Five-straight big games! ⇒ This one was supposed to be easy!
Stanford showed up, and so did the winds as Cal struggled to beat SU, a 1-10 team and the weakest opponent all year. Next game is 2006-12-28 in the Holiday Bowl in San Diego vs 9-3 Texas A&M.

Outline

• Large, important problems require powerful computers
• Why powerful computers must be distributed or parallel computers
• What are these?
• Principles of parallel computing performance

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 models

• Computational science paradigm:
  • Use high performance computer (HPC) systems to simulate the phenomenon
  • Based on physical laws & efficient numerical methods

Example Applications

• Science
  • Global climate modeling
  • Biology: genomics; protein folding; drug design; malaria simulations
  • Astrophysical modeling
  • Computational Chemistry, Material Sciences and Nanosciences
  • SETI@Home: Search for Extra-Terrestrial Intelligence

• Engineering
  • Semiconductor design
  • Earthquake and structural modeling
  • Fluid dynamics (airplane design)
  • Combustion (engine design)
  • Crash simulation
  • Computational Game Theory (e.g., Chess Databases)

• Business
  • Rendering computer graphic imagery (CGI), a la Pixar and ILM
  • Financial and economic modeling
  • Transaction processing, web services and search engines

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

Performance Requirements

• Terminology
  • FLOP – Floating point OPeration
  • Flops/second – standard metric for expressing the computing power of a system

• Example : Global Climate Modeling
  • Divide the world into a grid (e.g. 10 km spacing)
  • Solve fluid dynamics equations to determine what the air has done at that point every minute
    • Requires about 100 Flops per grid point per minute
  • This is an extremely simplified view of how the atmosphere works, to be maximally effective you need to simulate many additional systems on a much finer grid

Performance Requirements (2)

• Computational Requirements
  • To keep up with real time (i.e. simulate one minute per wall clock minute):
    • 8 Gflops/sec
  • Weather Prediction (7 days in 24 hours):
    • 56 Gflops/sec
  • Climate Prediction (50 years in 30 days):
    • 4.8 Tflops/sec
  • Climate Prediction Experimentation (50 yrs in 12 hrs):
    • 288 Tflops/sec

• Perspective
  • Pentium 4 1.4GHz, 1GB RAM, 4x100MHz FSB
    • ~0.3 Gflops/sec, effective
    • Climate Prediction would take ~1233 years

References:

www.epm.ornl.gov/chammp/chammp.html
What Can We Do?

- Wait for our machines to get faster?
  - Moore's law tells us things are getting better; why not stall for the moment?
- Heat issues ⇒ Moore on last legs!
  - Many believe so ... thus push for multi-core (fri)!
- Prohibitive costs:
  - Rock's law
  - Cost of building a semiconductor chip fabrication plant capable of producing chips in line w/Moore's law doubles every four years
  - In 2003, it cost $3 BILLION

Alternatively, many CPUs at once!

### Distributed computing

- Many computers (either homogeneous or heterogeneous) who get “work units” from a “central dispatcher” and return when they’re done to get more
  - “(Grid, Cluster) Computing”
- Parallel Computing
  - Multiple processors “all in one box” executing identical code on different parts of problem to arrive at a unified (meaningful) solution
  - “Supercomputing”

### Google’s MapReduce

- Remember CS61A?
  
  ```
  (reduce + (map square '(1 2 3))) ⇒
  (reduce + '(1 4 9)) ⇒
  14
  ```
- We told you “the beauty of pure functional programming is that it’s easily parallelizable”
  - Do you see how you could parallelize this?
  - What if the reduce function argument were associative, would that help?
- Imagine 10,000 machines ready to help you compute anything you could cast as a MapReduce problem!
  - This is the abstraction Google is famous for authoring (but their reduce not exactly the same as the CS61A’s reduce)
  - It hides LOTS of difficulty of writing parallel code!
  - The system takes care of load balancing, dead machines, etc

### MapReduce Programming Model

Input & Output: each a set of key/value pairs

Programmer specifies two functions:

#### map

```java
map (in_key, in_value) ⇒ 
  list(out_key, intermediate_value)
```

- Processes input key/value pair
- Produces set of intermediate pairs

#### reduce

```java
reduce (out_key, list(intermediate_value)) ⇒ 
  list(out_value)
```

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usu just one)

What do you mean by “intermediate”? What if a value has nothing to reduce? Only emit “intermediate” values that are meaningful

### Example: Word Occurrences in Files

```java
map(String input_key, 
     String input_value):
  // input_key: document name 
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");

reduce(String output_key, 
        Iterator intermediate_values):
  // output_key: a word
  // output_values: a list of counts
  int result = 0; 
  for each v in intermediate_values:
    result += ParseInt(v); 
    Emit(AsString(result));
```

- This is just an example
  - Real code at Google is more complicated and in C++
  - Real code at Hadoop (open source MapReduce) in Java
**Example: MapReduce Execution**

<table>
<thead>
<tr>
<th>Input</th>
<th>ah</th>
<th>ah</th>
<th>or</th>
<th>ah</th>
<th>or</th>
<th>or</th>
<th>or</th>
<th>uh</th>
<th>or</th>
<th>ah</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ah</td>
<td>ah</td>
<td>or</td>
<td>ah</td>
<td>or</td>
<td>or</td>
<td>or</td>
<td>uh</td>
<td>or</td>
<td>ah</td>
</tr>
<tr>
<td>Intermediates</td>
<td>ah</td>
<td>ah</td>
<td>or</td>
<td>or</td>
<td>or</td>
<td>or</td>
<td>or</td>
<td>uh</td>
<td>or</td>
<td>ah</td>
</tr>
<tr>
<td>Group by Key</td>
<td>ah:1</td>
<td>ah:1</td>
<td>or:1</td>
<td>or:1</td>
<td>or:1</td>
<td>or:1</td>
<td>or:1</td>
<td>or:1</td>
<td>or:1</td>
<td>or:1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>4</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
</table>

**How Do I Experiment w/MapReduce?**

1. Work for Google. :-)
2. Use the Open Source versions
   - Hadoop: map/reduce & distributed file system: lucene.apache.org/hadoop/
   - Nutch: crawler, parsers, index: lucene.apache.org/nutch/
   - Lucene Java: text search engine library: lucene.apache.org/java/docs/
3. Wait until 2007Fa
   - (You heard it here 1st!) Google will be giving us:
     - Single Rack, 40 dual proc xeon, 4GB Ram, 600GB Disk
     - Also available as vmware image
     - We’re developing bindings for Scheme for 61A
     - It will be available to students & researchers

**Recent History of Parallel Computing**

- Parallel Computing as a field exploded in popularity in the mid-1990s
- This resulted in an “arms race” between universities, research labs, and governments to have the fastest supercomputer in the world
  - LINPACK (solving dense system of linear equations) is benchmark

**Current Champions (Nov 2006)**

- **BlueGene/L**
  - eServer Blue Gene Solution
  - IBM DOE / NNSA / LLNL
  - Livermore, CA, United States
  - 65,536 dual-processors, 280.6 Tflops/s
  - 0.7 GHz PowerPC 440, IBM

- **Red Storm**
  - Sandia / Cray Red Storm
  - NNSA / Sandia National Laboratories
  - Albuquerque, NM, United States
  - 26,544 Processors, 101.4 Tflops/s
  - 2.4 GHz dual-core Opteron, Cray Inc.

- **BGW**
  - eServer Blue Gene Solution
  - IBM TJ Watson Research Center
  - Yorktown Heights, NY, United States
  - 40,960 Processors, 91.3 Tflops/s
  - 0.7 GHz PowerPC 440, IBM

**Synchronization (1)**

- How do processors communicate with each other?
- How do processors know when to communicate with each other?
- How do processors know which other processor has the information they need?
- When you are done computing, which processor, or processors, have the answer?

**Synchronization (2)**

- Some of the logistical complexity of these operations is reduced by standard communication frameworks
  - Message Passing Interface (MPI)
- Sorting out the issue of who holds what data can be made easier with the use of explicitly parallel languages
  - Unified Parallel C (UPC)
  - Titanium (Parallel Java Variant)
- Even with these tools, much of the skill and challenge of parallel programming is in resolving these problems
Parallel Processor Layout Options

- **Shared Memory**
  - Bus
  - Memory

- **Distributed Memory**
  - Interconnect
  - Memory

---

Parallel Locality

- We now have to expand our view of the memory hierarchy to include remote machines
- Remote memory behaves like a very fast network
  - Bandwidth vs. Latency becomes important

**Reg & Cache**

**Memory**

**Remote Memory**

**Local and Remote Disk**

---

Amdahl’s Law

- Applications can almost never be completely parallelized
  - Let $s$ be the fraction of work done sequentially, so $(1-s)$ is fraction parallelizable, and $P$ = number of processors
  - Speedup($P$) = Time(1) / Time($P$)
    $\leq \frac{1}{s + ((1-s) / P)}$, and as $P \to \infty$
    $\leq \frac{1}{s}$
  - Even if the parallel portion of your application speeds up perfectly, your performance may be limited by the sequential portion

---

Parallel Overhead

- Given enough parallel work, these are the biggest barriers 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 ms (many Mflops) 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

---

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
- Algorithms need to carefully balance load

---

Summary

- Parallel & Distributed Computing is a multi-billion dollar industry driven by interesting and useful scientific computing & business applications
- It is extremely unlikely that sequential computing will ever again catch up with the processing power of parallel or distributed systems
- Programming parallel systems can be extremely challenging (distributed computing less so, thanks to APIs), but is built upon many of the concepts you’ve learned this semester in 61C