The Internet is a great way to get on the net. - Senator Bob Dole.
Before We Get Started…

- Last time
  - Wrap up overview of C programming
  - Start overview of parallel computing
    - Focused primarily on the limitations with the sequential computing model
    - These limitations and Moore’s law usher in the age of parallel computing

- Today
  - Discuss parallel computing models, hardware and software
  - Start discussion about GPU programming and CUDA

- Thank you, to those of you who took the time to register for auditing
The memory baseline is 64 KB DRAM in 1980 with a 1.07/year improvement in latency. CPU speed improved at 1.25/year till 1986, 1.52/year until 2004, and 1.2/year thereafter.
“Parallelism for Everyone”
Parallelism changes the game
- A large percentage of people who provide applications are going to have to care about parallelism in order to match the capabilities of their competitors.

competitive pressures = demand for parallel applications
Intel Larrabee and Knights Ferris

- Paul Otellini, President and CEO, Intel
  - "We are dedicating all of our future product development to multicore designs"
  - "We believe this is a key inflection point for the industry."

Larrabee a thing of the past now. Knights Ferry and Intel’s MIC (Many Integrated Core) architecture with 32 cores for now. Public announcement: May 31, 2010
Putting things in perspective…

<table>
<thead>
<tr>
<th>The way business has been run in the past</th>
<th>It will probably change to this…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increasing clock frequency is primary method of performance improvement</td>
<td>Processors parallelism is primary method of performance improvement</td>
</tr>
<tr>
<td>Don’t bother parallelizing an application, just wait and run on much faster sequential computer</td>
<td>Nobody is building one processor per chip. This marks the end of the La-Z-Boy programming era</td>
</tr>
<tr>
<td>Less than linear scaling for a multiprocessor is failure</td>
<td>Given the switch to parallel hardware, even sub-linear speedups are beneficial as long as you beat the sequential</td>
</tr>
</tbody>
</table>
End: Discussion of Computational Models and Trends

Beginning: Overview of HW&SW for Parallel Computing
Amdhal’s Law


“A fairly obvious conclusion which can be drawn at this point is that the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude”

• Let $r_s$ capture the amount of time that a program spends in components that can only be run sequentially

• Let $r_p$ capture the amount of time spent in those parts of the code that can be parallelized.

• Assume that $r_s$ and $r_p$ are normalized, so that $r_s + r_p = 1$

• Let $n$ be the number of threads used to parallelize the part of the program that can be executed in parallel

• The “best case scenario” speedup is

$$S = \frac{T_{old}}{T_{new}} = \frac{1}{r_s + \frac{r_p}{n}}$$
Amdahl’s Law

Sometimes called the law of diminishing returns

In the context of parallel computing used to illustrate how going parallel with a part of your code is going to lead to overall speedups

The art is to find for the same problem an algorithm that has a large $r_p$
  - Sometimes requires a completely different angle of approach for a solution

Nomenclature: algorithms for which $r_p=1$ are called “embarrassingly parallel”
Example: Amdhal’s Law

- Suppose that a program spends 60% of its time in I/O operations, pre and post-processing.
- The rest of 40% is spent on computation, most of which can be parallelized.
- Assume that you buy a multicore chip and can throw 6 parallel threads at this problem. What is the maximum amount of speedup that you can expect given this investment?
- Asymptotically, what is the maximum speedup that you can ever hope for?
A Word on “Scaling”

- **Algorithmic Scaling** of a solution algorithm
  - You only have a mathematical solution algorithm at this point
  - Refers to how the effort required by the solution algorithm scales with the size of the problem
  - Examples:
    - Naïve implementation of the N-body problem scales like $O(N^2)$, where $N$ is the number of bodies
    - Sophisticated algorithms scale like $O(N \cdot \log N)$
    - Gauss elimination scales like the cube of the number of unknowns in your linear system

- Scaling on an implementation on a certain architecture
  - **Intrinsic Scaling**: how the wall-clock run time increase with the size of the problem
  - **Strong Scaling**: how the wall-clock run time of an implementation changes when you increase the processing resources
  - **Weak Scaling**: how the wall-clock run time changes when you increase the problem size but also the processing resources in a way that basically keeps the ration of work/processor constant
  - Order of relevance: strong, intrinsic, weak

- A thing you should worry about: is the Intrinsic Scaling similar to the Algorithmic Scaling?
  - If Intrinsic Scaling significantly worse than Algorithmic Scaling:
    - You might have an algorithm that thrashes the memory badly, or
    - You might have a sloppy implementation of the algorithm
Overview of Large Multiprocessor Hardware Configurations
Newton: 24 GPU Cluster
~ Hardware Configurations ~

Legend, Connection Type:
- Ethernet Connection
- Fast Infiniband Connection

Remote User 1
Remote User 2
Remote User 3
Lab Computers
Internet
Ethernet Router
Head Node
Gigabit Ethernet Switch
Compute Node 1
Compute Node 2
Compute Node 3
Compute Node 4
Compute Node 5
Compute Node 6
Network-Attached Storage

Compute Node Architecture
- CPU 0 Intel Xeon 5520
- CPU 1 Intel Xeon 5520
- RAM 46 GB DDR3
- Infiniband Card QDR
- Hard Disk 1TB
- Tesla C1060
- 4GB RAM
- 240 Cores
- PCIe x16 2.0

User 1
User 2
User 3
Ethernet Connection
Fast Infiniband Connection
Internet Connection
Gigabit Ethernet Switch
Switch
Remote User 1
Remote User 2
Remote User 3
Lab Computers
Internet
Ethernet Router
Head Node
Gigabit Ethernet Switch
Compute Node 1
Compute Node 2
Compute Node 3
Compute Node 4
Compute Node 5
Compute Node 6
Network-Attached Storage

Legend, Connection Type:
- Ethernet Connection
- Fast Infiniband Connection

Remote User 1
Remote User 2
Remote User 3
Lab Computers
Internet
Ethernet Router
Head Node
Gigabit Ethernet Switch
Compute Node 1
Compute Node 2
Compute Node 3
Compute Node 4
Compute Node 5
Compute Node 6
Network-Attached Storage

Compute Node Architecture
- CPU 0 Intel Xeon 5520
- CPU 1 Intel Xeon 5520
- RAM 46 GB DDR3
- Infiniband Card QDR
- Hard Disk 1TB
- Tesla C1060
- 4GB RAM
- 240 Cores
- PCIe x16 2.0

User 1
User 2
User 3
Ethernet Connection
Fast Infiniband Connection
Internet Connection
Gigabit Ethernet Switch
Switch
Remote User 1
Remote User 2
Remote User 3
Lab Computers
Internet
Ethernet Router
Head Node
Gigabit Ethernet Switch
Compute Node 1
Compute Node 2
Compute Node 3
Compute Node 4
Compute Node 5
Compute Node 6
Network-Attached Storage

Legend, Connection Type:
- Ethernet Connection
- Fast Infiniband Connection

Remote User 1
Remote User 2
Remote User 3
Lab Computers
Internet
Ethernet Router
Head Node
Gigabit Ethernet Switch
Compute Node 1
Compute Node 2
Compute Node 3
Compute Node 4
Compute Node 5
Compute Node 6
Network-Attached Storage

Compute Node Architecture
- CPU 0 Intel Xeon 5520
- CPU 1 Intel Xeon 5520
- RAM 46 GB DDR3
- Infiniband Card QDR
- Hard Disk 1TB
- Tesla C1060
- 4GB RAM
- 240 Cores
- PCIe x16 2.0
Some Nomenclature

- Shared addressed space: when you invoke address “0x0043fc6f” on one machine and then invoke “0x0043fc6f” on a different machine they actually point to the same global memory space
  - Issues: memory coherence
    - Fix: software-based or hardware-based

- Distributed addressed space: the opposite of the above

- Symmetric Multiprocessor (SMP): you have one machine that shares amongst all its processing units a certain amount of memory (same address space)
  - Mechanisms should be in place to prevent data hazards (RAW, WAR, WAW). Goes back to memory coherence

- Distributed shared memory (DSM):
  - Also referred to as distributed global address space (DGAS)
  - Although physically memory is distributed, it shows as one uniform memory
  - Memory latency is highly unpredictable
Example, SMP

- Shared-Memory Multiprocessor (SMP) Architecture

Usually SRAM

Usually DRAM

Courtesy of Elsevier, Computer Architecture, Hennessey and Patterson, fourth edition
Comments, SMP Architecture

- Multiple processor-cache subsystems share the same physical off-chip memory

- Typically connected to this off-chip memory by one or more buses or a switch

- Key architectural property: uniform memory access (UMA) time to all of memory from all the processors
  - This is why it’s called symmetric
SRAM vs. DRAM

- **SRAM – Static Random Access Memory**
  - Six transistors
  - Need only be set once, no need to recharge as long as power is not cut off
  - Bulky and expensive
  - Very fast
  - Usually used for cache memory

- **DRAM – Dynamics Random Access Memory**
  - One transistor and two capacitors
  - The “Dynamic” attribute: Capacitors need to be constantly recharged
    - Therefore, longer access times, more power thirsty
    - Compact
    - Used for off-chip memory
Example

- Distributed-memory multiprocessor architecture (Newton, for instance)
Basic architecture consists of nodes containing a processor, some memory, typically some I/O, and an interface to an interconnection network that connects all the nodes.

Individual nodes may contain a small number of processors, which may be interconnected by a small bus or a different interconnection technology, which is less scalable than the global interconnection network.

Popular interconnection network: Mellanox and Qlogic InfiniBand
- Bandwidth: 40 GB/sec
- Latency: in the microsecond range
- Requires special network cards: HCA – “Host Channel Adaptor”

InfiniBand offers point-to-point bidirectional serial links intended for the connection of processors with high-speed peripherals such as disks.
- Basically, a protocol and implementation for communicating data very fast
- It supports several signaling rates and, as with PCI Express, links can be bonded together for additional throughput
- Similar technologies: Fibre Channel, PCI Express, Serial ATA, etc.
Examples…

- **Shared-Memory**
  - Nehalem micro-architecture, released in October 2008
  - AMD “Barcelona” (quad-core)
  - Sun Niagara

- **Distributed-Memory**
  - IBM BlueGene/L
  - Cell (see [http://users.ece.utexas.edu/~adnan/vlsi-07/hofstee-cell.ppt](http://users.ece.utexas.edu/~adnan/vlsi-07/hofstee-cell.ppt))

- **Mini-cores**
  - GPGPUs – General Purpose GPUs
Flynn’s Taxonomy of Architectures

- SISD - Single Instruction/Single Data
- SIMD - Single Instruction/Multiple Data
- MISD - Multiple Instruction/Single Data
- MIMD - Multiple Instruction/Multiple Data
Single Instruction/Single Data Architectures

PU – Processing Unit

Your desktop, before the spread of dual core CPUs

Flavors of SISD

Instructions:

Pipelining

Instruction-Level Parallelism (ILP)
More on pipelining...
Related to the Idea of Pipelining...

- Most processors have multiple pipelines for different tasks, and can start a number of different operations each cycle
- Example: each core in an Intel Core 2 Duo chip
  - 14-stage pipeline
  - 3 integer units (ALU)
  - 1 floating-point addition unit (FPU)
  - 1 floating-point multiplication unit (FPU)
  - 2 load/store units
  - In principle, capable of producing 3 integer and 2 FP results per cycle
  - FP division is very slow
Single Instruction/Multiple Data Architectures

Processors that execute same instruction on multiple pieces of data: NVIDIA GPUs

Single Instruction/Multiple Data [Cntd.]

- Each core runs the same set of instructions on different data
- Examples:
  - Graphics Processing Unit (GPU): processes pixels of an image in parallel
  - CRAY’s vector processor, see image below

Slide Source: Klimovitski & Macri, Intel
SISD versus SIMD

Writing a compiler for SIMD architectures is VERY difficult (inter-thread communication complicates the picture...)

Slide Source: ars technica, Peakstream article
Multiple Instruction/Single Data

Not useful, not aware of any commercial implementation...

Multiple Instruction/Multiple Data

As of 2006, all the top 10 and most of the TOP500 supercomputers were based on a MIMD architecture

Multiple Instruction/Multiple Data

- The sky is the limit: each PU is free to do as it pleases
- Can be of either shared memory or distributed memory categories
HPC: Where Are We Today?
[Info lifted from Top500 website: http://www.top500.org/]

![Image of Top500 November 2010 rankings](image)

### 2010 Top 500 Rankings

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Location</th>
<th>Country</th>
<th>Cores</th>
<th>Rmax (FLOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tianhe-1A</td>
<td>NUDT/NBSC/Tianjin</td>
<td>China</td>
<td>186,368</td>
<td>2.57</td>
</tr>
<tr>
<td>2</td>
<td>Jaguar</td>
<td>DOE/SC/ORNL</td>
<td>USA</td>
<td>224,162</td>
<td>1.76</td>
</tr>
<tr>
<td>3</td>
<td>Nebulae</td>
<td>NSCC</td>
<td>China</td>
<td>120,640</td>
<td>1.27</td>
</tr>
<tr>
<td>4</td>
<td>Tsubame 2.0</td>
<td>TITech</td>
<td>Japan</td>
<td>73,278</td>
<td>1.19</td>
</tr>
<tr>
<td>5</td>
<td>Hopper</td>
<td>DOE/SC/LBNL</td>
<td>USA</td>
<td>153,408</td>
<td>1.05</td>
</tr>
</tbody>
</table>
Where Are We Today?
[Cntd.]

- **Abbreviations/Nomenclature**
  - MPP – Massively Parallel Processing
  - Constellation – subclass of cluster architecture envisioned to capitalize on data locality
  - MIPS – “Microprocessor without Interlocked Pipeline Stages”, a chip design of the MIPS Computer Systems of Sunnyvale, California
  - SPARC – “Scalable Processor Architecture” is a RISC instruction set architecture developed by Sun Microsystems (now Oracle) and introduced in mid-1987
  - Alpha - a 64-bit reduced instruction set computer (RISC) instruction set architecture developed by DEC (Digital Equipment Corporation was sold to Compaq, which was sold to HP)
Where Are We Today?

[Cntd.]

- How is the speed measured to put together the Top500?
  - Basically reports how fast you can solve a dense linear system

![Diagram of installation type over time]

**HPLINPACK**

A Portable Implementation of the High Performance Linpack Benchmark for Distributed Memory Computers

- Algorithm: recursive panel factorizations, multiple lookahead depths, bandwidth reducing swapping
- Easy to install, only needs MPI + BLAS or VSIPL
- Highly scalable and efficient from the smallest cluster to the largest supercomputers in the world

FIND OUT MORE AT [http://icl.eecs.utk.edu/hpl/](http://icl.eecs.utk.edu/hpl/)
Some Trends…

- Consequence of Moore’s law
  - Transition from a speed-based compute paradigm to a concurrency-based compute paradigm

- Amount of power for supercomputers is a showstopper
  - Example:
    - Exascale Flops/s rate: reach it by 2018
    - Budget constraints: must be less than $200 million
    - Power constraints: must require less than 20 MW hour
  - Putting things in perspective:
    - World’s (China’s) fastest supercomputer: 4.04 Mwatts for 2.57 Petaflop/s
    - Oak Ridge Jaguar’s – US fastest supercomputer: 7.0 Mwats for 1.76 Petaflop/s
    - Faster machine for less power: the advantage of GPU computing
Parallel Programming Support (non-GPU)

- Message Passing Interface (MPI)
  - Originally aimed at distributed memory architectures, now very effective on shared memory

- OpenMP

- Threads
  - Pthreads ("P" comes from Posix)
  - Cell threads

- Parallel Libraries
  - Intel’s Thread Building Blocks (TBB) - mature
  - Microsoft’s Task Parallel Library - mature
  - SWARM (GTech) – small scope
  - STAPL (Standard Template Adaptive Parallel Library, B. Stroustrup Texas A&M) – undergoing effort
GPU Parallel Programming Support

- **CUDA (NVIDIA)**
  - C/C++ extensions

- **Brook (Stanford)**
  - Relies on language extensions
    - Draws on OpenGL v1.3+, DirectX v9+ or AMD's Close to Metal for the computational backend
    - Runs on Windows and Linux

- **Brook+ (AMD/ATI)**
  - AMD-enhanced implementation of Brook

- **SH (Waterloo)**
  - Became RapidMind, commercial venture, acquired in 2009 by Intel
  - Library and language extensions
  - Works on multicores as well

- **PeakStream**
  - Now defunct, acquired by Google, June 2007
Why Dedicate So Much Time to GPU?

- It’s fast for a variety of jobs
  - Really good for data parallelism (requires SIMD)
  - Bad for task parallelism (requires MIMD)

- It’s cheap to get one ($120 to $480)

- It’s everywhere
  - There is incentive to produce software since there are many potential users of it…
GPU Proved Fast in Several Applications

- **146X**
  - Medical Imaging
  - U of Utah

- **36X**
  - Molecular Dynamics
  - U of Illinois, Urbana

- **18X**
  - Video Transcoding
  - Elemental Tech

- **50X**
  - Matlab Computing
  - AccelerEyes

- **100X**
  - Astrophysics
  - RIKEN

- **149X**
  - Financial simulation
  - Oxford

- **47X**
  - Linear Algebra
  - Universidad Jaime

- **20X**
  - 3D Ultrasound
  - Techniscan

- **130X**
  - Quantum Chemistry
  - U of Illinois, Urbana

- **30X**
  - Gene Sequencing
  - U of Maryland
CPU vs. GPU – Flop Rate (GFlop/Sec)

- Single Precision
- Double Precision

- Tesla 8-series
- Tesla 10-series
- Tesla 20-series
- Westmere 3 GHz
- Nehalem 3 GHz
GPU vs. CPU – Memory Bandwidth

[GB/sec]

GB/Sec

2003 2004 2005 2006 2007 2008 2009 2010

Tesla 20-series
Tesla 10-series
Tesla 8-series
Nehalem 3 GHz
Westmere 3 GHz

[93x55] GPU vs. CPU – Memory Bandwidth [GB/sec]
[495x702] 41
[460x182] 20
[389x182] 40
[354x182] 60
[318x174] 80
[282x174] 100
[247x174] 120
[211x174] 140
[2003 2004 2005 2006 2007 2008 2009 2010]

[444x619] Tesla 10-series
[430x453] Tesla 20-series
[414x531] Tesla 20-series
[347x153] [GB/sec]
## Key Parameters

### GPU, CPU

<table>
<thead>
<tr>
<th></th>
<th>GPU – NVIDIA Tesla C2050</th>
<th>CPU – Intel core i7 975 Extreme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Cores</td>
<td><strong>448</strong></td>
<td><strong>4</strong></td>
</tr>
<tr>
<td>Memory</td>
<td>3 GB</td>
<td>- 32 KB L1 cache / core</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 256 KB L2 (I&amp;D)cache / core</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 8 MB L3 (I&amp;D) shared by all cores</td>
</tr>
<tr>
<td>Clock speed</td>
<td>1.15 GHz</td>
<td>3.20 GHz</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>140 GB/s</td>
<td>32.0 GB/s</td>
</tr>
<tr>
<td>Floating point</td>
<td><strong>515 x 10^9</strong> Double</td>
<td><strong>70 x 10^9</strong> Double</td>
</tr>
<tr>
<td>operations/s</td>
<td>Precision</td>
<td>Precision</td>
</tr>
</tbody>
</table>
IBM BlueGene/L

- Entry model: 1024 dual core nodes
- 5.7 Tflop/s
- Linux OS
- Dedicated power management solution
- Dedicated IT support
- Only decent options for productivity tools (debugging, profiling, etc.)
  - TotalView
- Price (2007): $1.4 million