Logistics

• NERSC accounts should be online soon
  ➢ You should have received an email from them
• SCL accounts are online.

• You should have read chapter 7 by now.
  ➢ 😊

• Homework #3 is posted
Logistics [2]

Are there any Questions??
Performance

• A key idea regarding high-end computing technology and applications.

• Several kinds of performance can be measured and all are important.
  ➢ Computing
  ➢ Networking
  ➢ Number of tasks (transactions) computed in time
  ➢ Time to solution.

• Distributed Computing focuses on
  ➢ Computing, Networking, Number of transactions

• Parallel Computing focuses on
  ➢ Computing, Networking, Time to solution.
Performance Attributes

• Definitions
  ➢ Machine Size
  ➢ Sequential Time
  ➢ Parallel Time
  ➢ Speed
  ➢ Peak Speed
  ➢ Speed Up
  ➢ Efficiency
  ➢ Startup time
  ➢ Bandwidth
  ➢ Asymptotic Bandwidth

- Definitions

  - Machine Size: $n$, $N_p$, $N_{proc}$
  - Sequential Time: $T_1$, $T_s$
  - Parallel Time: $T_n$, $T_p$
  - Speed: $P_n$, $R_n$, MFR = flops/time
  - Peak Speed: $P_{peak}$, $R_{macho}$
  - Speed Up: $S_n = T_s / T_p = T_1 / T_n$
  - Efficiency: $E_n = S_n / n = S_n / N_p$
  - Startup time: $t_0$, $\tau_0$
  - Bandwidth: $r$
  - Asymptotic Bandwidth: $r_{\infty}$
Prediction vs. Analysis

• The programming models we looked at earlier are used for “prediction”
  ➢ PRAM
  ➢ BSP
  ➢ Phase Parallel

• Some are more complex than others.

• Now we focus on the Analysis of data from similar models.
  ➢ Amdahl’s Law
  ➢ Gustafson-Barisis’s Law
  ➢ Karp-Flatt Metric
The Overview

• **Amdahl’s Law**
  - Decide whether a program merits a parallel implementation.

• **Gustafson-Barsis’s Law**
  - Performance of a parallel program.

• **Karp-Flatt metric**
  - What is the principle barrier to performance
    - Sequential code?
    - Parallel overhead?
What is Speedup

• The ratio between sequential execution time and parallel execution time.

\[ S_n = \frac{T_s}{T_p} = \frac{T_1}{T_n} \]

• The operations that contribute to the overall time of an executing program.
  - Computations of sequential code \( \sigma(n) \)
  - Computations of parallel code \( \varphi(n) \)
  - Parallel Overhead \( \kappa(n,p) \)
    - Communication
    - Redundant computations
    - Extra computations
What is Speedup [2]

• $T_s = \sigma(n) + \varphi(n)$

  ➢ $n$ is the problem size

• $T_p = \sigma(n) + \varphi(n)/p + \kappa(n,p)$

  ➢ $p$ is the number of processes or threads
    ▪ The number parallel execution entities involved.

• $S_n = [\sigma(n) + \varphi(n)]/[\sigma(n) + \varphi(n)/p + \kappa(n,p)]$
• $= ? (n,p)$
• $= is optimistic! WHY?$
Some Assumptions/Comments

- **No load imbalance**
- **For Amdahl’s Law** $\kappa(n,p)$ is zero 😊
  - No overhead
  - No communications 😊
- **For a fixed size problem.**
  - More processes/threads decreases the computation time.
    - Not necessarily in an ideal fashion
  - More processes/threads increases overhead and communications.
    - Unfortunately
Poor Scaling

![Graph showing speedup against processors, illustrating poor scaling.](image)
Good Scaling

![Graph showing good scaling with speedup increasing linearly with the number of processors. The black line represents ideal scaling, while the red line shows actual speedup.](image-url)
Efficiency

- Efficiency is the ratio of the speedup to the number of processors used.
  - A measure of the processor utilization.

- \( E_n = S_n/p = T_s/T_p/p \)

- \( = \left[ \sigma(n) + \varphi(n) \right]/\left[ \sigma(n) + \varphi(n)/p + \kappa(n,p) \right]/p \)

- \( = \left[ \sigma(n) + \varphi(n) \right]/\left[ p\sigma(n) + \varphi(n) + p\kappa(n,p) \right] \)

- \( = \varepsilon(n,p) \)
Efficiency

Measured vs Ideal Efficiency

- Efficiency
- Ideal
Efficiency [2]

Measured vs Ideal Efficiency

Efficiency vs Processors
Efficiency [3]

Measured vs Ideal Efficiency

Efficiency vs Processors
Amdahl’s Law

\[
f = \frac{\sigma(n)}{\sigma(n) + \varphi(n)}
\]

Serial Fraction

\[
\psi(n, p) \leq \frac{1}{f + (1 - f) / p}
\]
Examples

- You have a code that spends 93% of its time in code that can be effectively parallelized
  - What is the speedup possible on 7 processors?

\[
\psi(n, p) \leq \frac{1}{0.07 + (1 - 0.07) / 7} = 4.93
\]

- What is the maximum speedup possible?

\[
\lim_{p \to \infty} \frac{1}{0.07 + (1 - 0.07) / p} = \frac{1}{0.07} = 14.3
\]
Amdahl Effect

• For a fixed number of processors speedup usually increases with the problem size.
  - Not getting good speedup increase the problem size and you will see better speedup!
• The overhead $\kappa(n,p)$ usually has a lower problem size and processor count dependence than the parallel algorithm component, $\varphi(n)$
• For example:
  - $\kappa(n,p) = O(500\log n + (n/25)\log p)$
  - $\varphi(n) = O(n^2)$
Speedup vs problem size

- Ideal
- n=1000
- n=5000
- n=10000

processors

speedup

0.00
0 1024 2048 3072 4096

2/9/2005 ComS 425 Spring 2005
Speedup vs problem size

- Red line: n=1000
- Blue line: n=5000
- Green line: n=10000
- Black line: Ideal

The graph shows the relationship between speedup and problem size across different processor counts. The x-axis represents the number of processors, ranging from 0 to 1024, while the y-axis represents speedup, ranging from 0.00 to 1024.00. The lines indicate how the speedup changes as the problem size increases with varying numbers of processors. The ideal line shows the expected linear increase in speedup with processor count.
Speedup vs problem size

- Speedup
- Processors
- n=1000
- n=5000
- n=10000
- Ideal
Gustafson-Barsis’s Law

- Treats time as the constant and motivation is the scaled problem size.
  - How big of a problem can I solve in a fixed amount of time.
  - Ignores $\kappa(n,p)$ the overhead
- Opposite of Amdahl’s Law.
- Assumes a fraction of time, $s$, spent in parallel code doing sequential operations

$$\psi (n, p) \leq p + (1 - p)s$$

- Often called Scaled Speedup.
- John Gustafson, ISU grad, SCL Scientist.
  - Now at Sun
Examples

• If you have a code that has a 10% serial execution time on 128 processors, what is the scaled speedup for the code?
  \[ 128 + (1-128) \times 0.10 = 115.3 \]

• What is the maximum fraction of serial operations that can exist in a code to achieve a speedup of 1000 for a 1024 processor system?
  \[ 1000 = 1024 + (1-1024)s \]
  \[ s = \frac{24}{1023} = 0.0235 \]
  \[ s = 2.35\% \]
Karp-Flatt Metric

• The experimentally determined serial fraction is defined as the ratio of the serial time plus the overhead, $\kappa(n,p)$, to the time to compute on one process with the parallel code.

$$ e = \frac{\sigma(n) + \kappa(n,p)}{T(n,1)} $$

$$ e = \frac{1/\psi - 1/p}{1 - 1/p} $$
Karp-Flatt Metric

- Takes into account the parallel overhead
- Can be used to detect other sources of overhead that simple models ignore.
  - Load imbalance
  - Communication
  - Synchronization
  - System constraints.
- For a fixed problem size the efficiency typically decreases.
  - If $e$ is roughly constant over a number of runs then there is a limited amount of parallelism.
  - If $e$ is increasing over a number of runs then there is some additional overhead as the number of processors scale.
### Examples

$$e = \frac{1/\psi - 1/p}{1 - 1/p}$$

<table>
<thead>
<tr>
<th>$p$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\psi (p)$</td>
<td>1.82</td>
<td>2.50</td>
<td>3.08</td>
<td>3.57</td>
<td>4.00</td>
<td>4.38</td>
<td>4.71</td>
</tr>
<tr>
<td>$e$</td>
<td>0.10</td>
<td>0.10</td>
<td>0.10</td>
<td>0.10</td>
<td>0.10</td>
<td>0.10</td>
<td>0.10</td>
</tr>
</tbody>
</table>
Examples

\[
e = \frac{1 \psi - 1}{1 - \frac{1}{p}}
\]

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>? (p)</td>
<td>1.87</td>
<td>2.61</td>
<td>3.23</td>
<td>3.73</td>
<td>4.14</td>
<td>4.46</td>
<td>4.71</td>
</tr>
<tr>
<td>( p )</td>
<td>1.87</td>
<td>2.61</td>
<td>3.23</td>
<td>3.73</td>
<td>4.14</td>
<td>4.46</td>
<td>4.71</td>
</tr>
<tr>
<td>( e )</td>
<td>0.070</td>
<td>0.075</td>
<td>0.080</td>
<td>0.085</td>
<td>0.090</td>
<td>0.095</td>
<td>0.100</td>
</tr>
</tbody>
</table>
For Homework #3

- You are going to be using a batch system.
- You will need to tell the batch system how much time you want to use for each job!!
- You know the relative ordering of your algorithms “sequentially”
  - Your ddot algorithm should be the slowest 😊
- How do you predict the timings needed ??
How do you predict Timings?

- **DDOT is** $O(N^3)$

  - IBM SP 500,500,500 time is 13.95 seconds
  - What is the time for 1000,1000,1000?
    - $13.95 \times (1000/500)^3 = 13.95 \times 2^3 = 13.95 \times 8 = 111.6$ seconds
    - Actual time is 124.83 seconds
  - What is time for 1500,1501,1499 ?
    - $13.95 \times (1500/500)(1501/500)(1499/500) = 376.65$
    - Actual time is 450.71 seconds
    - Or
      - $124.83 \times (1500/1000)(1501/1000)(1499/1000) = 421.30$
  - Accurate to 10% (over or under).
    - $(124.83/111.6) = 1.12$
    - $(450.71/421.30) = 1.07$