HY425 Lecture 13: Improving Cache Performance

Dimitrios S. Nikolopoulos

University of Crete and FORTH-ICS

November 25, 2011

Multilevel caches

Motivation

- Bigger caches bridge gap between CPU and DRAM
- Smaller caches keep pace with CPU speed
- Mutli-level caches a compromise between the two
  - L2 cache captures misses from L1 cache
  - L2 cache provides additional on-chip caching space

Performance analysis

$$\begin{aligned} AMAT &= \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times \text{Miss penalty}_{L2} \\ \text{miss penalty}_{L1} &= \text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2} \\ AMAT &= \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2}) \end{aligned}$$

Multilevel cache miss rates

Local vs. global miss rate

- Local miss rate: Number of misses in the cache divided by the number of access to this cache
- Global miss rate: Number of misses in the cache divide by the number of accesses issued from the GPU

Performance analysis

$$\text{Average memory stalls per instruction} = \text{Miss per instruction}_{L1} \times \text{Hit time}_{L2} + \text{Misses per instruction}_{L2} \times \text{Miss penalty}_{L2}$$

AMAT in multilevel caches

Example

- Write-back first-level cache
- 40 misses per 1000 memory references L1
- 20 misses per 1000 memory references L2
- L2 cache miss penalty = 100 cycles
- L1 hit time = 1 cycle
- L2 cache hit time = 10 cycles
- 1.5 memory references per instructions

$$\text{Miss rate}_{L1} = \frac{40}{1000} = 0.04$$

$$\text{Miss rate}_{L2,local} = \frac{\text{misses in L2}}{\text{misses in L1}} = \frac{20}{40} = 0.50$$

$$AMAT = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

$$AMAT = 1.0 + 0.04 \times (10 + 0.50 \times 100) = 3.4\text{cycles}$$

AMAT in multilevel caches

Example (cont.)

- Write-back first-level cache
- 40 misses per 1000 memory references L1
- 20 misses per 1000 memory references L2
- L2 cache miss penalty = 100 cycles
- L1 hit time = 1 cycle
- L2 cache hit time = 10 cycles
- 1.5 memory references per instruction

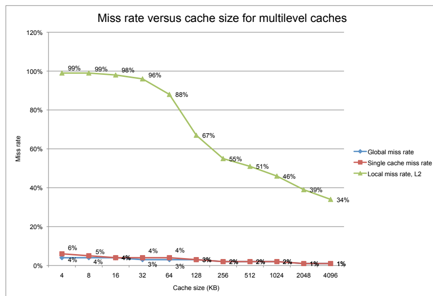
$$\text{Average memory stalls per instruction} = \text{Misses per instruction}_{L1} \times \text{Hit time}_{L2} + \text{Misses per instruction}_{L2} \times \text{Miss penalty}_{L2}$$

$$40 \times \frac{1.5}{1000} \times 10 + 20 \times \frac{1.5}{1000} \times 100 = 3.6 \text{ clock cycles}$$

L2 cache performance implications

L2 cache miss rate versus size

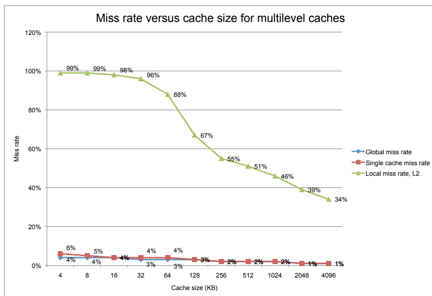
- Two-level cache uses 32 KB L1-I, 32 KB L1-D cache
- 64B block size, 2-way associative caches



L2 cache performance implications

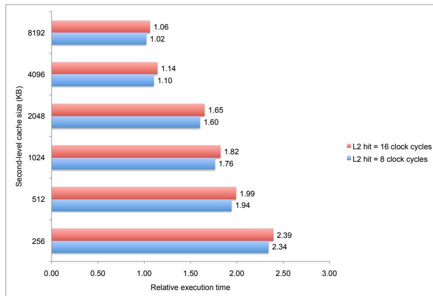
L2 cache miss rate versus size

- Global miss rate follows one-level cache miss rate
- Local L2 cache miss rate is poor performance indicator



L2 cache performance implications

Impact of L2 cache size on execution time



Inclusion property

L1 cache contents always in L2 cache?

- Mutil-level inclusion guarantees that L1 data is always present in L2, L2 in L3, ...
  - Simplifies cache consistency check in multiprocessors, or between I/O devices and processor
  - Complicates the use of different block sizes for L1 and L2 – L1 refill may require storing more than one blocks
- Mutil-level exclusion guarantees that L1 data is never present in L2, L2 data never present in in L3, ...
  - Reasonable choice for systems with small L2 cache relative to the L1 cache
  - Effective expansion of total caching space with a slower cache

Critical word first and early restart

Critical word first

- Cache block size tends to increase to exploit spatial locality
- Any given reference needs only one word from a multi-word block
- CWF fetches requested word first and sends it to processor
- Processor continues execution while rest of the block is fetched

Early restart

- Fetches words in the order stored in the block
- As soon as critical word arrives, sends to processor and processor restarts

Giving priority to read misses over writes

Write-through caches

- Write buffer holds written data to mask memory latency
- Write buffer may hold values needed by a later read miss

```
SW R3, 512(R0)      ;M[512] = R3   (cache index 0)
LW R1, 1024(R0)     ;R1 = M[1024]  (cache index 0)
LW R2, 512(R0)      ;R2 = M[512]   (cache index 0)
```

- Store to 512[R0] with block from cache index 0 waits in write buffer
- Load to 1024[R0] misses and brings new block in cache index 0
- Second load attempts to bring block from 512[R0] (held in write buffer)
- Memory RAW hazard

Giving priority to read misses over writes

Write-through caches

- Check contents of write buffer on read miss
- If no conflict then let missing read bypass pending write

Write-back caches

- Slow path: write dirty block to memory, then fetch new block from memory to cache
- Faster path:write dirty block to write buffer, then fetch new block from memory to cache, then write back dirty block

Merging write buffer

Write buffer organization

- Processor blocks on write if write buffer full
- Processor checks write address with address in write buffer
- Processor merges writes to same address if address is present in write buffer
- Assume write buffer with 4 entries, with 4 64-bit words each
- Writes to same cache block in different cycles, no write merging

Write buffer, no write merging

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	

Merging write buffer

Write buffer organization

- Processor blocks on write if write buffer full
- Processor checks write address with address in write buffer
- Processor merges writes to same address if address is present in write buffer
- Assume write buffer with 4 entries, with 4 64-bit words each
- Writes to same cache block in different cycles, write merging

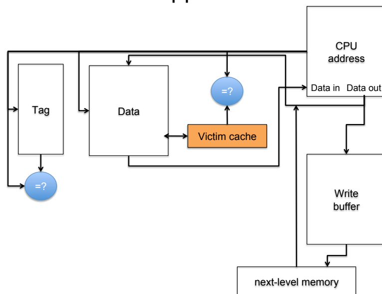
Write buffer,write merging

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

Victim cache

Tiny cache holds evicted cache blocks

- Small (e.g. 4-entry) fully associative buffer for evicted blocks
- Proposed to reduce impact of conflicts on direct-mapped caches
    - Victim cache + Direct-mapped cache $\approx$ associative cache

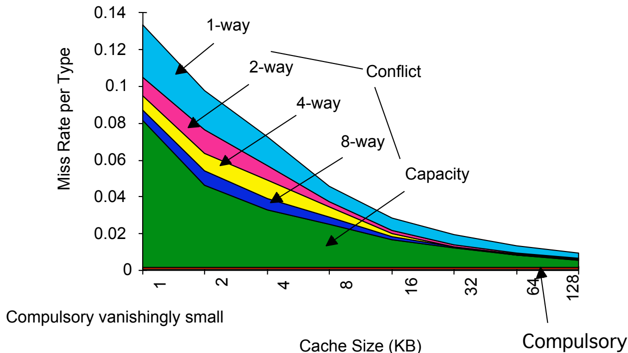


3 C's model

Characterization of cache misses

- Compulsory miss: Miss that happens due to the first access to a block since program began execution. Also called cold-start miss.
- Capacity miss: Miss that happens because a block that has been fetched in the cache needed to be replaced due to limited capacity (all blocks valid in the cache, cache needed to select victim block). Block had been fetched, replaced, and re-fetched to count as capacity miss.
- Conflict miss: MIss that happens because address of block maps to same location in the cache with other block(s) in memory. Block had been fetched, replaced, re-fetched, and cache has invalid locations that could hold the block if a different address mapping scheme were used, to count as conflict miss (as opposed to compulsory miss with first-time fetch).

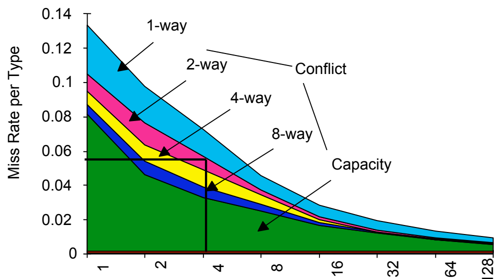
Associativity and conflict misses



Associativity and conflict misses

2 to 1 cache rule

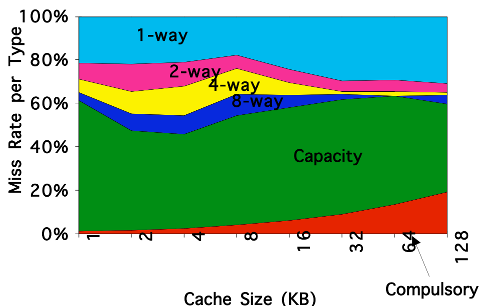
miss rate 1-way associative cache of size X
= miss rate 2-way associative cache of size X/2



Associativity and conflict misses

Miss rate distribution

- Associativity tends to increase in modern caches

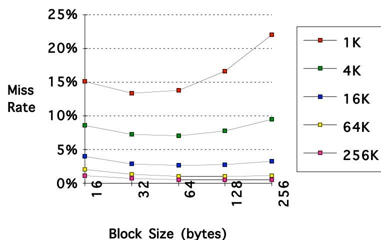


Increasing block size

Spatial locality

- Larger block size usually reduces compulsory misses
- Larger block size increases miss penalty, since processor needs to fetch more data
- Increasing block size may increase conflict misses, if spatial locality is poor (most words in fetched block not used)
- Increasing block size may increase capacity misses, if spatial locality is poor (most words in fetched block not used)

Block size impact



Miss rate versus block size

SPEC92 benchmarks

	Cache size			
Block size	**4K**	**16K**	**64K**	**256K**
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

AMAT versus block size

SPEC92 benchmarks

- Example assumes 80-cycle memory latency, 16 bytes per 2 cycles pipelined memory throughput

		Cache size			
Block size	Miss penalty	4K	16K	64K	256K
16	82	8.027	4.231	2.673	1.894
32	84	7.082	3.411	2.134	1.588
64	88	7.160	3.323	1.933	1.449
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

Larger caches

Implications of higher cache capacity

- Reduction of capacity misses
- Longer hit time
- Increased area, power, and cost
  - Very large multi-MB caches often placed off-chip

Increasing associativity

Example

- Higher associativity increases hit time
- Increased hit time for L1 cache means increased cycle time
- Assume

$$\begin{aligned}
\text{Hit time} &= 1.0 \text{ cycle} \\
\text{Miss penalty}_{direct-mapped} &= 25 \text{ cycles to perfect L2 cache} \\
\text{Clock cycle time}_{2-way} &= 1.36 \times \text{Clock cycle time}_{1-way} \\
\text{Clock cycle time}_{4-way} &= 1.44 \times \text{Clock cycle time}_{1-way} \\
\text{Clock cycle time}_{8-way} &= 1.52 \times \text{Clock cycle time}_{1-way} \\
AMAT_{8-way} &< AMAT_{4-way}? \\
AMAT_{4-way} &< AMAT_{2-way}? \\
AMAT_{2-way} &< AMAT_{1-way}?
\end{aligned}$$

Increasing associativity

Example

- Assume

$$\begin{aligned}
\text{Hit time} &= 1.0 \text{ cycle} \\
\text{Miss penalty}_{direct-mapped} &= 25 \text{ cycles to perfect L2 cache} \\
\text{Clock cycle time}_{2-way} &= 1.36 \times \text{Clock cycle time}_{1-way} \\
\text{Clock cycle time}_{4-way} &= 1.44 \times \text{Clock cycle time}_{1-way} \\
\text{Clock cycle time}_{8-way} &= 1.52 \times \text{Clock cycle time}_{1-way} \\
AMAT_{8-way} &= 1.52 + \text{Miss rate}_{8-way} \times 25.0 \\
AMAT_{4-way} &= 1.44 + \text{Miss rate}_{4-way} \times 25.0 \\
AMAT_{2-way} &= 1.36 + \text{Miss rate}_{2-way} \times 25.0 \\
AMAT_{1-way} &= 1.00 + \text{Miss rate}_{1-way} \times 25.0
\end{aligned}$$

AMAT versus associativity

SPEC92 benchmarks

	Associativity			
Cache size (KB)	One-way	Two-way	Four-way	Eight-way
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.23	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.14	2.18	2.25
128	1.32	1.66	1.75	1.82
256	1.20	1.55	1.59	1.66

Way prediction

Way prediction

- Predict way (cache bank) of next cache access
  - Block predictor bit per way
  - 85% accuracy on Alpha 21264
- Multiplexor set early to select predicted block
- If tag match succeeds (hit) predicted block returned in one cycle
- If tag match fails (miss) rest of the blocks checked in second cycle
  - Two hit times: fast hit (way predicted), slow hit (way mispredicted)
- Effective technique for power reduction
  - Supply power only to tag arrays of selected ways

Pseudoassociativity

- Cache organized as direct-mapped cache with pseudosets
  - Pseudosets contain blocks in different lines of the tag/data arrays
- Hit processed as in direct-mapped cache
- On miss, check other block in pseudo-set
- Two hit times: fast hit (hit, direct mapped), slow hit (hit pseudoset)
- Increases hit time of direct-mapped cache, especially if slow hits are many
- May increase miss penalty, overhead to select pseudo-way

Compiler optimizations

Cache-aware optimizations in software

- Code transformations to improve:
  - Spatial locality, through higher utilization of fetched cache blocks
  - Temporal locality, through reduction of the reuse distance of cache blocks
  - Examples: loop interchange, loop blocking, loop fusion, loop fision
- Data layout and data structure transformations to improve:
  - Spatial locality, through higher utilization of fetched cache blocks
  - Examples: array merging, structure/object class member reordering in memory, block array layouts

Array merging

Data structure reorganization for spatial locality

```
/* Before */
int val[SIZE];
int key[SIZE];
```

- Assume code accessing val[i], key[i], for every i
- Accesses to val and key may conflict in direct-mapped caches
- Solution, merge arrays, accesses to val[i], key[i] do not conflict in the cache, spatial locality exploited

```
/* After */
struct merge {
  int val;
  int key;
}
struct merge merged_array[SIZE];
```

Loop interchange

Code transformation for spatial locality

```
/* Before */
for (j = 0; j < 100; j++)
  for (i = 0; i < 100; i++)
    x[i][j] = 2 * x[i][j];
```

- Arrays in C stored in row-major order
- Innermost loop over column of x
- Long strides lead to poor spatial locality

```
/* After */
for (i = 0; i < 5000; i = i+1)
  for (j = 0; j < 100; j = j+1)
    x[i][j] = 2 * x[i][j];
```

Data blocking

Code transformations for temporal locality

- Reduce reuse distance for same data
- Organize code so that data is accessed in blocks
- Best performance if block accessed many times and few accesses to data outside block

Example: Matrix multiplication

```
/* Before */
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;
};
```

Data blocking

Array accesses without blocking

- Snapshot with i=1
- Assume cache line holds one array element
- Two innermost loops access N^2 elements of z, N elements of y, N elements of x
- $N \times (N^2 + 2N) = 2N^2 + N^3$ capacity misses
- Need cache space at least $N^2 + N$ to exploit temporal locality

	X	0	1	2 (j)	3	4	5
	0						
	1						
i	2						
	3						
	4						
	5						

	Y	0	1	2 (k)	3	4	5
	0						
	1						
i	2						
	3						
	4						
	5						

	Z	0	1	2 (j)	3	4	5
	0						
	1						
i	2						
	3						
	4						
	5						

Data blocking

Example: Blocked matrix multiplication

```
/* After */
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;
        };
```

- Load a block of z of size $B \times B$
- Compute partial sum for B elements of x
- Load next block
- $\frac{N}{B} \times \frac{N}{B} \times (N \times 2B + B^2) = \frac{2N^3}{B} + N^2$capacity misses

Blocked matrix multiplication

				j			
	X	0	1	2	3	4	5
	0						
	1						
i	2						
	3						
	4						
	5						

				k			
	Y	0	1	2	3	4	5
	0						
	1						
i	2						
	3						
	4						
	5						

				j			
	Z	0	1	2	3	4	5
	0						
	1						
i	2						
	3						
	4						
	5						

Cache performance improvement

Reducing cache miss penalty

- Multi-level caches
- Critical-word first and early restart
- Priority to read misses over writes
- Merging write buffer
- Victim cache

Cache performance improvement

Reducing cache miss rate

- Higher associativity
- Larger block size
- Way prediction
- Pseudo-associativity
- Compiler transformations