18-447 Lecture 21:
Parallel Architecture Overview

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

- Your goal today
  - develop a general appreciation for parallel computer architecture
  - set the context for the focused topics to come
- Notices
  - Final Exam: Thursday, May 11, 2017, 5:30pm~8:30p.m
  - Handout #15: HW5 (on Blackboard)
  - Lab 4 status check “this week”
- Readings
  - P&H Ch 6
### Midterm 1 Statistics

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>total</th>
<th>estm</th>
</tr>
</thead>
<tbody>
<tr>
<td>points</td>
<td>10</td>
<td>6</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>9</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>mean</td>
<td>5.7</td>
<td>3.7</td>
<td>7.2</td>
<td>17</td>
<td>14.8</td>
<td>6.6</td>
<td>8.3</td>
<td>4.0</td>
<td>67.6</td>
<td>74.1</td>
</tr>
<tr>
<td>std dev.</td>
<td>4.2</td>
<td>2.4</td>
<td>2.1</td>
<td>3.0</td>
<td>7.0</td>
<td>3.9</td>
<td>2.9</td>
<td>4.0</td>
<td>15.4</td>
<td>8.8</td>
</tr>
<tr>
<td>median</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>18</td>
<td>19</td>
<td>6.5</td>
<td>10</td>
<td>5.5</td>
<td>69</td>
<td>75</td>
</tr>
<tr>
<td>mode</td>
<td>10</td>
<td>6</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>9</td>
<td>85</td>
<td>70</td>
</tr>
<tr>
<td>max</td>
<td>10</td>
<td>6</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>9</td>
<td>95</td>
<td>90</td>
</tr>
<tr>
<td>min</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>
A Non-Parallel “Architecture”

- By far the most common architectural paradigm
- Memory holds both program and data
  - instructions and data in a linear memory array
  - instructions can be modified just like data
- Sequential instruction processing
  1. program counter (PC) identifies the current instruction
  2. instruction is fetched from memory
  3. instruction execution causes some state (e.g. memory) to be updated as a specific function of current state
  4. program counter is advanced (according to instruction)
  5. repeat

* atomic
* sequential
* inorder
Parallelism Defined

- $T_1$ (call it “Work”):
  - time to complete work with 1 PE
- $T_\infty$ (call it “Critical Path”):
  - time to complete work given infinite PEs
  - $T_\infty$ lower-bounded by dataflow dependencies
- Average Parallelism:
  $$P_{avg} = \frac{T_1}{T_\infty}$$
- For a system with $p$ PEs
  $$T_p \geq \max\{ \frac{T_1}{p}, T_\infty \}$$
  $$S_p \leq \min\{ p, \frac{T_1}{T_\infty} \}$$
- When $P_{avg} >> p$
  $$T_p \approx \frac{T_1}{p}$$ and $$S_p \approx p$$

Big Picture First: Flynn’s Taxonomy

<table>
<thead>
<tr>
<th>SISD: your vanilla uniprocessor</th>
<th>MISD: ??</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMD: multiple PEs controlled by a common instruction stream/control-flow but working on data independently</td>
<td>MIMD: fully independent programs/control-flows working in parallel</td>
</tr>
</tbody>
</table>
**Data Parallelism**

- Data Parallelism is abundant in many matrix operations and “scientific” kernels in general
- Example: DAXPY/LINPACK (inner loop of matrix-mult)
  ```c
  double Y[N], X[N];
  for (i=0; i<n; i++) {
      Y[i]=a*X[i]+Y[i]
  }
  ```
  - Y and X are vectors
  - same operations repeated on each \( Y[i] \) and \( X[i] \)
  - no data dependence across iterations
- There are programming languages and computing platforms that support “vectors” as native data types
  \[
  Y = a \times X + Y
  \]
Data Parallelism ≠ Data Parallel Execution

- Short Vector: better performance by parallel operations
  
  \[ C = A + B \]

- Long Vector: better performance by deep pipelining (no inter-dependence between operations)

\[
\begin{array}{cccccccc}
  a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\
  b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
  c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
  a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\
  b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \\
\end{array}
\]

SIMD ≠ Vector/Matrix

- Parallel Bubble Sort (assume one number per PE)

Repeat

if (even node)
  if (my num>right neighbor’s num)
    switch them
else // odd node
  do nothing

if (odd node)
  if (my num>right neighbor’s num)
    switch them
else // even node
  do nothing

If-then-else implemented as predicated operations (see Slide L10-26)
MIMD ≠ Concurrent Execution Either

- Multithreading Pipelines [e.g., HEP, Smith]
  - each cycle, select a “ready” thread from scheduling pool
  - only one instruction per context in flight at once
  - on a long latency stall, remove the context from scheduling
  - simpler and faster pipeline implementation since
    - no data dependence, hence no stall or forwarding
    - no penalty in making pipeline deeper

A Spotty Tour of the MP Universe
Variety in the details

- E.g., Work distribution
  - granularity of parallelism (how finely is the work divided), ranging from whole programs down to bits
  - regularity (do all of “nodes” look about the same and look out to about the same environment)
  - static vs. dynamic (load-balancing)

- E.g., Communication
  - message-passing vs. shared memory
  - granularity of communication (words to pages)
  - interconnect and interface design/performance
    - topology, overhead vs. latency, synchronization

- Most real implementations combine multiple programming and implementation paradigms

SIMD: Big-Irons

- Sea of PEs on a regular network grid
  - synchronized common control
  - direct access to local mem
  - nearest-neighbor exchanges (e.g., NEWS)
  - special broadcast/reduction, etc.

- E.g., Thinking Machines CM-2
  - thousand of bit-sliced PEs lock-step controlled by a common sequencer
  - “hypercube” topology
  - special external I/O nodes
SIMD: Vector Machines

- Vector registers (e.g., Cray 1 had 64 registers by 64 words by 64 bits)
- Deeply pipelined vector functional units
- Equally high-performance load-store units and multi-banked memory
- E.g., Cray 1, circa 1976
  - 12 pipelines, 12.5ns
  - built from ECL 4-input NAND and SRAM chips
  - no caches!!
  - 2x25ton compressor for cooling
  - 250MIPS peak for ~10M 1970$

SIMD: Modern Renditions

- E.g., Intel SSE (Streaming SIMD Extension) circa 1999
  - 16 new 128-bit “vector” registers, each for 4 floats or 2 doubles
  - new SIMD instructions: load/store, arithmetic, shuffle, bit-wise
  - SSE4 with true full-width operations—Core i7 can do 4 sp-mult & 4 sp-add per cyc per core, (24GFLOPS@3GHz)
    Latest AVX 2 doubles the above (over 1TFLOPS/chip)

- E.g., NVIDIA GeForce 8800GTX (2008)
  - 16 processors per chip
  - 8 multithreaded pipelines per processor supporting 32 SIMD threads (345.6GFLOPs peak per processor)
  - allows diverging threads by following multiple diverging paths with a subset of threads at a time

Latest GTX1080 peaks at over 8 TFLOPS
ASCII Red, 1996
what a TeraFLOP meant 20 years ago

- First computer to exceed 1 teraflop
- A $50M, 1600ft² system
  - ~10K 200MHz PentiumPro’s
  - ~1 TByte DRAM
  - 500kW to power the computer
  - 500kW to cool the computer center
- Advanced Simulation and Computing Program (ASCI)
  - how do you know if the warheads are still good if you can’t blow one up to find out?
  - require ever more expensive simulation as stockpile aged, Red 1.3TFLOPS 1996; Blue Mountain/Pacific 4TFLOPS 1998; White 12TFLOPS 2000; Purple 100TFLOPS 2005; ... Cray Titan 27PFLOPS present day

MIMD Shared Memory:
Symmetric Multiprocessors (SMPs)

- Symmetric means
  - identical processors connected to common shared memory
  - all processors have equal access to system (memory and I/O)
  - OS can schedule any process on any processor
- Uniform Memory Access (UMA)
  - processor/memory connected by bus or crossbar
  - all processors have equal memory access performance
  - “snoop”-based cache coherence needed
    (a later lecture on this)
MIMD Shared Memory: Most Common Form of Multiprocessors Today

- General-purpose multicore processors implement the SMP paradigm on a single chip
- Latest Intel Xeon has up to 24 cores per socket

![Intel Xeon e5345](figure)

MIMD Shared Memory: Big Irons

- Distributed Shared Memory
  - True UMA very expensive to scale due to concentration of bandwidth
  - Large scale SMPs have distributed memory with non-uniform memory (NUMA)
    - “local” memory pages (faster to access)
    - “remote” memory pages (slower to access)
    - cache-coherent still possible (but requires complicated distributed, message-based protocols (another lecture)
  - E.g., SGI Origin 2000
    - upto 512 CPUs and 512GB DRAM ($40M)
    - 48 128-CPU system was collectively the 2nd fastest computer (3TFLOPS) in 1999

![Interconnection Network](figure)
MIMD Message Passing: by network interface unit (NIU) placement

- Beowulf Clusters (I/O bus)
  - basic Linux PCs connected by standard Ethernet
  - supports MPI message passing library
- High-Performance Clusters (I/O bus)
  - still stock workstations/servers but more exotic interconnects
  - e.g., Myrinet, HIPPI, Infiniband, etc.
- Supers (memory bus)
  - stock CPUs on custom platform
  - e.g., Cray XT5 (2011 era “world’s fastest computer” with 224,162 Opteron processors)
- Inside the CPU
  - single-instruction send/receive
  - e.g., iAPX 432 (1981), Transputers (80s)

The difference is in the cost of communication

- Processors may have to wait for data because communication is not instantaneous
  - latency of network transit time and delay through intervening mechanisms
  - time to convey payload over finite link bandwidth
  - a function of network implementation

  What if the data needed is not computed yet?

- Processors must do something to send/receive a message because communication is not spontaneous
  - send/receive and protocol instructions
  - stall time due to blocking access to the NIU (e.g., mmap’ed read, or write-buffer stalls)
  - a function of NIU design and placement
- Shared-memory programs also worry about the same!
Implications of Comm. Performance

- **Low Bandwidth**
  - cannot communicate a large amount of data
  - must have a lot of work to do per byte communicated
  - only scalable for applications with high “arithmetic intensity”
- **High Overhead**
  - cannot communicate frequently
  - can only exploit coarse-grain parallelism
  - If interface is DMA, the size of the communication is not necessarily limited
- **High Latency**
  - producer cannot send data at the last minute
  - must have high average parallelism (more work/time between production and use of data)
- **Low performance in each category limits**
  - the kind of applications that can speed up, and
  - how much they can speed up

---

Can’t Forget: Dataflow Architectures

- Reduce a program to its dataflow graph
  - no control flow, no program counter
  - any operation with all inputs available can “fire”
  - in principle can achieve the finest-grain, highest degree parallelism available
- Developments
  - researched since the 70’s with many working prototypes, e.g., Sigma-1, EM4, Monsoon
  - most recent efforts: TRIPS and Wavescalar
  - modern superscalar OOO CPUs based on the same principle (not necessarily directly influenced)
- The hard problems
  - the notion of memory
  - resource management
  - language and compiler

[Figure and example from Arvind]
# Top500 – Nov 2016

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>System</th>
<th>Nodes</th>
<th>Speed (TFlops)</th>
<th>Rating (Efficial)</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>National Supercomputing Center in Nanjing, China</td>
<td>Sunway TaihuLight</td>
<td>13,048</td>
<td>93,514.6</td>
<td>125,146.9</td>
<td>15,970</td>
</tr>
<tr>
<td>2</td>
<td>National Super Computer Center in Eugene, Oregon, USA</td>
<td>Titan/BlueGene/Q</td>
<td>3,120,000</td>
<td>3,116.2</td>
<td>54,102.4</td>
<td>17,680</td>
</tr>
<tr>
<td>3</td>
<td>Oak Ridge National Laboratory, USA</td>
<td>Titan</td>
<td>566,640</td>
<td>17,997.0</td>
<td>21,518.9</td>
<td>6,229</td>
</tr>
<tr>
<td>4</td>
<td>Oak Ridge National Laboratory, USA</td>
<td>Sequoia</td>
<td>1,972,864</td>
<td>17,773.2</td>
<td>20,132.7</td>
<td>7,895</td>
</tr>
<tr>
<td>5</td>
<td>Oak Ridge National Laboratory, USA</td>
<td>Oahe</td>
<td>422,832</td>
<td>1618.7</td>
<td>21,861.7</td>
<td>3,838</td>
</tr>
<tr>
<td>6</td>
<td>Joint Center for Advanced High Performance Computing, Japan</td>
<td>Oakforest-PACS</td>
<td>156,124</td>
<td>12,365.6</td>
<td>24,172.5</td>
<td>2,791</td>
</tr>
<tr>
<td>7</td>
<td>Swiss National Supercomputing Center, Switzerland</td>
<td>K computer</td>
<td>705,024</td>
<td>10,510.0</td>
<td>11,280.4</td>
<td>12,640</td>
</tr>
<tr>
<td>8</td>
<td>Tokyo Institute of Technology, Japan</td>
<td>Fugaku</td>
<td>236,720</td>
<td>9,773.0</td>
<td>18,980.0</td>
<td>1,312</td>
</tr>
<tr>
<td>9</td>
<td>TACC</td>
<td>786,432</td>
<td>0.9366</td>
<td>16,546.3</td>
<td>3,748</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Joint Center for Advanced High Performance Computing, Japan</td>
<td>Fugaku</td>
<td>337,558</td>
<td>0.1389</td>
<td>11,979.0</td>
<td>4,203</td>
</tr>
</tbody>
</table>


---

# Top500: Architecture

[Architecture Performance Chart](http://www.top500.org)

Screenshot: [http://www.top500.org](http://www.top500.org)
Top500: Application Area

[Image: Application Area - Performance Share]

Screenshot: http://www.top500.org

Top500: Moore’s Law?

[Image: Projected Performance Development]

Screenshot: http://www.top500.org