Course Information

- Class: Mon/Wed 6:40-8pm, CoRE 538
- Instructor: Dr. Ivan Rodero
- Class website: http://www.ece.rutgers.edu/~irodero
- Office hours: Mon/Wed 4-5pm, CoRE 510
- Grading policy
  - Undergraduate (ECE 451)
    • Programming Assignments: 30%, Midterm: 40%, Final Project (Groups of 3 Students): 30%, Quizzes/Class Participation: 10%
  - Graduate (ECE 566)
    • Programming Assignments: 30%, Midterms: 40%, Final Project (Individual): 30%, Quizzes/Class Participation: 10%
Objectives

• Introduction to parallel/distributed programming
  – parallel/distributed programming
  • Approaches, paradigms, tools, etc.
  – issues and approaches in parallel/distributed application development
  – current state of parallel/distributed architectures and systems
  – current and future trends

Why we need powerful computers
Demand for Computational Speed

- Continual demand for greater computational speed from a computer system than is currently possible
- Areas requiring great computational speed include numerical modeling and simulation of scientific and engineering problems.
- Computations must be completed within a "reasonable" time period.

Simulation: The Third Pillar of Science

- Traditional scientific and engineering paradigm:
  - Do theory or paper design.
  - Perform experiments or build system.
- Limitations:
  - Too difficult -- build large wind tunnels.
  - Too expensive -- build a throw-away passenger jet.
  - Too slow -- wait for climate or galactic evolution.
  - Too dangerous -- weapons, drug design, climate experimentation.
- Computational science paradigm:
  - Use high performance computer systems to simulate the phenomenon
    - Base on known physical laws and efficient numerical methods.
Some Particularly Challenging Computations

- **Science**
  - Global climate modeling
  - Astrophysical modeling
  - Biology: Genome analysis; protein folding (drug design)

- **Engineering**
  - Crash simulation
  - Semiconductor design
  - Earthquake and structural modeling

- **Business**
  - Financial and economic modeling
  - Transaction processing, web services and search engines

- **Defense**
  - Nuclear weapons -- test by simulations
  - Cryptography

Units of Measure in HPC

- High Performance Computing (HPC) units are:
  - Flop/s: floating point operations
  - Bytes: size of data

- Typical sizes are millions, billions, trillions...

<table>
<thead>
<tr>
<th></th>
<th>Mflop/s = 10^6 flop/sec</th>
<th>Mbyte = 10^6 byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mega</td>
<td></td>
<td>(also 2^20 = 1048576)</td>
</tr>
<tr>
<td>Giga</td>
<td>Gflop/s = 10^9 flop/sec</td>
<td>Gbyte = 10^9 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(also 2^30 = 1073741824)</td>
</tr>
<tr>
<td>Tera</td>
<td>Tflop/s = 10^{12} flop/sec</td>
<td>Tbyte = 10^{12} byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(also 2^{40} = 10995211627776)</td>
</tr>
<tr>
<td>Peta</td>
<td>Pflop/s = 10^{15} flop/sec</td>
<td>Pbyte = 10^{15} byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(also 2^{50} = 1125899906842624)</td>
</tr>
<tr>
<td>Exa</td>
<td>Eflop/s = 10^{18} flop/sec</td>
<td>Ebyte = 10^{18} byte</td>
</tr>
</tbody>
</table>
Economic Impact of HPC

- Airlines:
  - System-wide logistics optimization systems on parallel systems.
  - Savings: approx. $100 million per airline per year.
- Automotive design:
  - Major automotive companies use large systems (500+ CPUs) for:
    - CAD-CAM, crash testing, structural integrity and aerodynamics.
    - One company has 500+ CPU parallel system.
  - Savings: approx. $1 billion per company per year.
- Semiconductor industry:
  - Semiconductor firms use large systems (500+ CPUs) for
    - device electronics simulation and logic validation
  - Savings: approx. $1 billion per company per year.
- Securities industry:
  - Savings: approx. $15 billion per year for U.S. home mortgages.

Global Climate Modeling Problem

- Problem is to compute:
  \[ f(\text{latitude, longitude, elevation, time}) \rightarrow \text{temperature, pressure, humidity, wind velocity} \]
- Approach:
  - Discretize the domain, e.g., a measurement point every 1km
  - Devise an algorithm to predict weather at time \( t+1 \) given \( t \)

- Uses:
  - Predict major events, e.g., El Nino
  - Use in setting air emissions standards

Source: http://www.epm.ornl.gov/chammp/chammp.html
Global Climate Modeling Computation

- One piece is modeling the fluid flow in the atmosphere
  - Solve Navier-Stokes problem
    - Roughly 100 Flops per grid point with 1 minute timestep
- Computational requirements:
  - To match real-time, need 5x 1011 flops in 60 seconds = 8 Gflop/s
  - Weather prediction (7 days in 24 hours) \( \rightarrow \) 56 Gflop/s
  - Climate prediction (50 years in 30 days) \( \rightarrow \) 4.8 Tflop/s
  - To use in policy negotiations (50 years in 12 hours) \( \rightarrow \) 288 Tflop/s
- To double the grid resolution, computation is at least 8x
- Current models are coarser than this

Heart Simulation

- Problem is to compute blood flow in the heart
- Approach:
  - Modeled as an elastic structure in an incompressible fluid.
    - The “immersed boundary method” due to Peskin and McQueen.
    - 20 years of development in model
    - Many applications other than the heart: blood clotting, inner ear, paper making, embryo growth, and others
  - Use a regularly spaced mesh (set of points) for evaluating the fluid
- Uses
  - Current model can be used to design artificial heart valves
  - Can help in understand effects of disease (leaky valves)
  - Related projects look at the behavior of the heart during a heart attack
  - Ultimately: real-time clinical work
Heart Simulation Calculation

The involves solving Navier-Stokes equations
   –64^3 was possible on Cray YMP, but 128^3 required for accurate model
     (would have taken 3 years).
   –Done on a Cray C90 -- 100x faster and 100x more memory
   –Until recently, limited to vector machines

- Needs more features:
  - Electrical model of the heart, and details of muscles, e.g.,
    - Chris Johnson
    - Andrew McCulloch
  - Lungs, circulatory systems

Parallel Computing in Web Search

- Functional parallelism: crawling, indexing, sorting
- Parallelism between queries: multiple users
- Finding information amidst junk
- Preprocessing of the web data set to help find information

- General themes of sifting through large, unstructured data sets:
  - when to put white socks on sale
  - what advertisements should you receive
  - finding medical problems in a community
Document Retrieval Computation

- Approach:
  - Store the documents in a large (sparse) matrix
  - Use Latent Semantic Indexing (LSI), or related algorithms to “partition”
  - Needs large sparse matrix-vector multiply

- Matrix is compressed
- “Random” memory access
- Scatter/gather vs. cache miss per 2Flops

Ten million documents in typical matrix.
Web storage increasing 2x every 5 months.
Similar ideas may apply to image retrieval.

Transaction Processing

- Parallelism is natural in relational operators: select, join, etc.
- Many difficult issues: data partitioning, locking, threading.
Sequential Architecture Limitations

- Sequential architectures reaching physical limitation (speed of light, thermodynamics)
- Hardware improvements like pipelining, Superscalar, etc., are non-scalable and requires sophisticated Compiler Technology.
- Vector Processing works well for certain kind of problems.

How to Run Applications Faster?

- There are 3 ways to improve performance:
  - 1. Work Harder
  - 2. Work Smarter
  - 3. Get Help
- Computer Analogy
  - 1. Use faster hardware: e.g. reduce the time per instruction (clock cycle).
  - 2. Optimized algorithms and techniques, customized hardware
  - 3. Multiple computers to solve problem: That is, increase no. of instructions executed per clock cycle.
Parallel Computing

• Using more than one computer, or a computer with more than one processor, to solve a problem.

• High performance
  – greater than the fastest uniprocessor - very simple idea - that \( n \) computers operating simultaneously can achieve the result \( n \) times faster - it will not be \( n \) times faster for various reasons

• Increased “quality”
  – availability, accessibility, serviceability

• Price-performance

• Inherently parallel applications

• Fault tolerance, redundancy, larger amount of memory available, ….

Why powerful computers are parallel
Technology Trends: Microprocessor Capacity

2X transistors/Chip Every 1.5 years
Called “Moore’s Law”

Microprocessors have become smaller, denser, and more powerful.

Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

What are Scalable Systems?

- Wikipedia Definition:
  - A scalable system is one that can easily be altered to accommodate changes in the number of users, resources and computing entities affected to it. Scalability can be measured in three different dimensions:

  - Load scalability
    - A distributed system should make it easy for us to expand and contract its resource pool to accommodate heavier or lighter loads.

  - Geographic scalability
    - A geographically scalable system is one that maintains its usefulness and usability, regardless of how far apart its users or resources are.

  - Administrative scalability
    - No matter how many different organizations need to share a single distributed system, it should still be easy to use and manage.

- Some loss of performance may occur in a system that allows itself to scale in one or more of these dimensions.
Dimensions of Scalability

- Traditional High Performance Computing
- Grassroots Scalability – Mulicores
- Wide-area Scalability
- Application Ecosystems

HPC Performance Trends
HPC Performance Trends

Projected Performance Development

Top of the Top500

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Computer</th>
<th>Peak (Teraflop)</th>
<th>Installation Site</th>
<th>Country</th>
<th>Year</th>
<th>MFlops</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>Watson/L eServer Blue Gene</td>
<td>290.6</td>
<td>DOE Lawrence Livermore Nat Lab</td>
<td>USA</td>
<td>2005</td>
<td>131,072</td>
</tr>
<tr>
<td>IBM</td>
<td>eServer Blue Gene</td>
<td>91.29</td>
<td>IBM Thomas Watson Research</td>
<td>USA</td>
<td>2006</td>
<td>409,600</td>
</tr>
<tr>
<td>IBM</td>
<td>Power6 p575</td>
<td>6.37</td>
<td>DOE Lawrence Livermore Nat Lab</td>
<td>USA</td>
<td>2007</td>
<td>1,284,000</td>
</tr>
<tr>
<td>SGI</td>
<td>Columbia Altix, Intel/Quad</td>
<td>11.87</td>
<td>NASA Ames</td>
<td>USA</td>
<td>2004</td>
<td>1,016,000</td>
</tr>
<tr>
<td>Dell</td>
<td>Thunderbird Pentium/Dual</td>
<td>18.07</td>
<td>DOE Sandia Nat Lab</td>
<td>USA</td>
<td>2005</td>
<td>8000</td>
</tr>
<tr>
<td>Cray</td>
<td>T3E-3600</td>
<td>36.15</td>
<td>DOE Sandia Nat Lab</td>
<td>USA</td>
<td>2006</td>
<td>10,388,000</td>
</tr>
<tr>
<td>IBM</td>
<td>M6000 Earth Simulator</td>
<td>18.66</td>
<td>National Center for Supercomputing Applications</td>
<td>USA</td>
<td>2002</td>
<td>9,412</td>
</tr>
<tr>
<td>IBM</td>
<td>M6000</td>
<td>27.91</td>
<td>Barcelona Supercomputer</td>
<td>Spain</td>
<td>2005</td>
<td>4800</td>
</tr>
<tr>
<td>IBM</td>
<td>eServer Blue Gene</td>
<td>27.45</td>
<td>ASTRON University Groningen</td>
<td>Netherlands</td>
<td>2006</td>
<td>12,288</td>
</tr>
<tr>
<td>Cray</td>
<td>Jaguar</td>
<td>20.53</td>
<td>DOE Oak Ridge Nat Lab</td>
<td>USA</td>
<td>2006</td>
<td>5200</td>
</tr>
</tbody>
</table>
HPC Architecture Trends

![Architecture Share Over Time 1993-2007](chart)

HPC – Upcoming Hybrid Architectures

Roadrunner is a Petascale system in 2008

- "Connected Unit" cluster
  - 192 Opteron nodes
  - (180 w/ 3 dual-CPU blades connected w/ 4 PCIe x8 links)
- ~7,000 dual-core Opterons
  - ~50 TeraFlop/s (from Opterons)
  - 13,000 Cell HPC chips
  - 1.4 PetaFlop/s (from Cell)
- ~18 clusters
  - 2nd stage InfiniBand 4x DDR interconnects
  - (18 sets of 12 links to 6 switches)
  - 2nd Gen IB 4x DDR
HPC Scalability Issues

- Parallelism Model/Paradigm
  - SIMD, SPMD, MIMD, Vector
  - DAG/Event Driven
  - Hierarchical/Multi-level

- Runtime Management
  - Load-balancing
  - Adaptation
  - Fault-Tolerance
  - Data management

- Locality! Locality! Locality!
  - Memory, Cache
  - Communication

Principles of Parallel Computing

- Parallelism and Amdahl's Law
- Finding and exploiting granularity
- Preserving data locality
- Load balancing
- Coordination and synchronization
- Performance modeling

All of these things make parallel programming more difficult than sequential programming.
Finding Enough Parallelism

- Suppose only part of an application seems parallel
- Amdahl’s law
  - Let $s$ be the fraction of work done sequentially, so $(1-s)$ is fraction parallelizable.
  - $P$ = number of processors.

\[
\text{Speedup}(P) = \frac{\text{Time}(1)}{\text{Time}(P)} \\
\leq \frac{1}{s + \frac{(1-s)}{P}} \\
\leq \frac{1}{s}
\]

Even if the parallel part speeds up perfectly, we may be limited by the sequential portion of code.

Overhead of Parallelism

- Given enough parallel work, this is the most significant barrier to getting desired speedup.
- Parallelism overheads include:
  - cost of starting a thread or process
  - cost of communicating shared data
  - cost of synchronizing
  - extra (redundant) computation
- Each of these can be in the range of milliseconds (= millions of flops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work.
Locality and Parallelism

- Large memories are slow, fast memories are small.
- Storage hierarchies are large and fast on average.
- Parallel processors, collectively, have large, fast memories -- the slow accesses to "remote" data we call "communication".
- Algorithm should do most work on local data.

Load Imbalance

- Load imbalance is the time that some processors in the system are idle due to
  - insufficient parallelism (during that phase).
  - unequal size tasks.
- Examples of the latter
  - adapting to "interesting parts of a domain".
  - tree-structured computations.
  - fundamentally unstructured problems.
- Algorithm needs to balance load
  - but techniques to balance load often reduce locality
“Automatic” Parallelism in Modern Machines

- Bit level parallelism: within floating point operations, etc.
- Instruction level parallelism (ILP): multiple instructions execute per clock cycle.
- Memory system parallelism: overlap of memory operations with computation.
- OS parallelism: multiple jobs run in parallel on commodity SMPs.

There are limitations to all of these!
- Thus to achieve high performance, the programmer needs to identify, schedule and coordinate parallel tasks and data.

Grassroots Scalability – Motivations for Multicore

- Performance rose over 2000-2004:
  - Along with increase in power
  - GHz race
- Performance plateau in 2003
- Power plateau in 2003:
  - Power wall
- Instruction level parallelism (ILP) plateau in 2000:
  - Limits to automatic extraction of parallelism
How fast can a serial computer be?

Consider the 1 Tflop/s sequential machine:
- Data must travel some distance, $r$, to get from memory to CPU.
- To get 1 data element per cycle, this means $10^{12}$ times per second at the speed of light, $c = 3 \times 10^8$ m/s. Thus $r < c/10^{12} = 0.3$ mm.

Now put 1 Tbyte of storage in a 0.3 mm x 0.3 mm area:
- Each word occupies about 3 square Angstroms, or the size of a small atom.

Impact of Device Shrinkage

- What happens when the feature size shrinks by a factor of $x$?
- Clock rate goes up by $x$
  - actually less than $x$, because of power consumption
- Transistors per unit area goes up by $x2$
- Die size also tends to increase
  - typically another factor of $\sim x$
- Raw computing power of the chip goes up by $\sim x4$!
  - of which $x3$ is devoted either to parallelism or locality
Grassroots Scalability – Motivations for Multicore

- But now the bad news: the power $E$ burned by the chip depends on Capacity $C$ (transistors etc.), $V$ and $f$

  $$E = C \cdot V^2 \cdot f$$

  but because $f \approx V$

  $$E = C \cdot V^3$$

- So if we reduce the voltage $V$ by 20% we still get 0.8 of the original performance. But $E$ goes down dramatically because $0.8^3 \approx 0.5$!

- So with 2 processors (cores) we have the same power consumption as the original single core but $2 \times 0.8 = 1.6$ times the total performance!
Grassroots Scalability – Multicore is here to stay!

- **Single core trends are gloomy**
  - Performance is difficult to exploit
  - Perf/Power and Perf/Area deteriorate
- **Fill the die w/ cache has limited value**
  - Between 2MB and 4MB cache – gain is diminishing
- **Solution: More cores on a die**
  - Advantages – higher performance, constant perf/power, less complexity, lower wire delays
  - Challenges – scalability of interconnect, scalability of application, limited memory BW, power!!
- **Other solutions – more integration - memory controller, special functions/Accelerators, graphics, etc...**

**CELL B.E. Architecture**
NVIDIA GPU Architecture G80

ATI GPU Architecture – R600
Intel Polaris

**Teraflops Research Chip**
100 Million Transistors • 80 Tiles • 275 mm²

First tera-scale programmable silicon:
- Teraflops performance
- Tile design approach
- On-die mesh network
- Novel clocking
- Power-aware capability
- Supports 3D-memory
Not designed for IA or product

Grassroots Scalability – The Memory Bottleneck

**Memory bottleneck:**
- Cores share finite off-chip bandwidth
- Latency cannot be improved significantly
  ➔ and is already 100’s of cycles!
Grassroots Scalability – Hiding Latency

• To hide latency, need **more** parallelism than number of cores, **schedule** data:

\[
\text{Concurrency} = \text{parallelism} \times \text{latency}
\]

Grassroots Scalability – Key Issues

- Programming Model, Language and Compiler
  - Conventional
  - Parallelizing (Including JIT)
  - Parallelizing with Directives
  - New Parallel Languages (Transactional Memory, ...)

- Use of Threads
  - Functional
  - Speculative
  - Assist/Helper (Prefetching, Monitoring, Debugging, Tools, Virtualization, Security, FT-Lockstep, ...)

- Scalable Tool Models
Wide-Area Scalability: The Grid Computing Paradigm

**Grid Computing paradigm is an emerging way of thinking about a global infrastructure for:**

- Sharing data
- Distribute computation
- Coordinating problem solving processes
- Accessing remote instrumentation

---

**Key Enablers of Grid Computing - Exponentials**

- **Network vs. computer performance**
  - Computer speed doubles every 18 months
  - Storage density doubles every 12 months
  - Network speed doubles every 9 months
  - Difference = order of magnitude per 5 years

- **1986 to 2000**
  - Computers: x 500
  - Networks: x 340,000

- **2001 to 2010**
  - Computers: x 60
  - Networks: x 4000

*"When the network is as fast as the computer's internal links, the machine disintegrates across the net into a set of special purpose appliances"*

(George Gilder)
The Grid Concept

“Resource sharing & coordinated computational problem solving in dynamic, multi-institutional virtual organizations”

1. Enable integration of distributed resources
2. Using general-purpose protocols & infrastructure
3. To achieve better-than-best-effort service

Grids – An evolving vision …

- Seamless aggregation
  - aggregation of capacity
  - aggregation of capability

- Seamless compositions and interactions

- Autonomic behaviors
Wide-Area Scalability - Clouds

Clouds – Shapes and Sizes

- Amazon EC2/S3
- Google AppEngine
- Sun Caroline, Hydrazine
- 3Tera
- XCalibre Communications FlexiScale
- Elastra
- Demandware
- 10gen
- …
Cloud Computing

Mesos
Google App Engine
Rails One
Salesforce
Orchard
Gilly
Joyent
Amazon Web Services
Nineteen
XCelliro
Alamal

PaaS
SaaS
IaaS

Cloud Computing
Utility Computing
Grid Computing
Cluster Computing
Super Computing

1.2 In the you have what is lately called Cloud Computing. In green, some of the underlying work done that led to Cloud Computing. At the top are examples of each PaaS type.
Conclusion

• All processors will be multi-core
• All computer will be massively parallel
• All programmers will be parallel programmers
• All programs will be parallel programs

• The key challenge is how to enable mainstream developers to scale their applications on machines!
  – Algorithms – asynchronous, fault-tolerant
  – Programming models and systems
  – Adaptive runtime management
  – Data management
  – System support