Performance Metrics, Prediction, and Measurement

CPS343

Parallel and High Performance Computing

Spring 2020

Outline

Outline

Speedup

- Speedup is defined as

$$\text{Speedup on } N \text{ processors} = \frac{\text{sequential execution time}}{\text{execution time on } N \text{ processors}} = \frac{t_{\text{seq}}}{t_{\text{par}}}$$

- Generally we want to use execution times obtained using the best available algorithm.

  The best algorithm for a sequential program may be different that the best algorithm for a parallel program.

Efficiency

- Efficiency is defined as

$$\text{Efficiency} = \frac{\text{speedup}}{N} = \frac{t_{\text{seq}}}{N \cdot t_{\text{par}}}$$

- Given this definition, we expect

$$0 \leq \text{efficiency} \leq 1$$

Speedup

- We can write $\psi(n, N)$ for speedup to show it is a function of both problem size n and number of processors N.
- Definitions:
  - $\sigma(n)$: time required computation that is not parallelizable
  - $\varphi(n)$: time required for computation that is parallelizable
  - $\kappa(n, N)$: time for parallel overhead (communication, barriers, etc.)
- Note that
  - $t_{\text{seq}} = \sigma(n) + \varphi(n)$
  - $t_{\text{par}} = \sigma(n) + \varphi(n)/N + \kappa(n, N)$
- Using these parameters, the speedup $\psi(n, N)$ given by:

$$\psi(n, N) = \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/N + \kappa(n, N)}$$

Speedup

- Since $\kappa(n, N) \geq 0$, if it is dropped (i.e. assume there is no parallel overhead), we obtain an upper bound on speedup

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

- Note: we will often not explicitly write n, so it is common to see $\psi(N)$ for speedup for a given problem size (i.e., fixed n).

Outline

Amdahl's Law

First appearing in a paper by Gene Amdahl in 1967, this provides an upper bound on achievable speedup based on the fraction of computation that can be done in parallel.

- T is the time needed for an application to execute on a single CPU.
- α is the fraction of the computation that can be done in parallel...
- ...so $1 - \alpha$ is the fraction that must be carried out on a single CPU.

If we ignore parallel overhead we have

$$\psi(N) = \frac{t_{\text{seq}}}{t_{\text{par}}} \leq \frac{T}{(1-\alpha)T + \alpha(T/N)}$$

Amdahl's Law

Simplifying we have

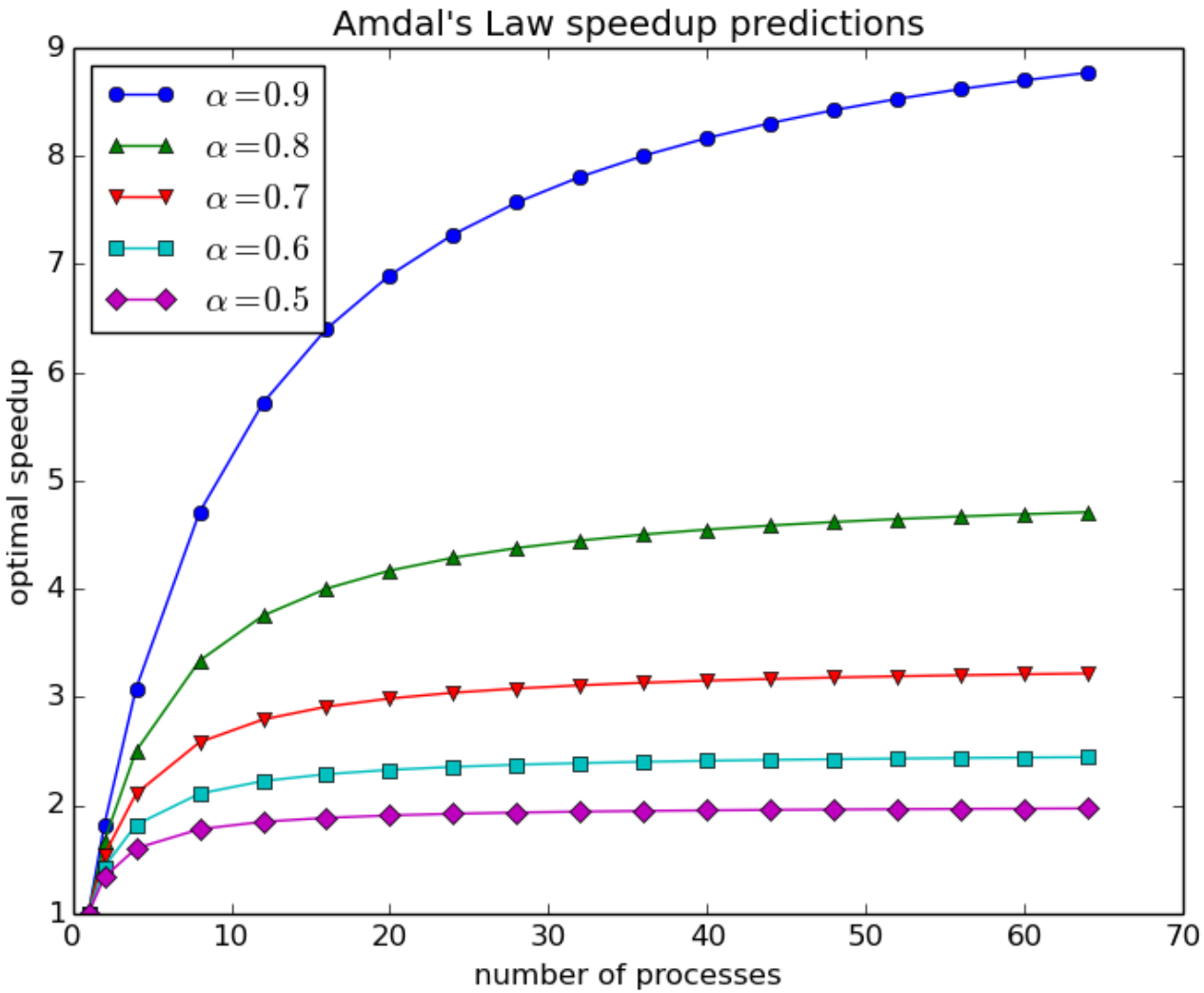
$$\psi(N) = \frac{t_{\text{seq}}}{t_{\text{par}}} \leq \frac{1}{(1-\alpha) + \alpha/N}$$

This is **Amdahl's Law**. Note that is says something rather discouraging: Even as $N \to \infty$ (i.e., the number of processors increases without bound), speedup is bounded by

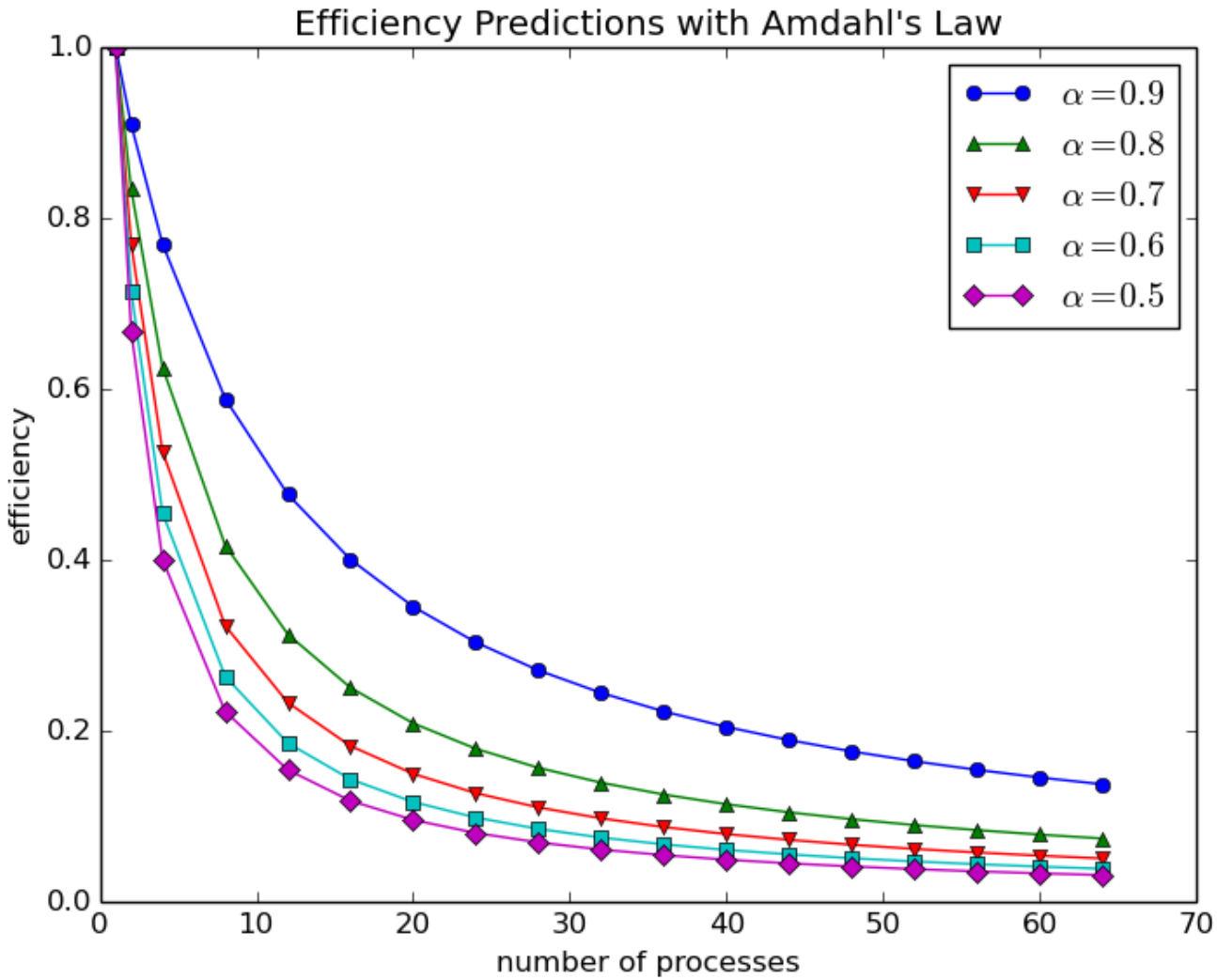
$$\psi \leq \frac{1}{1-\alpha}.$$

For example, if 90% of the computation can be parallelized so $\alpha = 0.9$, the speedup cannot be larger than $1/0.1 = 10$, regardless of the number of processors!

Amdahl's Law speedup prediction



Amdahl's Law efficiency prediction



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 $4500 + n$ μsec. If the computation portion takes $n^2/200$ μsec, what is the maximum speedup we can expect when n=10,000 and N processors are used?

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 $4500 + n$ μsec. If the computation portion takes $n^2/200$ μsec, what is the maximum speedup we can expect when n=10,000 and N processors are used?

Assume that the I/O must be done serially but that the computation can be parallelized. Computing α we find

$$\alpha = \frac{n^2/200}{(4500 + n) + n^2/200} = \frac{500000}{4500 + 10000 + 500000} = \frac{5000}{5145} \approx 0.97182$$

so, by Amdahl's Law,

$$\psi \leq \frac{1}{\left(1 - \frac{5000}{5145}\right) + \frac{5000}{5145N}} = \frac{5145}{145 + 5000/N}$$

This gives a maximum speedup of 6.68 on 8 processors and 11.27 on 16 processors.

The Gustafson-Barsis Law

Amdahl's Law reflects a particular perspective:

> *"We have a sequential program and want to figure out what speedup is attainable by parallelizing as much of it as possible."*

It turns out that focusing on parallelizing a sequential program is *not* a particularly good way to estimate how much speedup is possible.

This is because parallel implementations often look very different than a sequential implementation for the same program.

The Gustafson-Barsis Law

Amdahl's Law reflects a particular perspective:

> *"We have a sequential program and want to figure out what speedup is attainable by parallelizing as much of it as possible."*

It turns out that focusing on parallelizing a sequential program is *not* a particularly good way to estimate how much speedup is possible.

This is because parallel implementations often look very different than a sequential implementation for the same program.

A more useful characterization comes from consider the problem the other way around:

> *"We have a parallel program and want to figure out how much faster it is than a sequential program doing the same work."*

The Gustafson-Barsis 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 N processors?" Sometimes a better question is "how much **more work** can we get done in a *fixed amount of time* using N processors?"
- Let T be the total time a parallel program requires when using N processors. As before, let $0 \leq \alpha \leq 1$ be the fraction of execution time the program spends executing code in parallel. Then

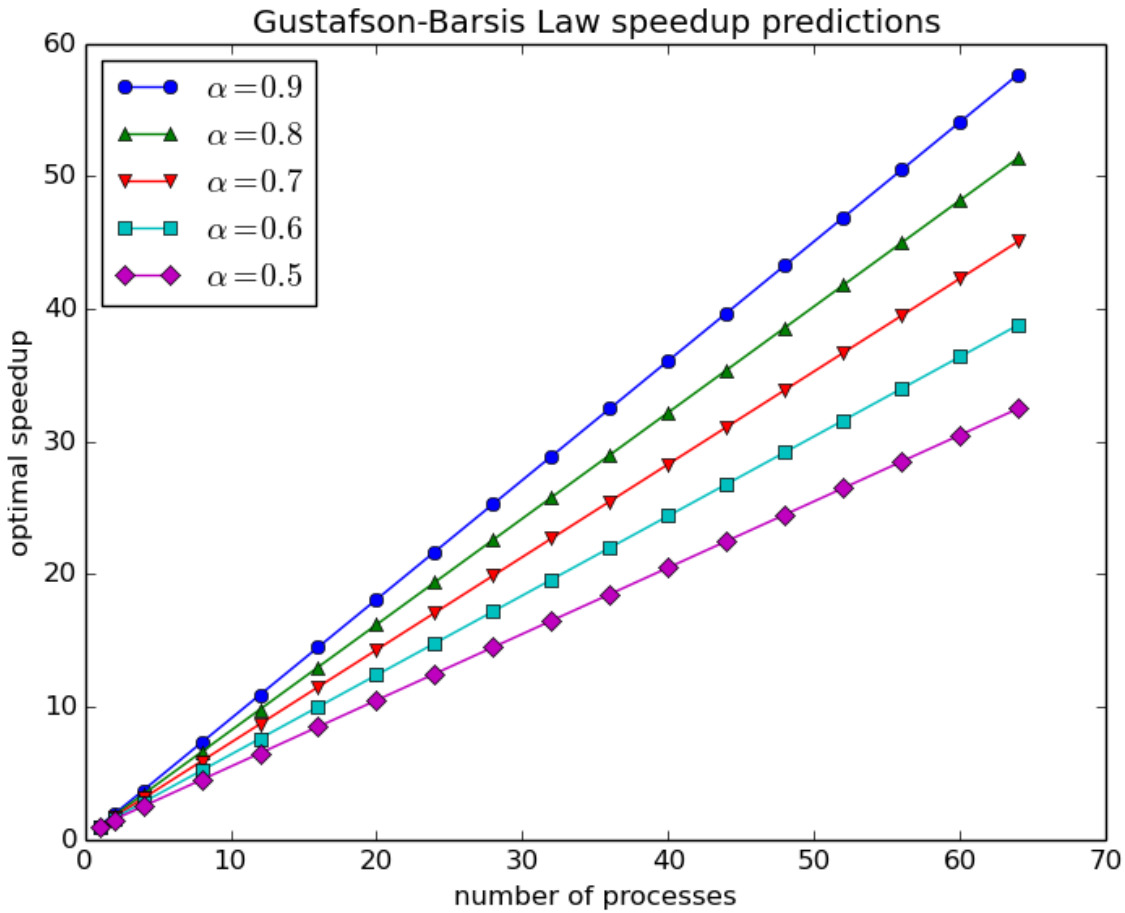
$$t_{\text{seq}} = (1 - \alpha)T + \alpha \cdot T \cdot N$$

so

$$\psi \leq \frac{t_{\text{seq}}}{t_{\text{par}}} = \frac{(1 - \alpha)T + \alpha \cdot T \cdot N}{T} = (1 - \alpha) + \alpha N$$

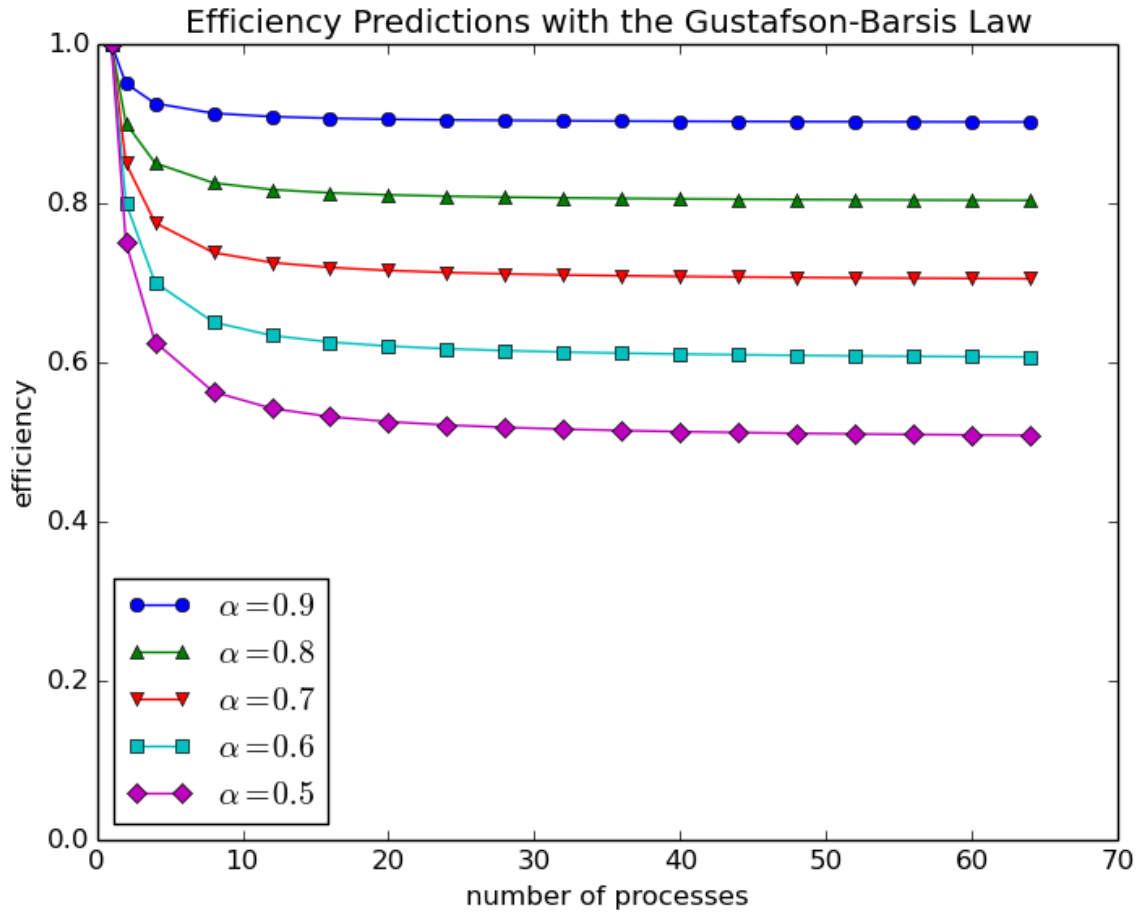
- This is the **Gustafson-Barsis Law** (1988).
- The speedup estimate it produces is sometimes called *scaled speedup*.

Gustafson-Barsis Law speedup prediction



This is much more encouraging than what Amdahl's Law showed us.

Gustafson-Barsis Law efficiency prediction



Again, this is much more encouraging!

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?

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 $\alpha = (134 - 12)/134 = 122/134$ so the scaled speedup is

$$(1 - \alpha) + \alpha N = \left(1 - \frac{122}{134}\right) + \frac{122}{134} \cdot 32 = 29.224$$

This means that the program is running approximately 29 times faster than the program would run on one processor..., assuming it *could* run on one processor.

The laws compared

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.

(These were taken from a past Wikipedia page for Gustafson's Law but no longer appear in the page as of Jan 2018).

Outline

Speedup revisited

- The above metrics ignore communication and other parallel overhead. When evaluating an approach to parallelizing a task, we should include estimates of communication costs since these can dominate parallel program time.
- As before, we take t_{seq} be the time for a sequential version of the program and t_{par} be the time for the parallel algorithm on N processors. Then speedup is

$$\psi = \frac{t_{\text{seq}}}{t_{\text{par}}}.$$

Parallel Execution Time

- Parallel execution time t_{par} can be broken down into two parts, computation time t_{comp} and communication time t_{comm}.

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

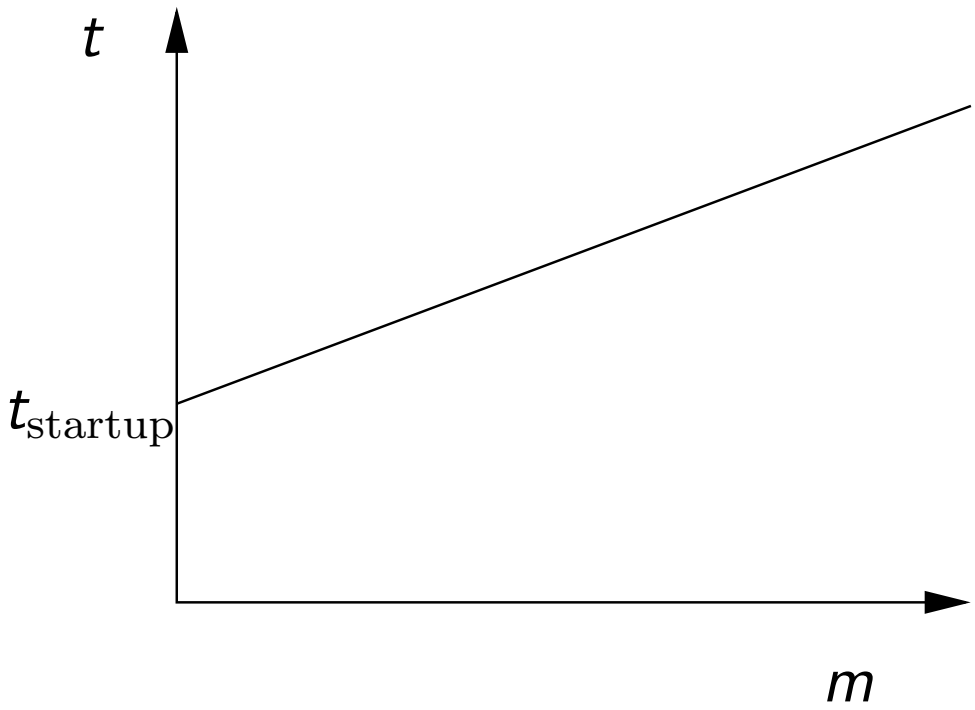
- Speedup is then

$$\psi = \frac{t_{\text{seq}}}{t_{\text{par}}} = \frac{t_{\text{seq}}}{t_{\text{comp}} + t_{\text{comm}}}$$

- The *computation/communication ratio* is $t_{\text{comp}}/t_{\text{comm}}$.

Communication Time

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



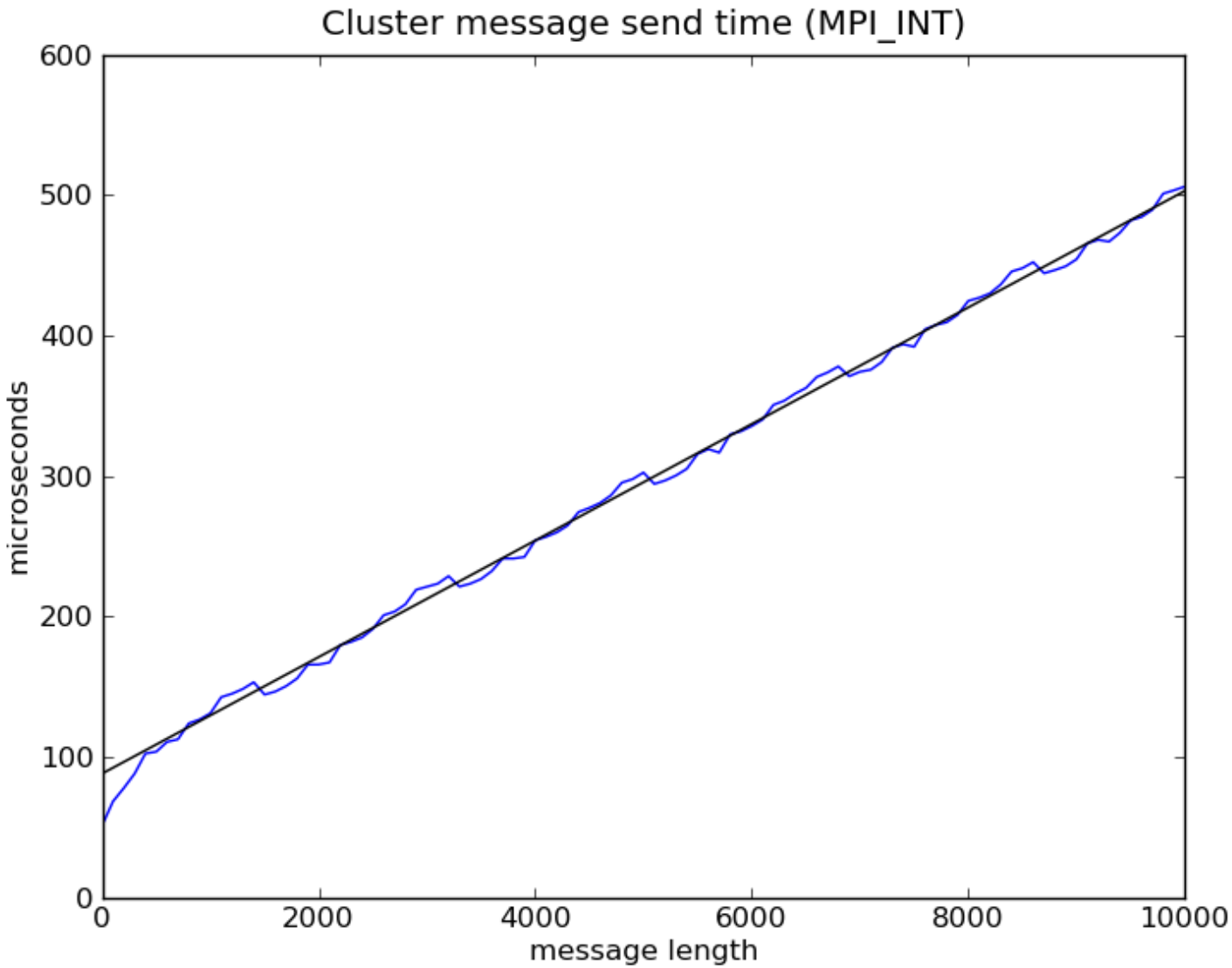
where m is the number of data items sent.

A communication timing experiment

The values of t_{startup} and t_{data} for a parallel cluster can be determined empirically.

- Test program sends messages ranging in length from 100 to 10000 integers between two nodes A and B.
- Each message is sent from node A and received by node B and then sent back and received by node A.
- Repeated 100 times
- Linear regression used to fit line to timing data
- Using the workstation cluster we had in Fall 2010 (just before we moved in KOSC), we determined $t_{\text{startup}} = 88.25\,\mu\text{sec}$ and $t_{\text{data}} = 0.0415\,\mu\text{sec}$ per integer, which corresponds to 96.43 MB/s (assuming 4 bytes per integer)

Workstation cluster timing data (Fall 2010)

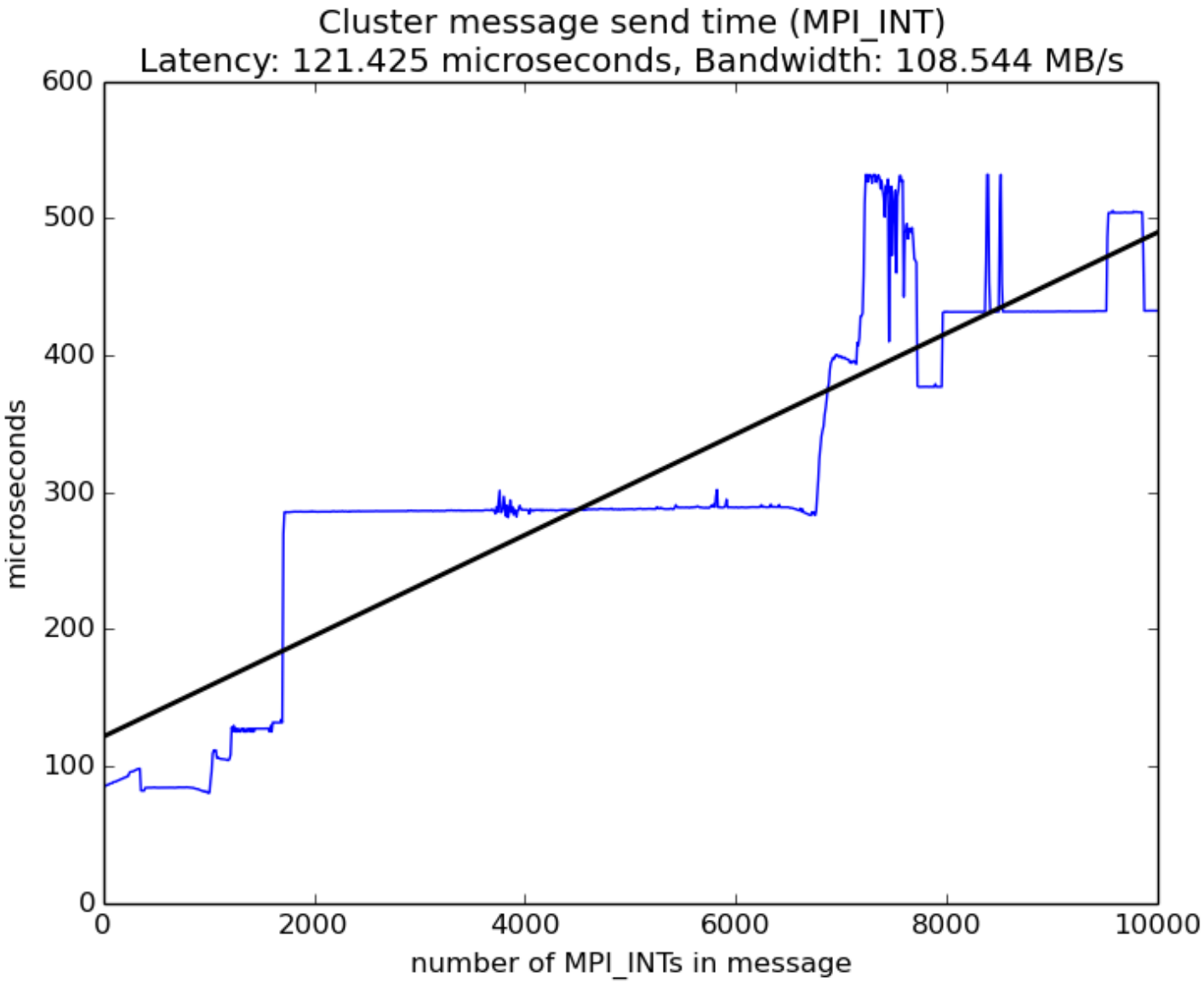


A communication timing experiment

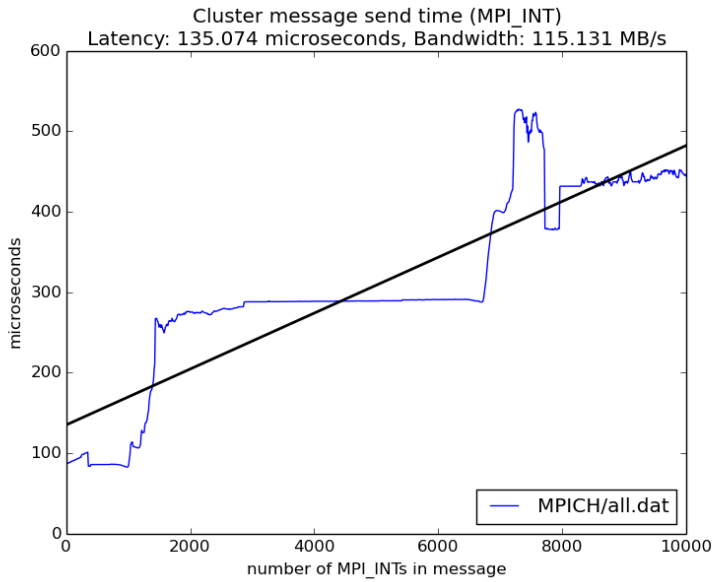
Repeating the experiement in Spring 2016 we found $t_{\text{startup}} = 121.425\,\mu\text{sec}$ and $t_{\text{data}} = 0.03685\,\mu\text{sec}$ per integer, which corresponds to 108.544 MB/s (assuming 4 bytes per integer)

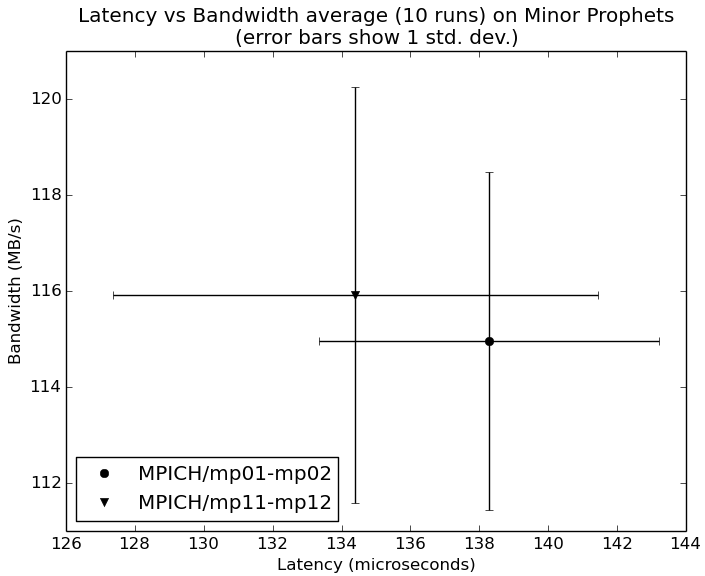
However, the performance curve was surprising...

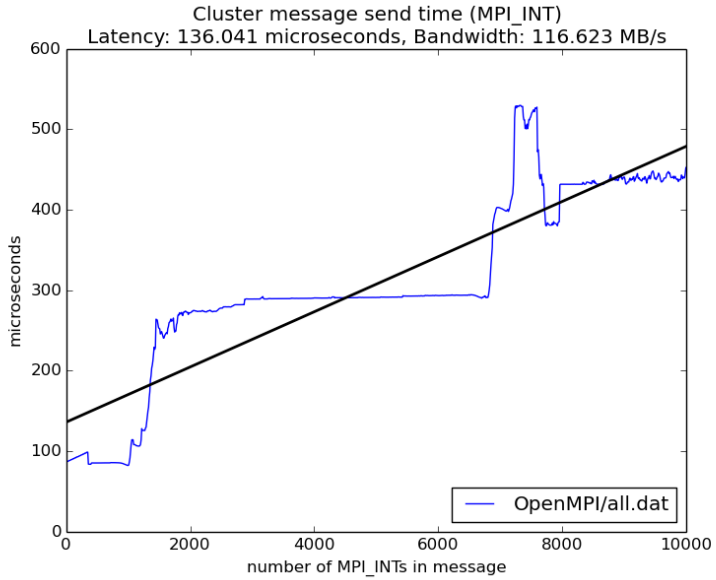
Workstation cluster timing data (Spring 2016)

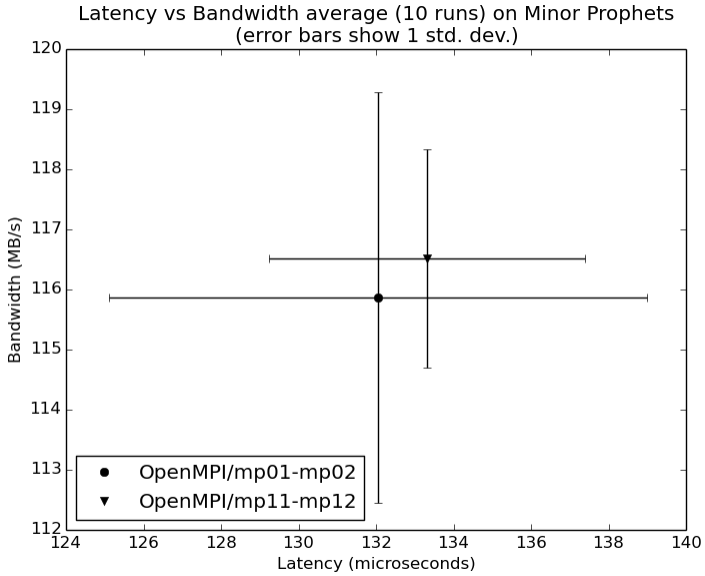


Minor Prophets cluster latency and bandwidth (2016)

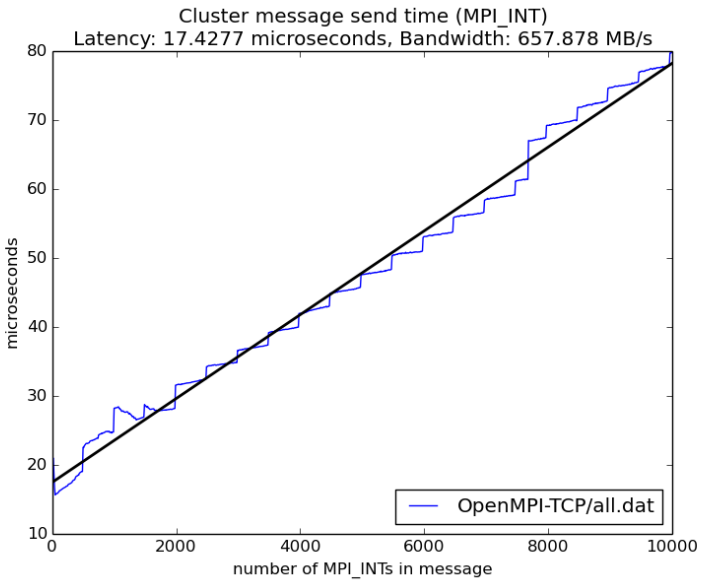


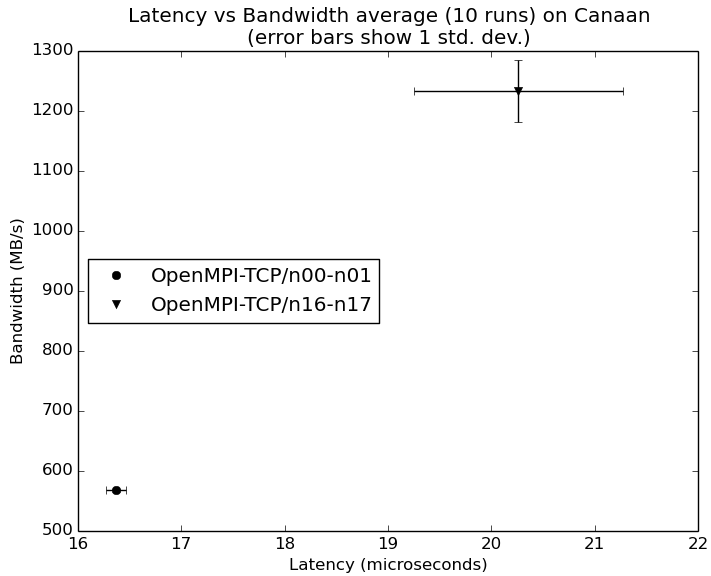


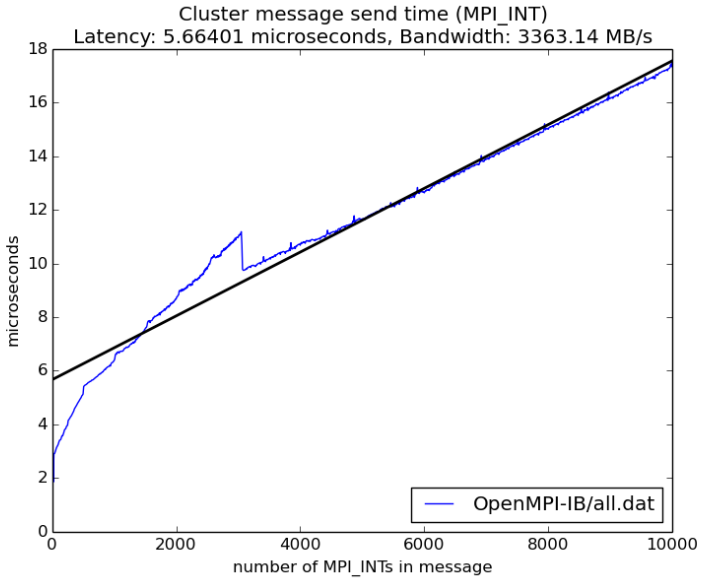


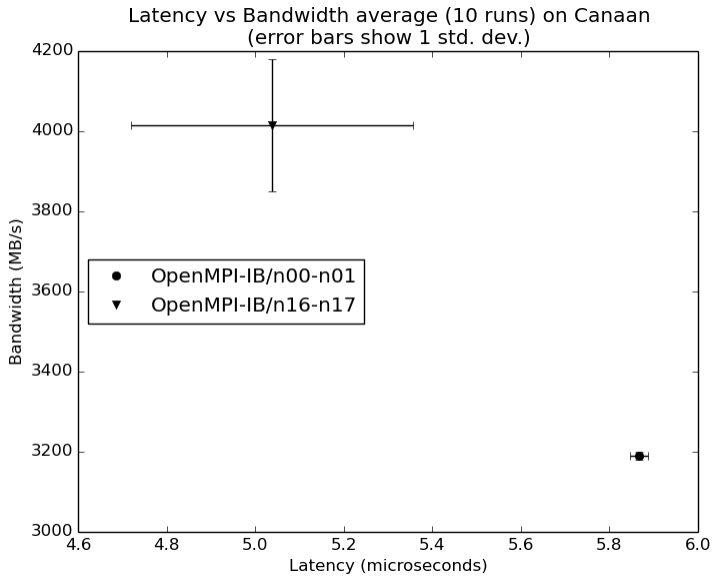


Canaan cluster latency and bandwidth (2016)

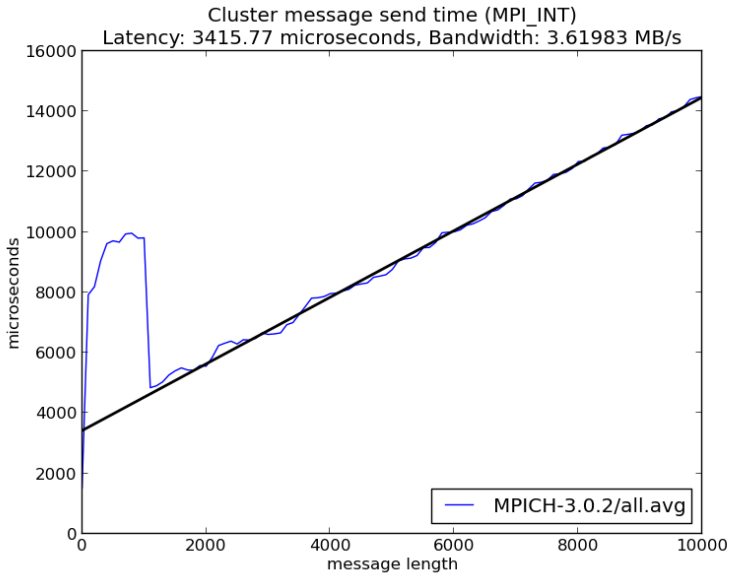


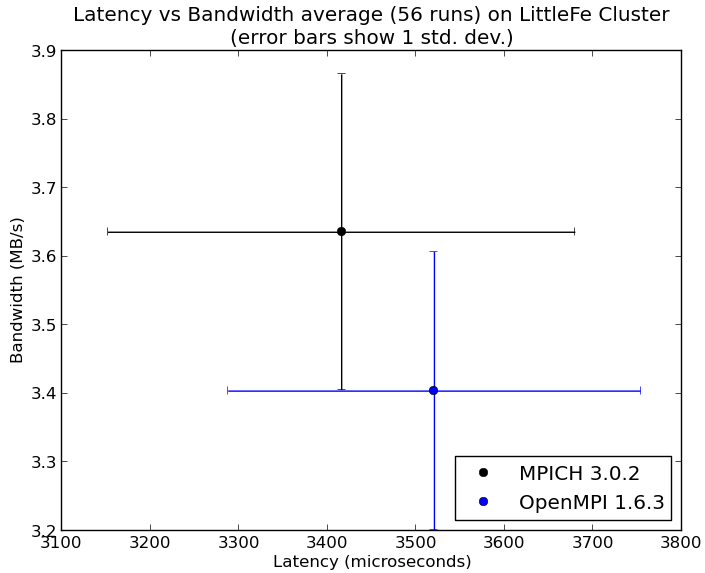


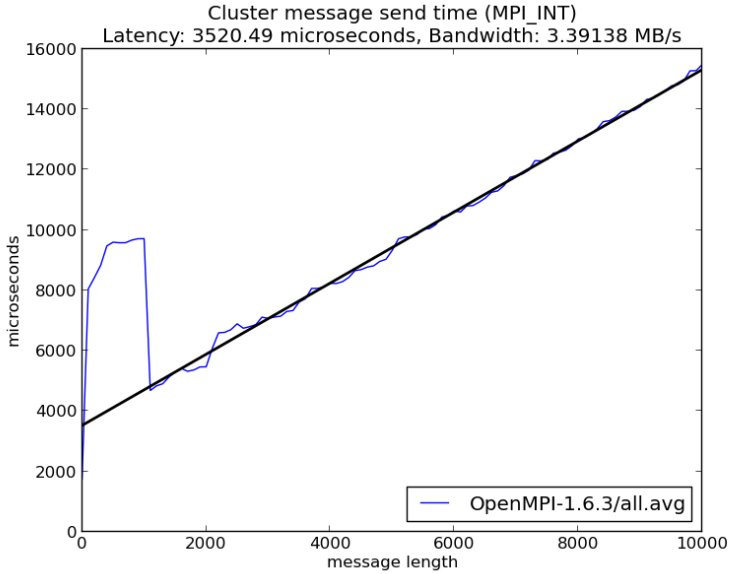




LittleFe cluster latency and bandwidth (2013)

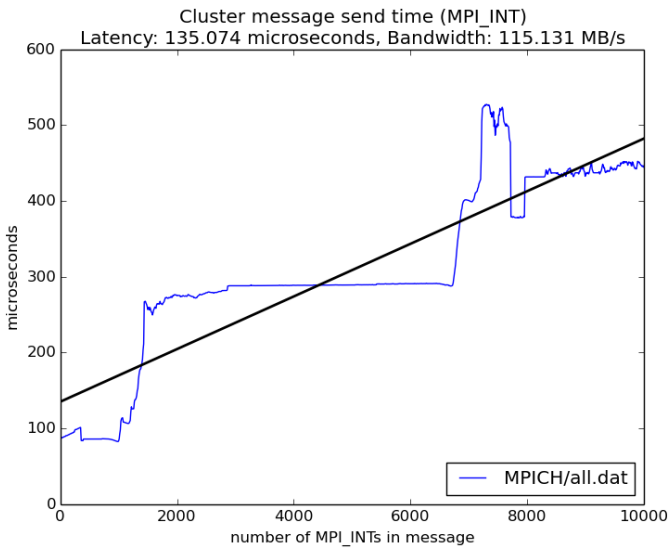


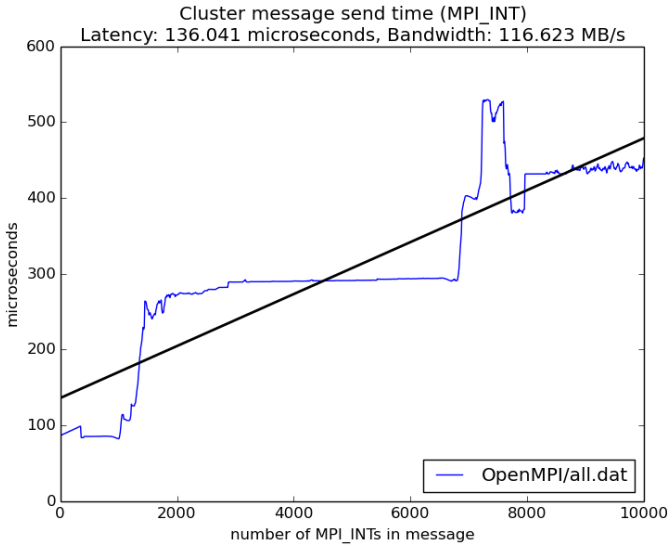




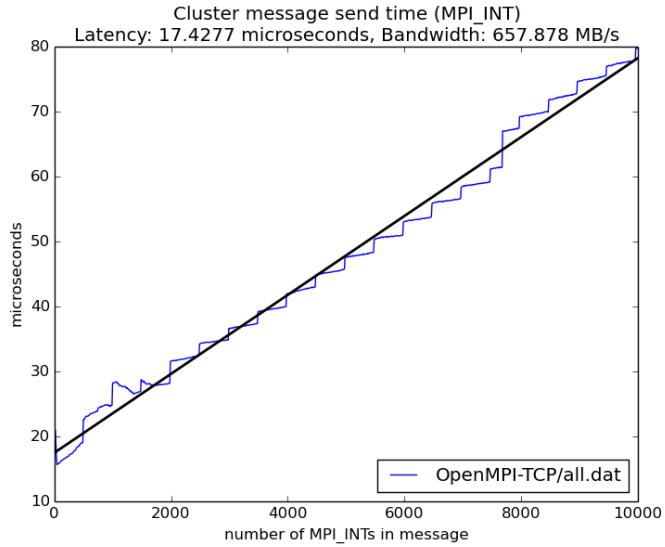
Cluster latency and bandwidth comparison

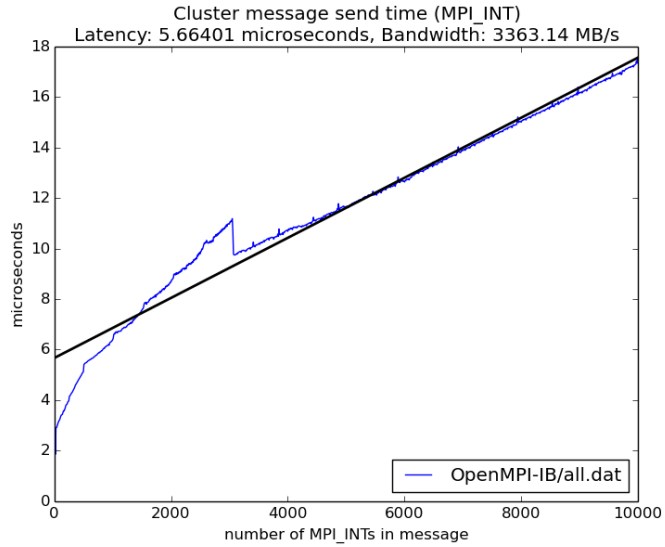
Minor Prophets





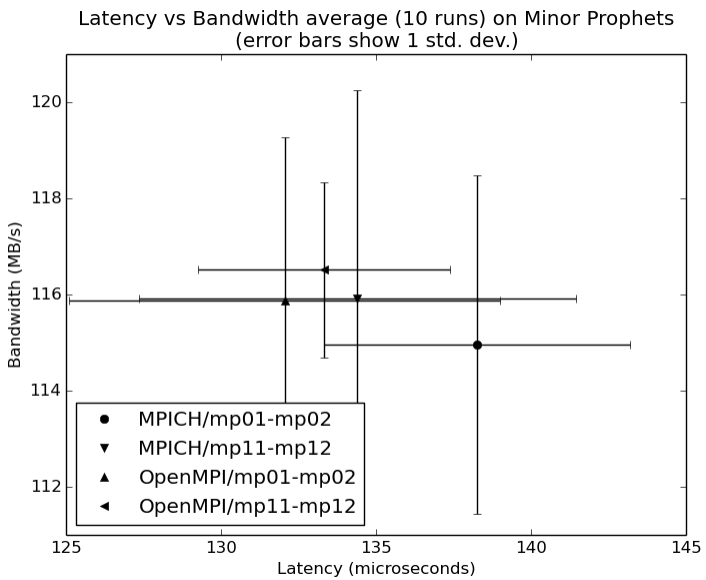
Canaan



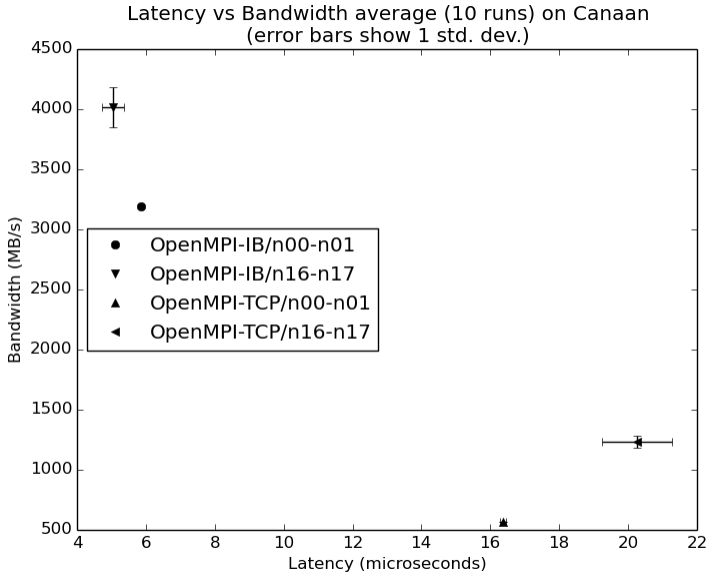


Cluster latency and bandwidth comparison

Minor Prophets



Canaan



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

Acknowledgements

Some material used in creating these slides comes from

- *Multicore and GPU Programming: An Integrated Approach*, Gerassimos Barlas, Morgan Kaufman/Elsevier, 2015.
- *Introduction to High Performance Scientific Computing*, Victor Eijkhout, 2015.
- *Parallel Programming in C with MPI and OpenMP*, Michael Quinn, McGraw-Hill, 2004.