COMP 633 - Parallel Computing

Lecture 6
September 6, 2018

**SMM (1)**
Memory Hierarchies and Shared Memory
Topics

- Memory systems
  - organization
  - caches and the memory hierarchy
  - influence of the memory hierarchy on algorithms

- Shared memory systems
  - PRAM model and feasibility
  - Taxonomy of shared memory systems
    - UMA, NUMA, cc-NUMA
Anatomy of a processor ↔ memory system

- Performance parameters of Random Access Memory (RAM)
  - latency $L$
    - elapsed time from presentation of memory address to arrival of data
      - address transit time
      - memory access time $t_{\text{mem}}$
      - data transit time
  - bandwidth $W$
    - number of values (e.g. 64 bit words) delivered to processor per unit time
      - naive implementation $W < 1/L$
Processor vs. memory performance

• The memory “wall”
  – Processors compute faster than memory serves data
    • increasing imbalance $t_{\text{arith}} \ll t_{\text{mem}}$
Improving memory system performance (1)

- Decrease latency $L$ to memory
  - speed of light is a limiting factor
    - bring memory closer to processor
  - decrease memory access time by decreasing memory size $s$
    - access time $\propto s^{1/2}$ (VLSI)
  - use faster memory technology
    - DRAM (Dynamic RAM) 1 transistor per stored bit
      - high density, low power, long access time, low cost
    - SRAM (Static RAM) 6 transistors per stored bit
      - low density, high power, short access time, high cost
Improving memory system performance (1)

- Decrease effective latency using cache memory
  - low latency access to some values, high latency for others

- Example
  - 90% of references are to cache with latency $L_1$
  - 10% of references are to memory with latency $L_2$
  - Average latency is $0.9L_1 + 0.1L_2$
Improving memory system performance (2)

- **Increase bandwidth $W$**
  - multiport (parallel access) memory
    - multiple reads, multiple exclusive writes per memory cycle
      - High cost, very limited scalability

- “blocked” memory
  - memory supplies block of size $b$ containing requested word
    - supports *spatial locality* in cache access
Improving memory system performance (2)

- Increase bandwidth $W$ (contd)
  - pipeline memory requests
    - requires *independent* memory references
  - interleave memory
    - problem: memory access is limited by $t_{\text{mem}}$
    - use $m$ separate memories (or memory banks)
    - $W \sim \frac{m}{L}$ if references *distribute* over memory banks
Latency hiding

- **Amortize** latency using a pipelined interleaved memory system
  - $k$ independent references in $\Omega(L + k \cdot t_{\text{proc}})$ time
    - $O(L/k)$ amortized (expected) latency per reference

- **Where do we get independent references?**
  - out-of-order execution of independent load/store operations
    - found in most modern performance-oriented processors
    - partial latency hiding: $k \approx 2 - 10$ references outstanding

  - vector load/store operations
    - small vector units (AVX512)
      - vector length 2-8 words (Intel Xeon)
      - partial latency hiding
    - high-performance vector units (NEC SX-9, SX-ACE)
      - vector length $k = L / t_{\text{proc}}$ (128 - 256 words)
      - crossbar network to highly interleaved memory ($\approx$ 16,000 banks)
      - full latency hiding: amortized memory access at processor speed

  - multithreaded operation
    - independent execution threads with individual hardware contexts
      - partial latency hiding: 2-way hyperthreading (Intel)
      - full latency hiding: 128-way threading with high-performance memory (Cray MTA)
Implementing PRAM

• How close can we come to O(1) latency PRAM memory in practice?

– requires processor to memory network
  • latency $L = \text{sum of}$
    – twice network latency
    – memory cycle time
    – serialization time for CR, CW
  • $L$ increases with $m$, $p$
    – $L$ too large with current technology

– examples
    – logarithmic depth combining network eliminates memory contention time for CR, CW
      » $\Omega(\lg p)$ latency in network is prohibitive
Implementing PRAM – a compromise

• Using latency hiding with a high-performance memory system
  – implements $p \cdot k$ processor EREW PRAM slowed down by a factor of $k$
    • use $m \geq p \left( \frac{t_{\text{mem}}}{t_{\text{proc}}} \right)$ memory banks to match memory reference rate of $p$ processors
    • total latency $2L$ for $k = \frac{L}{t_{\text{proc}}}$ independent random references at each processor
    • $O(t_{\text{proc}})$ amortized latency per reference at each processor

  – unit latency degrades in the presence of concurrent reads/writes

  – Bottom line: doable but very expensive and only limited scaling in $p$
Memory systems summary

• Memory performance
  – Latency is limited by physics
  – Bandwidth is limited by cost

• Cache memory: low latency access to some values
  – caching frequently used values
    • rewards *temporal locality* of reference
  – caching consecutive values
    • rewards *spatial locality* of reference
  – decrease *average* latency
    • 90 fast references, 10 slow references: effective latency = 0.9L₁ + 0.1L₂

• Parallel memories
  – 100 *independent* references ≈ 100 fast references
  – relatively expensive
  – requires parallel processing
Simple uniprocessor memory hierarchy

- Each component is characterized by
  - capacity
  - block size
  - (associativity)

- Traffic between components is characterized by
  - access latency
  - transfer rate (bandwidth)

- Example:
  - IBM RS6000/320H (ca. 1991)

<table>
<thead>
<tr>
<th>Storage component</th>
<th>Latency (cycles)</th>
<th>Transfer Rate (words [8B] / cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>1,000,000</td>
<td>0.001</td>
</tr>
<tr>
<td>Main memory</td>
<td>60</td>
<td>0.1</td>
</tr>
<tr>
<td>Cache</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Registers</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
Cache operation

- ABC cache parameters
  - associativity
  - block size
  - capacity
- CCC performance model
  - cache misses can be
    - compulsory
    - capacity
    - conflict
Cache operation: read

associativity = 256-way
block size = 64B = 512b

40-bit address

Tag  Index  blk

Valid  Tag  Data
<1>  <26>  <512>

40-bit address

MUX

1,2,4,8 bytes
The changing memory hierarchy

- **IBM RS6000 320H - 25 MHz (1991)**

<table>
<thead>
<tr>
<th>Storage component</th>
<th>Latency (cycles)</th>
<th>Transfer Rate (words [8B] / cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>1,000,000</td>
<td>0.001</td>
</tr>
<tr>
<td>Main memory</td>
<td>60</td>
<td>0.1</td>
</tr>
<tr>
<td>Cache</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Registers</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Intel Xeon 61xx [per core @3GHz] (2017)**

<table>
<thead>
<tr>
<th>Storage component</th>
<th>Latency (cycles)</th>
<th>Transfer Rate (words [8B] / cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDD</td>
<td>18,000,000</td>
<td>0.00007</td>
</tr>
<tr>
<td>SSD</td>
<td>300,000</td>
<td>0.02</td>
</tr>
<tr>
<td>Main memory</td>
<td>250</td>
<td>0.2</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>48</td>
<td>0.5</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Registers</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
Computational Intensity

• Computational intensity of a problem
  \[ I = \frac{\text{(total # of arithmetic operations required)}}{\text{(size of input + size of result)}} \text{ in flops} \]
  \[ \text{in 64-bit words} \]

• BLAS - Basic Linear Algebra Subroutines
  – Asymptotic performance determined by computational intensity
    • \( A, B, C \in \mathbb{R}^{n \times n} \)
    • \( x, y \in \mathbb{R}^n \)
    • \( a \in \mathbb{R} \)

<table>
<thead>
<tr>
<th>name</th>
<th>defn</th>
<th>flops</th>
<th>refs</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</td>
<td>( y = ax )</td>
<td>( n )</td>
<td>( 2n )</td>
<td>( 0.5 )</td>
</tr>
<tr>
<td>BLAS 1 triad</td>
<td>( y = ax + y )</td>
<td>( 2n )</td>
<td>( 3n )</td>
<td>( 0.67 )</td>
</tr>
<tr>
<td>dot product</td>
<td>( x \cdot y )</td>
<td>( 2n )</td>
<td>( 2n )</td>
<td>( 1 )</td>
</tr>
<tr>
<td>BLAS 2 Matrix-vector</td>
<td>( y = y + Ax )</td>
<td>( 2n^2+n )</td>
<td>( n^2+3n )</td>
<td>( \sim 2 )</td>
</tr>
<tr>
<td>rank-1 update</td>
<td>( A = A + xy^T )</td>
<td>( 2n^2 )</td>
<td>( 2n^2+2n )</td>
<td>( \sim 1 )</td>
</tr>
<tr>
<td>BLAS 3 Matrix product</td>
<td>( C = C + AB )</td>
<td>( 2n^3 )</td>
<td>( 4n^2 )</td>
<td>( n/2 )</td>
</tr>
</tbody>
</table>
Effect of the memory hierarchy on execution time

- \( C^{N \times N} = A^{N \times N} \cdot B^{N \times N} \) naïve implementation

```latex
do \ i = 1, N 
    do \ j = 1, N 
      do \ k = 1, N 
        \( C(i, j) = C(i, j) + A(i, k) \cdot B(k, j) \)
``` 

- machine
  - simple L1 cache
    - block size = 16 words
    - capacity = 512 blocks
    - fully associative
  - main memory
    - 4K pages
- Layout of A,B,C in memory
  - Fortran: column-major order
- RAM model suggests \( O(N^3) \) run time
  - actual time follows \( O(N^5) \) growth!

Performance of naïve \( N \times N \) matrix multiply on an IBM RS6000/320 uniprocessor. Time in clock cycles per multiply-add (note \( \log_{10} \) scales). Source: Alpern et al., “The Uniform Memory Hierarchy Model of Computation”, *Algorithmica*, 1994
Shared memory taxonomy

- **Uniform Memory Access (UMA)**
  - Processors and memory separated by network
  - All memory references cross network
  - Only practical for machines with full latency hiding
    - Parallel vector processors, multi-threaded processors
    - Expensive, rarely available in practice
Shared memory taxonomy

- Non-Uniform Memory Access (NUMA)
  - Memory is partitioned across processors
  - References are local or non-local
    - Local references
      - low latency
    - Non-local references
      - high latency
    - non-local : local latency
      - large

- Examples

- Poor performance unless extreme care is taken in data placement
Combining (N)UMA with cache memories

- Processor-local caches
  - Cache all memory references
  - Must reflect changes in value due to other processors in system
  - Cache-misses
    - Usual: compulsory, capacity, and conflict misses
    - New: coherence misses

- Cache-coherent UMA examples
  - Conventional PC-based SMP systems
    - Network is a shared bus
    - Limited scaling ($p \leq 4$)
    - Changing …
  - Server-class machines
    - Single card multisocket interconnect
    - Intel Xeon E5-4600 quad socket ($p = 32$)

- Cache-coherent NUMA examples
  - scales to larger processor count
    - IBM p690 ($p \leq 64$)
    - SGI Altix 4700 ($p \leq 1024$)
Shared-memory server: Intel Xeon series

- Xeon 7500 Nehalem (2010)
- 4 sockets
- 32 cores
- 64 memory channels
- 6 bidirectional QPI channels
- ~160 GB/s total memory bandwidth
Incorporating shared memory in the hierarchy

- Non-local shared memory
  - can be viewed as additional level in processor-memory hierarchy

- Shared-memory parallel programming
  - extension of memory hierarchy techniques
  - goal:
    * concurrent transfer through parallel levels

<table>
<thead>
<tr>
<th>Storage component</th>
<th>Latency (cycles)</th>
<th>Transfer Rate (words [8B] / cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>1,000,000</td>
<td>0.001</td>
</tr>
<tr>
<td>Non-local memory</td>
<td>180 - 500</td>
<td>0.1 - 0.01</td>
</tr>
<tr>
<td>Local memory</td>
<td>60</td>
<td>0.1</td>
</tr>
<tr>
<td>Cache</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Registers</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>