Cache Performance

CPU time = (CPU execution clock cycles + Memory stall clock cycles) x clock_cycle_time

Memory stall_clock_cycles = Memory accesses x Miss rate x Miss penalty

Example

- Assume every instruction takes 1 cycle
- Miss penalty = 20 cycles
- Miss rate = 10%
- 1000 total instructions, 300 memory accesses
- Memory stall cycles? CPU clocks?

Memory stall cycles = 300 * 0.10 * 20 = 600

CPU clocks = 1000 + 600 = 1600

60% slower because of cache misses!
Improving Cache Performance

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

Reducing Misses

° Classifying Misses: 3 Cs
  • Compulsory—The first access to a block ALWAYS misses, since that block is not in the cache. Such misses are also called cold start misses or first reference misses. (Misses in Infinite Cache)
  • Capacity—if the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being evicted from the cache and later retrieved. (Misses in Size X Cache)
  • Conflict—if the block-placement strategy is set associative or direct-mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called collision misses or interference misses. (Misses in N-way Associative, Size X Cache)

Cache Performance

° Your program and caches
° Can you affect performance?
° Think about 3Cs
Reducing Misses by Compiler Optimizations

° Instructions
• Reorder procedures in memory so as to reduce misses
• Profiling to look at conflicts
• McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks

° 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 a sequences of 2 loops that iterate the same number of times, and that use some variables in common
• Blocking: improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows

Merging Arrays Example

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

/* After */
struct merge {
  int val;
  int key;
};

struct merge merged_array[SIZE];

Reducing conflicts between val & key

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 * 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 * x[i][j];

Sequential accesses instead of striding through memory every 100 words
Loop Fusion Example

```c
/* 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];

/* 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

Blocking Example

```c
/* 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;

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

* Two Inner Loops:
  - Read all N x N 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:
  - 3 N x N => no capacity misses; otherwise ...

* Idea: compute on B x B submatrix that fits
Reducing Conflict Misses by Blocking

Conflicts misses in non Fully Associative caches vs. Blocking size
- Lam et al [1991] using a blocking factor B of 24 produced 1/5 the misses vs. a blocking factor of 48, although both fit in cache
- This result depends on array dimensions, also

Summary of Compiler Optimizations to Reduce Cache Misses

<table>
<thead>
<tr>
<th>Name</th>
<th>Merge arrays</th>
<th>Loop interchange</th>
<th>Loop fusion</th>
<th>Blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpenta (nasa7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc (nasa7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ftonc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>htrc (nasa7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>memo (nasa7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spice</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chedonky (nasa7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>omp donne</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary

- Cost Effective Memory Hierarchy
- Split Instruction and Data Cache
- 4 Questions
- CPU cycles/time, Memory Stall Cycles
- Your programs and cache performance

Next Time

- Virtual Memory