Review Problem 36

Why might a compiler do this transformation?

/ * Before */
for (j=0; j<2000; j++)
  for (i=0; i<2000; i++)
    x[i][j] += 1;
/ * After */
for (i=0; i<2000; i++)
  for (j=0; j<2000; j++)
    x[i][j] += 1;

\[ x[i][j] + x[i][j+1] \text{ adjacent} \]
The Solution

By taking advantage of the principle of locality:

Provide as much memory as is available in the cheapest technology.
Provide access at the speed offered by the fastest technology.

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
<th>Cache</th>
<th>Main Memory</th>
<th>Disk Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>1 cycle</td>
<td>1-7 cycles</td>
<td>100 cycles</td>
<td>10,000 cycles</td>
</tr>
<tr>
<td>Capacity</td>
<td>1x (norm.)</td>
<td>64-4Kx</td>
<td>4Mx</td>
<td>1Gx</td>
</tr>
</tbody>
</table>
Cache Terminology

Block – Minimum unit of information transfer between levels of the hierarchy
  Block addressing varies by technology at each level
  Blocks are moved one level at a time
Upper vs. lower level – “upper” is closer to CPU, “lower” is further away
Hit – Data appears in a block in that level
  Hit rate – percent of accesses hitting in that level
  Hit time – Time to access this level
    Hit time = Access time + Time to determine hit/miss
Miss – Data does not appear in that level and must be fetched from lower level
  Miss rate – percent of misses at that level = (1 – hit rate)
  Miss penalty – Overhead in getting data from a lower level
    Miss penalty = Lower level access time + Replacement time + Time to deliver to processor
    Miss penalty is usually MUCH larger than the hit time
Cache Access Time

Average access time

Access time = (hit time) + (miss penalty) \times (miss rate)

Want high hit rate & low hit time, since miss penalty is large

Average Memory Access Time (AMAT)

Apply average access time to entire hierarchy.
### Cache Access Time Example

<table>
<thead>
<tr>
<th>Level</th>
<th>Hit Time</th>
<th>Hit Rate</th>
<th>Access Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>1 cycle</td>
<td>95%</td>
<td>1 + 0.05 * 65 = 4.25</td>
</tr>
<tr>
<td>L2</td>
<td>10 cycles</td>
<td>90%</td>
<td>10 + 0.1 * 550 = 65</td>
</tr>
<tr>
<td>Main Memory</td>
<td>50 cycles</td>
<td>99%</td>
<td>50 + 0.01 * 50,000 = 550</td>
</tr>
<tr>
<td>Disk</td>
<td>50,000 cycles</td>
<td>100%</td>
<td>50,000</td>
</tr>
</tbody>
</table>

Note: Numbers are local hit rates – the ratio of access that go to that cache that hit (remember, higher levels filter accesses to lower levels)

\[
AMAT = 1 + 0.05 \left( 10 + 0.1 \left( 50 + 0.01 \left( 50,000 \right) \right) \right)
\]

\[
= 1 + 0.05 \times 10 + 0.05 \times 1 \times 50 + 0.05 \times 1 \times 0.1 \times 50,000
\]
Handling A Cache Miss

Processor expects a cache hit (1 cycle), so no effect on hit.

Instruction Miss
1. Send the original PC to the memory
2. Instruct memory to perform a read and wait (no write enables)
3. Write the result to the appropriate cache line
4. Restart the instruction

Data Miss
1. Stall the pipeline (freeze following instructions)
2. Instruct memory to perform a read and wait
3. Return the result from memory and allow the pipeline to continue
Exploiting Locality

Spatial locality
Move blocks consisting of multiple contiguous words to upper level

Temporal locality
Keep more recently accessed items closer to the processor
When we must evict items to make room for new ones, attempt to keep more recently accessed items
Cache Arrangement

How should the data in the cache be organized?

Caches are smaller than the full memory, so multiple addresses must map to the same cache “line”

**Direct Mapped** – Memory addresses map to particular location in that cache

**Fully Associative** – Data can be placed anywhere in the cache

**N-way Set Associative** – Data can be placed in a limited number of places in the cache depending upon the memory address
Direct Mapped Cache

4 byte direct mapped cache with 1 byte blocks
Optimize for spatial locality (close blocks likely to be accessed soon)

Memory Address

Cache Address

Bottom two bits of memory address = 0/6 cache size
Finding A Block

Each location in the cache can contain a number of different memory locations. Cache 0 could hold 0, 4, 8, 12, ...

We add a tag to each cache entry to identify which address it currently contains. What must we store?

- Portion of address \textit{NOT} used to find to specific line in the cache
- Valid bit - true if real data, false if garbage
Cache Tag & Index

Assume $2^9$ byte direct mapped cache with 1 byte blocks

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1
0 0 0 0 0 0 1 1
```

Cache Tag = 57

Cache Index = 03

Valid Bit  Tag  Data
Review Problem 37

If you can speed up any level's hit time by a factor of two, which is the best to speed up?

<table>
<thead>
<tr>
<th>Level</th>
<th>Hit Time</th>
<th>Hit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>1 cycle</td>
<td>95%</td>
</tr>
<tr>
<td>L2</td>
<td>10 cycles</td>
<td>90%</td>
</tr>
<tr>
<td>Main Memory</td>
<td>50 cycles</td>
<td>99%</td>
</tr>
<tr>
<td>Disk</td>
<td>50,000 cycles</td>
<td>100%</td>
</tr>
</tbody>
</table>

\[
\text{AMAT} = 1 + 0.05 \times 10 + 0.05 \times 0.1 \times 50 + 0.05 \times 0.1 \times 0.01 \times 50,000
\]

\[
= 1 + 0.5 + 0.25 + 2.5
\]

\[
= 4.25
\]
### Cache Access Example

Assume 4 byte cache

Access pattern:

<table>
<thead>
<tr>
<th>Address</th>
<th>Valid Bit</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001</td>
<td>0</td>
<td>0</td>
<td>M[i]</td>
</tr>
<tr>
<td>00110</td>
<td>1</td>
<td>1</td>
<td>MC6J</td>
</tr>
<tr>
<td>00001</td>
<td>2</td>
<td>2</td>
<td>M[26]</td>
</tr>
<tr>
<td>11010</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

CT: Cold Start
CI: Cold Insight
**Cache Size Example**

How many total bits are requires for a direct-mapped cache with 64 KB of data and 1-byte blocks, assuming a 32-bit address?

- **Index bits:** 16
- **Bits/block:** 256
  - Data: 85
  - Valid: 16
  - Tag: 16
- **Total size:** $2^{5} \times 2^{14} \times 2^{16} = 25 \times 8 \text{ KB} = 200 \text{ KB}$

Cache is 3x larger than the actual data capacity.
# Cache Block Overhead

Previous discussion assumed direct mapped cache 1 byte blocks

- Uses temporal locality by holding on to previously used values
- Does not take advantage of spatial locality
- Significant area overhead for tag memory

Take advantage of spatial locality & amortize tag memory via larger block size

<table>
<thead>
<tr>
<th>Valid Bit</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2^{n-1}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Blocks

Assume $2^{10}$ line direct mapped cache with $2^9$ byte blocks

![Diagram of cache blocks with valid bit, tag, index, byte select, and data.]
Cache Block Example

Given a cache with 64 blocks and a block size of 16 bytes, what block number does byte address $1200_{10}$ map to?

- **Byte Select** = $1200_{10}$ $\%$ 16 = 0
- **Addr / Blocksize**
  - Cache Tag, Cache Index$^3$ = Block Number = $\lfloor 1200_{10} / 16 \rfloor = 75_{10}$
  - Cache Index = Block Number $\%$ # of blocks = 75 $\%$ 64 = 11$_{10}$
  - Cache Tag = $\lfloor$ Block Number / # of blocks $\rfloor$ = $\lfloor 75 / 64 \rfloor$ = 1$_{10}$
Block Size Tradeoff

In general, larger block size take advantage of spatial locality **BUT**:
- Larger block size means larger miss penalty:
  - Takes longer time to fill up the block
- If block size is too big relative to cache size, miss rate will go up
  - Too few cache blocks

---

**Miss Penalty** vs **Miss Rate** vs **Average Access Time**

- **Miss Penalty** vs **Block Size**:
  - Exploits Spatial Locality
  - Fewer blocks: compromises temporal locality
- **Miss Rate** vs **Block Size**:
  - Increased Miss Penalty & Miss Rate
  - Optimal Block Size

---

150