Setting the Thoughts

- Why should we or CPU worry about performance of cache memory?
- How to model and then measure performance of cache memory?
- How does the performance of cache impacts overall CPU execution time?
- Where to look at for further enhancement of performance?
- What are the additional parameters that have emerged in the recent time?

Parameters of Cache Optimization

Avg Memory access time =
Hit time + (Miss rate X Miss penalty)

As the microprocessor progress towards multi-core system the bandwidth requirement also increases. Further, the problem of power dissipation also arises in multi-core processor. Therefore, while designing a cache memory these two additional parameters needs to be considered.

A cache memory must provide **higher bandwidth** and must dissipate as **minimum power** as possible.

Bandwidth Requirements

Bandwidth demand from microprocessor:

Microprocessor	16-bit address/ bus, microcoded	32-bit address/ bus, microcoded	5-stage pipeline, on-chip I & D caches, FPU	2-way superscalar, 64-bit bus	Out-of-order 3-way superscalar	Out-of-order superpipelined, on-chip L2 cache	Multicore OOO 4-way on chip L3 cache, Turbo
Product	Intel 80286	Intel 80386	Intel 80486	Intel Pentium	Intel Pentium Pro	Intel Pentium 4	Intel Core i7
Year	1982	1985	1989	1993	1997	2001	2010
Die size (mm^2)	47	43	81	90	308	217	240
Transistors	134,000	275,000	1,200,000	3,100,000	5,500,000	42,000,000	1,170,000,000
Processors/chip	1	1	1	1	1	1	4
Pins	68	132	168	273	387	423	1366
Latency (clocks)	6	5	5	5	10	22	14
Bus width (bits)	16	32	32	64	64	64	196
Clock rate (MHz)	12.5	16	25	66	200	1500	3333
Bandwidth (MIPS)	2	6	25	132	600	4500	50,000
Latency (ns)	320	313	200	76	50	15	4

Bandwidth Requirements

Main memory (DRAM) bandwidth capabilities:

Memory module	DRAM	Page mode DRAM	Fast page mode DRAM	Fast page mode DRAM	Synchronous DRAM	Double data rate SDRAM	DDR3 SDRAM
Module width (bits)	16	16	32	64	64	64	64
Year	1980	1983	1986	1993	1997	2000	2010
Mbits/DRAM chip	0.06	0.25	1	16	64	256	2048
Die size (mm^2)	35	45	70	130	170	204	50
Pins/DRAM chip	16	16	18	20	54	66	134
Bandwidth (MBytes/s)	13	40	160	267	640	1600	16,000
Latency (ns)	225	170	125	75	62	52	37

Power Dissipation Trends

Memory power is also increasing with
clock frequency and complexity of the design.

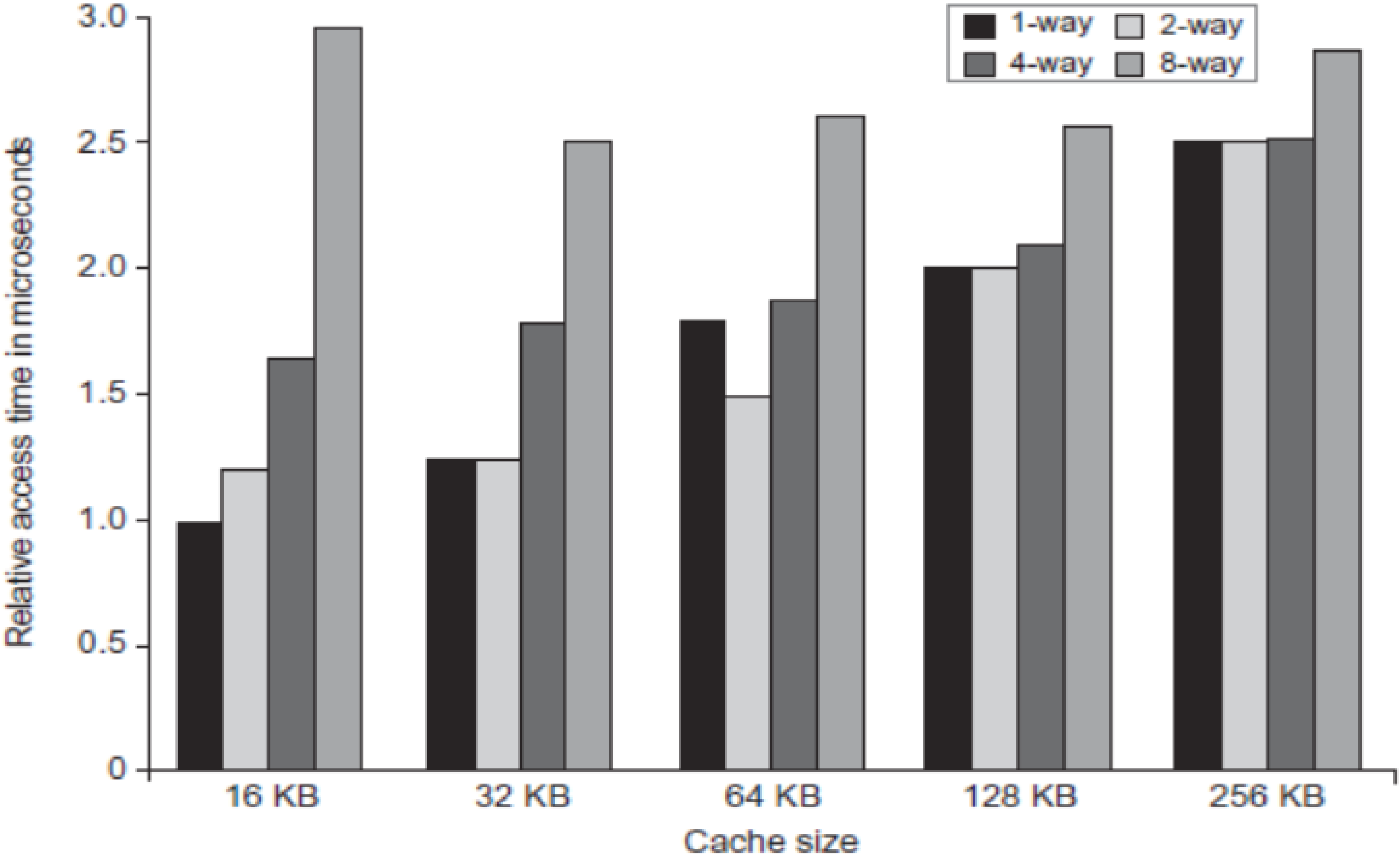
Five Parameters to Optimize

The Ideas	Impact on Parameters
Small and Simple L1 Cache	Reduce hit time Reduce power
Pipelined, banking, and non-blocking cache	Increase bandwidth. Varying impact on Power- can increase or decrease
Critical word first and merge write buffer	Reduce miss penalty. Might increase power
Compiler techniques	Reducing miss rate. Reduces power.
Prefetching: Hardware and Compiler based	Reducing miss penalty. Increase power (if prefetched blocks are unused.)

Ten Ideas

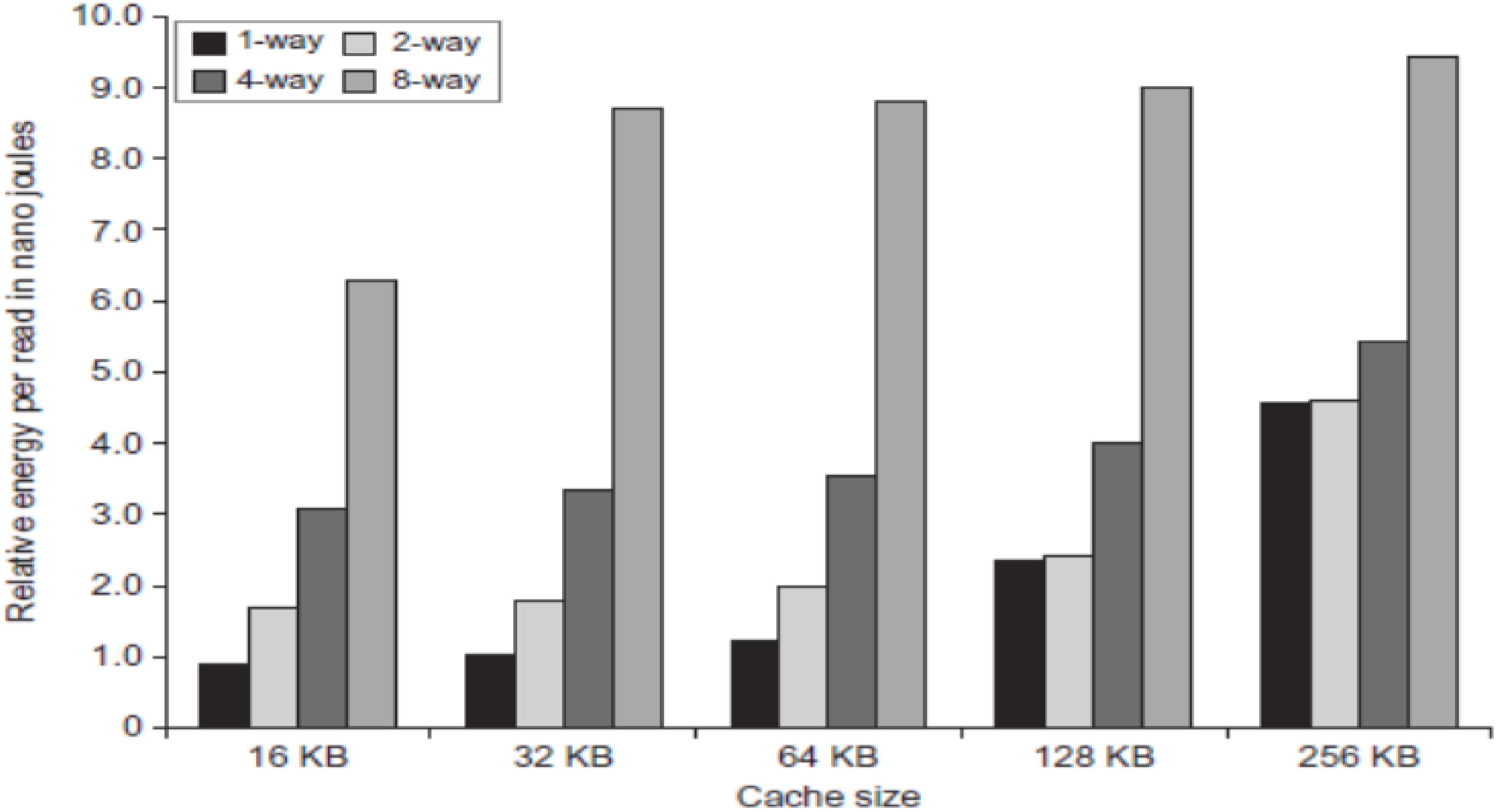
1. Small and simple first level cache to reduce hit time and power
2. Way prediction to reduce hit time
3. Pipelined access and multibanked caches to increase bandwidth
4. Non-blocking cache to increase band-width
5. Critical word first and early restart to reduce miss penalty
6. Merging write buffer to reduce miss penalty
7. Compiler optimization to reduce miss rate
8. Hardware prefetching of instructions and data to reduce miss penalty and miss rate
9. Compiler controlled prefetching to reduce miss penalty and miss rate
10. Using high bandwidth memory (HBM) to increase bandwidth (this will not be taught)

Small/Simple L1 Cache



Access time Vs Size and Associativity

Small/Simple L1 Cache



Energy Vs Size and Associativity

Way Prediction

- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - "Way selection"
  - Increases mis-prediction penalty

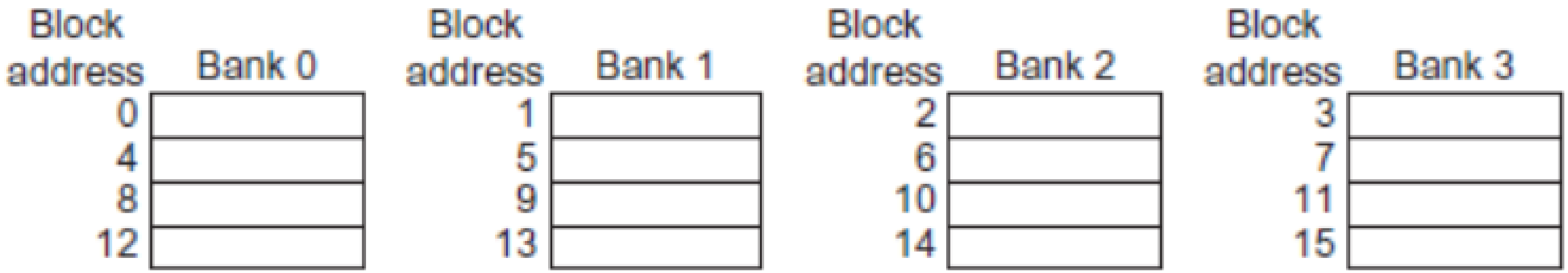
Pipelined Cache

- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

Multibanked Cache

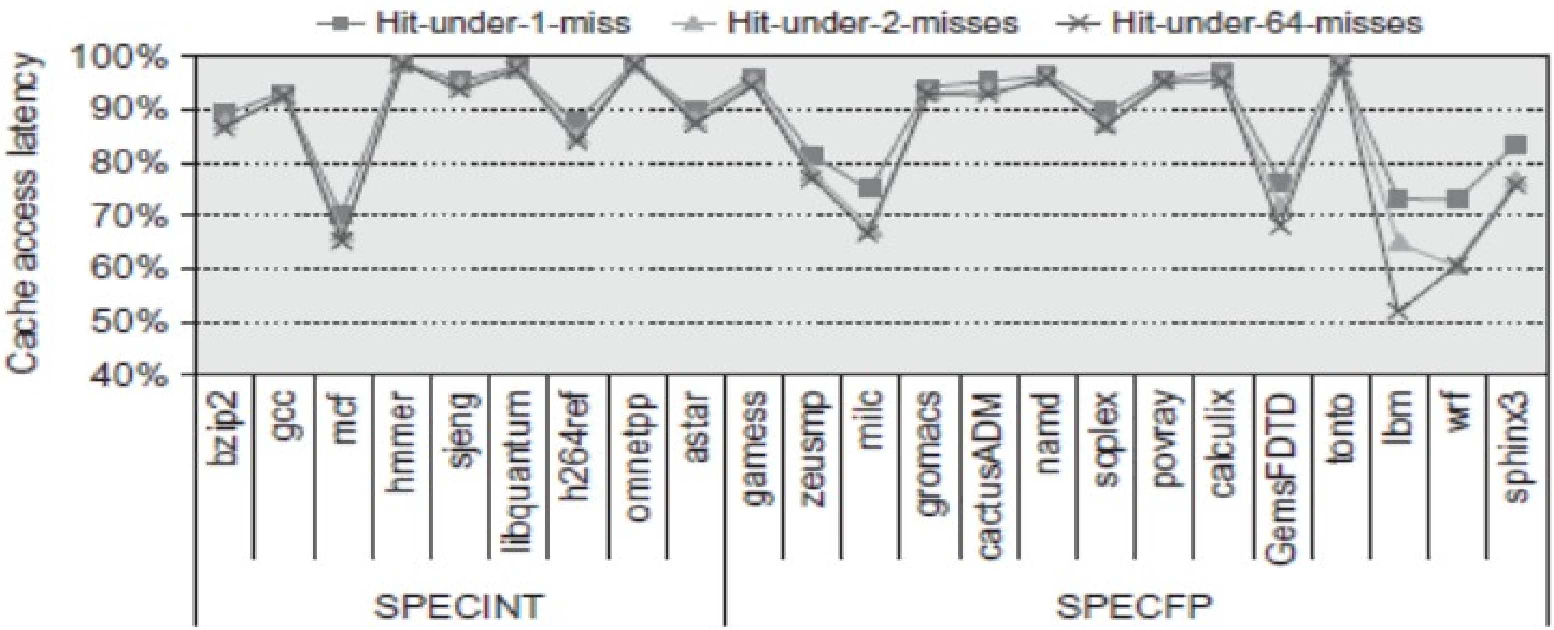
- Organize cache as independent banks to support simultaneous access to increase bandwidth
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2

- Interleave banks according to block address



Non-Blocking Caches

- Allow hits before previous misses complete
  - “Hit under miss”
  - “Hit under multiple miss”
- L2 must support this (DRAM access take too long)
- In general, processors can hide L1 miss penalty but not L2 miss penalty



Critical Word First and Early Restart

- Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
- Early restart
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives

- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses

Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

No write buffering

Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

Write buffering

Compiler Optimizations

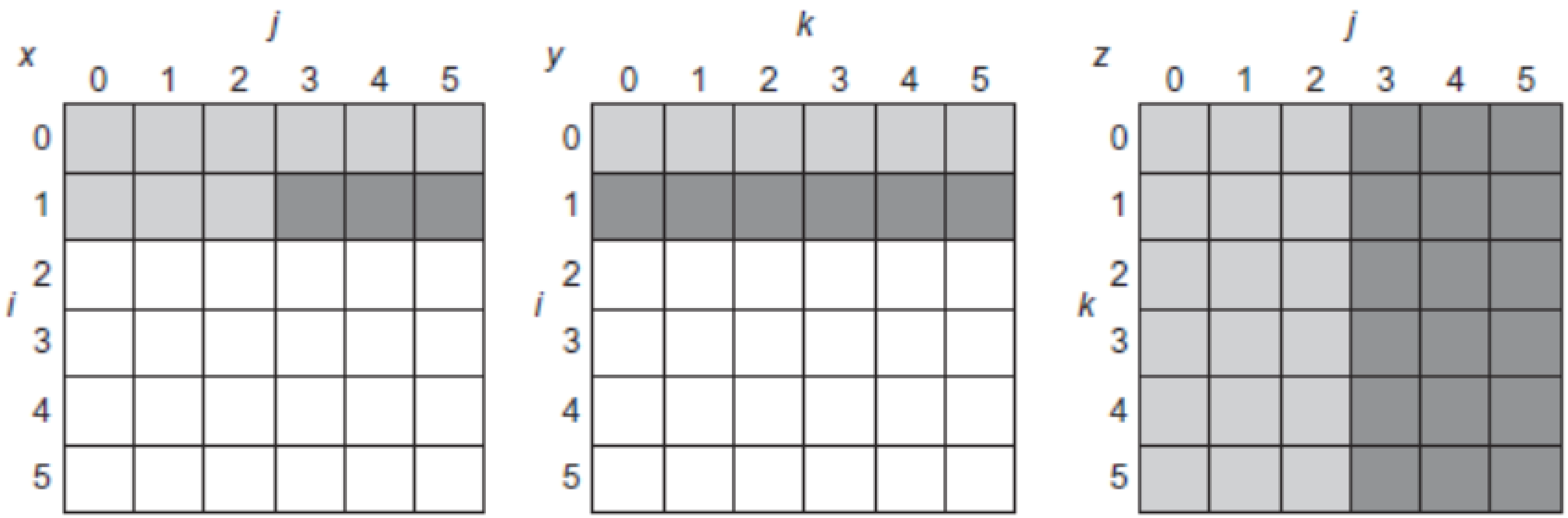
- Loop Interchange
  - Swap nested loops to access memory in sequential order
- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

Blocking

```
for (i = 0; i < N; i = i + 1)
  for (j = 0; j < N; j = j + 1)
  {
    r = 0;
    for (k = 0; k < N; k = k + 1)
      r = r + y[i][k]*z[k][j];
    x[i][j] = r;
};
```

Example from
Text Book

The problem!

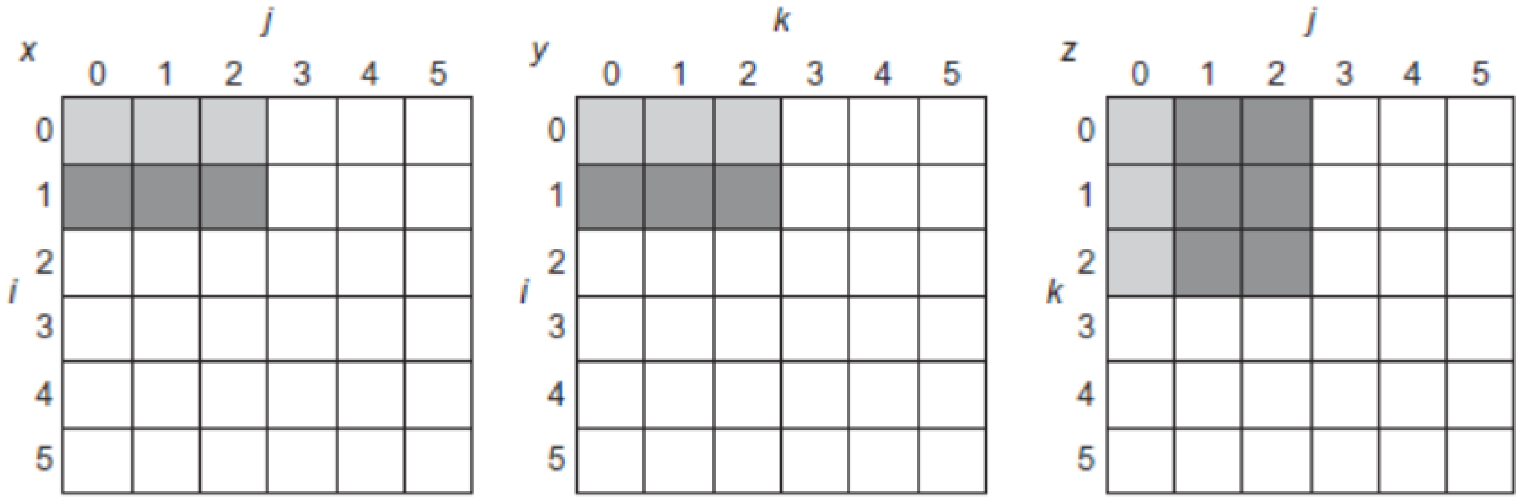


Blocking

```
for (jj = 0; jj < N; jj = jj + B)
  for (kk = 0; kk < N; kk = kk + B)
    for (i = 0; i < N; i = i + 1)
      for (j = jj; j < min(jj + B,N); j = j + 1)
      {
        r = 0;
        for (k = kk; k < min(kk + B,N); k = k + 1)
          r = r + y[i][k]*z[k][j];
        x[i][j] = x[i][j] + r;
};
```

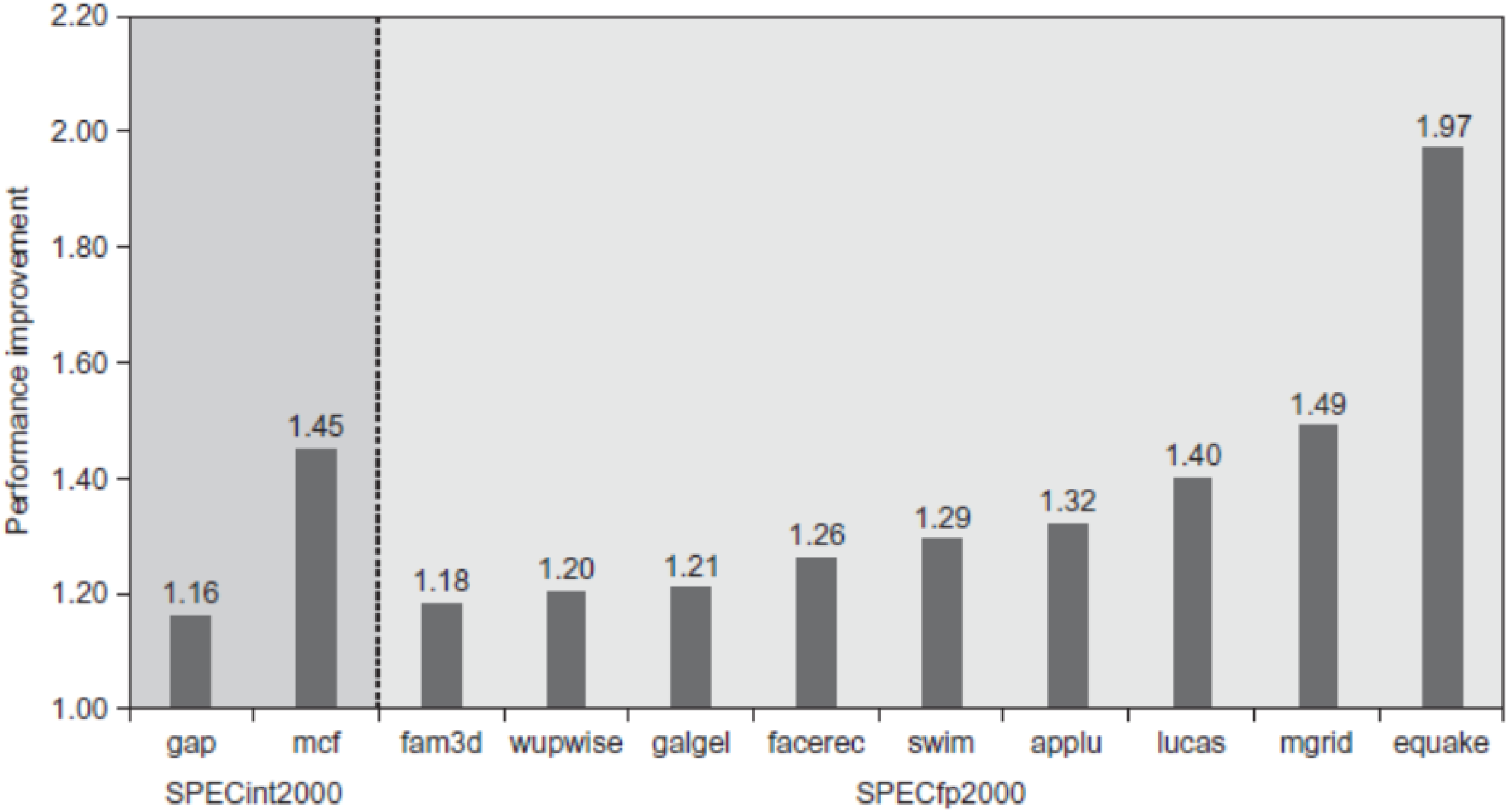
Example from
Text Book

Solution!



Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)



Pentium 4 study

Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn’t cause exceptions

- Register prefetch
  - Loads data into register
- Cache prefetch
  - Loads data into cache

- Combine with loop unrolling and software pipelining

(Loop unrolling and software pipeline to be covered during VLIW)

Thanks

Reference:

Chapter 2 of the Text Book.

The content in this presentation are from the text book and Companion presentation.