MPI in BigDFT

Stephan Mohr
University of Basel
stephan.mohr@unibas.ch

23.5.2012

Table of contents

Table of contents

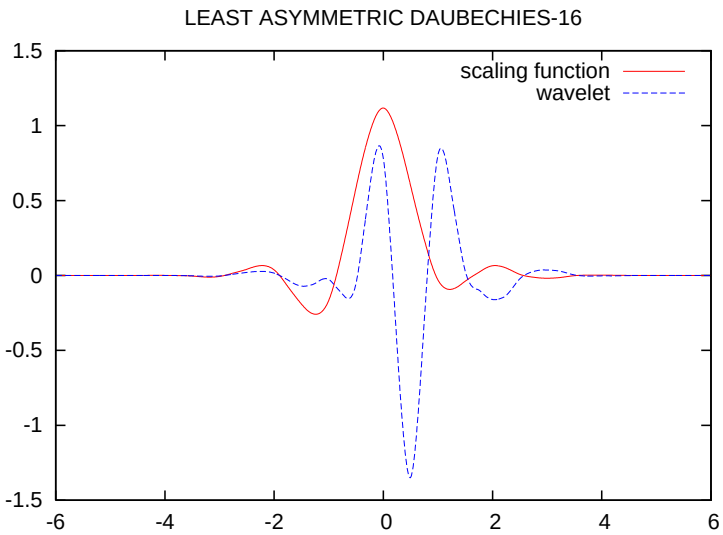
Table of contents

Table of contents

Table of contents

Outline

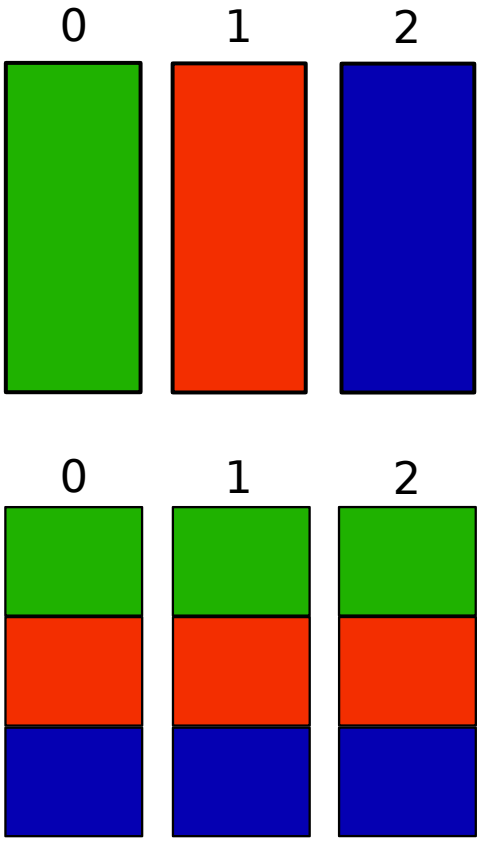
short code description



BigDFT is an electronic structure code based on Density Functional Theory. It uses wavelets as basis set and can do all standard electronic structure calculations.

It is a massively parallel code and exhibits both distributed memory (MPI) and shared memory (OpenMP) parallelization as well as GPU acceleration.

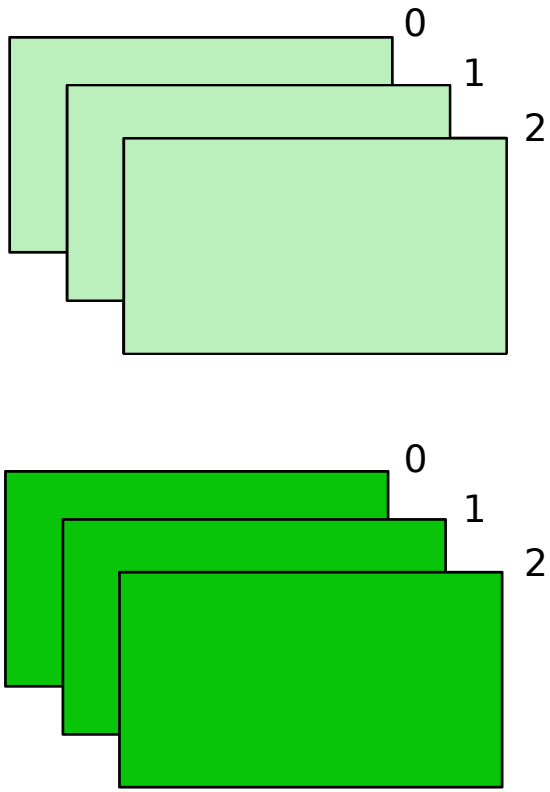
MPI_ALLTOALLV



Each process has one entire object.

Using an MPI_ALLTOALLV we "transpose" the data such that each process has a small amount of all objects.

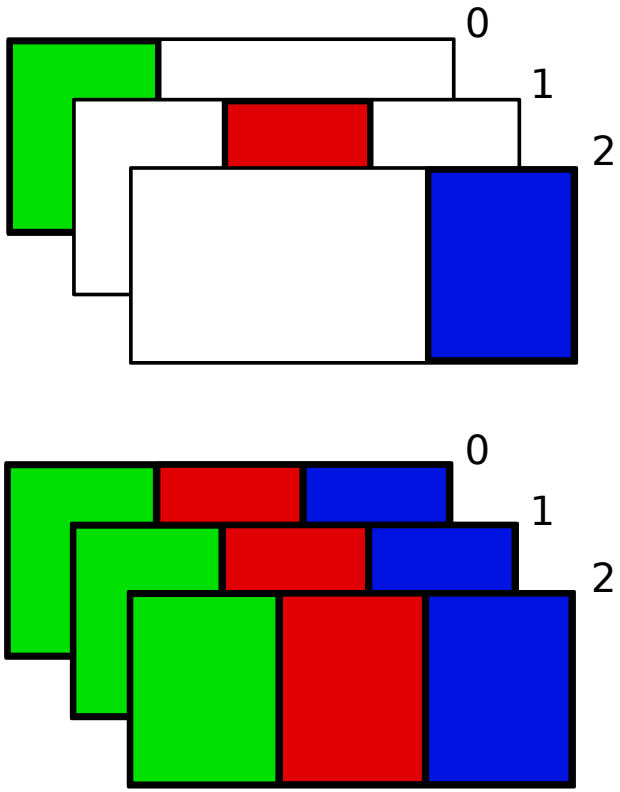
MPI_ALLREDUCE



Each process has calculated a global quantity, but only a fraction of it.

Using an MPI_ALLREDUCE we sum up this quantity and give it to all processes.

MPI_ALLGATHERV



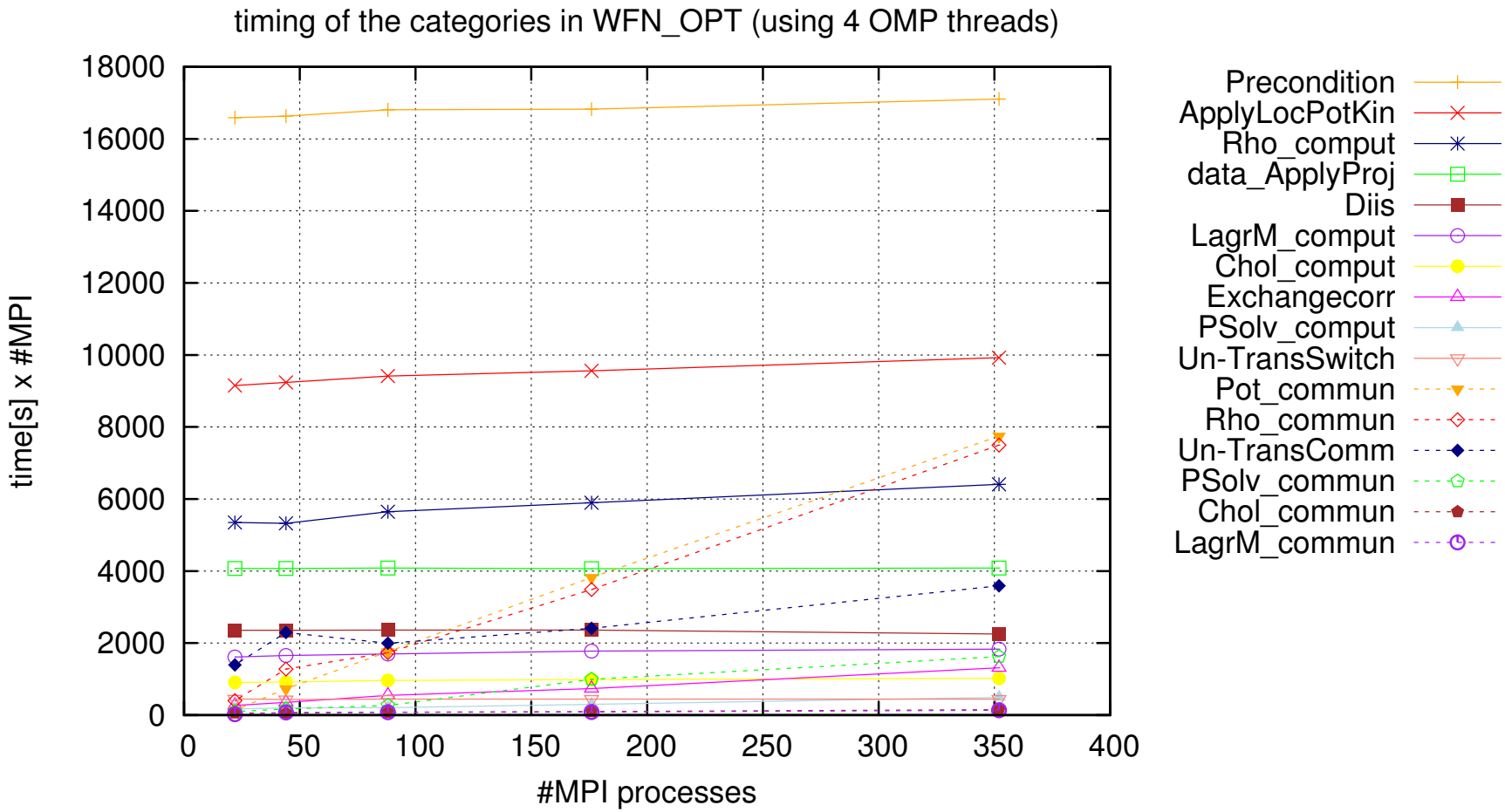
Each process has calculated some slice of a global quantity.

Using an MPI_ALLRGATHERV we collect these slices and distribute them to all processes.

Outline

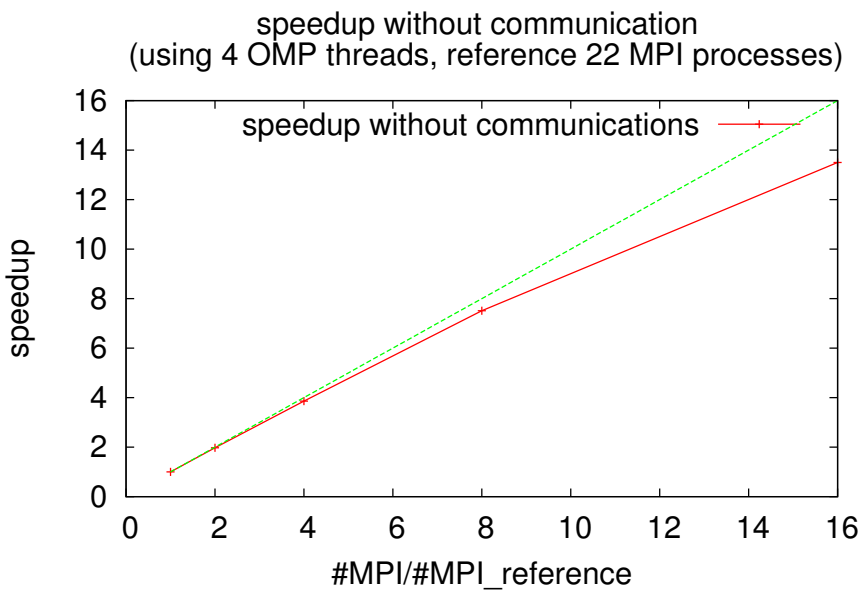
speedup of various categories

We want to determine the MPI speedup for various sections.



total speedup without communications

Here is the total MPI speedup excluding all communication parts:

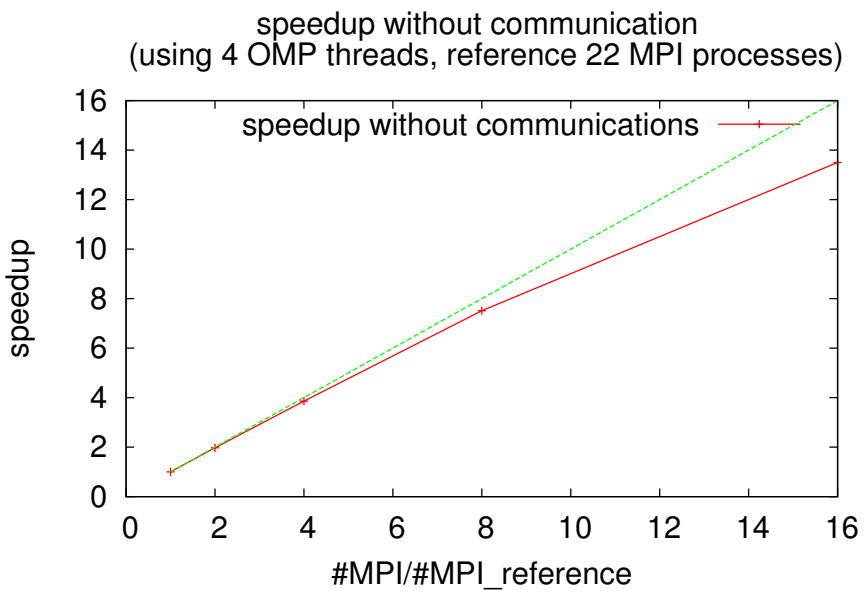


If we fit this curve to Amdahl's law $\frac{1}{(1-p)+\frac{p}{x}}$ we get a non-parallel amount of 1.2%. This would result in a maximal MPI speedup of 83.

Note: Most of the sequential code segments are parallelized with OpenMP, so the overall speedup is larger than this estimate!

total speedup without communications

Here is the total MPI speedup excluding all communication parts:

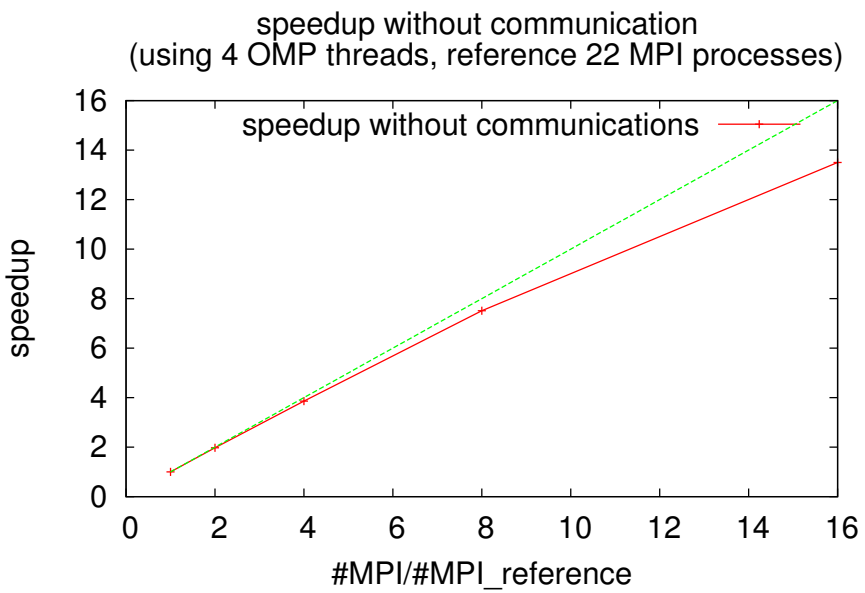


If we fit this curve to Amdahl's law $\frac{1}{(1-p)+\frac{p}{x}}$ we get a non-parallel amount of 1.2%. This would result in a maximal MPI speedup of 83.

Note: Most of the sequential code segments are parallelized with OpenMP, so the overall speedup is larger than this estimate!

total speedup without communications

Here is the total MPI speedup excluding all communication parts:



If we fit this curve to Amdahl's law $\frac{1}{(1-p)+\frac{p}{x}}$ we get a non-parallel amount of 1.2%. This would result in a maximal MPI speedup of 83.

Note: Most of the sequential code segments are parallelized with OpenMP, so the overall speedup is larger than this estimate!

Outline

OMP thread handles communication

Most of the MPI communications in BigDFT are done with collective routines and therefore don't allow for any overlap of communication and computation.

However such an overlap can be exploited as soon as several OpenMP threads are used:
1 OMP thread handles the communication, all other threads do the computation.

OMP thread handles communication

Most of the MPI communications in BigDFT are done with collective routines and therefore don't allow for any overlap of communication and computation.

However such an overlap can be exploited as soon as several OpenMP threads are used:
1 OMP thread handles the communication, all other threads do the computation.

performance gain

#cores	#MPI	#threads	no overlap [s]	overlap [s]	saving
352	176	2	551	545	1%
352	88	4	569	553	2%
352	44	8	703	618	12%
704	352	2	335	344	−2%
704	176	4	325	306	6%
704	88	8	367	349	5%
1408	352	4	222	206	7%
1408	176	8	228	209	8%

Outline

p2p would be natural choice

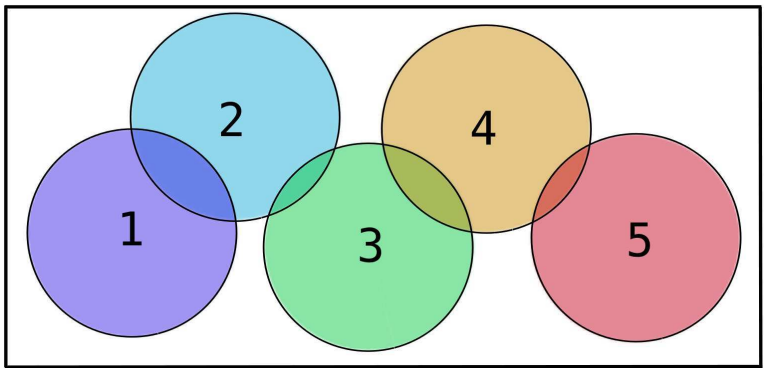
We are introducing new features into BigDFT where it seems to be natural to use p2p communications instead of collective communications.

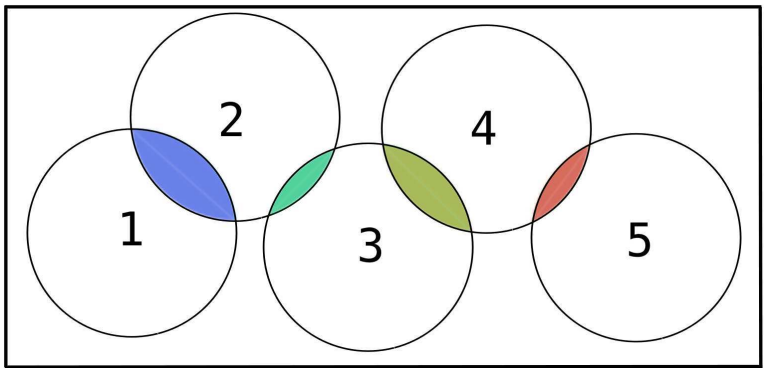
One such example is the overlap calculation for localized quantities:

p2p would be natural choice

We are introducing new features into BigDFT where it seems to be natural to use p2p communications instead of collective communications.

One such example is the overlap calculation for localized quantities:





p2p and collective implementation

We have to implementations to do this:

- p2p communication (MPI_ISEND, MPI_IRECV, MPI_WAITANY), i.e. each process communicates only with a subset of the other processes
- collective communication (MPI_ALLTOALLV), i.e. all processes communicate with all others

Here are the bandwidths we get for a typical problem size

p2p and collective implementation

We have to implementations to do this:

- p2p communication (MPI_ISEND, MPI_IRECV, MPI_WAITANY), i.e. each process communicates only with a subset of the other processes
- collective communication (MPI_ALLTOALLV), i.e. all processes communicate with all others

Here are the bandwidths we get for a typical problem size

p2p and collective implementation

We have to implementations to do this:

- p2p communication (MPI_ISEND, MPI_IRECV, MPI_WAITANY), i.e. each process communicates only with a subset of the other processes
- collective communication (MPI_ALLTOALLV), i.e. all processes communicate with all others

Here are the bandwidths we get for a typical problem size

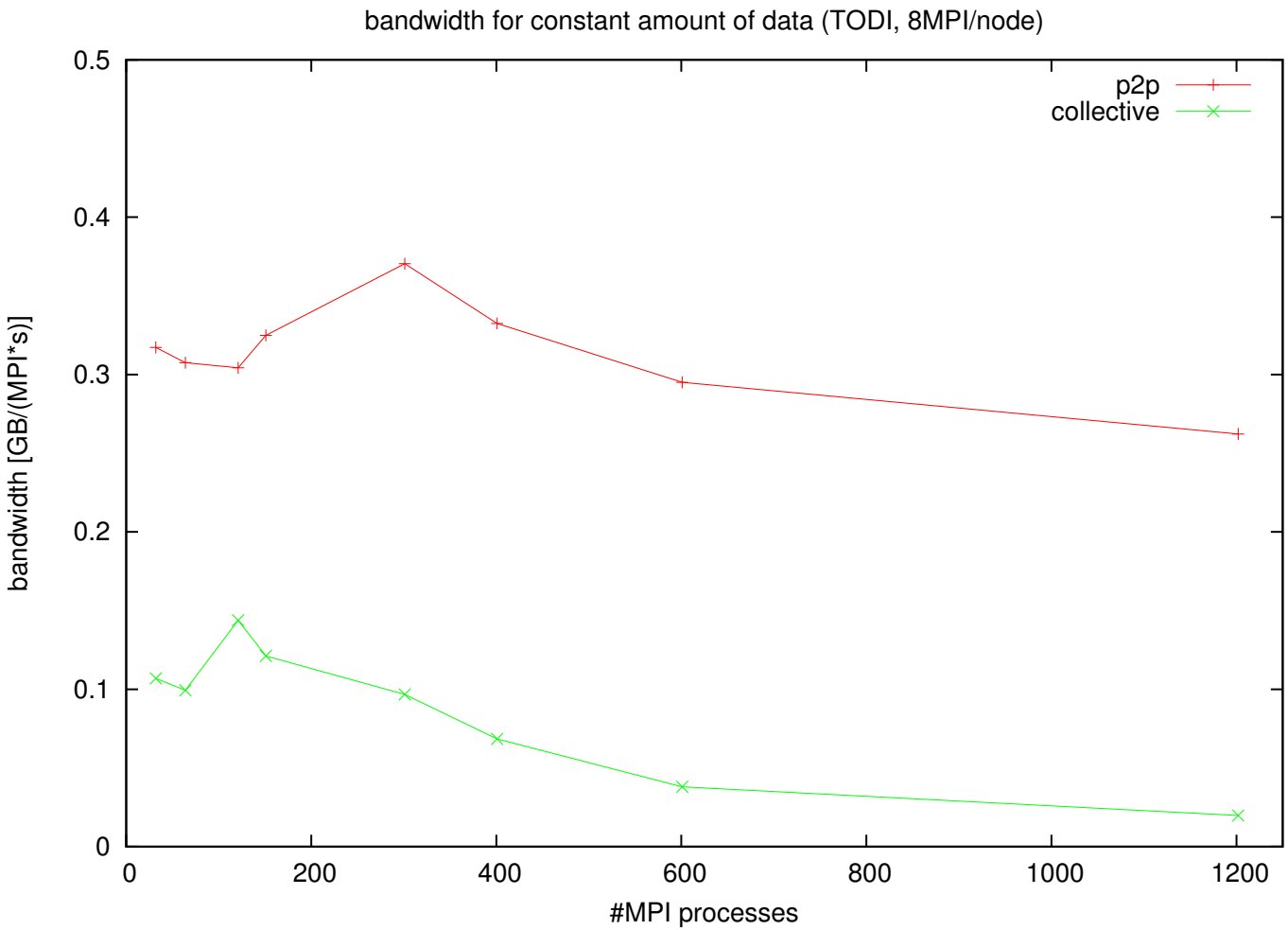
p2p and collective implementation

We have to implementations to do this:

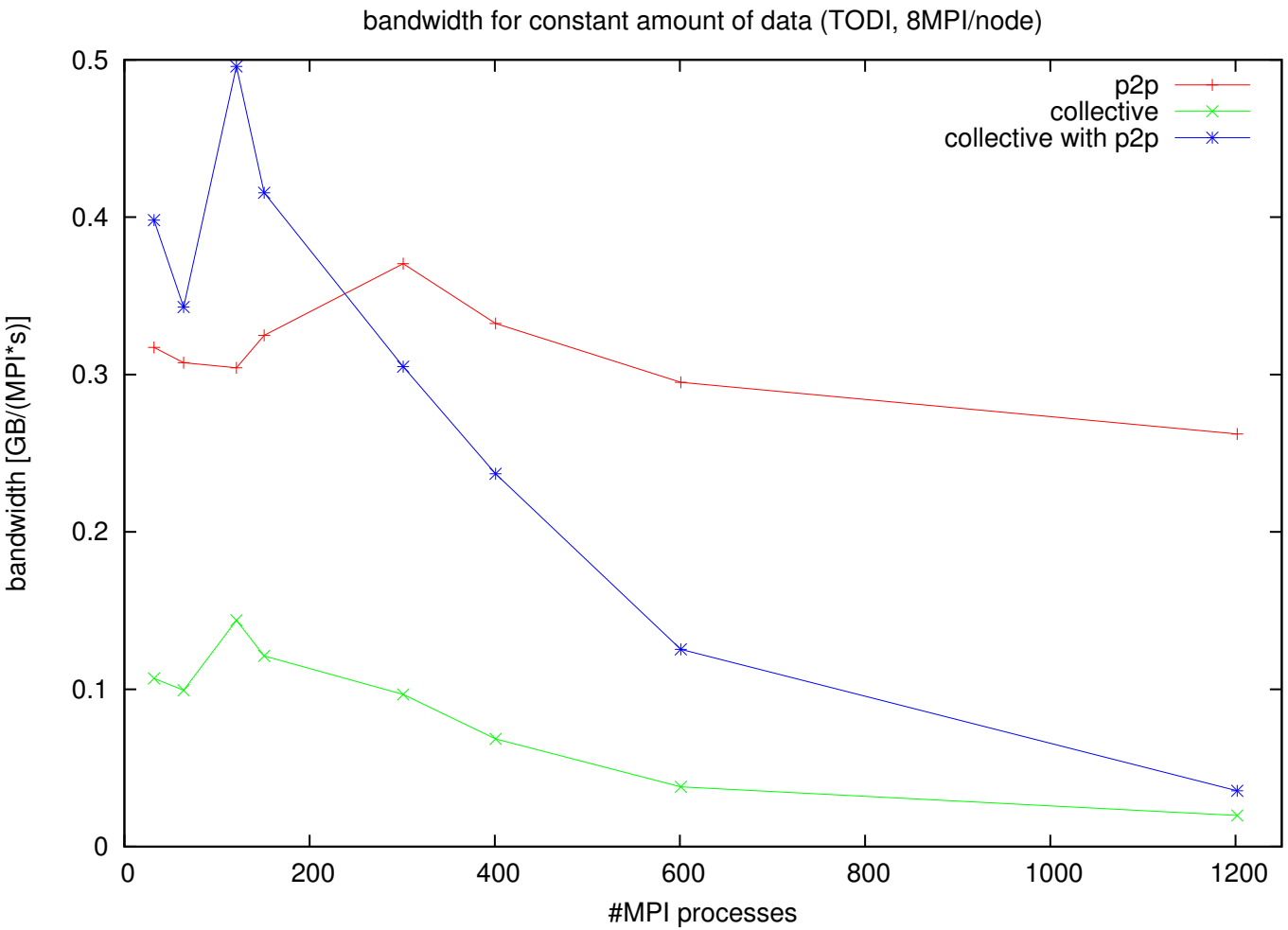
- p2p communication (MPI_ISEND, MPI_IRECV, MPI_WAITANY), i.e. each process communicates only with a subset of the other processes
- collective communication (MPI_ALLTOALLV), i.e. all processes communicate with all others

Here are the bandwidths we get for a typical problem size

bandwith for both implementations



bandwith for both implementations



time

Finally we care about time and not bandwidth!

The total amount of data to be sent is different for the modes:

- p2p: 43.85 GB
- collective: 1.04 GB

So in the end collective is still faster!

time

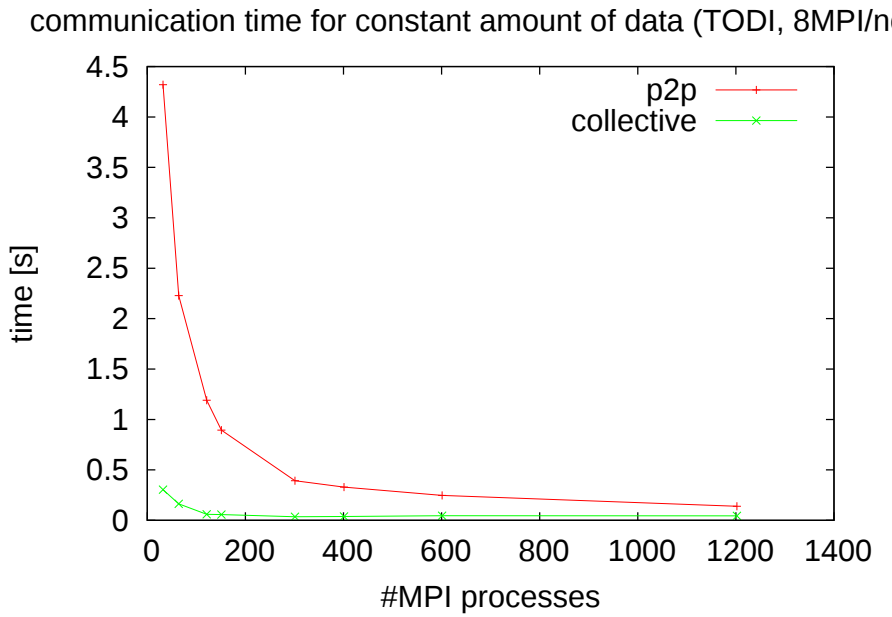
Finally we care about time and not bandwidth!

The total amount of data to be sent is different for the modes:

- p2p: 43.85 GB
- collective: 1.04 GB

So in the end collective is still faster!

time



Finally we care about time and not bandwidth!

The total amount of data to be sent is different for the modes:

- p2p: 43.85 GB
- collective: 1.04 GB

So in the end collective is still faster!

Outline

tested machines

We did test runs on various supercomputers:

- "Titane", a BullX machine (2 Xeon Nehalem quad-core per node, IB interconnect, mpibullx MPI libraries)
- "Babel", a IBM BG/P machine (4 cores per node, BlueGene interconnect, mpixlf90 compiler suite)
- "Rosa", a Cray XE6 (2 16-core AMD Opteron per node, Gemini interconnect)

tested machines

We did test runs on various supercomputers:

- "Titane", a BullX machine (2 Xeon Nehalem quad-core per node, IB interconnect, mpibullx MPI libraries)
- "Babel", a IBM BG/P machine (4 cores per node, BlueGene interconnect, mpixlf90 compiler suite)
- "Rosa", a Cray XE6 (2 16-core AMD Opteron per node, Gemini interconnect)

tested machines

We did test runs on various supercomputers:

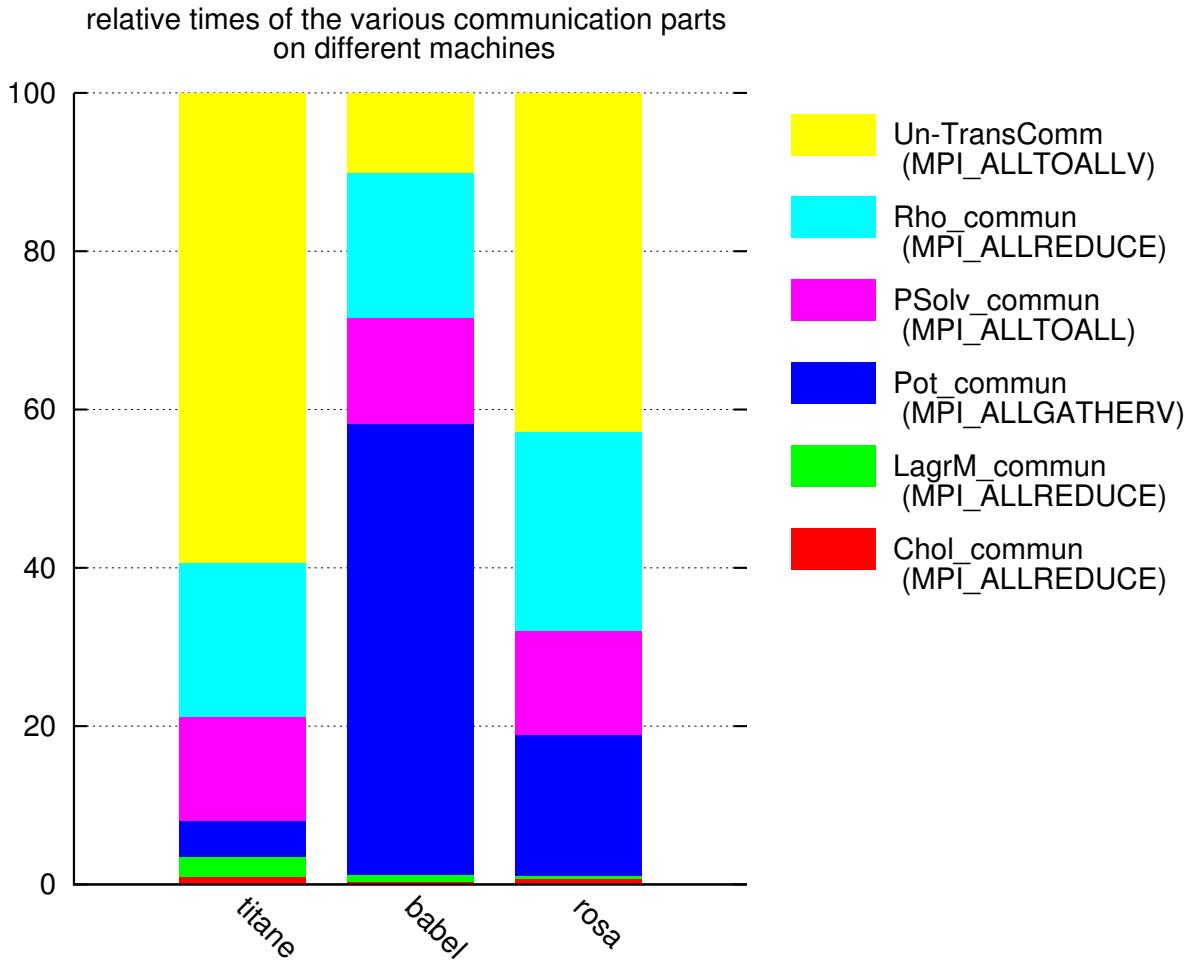
- "Titane", a BullX machine (2 Xeon Nehalem quad-core per node, IB interconnect, mpibullx MPI libraries)
- "Babel", a IBM BG/P machine (4 cores per node, BlueGene interconnect, mpixlf90 compiler suite)
- "Rosa", a Cray XE6 (2 16-core AMD Opteron per node, Gemini interconnect)

tested machines

We did test runs on various supercomputers:

- "Titane", a BullX machine (2 Xeon Nehalem quad-core per node, IB interconnect, mpibullx MPI libraries)
- "Babel", a IBM BG/P machine (4 cores per node, BlueGene interconnect, mpixlf90 compiler suite)
- "Rosa", a Cray XE6 (2 16-core AMD Opteron per node, Gemini interconnect)

relative timings



the optimal machine...