Today

- What is Parallel Processing? Why?
- Kinds of Parallel Processing
- Multiprocessing and Multithreading
- Measuring success
  - Speedup
  - Amdhal’s Law
- Bottlenecks to parallelism
Concurrent Systems

- Embedded-Physical Distributed

Claytronics

Sensor Networks
Concurrent Systems

- Embedded-Physical Distributed
  - Claytronics
  - Sensor Networks

- Geographically Distributed
  - Internet
  - Power Grid
Concurrent Systems

- Embedded-Physical Distributed
  - Claytronics
  - Sensor Networks

- Geographically Distributed
  - Internet
  - Power Grid

- Cloud Computing
  - EC2
  - Tashi
Concurrent Systems

- Embedded-Physical Distributed
  - Claytronics
  - Sensor Networks

- Geographically Distributed
  - Internet
  - Power Grid

- Cloud Computing
  - EC2
  - Tashi

- Parallel

© 2007-9 Goldstein
## Concurrent Systems

<table>
<thead>
<tr>
<th></th>
<th>Physical</th>
<th>Geographical</th>
<th>Cloud</th>
<th>Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geophysical location</td>
<td>+++</td>
<td>++</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Relative location</td>
<td>+++</td>
<td>+++</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Faults</td>
<td>+++</td>
<td>+++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Number of Processors</td>
<td>+++</td>
<td>+++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Network structure</td>
<td>varies</td>
<td>varies</td>
<td>fixed</td>
<td>fixed</td>
</tr>
<tr>
<td>Network connectivity</td>
<td>---</td>
<td>---</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
Concurrent System Challenge: Programming

The old joke:
How long does it take to write a parallel program?

One Graduate Student Year
Parallel Programming Again??

- Increased demand (multicore)
- Increased scale (cloud)
- Improved compute/communicate
- Change in Application focus
  - Irregular
  - Recursive data structures
Why Parallel Computers?

- **Parallelism:** Doing multiple things at a time
- **Things:** instructions, operations, tasks

- **Main (historical?) Goal**
  - Improve performance (Execution time or task throughput)
    - Execution time of a program governed by Amdahl’s Law

- **Other (more recent) Goals**
  - Reduce power consumption
    - If task is parallel, more slower units consume less power than 1 faster unit
      - $P = \frac{1}{2}CVF^2$ and $V \propto F$
  - Improve cost efficiency and scalability, reduce complexity
    - Harder to design a single unit that performs as well as $N$ simpler units
  - Improve dependability: Redundant execution in space
What is Parallel Architecture?

- A parallel computer is a collection of processing elements that cooperate to solve large problems fast.

Some broad issues:

- **Resource Allocation:**
  - how large a collection?
  - how powerful are the elements?
  - how much memory?

- **Data access, Communication and Synchronization**
  - how do the elements cooperate and communicate?
  - how are data transmitted between processors?
  - what are the abstractions and primitives for cooperation?

- **Performance and Scalability**
  - how does it all translate into performance?
  - how does it scale?
Flynn’s Taxonomy of Computers


- **SI SD**: Single instruction operates on single data element
- **SI MD**: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- **MI SD**: Multiple instructions operate on single data element
  - Closest form?: systolic array processor, streaming processor
- **MI MD**: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor
Types of Parallelism and How to Exploit Them

- **Instruction Level Parallelism**
  - Different instructions within a stream can be executed in parallel
  - Pipelining, out-of-order execution, speculative execution, VLIW
  - Dataflow

- **Data Parallelism**
  - Different pieces of data can be operated on in parallel
  - SIMD: Vector processing, array processing
  - Systolic arrays, streaming processors

- **Task Level Parallelism**
  - Different “tasks/threads” can be executed in parallel
  - Multithreading
  - Multiprocessing (multi-core)
Task-Level Parallelism: Creating Tasks

- Partition a single problem into multiple related tasks (threads)
  - Explicitly: Parallel programming
    - Easy when tasks are natural in the problem
      - Web/database queries
    - Difficult when natural task boundaries are unclear
  - Transparently/implicitly: Thread level speculation
    - Partition a single thread speculatively

- Run many independent tasks (processes) together
  - Easy when there are many processes
    - Batch simulations, different users, cloud computing workloads
  - Does not improve the performance of a single task
Multiprocessing Fundamentals
Multiprocessor Types

- Loosely coupled multiprocessors
  - No shared global memory address space
  - Multicomputer network
    - Network-based multiprocessors
  - Usually programmed via message passing
    - Explicit calls (send, receive) for communication

- Tightly coupled multiprocessors
  - Shared global memory address space
  - Traditional multiprocessing: symmetric multiprocessing (SMP)
    - Existing multi-core processors, multithreaded processors
  - Programming model similar to uniprocessors (i.e., multitasking uniprocessor) except
    - Operations on shared data require synchronization
Main Issues in Tightly-Coupled MP

- **Shared memory synchronization**
  - Locks, atomic operations

- **Cache consistency**
  - More commonly called cache coherence

- **Ordering of memory operations**
  - What should the programmer expect the hardware to provide?

- **Resource sharing, contention, partitioning**

- **Communication: Interconnection networks**

- **Load imbalance**
Aside: Hardware-based Multithreading

- **Idea:** Multiple threads execute on the same processor with multiple hardware contexts; hardware controls switching between contexts

- **Coarse grained**
  - Quantum based
  - Event based (switch-on-event multithreading)

- **Fine grained**
  - Cycle by cycle

- **Simultaneous**
  - Can dispatch instructions from multiple threads at the same time
  - Good for improving utilization of multiple execution units
Metrics of Multiprocessors
Parallel Speedup

Time to execute the program with 1 processor divided by
Time to execute the program with \( N \) processors
Parallel Speedup Example

- $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

- Assume each operation 1 cycle, no communication cost, each op can be executed in a different processor

- How fast is this with a single processor?
  - Assume no pipelining or concurrent execution of instructions

- How fast is this with 3 processors?
\[ R = a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0 \]

Single processor: 11 operations (data flow graph)

\[ \tau_1 = 11 \text{ cycles} \]
\[ R = a_4 x^4 + a_3 x^2 + a_2 x^2 + a_1 x + a_0 \]

Three processors: \( T_3 \) (execute with 3 procs.)

\[ T_3 = 5 \text{ cycles} \]
Speedup with 3 Processors

\[ T_3 = \frac{11}{5} = 2.2 \]

\[ \left( \frac{T_1}{T_3} \right) \]

Is this a fair comparison?
Revisiting the Single-Processor Algorithm

Revisit T1

Better single-processor algorithm:

\[ R = a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0 \]

\[ R = (((a_4 x + a_3) x + a_2) x + a_1) x + a_0 \]

(Horner's method)

\[ T_1 = 8 \text{ cycles} \]
\[
\text{Speedup with 3 proc.} = \frac{T_1^{\text{best}}}{T_3^{\text{best}}} = \frac{8}{5} = 1.6 \quad \text{(not 2.2)}
\]
Takeaway

- To calculate parallel speedup fairly you need to use the best known algorithm for each system with N processors.

- If not, you can get superlinear speedup.
Superlinear Speedup

- Can speedup be greater than P with P processing elements?

Consider:
- Cache effects
- Memory effects
- Working set

Happens in two ways:
- Unfair comparisons
- Memory effects
Utilization, Redundancy, Efficiency

- Traditional metrics
  - Assume all P processors are tied up for parallel computation

- Utilization: How much processing capability is used
  - \( U = \frac{\text{(# Operations in parallel version)}}{\text{(processors x Time)}} \)

- Redundancy: how much extra work is done with parallel processing
  - \( R = \frac{\text{(# of operations in parallel version)}}{\text{(# operations in best single processor algorithm version)}} \)

- Efficiency
  - \( E = \frac{\text{(Time with 1 processor)}}{\text{(processors x Time with P processors)}} \)
  - \( E = \frac{U}{R} \)
Utilization of a Multiprocessor

Utilization: How much processing capability we use.

\[
U = \frac{\text{Ops with proc.}}{P \times T_P}
\]

\[
U = \frac{10 \text{ operations (in parallel version)}}{3 \text{ processors} \times 5 \text{ time units}} = \frac{10}{15}
\]
Redundancy: How much extra work due to multiprocessing

\[ R = \frac{\text{Ops with } p \text{ proc.}}{\text{Ops with 1 proc.}} \]

best

best

= \frac{10}{8}

R is always \( \geq 1 \)

Efficiency: How much resource we use compared to how much resource we can get away with

\[ E = \frac{1}{T_1} \frac{T_1^{\text{best}}}{p \cdot T_p^{\text{best}}} \]

(tying up 1 proc for \( T_p \) time units)

(tying up \( p \) proc for \( T_p \) time units)

= \frac{8}{15}

\( (E = \frac{U}{R}) \)
You plan to visit a friend in Normandy France and must decide whether it is worth it to take the Concorde SST ($3,100) or a 747 ($1,021) from NY to Paris, assuming it will take 4 hours Pgh to NY and 4 hours Paris to Normandy.

<table>
<thead>
<tr>
<th></th>
<th>time NY-&gt;Paris</th>
<th>total trip time</th>
<th>speedup over 747</th>
</tr>
</thead>
<tbody>
<tr>
<td>747</td>
<td>8.5 hours</td>
<td>16.5 hours</td>
<td>1</td>
</tr>
<tr>
<td>SST</td>
<td>3.75 hours</td>
<td>11.75 hours</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Taking the SST (which is 2.2 times faster) speeds up the overall trip by only a factor of 1.4!
Amdahl’s law (cont)

Old program (unenhanced)

\[ T_1 \text{ time that can NOT be enhanced.} \]
\[ T_2 \text{ time that can be enhanced.} \]
\[ T_1 = T_{old} \]
\[ T_2 \leq T_{new} \]

New program (enhanced)

\[ T'_1 = T_1 \]
\[ T'_2 \leq T_2 \]

New time: \[ T' = T'_1 + T'_2 \]

Speedup: \[ S_{overall} = \frac{T}{T'} \]
Amdahl’s law (cont)

Two key parameters:

\[ F_{\text{enhanced}} = \frac{T_2}{T} \quad \text{(fraction of original time that can be improved)} \]
\[ S_{\text{enhanced}} = \frac{T_2}{T'} \quad \text{(speedup of enhanced part)} \]

\[
T' = T_1' + T_2' = T_1 + T_2' = T(1-F_{\text{enhanced}}) + T_2'
= T(1-F_{\text{enhanced}}) + \left(\frac{T_2}{S_{\text{enhanced}}}\right) \quad \text{[by def of } S_{\text{enhanced}}]\]
= T(1-F_{\text{enhanced}}) + T\left(\frac{F_{\text{enhanced}}}{S_{\text{enhanced}}}\right) \quad \text{[by def of } F_{\text{enhanced}}]\]
= T\left(1-F_{\text{enhanced}} + \frac{F_{\text{enhanced}}}{S_{\text{enhanced}}}\right)
\]

Amdahl’s Law:

\[ S_{\text{overall}} = \frac{T}{T'} = \frac{1}{\left(1-F_{\text{enhanced}} + \frac{F_{\text{enhanced}}}{S_{\text{enhanced}}}\right)} \]

Key idea: Amdahl’s law quantifies the general notion of diminishing returns. It applies to any activity, not just computer programs.
Trip example: Suppose that for the New York to Paris leg, we now consider the possibility of taking a rocket ship (15 minutes) or a handy rip in the fabric of space-time (0 minutes):

<table>
<thead>
<tr>
<th></th>
<th>time NY-&gt;Paris</th>
<th>total trip time</th>
<th>speedup over 747</th>
</tr>
</thead>
<tbody>
<tr>
<td>747</td>
<td>8.5 hours</td>
<td>16.5 hours</td>
<td>1</td>
</tr>
<tr>
<td>SST</td>
<td>3.75 hours</td>
<td>11.75 hours</td>
<td>1.4</td>
</tr>
<tr>
<td>rocket</td>
<td>0.25 hours</td>
<td>8.25 hours</td>
<td>2.0</td>
</tr>
<tr>
<td>rip</td>
<td>0.0 hours</td>
<td>8 hours</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Amdahl’s law (cont)

- Useful corollary to Amdahl’s law:
  \[ 1 \leq S_{overall} \leq \frac{1}{1 - F_{enhanced}} \]

<table>
<thead>
<tr>
<th>( F_{enhanced} )</th>
<th>( \text{Max } S_{overall} )</th>
<th>( F_{enhanced} )</th>
<th>( \text{Max } S_{overall} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1</td>
<td>0.9375</td>
<td>16</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.96875</td>
<td>32</td>
</tr>
<tr>
<td>0.75</td>
<td>4</td>
<td>0.984375</td>
<td>64</td>
</tr>
<tr>
<td>0.875</td>
<td>8</td>
<td>0.9921875</td>
<td>128</td>
</tr>
</tbody>
</table>

Moral: It is hard to speed up a program.

Moral++ : It is easy to make premature optimizations.
Caveats of Parallelism (I)

Why the reality? (diminishing returns)

\[ T_p = \alpha \cdot \frac{T_1}{p} + (1-\alpha) \cdot T_1 \]

- Parallelizable part/ fraction of the single-processor program
- Non-parallelizable part
Amdahl’s Law

\[
\text{Speedup with } p \text{ proc.} = \frac{T_1}{T_p} = \frac{1}{\frac{\alpha}{p} + (1-\alpha)}
\]

\[
\text{Speedup as } p \to \infty = \frac{1}{1-\alpha} \quad \text{but bottleneck for parallel speedup}
\]

Caveats of Parallelism (I): Amdahl’s Law

- **Amdahl’s Law**
  - \( f \): Parallelizable fraction of a program
  - \( P \): Number of processors

\[
\text{Speedup} = \frac{1}{1 - f + \frac{f}{P}}
\]


- Maximum speedup limited by serial portion: **Serial bottleneck**
Amdahl’s Law Implication 1

Amdahl’s Law illustrated

Adding more and more processors gives less and less benefit if $\alpha < 1$
Amdahl’s Law Implication 2

The benefit (speedup) is small until $\alpha \approx 1$.
Sequential Bottleneck

![Graph showing speedup vs. parallel fraction for different N values (N=10, N=100, N=1000). The x-axis represents the parallel fraction, and the y-axis represents speedup.]
Why the Sequential Bottleneck?

- Parallel machines have the sequential bottleneck

- Main cause: Non-parallelizable operations on data (e.g. non-parallelizable loops)
  
  ```
  for ( i = 0 ; i < N; i++)
  ```

- Single thread prepares data and spawns parallel tasks (usually sequential)
Another Example of Sequential Bottleneck
Implications of Amdahl’s Law on Design

- CRAY-1

- Well known as a fast vector machine
  - 8 64-element vector registers

- The fastest SCALAR machine of its time!
  - Reason: Sequential bottleneck!
Caveats of Parallelism (II)

- **Amdahl’s Law**
  - $f$: Parallelizable fraction of a program
  - $P$: Number of processors

\[
\text{Speedup} = \frac{1}{1 - f + \frac{f}{P}}
\]


- **Maximum speedup limited by serial portion:** Serial bottleneck

- **Parallel portion is usually not perfectly parallel**
  - **Synchronization** overhead (e.g., updates to shared data)
  - **Load imbalance** overhead (imperfect parallelization)
  - **Resource sharing** overhead (contention among $N$ processors)
Bottlenecks in Parallel Portion

- **Synchronization**: Operations manipulating shared data cannot be parallelized
  - Locks, mutual exclusion, barrier synchronization
  - **Communication**: Tasks may need values from each other
    - Causes thread serialization when shared data is contended

- **Load Imbalance**: Parallel tasks may have different lengths
  - Due to imperfect parallelization or microarchitectural effects
    - Reduces speedup in parallel portion

- **Resource Contention**: Parallel tasks can share hardware resources, delaying each other
  - Replicating all resources (e.g., memory) expensive
    - Additional latency not present when each task runs alone
Difficulty in Parallel Programming

- Little difficulty if parallelism is natural
  - “Embarrassingly parallel” applications
  - Multimedia, physical simulation, graphics
  - Large web servers, databases?

- Big difficulty is in
  - Harder to parallelize algorithms
  - Getting parallel programs to work correctly
  - Optimizing performance in the presence of bottlenecks

- Much of parallel computer architecture is about
  - Designing machines that overcome the sequential and parallel bottlenecks to achieve higher performance and efficiency
  - Making programmer’s job easier in writing correct and high-performance parallel programs
Parallel and Serial Bottlenecks

- How do you alleviate some of the serial and parallel bottlenecks in a multi-core processor?

- We will return to this question in the next few lectures

- Reading list:
Bottlenecks in the Parallel Portion

- Amdahl’s Law does not consider these

- How do synchronization (e.g., critical sections), and load imbalance, resource contention affect parallel speedup?

- Can we develop an intuitive model (like Amdahl’s Law) to reason about these?
  - A research topic

- Example papers:

- Need better analysis of critical sections in real programs
Readings

- **Required**
  - Suleman et al., “*Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures*,” ASPLOS 2009.
  - Joao et al., “*Bottleneck Identification and Scheduling in Multithreaded Applications*,” ASPLOS 2012.

- **Recommended**
  - Culler & Singh, Chapter 1
Related Video

- 18-447 Spring 2013 Lecture 30B: Multiprocessors
  - http://www.youtube.com/watch?v=7ozCK_Mgxfk&list=PL5PHm2jkkXmidJ0d59REog9jDnPDTG6IJ&index=31
Computer Architecture: Parallel Processing Basics

Prof. Onur Mutlu
Carnegie Mellon University
Backup slides
Readings

- Required

- Recommended
  - Culler & Singh, Chapter 1
Referenced Readings (I)

Referenced Readings (II)

Related Video

- 18-447 Spring 2013 Lecture 30B: Multiprocessors
  - http://www.youtube.com/watch?v=7ozCK_Mgxfk&list=PL5PHm2jkkXmidJ0d59REog9jDnPDTG6IjJ&index=31