Lecture 2

Energy and Power
Managing Memory Locality
The processor-memory gap

- The result of technological trends
- Difference in processing and memory speeds growing exponentially over time
An important principle: locality

• Programs generally exhibit two forms of locality in accessing memory
  ◆ Temporal locality (time)
  ◆ Spatial locality (space)

• Often involves loops
• Opportunities for reuse

\[
\text{for } t=0 \text{ to } T-1 \\
\text{for } i = 1 \text{ to } N-2 \\
\quad u[i] = \frac{(u[i-1] + u[i+1])}{2}
\]
Memory hierarchies

• Realize reuse through a hierarchy of smaller but faster memories
• Put things in faster memory if we reuse them frequently
Stampede’s Sandy Bridge Memory Hierarchy

- `/proc/cpuinfo` summarizes the processor
  - vendor_id: GenuineIntel
  - model name: Intel®Xeon® CPU E5-2680 0 @ 2.70GHz
  - cache size: 20480 KB
  - cpu cores: 8
- processor: 0 through processor: 16
- Detailed information at
  `/sys/devices/system/cpu/cpu/*/cache/index/*/`

![Memory Hierarchy Diagram]

Latency
- Registers: 4 Cycles
- L1 Data: 12 Cycles
- L2: 26-31 Cycles
- L3 shared: ~175-350 Cycles

Bandwidth
- Registers: 40-100 GB/s
- L1 Data: 30-60 GB/s
- L2: 20-40 GB/s
- L3 shared: 4-10 GB/s

www.cac.cornell.edu/Stampede/CodeOptimization/multicore.aspx
The 3 C’s of cache misses

- Cold Start
- Capacity
- Conflict
Managing locality with loop interchange

- The success of caching depends on the ability to **re-use** previously cached data
- Data access order affects re-use
- Assume a cache with 2 entries, each 2 words wide

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

**The 3 C’s**
- Cold Start
- Capacity
- Conflict

Scott B. Baden / CSE 262 / UCSD, Wi '15
Testbed

- 2.7GHz Power PC G5 (970fx)
- Caches: 128 Byte line size
  - 512KB L2 (8-way, 12 CP hit time)
  - 32K L1 (2-way, 2 CP hit time)
- TLB: 1024 entries, 4-way
- gcc version 4.0.1
  (Apple Computer, Inc. build 5370), -O2 optimization
- Single precision floating point
The results

for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    a[i][j] += b[i][j];

for (j=0; j<N; j++)
  for (i=0; i<N; i++)
    a[i][j] += b[i][j];

<table>
<thead>
<tr>
<th>N</th>
<th>IJ (ms)</th>
<th>JI (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.007</td>
<td>0.007</td>
</tr>
<tr>
<td>128</td>
<td>0.027</td>
<td>0.083</td>
</tr>
<tr>
<td>512</td>
<td>1.1</td>
<td>37</td>
</tr>
<tr>
<td>1024</td>
<td>4.9</td>
<td>284</td>
</tr>
<tr>
<td>2048</td>
<td>18</td>
<td>2,090</td>
</tr>
</tbody>
</table>
Blocking for Cache
Matrix Multiplication

• An important core operation in many numerical algorithms

• Given two *conforming* matrices $A$ and $B$, form the matrix product $A \times B$
  - $A$ is $m \times n$
  - $B$ is $n \times p$

• Operation count: $O(n^3)$ multiply-adds for an $n \times n$ square matrix

• Discussion follows from Demmel
  
  www.cs.berkeley.edu/~demmel/cs267_Spr99/Lectures/Lect02.html
Unblocked Matrix Multiplication

\[
\text{for } i := 0 \text{ to } n-1 \\
\text{for } j := 0 \text{ to } n-1 \\
\text{for } k := 0 \text{ to } n-1 \\
C[i,j] += A[i,k] \times B[k,j]
\]
Analysis of performance

for i = 0 to n-1
  // for each iteration i, load all of B into cache
  for j = 0 to n-1
    // for each iteration (i,j), load A[i,:] into cache
    // for each iteration (i,j), load and store C[i,j]
    for k = 0 to n-1
      C[i,j] += A[i,k] * B[k,j]
Analysis of performance

for i = 0 to n-1
    // n × n² / L loads = n³/L, L=cache line size
    B[:,i]

for j = 0 to n-1
    // n² / L loads = n²/L
    A[i,:

    // n² / L loads + n² / L stores = 2n² / L
    C[i,j]

for k = 0 to n-1
    C[i,j] += A[i,k] * B[k,j]

Total: (n³ + 3n²) / L
Flops to memory ratio

Let $q = \frac{\text{# flops}}{\text{main memory reference}}$

\[ q = \frac{2n^3}{n^3 + 3n^2} \]

$\approx 2$ as $n \to \infty$
Blocking for cache: motivation

- Assume a 2 level memory hierarchy, fast and slow
- Minimum running time $= f \times \gamma$ when all data is in fastest memory, $f=$# arithmetic ops, $\gamma =$ time / flop
- Actual time
  \[
  f \times \gamma + m \times t_m = f \times \gamma \times (1 + \frac{t_m}{\gamma} \times \frac{1}{q})
  \]
  $m =$ # words transferred
  $t_m =$ slow memory access time
  $q =$ $f/m =$ computational intensity
  $\frac{t_m}{\gamma}$ Establishes machine balance
- What is the computational intensity for matrix multiply?
**Blocked Matrix Multiply**

- Divide A, B, C into \( N \times N \) sub blocks
- All 3 blocks must fit into cache
- Assume we have a good quality library to perform matrix multiplication on subblocks
- Each sub block is \( b \times b \)
  - \( b = n/N \) is called the block size
  - How do we establish \( b \)?

\[
\begin{align*}
C[i,j] & = C[i,j] + A[i,k] \ast B[k,j]
\end{align*}
\]
Blocking Mmpy for cache: operational details

- All 3 blocks must fit into cache
- If \( M_{\text{fast}} = \) size of fast memory
  \[
  3b^2 \leq M_{\text{fast}} \Rightarrow b \leq (M_{\text{fast}}/3)^{1/2}
  \]
- To run at half peak speed
  \[
  M_{\text{fast}} \geq 3b^2 = 3(t_{m}/t_{f})^2
  \]
- We change the order of arithmetic, so slightly different answers due to roundoff
- **Lower bound Theorem (Hong & Kung, 1981):**
  Any reorganization of this algorithm (that uses only associativity) is limited to
  \[
  q = O( (M_{\text{fast}})^{1/2} )
  \]
  \[
  \text{#words moved between fast and slow memory} = \Omega (n^3 / (M_{\text{fast}})^{1/2} )
  \]
Blocked Matrix Multiplication

for i = 0 to N-1
    for j = 0 to N-1
        // load each block C[i,j] into cache, once :
        n^2

        // b = n/N = block size
        for k = 0 to N-1
            // load each block A[i,k] and B[k,j] N^3 times
            // = 2N^3 × (n/N)^2 = 2Nn^2

        // write each block C[i,j] once :
        n^2

    Total: (2*N+2)*n^2
Flops to memory ratio

Let \( q = \# \text{ flops} / \text{ main memory reference} \)

\[
q = \frac{2n^3}{(2N + 2)n^2} = \frac{n}{N + 1}
\]

\( \approx n/N = b \)

as \( n \to \infty \)
The results

\[ R_\infty = 4 \times 2.33 = 9.32 \text{ Gflops} \]

\(~87\%\) of peak
Required performance programming

• Hierarchical blocking
  ◆ Multiple levels of cache and/or TLB
  ◆ Cache friendly layouts
  ◆ Register blocking (with unrolling)

• SSE intrinsics

• Autotuning
  ◆ Computer generated variants & blocking factors
  ◆ PHiPAC → ATLAS, in Matlab
  ◆ Performance models not sufficiently accurate
  ◆ Need to tune to matrix size

• See Jim Demmel’s lecture
  www.cs.berkeley.edu/~dennel/cs267_Spr12/Lectures/lecture02_memhier_jwd12.ppt
Cache interference

- Copying and cache friendly layouts

Row major

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
</tbody>
</table>

Reorganized into 2x2 blocks

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>13</td>
<td>15</td>
</tr>
</tbody>
</table>

Column of matrix is stored in red cache lines

Larry Carter
Roofline model

- Peak performance as a function of q
- Top line corresponds to theoretical limit based on clock speed
- Lower rooflines correspond to successively degraded performance as we remove optimizations