Cache Optimizations (II)

Compiler optimization and multilevel cache
Improving Cache Performance

1. Reducing miss rates
   - Larger block size
   - larger cache size
   - higher associativity
   - victim caches
   - way prediction and Pseudoassociativity
   - compiler optimization

2. Reducing miss penalty
   - Multilevel caches
   - critical word first
   - read miss first
   - merging write buffers

3. Reducing miss penalty or miss rates via parallelism
   - Reduce miss penalty or miss rate by parallelism
   - Non-blocking caches
   - Hardware prefetching
   - Compiler prefetching

4. Reducing cache hit time
   - Small and simple caches
   - Avoiding address translation
   - Pipelined cache access
   - Trace caches
Reducing Misses by Compiler Optimizing Memory Layout or Access Pattern

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software

Instructions
- Reorder procedures in memory so as to reduce conflict misses
- Profiling to look at conflicts

Data
- **Merging Arrays**: improve spatial locality by single array of compound elements vs. 2 arrays
- **Loop Interchange**: change nesting of loops to access data in order stored in memory
- **Loop Fusion**: Combine 2 independent loops that have same looping and some variables overlap
- **Blocking**: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows
Merging Arrays Example

/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of structures */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];

- Reducing potential conflicts between val & key;
- Improve spatial locality
Loop Interchange Example

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

Sequential accesses instead of striding through memory every 100 words; improved spatial locality
Loop Fusion Example

/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        { a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j]; }

2 misses per access to a & c vs. one miss per access; improve temporal locality
Blocking Example: Dense 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;
        };

◆ Two Inner Loops:
  ■ Read all NxN elements of z[]
  ■ Read N elements of 1 row of y[] repeatedly
  ■ Write N elements of 1 row of x[]

◆ Capacity Misses a function of N & Cache Size:
  ■ 2N³ + N² ⇒ (assuming no conflict; otherwise …)

◆ Idea: compute on BxB submatrix that fits
Blocking Example

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

◆ B called Blocking Factor
◆ Capacity Misses from $2N^3 + N^2$ to $N^3/B + 2N^2$
◆ But may suffer from conflict misses
Reducing Conflict Misses by Blocking

Conflict misses in caches not FA vs. Blocking size
  • Choose the best blocking factor
Summary of Compiler Optimizations to Reduce Cache Misses (by hand)

Performance Improvement

- **merged arrays**
- **loop interchange**
- **loop fusion**
- **blocking**
Summary: Miss Rate Reduction

\[ \text{CPUtime} = \text{IC} \times \left( \frac{\text{CPI}_{\text{Execution}}}{\text{Instruction}} + \frac{\text{Memory accesses}}{\text{Instruction}} \right) \times \text{Miss rate} \times \text{Miss penalty} \times \text{Clock cycle time} \]

3 Cs: Compulsory, Capacity, Conflict
1. Reduce misses via larger cache
2. Reduce misses via larger block Size
3. Reduce misses via higher associativity
4. Reducing misses via pseudoassociativity
5. Reducing misses by compiler optimizations
Improving Cache Performance

1. Reducing miss rates
   - Larger block size
   - Larger cache size
   - Higher associativity
   - Way prediction and pseudoassociativity
   - Compiler optimization

2. Reducing miss penalty
   - Multilevel caches
   - Critical word first
   - Read miss first
   - Merging write buffers
   - Victim caches

3. Reducing miss penalty or miss rates via parallelism
   - Non-blocking caches
   - Hardware prefetching
   - Compiler prefetching

4. Reducing cache hit time
   - Small and simple caches
   - Avoiding address translation
   - Pipelined cache access
   - Trace caches
Multi-level Cache

❖ Add a second-level cache

❖ L2 Equations

\[ \text{AMAT} = \text{Hit Time}_L + \text{Miss Rate}_L \times \text{Miss Penalty}_L \]

\[ \text{Miss Penalty}_L = \text{Hit Time}_L + \text{Miss Rate}_L \times \text{Miss Penalty}_L \]

\[ \text{AMAT} = \text{Hit Time}_L + \text{Miss Rate}_L \times (\text{Hit Time}_L + \text{Miss Rate}_L \times \text{Miss Penalty}_L) \]

❖ Definitions:

- Local miss rate—misses in this cache divided by the total number of memory accesses to this cache (Miss rate$_L$)
- Global miss rate—misses in this cache divided by the total number of memory accesses generated by the CPU (Miss Rate$_L$ x Miss Rate$_L$)
- Global miss rate is what matters to overall performance
- Local miss rate is a factor in evaluating the effectiveness of L2 cache
Local vs. Global Miss Rates

Example:

- For 1000 inst., 40 misses in L1, 20 misses in L2
- L1 hit 1 cycle, L2 hit 10 cycles, miss 100
- 1.5 memory references per instruction

Ask: Local miss rate, AMAT, stall cycles per instruction, and those without L2 cache

With L2 cache
- Local miss rate = 50%
- \[ AMAT = 1 + 4\% \times (10 + 50\% \times 100) = 3.4 \]
- Average Memory Stalls per Instruction = \((3.4 - 1.0) \times 1.5 = 3.6\)

Without L2 cache
- AMAT = 1 + 4\% \times 100 = 5
- Average Memory Stalls per Inst = \((5 - 1.0) \times 1.5 = 6\)

Assume ideal CPI = 1.0, performance improvement = \((6 + 1) / (3.6 + 1) = 52\%\)
Comparing Local and Global Miss Rates

- First-level cache: split 64K+64K 2-way
- Second-level cache: 4K to 4M
- In practice: caches are inclusive

Global miss rate approaches single cache miss rate provided that the second-level cache is much larger than the first-level cache

Global miss rate is what matters
Compare Execution Times

- L1 configuration as in the last slide
- L2 cache 256K-8M, 2-way
- Normalized to 8M cache with 1-cycle latency

Performance is not sensitive to L2 latency
Larger cache size makes a big difference
Impacts of L2 Associativity

L2 cache:
- Direct mapped hit time = 10 cycles
- 2-way 10.1 cycles
- Local miss rate for direct mapped = 25%
- Local miss rate for two-way = 20%
- L2 Miss penalty = 100

Compare L1 miss penalty for 1-way and 2-way

Answer:
- Miss penalty 1-way = 10 + 25%\times100 = 35
- Miss penalty 2-way = 10.1 + 20%\times100 = 30.1

If treat 10.1 as 11 cycles:
- Miss penalty 2-way = 11 + 20%\times100 = 31