Paul's Norwegian Vacation

(or "Experiences with Cluster Computing")

Paul Sack

20 September, 2002

sack@stud.ntnu.no

www.stud.ntnu.no/~sack/

Outline

- Background information
- Work on clusters
- Profiling tools
- Porting Fortran 90 code
- Performance analysis
- MPI Broadcast w/ multicast

IDI Cluster: Clustis

- Description of clustis: 1.46 GHz AMD processors, 2 GB RAM, 40 GB drives

- Use OpenPBS batch system (PBS implementation)

- Classification of nodes:
  - `clustis`: master node
  - `node01`-`node08`: interactive use
  - `node09`-`node40`: batch use

- Several queues of different priorities:
  - privileged users
  - guest users
  - sif80bi users

- Documentation

- `qmpirun` script:
  - Usual MPI invocation:
    `mpirun -np 4 program`
  - To use batch system:
    `qmpirun -np 4 program`
  - Output saved in `program.out` and `program.err`
  - Creates and submits shell script for batch system
  - Cleans itself up when finished

- `mpirun` modified to support OpenPBS

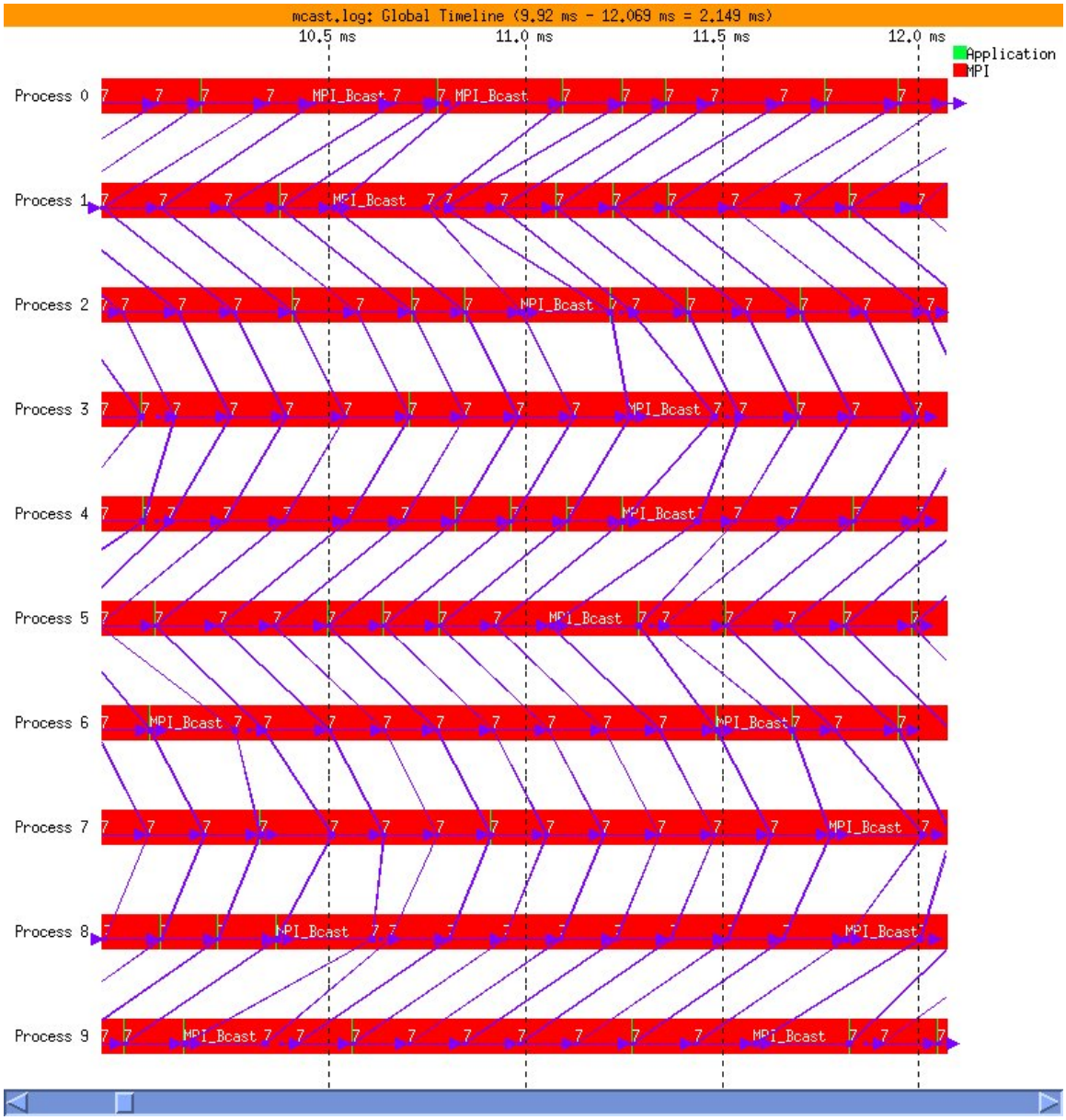
Itanium cluster

- Two IA-64 nodes and master

- Setup OpenPBS and `qmpirun` script as for clustis

Profiling Tools: Vampir

- Produced by Pallas GmbH in Cologne and Dortmund
- Easiest to use: wrappers for `mpicc`, `mpiCC`, `mpif90`
- Simple, effective design: static libraries
- Shows individual MPI calls during a timeslice
- Shows aggregate statistics in varying levels of detail
- Expensive: 75 000 Kroner for clustis


mcast.log: Global Timeline (9.92 ms - 12.069 ms = 2.149 ms)
10.5 ms
11.0 ms
11.5 ms
12.0 ms
Application
MPI
Process 0
Process 1
Process 2
Process 3
Process 4
Process 5
Process 6
Process 7
Process 8
Process 9
MPI_Bcast

Profiling Tools: Paradyn

- Written at University of Wisconsin
- Difficult to use: non-intuitive GUI interface
- Complex design: run-time code-patching
- Shows even more data than Vampir
- Does not work in batch environments
- Impossible to compile
- Inexpensive: Free

Profiling Tools: Jumpshot

- Written at various universities and research centres in USA
- Comes with MPICH
- Simple design: static libraries
- Has years of testing and works well
- Limited data: no aggregate statistics
- Also free


Pointer
Elapsed Time
6.087072
0.021492206
Zoom Operations
In
Out
Reset
Miscellaneous Operations
Options
Print
Close
0:0
1:0
2:0
3:0
4:0
5:0
6:0
7:0
6.069010
6.072983
6.076954
6.080926
6.084898
6.08887
6.092842
6.096813
Elapsed Time
BARRIER
REDUCE
RECV
SEND
All States
forward arrow
Change Orientation

Porting Fortran 90 Code

- Compilers not standard-compliant
- Most compilers don’t fully support standard **and** exceed the standard
- This step took the longest in the project
- SGI compiler
  - SGI supercomputers used for scientific computing
  - Scientists and mathematicians like Fortran
  - Hence, SGI writes a Fortran 90 compiler that produces very optimized code
  - Code originally written on SGI

Intel compilers

- NAGWare (NTNU site-licensed)
  - Compiler produces faulty code in array reshaping code
  - Debugger segfaults
  - NAGWare acknowledges bug, recommends up-grading (£££!)
- Intel
  - Intel also produces supercomputers
  - Good compiler, but strictly standard-conformant
  - My code has some obsolete Fortran 77 syntax
  - Free!

- Portland Group (Bergen and Linköping)
  - Compiler mishandles 3D-array reshape calls
  - Successfully rewrote this as a series of 2D-array reshape calls
  - Sometimes segfaults
- No GNU Fortran 90 compiler yet

Getting MPICH to work with Fortran 90

- Simple mapping in C between function/variable names and symbol names in object files (`foo() = foo`)
- Fortran 90 compilers sometimes add one underscore and sometimes add two (`foo() = foo_ or foo__`)
- MPICH configure script is supposed to handle this, but doesn't work
- Since Fortran 90 is mostly used for scientific computing, the Portland Group provides a custom configure script, which does work

- Porting
  - Used Portland Group compiler
  - Joakim Hove and Knut Petter assisted me in correcting many errors
  - Most of the errors were in extraneous portions of the code and could be commented out.
  - By comparing the code with standard-compliant code and a bit of guesswork, I managed to get all of the necessary modules to compile without error

Profiling: Systems

- Cluster in Bergen: 32 $\times$ dual 1.26 GHz Pentium IIIs 100 Megabit network
- Cluster in Linköping: 33 $\times$ 900 MHz Pentium IIIs 100 Megabit and SCI (full-duplex Gigabit) networks
- Supercomputer (embla): 512 $\times$ 600 MHz MIPS processors
- Did not use Clustis

Description of Program

- Simulation of behaviour of subatomic particles in near absolute-zero temperatures under an electromagnetic field

- Three-dimensional parallelopiped divided between processes

- One thousand iterations. Each iteration has a separate communication and a computation phase

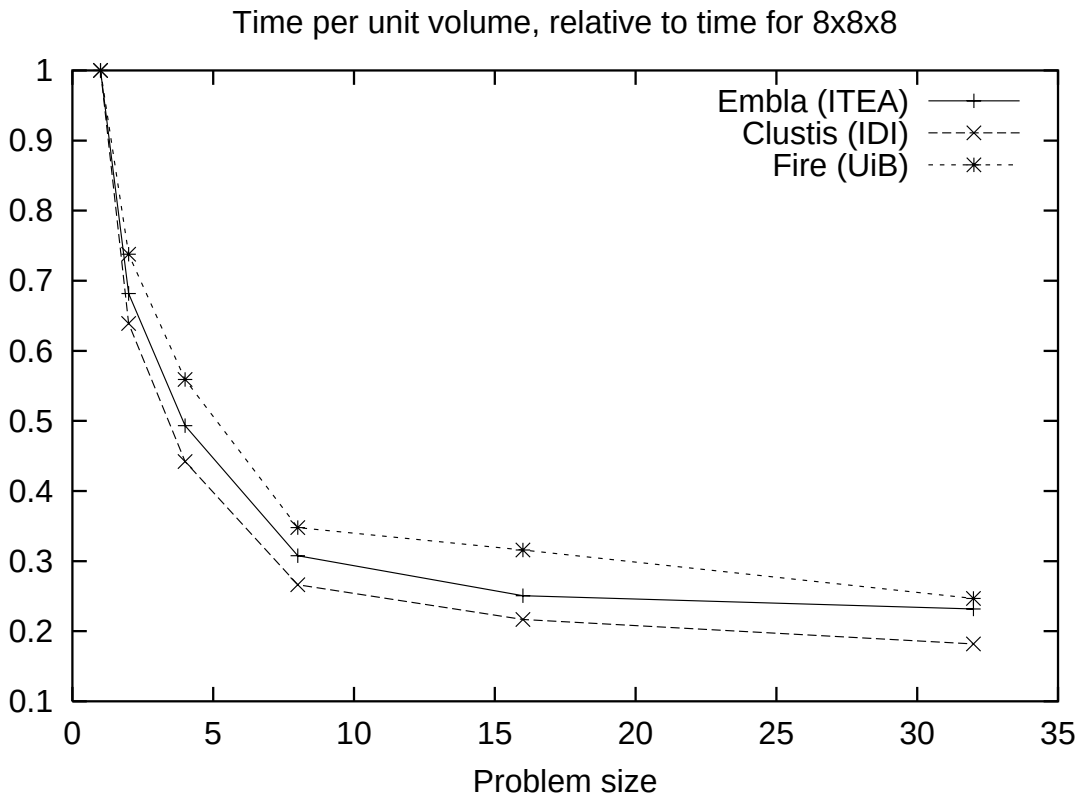
- Initial hypothesis:
  - Cluster better for small number of processes (faster processors)
  - Supercomputer better for large numer of processes (faster interconnect network)

Analysis of Problem Size

- Number of messages passed is proportional to the internal surface areas

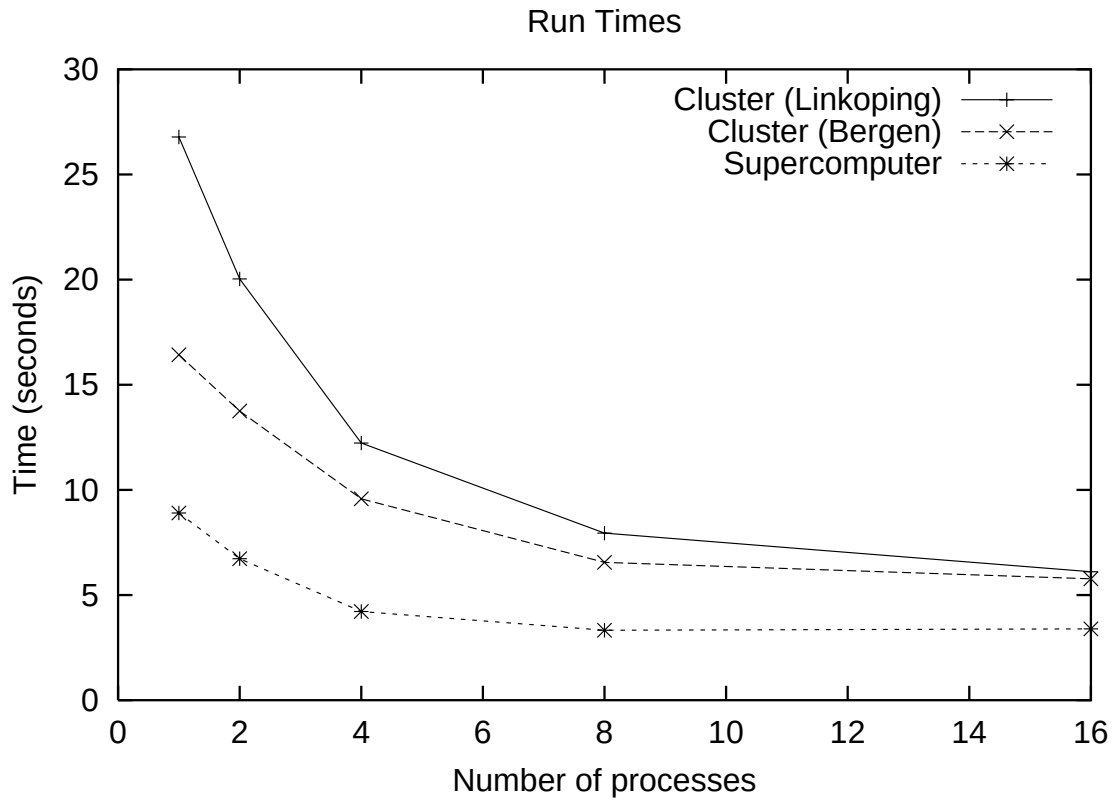
- Amount of computation proportional to the volume

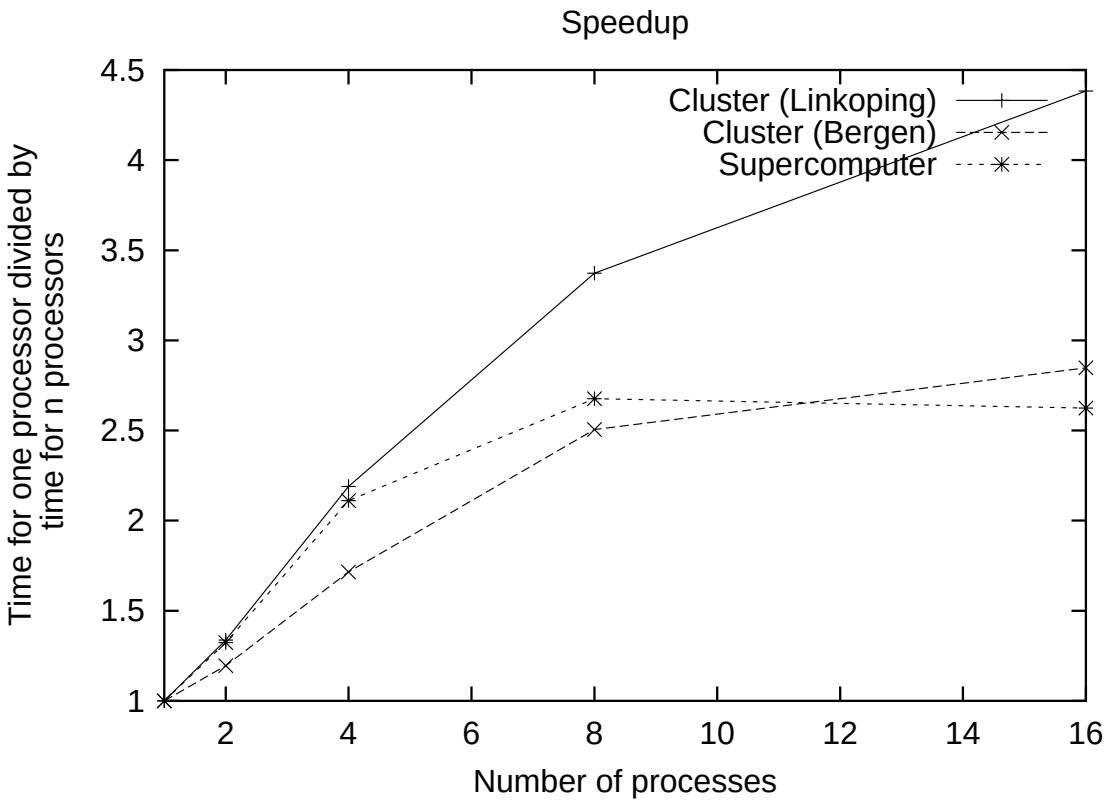
- I expected the program to perform better *per-unit-volume* with larger problem sizes

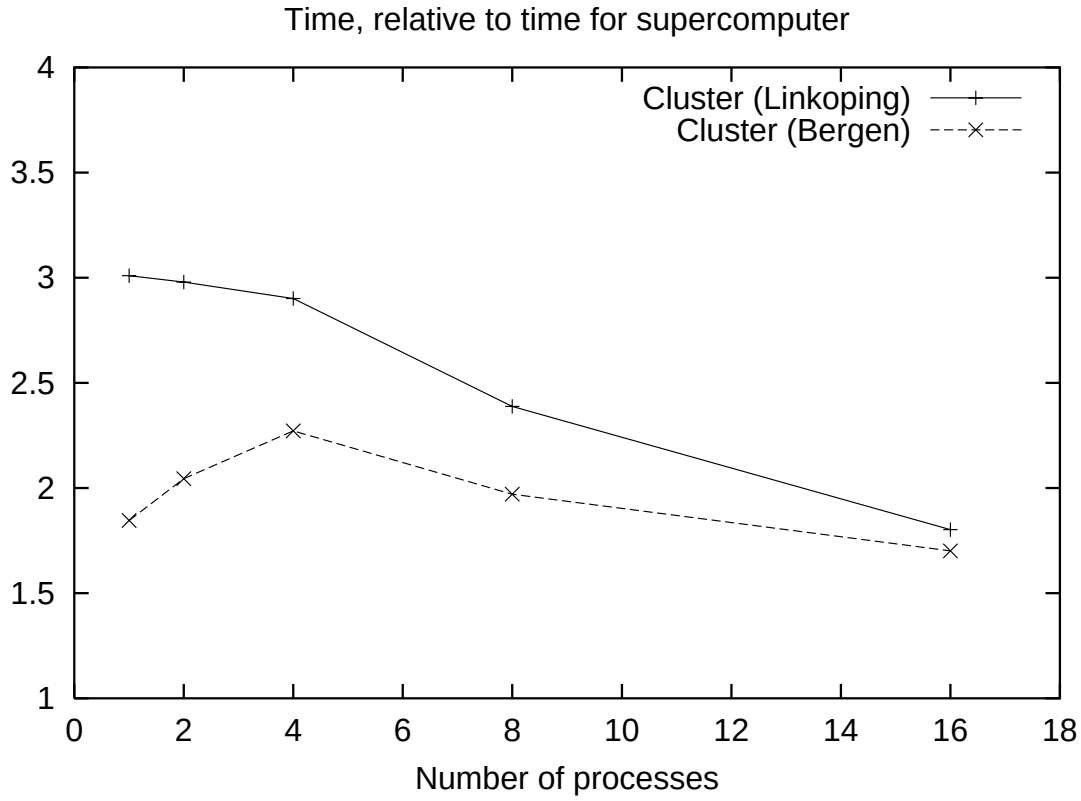


Profiling Results: Speedup

In this test, the problem size is constant and the number of processes used varies.




Speedup
Time for one processor divided by time for n processors
Number of processes
Cluster (Linkoping)
Cluster (Bergen)
Supercomputer
4.5
4
3.5
3
2.5
2
1.5
1
2
4
6
8
10
12
14
16


Time, relative to time for supercomputer
Number of processes
Cluster (Linkoping)
Cluster (Bergen)
4
3.5
3
2.5
2
1.5
1
0
2
4
6
8
10
12
14
16
18

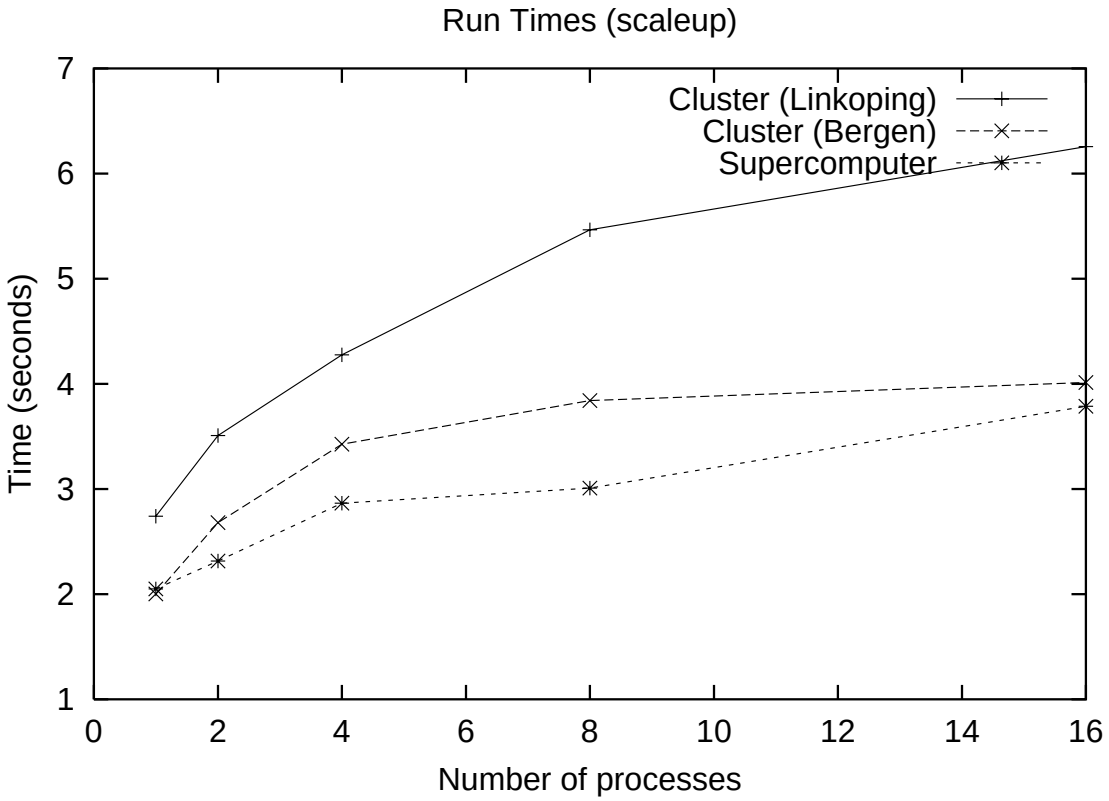
Profiling Results: Speedup

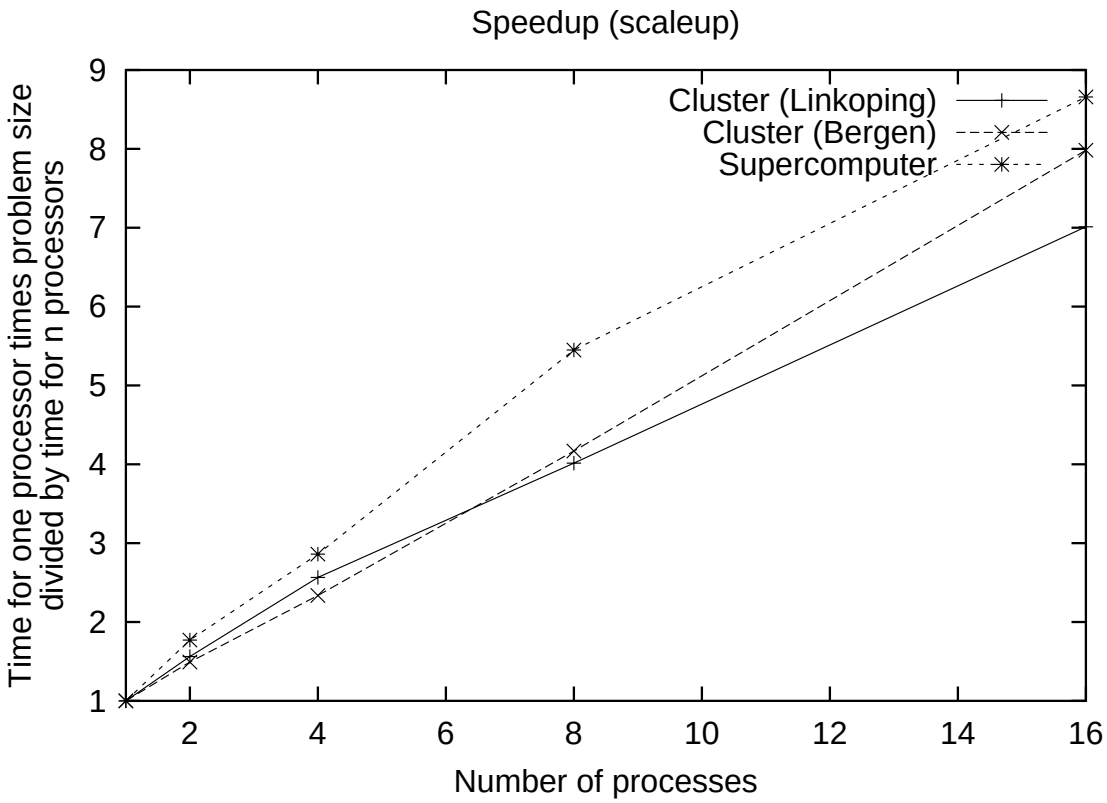
Observations:

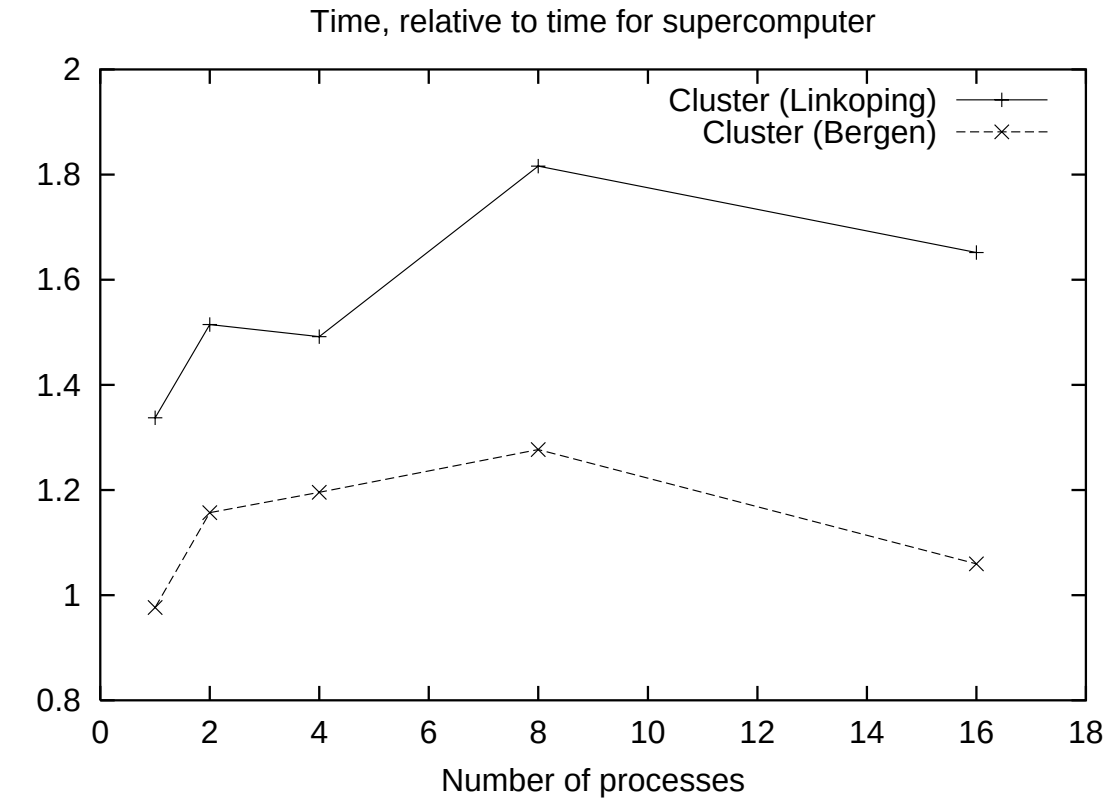
- Unexpectedly, the supercomputer outperforms the clusters for a small number of processes
  - SGI compiler produces more optimized code
  - MIPS CPUs have larger caches
- With a large number of processors, the two clusters' performance is approximately equal

Profiling Results: Scaleup

In this test, the problem size is grown in proportion to the number of processes. In other words, the problem size per process is kept constant.




Speedup (scaleup)
Time for one processor times problem size
divided by time for n processors
Number of processes
Cluster (Linkoping)
Cluster (Bergen)
Supercomputer
1
2
3
4
5
6
7
8
9
2
4
6
8
10
12
14
16


Time, relative to time for supercomputer
Number of processes
Cluster (Linkoping)
Cluster (Bergen)
0.8
1
1.2
1.4
1.6
1.8
2
0
2
4
6
8
10
12
14
16
18

Profiling Results: Scaleup

Observations:

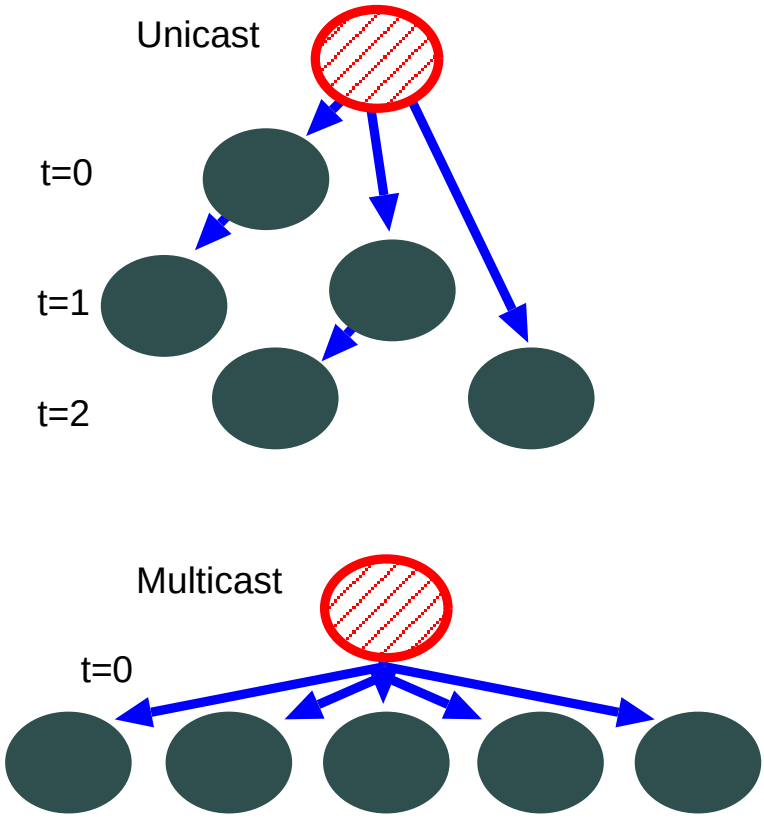
- The clusters fare much better in this test
- The Bergen cluster's performance is within 20% of the supercomputer's
- The Linköping cluster's performance is within 80% of the supercomputer's
- This is a more real-world test, since the amount of work per process is a reasonable amount.
- All the systems exhibit better speedup than in previous test

Profiling Results: Conclusions

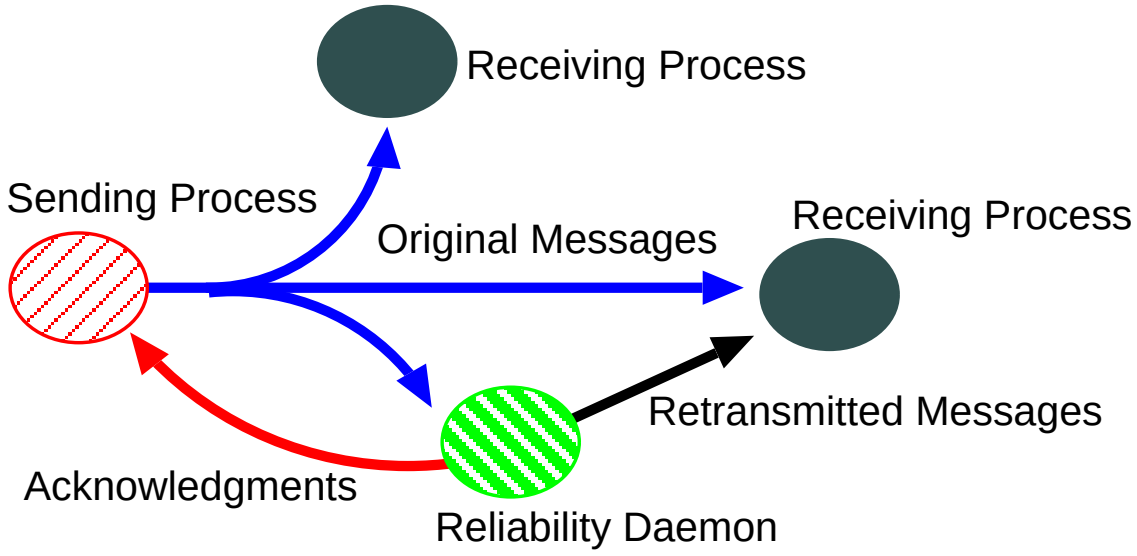
- Clusters perform quite well, considering their cost
- Middle ground: clusters with proprietary Gigabit networks
- Could not get SCI (Gigabit) network to work

MPI Broadcast w/ Multicast

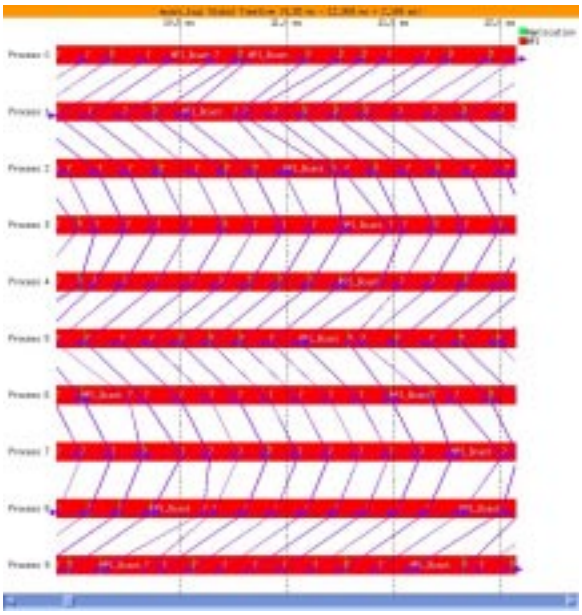
- Broadcast
- With Unicast (TCP/IP) is $O(\log_2 n)$
- With Multicast is $O(1)$
- But Multicast uses UDP, which is unreliable



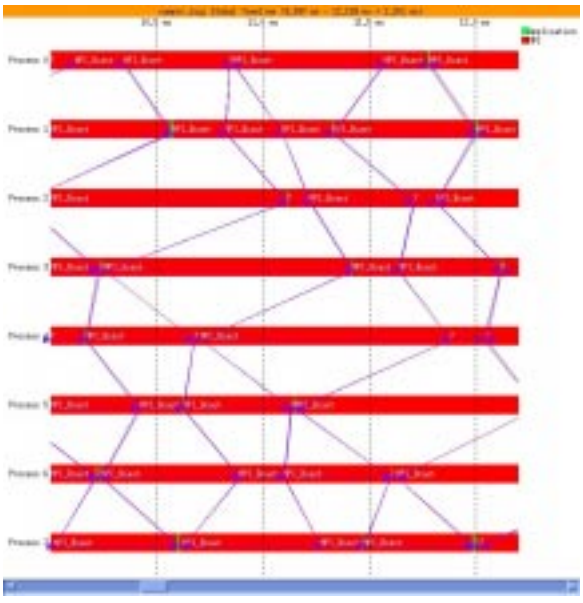
Reliability protocol



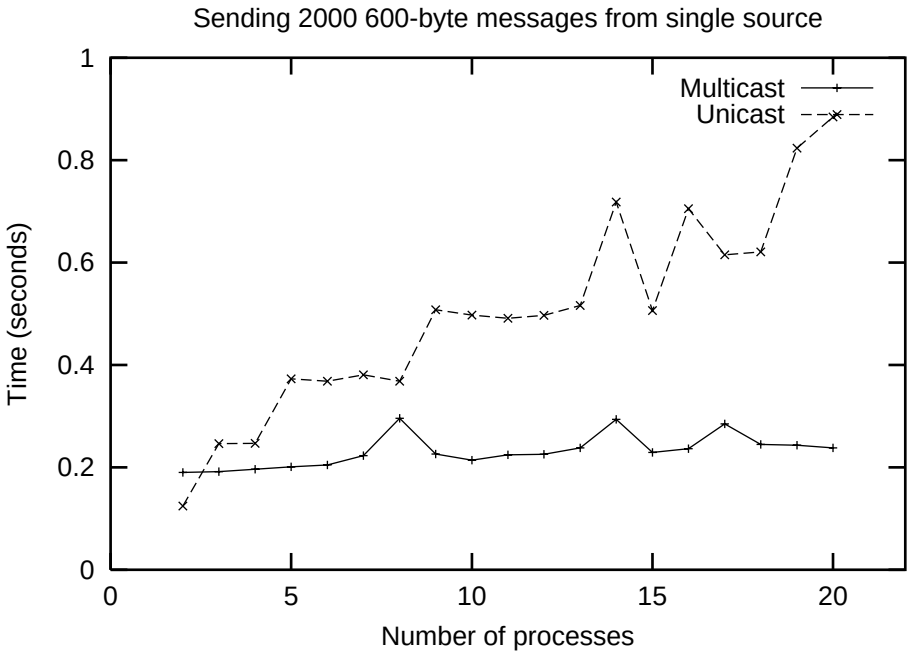
Multicast:

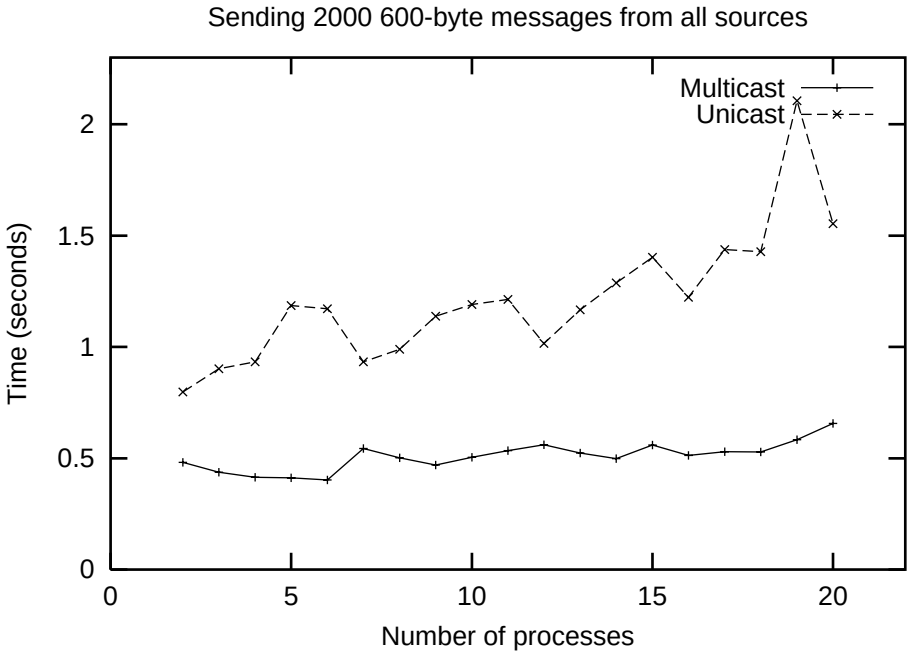


Unicast:



Results


Sending 2000 600-byte messages from single source
Multicast
Unicast
Time (seconds)
Number of processes
0
0.2
0.4
0.6
0.8
1
0
5
10
15
20


Sending 2000 600-byte messages from all sources
Multicast
Unicast
Time (seconds)
Number of processes
0
0.5
1
1.5
2
0
5
10
15
20