Comparison with trace cache:
  - Does shorter mispredict pipeline pay for size of trace cache (v. BAC)?
  - Path storage (trace cache, trace ID) v. tree storage.