Lecture 18b: Memory Hierarchy—7 Ways to Reduce Misses

Professor Alvin R. Lebeck
Computer Science 220
Fall 1999

Admin

• Midterm Friday in class ~1 hour
• Review Wednesday
Review: Who Cares About the Memory Hierarchy?

- Processor Only Thus Far in Course:
  - CPU cost/performance, ISA, Pipelined Execution

1980: no cache in µproc; 1995 2-level cache, 60% trans. on Alpha 21164 µproc (150 clock cycles for a miss!)

---

Cache Performance

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

Memory stall clock cycles = (Reads x Read miss rate x Read miss penalty + Writes x Write miss rate x Write miss penalty)

Memory stall clock cycles = Memory accesses x Miss rate x Miss penalty
Cache Performance

CPUtime = IC \times \left( CPI_{\text{execution}} + \text{Mem accesses per instruction} \times \text{Miss rate} \times \text{Miss penalty} \right) \times \text{Clock cycle time}

Misses per instruction = \text{Memory accesses per instruction} \times \text{Miss rate}

CPUtime = IC \times \left( CPI_{\text{execution}} + \text{Misses per instruction} \times \text{Miss penalty} \right) \times \text{Clock cycle time}

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 is not in the cache, so the block must be brought into the cache. These 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 discarded 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)*

Reducing Misses

1. Larger Blocksize
2. Higher Associativity
3. Victim Cache
4. Pseudo-Associativity
5. Hardware Prefetch
6. Software Prefetch
7. Program Transformation
7. 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 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
  – Nonlinear Data Layout: Don’t use canonical row-major or column-major layout for arrays

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**

/* 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
/* 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:
  – 3 NxN => no capacity misses; otherwise ...

• Idea: compute on BxB submatrix that fits

/* 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;
                     }

• Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
• B called Blocking Factor
• Conflict Misses Too?
Reducing Conflict Misses by Blocking

- Conflict misses in caches not FA vs. Blocking size
  - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

Layout and Cache Behavior

- Tile elements spread out in memory because of column-major mapping
- Fixed mapping into cache
- Self-interference in cache

© Alvin R. Lebeck 1999
Making Tiles Contiguous

- Elements of a quadrant are contiguous
- Recursive layout
- Elements of a tile are contiguous
- No self-interference in cache

Non-linear Layout Functions

- Different locality properties
- Different inclusion properties
- Different addressing costs

4-D blocked Morton order Hilbert order
### Performance Improvement

<table>
<thead>
<tr>
<th>CPU</th>
<th>UltraSPARC 2i</th>
<th>UltraSPARC 2</th>
<th>Alpha 21164</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock rate</td>
<td>300MHz</td>
<td>300 MHz</td>
<td>500MHz</td>
</tr>
<tr>
<td>L1 cache</td>
<td>16KB/32B/1</td>
<td>16KB/32B/1</td>
<td>8KB/32B/1</td>
</tr>
<tr>
<td>L2 cache</td>
<td>512KB/64B/1</td>
<td>2MB/64B/1</td>
<td>96KB/64B/3</td>
</tr>
<tr>
<td>L3 cache</td>
<td>2MB/64B/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>320MB</td>
<td>512MB</td>
<td>512MB</td>
</tr>
<tr>
<td>TLB entries</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Page size</td>
<td>8KB</td>
<td>8KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Ultra 10</th>
<th>Ultra 60</th>
<th>Miata</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLKXM</td>
<td>0.93</td>
<td>1.06</td>
<td>0.95</td>
</tr>
<tr>
<td>RECXM</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>STRASSENM</td>
<td>0.87</td>
<td>0.79</td>
<td>0.91</td>
</tr>
<tr>
<td>CHOL</td>
<td>0.78</td>
<td>0.85</td>
<td>0.67</td>
</tr>
<tr>
<td>STDHAAR</td>
<td>0.68</td>
<td>0.67</td>
<td>0.64</td>
</tr>
<tr>
<td>NONHAAR</td>
<td>0.62</td>
<td>0.61</td>
<td>0.58</td>
</tr>
</tbody>
</table>

### Comparison with TSS

**BMXM, comparison with TSS**

- TSS/Ultra 10
- 4D/Ultra 10, t=17
- 4D/Ultra 10, l=17
- 4D/Ultra 10, l=20

**BMXM, comparison with TSS**

- TSS/Miata
- 4D/Miata, t=17
- 4D/Miata, l=30
Summary of Compiler Optimizations to Reduce Cache Misses

Summary

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

- **3 Cs:** Compulsory, Capacity, Conflict
  - How to eliminate them
- **Program Transformations**
  - Change Algorithm
  - Change Data Layout
- **Implication:** Think about caches if you want high performance!