CSE 502 Graduate Computer Architecture

Lec 21-22 – Advanced Memory Hierarchy and Application Tuning

Larry Wittie
Computer Science, StonyBrook University

http://www.cs.sunysb.edu/~cse502 and ~lw

Slides adapted from David Patterson, UC-Berkeley cs252-s06
Outline

• Eleven Advanced Cache Optimizations
• Memory Technology and DRAM Optimizations
• Conclusions
Why More on Memory Hierarchy?

![Graph showing the growing performance gap between processor and memory from 1980 to 2010.](chart.png)
Review: 6 Basic Cache Optimizations

- Reducing hit time
  1. Giving Reads Priority over Writes
     - E.g., Read completes before earlier writes in write buffer
  2. Avoiding Address Translation during Cache Indexing (limited to caches with small indices)

- Reducing Miss Penalty
  3. Multilevel Caches

- Reducing Miss Rate
  4. Larger Block size (fewer Compulsory misses)
  5. Larger Cache size (fewer Capacity misses)
  6. Higher Associativity (fewer Conflict misses)
Eleven Advanced Cache Optimizations

- Reducing hit time
  1. Small and simple caches
  2. Way prediction
  3. Trace caches

- Increasing cache bandwidth
  4. Pipelined caches
  5. Multibanked caches
  6. Nonblocking caches

- Reducing Miss Penalty
  7. Critical word first
  8. Merging write buffers

- Reducing Miss Rate
  9. Compiler optimizations

- Reducing miss penalty or miss rate via parallelism
  10. Hardware prefetching
  11. Compiler prefetching
1. Fast Hit Times via Small, Simple Caches

- Index tag memory and then compare takes time
- \( \Rightarrow \) Small cache can help hit time since smaller memory takes less time to index to find right set of block(s) in cache
  - E.g., fast L1 caches were same small size for 3 generations of AMD microprocessors: K6, Athlon, and Opteron
  - Also, having a L2 cache small enough to fit on-chip with the processor avoids time penalty of going off chip (~10X longer data latency off-chip)

- Simple \( \Rightarrow \) direct mapping
  - Overlap tag check with data transmission since no choice (kill data out if tag bad)

- Access time estimate for 90 nm using CACTI model 4.0
  - Median ratios of access time relative to the direct-mapped caches are 1.32, 1.39, and 1.43 for 2-way, 4-way, and 8-way caches
2. Fast Hit Times via Way Prediction

• How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SetAssoc cache?

• **Way prediction**: keep extra bits in cache to predict the “way,” or block within the set, of next cache access.
  - Multiplexor is set early to select desired block, only 1 tag comparison performed that clock cycle in parallel with reading the cache data
  - Miss ⇒ 1st check other blocks for matches in next clock cycle

  ![Hit Time Diagram](image)

• **Accuracy** ≈ 85%

• **Drawback**: hard to tune CPU pipeline if hit time varies from 1 or 2 cycles
  - Used for instruction caches vs. data caches so each way-miss extra cycle before fetch any following instructions
3. Fast Hit times via Micro-Ops Trace Cache (Pentium 4 only; and last time?)

- Find more instruction level parallelism?
  How avoid translation from x86 to microops?

- Trace cache in Pentium 4
  1. Dynamic traces of the executed instructions vs. static sequences of instructions as determined by layout in memory
     - Built-in branch predictor
  2. Cache the micro-ops vs. x86 instructions
     - Decode/translate from x86 to micro-ops whenever trace cache misses

  \[+\] 1. \(\Rightarrow\) better utilize long blocks (do not exit in middle of block, do not enter at label in middle of block)

  \[-\] 1. \(\Rightarrow\) complicated address mapping since addresses no longer aligned to power-of-2 multiples of word size

  \[-\] 1. \(\Rightarrow\) instructions may appear multiple times in multiple dynamic traces due to different branch outcomes
4: Increase Cache Bandwidth by Pipelining

- Pipeline cache access to maintain bandwidth, even though pipe gives higher latency for each access
- Number of instruction cache access pipeline stages:
  1 for Pentium
  2 for Pentium Pro through Pentium III
  4 for Pentium 4 \{almost = 4 CPU ports to icache\}
- ⇒ greater penalty on mispredicted branches: restart pipelined stream of memory addresses at new PC
- ⇒ more clock cycles between the issue of a load and the availability of the loaded data
5. Increase Cache Bandwidth: Non-Blocking Caches – Major for MPs

- **Non-blocking cache** or **lockup-free cache** allow data cache to continue to supply cache hits during a miss
  - helps if Full/Empty bits on all registers (to allow execution to go on until missed datum is actually used) or out-of-order execution
  - requires **multi-bank memories** for the non-blocking cache
- "**hit under miss**" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- "**hit under multiple miss**" or "**miss under miss**" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires multiple main memory banks (otherwise cannot support)
  - Pentium Pro allows 4 outstanding memory misses
Value of Hit Under Miss for SPEC (old data)

Hit Under i Misses

```
<table>
<thead>
<tr>
<th>Integer</th>
<th>Floating Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>If i   = 0</td>
<td>1</td>
</tr>
<tr>
<td>AMAT= 0.68</td>
<td>0.52</td>
</tr>
<tr>
<td>0.24</td>
<td>0.20</td>
</tr>
<tr>
<td>8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss, SPEC 92</td>
<td></td>
</tr>
</tbody>
</table>
```

• FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
• Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19

"Hit under i Misses"

AMAT = average miss access time
6: Increase Cache Bandwidth via Multiple Banks – Major for Multicore

• Rather than treat the cache as a single monolithic block, divide it into independent banks that can support simultaneous accesses
  – E.g., T1 (“Niagara”) L2 has 4 banks

• Banking works best when accesses naturally spread themselves across banks ⇒ mapping of addresses to banks affects behavior of memory system

• A simple mapping that works well is “sequential interleaving” ⇒ the next block of memory goes to the next bank of memory
  – Spread memory block indices sequentially across banks
  – E.g., if there are 4 banks, Bank 0 has all blocks whose index modulo 4 is 0; bank 1 has all blocks whose index modulo 4 is 1; ...
7. Reduce Miss Penalty: Early Restart (minor) and Critical Word First (Major)

- Do not wait for full block before restarting CPU
- **Early restart**—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Spatial locality ⇒ tend to want next sequential word, so first access to a block is normally to 1st word, but next is to 2nd word, which may stall again and so on, so benefit from early restart alone is not clear
- **Critical Word First**—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block
  - Long blocks more popular today ⇒ Critical Word 1st Widely Used
8. Merge Multiple Adjacent New Words in Write Buffer to Reduce Miss Penalty

- Write buffer allows processor to continue without waiting to finish write to next lower memory/cache
- If buffer contains blocks of modified words, not just a single word per entry, addresses can be checked to see if the address of a newly written datum matches an address in an existing write buffer entry
- If so, new datum is combined with that existing entry
- For write-through caches, can increase block sizes of writes to lower memory from writes to individual words to writes to several sequential words, which allows more efficient use of memory system
- The Sun T1 (Niagara) processor, among many others, uses write merging
9. Reduce Misses by Compiler Optimizations

- McFarling [1989] used software to reduce cache misses by 75% for an 8KB direct-mapped cache (4 byte blocks)

- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts (using tools they developed)

- Data (All 4 methods widely use in fast scientific codes.)
  - Merging Arrays: improve spatial locality by single array of compound elements vs. 2 arrays
  - Loop Interchange: change nesting of loops to access data in the order that they are stored in memory
  - Loop Fusion: Combine 2 non-dependent loops with the same looping structure so more accesses to all common variables in each iteration
  - Blocking: Improve temporal locality by accessing “blocks” of data (equal cache block size) repeatedly vs. going down whole columns or rows and moving from cache block to cache block rapidly
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];

Reduce conflicts between val & key; and 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 * x[i][j];
/* After, since x[i][j+1] follows x[i][j] in memory*/
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; improve 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];
        }

Two misses per access to a & c vs. one miss per access; improve spatial locality
Blocking Example

/* 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 are a function of N & Cache Size:
  – $2N^3 + N^2 \Rightarrow$ (assuming no conflict; otherwise …)

• Idea: Compute on BxB submatrix fitting in 1 cache block

For large N, these long accesses repeatedly flush cache blocks that are needed again soon

Better way
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 is **Blocking Factor**: Larger B => smaller blocks
• Capacity Misses fall from $2N^3 + N^2$ to $2N^3/B + N^2$
• Do Conflict Misses fall also?
Summary of Compiler Optimizations to Reduce Cache Misses (by hand)

- **merged arrays**
- **loop interchange**
- **loop fusion**
- **blocking**

Performance Improvement

- **vpenta** (nasa7)
- **gmyty** (nasa7)
- **tomcatv**
- **btrix** (nasa7)
- **mxm** (nasa7)
- **spice**
- **cholesky** (nasa7)
- **compress**
10. Reduce Misses by **Hardware Prefetching of Instructions & Data**

- Prefetching relies on having extra memory bandwidth that can be used without penalty since some prefetched values unused

- **Instruction Prefetching**
  - Typically, a CPU fetches 2 blocks on a miss: the requested block and the next consecutive block.
  - Requested block is placed in instruction cache when it returns, and prefetched block is placed into instruction stream buffer

- **Data Prefetching**
  - Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages
  - Prefetching invoked whenever 2 successive L2 cache misses to 1 page, if distance between those cache blocks is < 256 bytes

---

**Performance Improvement**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SPECint2000</th>
<th>SPECfp2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>gap</td>
<td>1.16</td>
<td>1.17</td>
</tr>
<tr>
<td>mcf</td>
<td>1.45</td>
<td>1.32</td>
</tr>
<tr>
<td>farad</td>
<td>1.18</td>
<td>1.36</td>
</tr>
<tr>
<td>wupwise</td>
<td>1.20</td>
<td>1.29</td>
</tr>
<tr>
<td>galgel</td>
<td>1.21</td>
<td>1.30</td>
</tr>
<tr>
<td>facerec</td>
<td>1.26</td>
<td>1.32</td>
</tr>
<tr>
<td>swim</td>
<td>1.29</td>
<td>1.40</td>
</tr>
<tr>
<td>applu</td>
<td>1.32</td>
<td>1.41</td>
</tr>
<tr>
<td>lucas</td>
<td>1.40</td>
<td>1.42</td>
</tr>
<tr>
<td>mgrid</td>
<td>1.49</td>
<td>1.43</td>
</tr>
<tr>
<td>equake</td>
<td>1.97</td>
<td>1.44</td>
</tr>
</tbody>
</table>
11. Reduce Misses by **Software** Prefetching Data

- **Data Prefetch**
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution

- **Issuing Prefetch Instructions takes time**
  - Is cost of issuing prefetch instructions < savings from reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth
Compiler Optimization vs. Memory Hierarchy Search

• Compiler tries to figure out memory hierarchy optimizations

• New approach: “Auto-tuners” 1st run variations of program on computer to find best combinations of optimizations (blocking, padding, ...) and algorithms, then produce C code to be compiled for that computer

• “Auto-tuner” targeted to numerical method
  – E.g., PHiPAC (BLAS), Atlas (BLAS), Sparsity (Sparse linear algebra), Spiral (DSP), FFT-W
Sparse Matrix – Search for Blocking

for finite element problem [Im, Yelick, Vuduc, 2005]

900 MHz Itanium 2, Intel C v8: ref=275 Mflop/s

Best: 4x2

Reference
### Best Sparse Blocking for 8 Computers

<table>
<thead>
<tr>
<th>row block size ($r$)</th>
<th>Intel Pentium M</th>
<th>Sun Ultra 2, Sun Ultra 3, AMD Opteron</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>IBM Power 4, Intel/HP Itanium</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Intel/HP Itanium 2</td>
<td>IBM Power 3</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- All possible column block sizes selected for 8 computers. How could compiler know?
<table>
<thead>
<tr>
<th>Technique</th>
<th>Hit Time</th>
<th>Bandwidth</th>
<th>Miss Penalty</th>
<th>Miss Rate</th>
<th>HW cost/complexity</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small and simple caches</td>
<td>+</td>
<td></td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>Trivial; widely used</td>
</tr>
<tr>
<td>Way-predicting caches</td>
<td>+</td>
<td></td>
<td>+</td>
<td>1</td>
<td></td>
<td>Used in Pentium 4</td>
</tr>
<tr>
<td>Trace caches</td>
<td>+</td>
<td></td>
<td>+</td>
<td>3</td>
<td></td>
<td>Used in Pentium 4</td>
</tr>
<tr>
<td>Pipelined cache access</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>1</td>
<td></td>
<td>Widely used</td>
</tr>
<tr>
<td>Nonblocking caches</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>3</td>
<td></td>
<td>Widely used</td>
</tr>
<tr>
<td>Banked caches</td>
<td>+</td>
<td></td>
<td>+</td>
<td>1</td>
<td></td>
<td>Used in L2 of Opteron and Niagara</td>
</tr>
<tr>
<td>Critical word first and early restart</td>
<td>+</td>
<td></td>
<td>+</td>
<td>2</td>
<td></td>
<td>Widely used</td>
</tr>
<tr>
<td>Merging write buffer</td>
<td>+</td>
<td></td>
<td>+</td>
<td>1</td>
<td></td>
<td>Widely used with write through</td>
</tr>
<tr>
<td>Compiler techniques to reduce cache misses</td>
<td></td>
<td></td>
<td>+</td>
<td>0</td>
<td></td>
<td>Software is a challenge; some computers have compiler option</td>
</tr>
<tr>
<td>Hardware prefetching of instructions and data</td>
<td></td>
<td></td>
<td>+</td>
<td>2 instr., 3 data</td>
<td></td>
<td>Many prefetch instructions; AMD Opteron prefetches data</td>
</tr>
<tr>
<td>Compiler-controlled prefetching</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>3</td>
<td></td>
<td>Needs nonblocking cache; in many CPUs</td>
</tr>
</tbody>
</table>
Outline

- Eleven Advanced Cache Optimizations
- Memory Technology and DRAM optimizations
- Conclusions
Main Memory Background

- **Performance of Main Memory:**
  - **Latency**: Cache Miss Penalty
    » *Access Time*: time between request and word arrival
    » *Cycle Time*: minimum time between requests
  - **Bandwidth**: I/O & Large Block Miss Penalty (L2)

- **Main Memory is **DRAM**: Dynamic Random Access Memory**
  - Dynamic since needs to be *refreshed* periodically (every 8 ms, 1% time)
  - Addresses divided into 2 halves (Memory as a 2D matrix):
    » *RAS* or *Row Access Strobe*
    » *CAS* or *Column Access Strobe*

- **Cache uses **SRAM**: Static Random Access Memory**
  - No refresh (but needs 6 transistors/bit vs. 1 transistor/bit for DRAM)
  - *Size*: SRAM/DRAM - 4-8
  - *Cost/Cycle time*: SRAM/DRAM - 8-16
Main Memory Deep Background

• “Out-of-Core”, “In-Core,” “Core Dump”?  
• “Core memory” was used in the 1950s and 1960s  
• Each bit of core was a non-volatile, magnetic torus (a tiny red-brown donut, fuzzy with sense & driver wires)  
• Lost out to 4 Kbit DRAM (2005: 512Mbits / DRAM chip)  
• Access time of core was 750 ns, cycle time 1500-3000 ns
DRAM Logical Organization (4 Mbit = $2^{22}\text{b}$)

Square root of number of memory bits is in each RAS & CAS address
The Quest for DRAM Performance

1. Fast Page mode
   - Add timing signals that allow repeated accesses to row buffer without another row access time
   - Such a buffer comes naturally, since each array buffers 1024 to 2048 bits for each access

2. Synchronous DRAM (SDRAM)
   - Add a clock signal to DRAM interface, so that the repeated transfers would not suffer the time overhead of synchronizing with the DRAM controller

3. Double Data Rate (DDR SDRAM)
   - Transfer data on both the rising edge and falling edge of the DRAM clock signal ⇒ doubling the peak data rate
   - DDR2 lowers power by dropping the voltage from 2.5 to 1.8 volts + offers higher clock rates: up to 400 MHz
   - DDR3 drops to 1.5 volts + higher clock rates: up to 800 MHz

• Improved Bandwidth, not Latency
## DRAM name based on Peak Chip Transfers / Sec

### DIMM name based on Peak DIMM MBytes / Sec

<table>
<thead>
<tr>
<th>Standard Clock Rate (MHz)</th>
<th>M transfers / second</th>
<th>DRAM Name</th>
<th>Mbytes/s/ DIMM Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR 133</td>
<td>266</td>
<td>DDR266</td>
<td>2128</td>
</tr>
<tr>
<td>DDR 150</td>
<td>300</td>
<td>DDR300</td>
<td>2400</td>
</tr>
<tr>
<td>DDR 200</td>
<td>400</td>
<td>DDR400</td>
<td>3200</td>
</tr>
<tr>
<td>DDR2 266</td>
<td>533</td>
<td>DDR2-533</td>
<td>4264</td>
</tr>
<tr>
<td>DDR2 333</td>
<td>667</td>
<td>DDR2-667</td>
<td>5336</td>
</tr>
<tr>
<td>DDR2 400</td>
<td>800</td>
<td>DDR2-800</td>
<td>6400</td>
</tr>
<tr>
<td>DDR3 533</td>
<td>1066</td>
<td>DDR3-1066</td>
<td>8528</td>
</tr>
<tr>
<td>DDR3 666</td>
<td>1333</td>
<td>DDR3-1333</td>
<td>10664</td>
</tr>
<tr>
<td>DDR3 800 x 2</td>
<td>1600</td>
<td>DDR3-1600</td>
<td>12800</td>
</tr>
</tbody>
</table>

*Fastest for sale 4/06 ($125/GB)*
Need for Error Correction!

**Motivation:**
- Failures/time *proportional* to number of bits!
- As DRAM cells shrink, more vulnerable to errors

**Went through period in which failure rate was low enough without error correction that people did not do correction**
- DRAM banks too large now
- Servers have always corrected memory systems

**Basic idea: add redundancy through parity bits**
- Common configuration: Random error correction
  - SEC-DED (single error correct, double error detect)
  - One example: 64 data bits + 8 parity bits (11% overhead)
- Really want to handle failures of physical components as well
  - Organization is multiple DRAMs/DIMM, multiple DIMMs
  - Want to recover from failed DRAM and failed DIMM!
  - “Chip kill” handle failures the width of a single DRAM chip
And in Conclusion

- Memory wall inspires optimizations since so much performance lost there
  - Reducing hit time: Small and simple caches, Way prediction, Trace caches
  - Increasing cache bandwidth: Pipelined caches, Multibanked caches, Nonblocking caches
  - Reducing Miss Penalty: Critical word first, Merging write buffers
  - Reducing Miss Rate: Compiler optimizations
  - Reducing miss penalty or miss rate via parallelism: Hardware prefetching, Compiler prefetching

- “Auto-tuners” search replacing static compilation to explore optimization space?

- DRAM – Continuing Bandwidth innovations: Fast page mode, Synchronous, Double Data Rate
Unused Spring 2010
Reduce Conflict Misses by Blocking

- Conflict misses in non-F.A. caches vs. Blocking size
  - Lam et al [1991] found a blocking factor of 24 had a fifth the misses vs. 48 despite both fitting in one cache block