The Impact of Fetch Rate and Reorder Buffer Size on Speculative Pre-Execution

David M. Koppelman

Electrical & Computer Engineering Department
Louisiana State University
Baton Rouge, LA U.S.A.

Notes on These Slides

Slides for presentation made at the Workshop on Duplicating, Deconstructing, and Debunking, held in conjunction with the 30th International Symposium on Computer Architecture, 8 June 2003

Pages Added Since Presentation

Formulas for computing speedup components.

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

Dynamic Speculative Pre-Execution

Dynamic Speculative Pre-Execution:

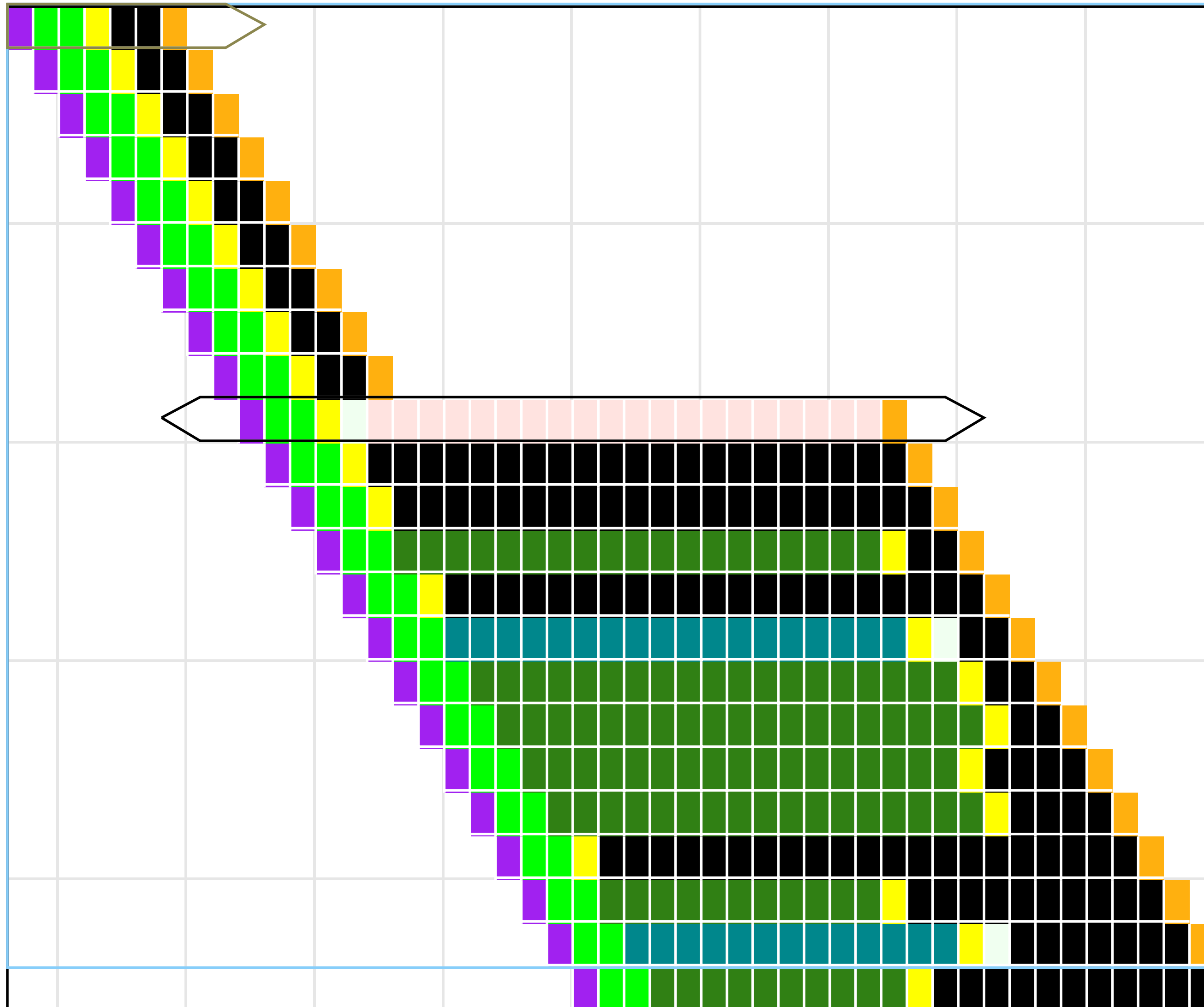
A technique to reduce the impact of load misses. . .

. . . by pre-executing copies of loads. . .

. . . along with instructions needed to compute their addresses.

Some schemes also pre-execute branches.

Load Miss (perl)

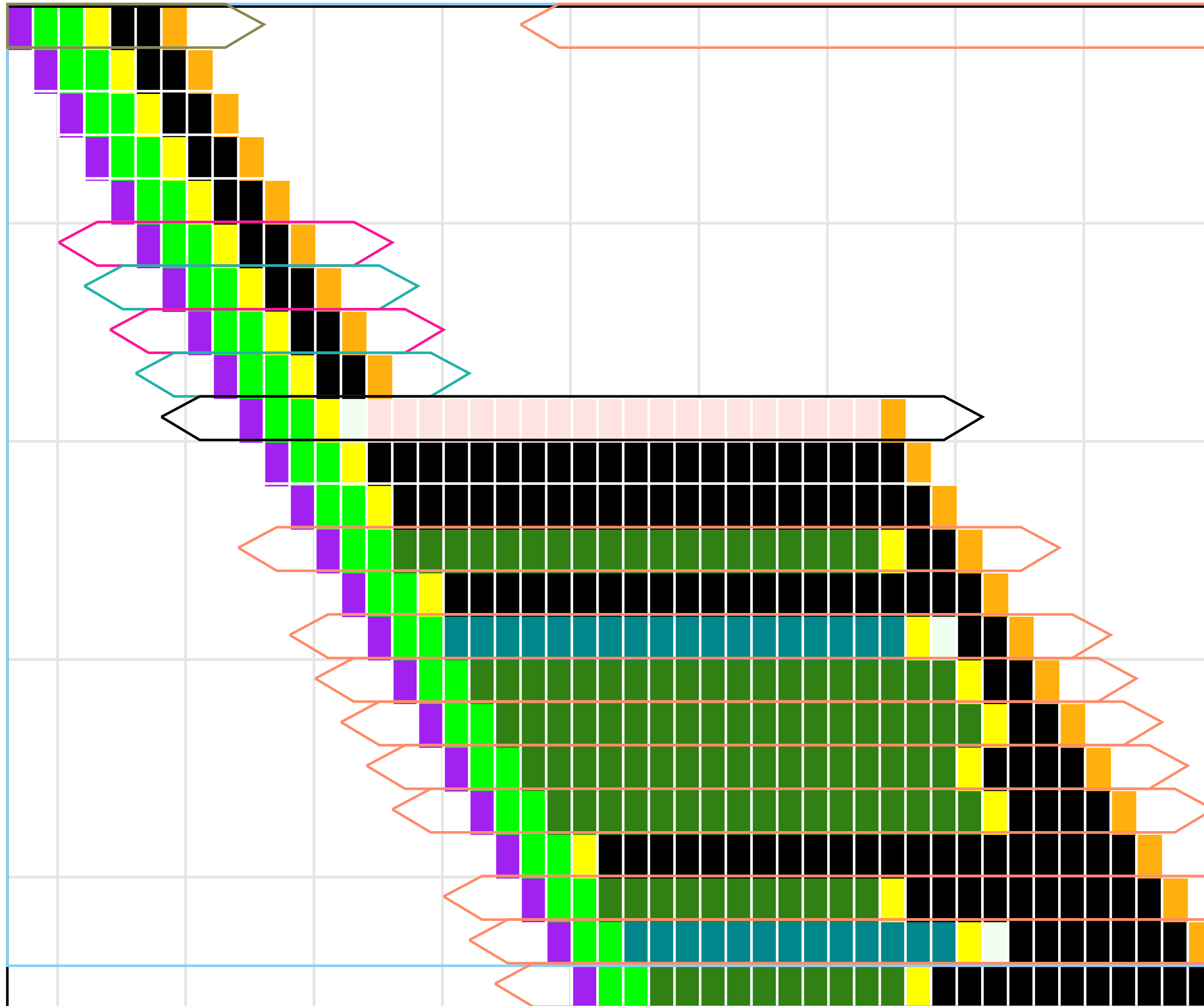


Perl_sv_grow+179
0x00103634 mov %i1, %o0
malloc
0x00075130 save %sp, -176, %sp
0x00075134 orcc %g0, %i0, %i3
0x00075138 bne +4i {malloc+6}
0x0007513c mov 4, %i2 {0x4}
malloc+6
0x00075148 cmp %i0, 80
0x0007514c bgu +8i {malloc+15}
0x00075150 sub %i0, 1, %g2
0x00075154 sethi %hi(0x204000), %g3
0x00075158 srl %g2, 2, %g2
0x0007515c add %g3, 120, %g3 {0x204078
0x00075160 ldsb [%g2 + %g3], %i2 {[b
0x00075164 ba +22i {malloc+35}
0x00075168 sethi %hi(0x216000), %g2
malloc+35
0x000751bc sll %i2, 2, %i1
0x000751c0 add %g2, 120, %i0 {0x216078}
0x000751c4 lduw [%i1 + %i0], %o3 {[nextf-
0x000751c8 cmp %o3, 0
0x000751cc bne +6i {malloc+45}
0x000751d0 cmp %o3, 0
malloc+45
0x000751e4 bne +73i {malloc+118}
0x000751e8 sethi %hi(0x218400), %g2

Time 6,968,169

Grid 5 insn X 5 cyc

Load Miss and Slice (perl)



Perl_sv_grow+179
0x00103634 mov %i1, %o0
malloc
0x00075130 save %sp, -176, %sp
0x00075134 **orcc %g0, %i0, %i3**
0x00075138 bne +4i {malloc+6}
0x0007513c mov 4, %i2 {0x4}
malloc+6
0x00075148 cmp %i0, 80
0x0007514c bgu +8i {malloc+15}
0x00075150 sub %i0, 1, %g2
0x00075154 sethi %hi(0x204000), %g3
0x00075158 srl %g2, 2, **%g2**
0x0007515c add %g3, 120, **%g3** {0x2040
0x00075160 **ldsb [%g2 + %g3], %i2 {[b**
0x00075164 ba +22i {malloc+35}
0x00075168 sethi %hi(0x216000), %g2
malloc+35
0x000751bc sll **%i2**, 2, %i1
0x000751c0 add %g2, 120, %i0 {0x216078}
0x000751c4 lduw [%i1 + %i0], %o3 {[ne
0x000751c8 cmp %o3, 0
0x000751cc bne +6i {malloc+45}
0x000751d0 cmp %o3, 0
malloc+45
0x000751e4 bne +73i {malloc+118}
0x000751e8 sethi %hi(0x218400), %g2

Time 6,968,169

Grid 5 insn X 5 cyc

Pre-Execution Idea

Preparation

Find *troublesome* (frequently missing) load.

Extract *slice*, load and predecessors . . .
. . . by examining instructions within a *construction window*.

Identify *trigger* instruction to initiate slice.

Put slice in cache, using trigger address for index.

Execution

Hardware usually added onto a simultaneous multithreading core.

When trigger encountered a *p-thread* is forked to run slice.

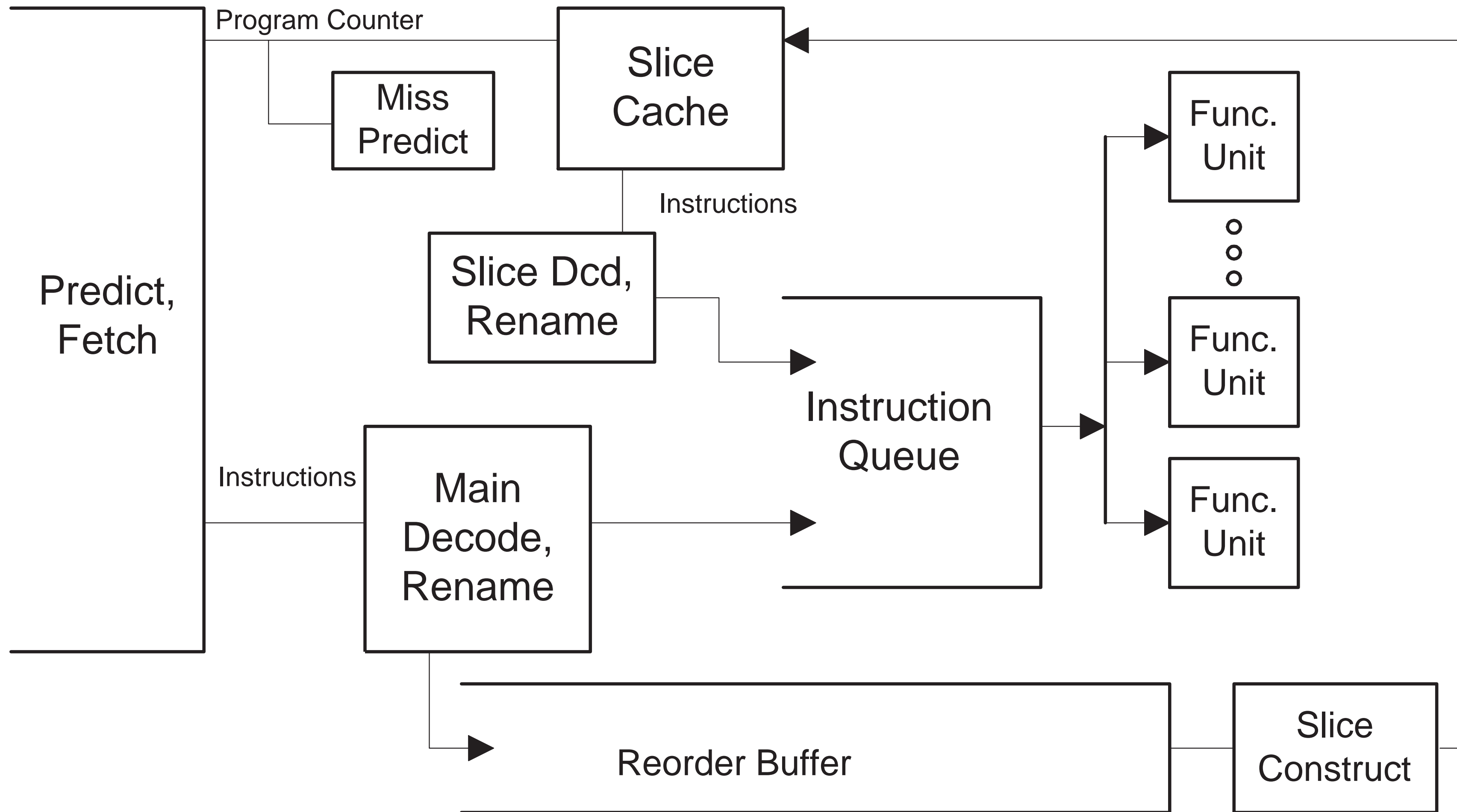
Slice's load, *prefetch*, should execute in advance of original, *target*, load.

Margin is number of cycles from initiation of prefetch to target.

When slice fi anished results discarded.

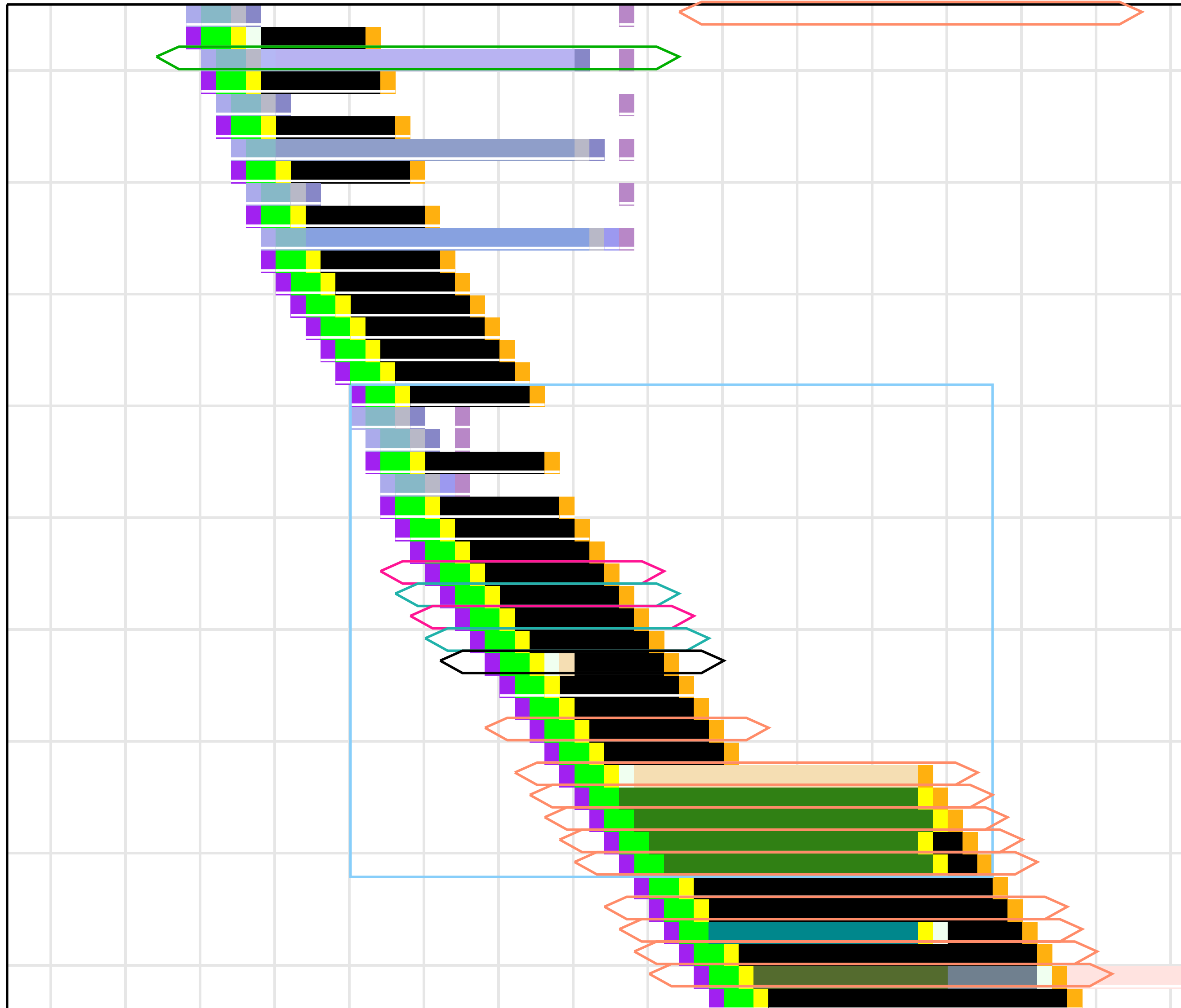
Typical Pre-Execution Hardware

7



7

Prefetch Load Miss (perl)

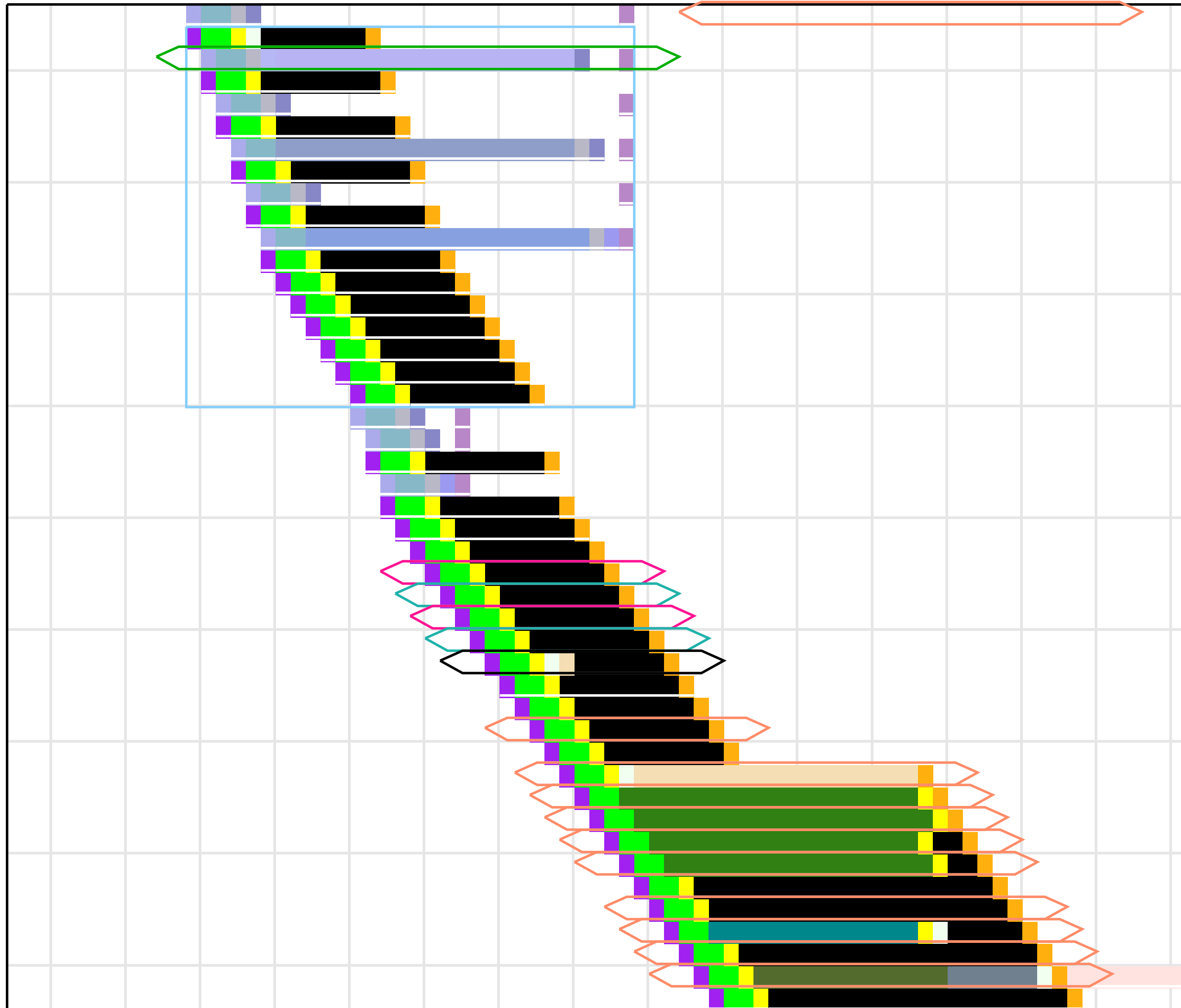


```
malloc+1
0x00075134   orcc  %g0, %i0, %i3
morecore
0x000756f8   save  %sp, -104, %sp
morecore+6
0x00075710   sethi  %hi(0x215c00), %i0
malloc+2
0x00075138   bne  +4i  {malloc+6}
morecore+11
0x00075724   lduw  [ %i0 + 876 ], %o3
malloc+3
0x0007513c   mov  4, %i2  {0x4}
malloc+6
0x00075148   cmp  %i0, 80
0x0007514c   bgu  +8i  {malloc+15}
0x00075150   sub  %i0, 1,  %g2
0x00075154   sethi  %hi(0x204000),   %g3
0x00075158   srl  %g2, 2,  %g2
0x0007515c   add  %g3, 120,  %g3  {0x2040
0x00075160   ldsb  [ %g2 + %g3 ], %i2  {[b
0x00075164   ba  +22i  {malloc+35}
0x00075168   sethi  %hi(0x216000), %g2
malloc+35
0x000751bc   sll  %i2 , 2,  %i1
0x000751c0   add  %g2, 120, %i0  {0x216078}
0x000751c4   lduw  [  %i1 + %i0 ],  %o3  {[ne
0x000751c8   cmp  %o3 , 0
0x000751cc   bne  +6i  {malloc+45}
```

Time 6,968,503

Grid 5 insn X 5 cyc

Prefetch Load Miss (perl)



Perl_yylex+121
0x00081264 lduh [%g2 + 160], %g2 {[PL_la
malloc+12
0x00075160 ldsb [%g2 + %g3], %i2
Perl_yylex+122
0x00081268 mov %o0, %i4
malloc+14
0x00075168 sethi %hi(0x216000), %g2
Perl_yylex+123
0x0008126c cmp %g2, 150
malloc+35
0x000751bc sll %i2, 2, %i1
Perl_yylex+124
0x00081270 bne +65i {Perl_yylex+189}
malloc+36
0x000751c0 add %g2, 120, %i0
Perl_yylex+125
0x00081274 mov 12, %o0 {0xc}
malloc+37
0x000751c4 lduw [%i1 + %i0], %o3
Perl_yylex+189
0x00081374 call +39867i {Perl_newOP}
0x00081378 mov 0, %o1 {0x0}
Perl_newOP
0x000a8260 save %sp, -96, %sp
0x000a8264 call -52301i {malloc}
0x000a8268 mov 24, %o0 {0x18}
malloc

Time 6,968,503

Grid 5 insn X 5 cyc

Pre-Execution Potential

10

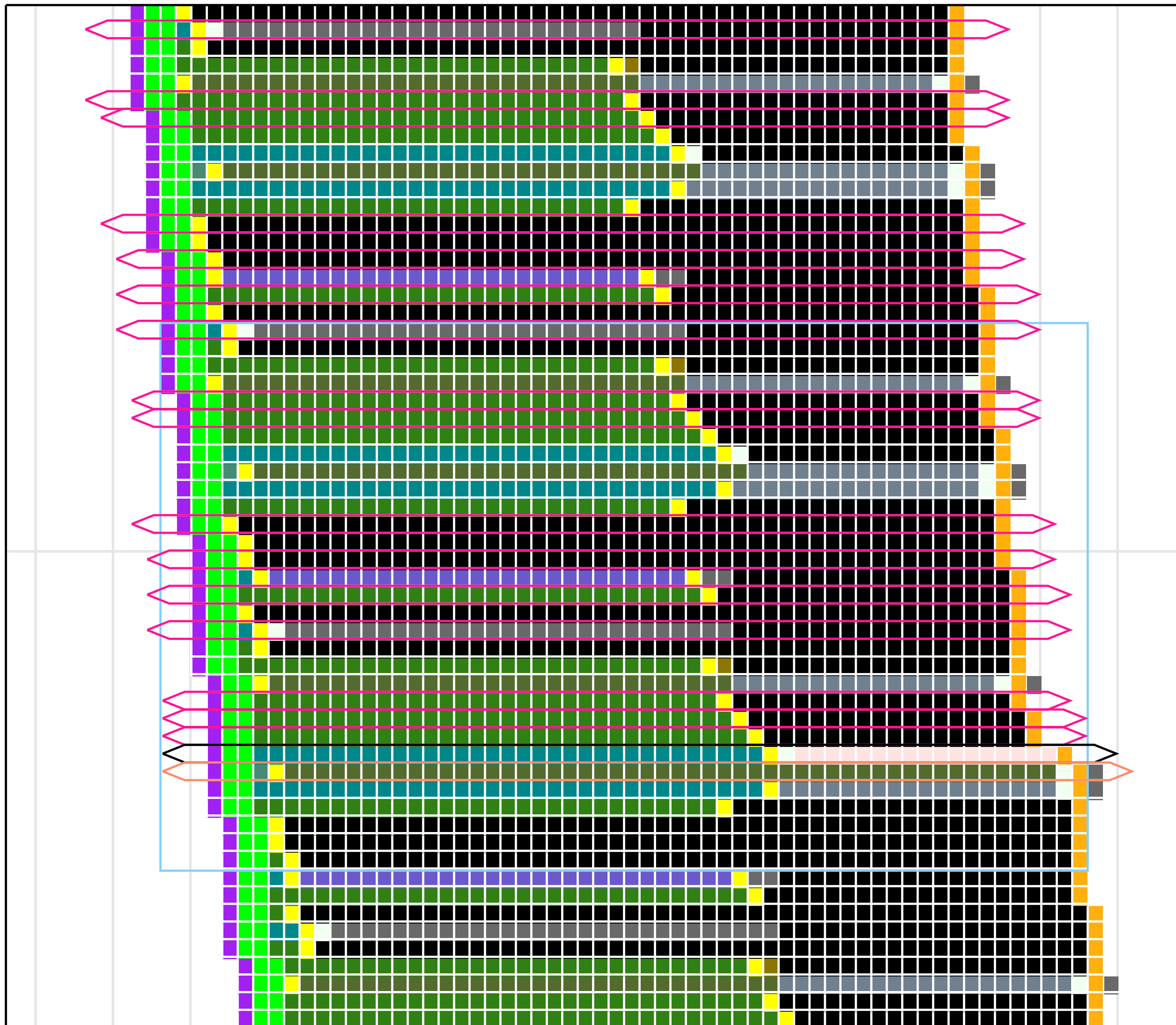
No shortage of load misses to help . . .

. . . but will prefetch really execute before target?

10

Small-Window, Low Margin (gzip)

11



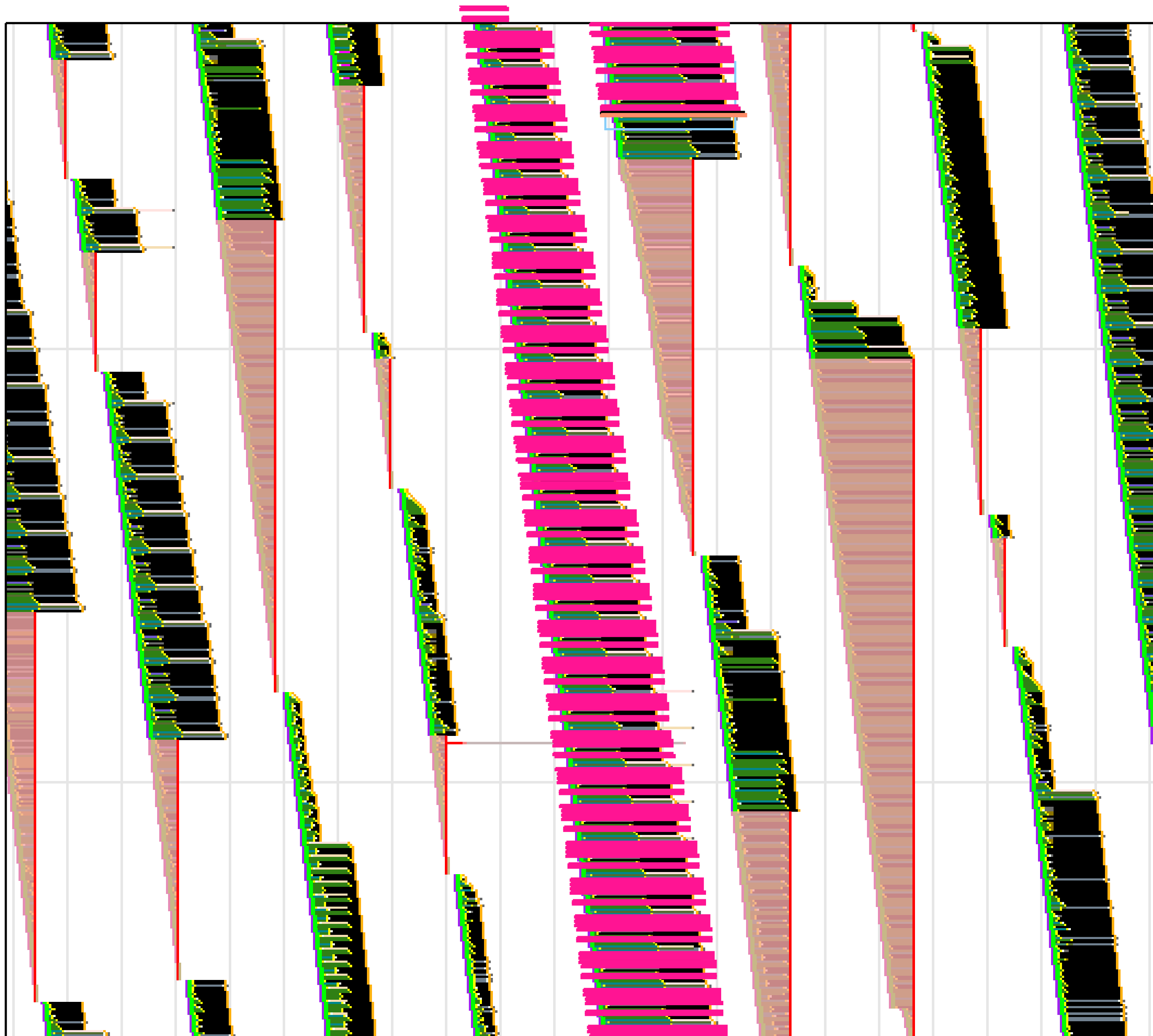
```
deflate+124
0x00012648   ldub  [ %g3 - 1 ],   %g3
0x0001264c   sll  %g4, 1, %g4
0x00012650   subcc  %o0, 1, %o0
0x00012654   stw  %o0, [ %g2 + 148 ]  {[prev_
0x00012658   xor  %o1 , %g3 , %g2
0x0001265c   and  %g2 , %l6 , %o4
0x00012660   sll  %o4, 1, %g2
0x00012664   lduh  [ %g2 + %l3 ], %g3
0x00012668   sth  %g3, [ %g4 + %l2 ]
0x0001266c   sth  %o3, [ %g2 + %l3 ]
0x00012670   bne,a  -15i  {deflate+119}
0x00012674   add  %o3 , 1,  %o3
deflate+119
0x00012634   sethi  %hi(0x72000), %g2
0x00012638   add  %o3 , %l4 , %g3
0x0001263c   lduw  [ %g2 + 148 ], %o0  {[prev_
0x00012640   sll  %o4 , 5,  %o1
0x00012644   and  %o3, %l6, %g4
0x00012648   ldub  [ %g3 - 1 ],   %g3
0x0001264c   sll  %g4, 1, %g4
0x00012650   subcc  %o0, 1, %o0
0x00012654   stw  %o0, [ %g2 + 148 ]  {[prev_
0x00012658   xor  %o1 , %g3 , %g2
0x0001265c   and  %g2 , %l6 , %o4
0x00012660   sll  %o4 , 1,  %g2
0x00012664   lduh  [ %g2 + %l3 ],  %g3
0x00012668   sth  %g3 , [ %g4 + %l2 ]
0x0001266c   sth  %o3, [ %g2 + %l3 ]
0x00012670   bne,a  -15i  {deflate+119}
0x00012674   add  %o3, 1, %o3
deflate+119
```

Time 89,179,264

Grid 40 insn X 5 cyc

11

Large-Window, Low Margin (gzip)

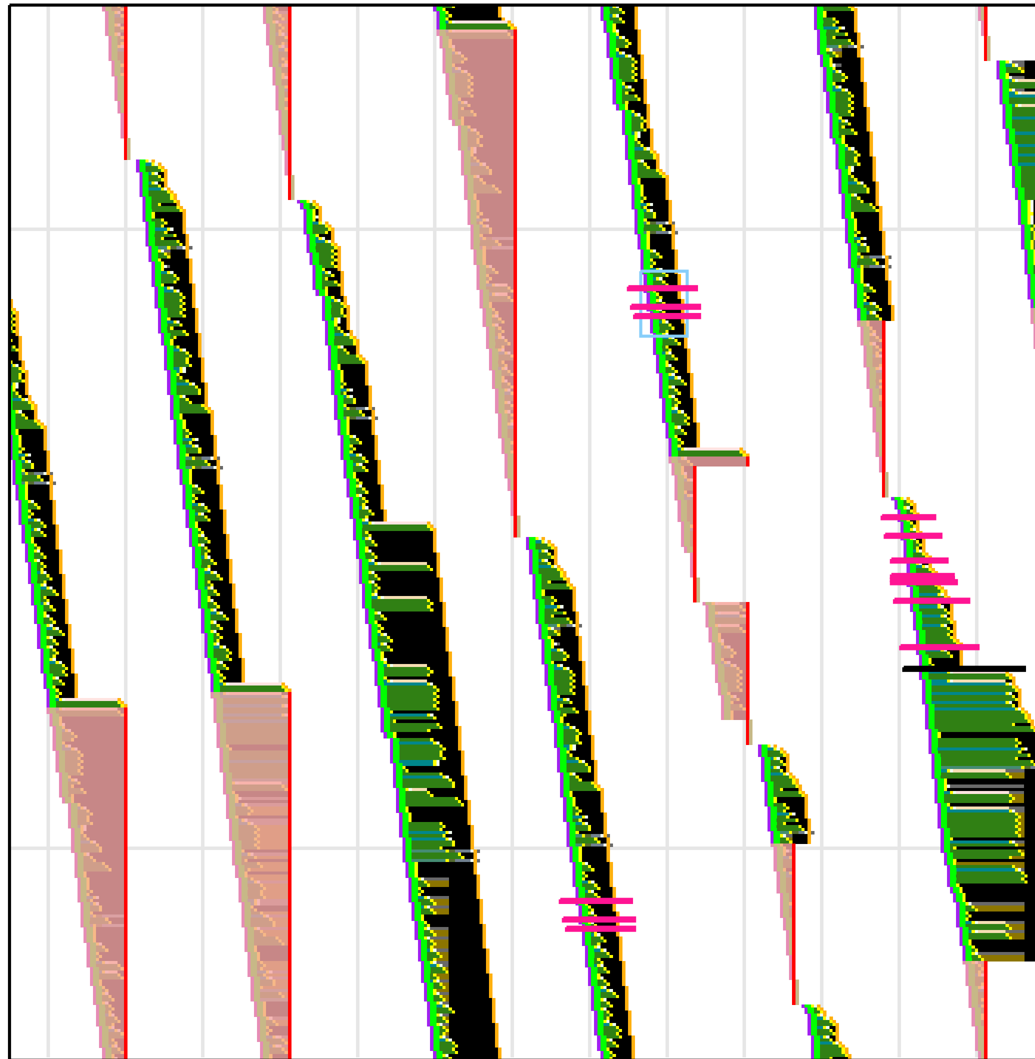


```
deflate+124
0x00012648   ldub  [ %g3 - 1 ],   %g3
0x0001264c   sll  %g4, 1, %g4
0x00012650   subcc  %o0, 1, %o0
0x00012654   stw  %o0, [ %g2 + 148 ]  {[prev_
0x00012658   xor   %o1 , %g3 , %g2
0x0001265c   and   %g2 , %l6 , %o4
0x00012660   sll  %o4, 1, %g2
0x00012664   lduh  [ %g2 + %l3 ], %g3
0x00012668   sth  %g3, [ %g4 + %l2 ]
0x0001266c   sth  %o3, [ %g2 + %l3 ]
0x00012670   bne,a  -15i  {deflate+119}
0x00012674   add   %o3 , 1,  %o3
deflate+119
0x00012634   sethi  %hi(0x72000), %g2
0x00012638   add   %o3 , %l4 , %g3
0x0001263c   lduw  [ %g2 + 148 ], %o0  {[prev
0x00012640   sll   %o4 , 5,  %o1
0x00012644   and  %o3, %l6, %g4
0x00012648   ldub  [ %g3 - 1 ],   %g3
0x0001264c   sll  %g4, 1, %g4
0x00012650   subcc  %o0, 1, %o0
0x00012654   stw  %o0, [ %g2 + 148 ]  {[prev_
0x00012658   xor   %o1 , %g3 , %g2
0x0001265c   and   %g2 , %l6 , %o4
0x00012660   sll   %o4 , 1,  %g2
0x00012664   lduh  [ %g2 + %l3 ],  %g3
0x00012668   sth   %g3 , [ %g4 + %l2 ]
0x0001266c   sth  %o3, [ %g2 + %l3 ]
0x00012670   bne,a  -15i  {deflate+119}
0x00012674   add  %o3, 1, %o3
deflate+119
```

Time 89,178,997

Grid 200 insn X 25 cyc

Higher Margin, Ignoring Mispredictions (gcc)

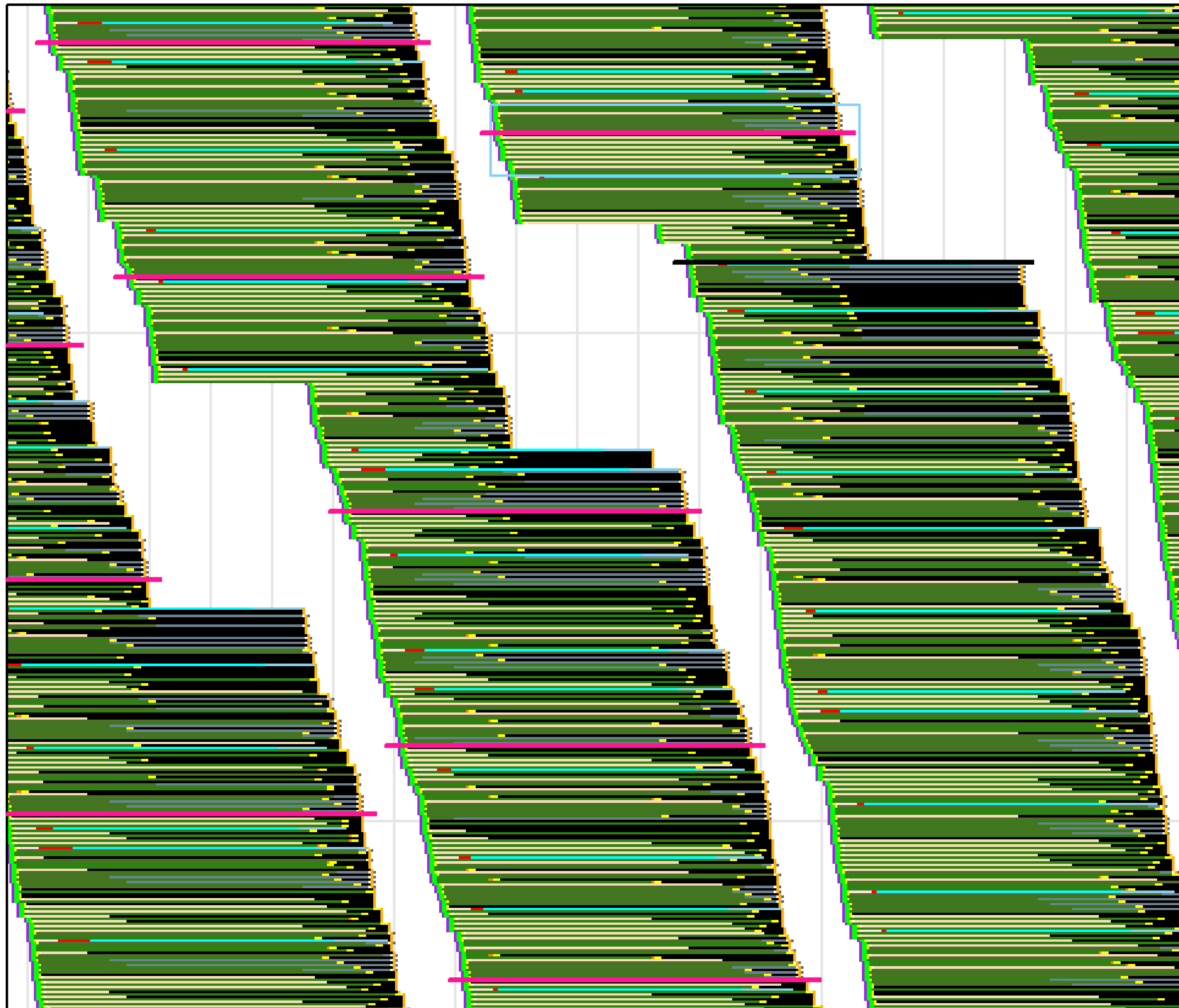


```
.LLM385+1  mark_target_live_regs+543  ../../gcc/resource.c:
0x001b4e24    cmp  %o0, 10
0x001b4e28    bne,a  +36i  {.LLM393+1  mark_target_live_
0x001b4e2c    lduw  [ %o4 + 8 ], %o4
.LLM393+1  mark_target_live_regs+580  ../../gcc/resource.c:
0x001b4eb8    cmp  %o4, 0
0x001b4ebc    bne,a  -38i  {.LLM385+1  mark_target_live_r
0x001b4ec4    ba  +49i  {.LLM403  mark_target_live_regs+
0x001b4ec8    mov  %l1 , %o2
.LLM403  mark_target_live_regs+632  ../../gcc/resource.c:15
0x001b4f88    cmp  %l1, 0
0x001b4f8c    be,a  +33i  {.LLM410  mark_target_live_regs
0x001b4f94    lduw  [ %l1 ], %o0
0x001b4f98    andcc  %o0, 32, %g0
0x001b4f9c    be,a  +14i  {.LLM407  mark_target_live_regs
0x001b4fa0    lduw  [  %o2 + 12 ],  %o2
.LLM407  mark_target_live_regs+651  ../../gcc/resource.c:16
0x001b4fd4    cmp  %o2, 0
0x001b4fd8    be  +14i  {.LLM410  mark_target_live_regs+
0x001b4fdc    mov  %o2 , %l1
0x001b4fe0    lduh  [ %o2 ], %o0
0x001b4fe4    cmp  %o0, 29
0x001b4fe8    bne  +11i  {.LLM410+1  mark_target_live_re
0x001b4fec    cmp  %l1, %i1
```

Time 83,166,363

Grid 200 insn X 25 cyc

High Margin (swim)



```
calc3_+241
0x0001337c  faddd  %f12, %f8, %f26
0x00013380  fmuld  %f22, %f14, %f2
0x00013384  fmuld  %f20, %f14, %f8
0x00013388  lddf  [ %l0 + %o1 ], %f12
0x0001338c  stdf  %f26, [ %l0 + %o0 ]
0x00013390  fsubd  %f30, %f16, %f16
0x00013394  stdf  %f30, [ %i3 + %o1 ]
0x00013398  fsubd  %f4, %f2, %f2
0x0001339c  stdf  %f4, [ %i5 + %o1 ]
0x000133a0  fsubd  %f10, %f8, %f8
0x000133a4  stdf  %f10, [ %l1 + %o1 ]
0x000133a8  faddd  %f16, %f0, %f0
0x000133ac  add  %o1 , 16,  %o0
0x000133b0  lddf  [ %i3 + %g2 ], %f30
0x000133b4  faddd  %f2, %f6, %f2
0x000133b8  lddf  [ %i5 + %g2 ], %f4
0x000133bc  fmuld  %f18, %f0, %f16
0x000133c0  lddf  [ %l1 + %g2 ], %f10
0x000133c4  faddd  %f8, %f12, %f8
0x000133c8  lddf  [ %l4 + %g2 ], %f0
0x000133cc  fmuld  %f18, %f2, %f2
0x000133d0  lddf  [ %l5 + %g2 ], %f6
0x000133d4  faddd  %f24, %f16, %f16
0x000133d8  lddf  [ %g4 + %g2 ], %f12
0x000133dc  fmuld  %f18, %f8, %f8
0x000133e0  lddf  [ %l2 + %g2 ], %f24
0x000133e4  stdf  %f16, [ %l2 + %o1 ]
```

Time 87,174,642

Grid 200 insn X 25 cyc

Hardware-Only Schemes

15

Collins, et al, Micro 2001.

- Dynamic Speculative Pre-Execution
- Troublesome load selection: frequent blocker of ROB.
- Construction window: 512 instructions.
- Slice: some optimization.
- Execution: up to 7 p-threads in ¤ight.

Moshovos, et al, ICS 2001

- Slice Processor
- Troublesome load selection: load miss predictor.
- Construction window: 32 instructions.
- Execution: up to 8 p-threads in ¤ight.

Pr fi ce Assisted Schemes

16

Roth & Sohi, HPCA 2001.

Data-Driven Multithreading

Troublesome load selection: trace analysis.

Construction window: 1024 instructions.

Execution: up to 3 p-threads in ¤ight.

Branches also pre-executed.

Results can be re-integrated. (No need to re-execute.)

Zilles & Sohi, ISCA 2001.

Troublesome load selection: trace analysis.

P-threads hand constructed and optimized.

Branches also pre-executed.

Remaining Issues

17

Focus of Other Studies

Resource consumption issues.

Slice construction and triggering techniques.

Remaining Issues

How well can ROB hide latency reduced by pre-execution?

How does fetch rate impact margin?

17

⋆⋆⋆ Deconstruction ⋆⋆⋆

18

Pre-Execution v. Enlarged Reorder Buffer (ROB)

Both hide load miss latency.

Both consume CPU core chip area.

Performance comparison helpful for tradeoffs.

Larger Reorder Buffer Advantage

No special hardware needed.

Pre-Execution Advantages

Reduced branch resolution time.

No need to schedule from enlarged window.

Speedup Components

Interesting difference: pre-execution can reduce branch resolution time.

Use two speedup components to measure effect.

Squash Speedup Component

Speedup due to reduced branch and jump resolution times.

Larger ROB cannot realize this type of speedup.

Stall Speedup Component

Speedup due to certain reduced correct-path stall cycles.

Both pre-execution and larger ROB can realize this speedup.

Manuscript Note

Manuscript uses less precise speedup components (see below).

Details of Speedup Components

Execution Time Components

Each cycle examine *slots* where hardware decodes instructions...
... n-way superscalar processor has n slots.

Find each of the following slot counts:

i_{qfw}, holds wrong-path instruction (not stalled),

i_{qsw}, holds stalled wrong-path instruction and ROB is full,

i_{qscc}, holds stalled correct-path instruction on mispredicted-branch or -jump resolution critical path and ROB is full,

i_{sf}, holds stalled correct-path instruction (not counting i_{qscc} instructions) and ROB is full,

i_{ofc}, holds correct-path instruction (not stalled),

i_{oe}, slot is empty or holds stalled instruction and ROB is not full.

Note: Full ROB causes pipeline to stall, stalls also caused by full instruction queues, lack of free physical registers, etc.

Component i_{qscc}, not used in manuscript.

Details of Speedup Components

21

Slot Categories

Let n denote decode width and T denote run time in cycles.

Total slots: $nT = i_{\mathrm{qfw}} + i_{\mathrm{qsw}} + i_{\mathrm{qscc}} + i_{\mathrm{sf}} + i_{\mathrm{ofc}} + i_{\mathrm{oe}}$

Squash slots: $i_{\mathrm{q}} = i_{\mathrm{qfw}} + i_{\mathrm{qsw}} + i_{\mathrm{qscc}}$

Stall slots: $i_{\mathrm{s}} = i_{\mathrm{sf}}$

Other slots: $i_{\mathrm{s}} = i_{\mathrm{ofc}} + i_{\mathrm{oe}}$

CPI is $\dfrac{i_{\mathrm{q}} + i_{\mathrm{s}} + i_{\mathrm{o}}}{n i_{\mathrm{ofc}}}$.

21

Let $i_{\mathrm{q}}(X)$, $i_{\mathrm{s}}(X)$, and $i_{\mathrm{o}}(X)$, denote counts for system X.

Total speedup of A over B is: $\dfrac{i_{\mathrm{q}}(B)+i_{\mathrm{s}}(B)+i_{\mathrm{o}}(B)}{i_{\mathrm{q}}(A)+i_{\mathrm{s}}(A)+i_{\mathrm{o}}(A)}$.

The *squash component of speedup* of A or B is: $\dfrac{i_{\mathrm{q}}(B)+i_{\mathrm{s}}(B)+i_{\mathrm{o}}(B)}{i_{\mathrm{q}}(A)+i_{\mathrm{s}}(B)+i_{\mathrm{o}}(B)}$.

The *stall component of speedup* of A or B is: $\dfrac{i_{\mathrm{q}}(B)+i_{\mathrm{s}}(B)+i_{\mathrm{o}}(B)}{i_{\mathrm{q}}(B)+i_{\mathrm{s}}(A)+i_{\mathrm{o}}(B)}$.

Base Simulated Systems

23

Common Parameter	Value
Decode Width	8-way superscalar
Reorder Buffer	256 instructions
Return-Address Stack	8 entries
L1 ICache	5-way, 328 kB, 256-B line
L1 DCache Hit Latency	2 cycle
L1 DCache	4-way, 16 kiB, 64-B line
L2 DCache	8-way, 256 kiB, 64-B line
L2 Hit Latency	16 cycles
L2 DCache Miss Latency	$\approx$ 100 cycles
ID to EX	3 cycles.
Global History	16 branches
Integer Units	80
Floating-Point Units	4
Memory Units	12
Instruction Queues	2048
Pre-Execution Parameter	**Value**
Construction Window	256
Injection Rate	4-insn / cycle
Maximum Slice Size	64 instructions
In-Flight Limit	8 slices
Slice Cache	2^{16} slices

Benchmarks

24

Chosen for pre-execution suitability and comparability.

- Olden: mst, em3d
- SPEC2000/Ref Inputs: vpr

Other benchmarks.

- SPEC2000/Ref Inputs: mcf, wupwise
- SPEC2000/Test Inputs: swim
- SPEC2000/Other Inputs: gzip, gcc, perl, TeX.

24

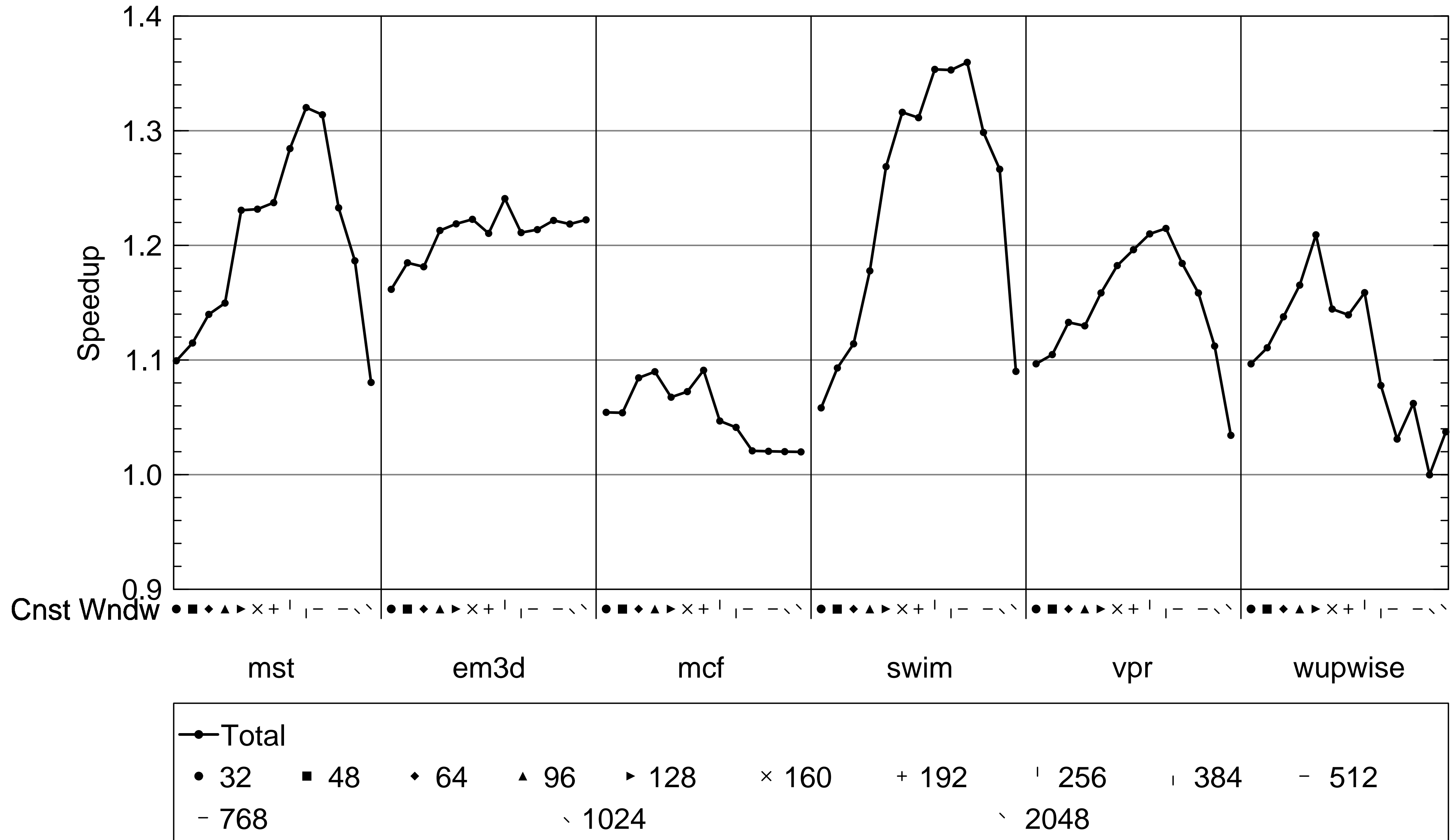
Pre-Execution Characterization, Construction Window Size

25

Speedup v. Construction Window

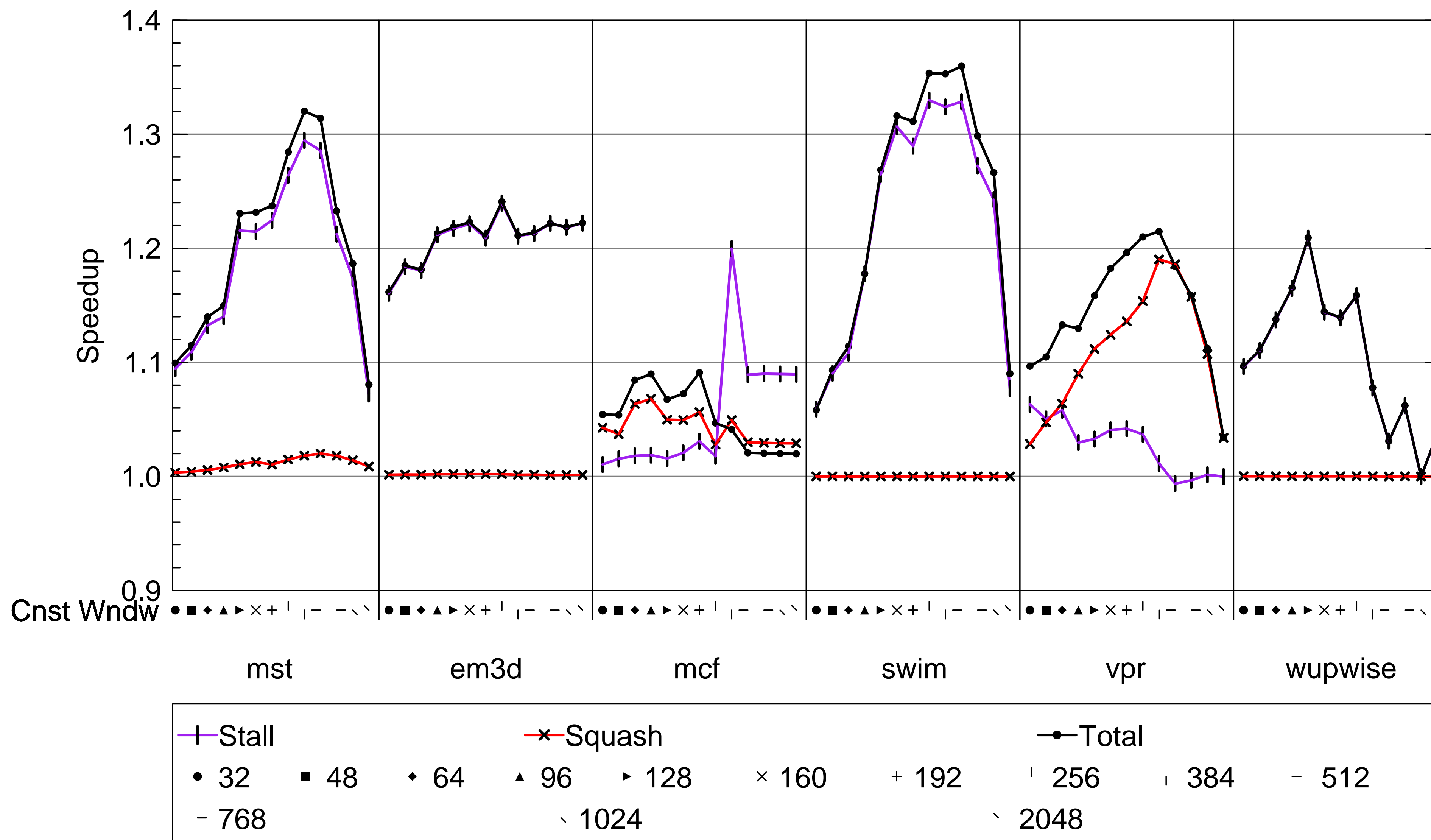
Vary construction window from 32 to 2048 instructions.

Construction Window Size, Benchmark Set 1



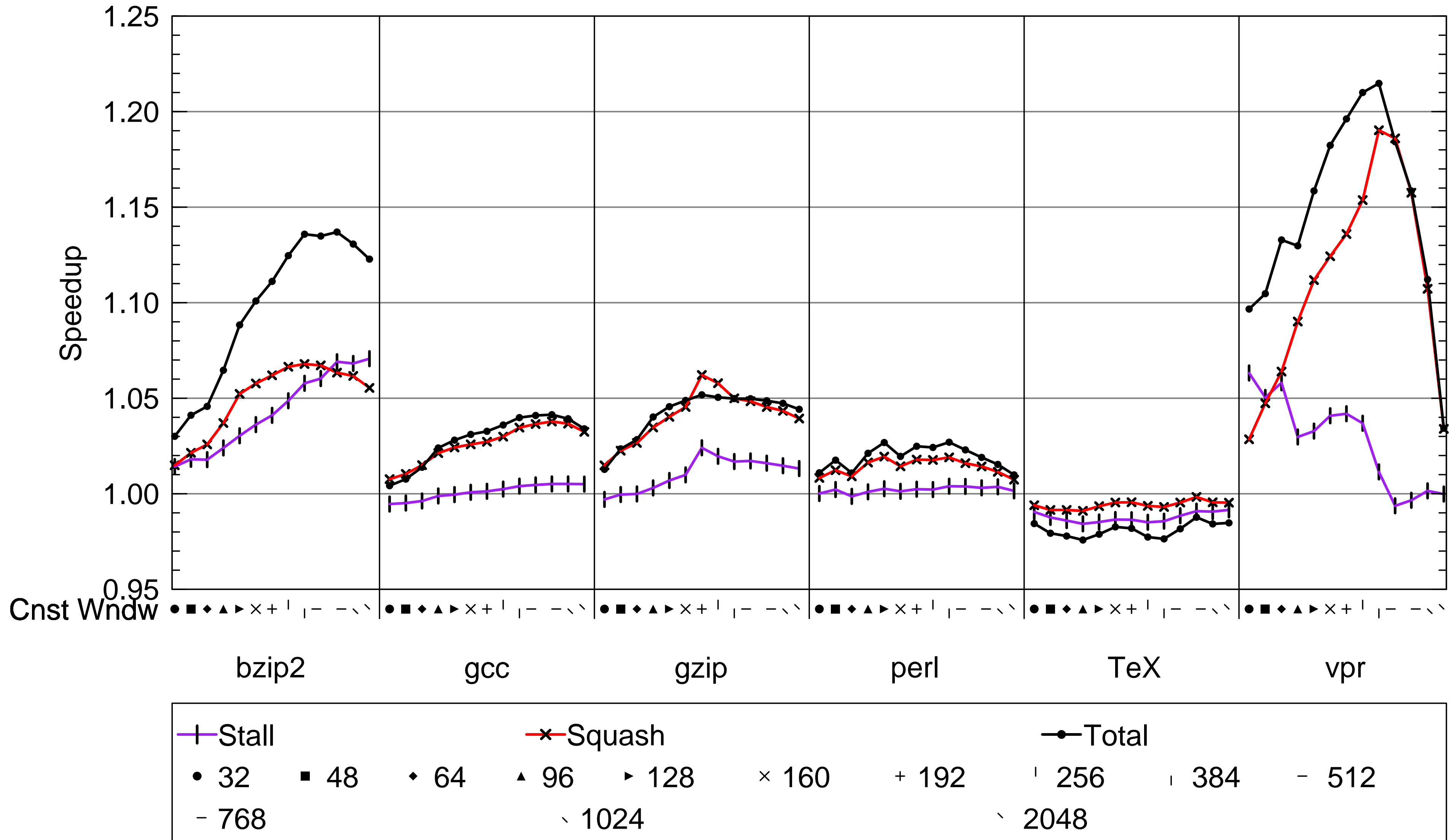
Construction Window Size, Benchmark Set 1

27

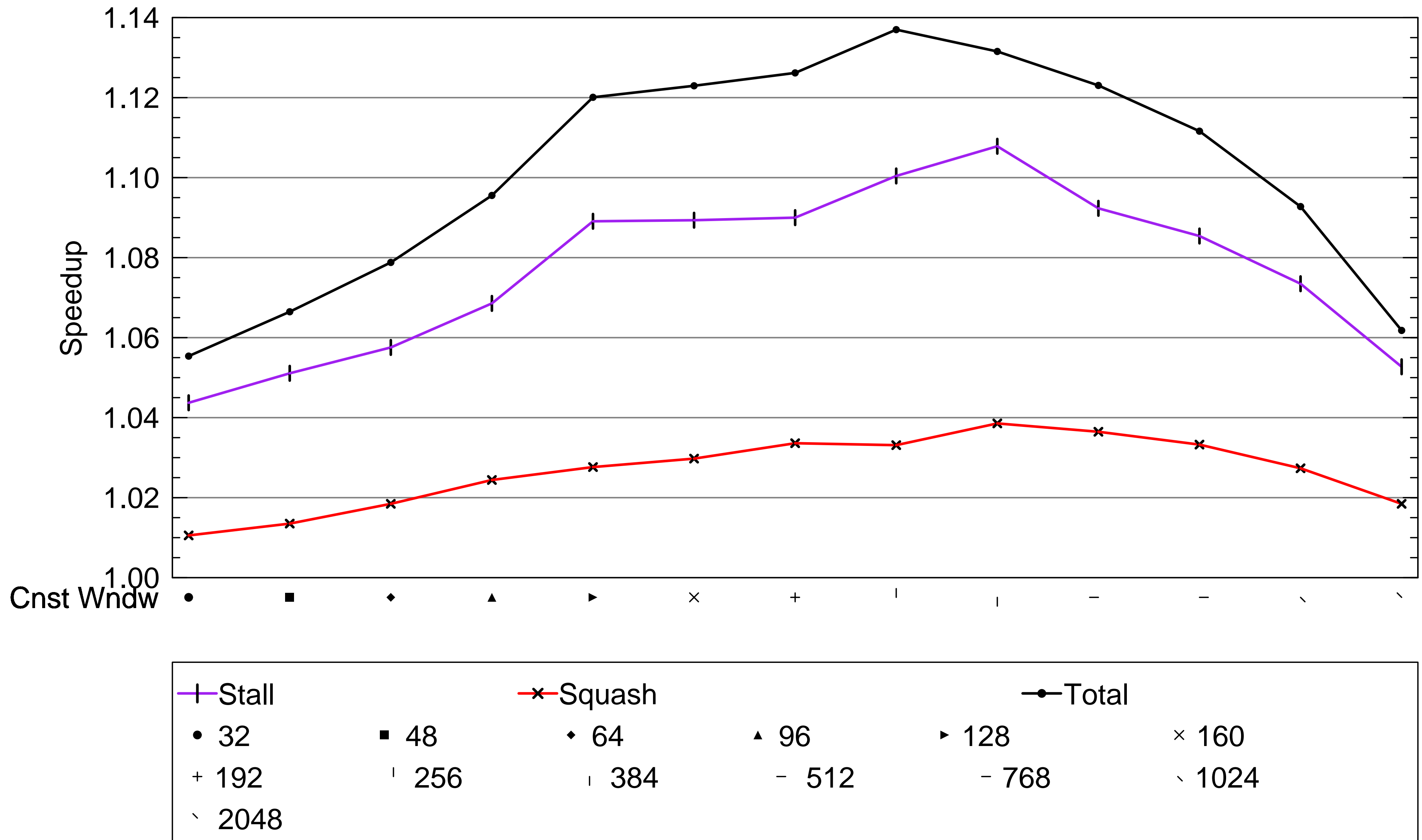


27

Construction Window Size, Benchmark Set 2



Construction Window Size, Average



Construction Window Size Comments

Average performance maxes at 256, at maximum margin of 32 cycles.

Limit appears due to limit of 8 threads.

Large Speedups

Large number of level-2 cache misses.

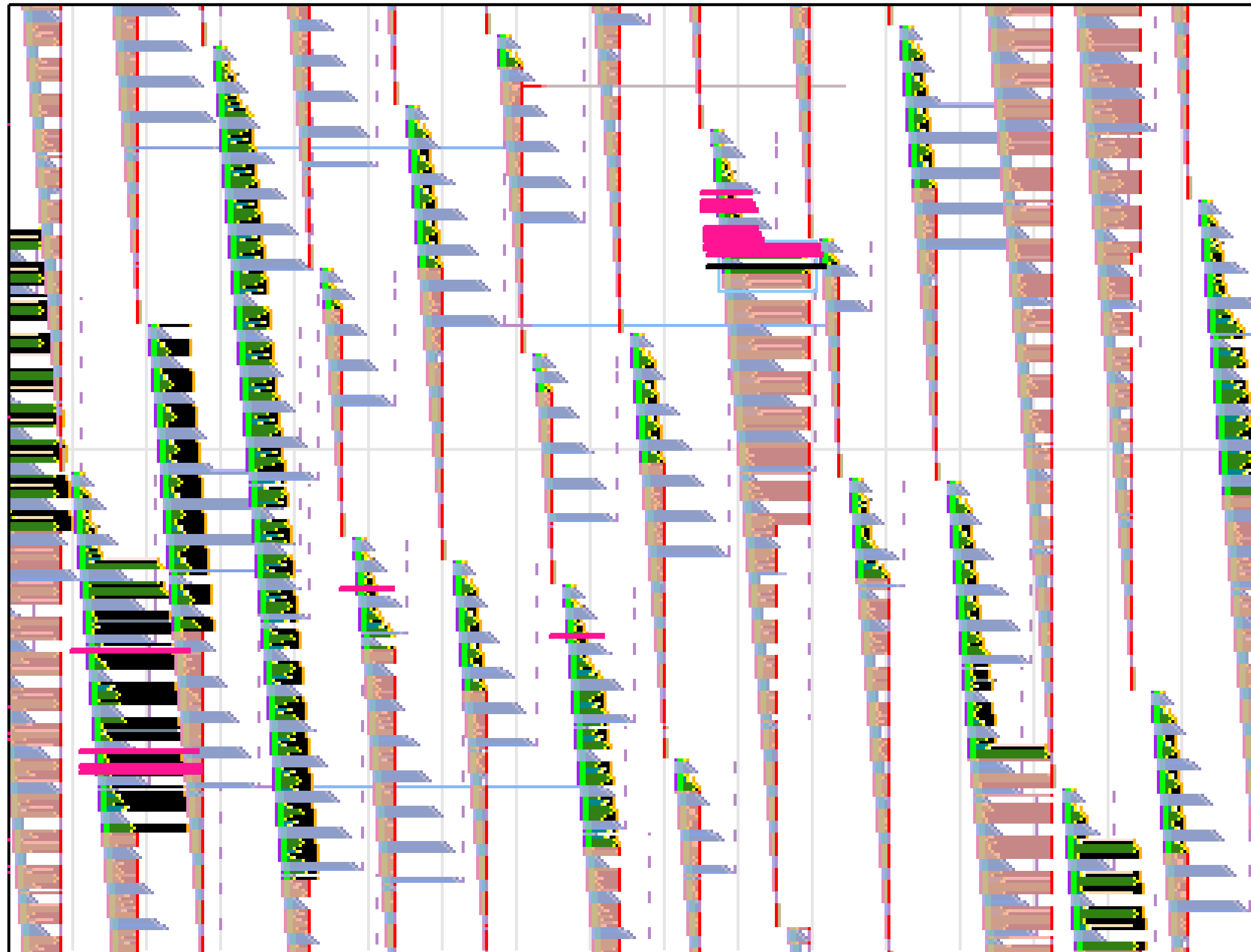
Speedup due mostly to stall component (except vpr).

Smaller Speedups

Squash plays a larger roll.

Results comparable with other studies.

Pre-Execution in bzip2

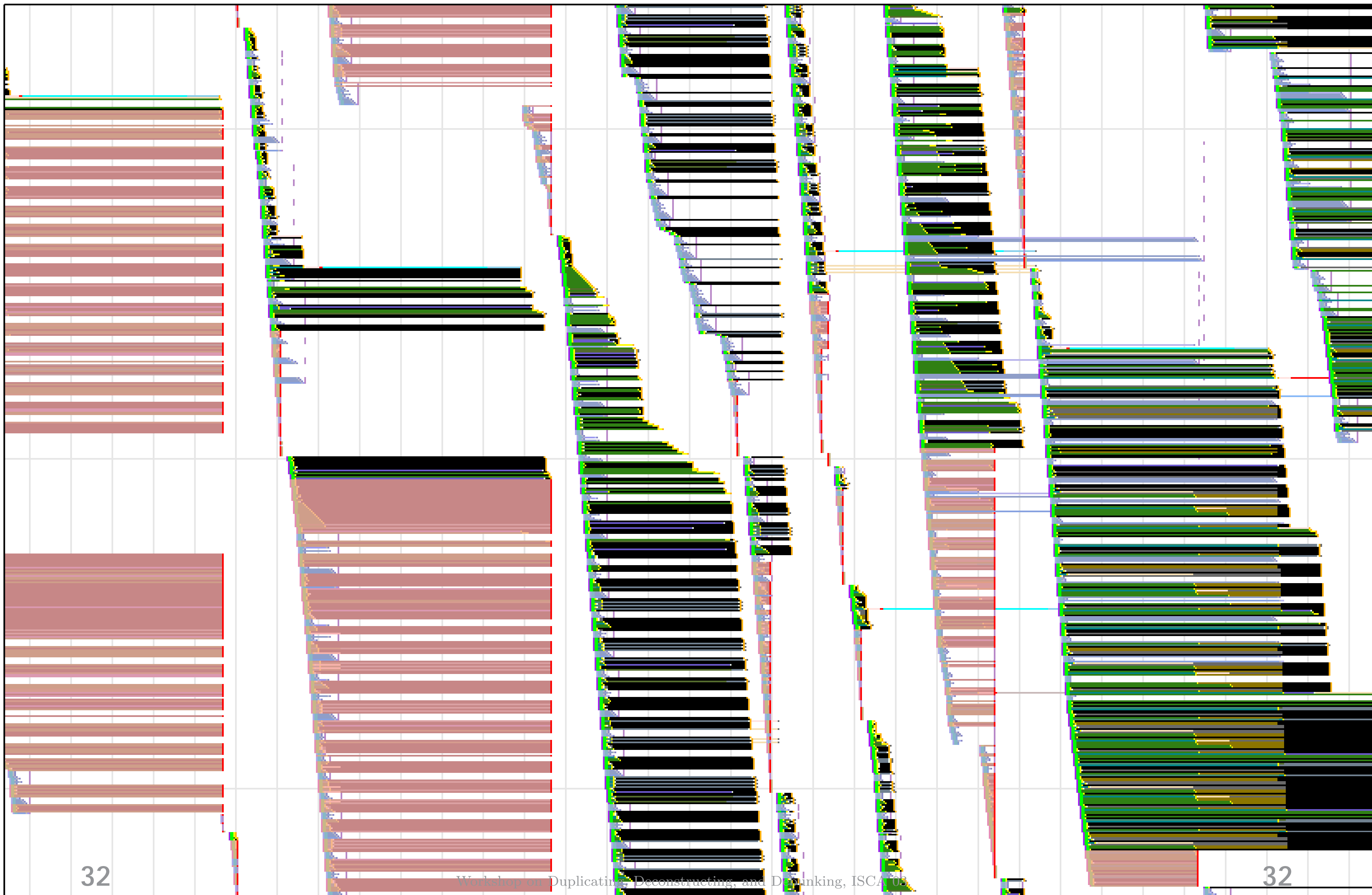


```
.LLM189  mainSimpleSort+45  blocksort.c:563
0x0001f058    add  %i0 , %l1 , %i0
0x0001f05c    mov  %g2, %o0
0x0001f060    ldub  [ %i1 + %i0 ], %g3
0x0001f064    ldub  [ %i1 + %g1 ], %g2
.LLM384+1  mainSimpleSort+533  blocksort.c:42
0x0001f7f8    ldub  [ %i1 + %g1 ], %g2
.LLM190+3  mainSimpleSort+49  blocksort.c:408
0x0001f068    cmp  %g3 , %g2
.LLM280  mainSimpleSort+272  blocksort.c:574
0x0001f3e4    add  %o7, 1, %o7
.LLM374  mainSimpleSort+504  blocksort.c:588
0x0001f784    add  %o7, 1, %o7
.LLM465+2  mainSimpleSort+731  blocksort.c:59
0x0001fb10    mov  %o7, %i0
.LLM190+4  mainSimpleSort+50  blocksort.c:408
0x0001f06c    bne  +80i  {.LLM212+6  mainSi
0x0001f070    cmp  %g2, %g3
0x0001f074    add  %i0, 1, %i0
0x0001f078    add  %g1, 1, %g1
0x0001f07c    ldub  [ %i1 + %i0 ], %g3
0x0001f080    ldub  [ %i1 + %g1 ], %g2
```

Time 40,080,754

Grid 200 insn X 25 cyc

Pre-Execution in vpr



Time 117,235,010

Grid 200 insn X 25 cyc

ROB Size $_3$ Construction Window Size

33

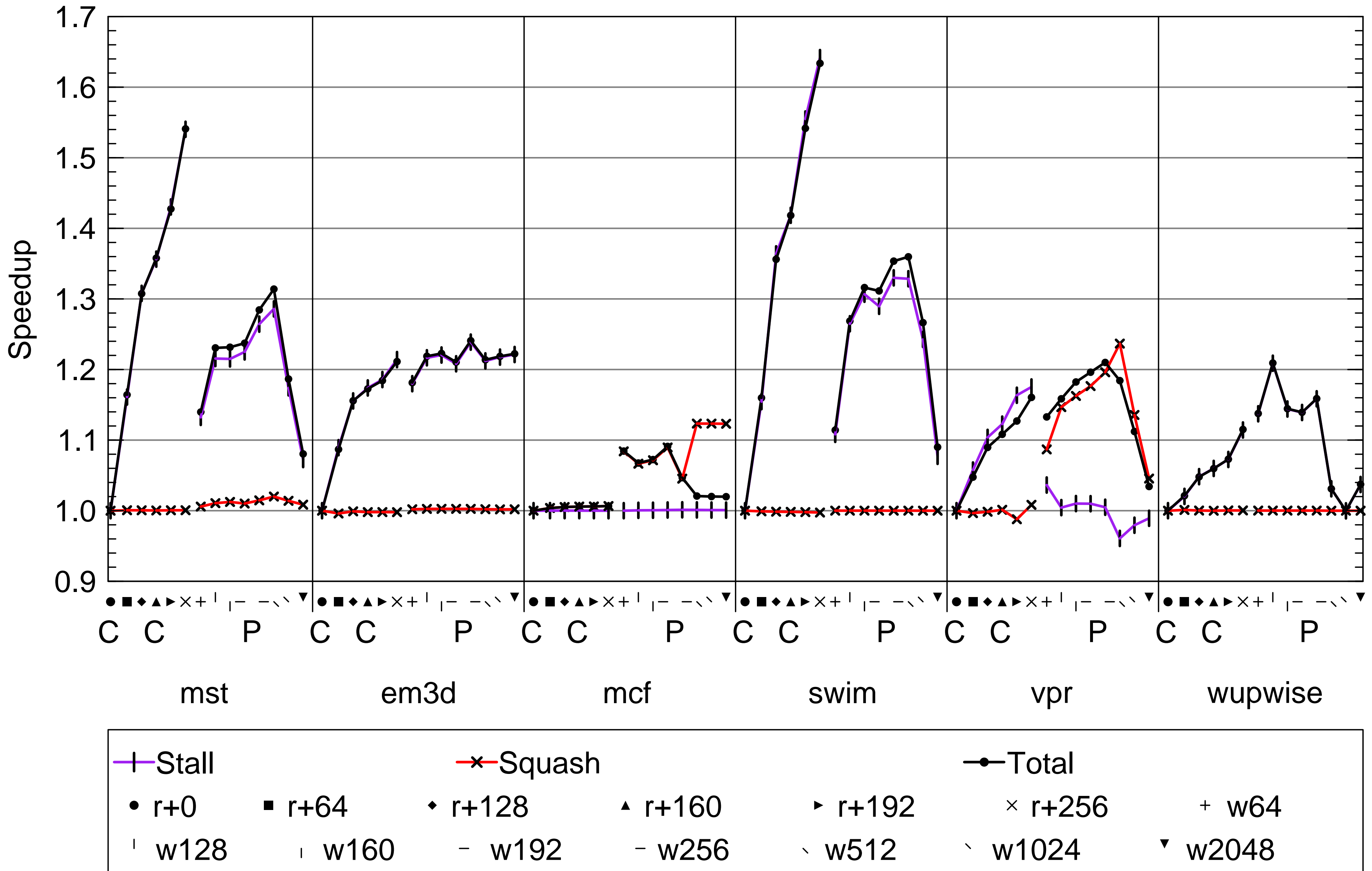
Conventional Systems: ROB 256 to 512.

Pre-Execution Systems: Construction Window 32 - 2048,. . .
. . . increments match change in ROB size.

Speedup is over conventional system with 256-entry ROB.

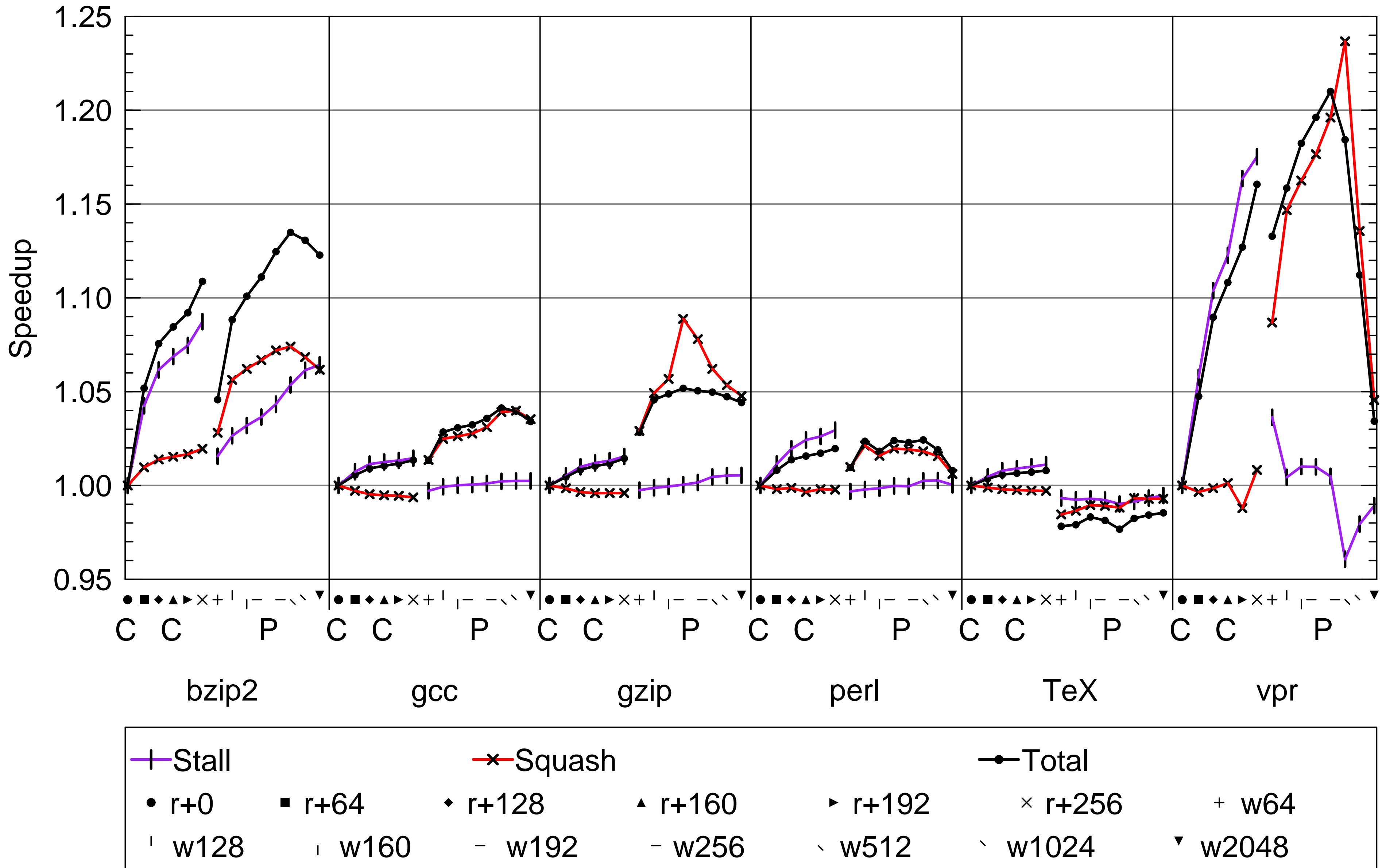
33

Reorder Buffer 256-512, Construction Window 32-2048, Set 1



Reorder Buffer 256-512, Construction Window 32-2048, Set 2

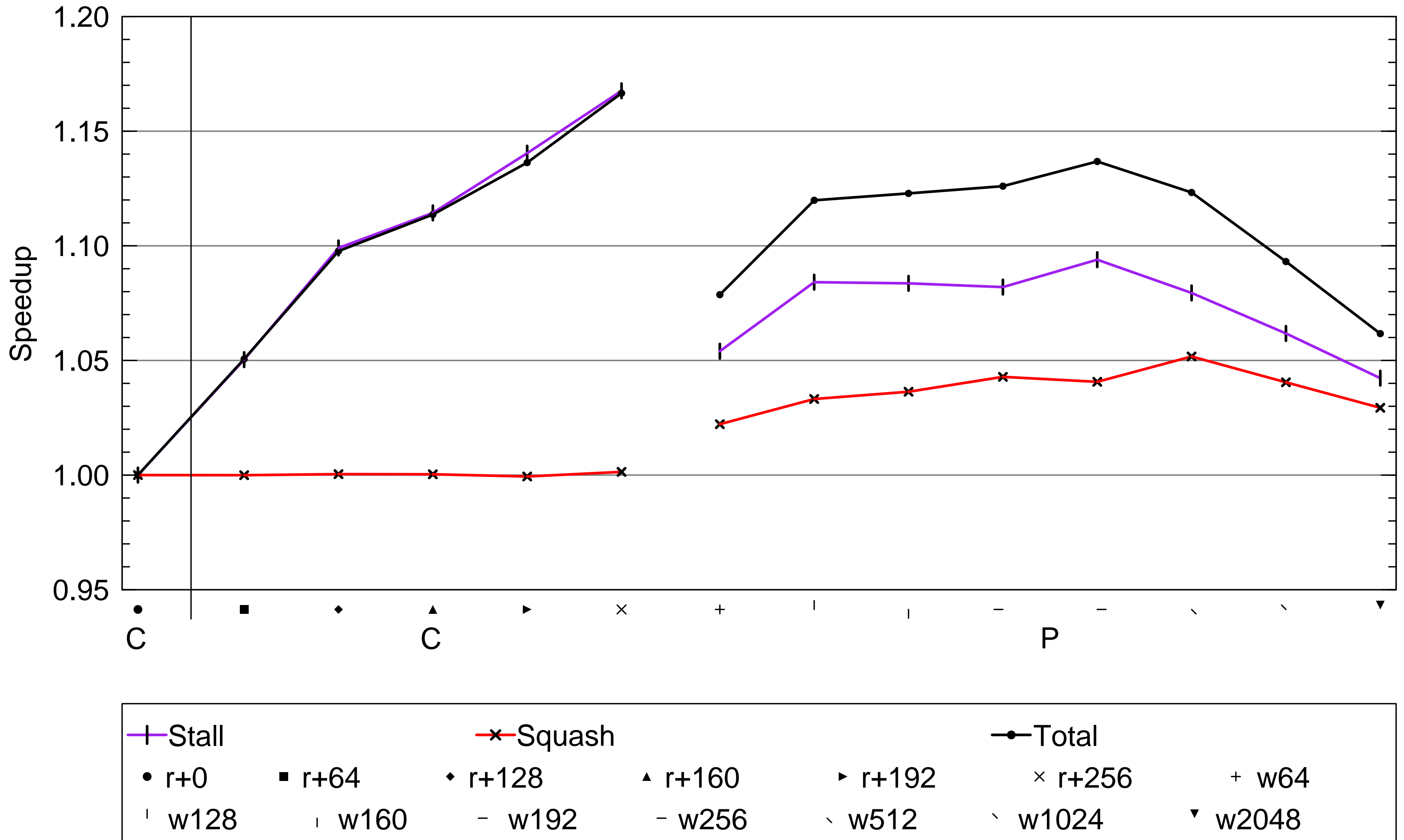
35



35

Reorder Buffer 256-512, Construction Window 32-2048, Average

36



36

Results

ROB more effective at stall component.

Exceptions are em3d and wupwise.

Pre-execution's impact on squash provides advantage over ROB for many benchmarks.

Pre-Execution v. 512-Entry ROB with Varying Pipeline Lengths

38

Doubling ROB size (and scheduling queues) might have a critical path impact.

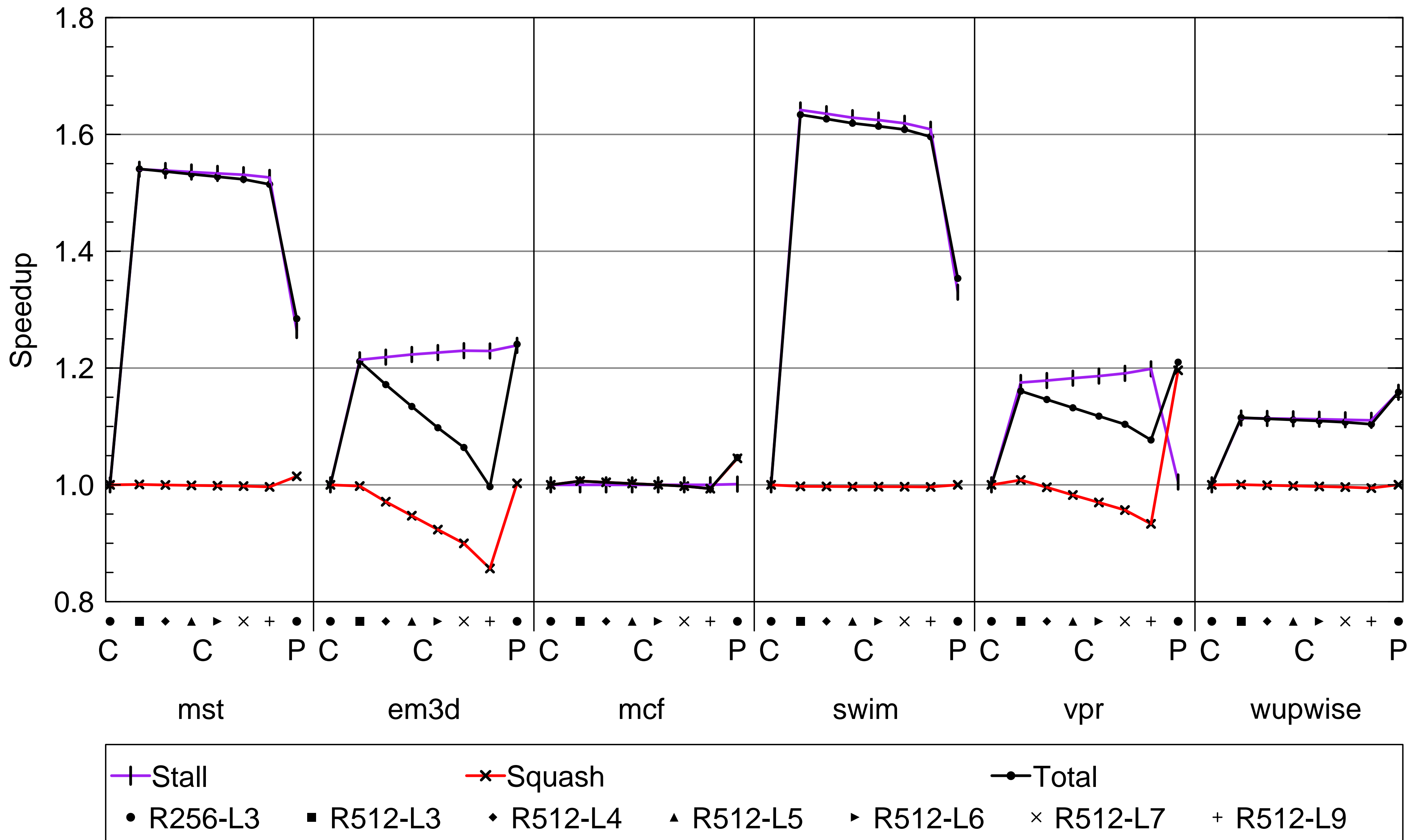
Compensate by increasing minimum number of cycles from decode to execute (ID-to-EX).

Systems

- Conventional, 256-entry ROB, 3-cycle ID-to-EX pipeline.
- Conventional, 512-entry ROB, 3- to 9-cycle ID-to-EX pipeline.
- Pre-execution, 256-entry ROB, 3-cycle ID-to-EX pipeline.

Pipeline Lengths, Set 1

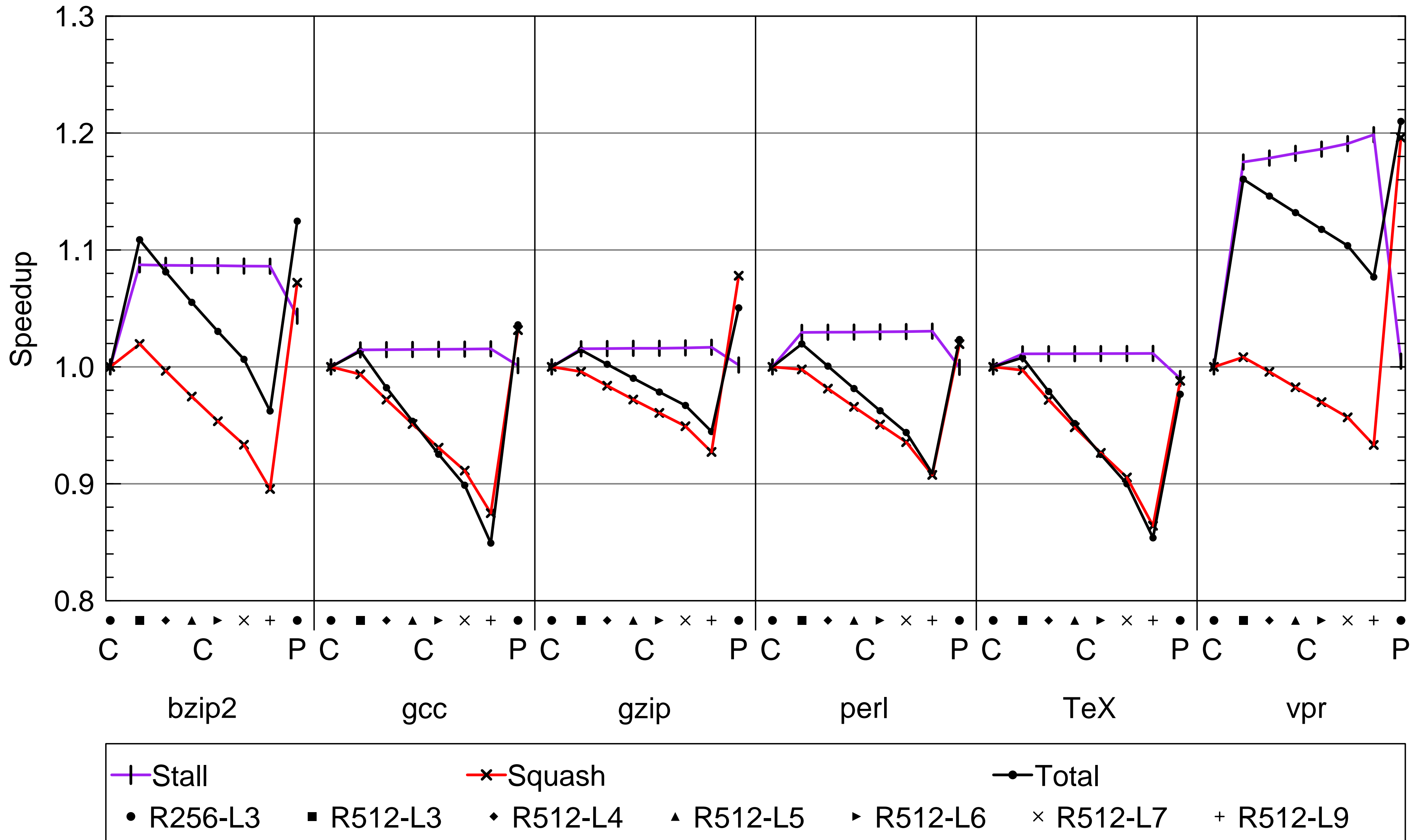
39



39

Pipeline Lengths, Set 2

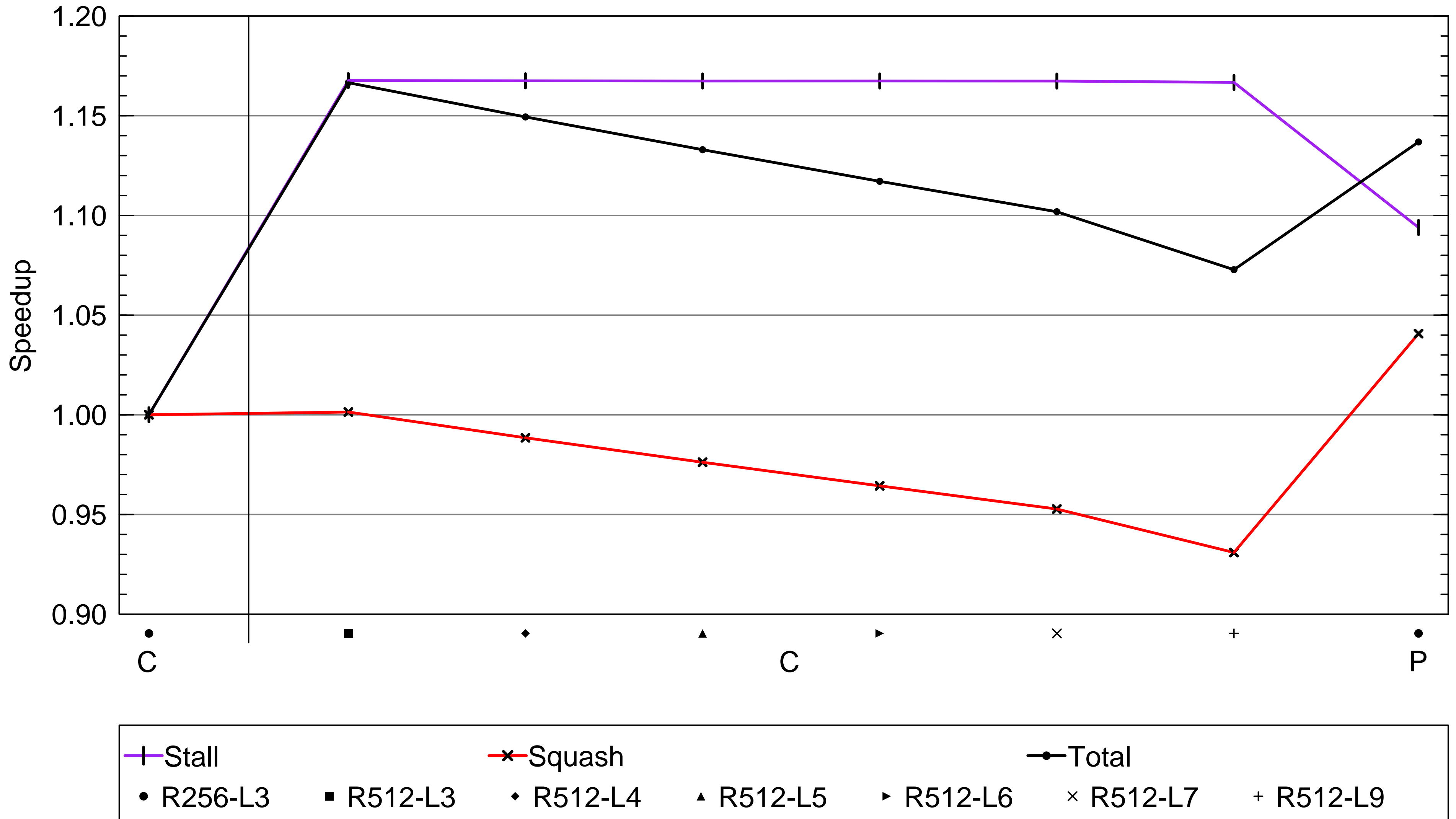
40



40

Pipeline Lengths, Average

41



41

Pre-Execution v. 512-Entry ROB with Varying Pipeline Lengths

42

Results

Pipeline delay handicaps conventional system in ways pre-execution helps.

Pre-Execution and Fetch Rate

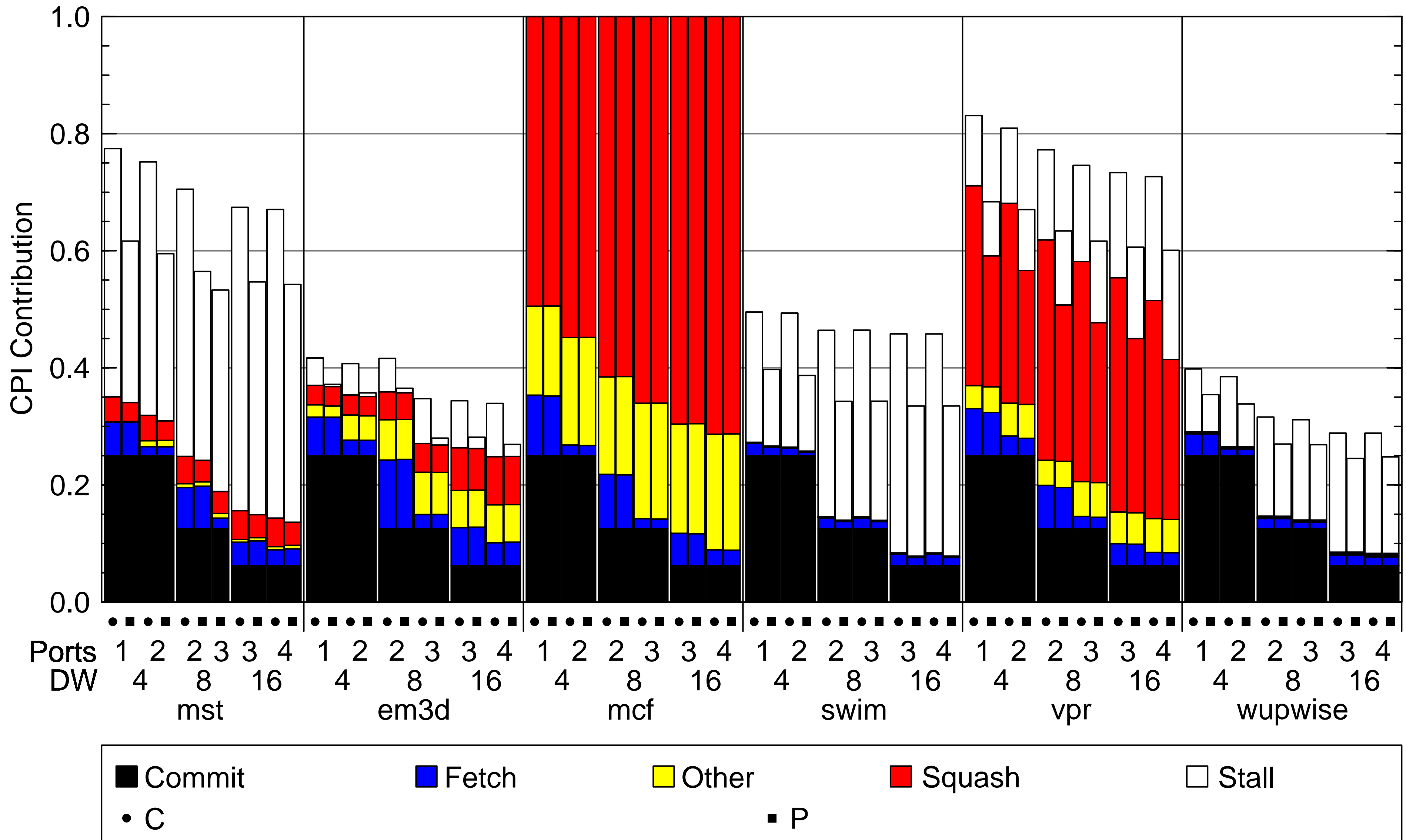
Systems (Conventional and Pre-Execution)

4-way, 1- and 2-cache ports.

8-way, 2- and 3-cache ports. (Three-port predicts two blocks/cycle.)

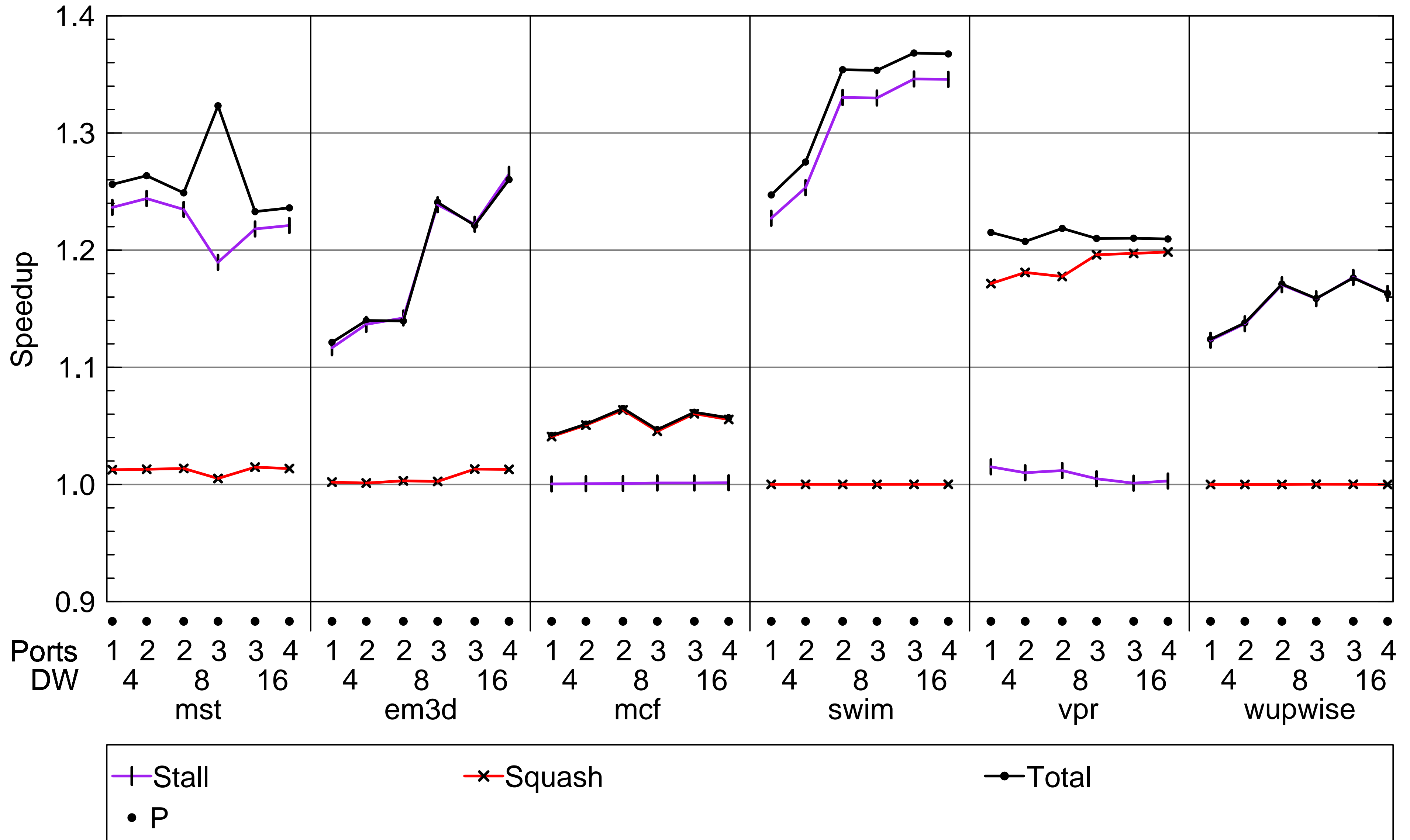
16-way, 3- and 4-cache ports. (Four-port predicts three blocks/cycle.)

Fetch Rate Set 1, CPI



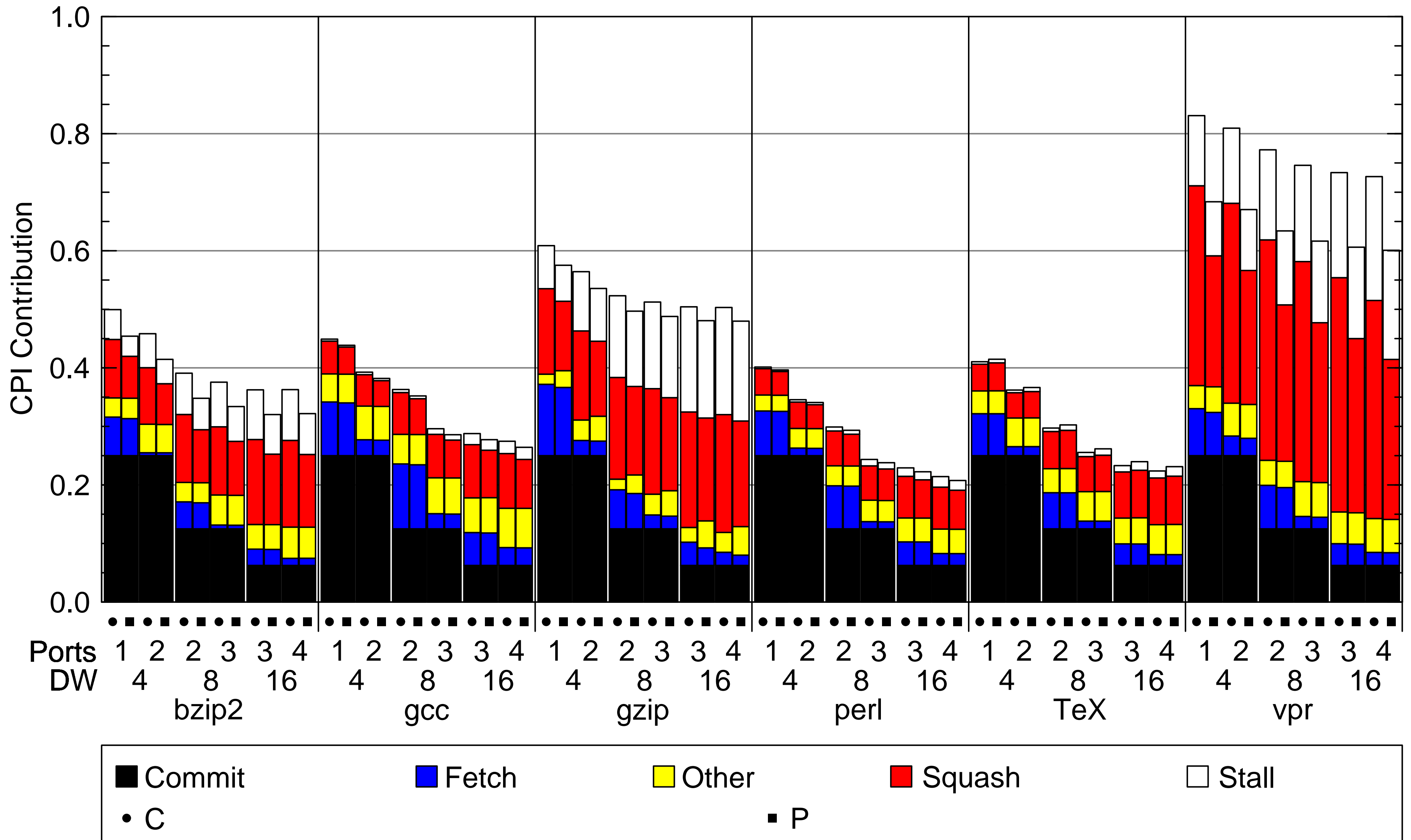
Fetch Rate Set 1, Speedup

45



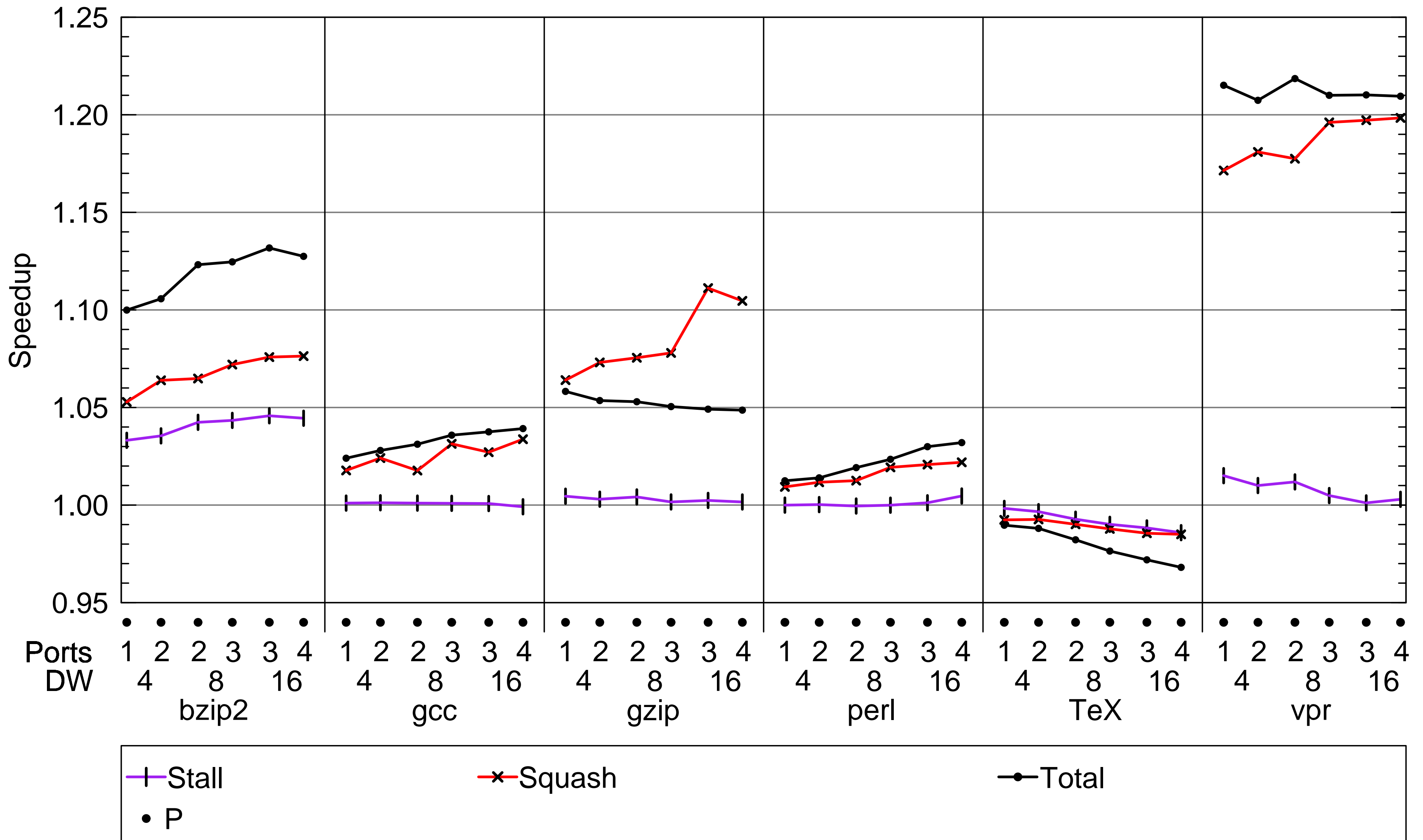
45

Fetch Rate Set 2, CPI



Fetch Rate Set 2, Speedup

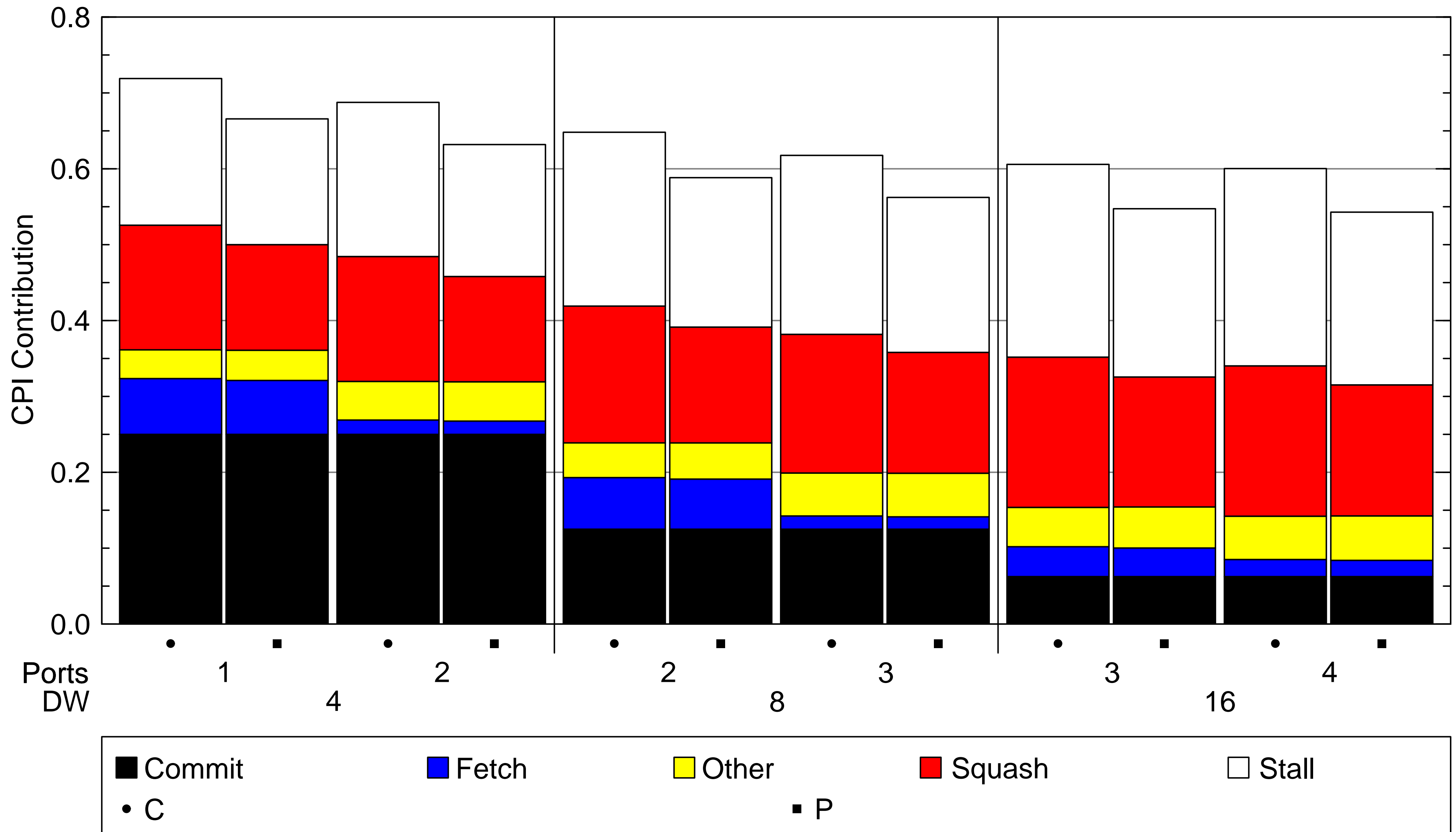
47



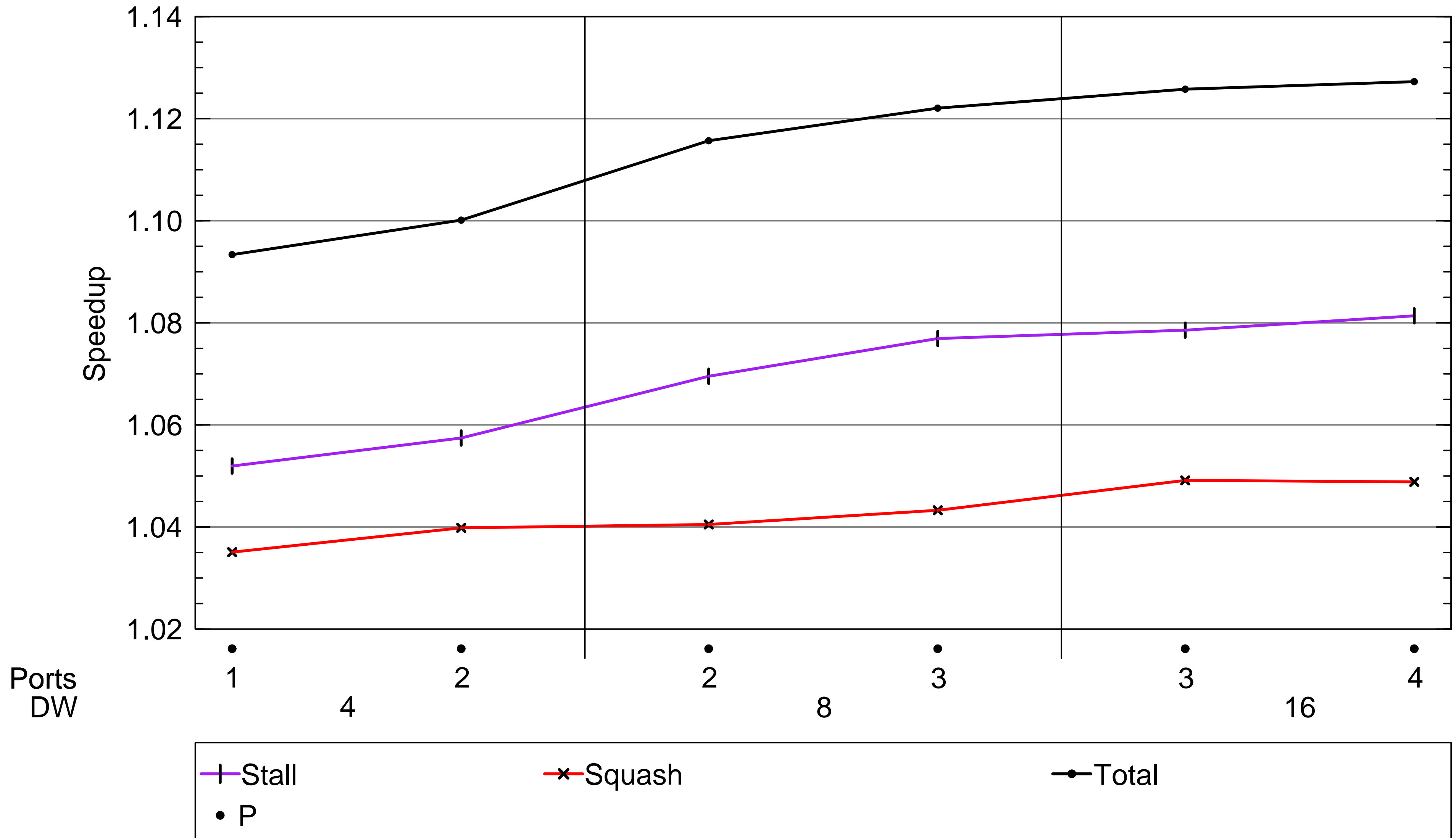
47

Fetch Rate Average, CPI

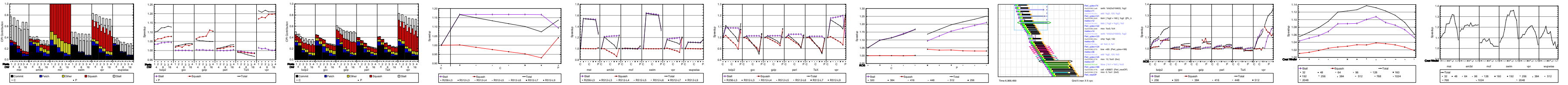
48



Fetch Rate Average, Speedup



Conclusions



Doubled ROB size can attain comparable performance . . .
. . . though pre-execution better on many benchmarks.

When larger ROB “paid for” with longer pipeline, pre-execution is clearly better.

Fetch rate has minor impact on pre-execution effectiveness.