Application Optimization with non-blocking Collective Operations

– A case study with a three-dimensional FFT –

Torsten Höfler

Department of Computer Science
Indiana University / Technical University of Chemnitz

Commissariat à l'Énergie Atomique
Direction des applications militaires (CEA-DAM)
Bruyéres-le-chatel, France
12th January 2007

Outline

Outline

What is it?

Non-blocking Send/Recv

- MPI_Isend/MPI_Irecv + MPI_Test/MPI_Wait
- avoid deadlock situations and enable overlap

Collective Operations

- MPI_Bcast/MPI_Reduce/...
- often-used comm. patterns and performance portability
- $\rightarrow$ cf. BLAS for communication

Non-blocking Collective Operations

- MPI_Ibcast/MPI_Ireduce/... + MPI_Test/MPI_Wait
- combines all advantages
- overlap + performance portability

What is it?

Where do I find it in the Standard?

- not part of MPI-2
- explicit programming model (threads) $\Rightarrow$ not viable
- implemented as an addition to MPI-2

Why should I invest the additional effort?

- two main advantages:
  1. hide communication latency
  2. lower the effects of process skew (introduced by OS noise or the algorithm)

What is it?

Where do I find it in the Standard?

- not part of MPI-2
- explicit programming model (threads) ⇒ not viable
- implemented as an addition to MPI-2

Why should I invest the additional effort?

- two main advantages:
  1. hide communication latency
  2. lower the effects of process skew (introduced by OS noise or the algorithm)

What is overlap and how does it help?

Hardware Parallelism

- today's computers communicate without CPU involvement
- communication in the background, CPU is freed

Ah, my program runs faster!?

- not much - "blocking communication" blocks the CPU :-(
- CPU waits until the communication is finished
- non-blocking communication gives control to the user

But I heard that non-blocking Send/Recv is slow

- depends on the MPI library
- some are implemented badly
  (e.g. operation is performed blocking during MPI_Wait)

What is overlap and how does it help?

Hardware Parallelism

- today's computers communicate without CPU involvement
- communication in the background, CPU is freed

Ah, my program runs faster!?

- not much - "blocking communication" blocks the CPU :-(
- CPU waits until the communication is finished
- non-blocking communication gives control to the user

But I heard that non-blocking Send/Recv is slow

- depends on the MPI library
- some are implemented badly
(e.g. operation is performed blocking during MPI_Wait)

What is overlap and how does it help?

Hardware Parallelism

- today's computers communicate without CPU involvement
- communication in the background, CPU is freed

Ah, my program runs faster!?

- not much - "blocking communication" blocks the CPU :-(
- CPU waits until the communication is finished
- non-blocking communication gives control to the user

But I heard that non-blocking Send/Recv is slow

- depends on the MPI library
- some are implemented badly
  (e.g. operation is performed blocking during MPI_Wait)

What can I gain with overlap?

The Latency of Collective Operations

- often implemented on top of point-to-point messages
- scales logarithmic $O(log_2 P)$ or linear $O(P)$ in P

Ok, how much is that?

- simple network model (Hockney) with 1 byte messages
- time to send from host i to host j ($j \neq i$): L
- L is network dependent:
  - Fast Ethernet: $L = 50 - 60\mu s$
  - Gigabit Ethernet: $L = 15 - 20\mu s$
  - InfiniBand™ : $L = 2 - 7\mu s$

$\Rightarrow$ $1\mu s \approx 4000$ FLOP of a 2GHz Machine

What can I gain with overlap?

The Latency of Collective Operations

- often implemented on top of point-to-point messages
- scales logarithmic $O(log_2 P)$ or linear $O(P)$ in P

Ok, how much is that?

- simple network model (Hockney) with 1 byte messages
- time to send from host i to host j ($j \neq i$): L
- L is network dependent:
  - Fast Ethernet: $L = 50 - 60\mu s$
  - Gigabit Ethernet: $L = 15 - 20\mu s$
  - InfiniBand[TM] : $L = 2 - 7\mu s$

$\Rightarrow$ $1\mu s \approx 4000$ FLOP of a 2GHz Machine

Process Skew

- caused by OS interference or unbalanced application
- especially if processors are overloaded
- worse for big systems
- can cause dramatic performance decrease
- all nodes wait for the last

Example

Petrini et. al. (2003) *"The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q"*

Process Skew

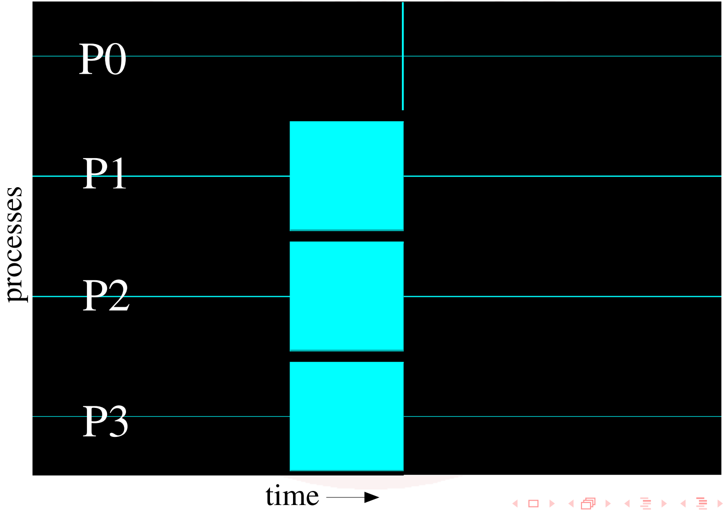
- caused by OS interference or unbalanced application
- especially if processors are overloaded
- worse for big systems
- can cause dramatic performance decrease
- all nodes wait for the last

Example

Petrini et. al. (2003) *"The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q"*

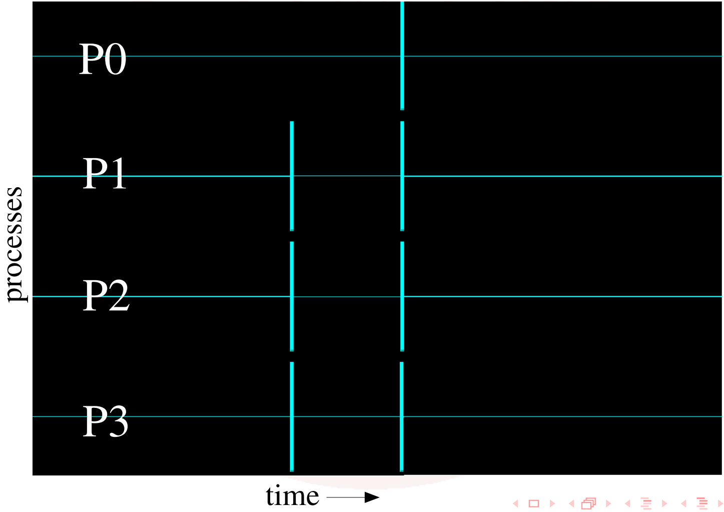
Process Skew - MPI_BCAST Example - Jumpshot

process 0 delayed, black=calculation time, blue=MPI time



Process Skew - MPI_IBCAST Example - Jumpshot

process 0 delayed, black=calculation time, blue=MPI time



Great! How do I use it?

Proposal & Interface Definition

Hoefler et. al. (2006): *"Non-Blocking Collective Operations for MPI-2"*

Implementation - LibNBC

- needs only ANSI C + MPI-1
- BSD License
- download from `http://www.unixer.de/NBC`

LibNBC Usage

```
NBC_Ibcast(buf1, p, MPI_INT, 0, comm, &req);
NBC_Wait(&req);
```

Great! How do I use it?

Proposal & Interface Definition

Hoefler et. al. (2006): *“Non-Blocking Collective Operations for MPI-2”*

Implementation - LibNBC

- needs only ANSI C + MPI-1
- BSD License
- download from `http://www.unixer.de/NBC`

LibNBC Usage

```
NBC_Ibcast(buf1, p, MPI_INT, 0, comm, &req);
NBC_Wait(&req);
```

Outline

Acknowledgements

I want to thank some inspiring people!
(alphabetically)

- George Bosilca, University of Tennessee (LibNBC)
- Peter Gottschling, Indiana University (3D-CG Solver, Apps)
- Andrew Lumsdaine, Indiana University (LibNBC, Apps)
- Wolfgang Rehm, TU Chemnitz (LibNBC, Apps)
- Jeff Squyres, Cisco Systems (LibNBC)
- Gilles Zerah, CEA-DAM France (problem of 3D-FFT)

(incomplete) Classification of parallel Algorithms

Independent Data Applications

- 3D-CG Poisson solver (inner and halo parts)
- many implicit iterative solvers (inner and halo parts)

Independent Data in Loops

- parallel compression (blocks independent)
- multi-dimensional FFT (lines/planes independent)

Dependent Data in Loops

- parallel Gauss Method (HPL, panel broadcast)
- parallel Cholesky (strong data dependency)

(incomplete) Classification of parallel Algorithms

Independent Data Applications

- 3D-CG Poisson solver (inner and halo parts)
- many implicit iterative solvers (inner and halo parts)

Independent Data in Loops

- parallel compression (blocks independent)
- multi-dimensional FFT (lines/planes independent)

Dependent Data in Loops

- parallel Gauss Method (HPL, panel broadcast)
- parallel Cholesky (strong data dependency)

(incomplete) Classification of parallel Algorithms

Independent Data Applications

- 3D-CG Poisson solver (inner and halo parts)
- many implicit iterative solvers (inner and halo parts)

Independent Data in Loops

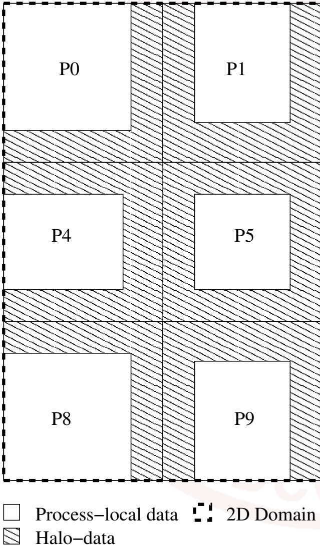
- parallel compression (blocks independent)
- multi-dimensional FFT (lines/planes independent)

Dependent Data in Loops

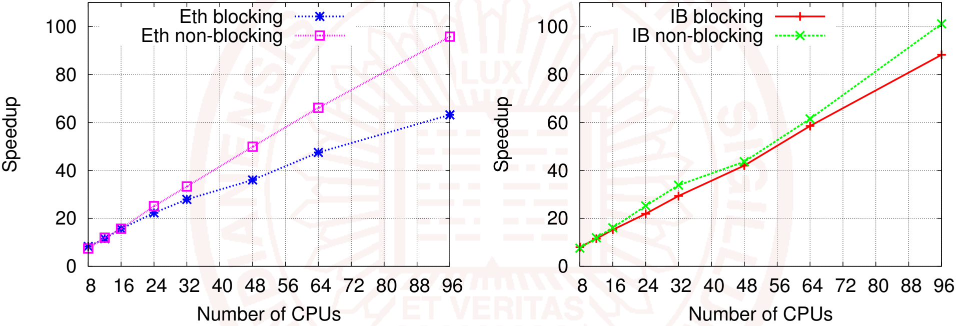
- parallel Gauss Method (HPL, panel broadcast)
- parallel Cholesky (strong data dependency)

An independent data Algorithm

3D Poisson Solver



3D-Poisson - Parallel Speedup (Best Case)



- “odin”@IU: 128 2 GHz dual Opteron 246 nodes
- Interconnect: Gigabit Ethernet, InfiniBand^{TM}
- System size 800x800x800 (1 node $\approx$ 5300s)

Parallel Compression

- block-by-block parallel compression
- gather compressed data to a single node
- compression could also be post-processing
- widely used to record experimental data

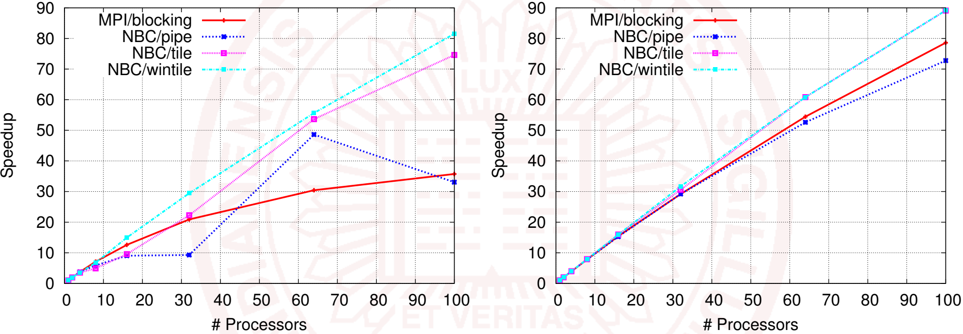
```
for(i=0; i < my_blocks; i++) {
 compress_block(i);
}
MPI_Gather(<block 0 to my_blocks-1>);
```

Pipelined Communication

- start non-blocking communication after some data is ready
- two parameters:
  1. tile-factor: number of elements per communication
  2. window-size: number of outstanding requests

```
for(i=0; i < my_blocks/tile; i++) {
 for(j=0; j < tile; j++)
  compress_block(i*tile + j);
 MPI_Igather(<block i to i+tile-1>);
}
MPI_Waitall(<Igather requests>);
```

Compression - Parallel Speedup (Best Case)

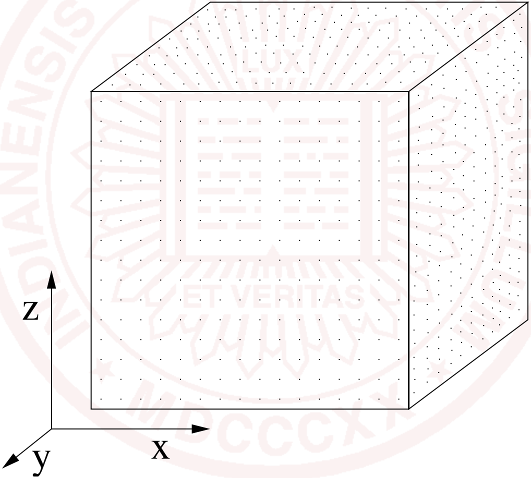


- “odin”@IU: 128 2 GHz dual Opteron 246 nodes
- Interconnect: Gigabit Ethernet, InfiniBand^{TM}
- System size 57.22 MB (1 node ≈ 9800s)

Outline

Domain Decomposition

Discretized 3D Domain (FFT-Box)



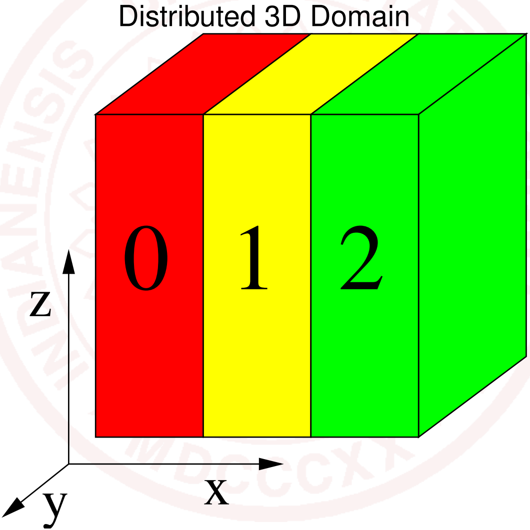
Domain Decomposition

Memory layout (3x3x3 box)
(coordinates xyz: 000 $\rightarrow$ 222)

	000	001	002	010	011	012	···
···	020	021	022	100	101	102	···
···	110	111	112	120	121	122	···
···	200	201	202	210	211	212	···
···	220	221	222				

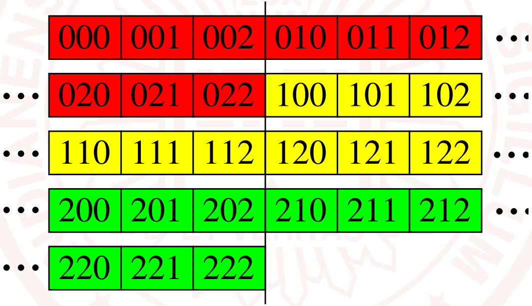
A parallel 3D-FFT

Domain Decomposition



Domain Decomposition

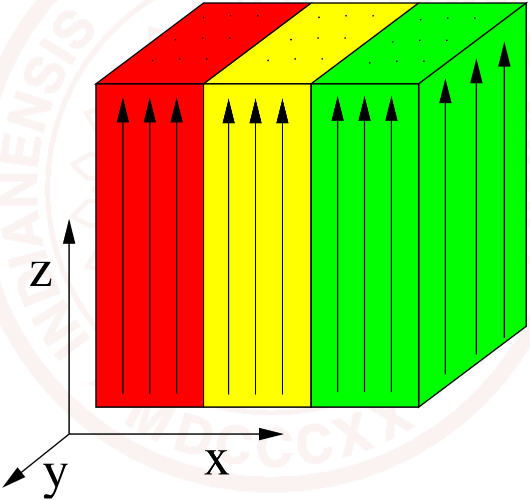
Blocked data distribution



A parallel 3D-FFT

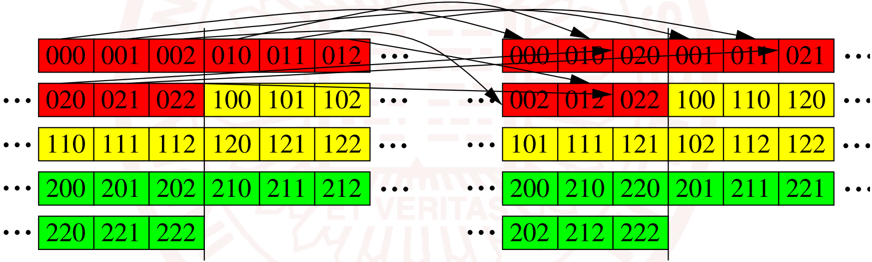
1D Transformation

1D Transformation in z Direction

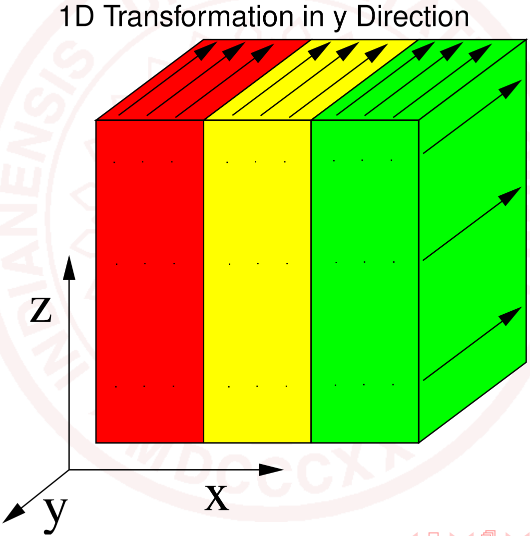


Rearrange Data Layout

rearrange from xyz to xzy (simply swap y and z indices)



1D Transformation

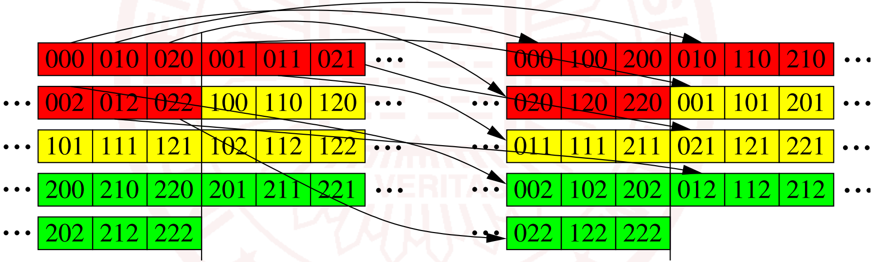


A parallel 3D-FFT

Rearrange Data Layout

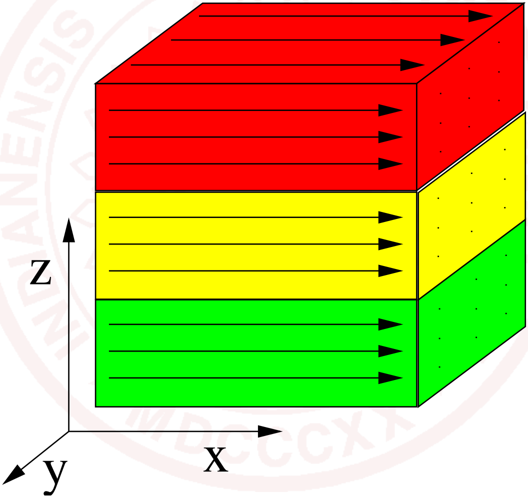
rearrange from xzy to yzx (parallel transpose)

⇒ MPI_Alltoall(v)



1D Transformation

1D Transformation in x Direction



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

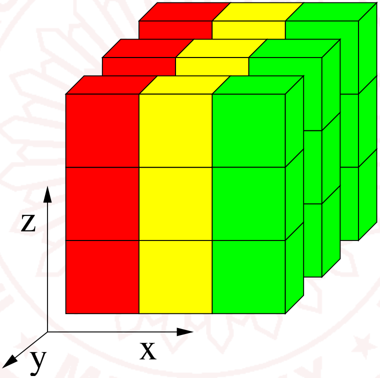
- start communication as early as possible
- achieve maximum overlap time

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

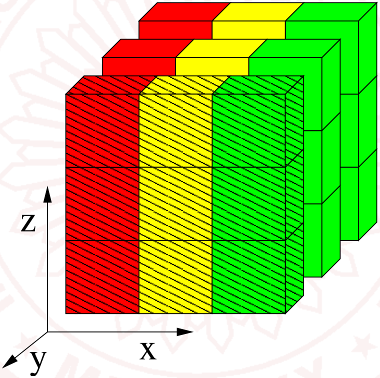
Data already transformed in y direction



1 block = 1 double value (3x3x3 grid)

Transformation in z Direction

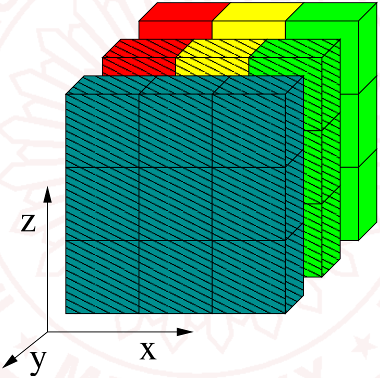
Transform first xz plane in z direction



pattern means that data was transformed in y and z direction

Transformation z Direction

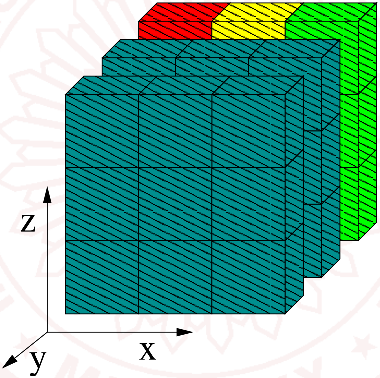
start MPI_Ialltoall of first xz plane and transform second plane



cyan color means that data is communicated in the background

Transformation in z Direction

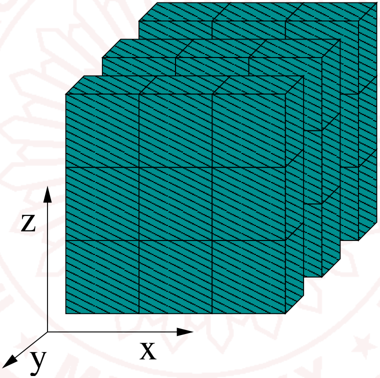
start MPI_Ialltoall of second xz plane and transform third plane



data of two planes is not accessible due to communication

Transformation in x Direction

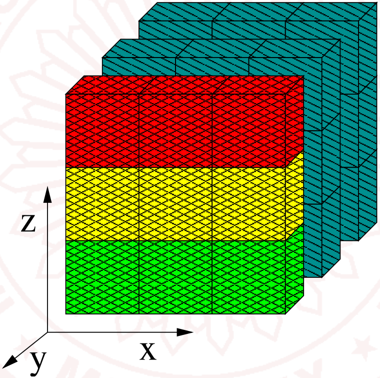
start communication of the third plane and ...



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

Transformation in x Direction

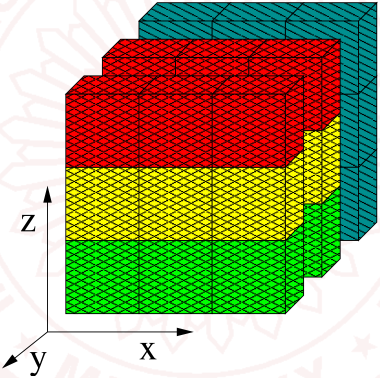
... so MPI_Wait for the first MPI_Ialltoall!



and transform first plane (new pattern means xyz transformed)

Transformation in x Direction

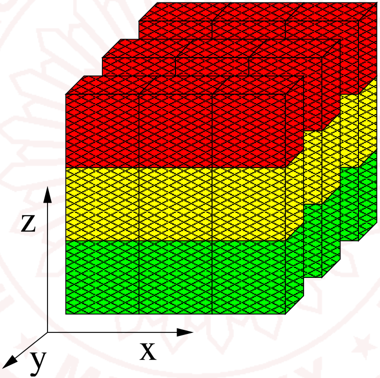
Wait and transform second xz plane



first plane's data could be accessed for next operation

Transformation in x Direction

wait and transform last xz plane



done! → 1 complete 1D-FFT overlaps a communication

Performance Tuning - Parameters

Tile factor

- number of z-planes to gather before MPI_Ialltoall is started
- very performance critical!
- not easily predictable

Window size

- number of outstanding communications
- not implemented yet
- not very performance critical → fine-tuning

MPI_Test interval

- progresses internal state and outstanding operations
- unneccessary in threaded NBC implementation (future)

Performance Tuning - Parameters

Tile factor

- number of z-planes to gather before MPI_Ialltoall is started
- very performance critical!
- not easily predictable

Window size

- number of outstanding communications
- not implemented yet
- not very performance critical → fine-tuning

MPI_Test interval

- progresses internal state and outstanding operations
- unneccessary in threaded NBC implementation (future)

Performance Tuning - Parameters

Tile factor

- number of z-planes to gather before MPI_Ialltoall is started
- very performance critical!
- not easily predictable

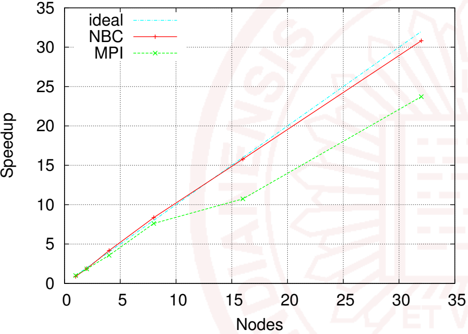
Window size

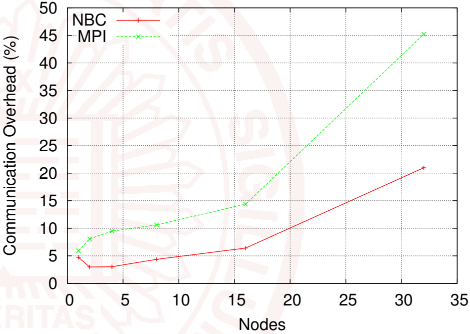
- number of outstanding communications
- not implemented yet
- not very performance critical → fine-tuning

MPI_Test interval

- progresses internal state and outstanding operations
- unneccessary in threaded NBC implementation (future)

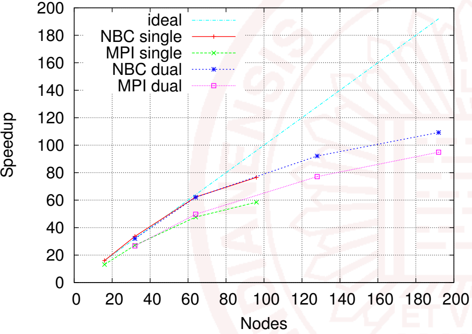
3D-FFT Benchmark Results (small input)

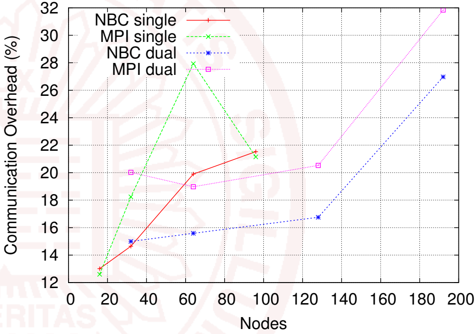




- “tantale”@CEA: 128 2 GHz quad Opteron 844 nodes
- Interconnect: InfiniBand™
- System size 128x128x128 (1 node $\approx$ 0.75 s)

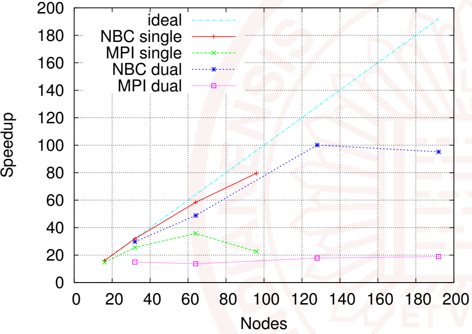
3D-FFT Benchmark Results (large input) - InfiniBand

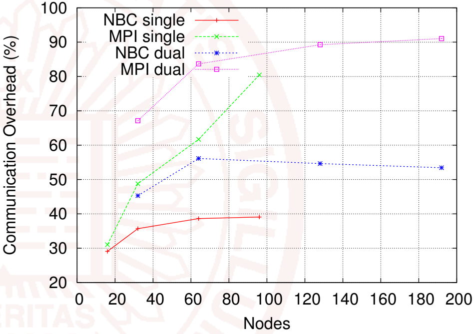




- “odin”@IU: 128 2 GHz dual Opteron 246 nodes
- Interconnect: InfiniBand™
- System size 512x512x512 (1 node ≈ 50s)

3D-FFT Benchmark Results (large input) - Ethernet





- “odin”@IU: 128 2 GHz dual Opteron 246 nodes
- Interconnect: Gigabit Ethernet
- System size 512x512x512 (1 node $\approx$ 50s)

Outline

Conclusions & Future Work

Conclusions

- applying NBC requires some effort
- NBC improves scaling
- common application patterns exist

Future Work

- tune FFT further (cache issues)
- automatic parameter assessment (?)
- parallel model for LibNBC
- LibNBC features (e.g. Fortran bindings)

Conclusions & Future Work

Conclusions

- applying NBC requires some effort
- NBC improves scaling
- common application patterns exist

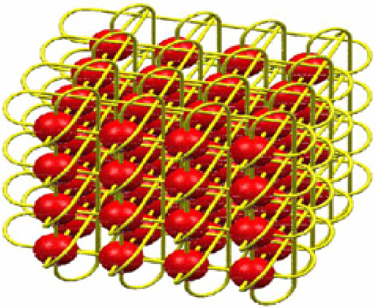
Future Work

- tune FFT further (cache issues)
- automatic parameter assessment (?)
- parallel model for LibNBC
- LibNBC features (e.g. Fortran bindings)

Discussion

THE END

try LibNBC: `http://www.unixer.de/NBC`



Thank you for your attention!