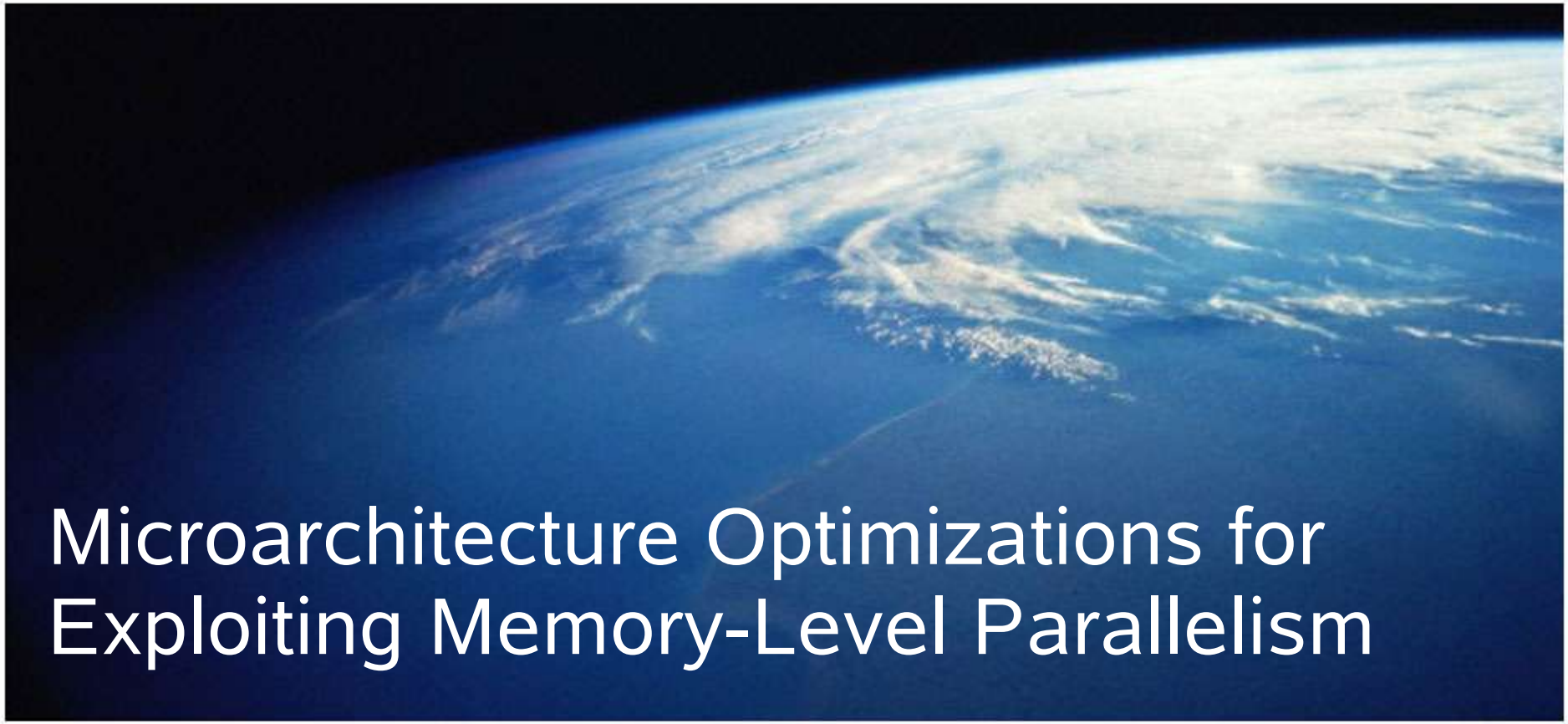


Microarchitecture Optimizations for Exploiting Memory-Level Parallelism

Yuan Chou, Brian Fahs, Santosh Abraham
Architecture and Advanced Development
Scalable Systems Group
Sun Microsystems

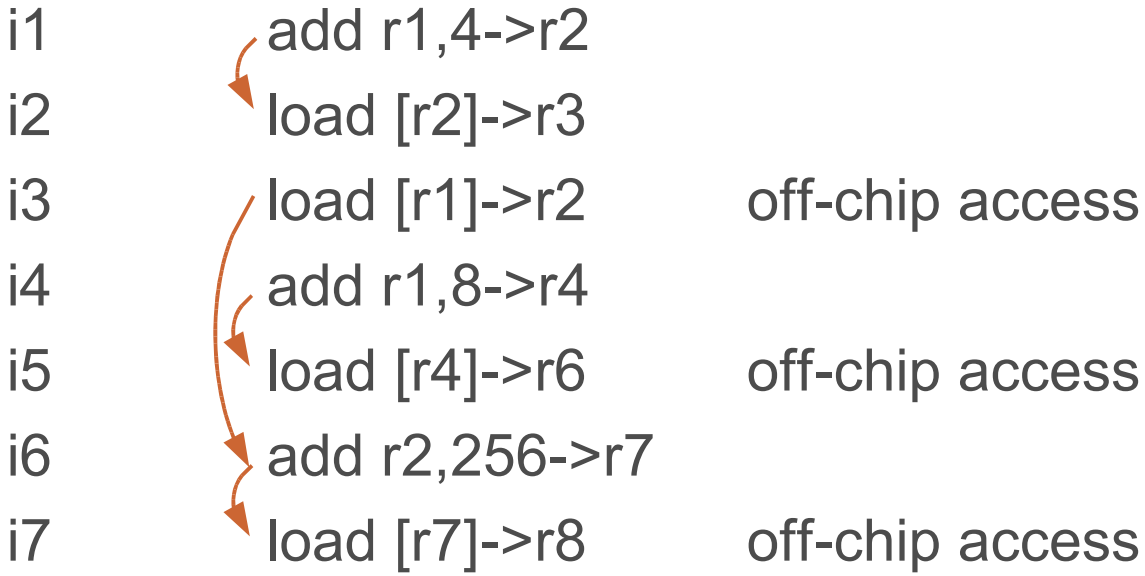
Sun microsystems
We make the net work.

Motivation

- Performance of many commercial apps bound by cost of memory accesses – in today's database workloads, 2/3 execution time spent in memory accesses
- Instruction-level parallelism ineffective in hiding latency of memory accesses – this latency continues to grow
- Promising alternative is to exploit memory-level parallelism (MLP) and overlap memory accesses with each other

What is MLP?

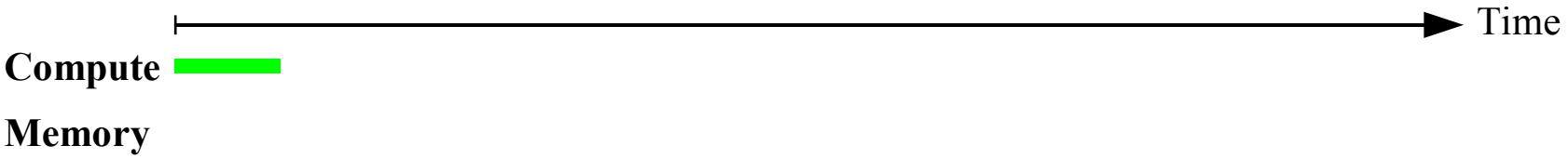
Example:



What is MLP?

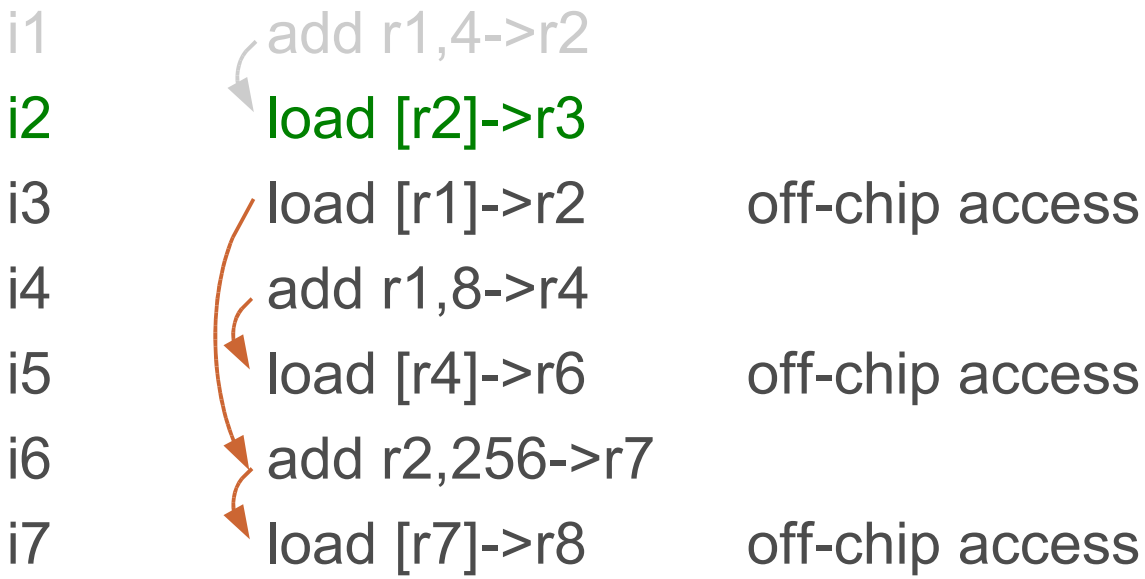
Example:

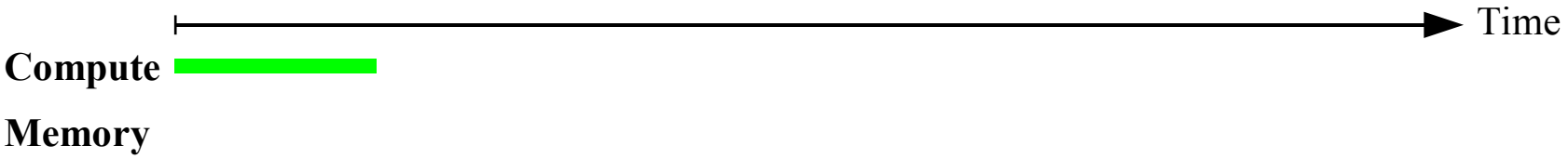
```
i1    add r1,4->r2
i2    load [r2]->r3
i3    load [r1]->r2      off-chip access
i4    add r1,8->r4
i5    load [r4]->r6      off-chip access
i6    add r2,256->r7
i7    load [r7]->r8      off-chip access
```



What is MLP?

Example:





What is MLP?

Example:

```
i1    add r1,4->r2
i2    load [r2]->r3
i3    load [r1]->r2      off-chip access
i4    add r1,8->r4
i5    load [r4]->r6      off-chip access
i6    add r2,256->r7
i7    load [r7]->r8      off-chip access
```

What is MLP?

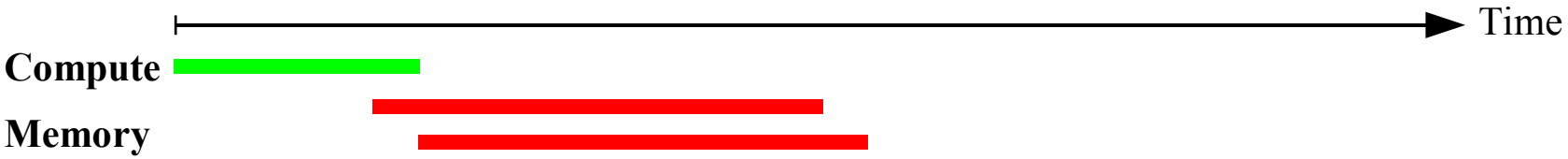
Example:

```
i1    add r1,4->r2
i2    load [r2]->r3
i3    load [r1]->r2      off-chip access
i4    add r1,8->r4
i5    load [r4]->r6      off-chip access
i6    add r2,256->r7
i7    load [r7]->r8      off-chip access
```

What is MLP?

Example:

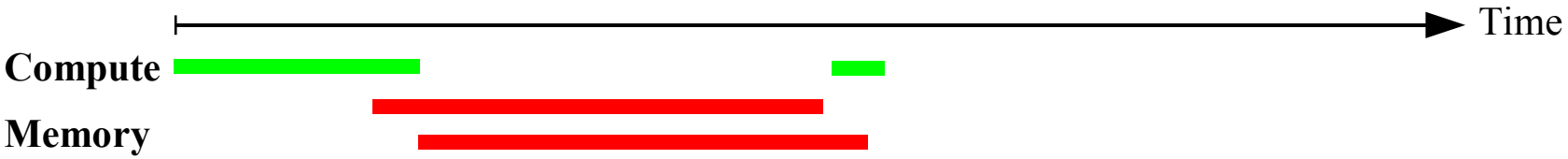
```
i1        add r1,4->r2
i2        load [r2]->r3
i3        load [r1]->r2        off-chip access
i4        add r1,8->r4
i5        load [r4]->r6        off-chip access
i6        add r2,256->r7
i7        load [r7]->r8        off-chip access
```



What is MLP?

Example:

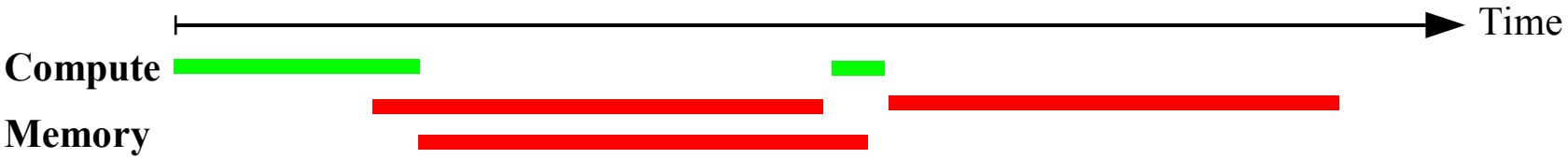
```
i1    add r1,4->r2
i2    load [r2]->r3
i3    load [r1]->r2        off-chip access
i4    add r1,8->r4
i5    load [r4]->r6        off-chip access
i6    add r2,256->r7
i7    load [r7]->r8        off-chip access
```



What is MLP?

Example:

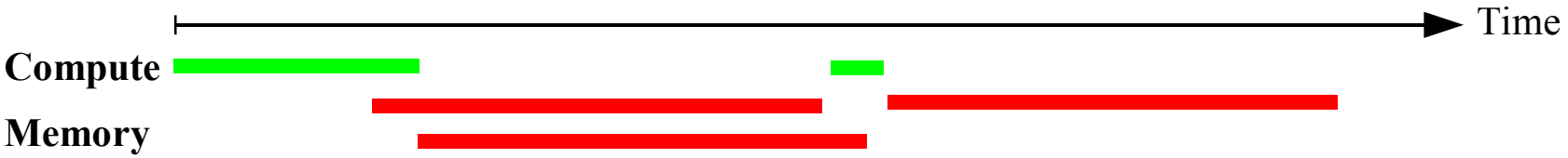
```
i1    add r1,4->r2
i2    load [r2]->r3
i3    load [r1]->r2        off-chip access
i4    add r1,8->r4
i5    load [r4]->r6        off-chip access
i6    add r2,256->r7
i7    load [r7]->r8        off-chip access
```



What is MLP?

Some notion of off-chip accesses being serviced in parallel ...

```
i1    add r1,4->r2
i2    load [r2]->r3
i3    load [r1]->r2      off-chip access
i4    add r1,8->r4
i5    load [r4]->r6      off-chip access
i6    add r2,256->r7
i7    load [r7]->r8      off-chip access
```



What is MLP?

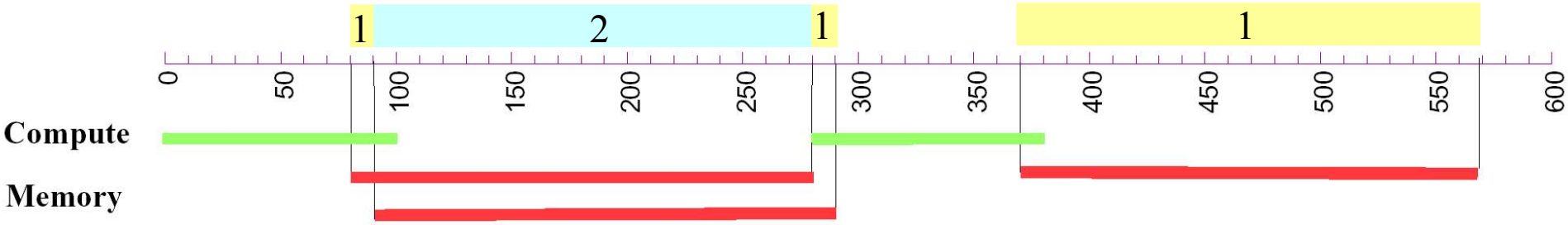
More precisely:

MLP = average number of useful long-latency off-chip accesses outstanding when there is at least one such access outstanding

What is MLP?

More precisely:

MLP = average number of useful long-latency off-chip accesses outstanding when there is at least one such access outstanding

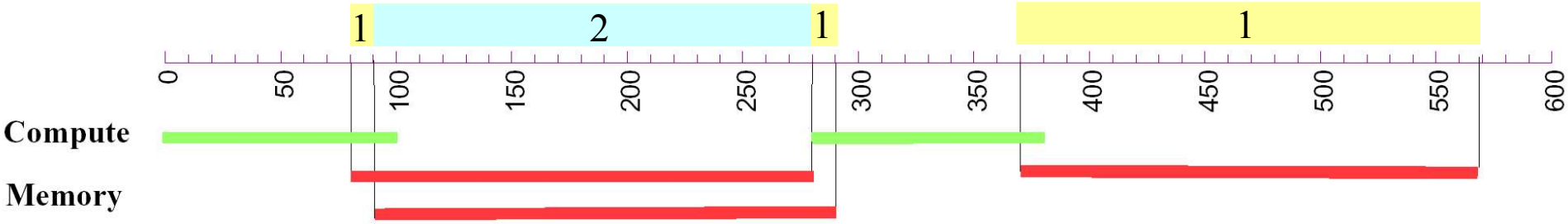


What is MLP?

More precisely:

MLP = average number of useful long-latency off-chip accesses outstanding when there is at least one such access outstanding

MLP = [(10+10+100)*1 + (90)*2] / 210 = 1.43



MLP and Overall Performance

$$CPI = CPI_{perf}(1 - Overlap_{CM}) + \frac{MissRate \times MissPenalty}{MLP}$$

on-chip CPI

off-chip CPI

MLP and Overall Performance

$$CPI = CPI_{perf}(1 - Overlap_{CM}) + \frac{MissRate \times MissPenalty}{MLP}$$

on-chip CPI

off-chip CPI

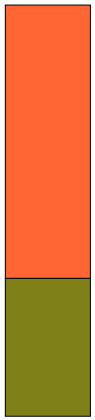
Example:

MLP = 1

off-chip CPI = 2

on-chip CPI = 1

CPI = 3



MLP and Overall Performance

$$CPI = CPI_{perf}(1 - Overlap_{CM}) + \frac{MissRate \times MissPenalty}{MLP}$$

on-chip CPI

off-chip CPI

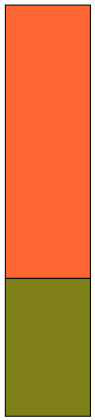
Example:

MLP = 1

off-chip CPI = 2

on-chip CPI = 1

CPI = 3

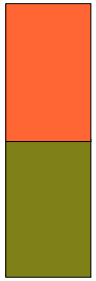


MLP = 2

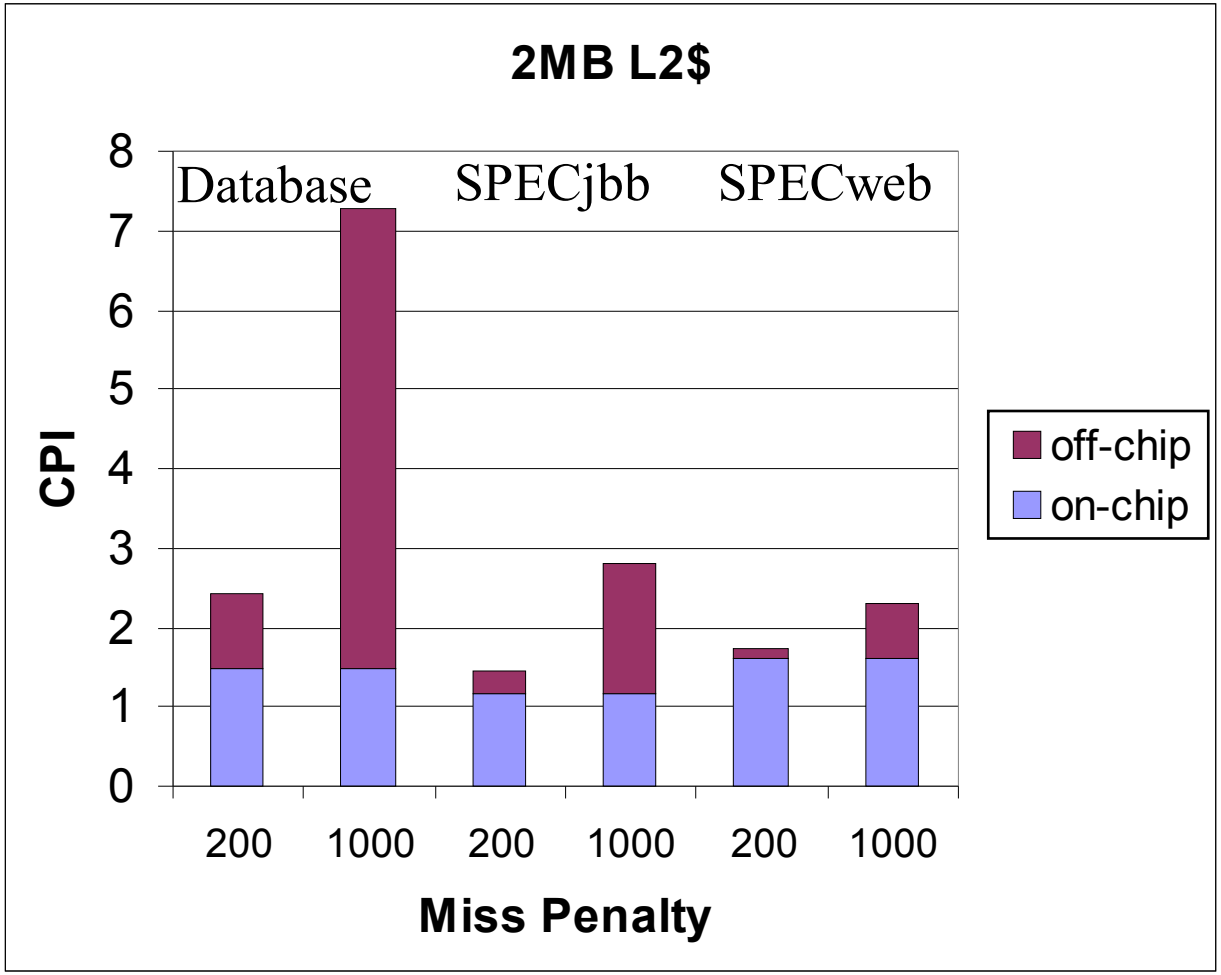
off-chip CPI = 1

on-chip CPI = 1

CPI = 2



Measured CPI Components



Off-chip miss rate (per 100 insts)

Database	0.84
SPECjbb	0.19
SPECweb	0.09

Off-Chip Access Clustering

	L2$ Miss Rate (per 100 insts)	Inter-Miss Dist (insts) Avg	<32	<64	<128
Database	0.84	119	50%	70%	82%
SPECjbb	0.19	526	50%	64%	70%
SPECweb	0.09	1111	34%	53%	69%

Off-Chip Access Clustering

	L2$ Miss Rate (per 100 insts)	Inter-Miss Dist (insts) Avg	<32	<64	<128
Database	0.84	119	50%	70%	82%
SPECjbb	0.19	526	50%	64%	70%
SPECweb	0.09	1111	34%	53%	69%

Off-chip access clustering suggests exploiting MLP feasible

MLP Limiters

Important microarchitecture limiters:

1. Issue window and reorder buffer (ROB) size
2. Serializing instructions
3. Instruction fetch off-chip accesses
4. Unresolvable mispredicted branches
5. Load and branch instruction issue restrictions

Issue Window and ROB Sizes

- Load requiring off-chip access blocks instruction retirement
- Issue window and ROB fills up, stalling processor

Example:

i1	load [r1]->r2	off-chip access
i2	add r1,r3->r4	
i3	load [r4]->r5	off-chip access
i4	add r1,r5->r6	
i5	load [r6]->r7	off-chip access

Assume:
Issue window size = 4
ROB size = 4

Serializing Instructions

- Most ISAs provide instructions for implementing synchronization primitives and for memory ordering
  e.g. CASA, LDSTUB and MEMBAR in SPARC ISA
- Straightforward implementation requires pipeline drain
- Can be fairly prevalent e.g. 0.6% CASA instructions in SPECjbb2000; used for Java object locking

Example:

```
i1      load 0(r1)->r2  off-chip access
i2      membar
i3      load 0(r3)->r4  off-chip access
```

Instruction Fetch Misses

- No subsequent off-chip accesses can be overlapped since they cannot be fetched
- Instruction fetch misses that are the first off-chip accesses are most expensive

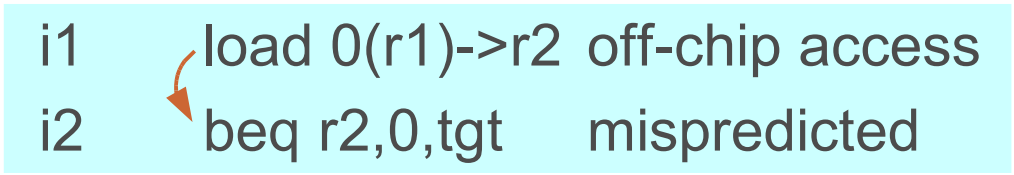
Example:

```
i1      add r1,r3->r4   off-chip instruction access
i2      load [r4]->r5   off-chip access
```

Unresolvable Mispredicted Branches

- Mispredicted branches dependent on off-chip access cannot resolve until off-chip access completes
- Unless control independent and data independent, subsequent off-chip accesses cannot be overlapped

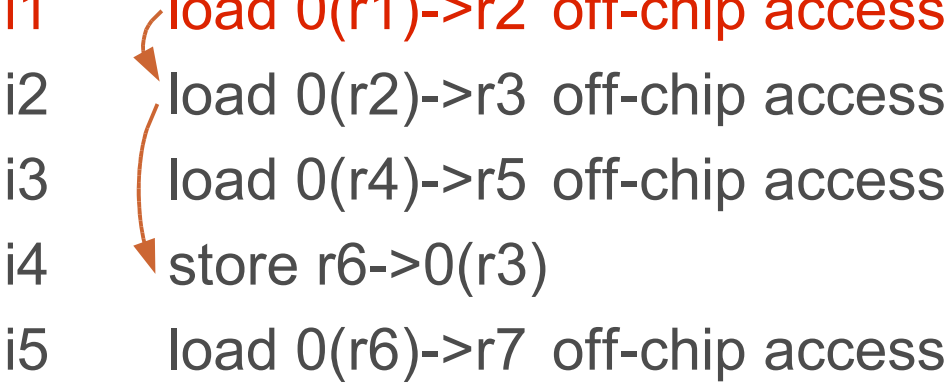
Example:



Load Issue Policy

- Issue in-order or out-of-order w.r.t. other loads
- Loads wait for earlier store addresses to resolve or speculate past earlier stores

Example:

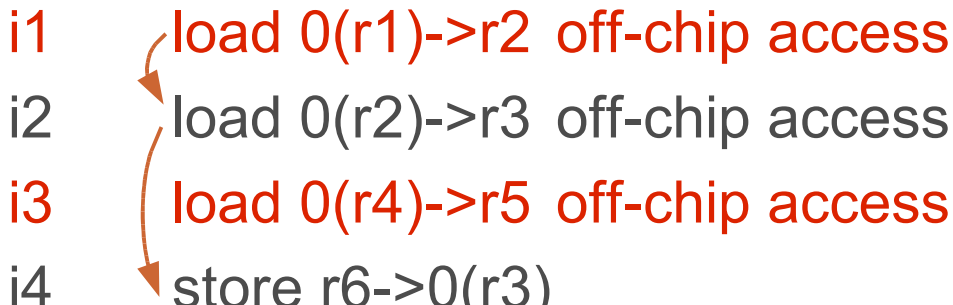


In-order load issue

Load Issue Policy

- Issue in-order or out-of-order w.r.t. other loads
- Loads wait for earlier store addresses to resolve or speculate past earlier stores

Example:

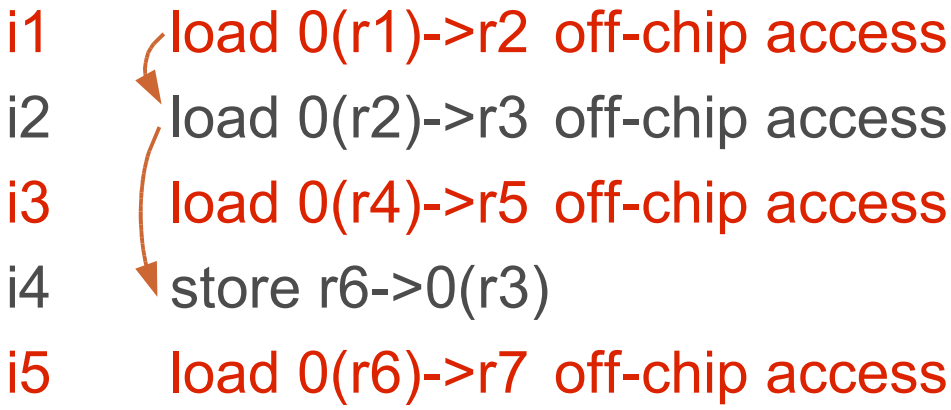


Out-of-order load issue, wait for earlier stores

Load Issue Policy

- Issue in-order or out-of-order w.r.t. other loads
- Loads wait for earlier store addresses to resolve or speculate past earlier stores

Example:

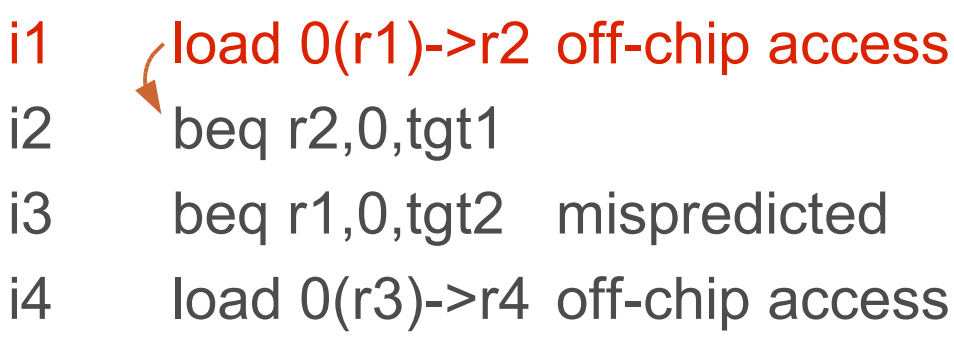


Out-of-order load issue, speculate past earlier stores

Branch Issue Policy

- Issue in-order or out-of-order w.r.t. other branches
- Makes difference when branch dependent on off-chip access prevents subsequent mispredicted branch from resolving

Example:



In-order branch issue

Branch Issue Policy

- Issue in-order or out-of-order w.r.t. other branches
- Makes difference when branch dependent on off-chip access prevents subsequent mispredicted branch from resolving

Example:

Out-of-order branch issue

Experimental Methodology

MLPsim

- Tool for measuring MLP
- Implements epoch MLP model (see paper)
- No need to model any on-chip computation
- No need to model timing of off-chip accesses
- Only need to determine which off-chip accesses can be overlapped
- Simple, small and easy to verify
- Results validated against cycle-accurate processor simulator

Benchmarks

Collectively, represent 3-tiered datacenter:

- Database workload
- SPECjbb2000 - server side Java, emphasizes business logic and object manipulation
- SPECweb99 – web server performance

Highly optimized binaries

Carefully validated traces

Simulation runs: warm 50M, collect statistics 100M

Default Processor Model

- 32KB 4-way 64B I$ and D$
- 2MB 4-way 64B L2$, no L3$
- 2K entry shared TLB

- 64K entry gshare, 16K entry BTB, 16 entry RAS
- 32 entry fetch buffer

- 64 entry issue window, 64 entry ROB
- Infinite load and store buffers
- 3-wide OOO instruction issue: OOO loads allowed to speculate past earlier stores, in-order branches

i.e. moderately aggressive out-of-order issue processor

Results

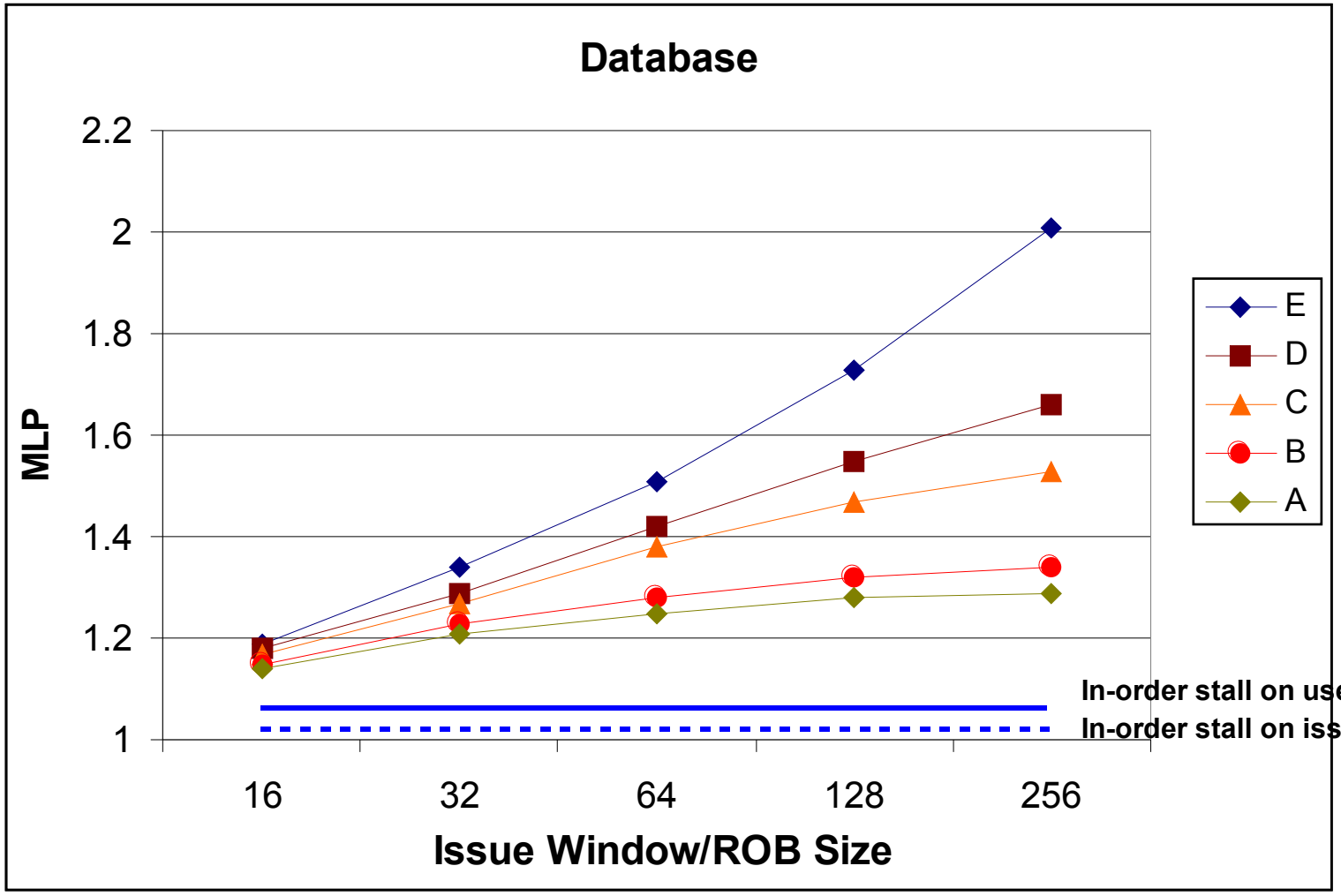
Experiments

1. Out-of-order issue policy, issue window / ROB sizes
2. Decoupled issue window / ROB
3. Cache size
4. Runahead execution
5. Limit study

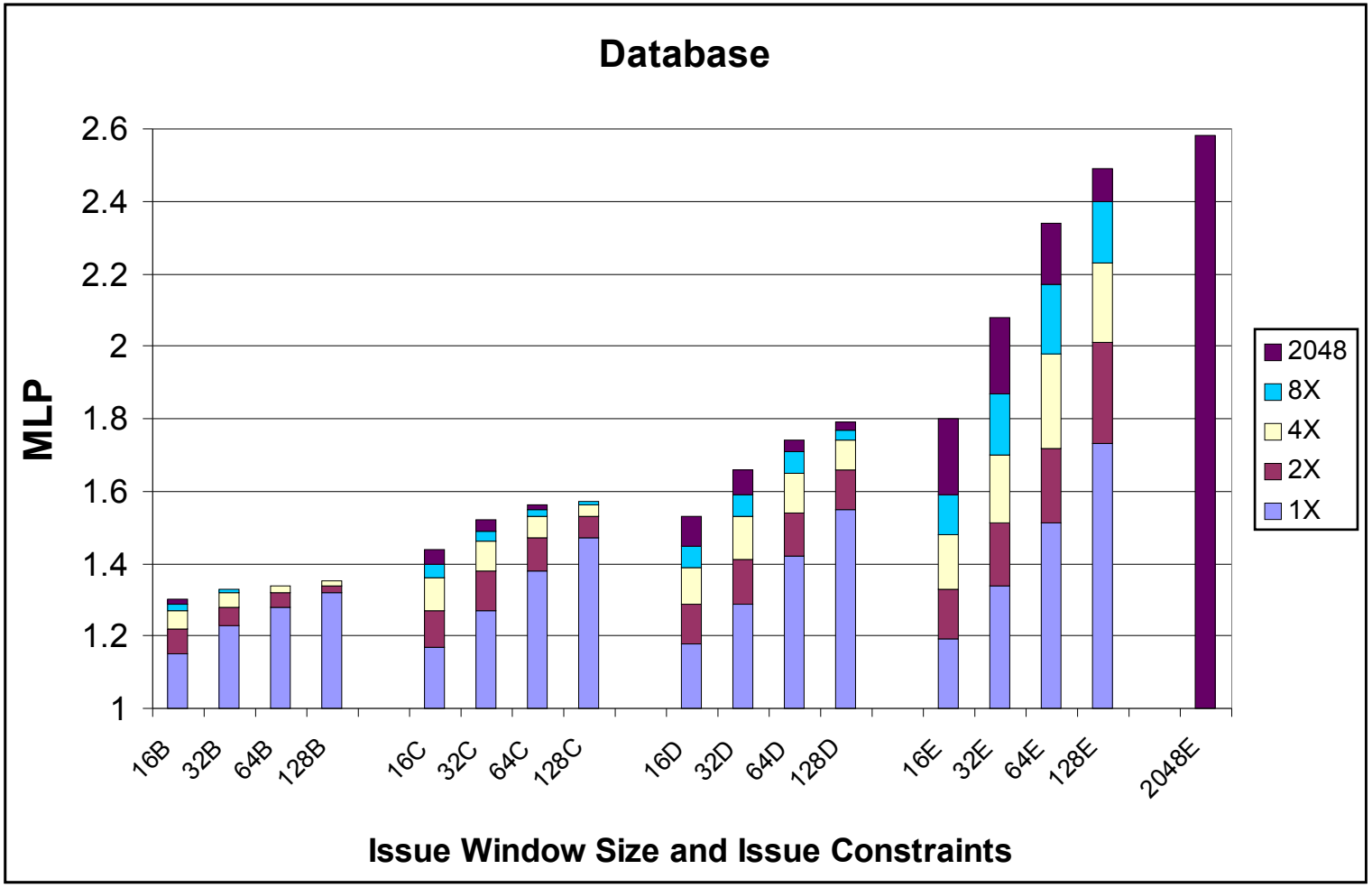
Out-of-Order Issue Policy

	Load Issue		Branch Issue	Serializing Insts
	out-of-order	store speculation	out-of-order	non-serializing
A	no	no	no	no
B	yes	no	no	no
C	yes	yes	no	no
D	yes	yes	yes	no
E	yes	yes	yes	yes

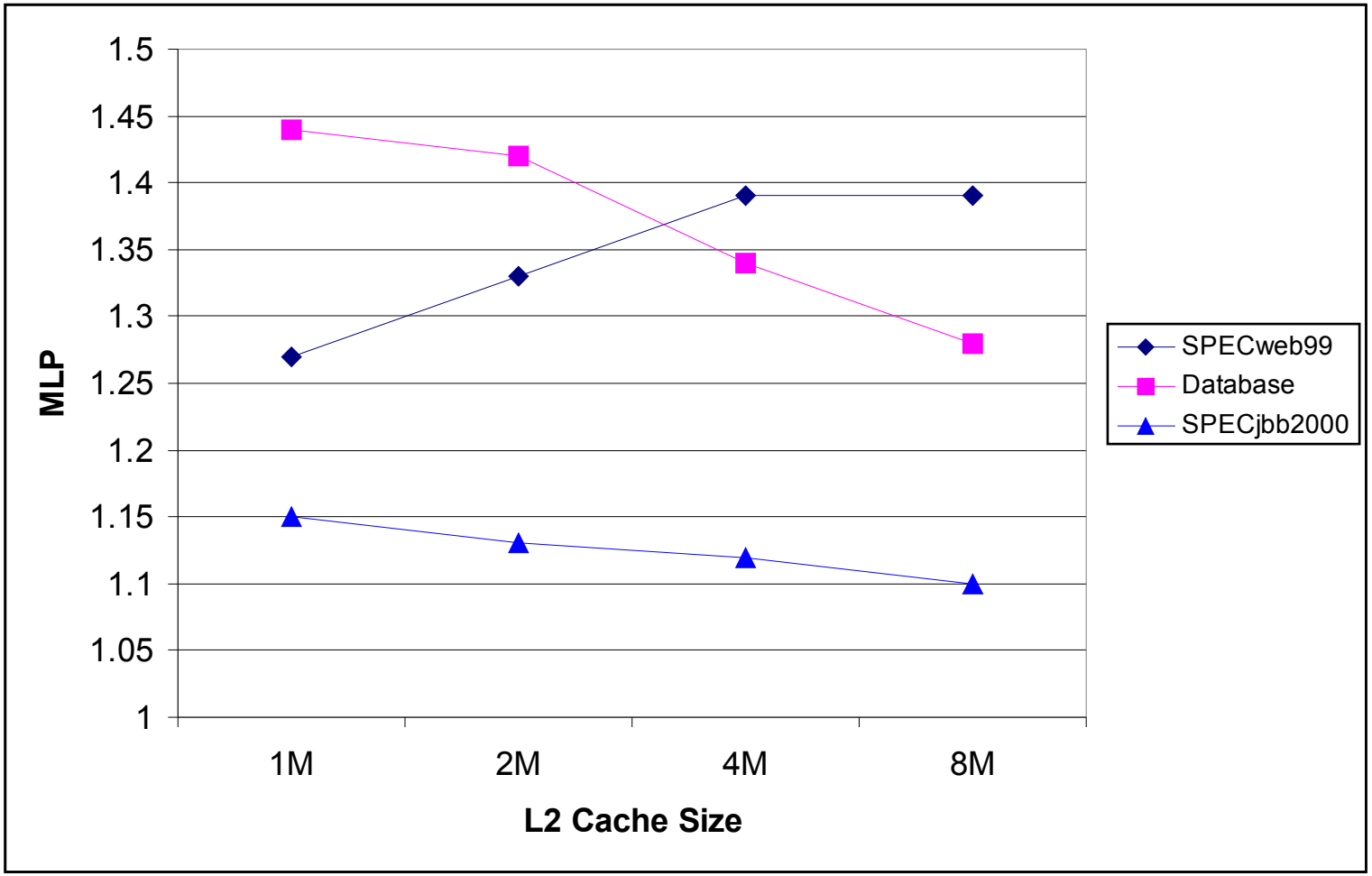
Out-of-Order Issue



Decoupled Issue Window/ROB



Cache Size Impact



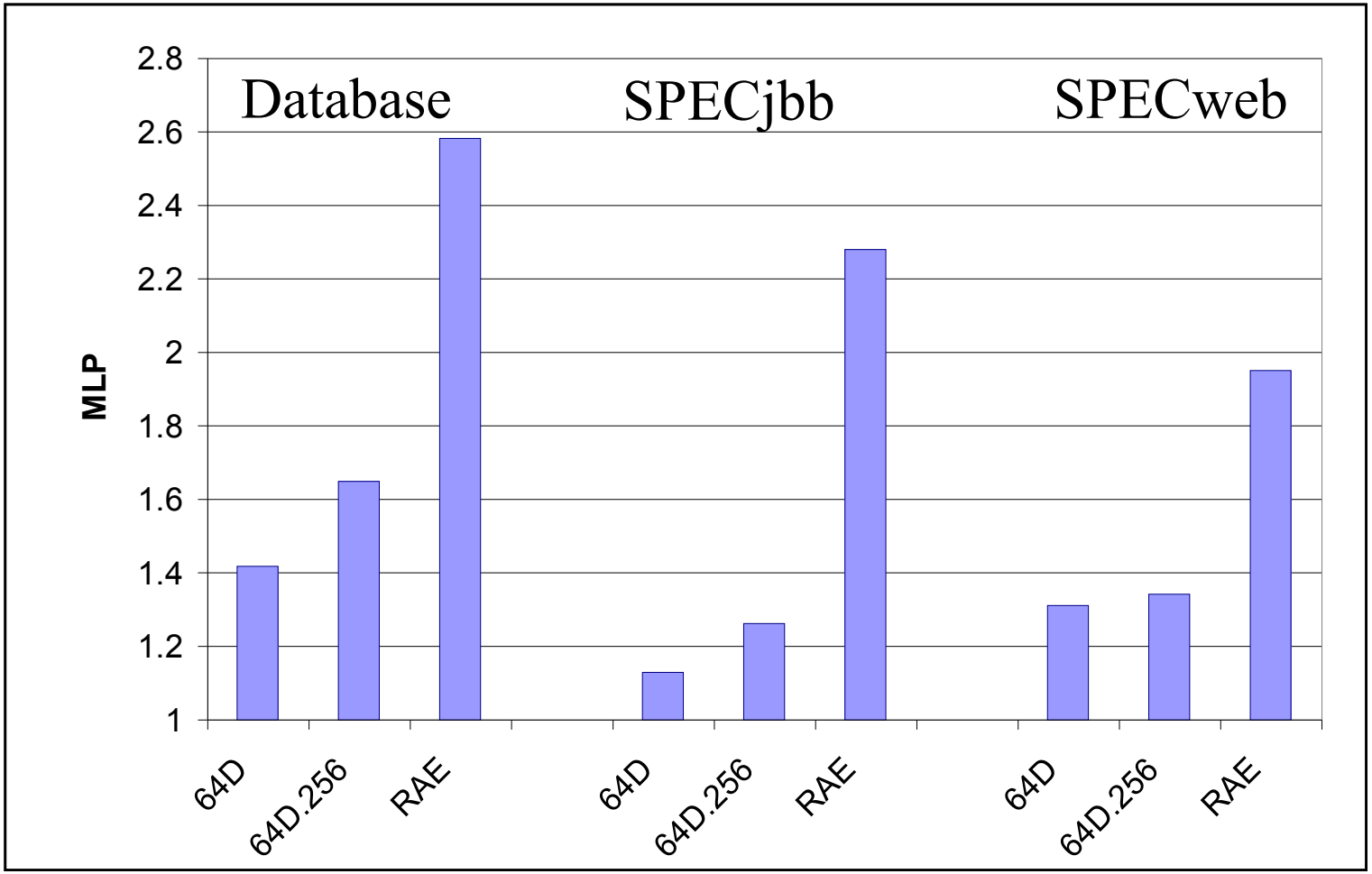
Runahead Execution (HW Scout)

- When load requiring off-chip access reaches head of ROB, transition to runahead execution (RAE) mode
- In RAE mode:
  - stores do not modify architected state
  - loads requiring off-chip accesses convert to prefetches; dependent insts dropped
  - speculate past serializing instructions
- When load data returns, terminate RAE mode

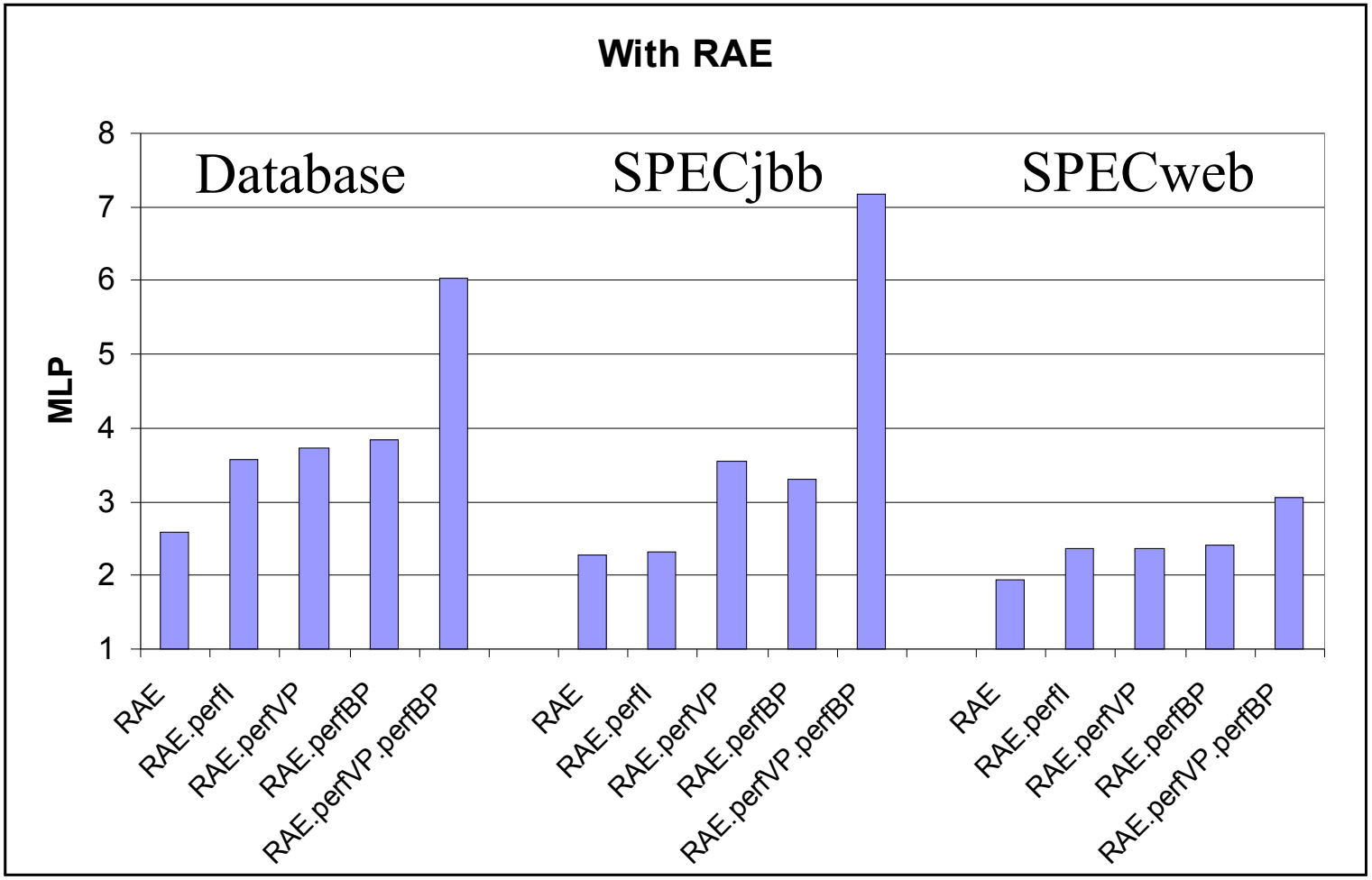
Improves MLP by removing MLP limiters:

1. ROB / issue window size and issue policy constraints
2. serializing instructions

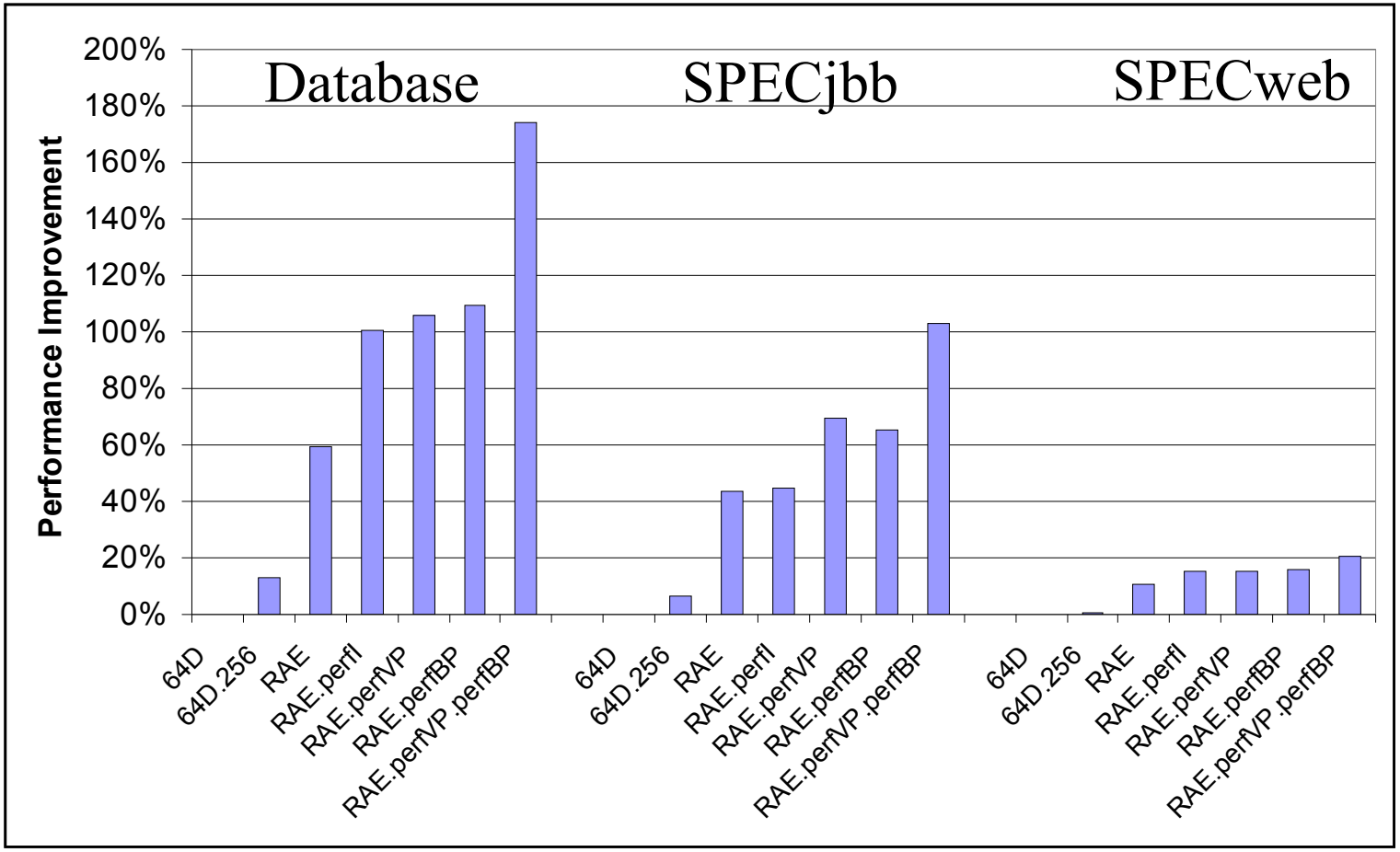
Runahead Execution



Limit Study


With RAE
MLP
8
7
6
5
4
3
2
1
Database
SPECjbb
SPECweb
RAE
RAE:perfI
RAE:perfVP
RAE:perfBP
RAE:perfVP:perfBP
RAE
RAE:perfI
RAE:perfVP
RAE:perfBP
RAE:perfVP:perfBP
RAE
RAE:perfI
RAE:perfVP
RAE:perfBP
RAE:perfVP:perfBP

Overall Performance



Results Summary

- Out-of-order execution moderately effective in exploiting MLP
- Instruction issue constraints must be relaxed to achieve benefits of large issue windows
- Serializing instructions major impediment to MLP
- Decoupling ROB from issue window improves MLP
- Runahead execution (HW scout) greatly improves MLP
- Limit study shows very significant MLP headroom
- For database and SPECjbb, MLP improvements translate into impressive performance gains

Conclusions

- Microarchitecture has profound impact on MLP
- For memory bound workloads, exploiting MLP is powerful technique for improving performance