We used to joke that “parallel computing is the future, and always will be,” but the pessimists have been proven wrong.”

— Tony Hey
Course Information

- **Lecture Time**: TuTh 10:00 am - 11:20 am
- **Location**: Melville Library N4006, West Campus
- **Instructor**: Rezaul A. Chowdhury
- **Office Hours**: TuTh 12:30 pm - 2:00 pm, 1421 Computer Science
- **Email**: rezaul@cs.stonybrook.edu
- **TA**: Unlikely!

- **Class Webpage**: 
Prerequisites

- **Required:** Background in algorithms analysis (e.g., CSE 373 or CSE 548)
- **Required:** Background in programming languages (C/C++)
- **Helpful but Not Required:** Background in computer architecture

- **Please Note:** This is not a course on
  - Programming languages
  - Computer architecture
- **Main Emphasis:** Parallel algorithms
The following topics will be covered

- Analytical modeling of parallel programs
- Scheduling
- Programming using the message-passing paradigm and for shared address-space platforms
- Parallel algorithms for dense matrix operations, sorting, searching, graphs, computational geometry, and dynamic programming
- Concurrent data structures
- Transactional memory, etc.
Grading Policy

- Homeworks (three: lowest score 5%, others 10% each): 25%
- Exams (two: higher one 20%, lower one 10%): 30%
  - Midterm (in-class): Oct 15
  - Final (in-class): Dec 5
- Group project (one): 30%
  - Proposal: Sep 19
  - Progress report (in-class): Oct 22 - 24
  - Final report: Dec 6
- Scribe note (one lecture): 10%
- Class participation & attendance: 5%
Programming Environment

This course is supported by an educational grant from

- Extreme Science and Engineering Discovery Environment (XSEDE): https://www.xsede.org

We have access to the following supercomputers

- **Stampede (Texas Advanced Comp. Center)**: 6,400 nodes; 16 cores (2 Intel Sandy Bridge) and 1/2 Intel Xeon Phi coprocessor(s) per node

- **Lonestar (Texas Advanced Comp. Center)**: 1,800+ nodes; 12 cores (2 Intel Westmere processors) per node

- **Trestles (San Diego Supercomputer Center)**: 300+ nodes; 32 cores (4 AMD Magny Cours processors) per node

- **Kraken (National Institute for Comp. Sciences)**: 9,000+ nodes; 12 cores (2 AMD Opteron Istanbul processors) per node

- **Keeneland KIDS (Georgia Tech)**: 120 nodes; 16 cores (2 Intel Sandy Bridge processors) and 3 NVIDIA Fermi GPUs per node
# World's Most Powerful Supercomputers in June, 2013

(www.top500.org)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>System</th>
<th>Coros</th>
<th>Rmax  (TFlop/s)</th>
<th>Rpeak (TFlop/s)</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>National University of Defense Technology/China</td>
<td>Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.20GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT</td>
<td>3120000</td>
<td>33862.7</td>
<td>54902.4</td>
<td>17808</td>
</tr>
<tr>
<td>2</td>
<td>DOE/SC/Oak Ridge National Laboratory United States</td>
<td>Titan - Cray XK7, Opteron 6274 16C 2.20GHz, Cray Gamini interconnect, NVIDIA K20x Cray Inc.</td>
<td>560840</td>
<td>17590.0</td>
<td>27112.5</td>
<td>8209</td>
</tr>
<tr>
<td>3</td>
<td>DOE/NNSA/LLNL United States</td>
<td>Sequoia - BlueGene/Q. Power BQC 16C 1.60GHz, Custom IBM</td>
<td>1572864</td>
<td>17173.2</td>
<td>20132.7</td>
<td>7990</td>
</tr>
<tr>
<td>4</td>
<td>RIKEN Advanced Institute for Computational Science (AICS) Japan</td>
<td>K computer, SPARC64 VIIIFx 2.0GHz, Tofu interconnect Fujitsu</td>
<td>705024</td>
<td>10510.0</td>
<td>11280.4</td>
<td>12660</td>
</tr>
<tr>
<td>5</td>
<td>DOE/SC/Argonne National Laboratory United States</td>
<td>Mira - BlueGene/Q. Power BQC 16C 1.60GHz, Custom IBM</td>
<td>786432</td>
<td>8586.6</td>
<td>10066.3</td>
<td>3945</td>
</tr>
<tr>
<td>6</td>
<td>Texas Advanced Computing Center/Univ. of Texas United States</td>
<td>Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.70GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell</td>
<td>462462</td>
<td>5168.1</td>
<td>8520.1</td>
<td>4510</td>
</tr>
<tr>
<td>7</td>
<td>Forschungszentrum Juelich (FZJ) Germany</td>
<td>JUQUEEN - BlueGene/Q. Power BQC 16C 1.60GHz, Custom Interconnect IBM</td>
<td>458752</td>
<td>5008.9</td>
<td>5872.0</td>
<td>2301</td>
</tr>
<tr>
<td>8</td>
<td>DOE/NNSA/LLNL United States</td>
<td>Vulcan - BlueGene/Q, Power BQC 16C 1.60GHz, Custom Interconnect IBM</td>
<td>393216</td>
<td>4293.3</td>
<td>5033.2</td>
<td>1972</td>
</tr>
<tr>
<td>9</td>
<td>Leibniz Supercomputing Centre in Munich Germany</td>
<td>SuperMUC - dDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM</td>
<td>147456</td>
<td>2897.0</td>
<td>3185.1</td>
<td>3423</td>
</tr>
<tr>
<td>10</td>
<td>National Supercomputing Center in Tianjin China</td>
<td>Tianhe-1A - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 NUDT</td>
<td>186368</td>
<td>2566.0</td>
<td>4701.0</td>
<td>4040</td>
</tr>
</tbody>
</table>
Recommended Textbooks


Why Parallelism?
Moore's Law

The diagram illustrates the exponential increase in transistor count over time, showing that the number of transistors on a microchip doubles approximately every two years. This trend is evident from the 1970s to the early 2010s, with specific points marking the introduction dates of various microprocessors and their corresponding transistor counts.

Source: Chung-Ta King, Department of Computer Science, National Tsing Hua University
Unicore Performance Has Hit a Wall!

Some Reasons

- Lack of additional ILP
  (Instruction Level Hidden Parallelism)
- High power density
- Manufacturing issues
- Physical limits
- Memory speed
Unicore Performance: No Additional ILP

“Everything that can be invented has been invented.”
— Charles H. Duell
Commissioner, U.S. patent office, 1899

Exhausted all ideas to exploit hidden parallelism?

— Multiple simultaneous instructions
— Dynamic instruction scheduling
— Branch prediction
— Out-of-order instructions
— Speculative execution
— Pipelining
— Non-blocking caches, etc.
Unicore Performance: High Power Density

- Dynamic power, \( P_d \propto V^2 f C \)
  
  \[ \begin{align*}
  V &= \text{supply voltage} \\
  f &= \text{clock frequency} \\
  C &= \text{capacitance}
  \end{align*} \]

- But \( V \propto f \)

- Thus \( P_d \propto f^3 \)

*Source: Patrick Gelsinger, Intel Developer Forum, Spring 2004 (Simon Floyd)*
Unicore Performance: Manufacturing Issues

- Frequency, \( f \propto \frac{1}{s} \)
  - \( s = \text{feature size (transistor dimension)} \)
- Transistors / unit area \( \propto \frac{1}{s^2} \)
- Typically, die size \( \propto \frac{1}{s} \)
- So, what happens if feature size goes down by a factor of \( x \)?
  - Raw computing power goes up by a factor of \( x^4 \)!
  - Typically most programs run faster by a factor of \( x^3 \) without any change!

Source: Kathy Yelick and Jim Demmel, UC Berkeley
Unicore Performance: Manufacturing Issues

- Manufacturing cost goes up as feature size decreases
  - Cost of a semiconductor fabrication plant doubles every 4 years (Rock’s Law)
- CMOS feature size is limited to 5 nm (at least 10 atoms)

Source: Kathy Yelick and Jim Demmel, UC Berkeley
Unicore Performance: Physical Limits

Execute the following loop on a serial machine in 1 second:

\[
\text{for ( } i = 0; i < 10^{12}; ++i \text{ )} \\
z[i] = x[i] + y[i];
\]

- We will have to access \(3 \times 10^{12}\) data items in one second
- Speed of light is, \(c \approx 3 \times 10^8\) m/s
- So each data item must be within \(c / 3 \times 10^{12} \approx 0.1\) mm from the CPU on the average
- All data must be put inside a \(0.2\) mm \(\times\) \(0.2\) mm square
- Each data item (\(\geq 8\) bytes) can occupy only \(1\) Å\(^2\) space! (size of a small atom!)

Source: Kathy Yelick and Jim Demmel, UC Berkeley
Unicore Performance: Memory Wall

Source: Rick Hetherington, Chief Technology Officer, Microelectronics, Sun Microsystems
Unicore Performance Has Hit a Wall!

Some Reasons

― Lack of additional ILP
   (Instruction Level Hidden Parallelism)
― High power density
― Manufacturing issues
― Physical limits
― Memory speed

“Oh Sinnerman, where you gonna run to?”

― Sinnerman (recorded by Nina Simone)
Where You Gonna Run To?

- Changing $f$ by 20% changes performance by 13%
- So what happens if we overclock by 20%?
- And underclock by 20%?

Source: Andrew A. Chien, Vice President of Research, Intel Corporation
Where You Gonna Run To?

— Changing $f$ by 20% changes performance by 13%
— So what happens if we overclock by 20%?
— And underclock by 20%?

Source: Andrew A. Chien, Vice President of Research, Intel Corporation
Where You Gonna Run To?

- Changing $f$ by 20% changes performance by 13%
- So what happens if we overclock by 20%?
- And underclock by 20%?

Source: Andrew A. Chien, Vice President of Research, Intel Corporation
Moore’s Law Reinterpreted

Source: Report of the 2011 Workshop on Exascale Programming Challenges
Cores / Processor (General Purpose)

Source: Andrew A. Chien, Vice President of Research, Intel Corporation
No Free Lunch for Traditional Software

Source: Simon Floyd, Workstation Performance: Tomorrow's Possibilities (Viewpoint Column)
Top 500 Supercomputing Sites (Cores / Socket)

Source: www.top500.org
Insatiable Demand for Performance

Numerical Weather Prediction

Problem: \(( \text{temperature, pressure, ..., humidity, wind velocity} ) \leftarrow f( \text{longitude, latitude, height, time} )\)

Approach (very coarse resolution):

- Consider only modeling fluid flow in the atmosphere
- Divide the entire global atmosphere into cubic cells of size 1 mile \(\times\) 1 mile \(\times\) 1 mile each to a height of 10 miles
  \(\approx 2 \times 10^9\) cells
- Simulate 7 days in 1 minute intervals
  \(\approx 10^4\) time-steps to simulate
- 200 floating point operations (flop) / cell / time-step
  \(\approx 4 \times 10^{15}\) floating point operations in total
- To predict in 1 hour \(\approx 1\) Tflop/s (Tera flop/sec)
Some Useful Classifications of Parallel Computers
Parallel Computer Memory Architecture (Shared Memory)

- All processors access all memory as global address space
- Changes in memory by one processor are visible to all others
- Two types:
  - Uniform Memory Access (UMA)
  - Non-Uniform Memory Access (NUMA)

Source: Blaise Barney, LLNL
Parallel Computer Memory Architecture (Shared Memory)

Advantages

- User-friendly programming perspective to memory
- Fast data sharing

Disadvantages

- Difficult and expensive to scale
- Correct data access is user responsibility

Source: Blaise Barney, LLNL
Parallel Computer Memory Architecture (Distributed Memory)

- Each processor has its own local memory — no global address space
- Changes in local memory by one processor have no effect on memory of other processors
- Communication network to connect inter-processor memory

Source: Blaise Barney, LLNL
Parallel Computer Memory Architecture (Distributed Memory)

Advantages

- Easily scalable
- No cache-coherency needed among processors
- Cost-effective

Disadvantages

- Communication is user responsibility
- Non-uniform memory access
- May be difficult to map shared-memory data structures to this type of memory organization

Source: Blaise Barney, LLNL
Parallel Computer Memory Architecture (Hybrid Distributed-Shared Memory)

- The share-memory component can be a cache-coherent SMP or a Graphics Processing Unit (GPU).
- The distributed-memory component is the networking of multiple SMP/GPU machines.
- Most common architecture for the largest and fastest computers in the world today.

Source: Blaise Barney, LLNL
Flynn’s classical taxonomy (1966):

Classification of multi-processor computer architectures along two independent dimensions of *instruction* and *data*.

<table>
<thead>
<tr>
<th></th>
<th>Single Data (SD)</th>
<th>Multiple Data (MD)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single Instruction (SI)</strong></td>
<td>SISD</td>
<td>SIMD</td>
</tr>
<tr>
<td><strong>Multiple Instruction (MI)</strong></td>
<td>MISD</td>
<td>MIMD</td>
</tr>
</tbody>
</table>
Flynn’s Taxonomy of Parallel Computers

SISD

- A serial (non-parallel) computer
- The oldest and the most common type of computers
- Example: Uniprocessor unicore machines

Source: Blaise Barney, LLNL
Flynn’s Taxonomy of Parallel Computers

SIMD

- A type of parallel computer
- All PU’s run the same instruction at any given clock cycle
- Each PU can act on a different data item
- Synchronous (lockstep) execution
- Two types: processor arrays and vector pipelines
- Example: GPUs (Graphics Processing Units)

Source: Blaise Barney, LLNL
Flynn’s Taxonomy of Parallel Computers

MISD

- A type of parallel computer
- Very few ever existed

MIMD

- A type of parallel computer
- Synchronous /asynchronous execution
- Examples: most modern supercomputers, parallel computing clusters, multicore PCs

Source: Blaise Barney, LLNL
Parallel Algorithms
Warm-up

“The way the processor industry is going, is to add more and more cores, but nobody knows how to program those things. I mean, two, yeah; four, not really; eight, forget it.”

— Steve Jobs, NYTimes interview, June 10 2008
Consider the following loop:

\[
\text{for } i = 1 \text{ to } n \text{ do } \quad C[i] \leftarrow A[i] \times B[i]
\]

- Suppose you have an infinite number of processors/cores
- Ignore all overheads due to scheduling, memory accesses, communication, etc.
- Suppose each operation takes a constant amount of time
- How long will this loop take to complete execution?
Consider the following loop:

\[
\text{for } i = 1 \text{ to } n \text{ do} \\
C[i] \leftarrow A[i] \times B[i]
\]

- Suppose you have an infinite number of processors/cores
- Ignore all overheads due to scheduling, memory accesses, communication, etc.
- Suppose each operation takes a constant amount of time
- How long will this loop take to complete execution?
  - \(O(1)\) time
Now consider the following loop:

\[
c \leftarrow 0
\]

\[
\text{for } i = 1 \text{ to } n \text{ do }
\]

\[
c \leftarrow c + A[i] \times B[i]
\]

— How long will this loop take to complete execution?
Parallel Algorithms Warm-up (2)

Now consider the following loop:

\[
\begin{align*}
  c &\leftarrow 0 \\
  \text{for } i = 1 \text{ to } n \text{ do} \\
  &\quad c \leftarrow c + A[i] \times B[i]
\end{align*}
\]

— How long will this loop take to complete execution?

— \( O(\log n) \) time
Now consider quicksort:

\[
QSort(\ A\ )
\]

\[
if \ |A| \leq 1 \ return \ A
\]

\[
else \ p \leftarrow A[ \ rand(\ |A| ) ]
\]

\[
return \ QSort(\ \{ x \in A: x < p \} )
\]

\[
# \{ p \} #
\]

\[
QSort(\ \{ x \in A: x > p \} )
\]

— Assuming that \( A \) is split in the middle everytime, and the two recursive calls can be made in parallel, how long will this algorithm take?
Now consider quicksort:

\[
\text{QSort}( A )
\]

\[
\text{if } |A| \leq 1 \text{ return } A
\]

\[
\text{else } p \leftarrow A[ \text{rand}( |A| ) ]
\]

\[
\text{return } \text{QSort}( \{ x \in A : x < p \} )
\]

\[
\# \{ p \} #
\]

\[
\text{QSort}( \{ x \in A : x > p \} )
\]

— Assuming that \( A \) is split in the middle everytime, and the two recursive calls can be made in parallel, how long will this algorithm take?

— \( O( \log^2 n ) \) ( if partitioning takes logarithmic time )

— \( O( \log n ) \) ( but can be partitioned in constant time )