Outline

1. Analyzing Parallel Programs
   - Speedup
   - Amdahl’s Law and Gustafson-Barsis’s Law
   - Evaluating Parallel Algorithms
   - Example
Acknowledgements

Some material used in creating these slides comes from

Analyzing Parallel Programs

- Speedup
- Amdahl’s Law and Gustafson-Barsis’s Law
- Evaluating Parallel Algorithms
- Example
**Speedup**

- Speedup is defined as

\[
\text{Speedup on } p \text{ processors} = \frac{\text{sequential program execution time}}{\text{execution time on } p \text{ processors}}
\]

- Generally we want to use execution times obtained using the best available algorithm. The best algorithm for a sequential program may be different than that the best algorithm for a parallel program.
Quinn uses $\psi(n, p)$ to show speedup is a function of both problem size $n$ and number of processors $p$.

- $\sigma(n)$: time required by sequential computation
- $\varphi(n)$: time required for computation that is parallelizable
- $\kappa(n, p)$: time for parallel overhead (communication, barriers, etc.)

Using these parameters, the speedup $\psi(n, p)$ is bounded from above:

$$
\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n, p)}
$$

In the simplest case we want to drop the $\kappa(n, p)$ term. Since it is positive, we have that

$$
\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p}
$$

This also provides an upper bound on speedup, which we can think of as speedup in the absence of any parallel overhead.
## Analyzing Parallel Programs

- Speedup
- Amdahl’s Law and Gustafson-Barsis’s Law
- Evaluating Parallel Algorithms
- Example
First appeared in a paper by Gene Amdahl in 1967. Provides an upper bound on achievable speedup based on fraction of computation that must be done sequentially. Let \( 0 \leq f \leq 1 \) be this fraction and take \( \psi(p) \) to be the speedup independent of problem size \( n \). Then

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

This shows that speedup, even as the number of processors increases without bound, is limited by \( 1/f \). If 20% of the computation is inherently sequential, then speedup cannot be larger than \( 1/0.2 = 5 \) regardless of the number of processors.
Amdahl’s Law

Amdahl's law speedup predictions

- $f=0.1$
- $f=0.2$
- $f=0.3$
- $f=0.4$
- $f=0.5$

Number of processors vs. optimal speedup
Amdahl’s Law example

Suppose a serial program reads $n$ data from a file, performs some computation, and then writes $n$ data back out to another file. The I/O time is measured and found to be $4500 + n \mu sec$. If the computation portion takes $n^2/200 \mu sec$, what is the maximum speedup we can expect when $n = 10,000$ and $p$ processors are used?

Computing $f$ we find

$$f = \frac{4500 + 10000}{4500 + 10000 + 500000} = \frac{145}{5145}$$

so, by Amdahl’s Law,

$$\psi \leq \frac{1}{\frac{145}{5145} + \frac{5000}{5145p}} = \frac{5145}{145 + 5000/p}$$

This gives a maximum speedup of 6.68 on 8 processors and 11.27 on 16 processors.
Gustafson-Barsis’s Law

- Amdahl’s law focuses on speedup as a function of increasing the number of processors; i.e., “how much faster can we get a fixed amount of work done using \( p \) processors?” Sometimes the question is “how much more work can we get done in a fixed amount of time using \( p \) processors?”

- Let \( 0 \leq s \leq 1 \) be the fraction of execution time a parallel program spends executing sequential code. Then

\[
\psi \leq p + (1 - p)s
\]

- This is useful especially when there is no “serial” version of the program as we can estimate speedup over an equivalent sequential program even if no such program exists.

- The speedup predicted by Gustafson-Barsis’s Law is called *scaled speedup*.

- Need some way to figure out \( s \), perhaps a profiling tool.
Gustafson-Barsis’s Law example

A parallel program takes 134 seconds to run on 32 processors. The total time spent in the sequential part of the program was 12 seconds. What is the scaled speedup?

Here $s = 12/134$ so the scaled speedup is

$$32 + (1 - 32) \frac{12}{134} = 29.224$$

This means that the program is running approximately 29 times faster than the program would take to run on one processor, assuming it could run on one processor.
The laws compared

The Wikipedia page for Gustafson’s Law offers the following metaphor to contrast the two laws.

**Amdahl’s Law approximately suggests:** Suppose a car is traveling between two cities 60 miles apart, and has already spent one hour traveling half the distance at 30 mph. No matter how fast you drive the last half, it is impossible to achieve 90 mph average before reaching the second city. Since it has already taken you 1 hour and you only have a distance of 60 miles total; going infinitely fast you would only achieve 60 mph.

**Gustafson-Barsis’s Law approximately suggests:** Suppose a car has already been traveling for some time at less than 90mph. Given enough time and distance to travel, the car’s average speed can always eventually reach 90mph, no matter how long or how slowly it has already traveled. For example, if the car spent one hour at 30 mph, it could achieve this by driving at 120 mph for two additional hours, or at 150 mph for an hour.
Outline

1. Analyzing Parallel Programs
   - Speedup
   - Amdahl’s Law and Gustafson-Barsis’s Law
   - Evaluating Parallel Algorithms
   - Example
The above metrics ignore communication and other parallel overhead. When evaluating an approach to parallelizing a task, we next should include estimates of communication costs, which can dominate parallel program time.

Let $t_s$ be the time for a sequential version of the program and $t_p$ be the time for the parallel algorithm on $p$ processors. Then speedup is $t_s/t_p$. 
Parallel execution time $t_p$ can be broken down into two parts, computation time $t_{comp}$ and communication time $t_{comm}$.

$$t_p = t_{comp} + t_{comm}$$

Speedup is then

$$\psi = \frac{t_s}{t_p} = \frac{t_s}{t_{comp} + t_{comm}}$$

The computation/communication ratio is $t_{comp}/t_{comm}$. 

Message transfer time

Typically the time for communication can be broken down into two parts, the time $t_{\text{startup}}$ necessary for building the message and initiating the transfer, and the time $t_{\text{data}}$ required per data item in the message. At a first approximation this looks like

$$t = t_{\text{startup}} + m t_{\text{data}}$$

where $m$ is the number of data items sent.
A communication timing experiment

- The values of $t_{startup}$ and $t_{data}$ can be determined empirically
- Test program written that sends messages ranging in length from 100 to 10000 integers between two nodes
- Each message sent (and received) and then sent back (and received)
- Repeated 100 times
- Linear regression used to fit line to timing data recorded for upper 80% of message lengths
- Determined $t_{startup} = 88.25\ \mu\text{sec}$ and $t_{data} = 0.0415\ \mu\text{sec}$ per integer, which corresponds to 96.43 MB/s (assuming 4 bytes per integer)
Old workstation cluster timing data (Fall 2010)

Cluster message send time (MPI_INT)

message length vs. microseconds

- Message length increases from 0 to 10,000.
- Time in microseconds ranges from 0 to 600.

The graph shows a linear relationship between message length and time taken to send a message, indicating a consistent and predictable communication delay.

This data is crucial for understanding the performance characteristics of the cluster and for optimizing parallel applications.
Minor Prophets cluster latency and bandwidth

Cluster message send time (MPI_INT)
Latency: 83.0779 microseconds, Bandwidth: 87.6022 MB/s

Cluster message send time (MPI_INT)
Latency: 117.058 microseconds, Bandwidth: 94.9853 MB/s

Latency vs Bandwidth average (56 runs) on Minor Prophet Cluster
(error bars show 1 std. dev.)
LittleFe cluster latency and bandwidth

Cluster message send time (MPI INT)
Latency: 3415.77 microseconds, Bandwidth: 3.6193 MB/s

Cluster message send time (MPI INT)
Latency: 3520.49 microseconds, Bandwidth: 3.3913 MB/s

Latency vs Bandwidth average (56 runs) on LittleFe Cluster
(error bars show 1 std. dev.)

- MPICH 3.0.2
- OpenMPI 1.6.3
Cluster latency and bandwidth comparison

**Minor Prophets**

- Cluster message send time (MPI_INT)
- Latency: 83.0779 microseconds, Bandwidth: 87.6022 MB/s

**LittleFe**

- Cluster message send time (MPI_INT)
- Latency: 3415.77 microseconds, Bandwidth: 3.61983 MB/s
Cluster latency and bandwidth comparison

**Minor Prophets**

Latency vs Bandwidth average (56 runs) on Minor Prophet Cluster
(error bars show 1 std. dev.)

- Black circle: MPICH 3.0.2
- Red circle: OpenMPI 1.4.3
- Blue circle: OpenMPI 1.6.3

**LittleFe**

Latency vs Bandwidth average (56 runs) on LittleFe Cluster
(error bars show 1 std. dev.)

- Black circle: MPICH 3.0.2
- Blue circle: OpenMPI 1.6.3

Error bars show 1 standard deviation in data values averaged to produce plots.
Outline

1. Analyzing Parallel Programs
   - Speedup
   - Amdahl’s Law and Gustafson-Barsis’s Law
   - Evaluating Parallel Algorithms
   - Example
Example

Consider a parallel matrix-vector product where $A$ is an $n \times n$ matrix. We compute $y = Ax$ by having the rank 0 process broadcast $x$ and send $n/p$ rows of $A$ to all processes. Each process then computes its portion of the product and returns it to the rank 0 process.

- The time to broadcast $x$ is $t_{startup} + nt_{data}$. Since we broadcast, we don’t need to multiply this by $p$ (communication is done simultaneously).
- The time to send the rows of $A$ is $(n/p)(t_{startup} + nt_{data})$.
- Finally, the time needed to collect the result is $p(t_{startup} + (n/p)t_{data})$. 
Example

- The total communication time is given by

  \[ t_{\text{comm}} = t_{\text{startup}} + n t_{\text{data}} + (n/p)(t_{\text{startup}} + n t_{\text{data}}) + p(t_{\text{startup}} + (n/p)t_{\text{data}}) \]

- The computation time is given by \( t_{\text{comp}} = 2Kn^2/p \) where \( K \) is the floating point computation rate. Total time for parallel execution is \( t_p \):

  \[ t_p = (1 + n/p)(t_{\text{startup}} + n t_{\text{data}}) + p(t_{\text{startup}} + (n/p)t_{\text{data}}) + \frac{2Kn^2}{p} \]

- The serial execution time is \( t_s = 2Kn^2 \) so the speedup is

  \[ \psi = \frac{2Kn^2}{(1 + n/p)(t_{\text{startup}} + n t_{\text{data}}) + p(t_{\text{startup}} + (n/p)t_{\text{data}}) + \frac{2Kn^2}{p}} \]
Not surprisingly, when using the value of $t_{\text{startup}}$ from above and $t_{\text{data}}$ as twice the number above (8 bytes for MPI\_DOUBLE rather than 4 bytes for MPI\_INT), and assuming a computation rate of 200 MFLOPS, the speedups are all less than 0.8 for matrix sizes up to $1000 \times 1000$ when 16 or fewer processors are used. If, however, we take out the communication time necessary to send the matrix rows, we obtain speedups of close to 5 with 16 processors.

Interestingly, if we assume that when the partial results are communicated back to the rank 0 process using a gather operation such that the transmissions can occur simultaneously, things improve again, with a speedup of nearly 6 with 16 processors.