A Case for Standard Non-Blocking Collective Operations

T. Hoefler[1,2], P. Kambadur[1], R. L. Graham[3], G. Shipman[4] and A. Lumsdaine[1]

[1]Open Systems Lab
Indiana University
Bloomington, IN 47405, USA

[2]Computer Architecture Group
Technical University of Chemnitz
Chemnitz, 09111 Germany

[3]National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge TN 37831, USA

[4]Advanced Computing Laboratory
Los Alamos National Laboratory
Los Alamos, NM 87545, USA

EuroPVM/MPI 2007

Paris, France

1st October 2007

Collective Operations

S. Gorlatch: *"Send Receive considered harmful"*

"Parallel programming based on message passing can be improved by expressing communication in a structured manner without using send-receive statements."

Collective Operations

- provide common communication patterns
- improve performance (portability)
- programability and code maintenance (readability)
- avoid implementation errors

Overlapping Communication and Computation

J. Sancho: *"Quantifying the Potential Benefit of Overlapping Communication and Computation in Large-Scale Scientific Applications"*

"This result indicates that for a potentially large class of real-world applications, future high performance computing systems could use lower performance and more cost-effective networks without negatively impacting application performance if the overlap of communication with computation were fully exploited."

Overlap of Communication and Computation

- hides network latency
- enables the use of cheaper networking technology
- increases the tolerance of process skew
- collective communication overlap?

Non-blocking Collective Operations

R. Brightwell: *"Implications of Application Usage Characteristics for Collective Communication Offload"*

"Even so, current code shows the opportunity to overlap from thousands of instructions (several microseconds) to hundreds of thousands of instructions (approaching a millisecond) with many of the collective calls."

Non-blocking collectives

- combine many advantages
- are not available in the current MPI-2.0 standard
- begin to gain more and more attention
- missed by scientists who replace collective patterns with non-blocking point-to-point operations (cf. Danalis et al. *"Transformations to Parallel Codes for Communication-Computation Overlap"*)

Our Standard Proposal

- we define an API similar to MPI
- relaxes the MPI semantics (allows more than one collective per communicator)
- no tags (ordering is crucial for matching)
- no defined asynchronous progress - as we are used to ;-)

Programming Interface Example

```
MPI_Request request;
MPI_Ibarrier(comm, &request);
...
/* computation, other MPI communications */
...
MPI_Wait(request, status);
```

LibNBC - Non-blocking Collectives for MPI-2

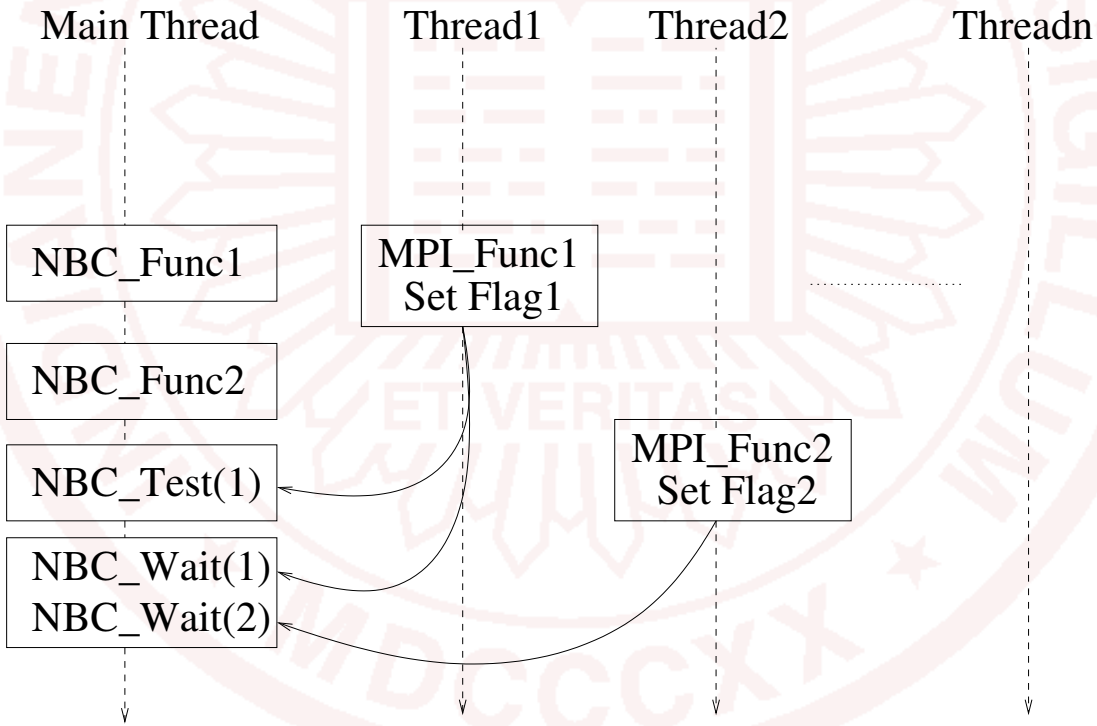
- open-source tuned implementation of asynchronous collectives
- two implementation options evaluated:
  - threads (recommended by MPI Forum)
  - point-to-point message based implementation

Programming Interface Example

```
NBC_Handle request;
NBC_Ibarrier(comm, &request);
...
/* computation, other MPI communications */
...
ret = NBC_Wait(request);
```

Implementation Option 1 - Threads

- needs MPI_THREAD_MULTIPLE
- task queue model
- occupies one CPU per process for communication
- MPI_COMM_DUP problems (Gropp et al. *"Issues in Developing a Thread-Safe MPI Implementation"*)



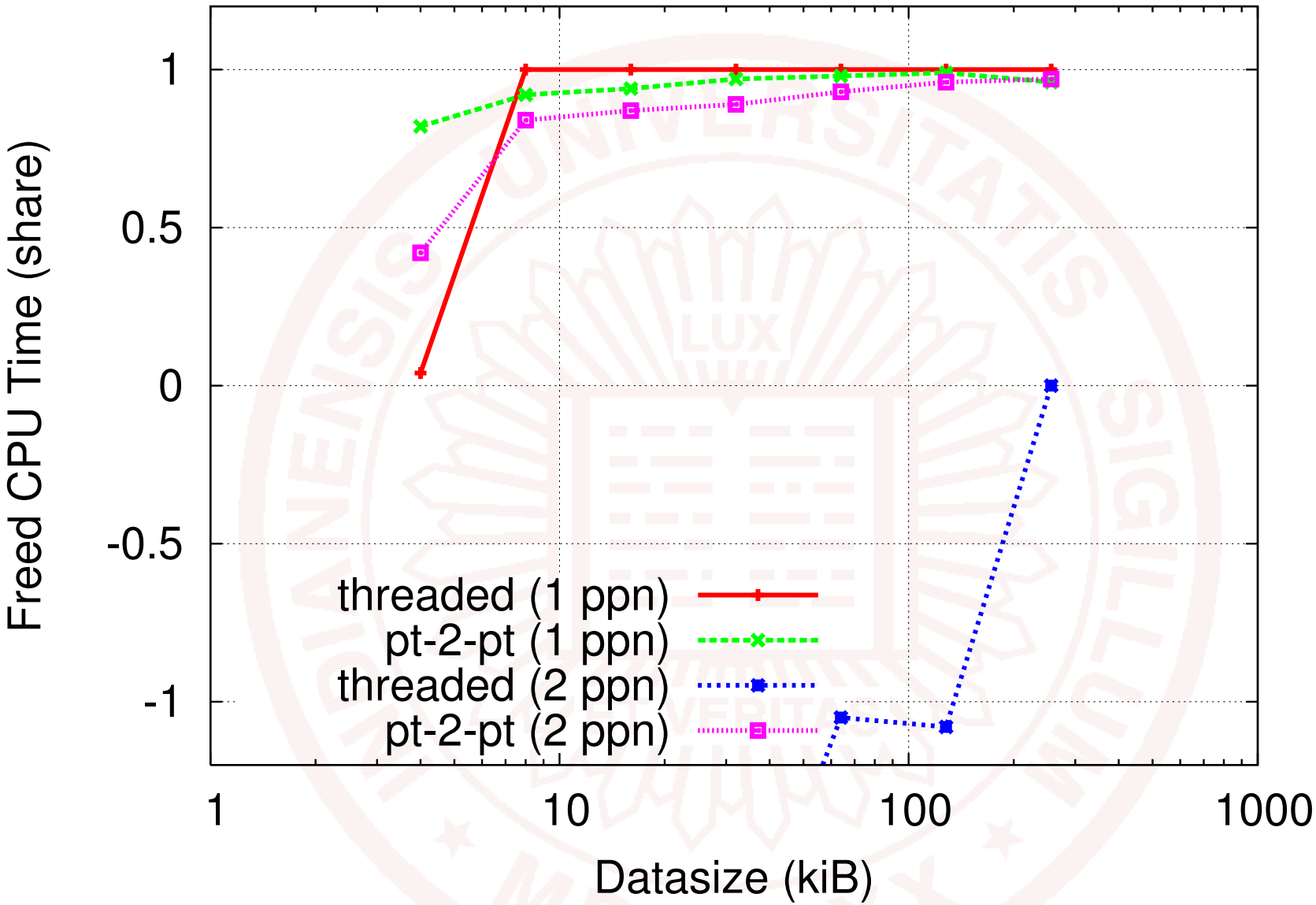
Implementation Option 2 - Point-to-point

- needs only MPI-1 functions (MPI-2 for Fortran bindings)
- schedule-based execution
- performance depends on overlap of ptp messages
- implementation of the fastest known algorithms (static selection)
- available since 2006, first users adapt their codes
- no matching with MPI collective functions

recv from 0	end	send to 3	end	send to 5

Figure: Binomial broadcast schedule on rank 1 of 6.

Microbenchmarks - Freed CPU Time



- 64/128 processes on 64 Coyote System nodes (Dual Opteron, IB)

Non-blocking 3D-FFT

Derivation from “normal” implementation

- distribution identical to “normal” 3D-FFT
- first FFT in z direction and index-swap identical

Design Goals to Minimize Communication Overhead

- start communication as early as possible
- achieve maximum overlap time

Solution

- start MPI_Ialltoall as soon as first xz-plane is ready
- calculate next xz-plane
- start next communication accordingly ...
- collect multiple xz-planes (tile factor)

Non-blocking 3D-FFT

Derivation from “normal” implementation

- distribution identical to “normal” 3D-FFT
- first FFT in z direction and index-swap identical

Design Goals to Minimize Communication Overhead

- start communication as early as possible
- achieve maximum overlap time

Solution

- start MPI_Ialltoall as soon as first xz-plane is ready
- calculate next xz-plane
- start next communication accordingly ...
- collect multiple xz-planes (tile factor)

Non-blocking 3D-FFT

Derivation from “normal” implementation

- distribution identical to “normal” 3D-FFT
- first FFT in z direction and index-swap identical

Design Goals to Minimize Communication Overhead

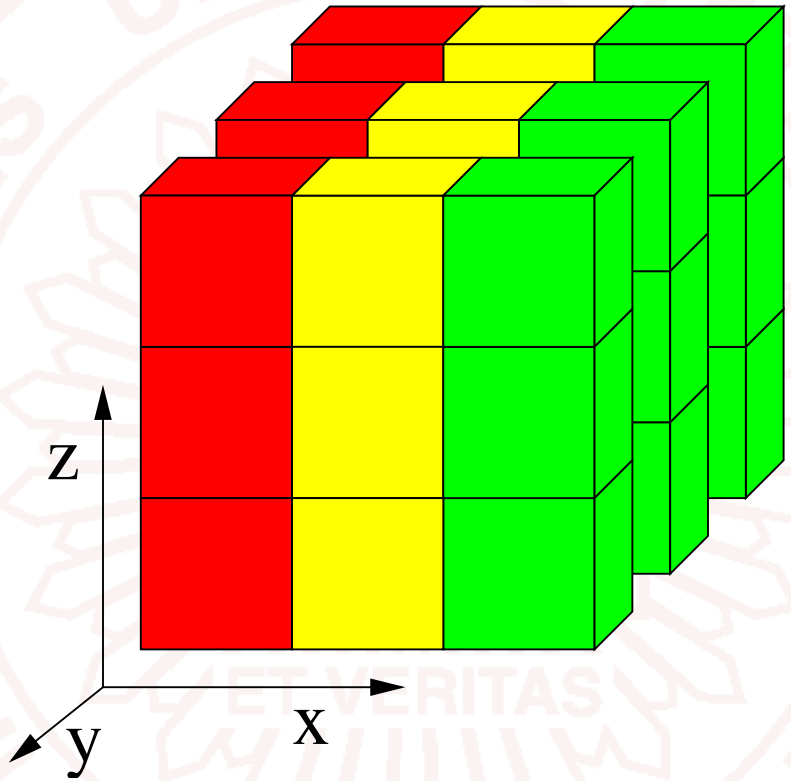
- start communication as early as possible
- achieve maximum overlap time

Solution

- start MPI_Ialltoall as soon as first xz-plane is ready
- calculate next xz-plane
- start next communication accordingly ...
- collect multiple xz-planes (tile factor)

Transformation in z Direction

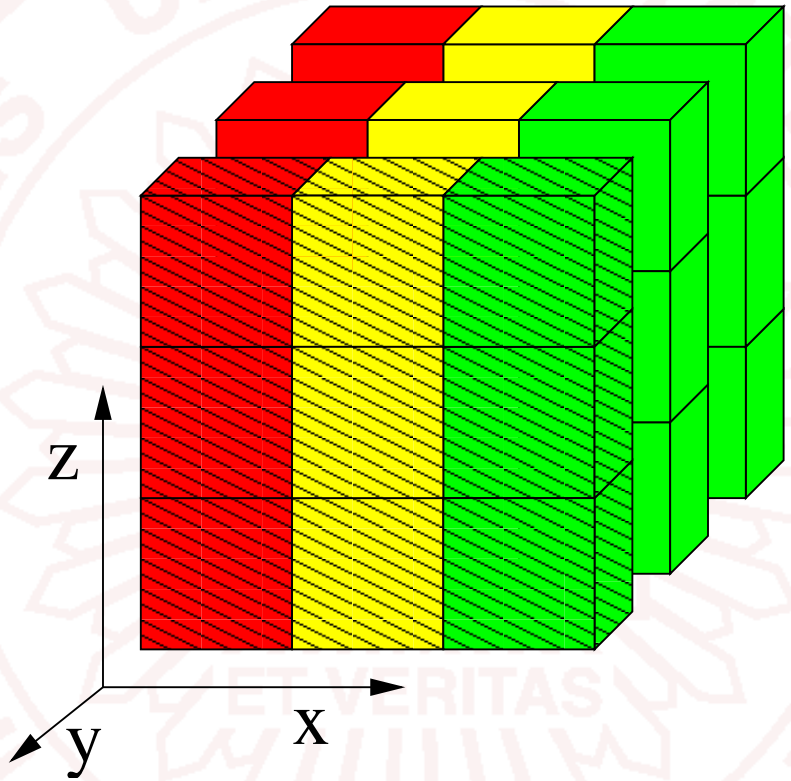
Data already transformed in y direction



1 block = 1 double value (3x3x3 grid)

Transformation in z Direction

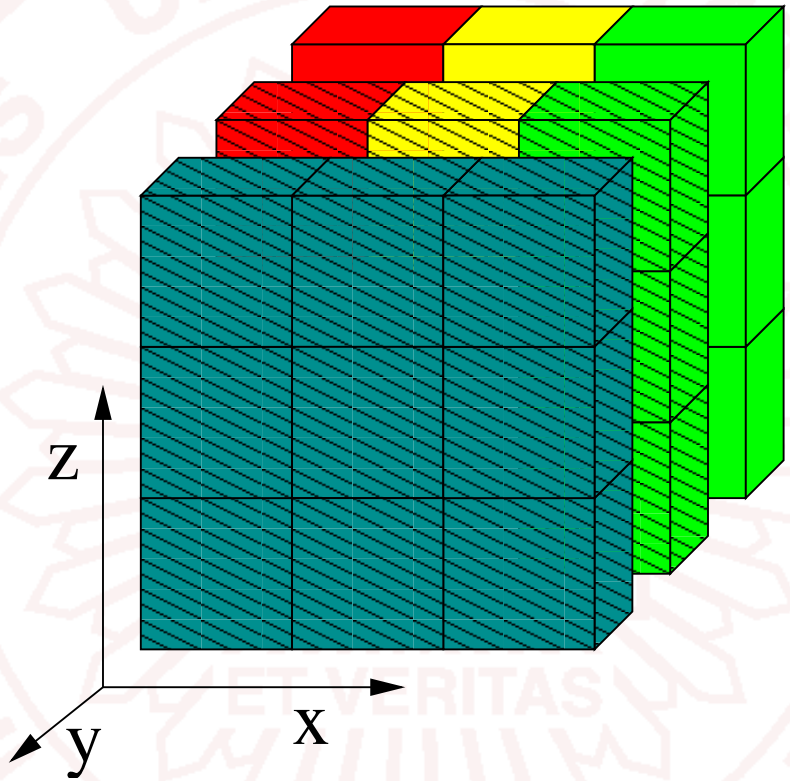
Transform first xz plane in z direction



pattern means that data was transformed in y and z direction

Transformation z Direction

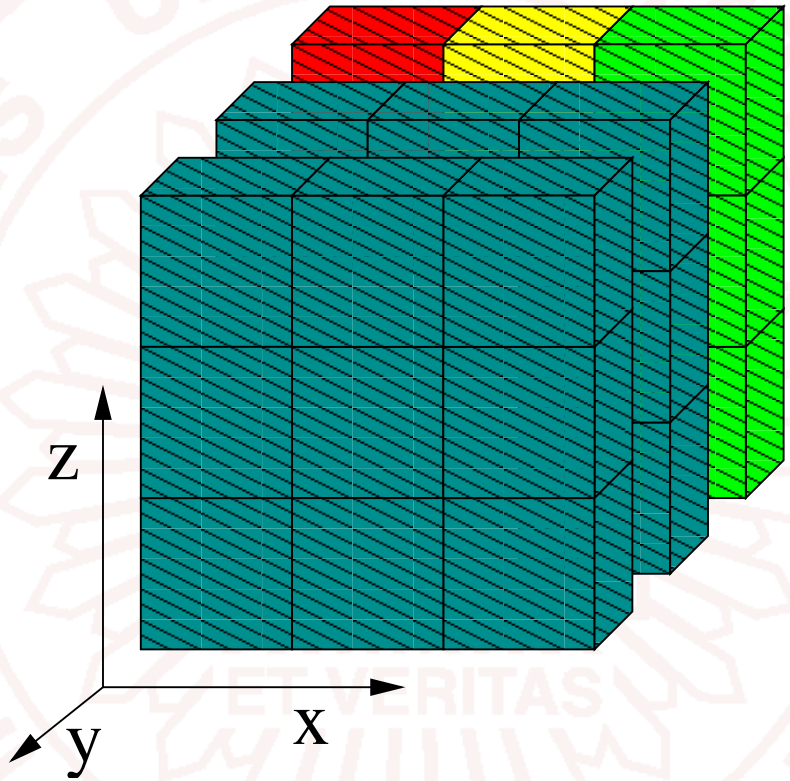
start MPI_Ialltoall of first xz plane and transform second plane



cyan color means that data is communicated in the background

Transformation in z Direction

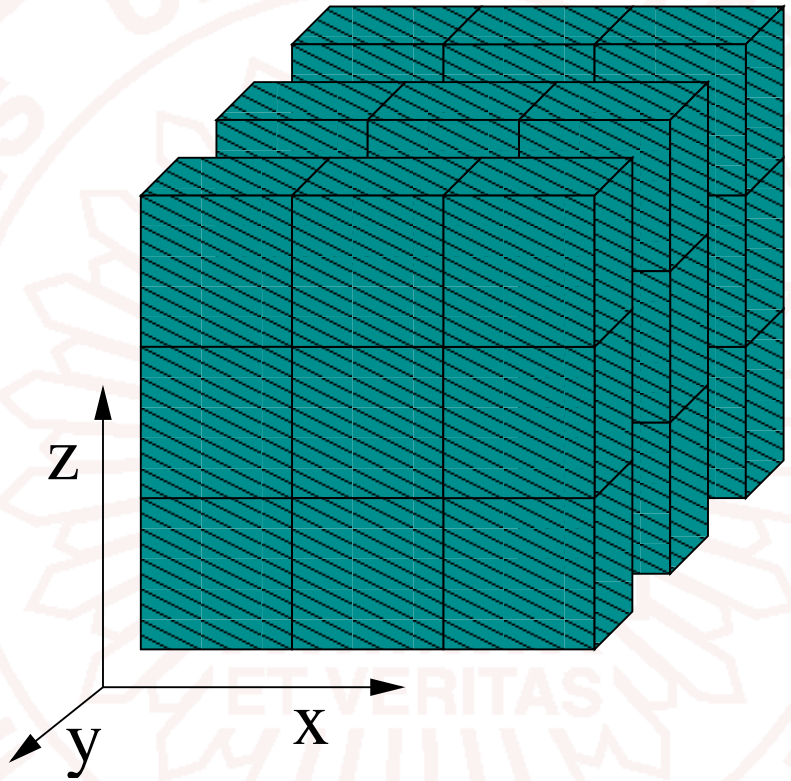
start MPI_Ialltoall of second xz plane and transform third plane



data of two planes is not accessible due to communication

Transformation in x Direction

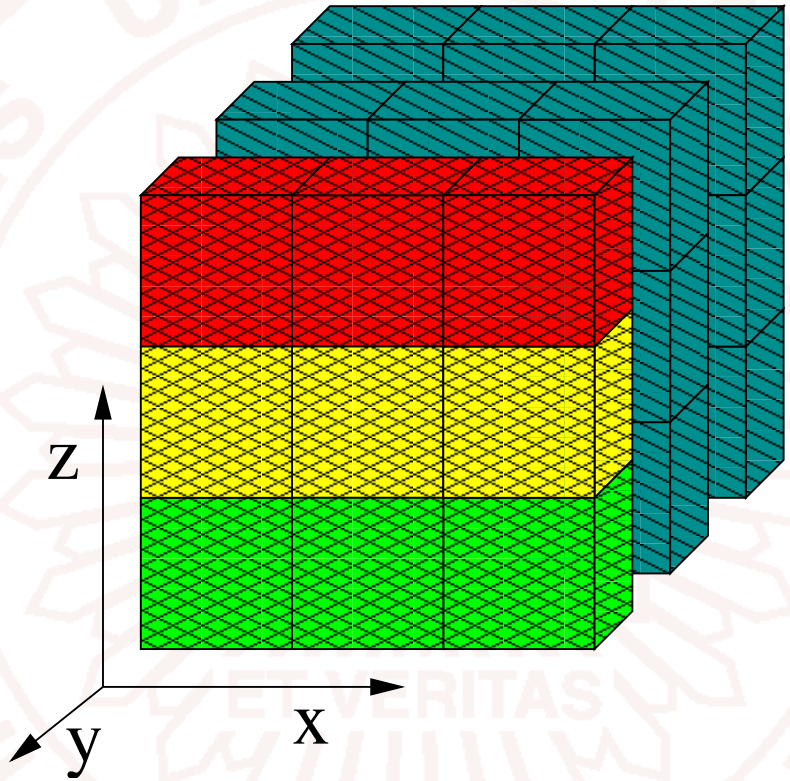
start communication of the third plane and ...



we need the first xz plane to go on ...

Transformation in x Direction

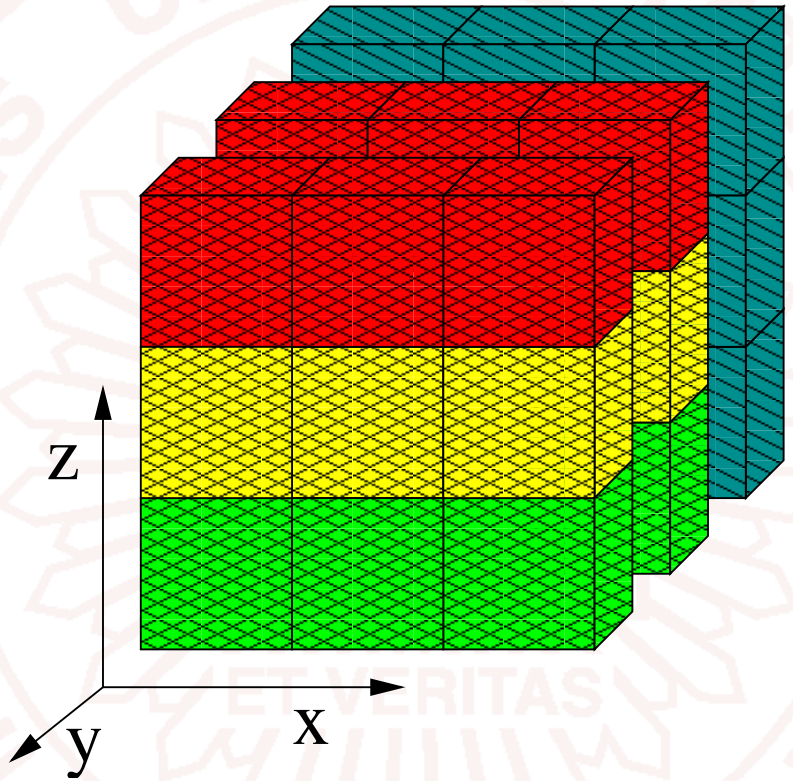
... so MPI_Wait for the first MPI_Ialltoall!



and transform first plane (new pattern means xyz transformed)

Transformation in x Direction

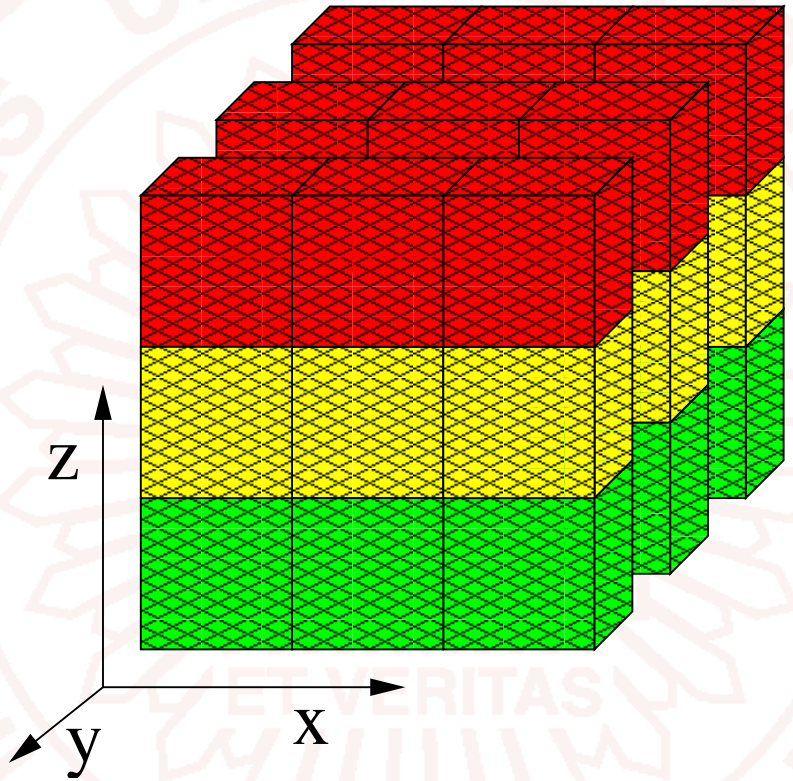
Wait and transform second xz plane



first plane's data could be accessed for next operation

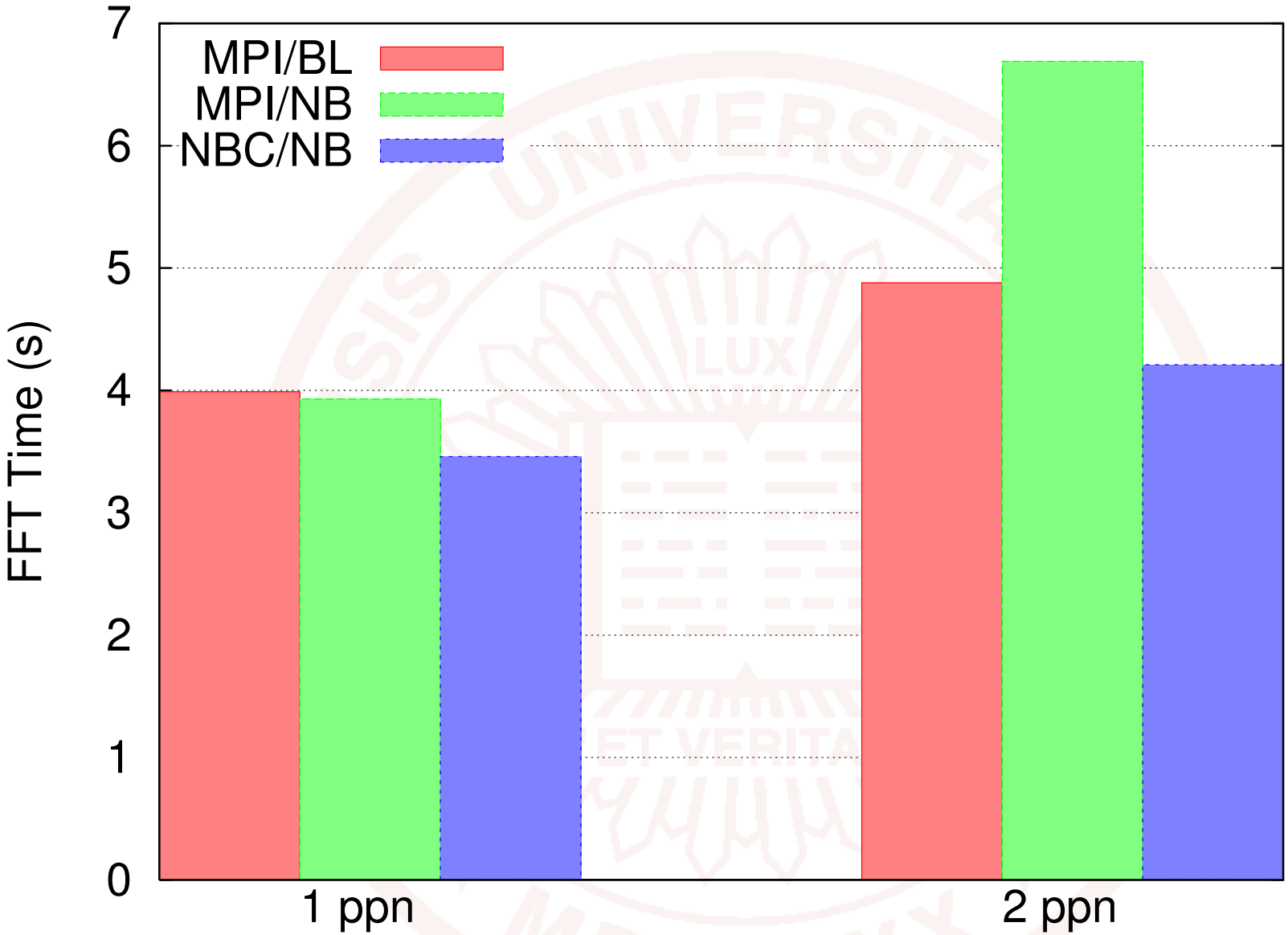
Transformation in x Direction

wait and transform last xz plane



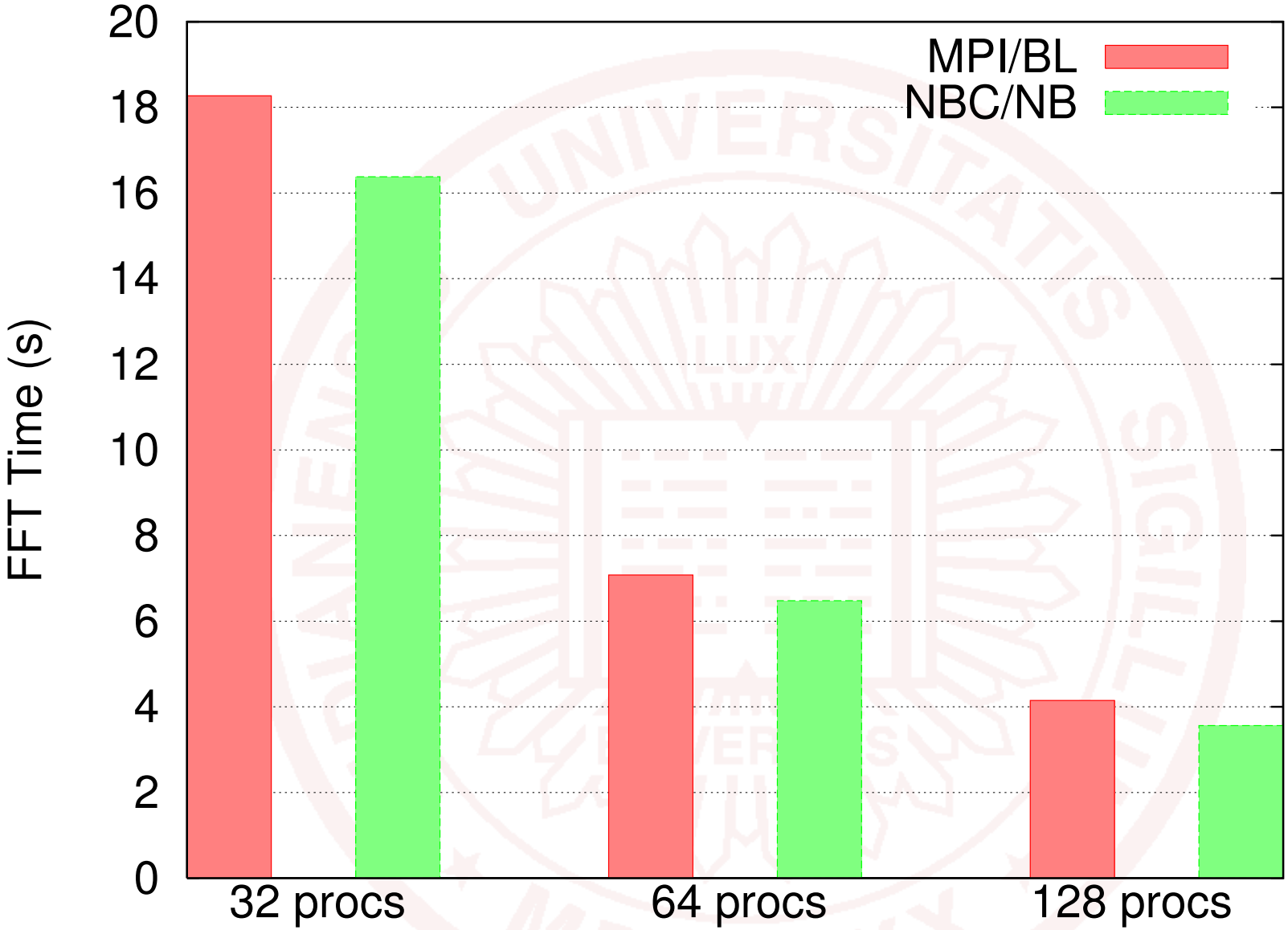
done! $\rightarrow$ 1 complete 1D-FFT overlaps a communication

1024^3 3d-FFT over InfiniBand



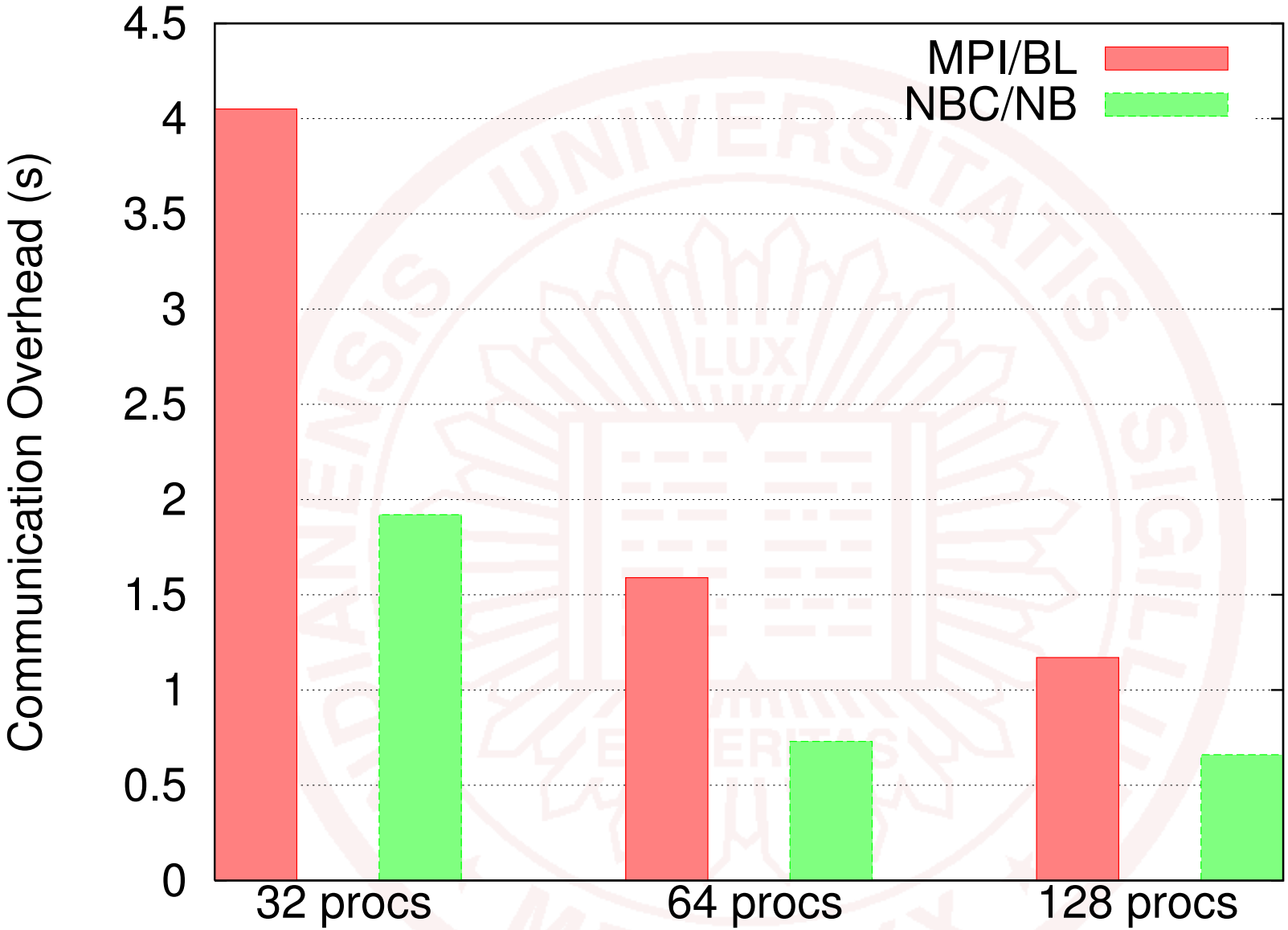
- P=128, “Coyote”@LANL - 128/64 dual socket 2.6GHz Opteron nodes

1024^3 3d-FFT on the XT4



- "Jaguar"@ORNL - Cray XT4, dual socket dual core 2.6GHz Opteron

1024^3 3d-FFT on the XT4 (Communication Overhead)



- "Jaguar"@ORNL - Cray XT4, dual socket dual core 2.6GHz Opteron

It works, why do we need it standardized?

- enable hardware-optimized implementations
- solve software technological dilemma of double implementation
- enable matching of blocking and non-blocking collectives
- use MPI test and wait functions
- overcome problems with first call and `Comm_dup()`
- $\Rightarrow$ widen user-base

Conclusions and Future Work

Conclusions

- a threaded implementation “wastes” cores
  - may be acceptable in the future
  - not on today’s dual/tri/quad core systems
  - $\Rightarrow$ LibNBC is currently implemented on top of pt2pt
- standardization would solve many problems!

Future Work:

- tuning LibNBC and adding it to Open MPI
- port applications to use NBC
- $\Rightarrow$ We would like to collaborate with scientists!

LibNBC Download/Further Information

`http://www.unixer.de/research/nbcoll/`

Conclusions and Future Work

Conclusions

- a threaded implementation “wastes” cores
  - may be acceptable in the future
  - not on today’s dual/tri/quad core systems
  - $\Rightarrow$ LibNBC is currently implemented on top of pt2pt
- standardization would solve many problems!

Future Work:

- tuning LibNBC and adding it to Open MPI
- port applications to use NBC
- $\Rightarrow$ We would like to collaborate with scientists!

LibNBC Download/Further Information

http://www.unixer.de/research/nbcoll/

Conclusions and Future Work

Conclusions

- a threaded implementation “wastes” cores
  - may be acceptable in the future
  - not on today’s dual/tri/quad core systems
  - $\Rightarrow$ LibNBC is currently implemented on top of pt2pt
- standardization would solve many problems!

Future Work:

- tuning LibNBC and adding it to Open MPI
- port applications to use NBC
- $\Rightarrow$ We would like to collaborate with scientists!

LibNBC Download/Further Information

`http://www.unixer.de/research/nbcoll/`