Administrivia

- IPython notebooks from Wednesday are posted.
- Please finish HW0 reviews by next Tuesday.
- HW1 out.
  - AWS setup poll (in-class).
- Next Reading coming soon.
Analysis of Parallel Programs
Two measures:

Speedup
&
Scaling
Speedup

- $T_S =$ Time to run a program on a single processor.
- $T_P =$ Time to run on P processors.
- Speedup for P processors is $T_S / T_P$
• 1 minute load time, 9 minutes compute time on a single processor.

• On 4 processors, 1 minute load time and 3 minutes compute:

• Speedup for P=4 is $(10 \div 4)$ or 2.5.
Amdahl's Law

If \((1 / N)\) of compute time is in serial code, maximum speedup is \(N\) on any number of processors.
Gustafson-Barsis’s Law

If $\frac{1}{N}$ of compute time is in serial code,

speedup on $P$ processors is

$$S(P) = P - \frac{1}{N} \cdot (P - 1)$$

?!
Amdahl

Time

N = 5 or more
Gustafson-Barsis

Serial
Non-serial

Time

N = 1
Wait a second, I have 2 computers. Why am I solving a problem I can solve with N = 1?
Gustafson-Barsis

Time

N = 2
Gustafson-Barsis

Time

N = 4
Scalability of Parallel Algorithms

• Two ways to think about how programs scale:
  - Strong scaling: Keep problem the same, add processors.
  - Weak scaling: Problem sizes grows linearly with number of processors.

• Saying that an algorithm “scales” doesn’t tell you which way.
Strong Scaling

- Speedup on the same size problem.
- Perfect Strong Scaling: Speedup of $P$ on $P$ processors.
- Typically, small data but computationally intense.
- At some point, has to break down.
Strong Scaling Examples

- Large Matrix Multiplication, Fluid Dynamics, Weather Simulation, Molecular Dynamics, Fluid Dynamics….
- Image processing & Graphics
- Anything “Embarassingly Parallel”
Weak Scaling

- Problem grows proportionally to processors.
- Note that “proportionally” is not well-defined.
  - What is double an NxN matrix multiply?
    - $2N \times 2N$ — double N
    - $1.4 N \times 1.4 N$ — double entries
    - $1.26 N \times 1.26 N$ — double operations
Weak Scaling Examples

- Large-scale data analyses.
- Machine Learning
- Image Processing and Graphics (also Strong)
- “Web-scale” analysis
Demonstrating strong scaling requires solving a fixed problem size faster and faster, while demonstrating weak scaling requires solving an increasing problem size within a fixed time budget. As their names imply, demonstrating strong scaling is typically more challenging than demonstrating weak scaling.

**Amdahl's Law**

Amdahl's Law, originally articulated by Gene Amdahl in 1967, provides an upper bound on the speedup to be expected from a particular optimization. Assuming an optimization accelerates some fraction of the program's total runtime ($f$) by a factor of $s_f$ (for speedup).

Amdahl's Law states that the maximum overall speedup is:

$$\text{Speedup} = \frac{1}{1 - \frac{1}{f}} + \frac{f}{s_f}$$

The key consequence of Amdahl's Law is that the overall speedup of an optimization is limited by the non-optimizable portion of an application's execution time.

**Gustafson's Law**

Whereas Amdahl's Law assumes a fixed problem size, Gustafson's Law, stated by John L. Gustafson in 1988, quantifies speedup if the problem size is allowed to increase. We derive Gustafson's Law here.

In the context of parallelization, serial work quickly dashes hopes of demonstrating strong scaling. For an exploration of Amdahl's implications for multicore architectures, see related reading "Amdahl's Law in the Multicore Era".

Martha A. Kim
Weak / Strong vs. Cluster / Multicore

• Statements:
  • Weak scaling algorithms are better for clusters.
  • Strong scaling algorithms are better for multicore.

• Are these correct? Why? At what problem sizes?
Next Time

• Python Multiprocessing

• On the Horizon:
  • Python multithreading
  • The dreaded Global Interpreter Lock (GIL)
  • Low-level multithreading in Python