R and the Message Passing Interface on the Little Fe Cluster

Dr. Erin Hodgess

September 12, 2012

Discussion Topics

- Overview
- Little Fe
- BCCD
- Parallel Programming
- MPI
- R with MPI
- Results

Overview

- At SuperComputing 2011, the University of Houston - Downtown received a Little Fe Cluster for teaching and research purposes.
- The Little Fe was actually built on-site in Seattle and transported back to Houston.

The Little Fe Cluster

The Little Fe Cluster

- LittleFe is a complete multinode Beowulf style portable computational cluster designed as an educational appliance for reducing the friction associated with teaching high performance computing (HPC) and computational science in a variety of settings.

The Little Fe Cluster

- The entire package costs less than $\$3,000$, weighs less than 50 pounds, travels easily, and sets up in five minutes

The BCCD software

- The software stack of choice for LittleFe units is the Bootable Cluster CD (BCCD).
- The BCCD is a ready–to-run custom Debian Linux distribution that includes all of the software needed to teach HPC and computational science, e.g. MPI (MPICH2 and OpenMPI), OpenMP, CUDA, Hybrid Models etc.

The BCCD software

- It comes in Live CD flavor that can be booted from either a CD or USB, and if one wished could later be installed onto the hard drive.
- Our software is installed onto Little Fe.

The BCCD software

- BCCD comes with different teaching curriculum modules in the areas of computational sciences.
- These module include, the N-Body problems, Molecular dynamic, Area under the curve, Conway's Game of Life, Parameter Space, HPL-benchmarking, Pandemic, Sieve, Tree-sort, CUDA, and MPI hello-world.

The BCCD software

- The original project goal was to make BCCD evolve into a more useful and user friendly tool for petascale education.
- This was done by the following two sub projects:
  - Updating all of the existing teaching modules that are currently being shipped with BCCD and develop new modules. This would include instrumenting, documenting and improving all the software packages that are part of the BCCD with the PetaKit benchmarking software we developed previously.
  - These would be used for individuals teaching themselves about hybrid models or by faculty teaching a class with a unit on one or more hybrid models.

The BCCD software

- LittleFe/BCCD serves as an on-ramp to national computational resources such as XSEDE.
- By using the same compilers, parallel libraries and job submission tools as commonly found on XSEDE clusters, people can learn to use those tools in a simple and low-friction environment

Parallel Programming

- According to Pacheco, a parallel computer is simply a computer with multiple processors that can work together on solving a problem.
- Suppose you are doing a jigsaw puzzle
- It takes you an hour to complete it
- If someone else joins you, it may take 40 minutes.

Parallel Programming

- You typically do not not get linear speed up.
- You usually have to ask for a puzzle piece from the other person.
- Therefore, communication must take place.

MPI

- Hence, we use MPI, or the Message Passing Interface to do our communication.
- The flavor that we use is Open MPI.
- These are libraries of functions called from C or Fortran, and now R.
- A typical function might be *mpi send* or *mpi recv*.

MPI

- MPI Applications fall into the Single Instruction Multiple Data family, or SIMD.
- There is a manager and a set of workers.
- We will see an example in a moment.
- The advantage is that the SIMD applications are usually easy to program.
- The disadvantage is that some processes will remain idle.

R and MPI

- How does R work with MPI?
- Like any good statistical question, the answer is "It depends".
- When you have scheduling software, there is one approach.
- Without scheduling software, the approach is completely different.

R and MPI

- Hao Yu from the University of Western Ontario wrote the Rmpi library.
- These are the bindings to the appropriate MPI library functions.
- The best way to use Rmpi is on a UNIX or Linux system.

R and MPI


```
bccd@node000:~$ bccd-snarfhosts
bccd@node000:~$ cat machines
node000 slots=2
node014.bccd.net slots=2
node013.bccd.net slots=2
node012.bccd.net slots=2
node011.bccd.net slots=2
bccd@node000:~$ mpirun –machinefile
machines TestRmpi.sh stuff1.R
1 worker
manager
2 worker
3 worker
5 worker
4 worker
7 worker
9 worker
6 worker
```

R and MPI


```
bccd@node000:~$ cat TestRmpi.sh
#!/bin/sh
R --slave < $1

bccd@node000:~$ cat stuff1.R
library(Rmpi)
if(0==mpi.comm.rank(comm=0 )) {
cat("manager\n")
} else {
cat(mpi.comm.rank(comm=0 ),"worker\n")
}

mpi.quit()
```

R and MPI

- We simulated a time series:

$$(1 - \phi B)x_t = a_t,$$

in which $\phi = 0.5$, B is the backshift operator such that $B^j = x_{t-j}$ and a_t is a Gaussian white noise series with a constant variance of σ_a^2.

- We simulated 360 values, to represent 30 years of monthly data.

R and MPI

- We then aggregated the series to produce a quarterly series with 120 observations.

$$Y_T = \sum_{m(T-1)+1}^{mT}$$

- We then fit an autoregressive (AR) model of order 1 to the aggregate series.
- We obtained the estimated value of the AR coefficient
- We did this for serial code, 4, 8, and 10 cores.

Results

- The results for the 4 cores were fairly dreadful. There were 3 sets of results in which it was better to run the serial code than the parallel code.
- The 8 core results were typically in the middle.
- Even with the 10 cores, we typically got about 4.5 - 5 times speed up.

Results

Times in Seconds for $\phi = 0.5$

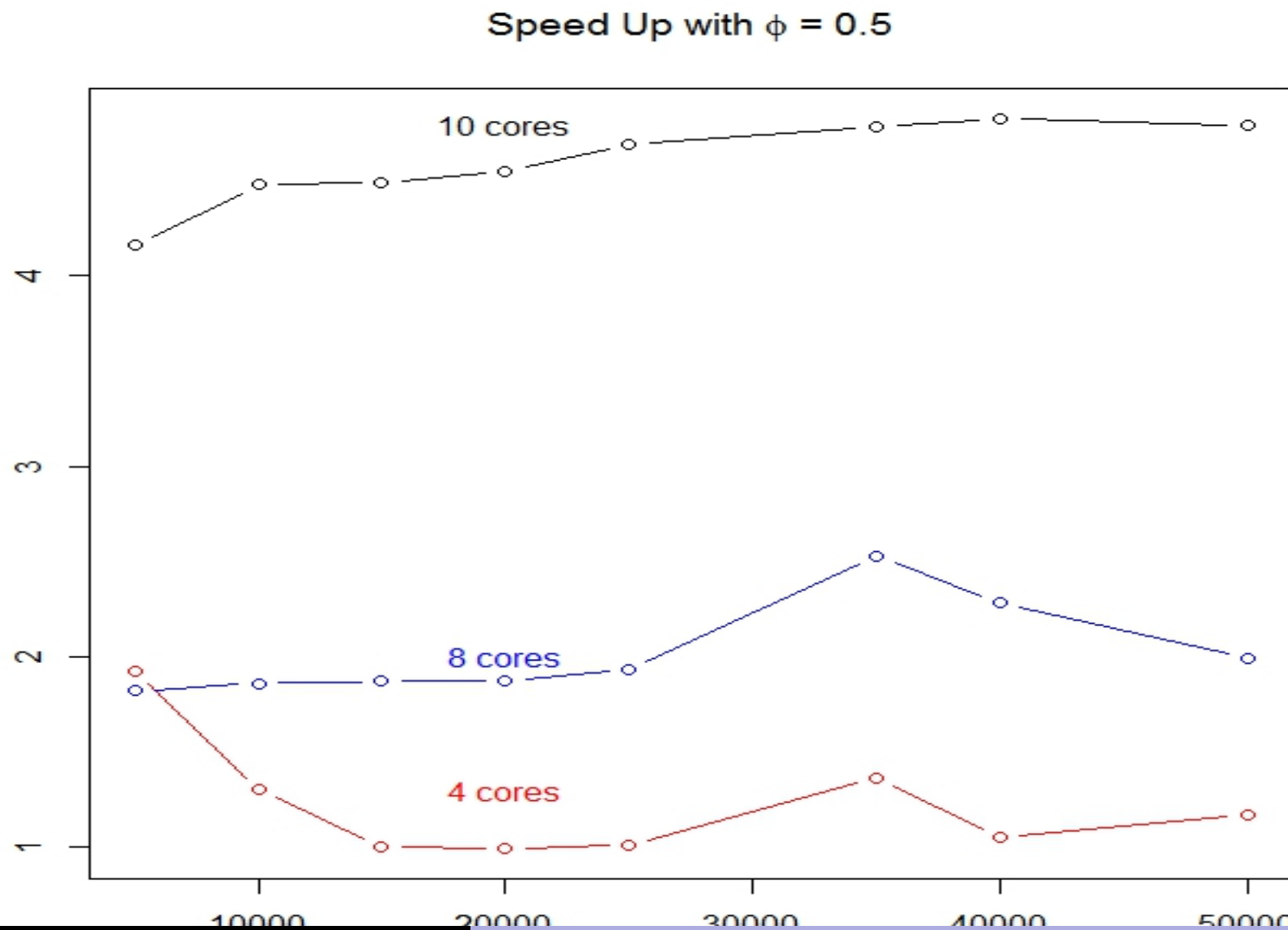
Reps	Serial	4 cores	8 cores	10 cores
5000	212.375	110.765	116.628	51.053
10000	428.572	330.315	235.901	95.601
15000	639.299	636.783	341.967	142.329
20000	853.835	862.592	457.114	187.676
25000	1090.983	1077.920	565.093	232.502
35000	1552.590	1141.524	613.720	624.523
40000	1791.702	1733.088	785.720	370.827
50000	2221.351	1896.551	1115.778	462.457

Results

Speed Up

Reps	4 cores	8 cores	10 cores
5000	1.92	1.82	4.16
10000	1.30	1.86	4.48
15000	1.00	1.87	4.49
20000	0.99	1.87	4.55
25000	1.01	1.93	4.69
35000	1.36	2.53	4.78
40000	1.05	2.28	4.83
50000	1.17	1.99	4.79

Results


Speed Up with ϕ = 0.5
10 cores
8 cores
4 cores
1
2
3
4
10000
20000
30000
40000
50000

Results

Times in Seconds for $\phi = 0.9$

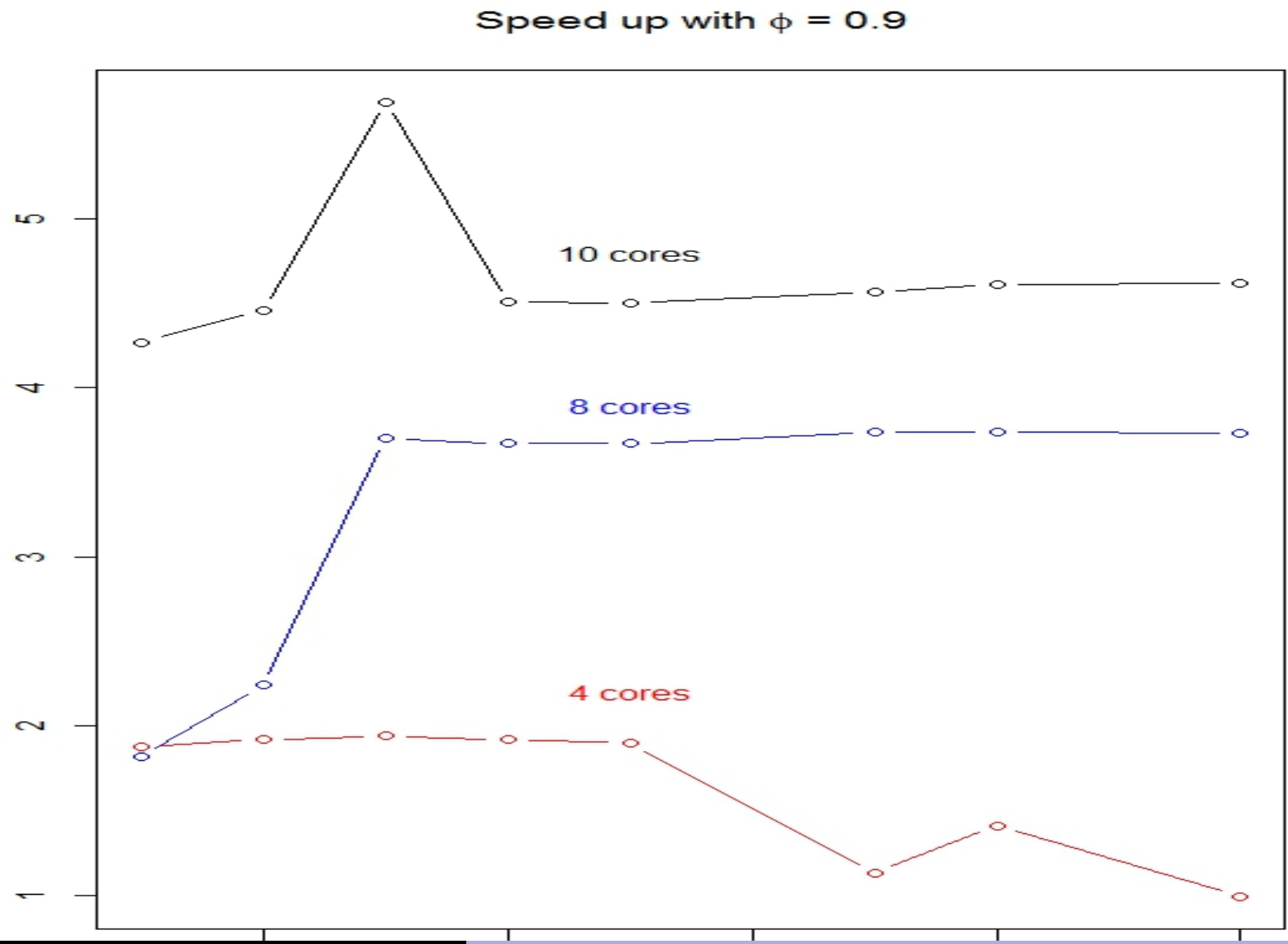
Reps	Serial	4 cores	8 cores	10 cores
5000	230.683	122.587	126.685	54.028
10000	461.598	240.046	206.086	103.544
15000	692.959	356.343	187.198	151.832
20000	915.517	477.571	249.265	203.100
25000	1136.443	596.609	309.383	252.659
35000	1610.167	1418.390	430.607	352.430
40000	1848.317	1313.246	493.638	400.668
50000	2314.064	2337.964	620.899	501.381

Results

Speed Up

Reps	4 cores	8 cores	10 cores
5000	1.88	1.82	4.27
10000	1.92	2.24	4.46
15000	1.94	3.70	5.69
20000	1.92	3.67	4.51
25000	1.90	3.67	4.50
35000	1.13	3.74	4.57
40000	1.41	3.74	4.61
50000	0.99	3.73	4.62

Results



Results

- We also looked at disaggregation via Box-Jenkins models.
- We took our aggregate series, Y_T and fit an ARMA(1,1) model.
- We produced the autocovariances from the aggregate model and then those of the disaggregate model.
- Finally, we produced the disaggregate series.
- Our equation is:

$$\hat{x}_t = \mathbf{V}_x(\mathbf{C^0})'\mathbf{V}_Y^{-1}Y_T$$

- We measured the underlying ϕ value.

Results

Disaggregation Times in Seconds for $\phi = 0.9$

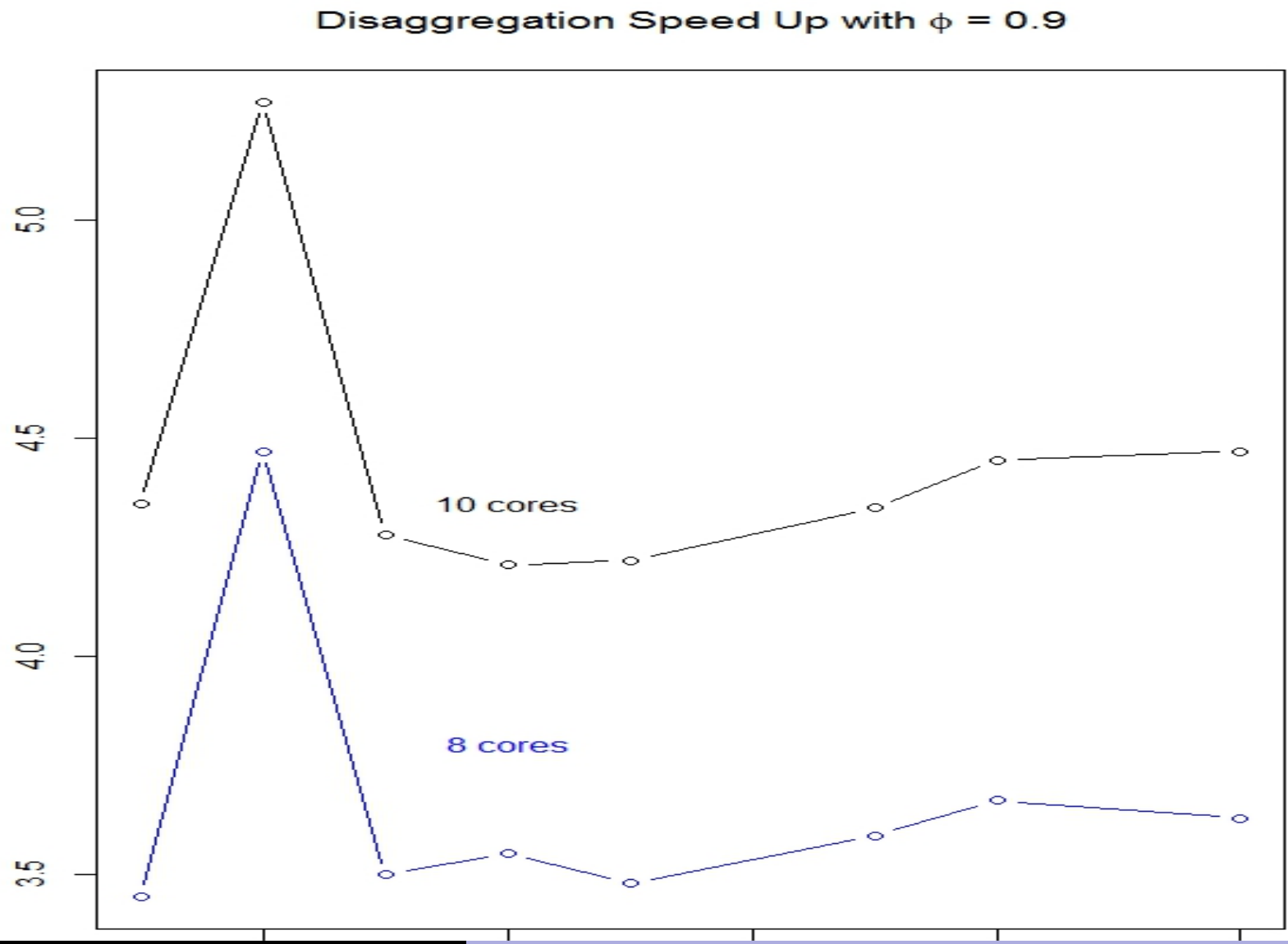
Reps	Serial	8 cores	10 cores
5000	551.540	158.983	126.713
10000	1336.457	299.774	253.434
15000	1551.974	443.223	362.743
20000	2078.349	586.219	493.174
25000	2593.384	745.258	614.164
35000	3718.948	1035.005	857.138
40000	4323.872	1178.625	970.826
50000	5393.012	1487.129	1205.813

Results

Speed Up for Disaggregation

Reps	8 cores	10 cores
5000	3.45	4.35
10000	4.47	5.27
15000	3.50	4.28
20000	3.55	4.21
25000	3.48	4.22
35000	3.59	4.34
40000	3.67	4.45
50000	3.63	4.47

Results



Questions?

Thank you!