New-School Machine Structures (It’s a bit more complicated!)

• Parallel Requests
  Assigned to computer
  e.g., Search “Katz”
• ParallelThreads
  Assigned to core
  e.g., Lookup, Ads
• Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
• Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
• Hardware descriptions
  All gates @ one time
• Programming Languages

Caches are software-invisible

• Load and store instructions just access large memory (32-bit addresses in MIPS); hardware automatically moves data in and out of cache
• Even if programmer writes applications not knowing about caches, we observe temporal and spatial locality in memory accesses
• Performance improves (over no caches) even when programmer unaware of cache’s existence

CPI/Miss Rates/DRAM Access
Specint2006 on AMD Barcelona (64KB L1, 512KB L2)

<table>
<thead>
<tr>
<th>Name</th>
<th>CPI</th>
<th>L1 Perf. misses/1000 instr</th>
<th>L2 Perf. misses/1000 instr</th>
<th>L1 Perf. instructions accesses/1000 instr</th>
<th>L2 Perf. instructions accesses/1000 instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>perl</td>
<td>0.75</td>
<td>3.5</td>
<td>1.1</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>bear</td>
<td>0.80</td>
<td>11.0</td>
<td>5.8</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td>1.72</td>
<td>24.3</td>
<td>13.4</td>
<td>14.8</td>
<td></td>
</tr>
<tr>
<td>mcf</td>
<td>10.00</td>
<td>106.8</td>
<td>88.0</td>
<td>88.5</td>
<td></td>
</tr>
<tr>
<td>go</td>
<td>1.09</td>
<td>4.5</td>
<td>1.4</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>tower</td>
<td>0.80</td>
<td>4.4</td>
<td>2.5</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>x86pgm</td>
<td>0.86</td>
<td>1.9</td>
<td>0.6</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>lihotquantum</td>
<td>1.61</td>
<td>33.0</td>
<td>33.1</td>
<td>47.7</td>
<td></td>
</tr>
<tr>
<td>h264enc</td>
<td>0.80</td>
<td>8.8</td>
<td>1.6</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>smmp</td>
<td>2.94</td>
<td>36.9</td>
<td>27.7</td>
<td>29.8</td>
<td></td>
</tr>
<tr>
<td>aster</td>
<td>1.79</td>
<td>16.3</td>
<td>9.2</td>
<td>8.2</td>
<td></td>
</tr>
<tr>
<td>smpack2k</td>
<td>2.70</td>
<td>38.0</td>
<td>15.8</td>
<td>11.4</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>1.35</td>
<td>13.6</td>
<td>7.5</td>
<td>5.4</td>
<td></td>
</tr>
</tbody>
</table>

Performance Programming:
Adjust software accesses to improve miss rate

• Now that understand how caches work, can revise program to improve cache utilization
  – Cache size
  – Block size
  – Multiple levels
• “Cache-Aware” performance optimizations
  – but code would still work even if no caches present
Performance of Loops over Arrays

- Array performance often limited by memory speed
- OK to access memory in different order as long as get correct result
- Goal: Increase performance by minimizing traffic from cache to memory
  - That is, reduce Miss rate by getting better reuse of data already in cache

Alternate Matrix Layouts in Memory

- A matrix is a 2-D array of elements, but memory addresses are "1-D" (0...MaximumMemoryAddress)
- Conventions for matrix layout:
  - by column, or "column major" (Fortran default); \( A(i,j) \) at \( A+i+j*n \)
  - by row, or "row major" (C default) \( A(i)[j] \) at \( A+i*n+j \)

Cache Blocks* in Matrix

Column Major
Row Major

Individual multi-word cache block

One row of 2D matrix

*Cache Line is alternative name for Cache Entry or Block

Loop Interchange: Flashcard quiz

```c
for(j=0; j < N; j++) {
    for(i=0; i < M; i++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

What kind of locality does this improve?

Spatial
Temporal
Both
Neither

Loop Fusion: Flashcard Quiz

```c
for(i=0; i < N; i++)
    a[i] = b[i] * c[i];
for(i=0; i < N; i++)
    d[i] = a[i] * c[i];
```

What kind of locality does this improve?

Spatial
Temporal
Both
Neither

Administrivia

- Lab #6: More MIPS
- Project 2b: MIPS Emulator, due Sunday
- Midterm, a week from Tuesday
Cache Blocking (aka Cache Tiling)

- “shrink” problem by performing multiple iterations within smaller cache blocks
- Also known as cache tiling
- Don’t confuse term “cache blocking” with:
  - cache blocks, i.e., individual cache entries or lines
  - (or later, blocking versus non-blocking caches)
- Use Matrix Multiply as example: Next Lab and Project 3

Matrix Multiplication

\[
\begin{align*}
\text{Matrix Multiplication} & \\
\begin{bmatrix} a & b \\ \end{bmatrix} \times \begin{bmatrix} c \\ \end{bmatrix} = \begin{bmatrix} d \\ \end{bmatrix} & \quad a \times c = d
\end{align*}
\]

Simple Matrix Multiply - www.youtube.com/watch?v=y0bTaD0hxw
100 x 100 Matrix, Cache 1000 blocks, 1 word/block

The simplest algorithm

Assumption: the matrices are stored as 2-D N x N arrays

\[
\begin{align*}
\text{The simplest algorithm} & \\
\text{for } (i=0; i<N; i++) & \\
\text{for } (j=0; j<N; j++) & \\
\text{for } (k=0; k<N; k++) & \\
\quad c[i][j] & += a[i][k] \times b[k][j];
\end{align*}
\]

Advantage: code simplicity
Disadvantage: Marches through memory and caches

Matrix Multiplication

\[
\begin{align*}
\text{Matrix Multiplication} & \\
\begin{bmatrix} a & b \\ \end{bmatrix} \times \begin{bmatrix} c \\ \end{bmatrix} = \begin{bmatrix} d \\ \end{bmatrix} & \quad a \times c = d
\end{align*}
\]

Simple Matrix Multiply - www.youtube.com/watch?v=y0bTaD0hxw
100 x 100 Matrix, Cache 1000 blocks, 1 word/block

Improving reuse via Blocking: 1st “Naïve” Matrix Multiply

\[
\begin{align*}
\text{Improving reuse via Blocking: 1st “Naïve” Matrix Multiply} & \\
\text{(implements } C = A \times B) & \\
\text{for } i = 1 \text{ to } N & \\
\text{(read row } i \text{ of } A \text{ into cache)} & \\
\text{for } j = 1 \text{ to } N & \\
\text{(read } a[i,j] \text{ into cache)} & \\
\text{for } k = 1 \text{ to } N & \\
\text{(read column } k \text{ of } B \text{ into cache)} & \\
\quad c[i,j] & += a[i,k] \times b[k,j];
\end{align*}
\]

Advantage: code simplicity
Disadvantage: Marches through memory and caches

Matrix Multiplication

\[
\begin{align*}
\text{Matrix Multiplication} & \\
\begin{bmatrix} a & b \\ \end{bmatrix} \times \begin{bmatrix} c \\ \end{bmatrix} = \begin{bmatrix} d \\ \end{bmatrix} & \quad a \times c = d
\end{align*}
\]

Simple Matrix Multiply - www.youtube.com/watch?v=y0bTaD0hxw
100 x 100 Matrix, Cache 1000 blocks, 1 word/block

Blocked Matrix Multiply

Consider A, B, C to be N-by-N matrices of b-by-b subblocks where b=n / N is called the block size

\[
\begin{align*}
\text{Blocked Matrix Multiply} & \\
\text{for } i = 1 \text{ to } N & \\
\text{(read block } C[i,j] \text{ into cache)} & \\
\text{for } k = 1 \text{ to } N & \\
\text{(read block } A[i,k] \text{ into cache)} & \\
\text{(read block } B[k,j] \text{ into cache)} & \\
\text{C[i,j] = } & C[i,j] + A[i,k] \times B[k,j] \text{ (do a matrix multiply on blocks)} & \\
\text{write block } C[i,j] \text{ back to main memory)} & \\
\end{align*}
\]

Blocked Matrix Multiply - www.youtube.com/watch?v=FWi6gG/MNh0
100 x 100 Matrix, 1000 cache blocks, 1 word/block, block 30x30
### Blocked Algorithm

- The blocked version of the i-j-k algorithm is written simply as \((A,B,C)\) are submatricies of \((a,b,c)\)

```plaintext
for (i=0;i<N/r;i++)
  for (j=0;j<N/r;j++)
    for (k=0;k<N/r;k++)
      \(C[i][j] += A[i][k] * B[k][j]\)
```

- \(r = \text{block (sub-matrix) size} \) (Assume \(r\) divides \(N\))
- \(X[i][j] = \text{a sub-matrix of } X, \text{ defined by block row } i \text{ and block column } j\)

### Maximum Block Size

- The blocking optimization works only if the blocks fit in cache.
- That is, 3 blocks of size \(r x r\) must fit in memory (for \(A, B, \text{ and } C\))
- \(M = \text{size of cache (in elements/words)}\)
- We must have: \(3r^2 = M, \text{ or } r = \sqrt{(M/3)}\)

### Sources of Cache Misses (3 C’s)

- **Compulsory** (cold start, first reference):
  - 1st access to a block, “cold” fact of life, not a lot you can do about it.
  - If running billions of instructions, compulsory misses are insignificant
- **Capacity**:
  - Cache cannot contain all blocks accessed by the program
- **Conflict** (collision):
  - Multiple memory locations mapped to the same cache location

### Flashcard Quiz: With a fixed cache capacity, what effect does a larger block size have on the 3Cs?

- **Decreases compulsory, increases conflicts**
- **Increases conflicts**

### Flashcard Quiz: With a fixed cache block size, what effect does a larger cache capacity have on the 3Cs?

- **Increases compulsory, decreases conflicts**
- **Increases conflicts, decreases capacity misses**
- **Decreases compulsory, decreases conflicts**
- **Decreases conflicts, decreases capacity misses**
...and in Conclusion

- Although caches are software-invisible, a “cache-aware” performance programmer can improve performance by large factors by changing order of memory accesses.
- Three C’s of cache misses
  - Compulsory
  - Capacity
  - Conflict