Lecture 1

Introduction
Welcome to CSE 260!

• Your instructor is Scott Baden
  - baden@ucsd.edu
  - Office hours in EBU3B Room 3244
    - Today after class, will set permanent time thereafter

• The class home page is
  http://cseweb.ucsd.edu/classes/wi15/cse262-a

• Course Resources
  - Piazza
  - Stampede @ TACC – request an XSEDE portal account
Readings

- Assigned class readings will consist of on-line material
- There is no required text
- However, these cse 260 texts may be useful
  - *An Introduction to Parallel Programming*, by Peter Pacheco, Morgan Kaufmann (2011)
- Lecture slides

  cseweb.ucsd.edu/classes/wi15/cse262-a/lectures.html
Course Requirements

• Do the readings **before** lecture and be prepared to discuss them in class
• 4-5 assignments
  ◆ #1 (Questionnaire) has been posted
  ◆ #2 is a programming lab, due at end of Week3
• Project – 3 in-class presentations
• A2 and project
  ◆ Teams of 2 or 3, teams of 1 with permission
  ◆ Find a partner via “looking for a partner” Piazza folder
What you’ll learn in this class

• How to solve computationally intensive problems on heterogeneous (hybrid) computers effectively
  - **Software techniques**: run time support, data driven execution, load balancing
  - **Performance tradeoffs**: latency hiding, managing locality within complicated memory hierarchies
  - **Algorithms**: communication avoiding (CA), irregular problems

• Assumes that you have a prior background in parallel computation
Background

• CSE 260 or another parallel computation course?
• CSME
• MPI
• CUDA
• RPC
• Multithreading: pthreads or C++11
• Sparse factorization

\[
\nabla \cdot u = 0
\]

\[
\frac{D\rho}{Dt} + \rho(\nabla \cdot v) = 0
\]

\[
f(a) + \frac{f'(a)}{1!} (x - a) + \frac{f''(a)}{2!} (x - a)^2 + \ldots
\]
Class presentation technique

- Learning is not a passive process
- I won’t just “broadcast” the material
- Consider the slides as talking points, class discussions driven by your interest
- Class participation is important to keep the lecture active
- Complete the assigned readings before class
- Different lecture modalities
  - The 2 minute pause
  - In class problem solving
The 2 minute pause

• **Opportunity in class** to develop understanding, make sure you “grok” it
  - By trying to explain to someone else
  - Getting your brain actively working on it

• **What will happen**
  - I pose a question
  - You discuss with 1-2 people around you
    - Most important is your understanding of why the answer is correct
  - After most people seem to be done
    - I’ll ask for quiet
    - A few will share what their group talked about
      - Good answers are those where you were wrong, then realized…
Policies

• Academic Integrity
  - Do you own work
  - Plagiarism and cheating will not be tolerated

• By taking this course, you agree to abide by the following the course polices:

  http://cseweb.ucsd.edu/classes/wi15/cse262--a
The rest of the lecture

- Technological disruption and its consequences
- Exascale
- A brief overview of parallel computation
What is parallel processing?

• Decompose a workload onto *simultaneously executing physical resources*

• Multiple processors co-operate to process a related set of tasks – *tightly coupled*

• Improve some aspect of performance
  - Speedup: 100 processors run $\times 100$ faster than one
  - Capability: Tackle a larger problem, more accurately
  - Algorithmic, e.g. search
  - Locality: more cache memory and bandwidth
Parallel Processing, Concurrency & Distributed Computing

- **Parallel processing**
  - Performance (and capacity) is the main goal
  - More tightly coupled than distributed computation

- **Concurrency**
  - Concurrency control: serialize certain computations to ensure correctness, e.g. database transactions
  - Performance need not be the main goal

- **Distributed computation**
  - Geographically distributed
  - Multiple resources computing & communicating unreliably
  - “Cloud” computing, large amounts of storage
  - Looser, coarser grained communication and synchronization

- May or may not involve separate physical resources, e.g. multitasking “Virtual Parallelism”
Granularity

• A measure of how often a computation communicates, and what scale
  - Distributed computer: a whole program
  - Multicomputer: function, a loop nest
  - Multiprocessor: + memory reference
  - Multicore: similar to a multiprocessor but perhaps finer grained
  - GPU: kernel thread
  - Instruction level parallelism: instruction, register
The impact of technology
### Supercomputer Evolution

**Transformational performance: \( \times 10^3 \) /10 years**

<table>
<thead>
<tr>
<th>Year</th>
<th>Supercomputer</th>
<th>Performance</th>
<th>Power Consumption</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1976</td>
<td>Cray-1</td>
<td>240 MFlops</td>
<td>115KW</td>
<td>2KF/W</td>
</tr>
<tr>
<td>1996</td>
<td>ASCI Red</td>
<td>1TFlop</td>
<td>850KW</td>
<td>1MF/W</td>
</tr>
<tr>
<td>2008</td>
<td>Roadrunner</td>
<td>1 PFlop</td>
<td>2.35 MW</td>
<td>0.4GF/W</td>
</tr>
<tr>
<td>2013</td>
<td>Tianhe-2</td>
<td>33.9 PFlops</td>
<td>17.6 (24) MW</td>
<td>1.9 GF/W</td>
</tr>
<tr>
<td>1985-90</td>
<td>Cray-2</td>
<td>240 MFlops</td>
<td>195KW</td>
<td>2KF/W</td>
</tr>
<tr>
<td>2013</td>
<td>Ipad-3 today</td>
<td>1 TFlop</td>
<td>850KW</td>
<td>1MF/W</td>
</tr>
</tbody>
</table>

Cray-2: 1985-90, 1.9 GFlops, 195 KW (10 KF/W), Ipad-3 today (10W)

\( 10^{18} \) ca. 2022 \( \leq 20 \text{MW} \)

Scott B. Baden / CSE 262 / UCSD, Wi '15
Performance Trends

PERFORMANCE DEVELOPMENT

1 Eilop/s
100 Piop/s
10 Piop/s
1 Piop/s
100 Tlop/s
10 Tlop/s
1 Tlop/s
100 Glop/s
10 Glop/s
1 Glop/s


SUM
N=1
N=500

250 Piop/s
33.9 Piop/s
118 Piop/s

top500.org
10^{18}

- What is an exaflop? 1M Gigaflops!
- The first Exascale computer ~ 2020
  - 20+ MWatts: 50+ GFlops/Watt
- Top 500 #1, Tianhe-2: 1.9 GF/W (17.6MW)
  - Exascale extrapolation: 521 MW
    #49 Green500 [Does not include 6.2 MW extra]
  - #1 on the green list delivers 4.4 GF/W
    GSIC Center Tokyo Inst Tech, TSUBAME-KFC
    Xeon E5 and NVIDIA K20x
- Exascale adds a significant power barrier
- Per capita power consumption in the EU (IEA) in 2009: 0.77kW [Tianhe-2 ~ 20K]
Consequences of evolutionary disruption

- **New capabilities:** modelling, healthcare…
- Changes the common wisdom for solving a problem including the implementation
- Simplified processor design, but more user control over the hardware resources
Today’s mobile computer would have been yesterday’s supercomputer

- Cray-1 Supercomputer
- 80 MHz processor
- 240 Mflops/sec peak
- 3.4 Mflops Linpack
- 8 Megabytes memory
- Water cooled
- 1.8m H x 2.2m W
- 4 tons
- Over $10M in 1976

- iPad Air 2
  - Apple A8X SoC
    - 1.5GHz 3-core Apple swift processor (4600 Megaflops)
    - PowerVR, 8-core GXA6850 GPU (230 Gflops)
    - 3 Billion transistors
  - 2GB RAM, 32KB L1, 2MB L2, 4MB L3
  - Up to 128GB Flash storage
  - Color display
  - Wireless or phone Networking
  - Air cooled
  - ~ 6.1 x 170 x 240 mm, 437 g
  - $699 in January 2015

www.anandtech.com/show/8716/apple-a8xs-gpu-gxa6850-even-better-than-i-thought

Scott B. Baden / CSE 262 / UCSD, Wi '15 20
Technological trends of scalable HPC systems

- Performance increases coming from improvements at the node level
- Hybrid processors
- Complicated software-managed parallel memory hierarchy
- Memory/core is shrinking
- Communication costs increasing relative to computation

2x/ 3-4 years

Peak performance [Top500, 13]
The age of the multi-core processor

- On-chip parallel computer
- IBM Power4 (2001), many others follow (Intel, AMD, Tilera, Cell Broadband Engine)
- First dual core laptops (2005-6)
- GPUs (nVidia, ATI): desktop supercomputer
- In smart phones, behind the dashboard: blog.laptopmag.com/nvidia-tegrak1-unveiled
- Everyone has a parallel computer at their fingertips
- If we don’t use parallelism, we lose it!

realworldtech.com
The GPU

- Specialized many-core processor
- Massively multithreaded, long vectors
- Reduced on-chip memory per core
- Explicitly manage the memory hierarchy
Why is parallelism inevitable?

- Physical limits on processor clock speed and heat dissipation
- A parallel computer increases memory capacity and bandwidth as well as the computational rate
Performance and Implementation Issues

• Because data motion costs growing relative to computation
  ‣ Conserve locality
  ‣ Hide latency

• Little’s Law [1961]

\[ \text{# threads} = \text{performance} \times \text{latency} \]

\[ T = p \times \lambda \]

• \( p \) and \( \lambda \) increasing with time
  \( p = 1 - 8 \text{ flops/cycle} \)
  \( \lambda = 500 \text{ cycles/word} \)
Have you written a parallel program?

- Threads (C++11 or pthreads)
- MPI
- RPC
- CUDA
Motivating Applications
A Motivating Application - TeraShake

Simulates a 7.7 earthquake along the southern San Andreas fault near LA using seismic, geophysical, and other data from the Southern California Earthquake Center

epicenter.usc.edu/cmeportal/TeraShake.html
How TeraShake Works

• Divide up Southern California into *blocks*
• For each block, get all the data about geological structures, fault information, …
• Map the blocks onto processors of the supercomputer
• Run the simulation using current information on fault activity and on the physics of earthquakes
Animation
The Payoff

• Capability
  ◆ We solved a problem that we couldn’t solve before, or under conditions that were not possible previously

• Performance
  ◆ Solve the same problem in less time than before
  ◆ This can provide a capability if we are solving many problem instances

• The result achieved must justify the effort
  ◆ Enable new scientific discovery
  ◆ Software costs must be reasonable
I increased performance – so what’s the catch?

• A well behaved single processor algorithm may behave poorly on a parallel computer, and may need to be reformulated numerically

• Currently, there currently exists no tool that can convert a serial program into an efficient parallel program

  … for all applications … all of the time… on all hardware

• **Performance programming** issues
  ‣ Data motion and locality
  ‣ Load balancing
  ‣ Serial sections
  ‣ Low level details
Performance differs across application domains

- Collela’s 7 dwarfs, patterns of communication and computation that persist over time and across implementations
  - Structured grids
    - Panfilov method
  - Dense linear algebra
    - Matrix multiply, Vector-Mtx Mpy
      Gaussian elimination
  - N-body methods
  - Sparse linear algebra
    - In a sparse matrix, we take advantage of knowledge about the locations of non-zeros, improving some aspect of performance
  - Unstructured Grids
  - Spectral methods (FFT)
  - Monte Carlo
Application-specific knowledge is important

- The more we know about the application…

  … specific problem … math/physics … initial data …
  … context for analyzing the output…
  … the more we can improve performance

- Particularly challenging for irregular problems
- Parallelism introduces many new tradeoffs
  - Redesign the software
  - Rethink the problem solving technique
The two mantras for high performance

- Domain-specific knowledge is important in optimizing performance, especially locality.

- Significant time investment for each 10-fold increase in a performance.