# Optimization of a parallel 3d-FFT with non-blocking collective operations

Torsten Hoefler, Gilles Zérah

Chair of Computer Architecture
Technical University of Chemnitz

Département de Physique Théorique et Appliquée
Commissariat à l'Énergie Atomique/DAM



3rd International ABINIT Developer Workshop

Liège, Belgium

29th January 2007

1. Introduction
   - Short introduction to non-blocking collectives
2. Three dimensional FFTs
   - Traditional parallel 3d-FFT
   - Parallel 3d-FFT with maximum overlap
   - Parallel cache optimized 3d-FFT with partial overlap
3. Implementation in ABINIT
   - Avoidance of the transformation of zeroes
   - Autotuning of parameters
   - Preliminary Performance Results

# Non-blocking Collective Operations

## Advantages - Overlap

- leverage hardware parallelism (e.g. InfiniBand™)
- overlap similar to non-blocking point-to-point

## Usage?

- extension to MPI-2
- ”mixture” between non-blocking ptp and collectives
- uses MPI_Requests and MPI_Test/MPI_Wait

```
MPI_Ibcast(buf1, p, MPI_INT, 0, comm, &req);
MPI_Wait(&req);
```

## Availability

Prototype LibNBC: requires ANSI-C and MPI-2

`LibNBC dowload and documentation:`
`http://www.unixer.de/NBC`

## Documentation

T. HOEFLER, J. SQUYRES, W. REHM, AND A. LUMSDAINE: *A Case for Non-Blocking Collective Operations. In Frontiers of High Performance Computing and Networking, pages 155-164, Springer Berlin, ISBN: 978-3-540-49860-5 Dec. 2006*

T. HOEFLER, J. SQUYRES, G. BOSILCA, G. FAGG, A. LUMSDAINE, AND W. REHM: *Non-Blocking Collective Operations for MPI-2. Open Systems Lab, Indiana University. presented in Bloomington, IN, USA, Aug. 2006*

# Performance Benefits?

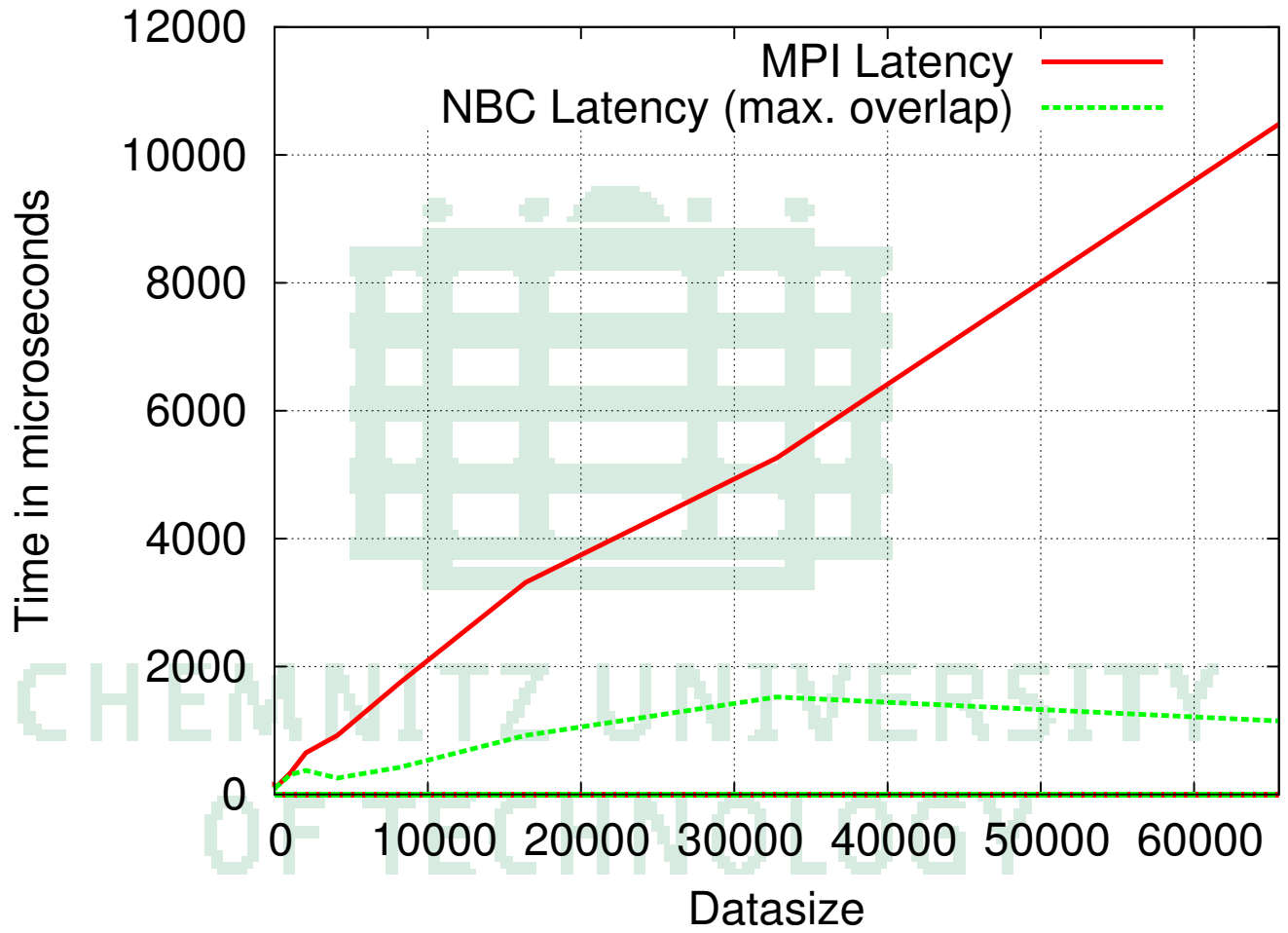
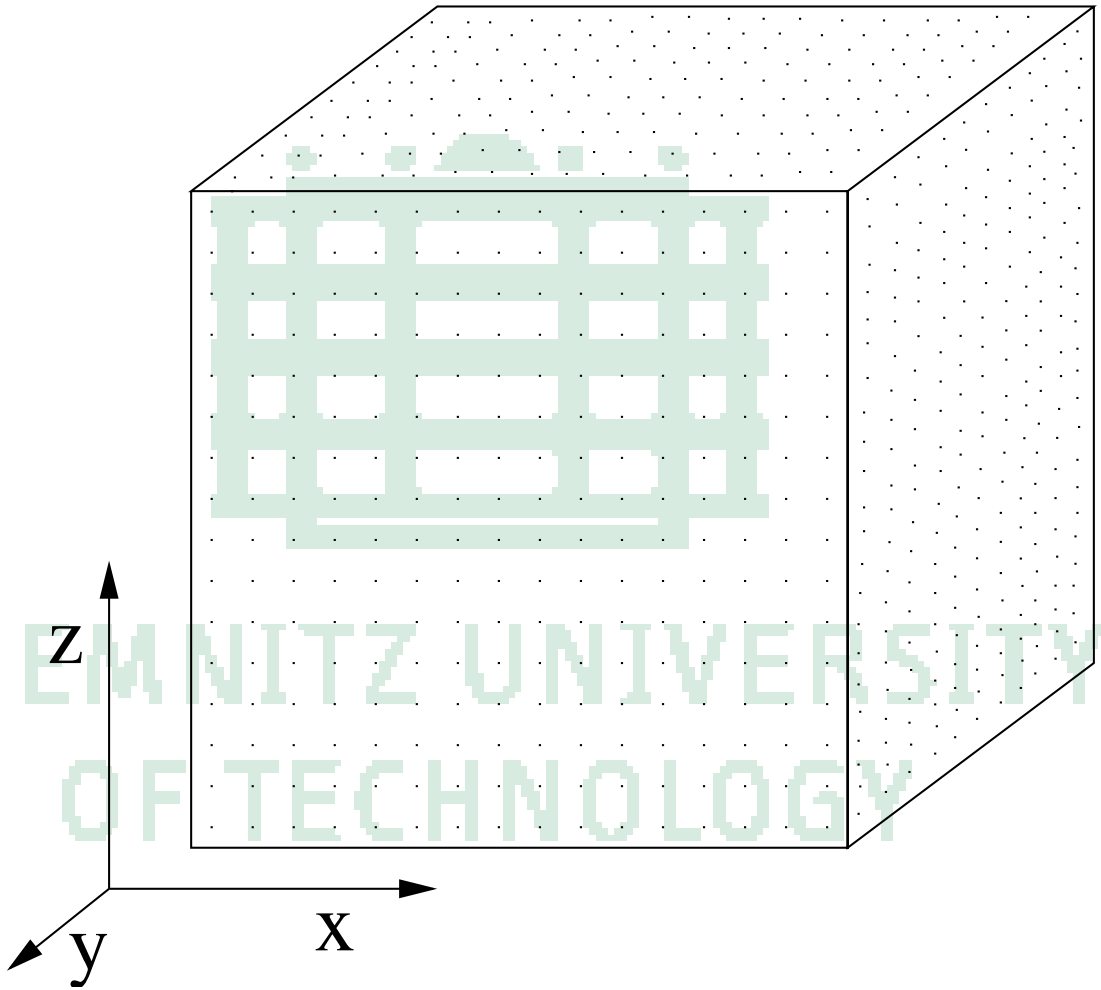



Figure: MPI_Alltoall latency on the “tantale” cluster@CEA

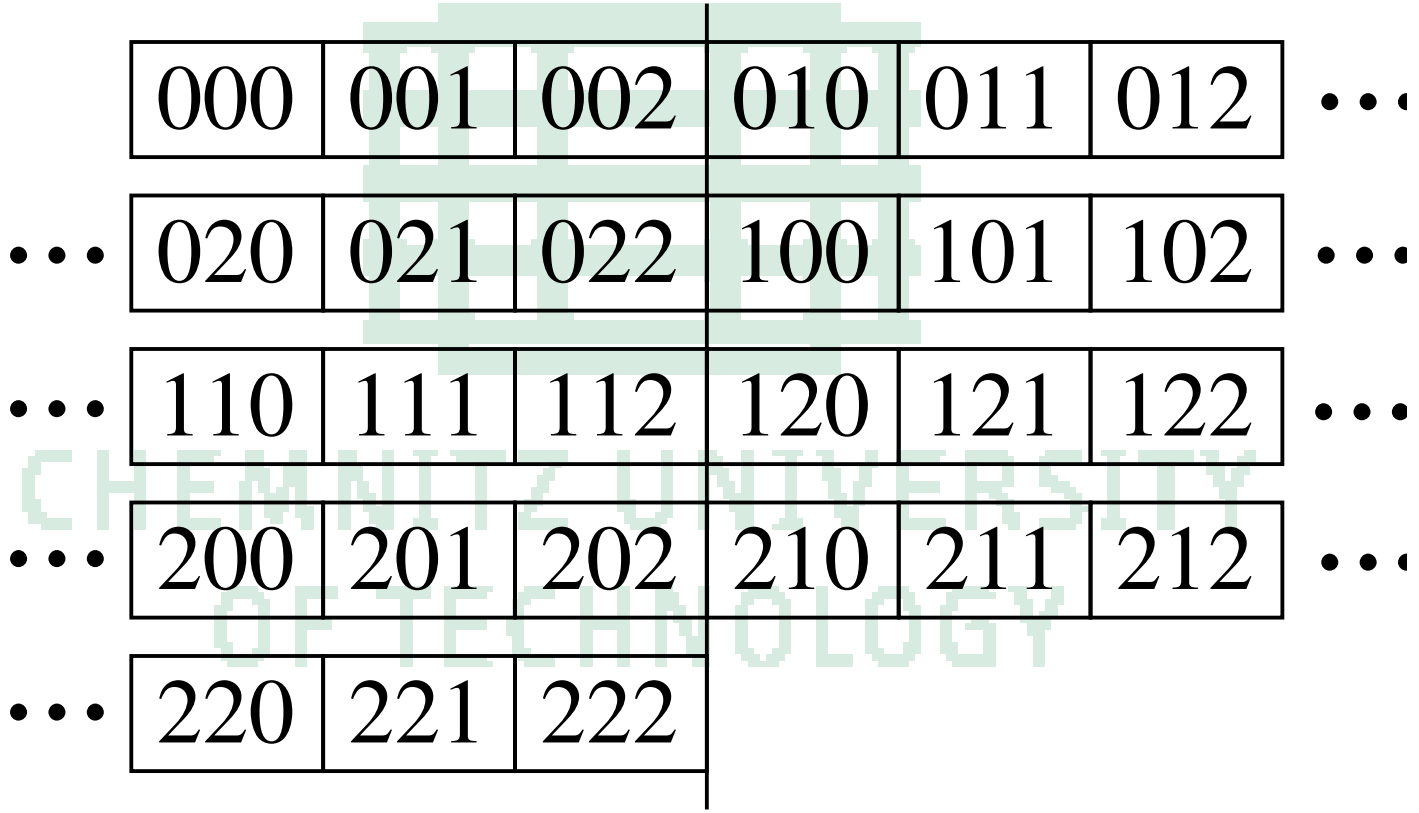
# Domain Decomposition

Discretized 3D Domain (FFT-Box)

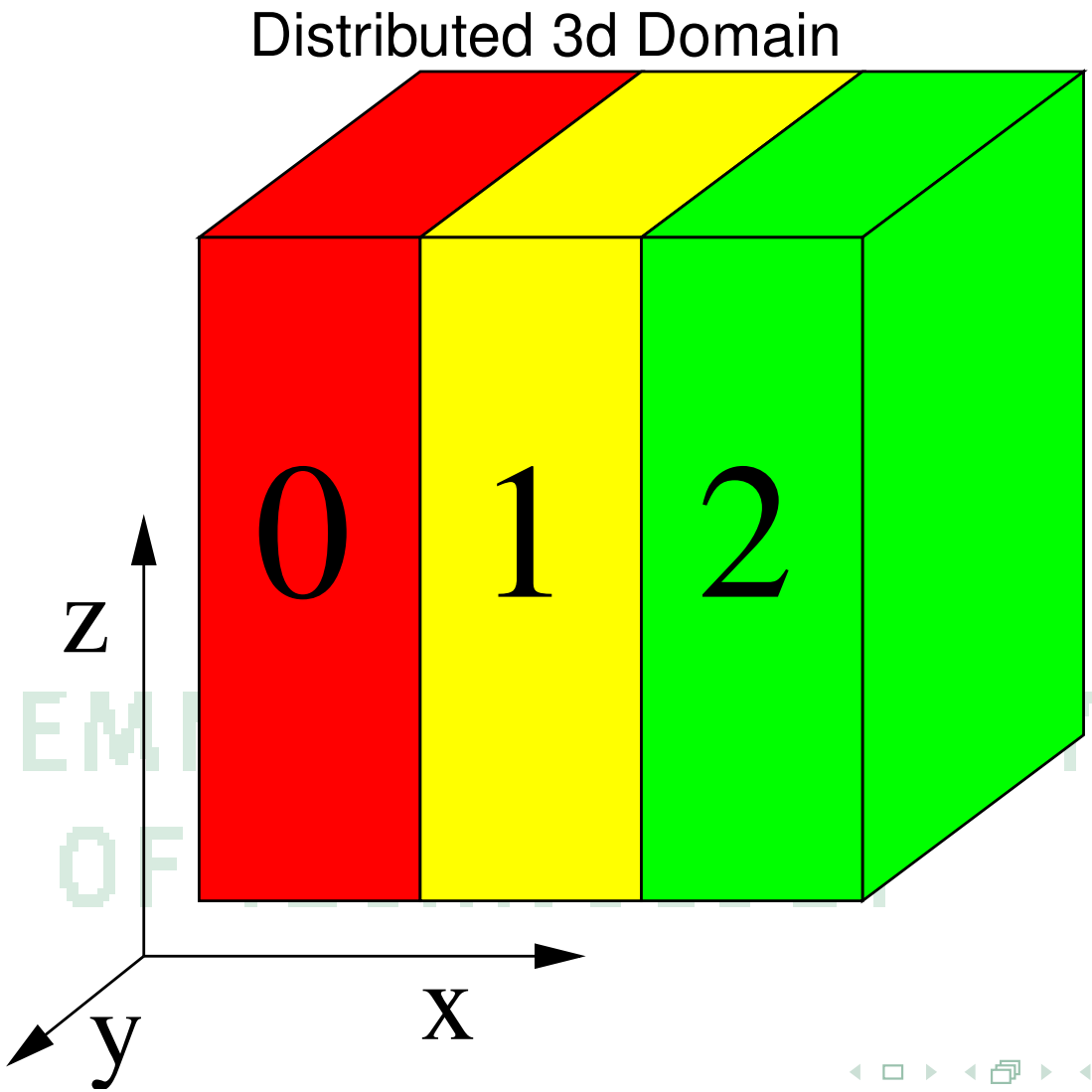


# Domain Decomposition

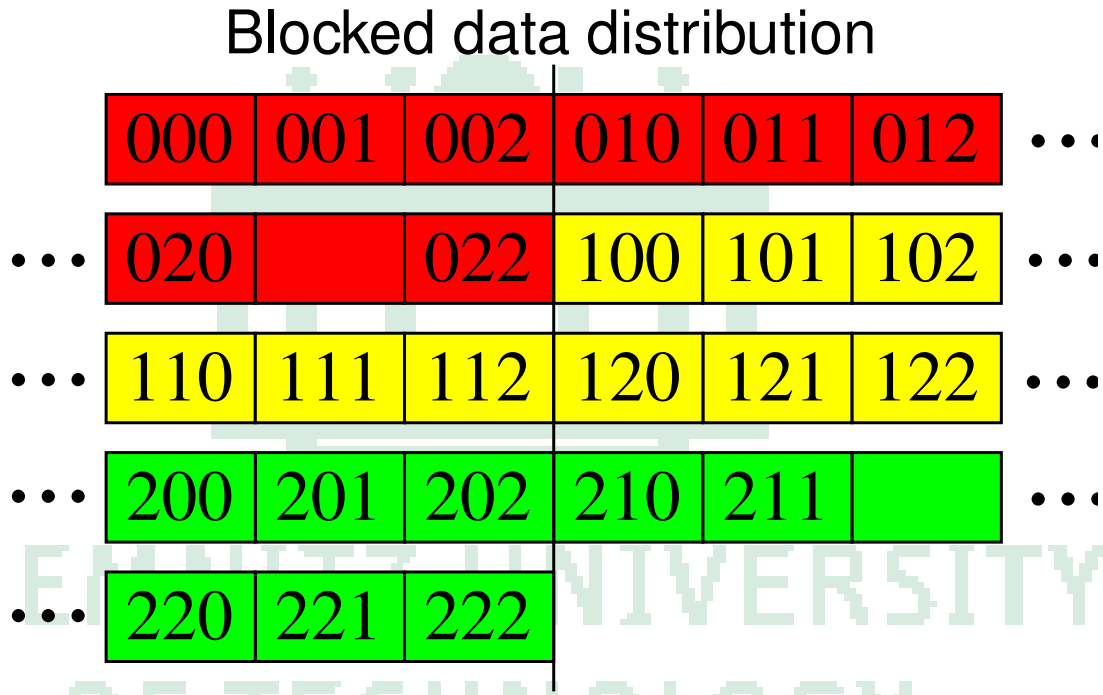
Memory layout (3x3x3 box)
(coordinates xyz: 000 $\rightarrow$ 222)
z-lines coloured!



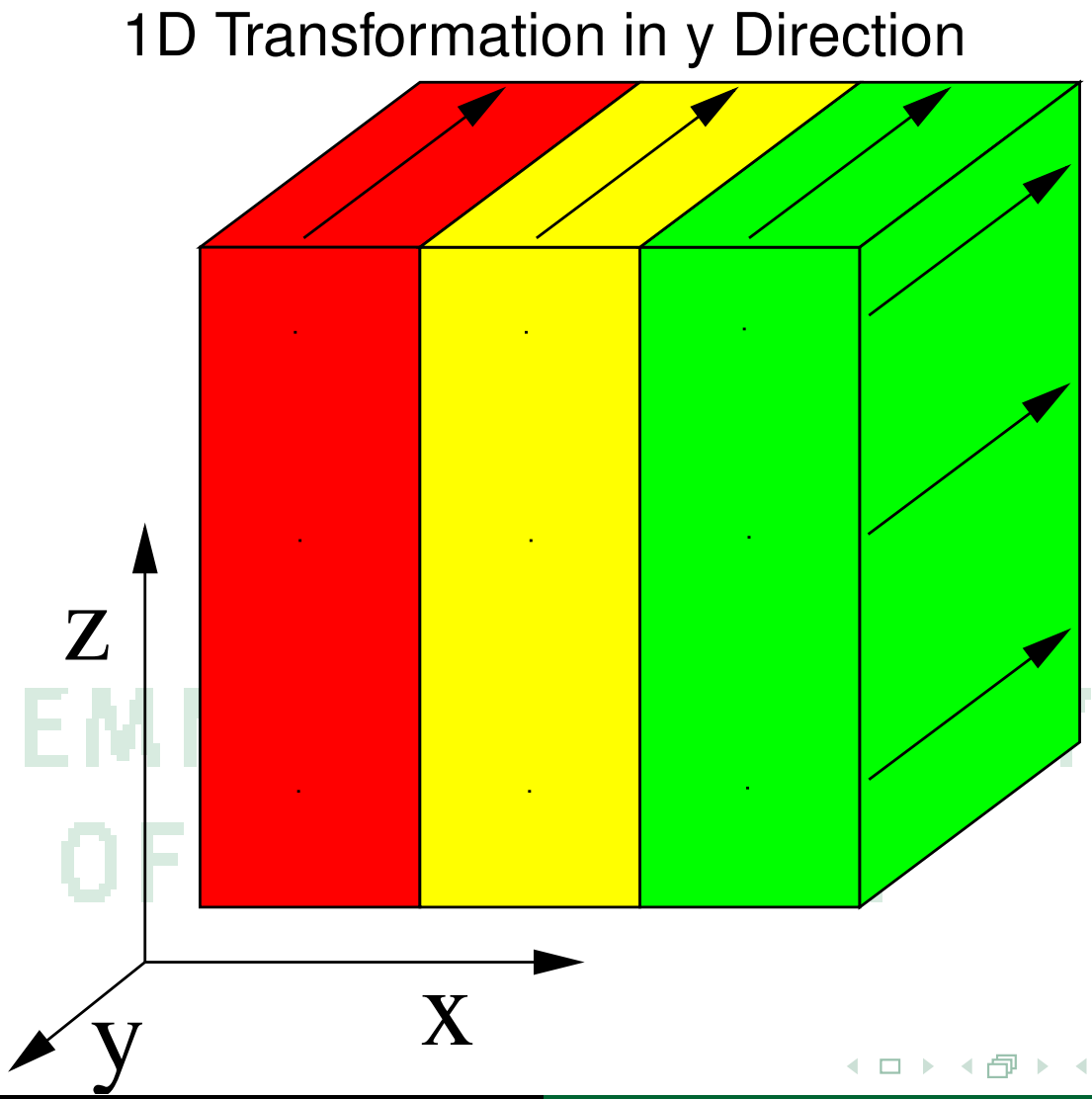
# Domain Decomposition



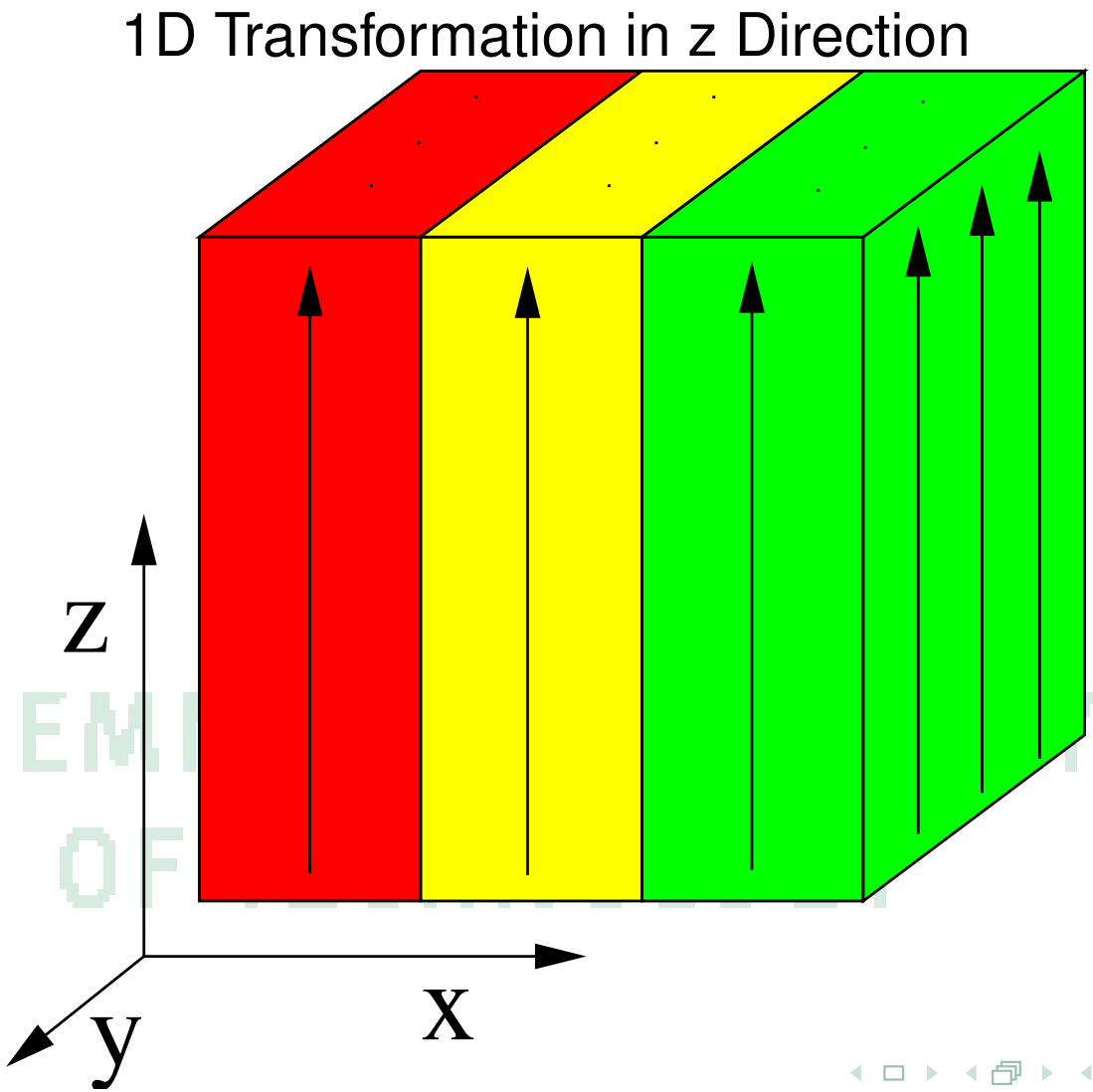
# Domain Decomposition



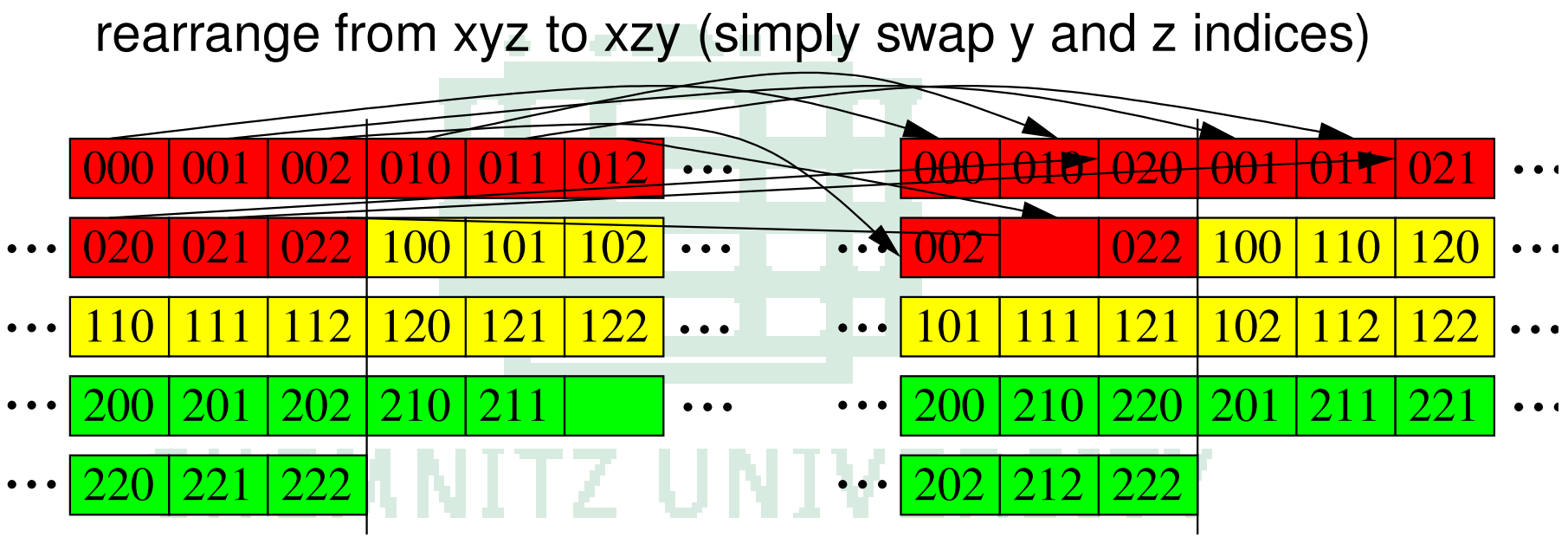
# 1D Transformation



# 1D Transformation

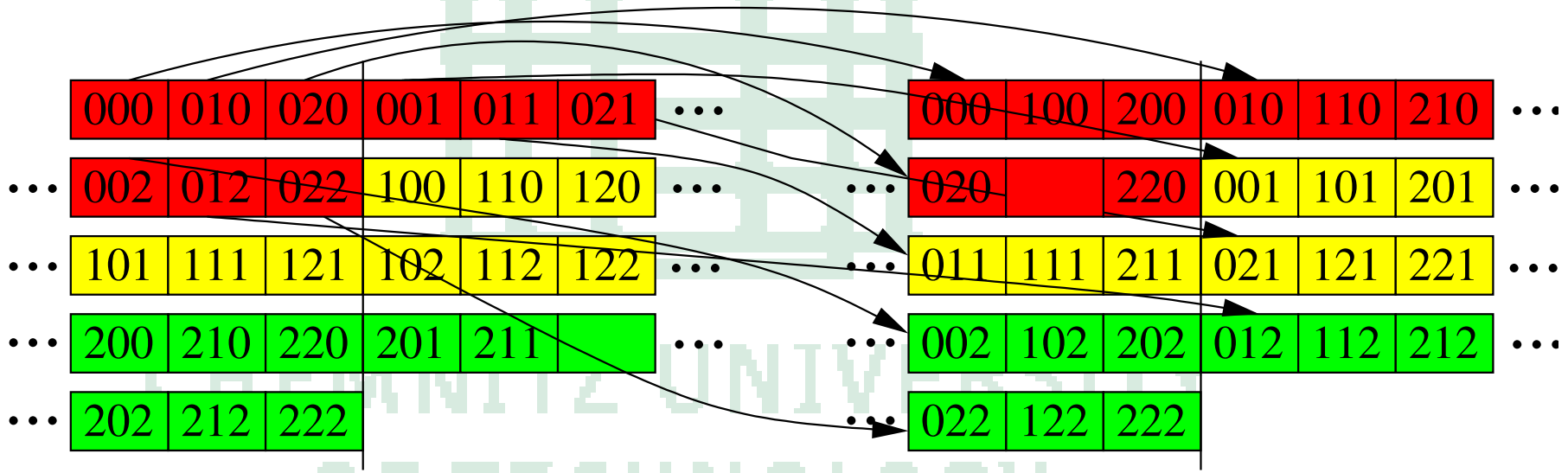


# Rearrange Data Layout

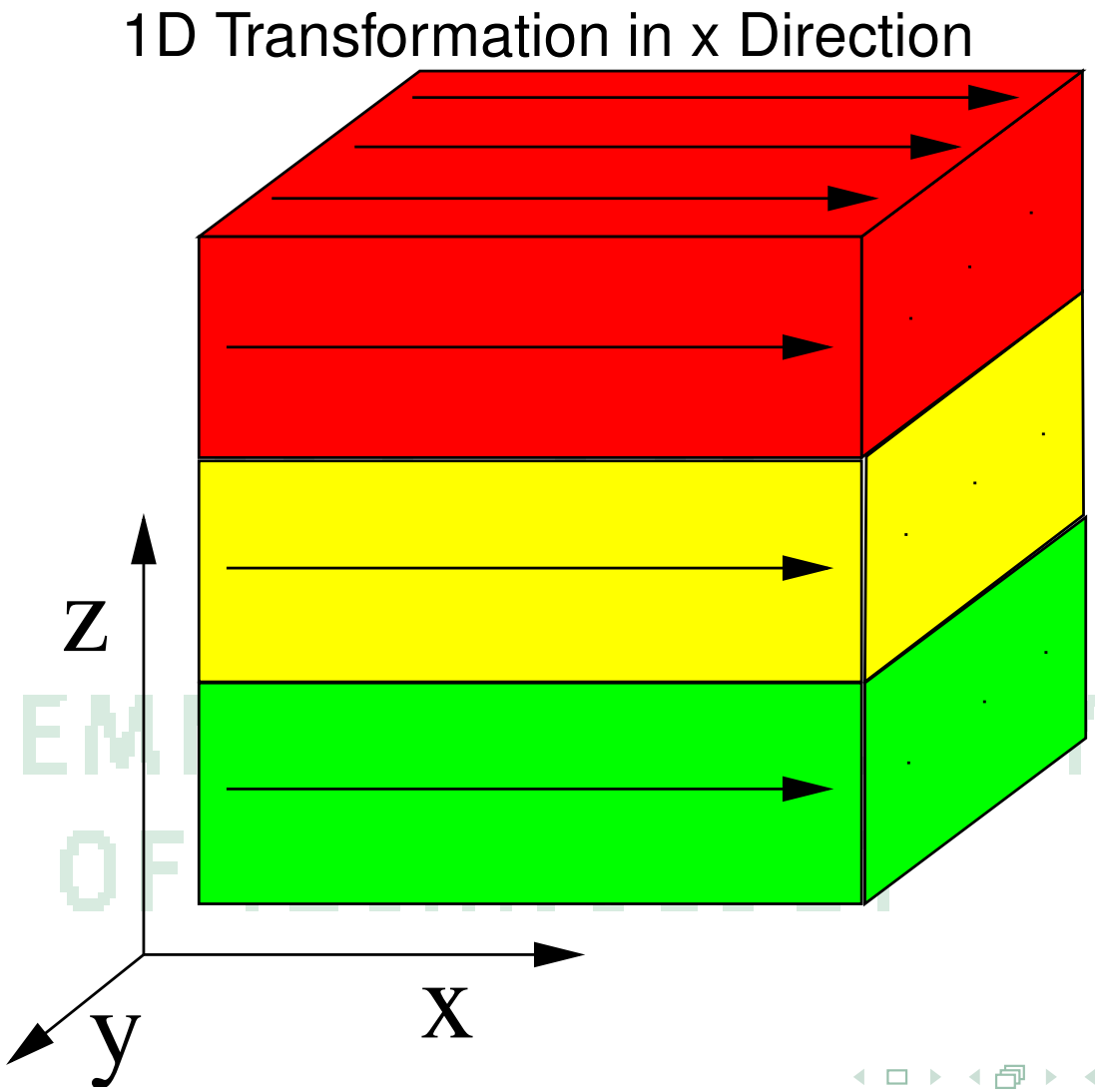


# Rearrange Data Layout

rearrange from xzy to yzx (parallel transpose)
$\Rightarrow$ MPI_Alltoall(v)



# 1D Transformation



# Non-blocking 3D-FFT

## Derivation from “normal” implementation

- distribution identical to “normal” 3D-FFT
- first FFT in y direction and local data transpose

## Design Goals to Minimize Communication Overhead

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

## Solution

- start NBC_Ialltoall as soon as first xz-plane is ready
- calculate next xz-plane
- start next communication accordingly ...
- collect multiple xz-planes (A2A data size)

# Non-blocking 3D-FFT

## Derivation from “normal” implementation

- distribution identical to “normal” 3D-FFT
- first FFT in y direction and local data transpose

## Design Goals to Minimize Communication Overhead

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

## Solution

- start NBC_Ialltoall as soon as first xz-plane is ready
- calculate next xz-plane
- start next communication accordingly ...
- collect multiple xz-planes (A2A data size)

# Non-blocking 3D-FFT

## Derivation from “normal” implementation

- distribution identical to “normal” 3D-FFT
- first FFT in y direction and local data transpose

## Design Goals to Minimize Communication Overhead

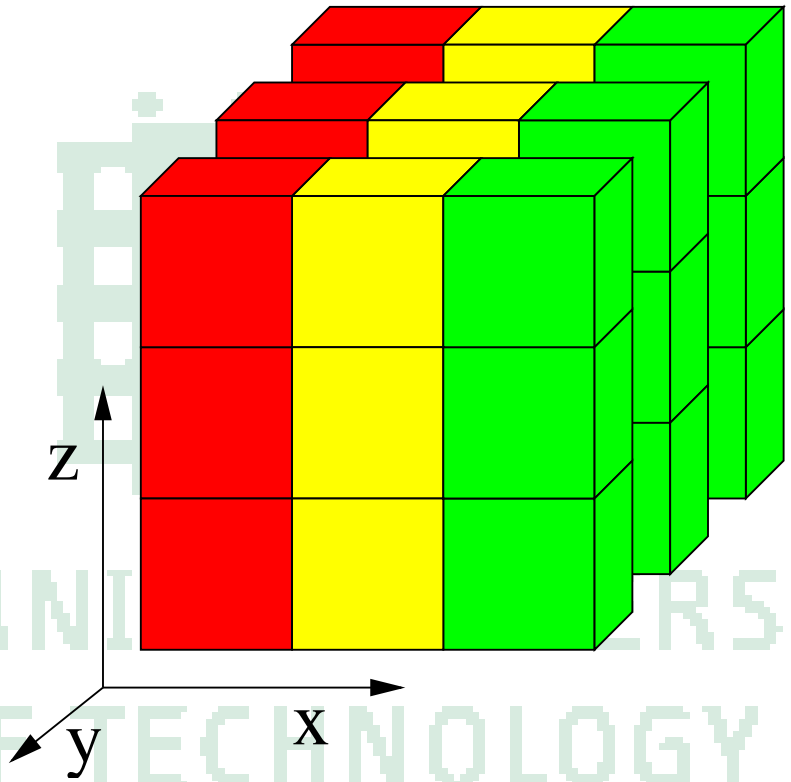
- start communication as early as possible
- achieve maximum overlap time

## Solution

- start NBC_Ialltoall as soon as first xz-plane is ready
- calculate next xz-plane
- start next communication accordingly ...
- collect multiple xz-planes (A2A data size)

# Transformation of in z Direction

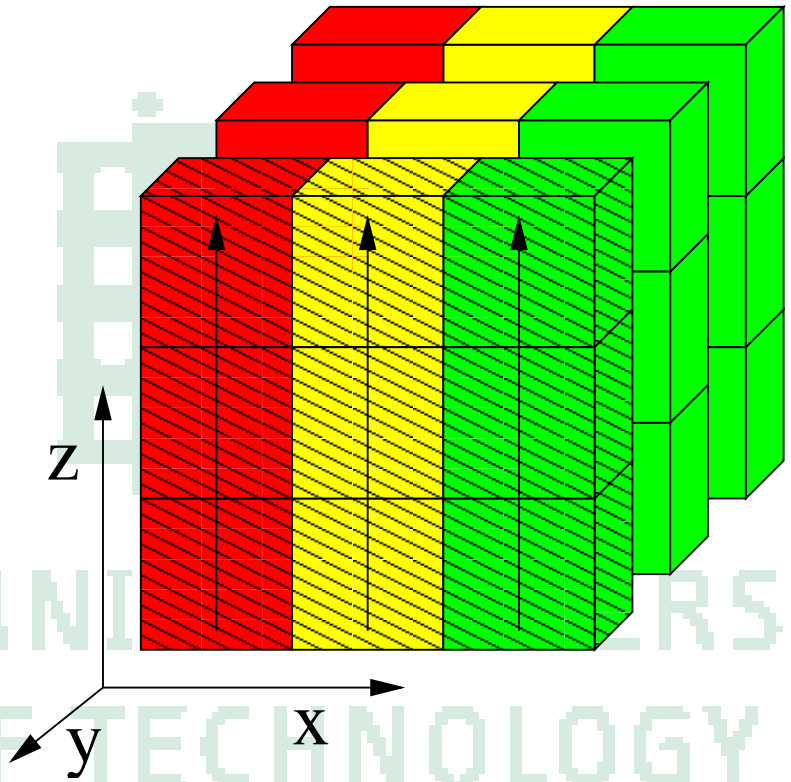
Data already transformed in y direction




1 block = 1 double value (3x3x3 grid)

# Transformation of in z Direction

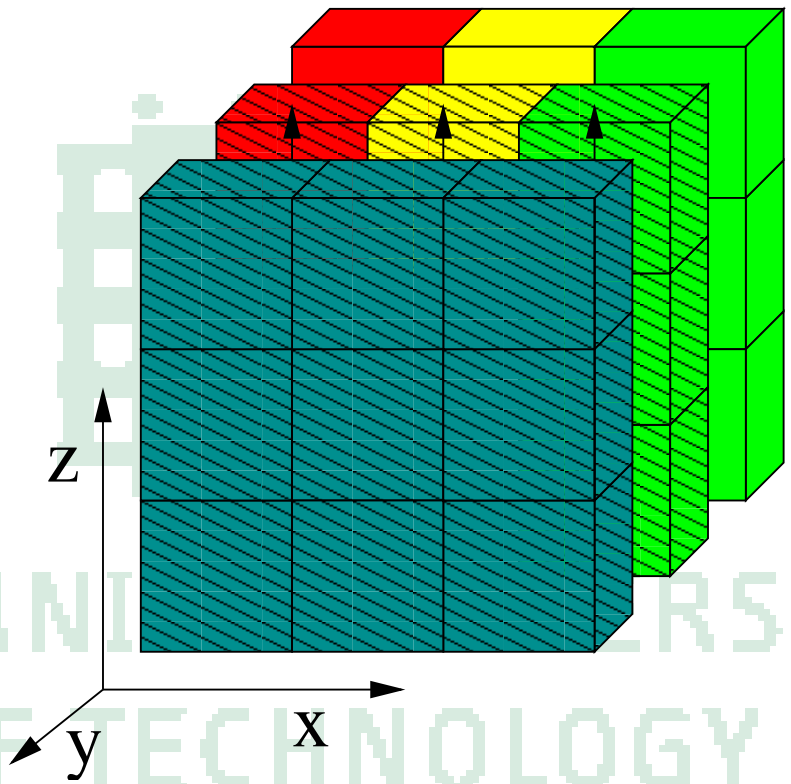
Transform first xz plane in z direction in parallel




pattern means that data was transformed in y and z direction

# Transformation of in z Direction

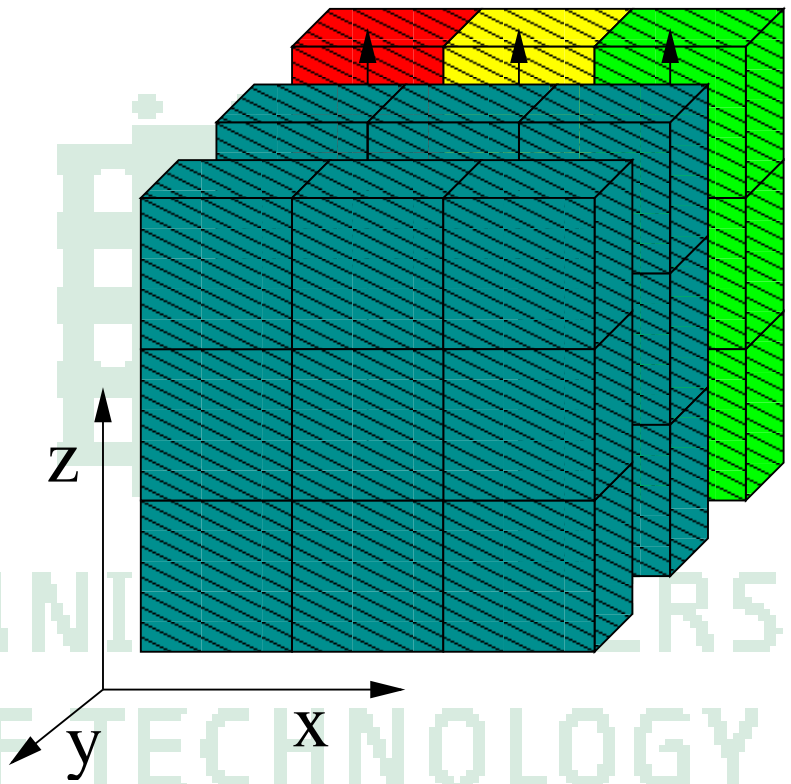
start NBC_Ialltoall of first xz plane and transform second plane




cyan color means that data is communicated in the background

# Transformation of in z Direction

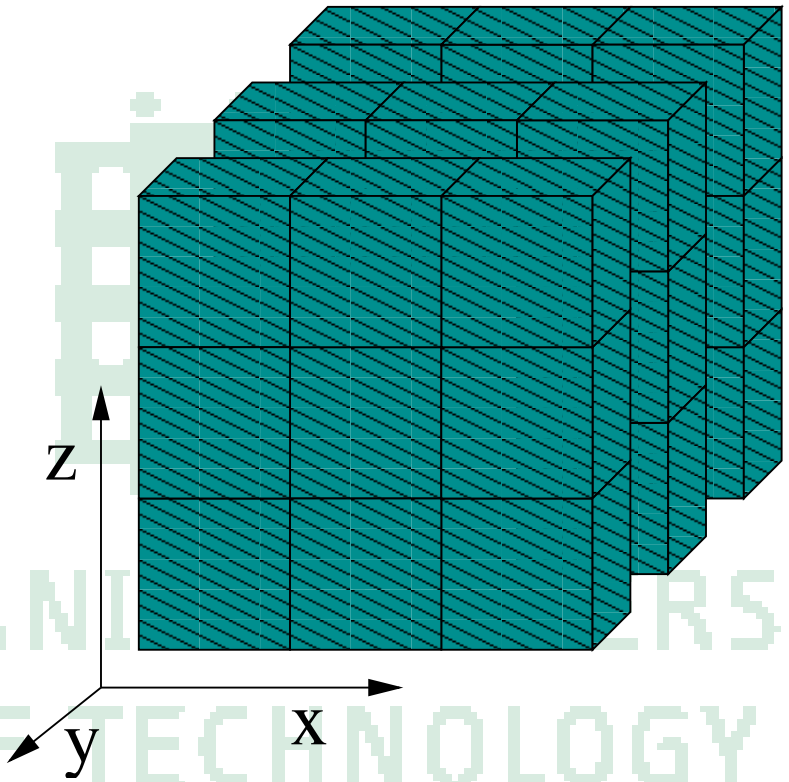
start NBC_Ialltoall of second xz plane and transform third plane




data of two planes is not accessible due to communication

# Transformation of in z Direction

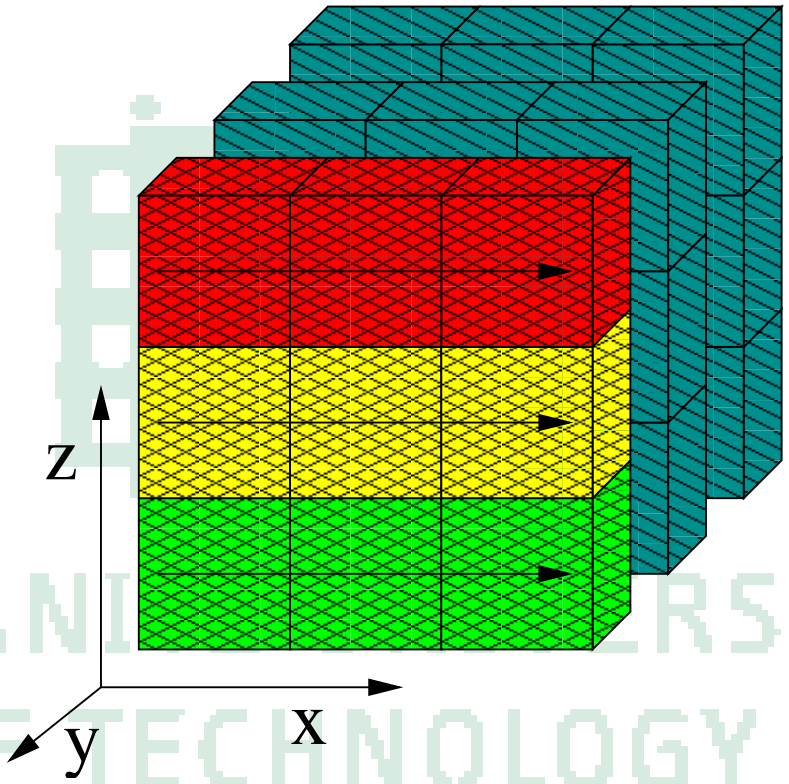
start communication of the third plane and ...




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

# Transformation of in x Direction

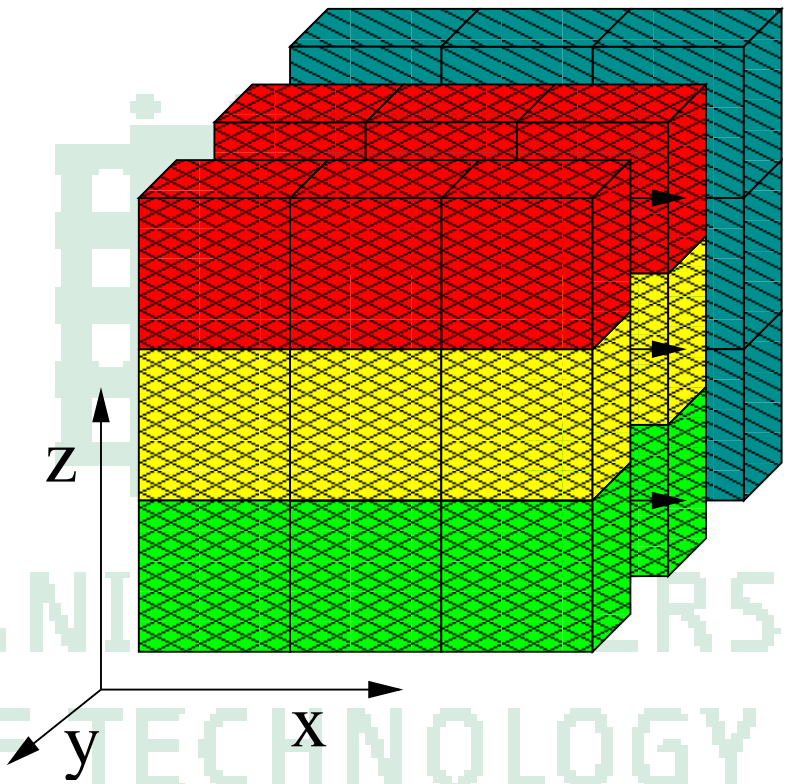
... so NBC_Wait for the first NBC_Ialltoall!




and transform first plane (pattern means xyz transformed)

# Transformation of in x Direction

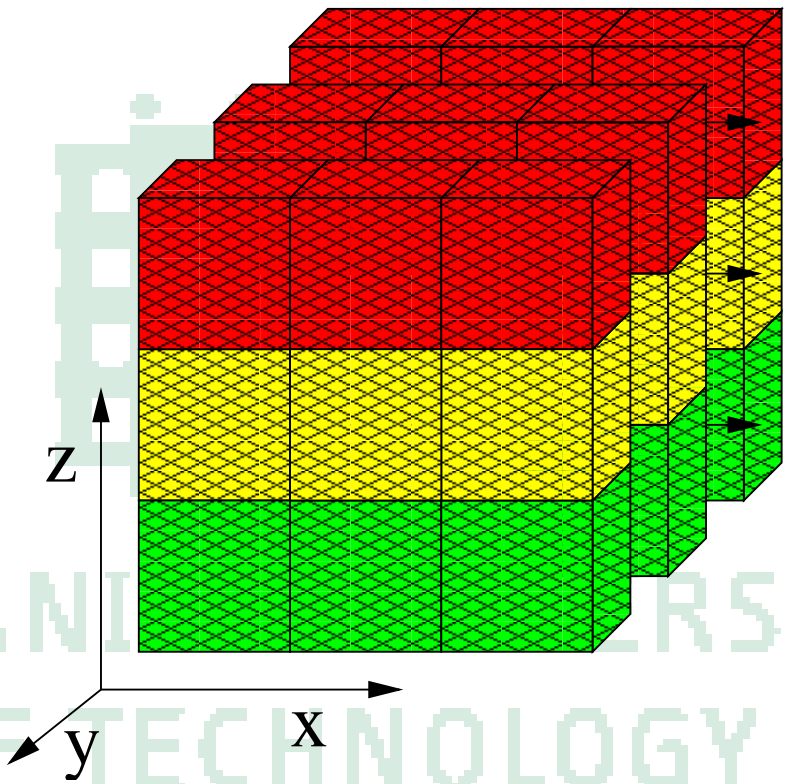
Wait and transform second xz plane




first plane's data could be accessed

# Transformation of in x Direction

wait and transform last xz plane




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

# Parameter and Problems

## Tile factor

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

## Window size and MPI_Test interval

- Window size = number of outstanding communications
- not very performance critical → fine-tuning
- MPI_Test progresses internal state of MPI
- unneccessary in threaded NBC implementation (future)

## Problems?

- **NOT** cache friendly :-(

# Parameter and Problems

## Tile factor

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

## Window size and MPI_Test interval

- Window size = number of outstanding communications
- not very performance critical → fine-tuning
- MPI_Test progresses internal state of MPI
- unneccessary in threaded NBC implementation (future)

## Problems?

- **NOT** cache friendly :-(

# Parameter and Problems

## Tile factor

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

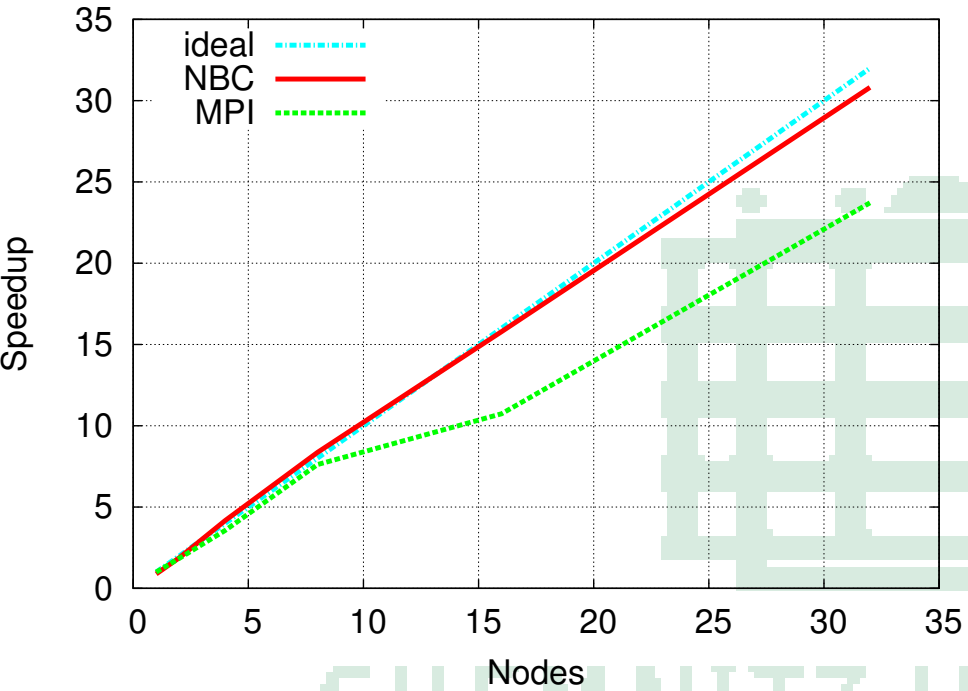
## Window size and MPI_Test interval

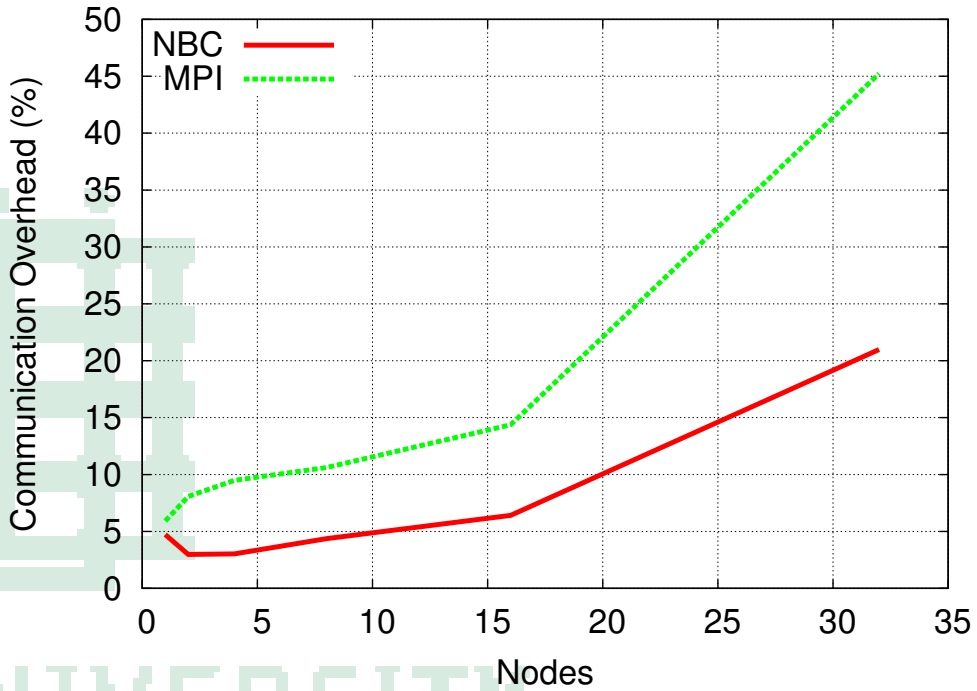
- Window size = number of outstanding communications
- not very performance critical → fine-tuning
- MPI_Test progresses internal state of MPI
- unneccessary in threaded NBC implementation (future)

## Problems?

- **NOT** cache friendly :-(

# 3D-FFT Benchmark Results (small input)

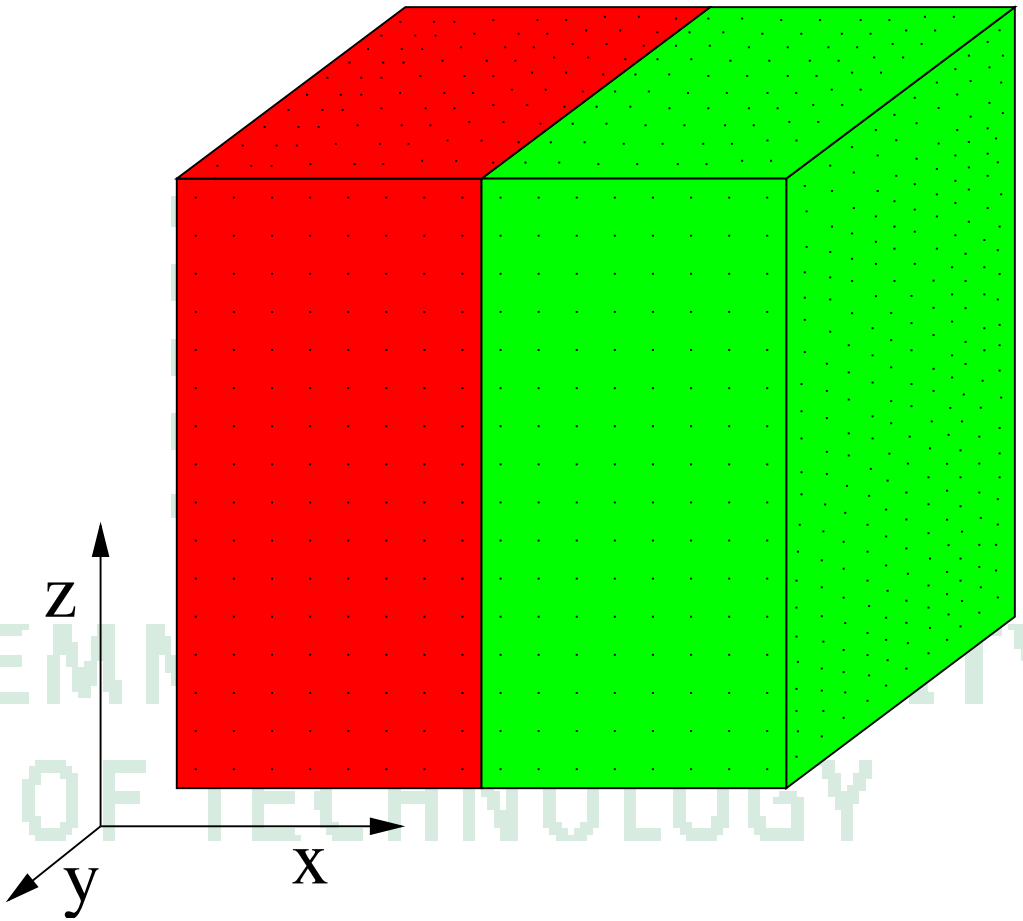



- “tantale”@CEA: 128 2 GHz Quad Opteron 844 nodes
- Interconnect: InfiniBand™
- System size 128x128x128 (1 CPU ≈ 0.75 s)

# Cache optimal implementation

cache optimality by yz transforming plane by plane (in cache)!






→ we need all yz-planes before we can start x transform :-(

# Applying Non-blocking collectives

## Pipelined communication

- retain plane-by-plane transform
- simple pipelined scheme
- start A2A of plane as soon as it is transformed
- wait for all before x transform
- A2A overlapped with computation of remaining planes
- last A2A blocks (immediate wait :-( )

## Issues

- less overap potential
- plane coalescing to adjust datasize
- new parameter: “pipeline depth” (# of A2As)

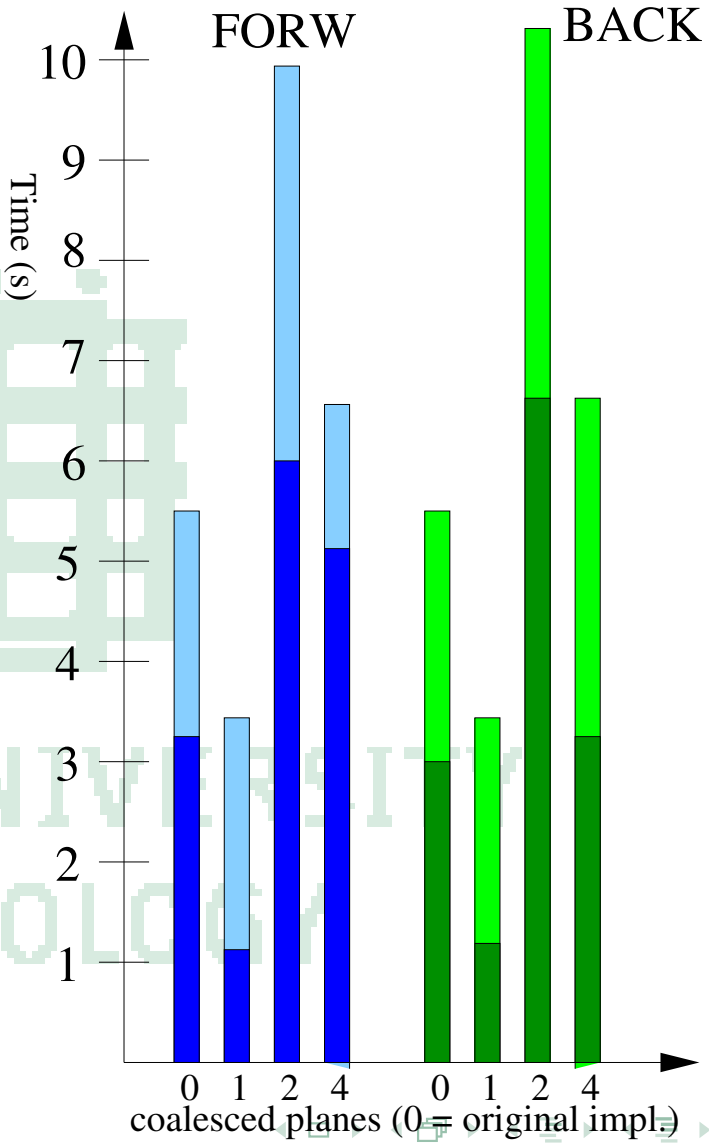
# 3D-FFT Benchmark Results (small input)

## System

- “tantale”@CEA
- 2 GHz Quad Opteron
- InfiniBand™

## Parameters

- 128x128x128
- 16 CPUs, 4 nodes
- 1 CPU ≈ 28 s
- 8 planes/proc
- 16kb/plane



1. Introduction
   - Short introduction to non-blocking collectives
2. Three dimensional FFTs
   - Traditional parallel 3d-FFT
   - Parallel 3d-FFT with maximum overlap
   - Parallel cache optimized 3d-FFT with partial overlap
3. Implementation in ABINIT
   - Avoidance of the transformation of zeroes
   - Autotuning of parameters
   - Preliminary Performance Results

# Avoidance of the transformation of zeroes

## ABINIT Implementation

- changed routines `back`, `forw`, `back_wf` and `forw_wf`
- some minor changes to others (input params ...)

## The routines `back_wf` and `forw_wf`

- avoid transformation of zeroes
- less computation and less communication
- changed communication (boxcut=2):
  - forw_wf: $nz/p$ planes, each has $nx/2 \cdot ny/(2 \cdot p)$ doubles
  - back_wf: $nz/(2 \cdot p)$ planes, each has $nx/2 \cdot ny/p$ doubles

## New Parameters

- all routines have different # planes $\rightarrow$ three parameters

1. Introduction
   - Short introduction to non-blocking collectives
2. Three dimensional FFTs
   - Traditional parallel 3d-FFT
   - Parallel 3d-FFT with maximum overlap
   - Parallel cache optimized 3d-FFT with partial overlap
3. Implementation in ABINIT
   - Avoidance of the transformation of zeroes
   - Autotuning of parameters
   - Preliminary Performance Results

# Autotuning of parameters

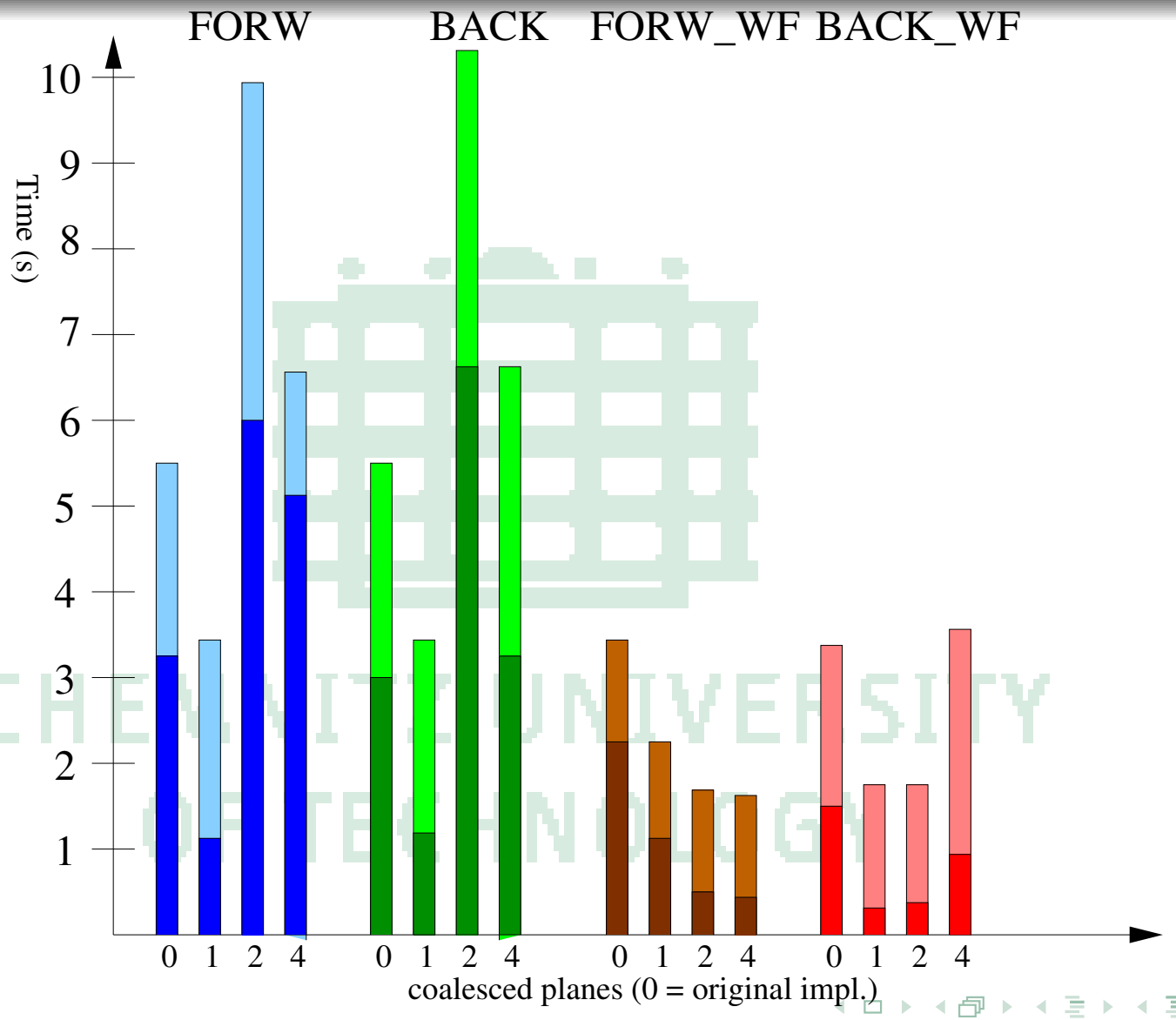
## Three new input parameters

- `fftplanes_fourdp`, `fftplanes_forw_wf`, and `fftplanes_back_wf`
- default = 0 $\rightarrow$ standard MPI implementation
- performance critical
- complicated to determine by hand

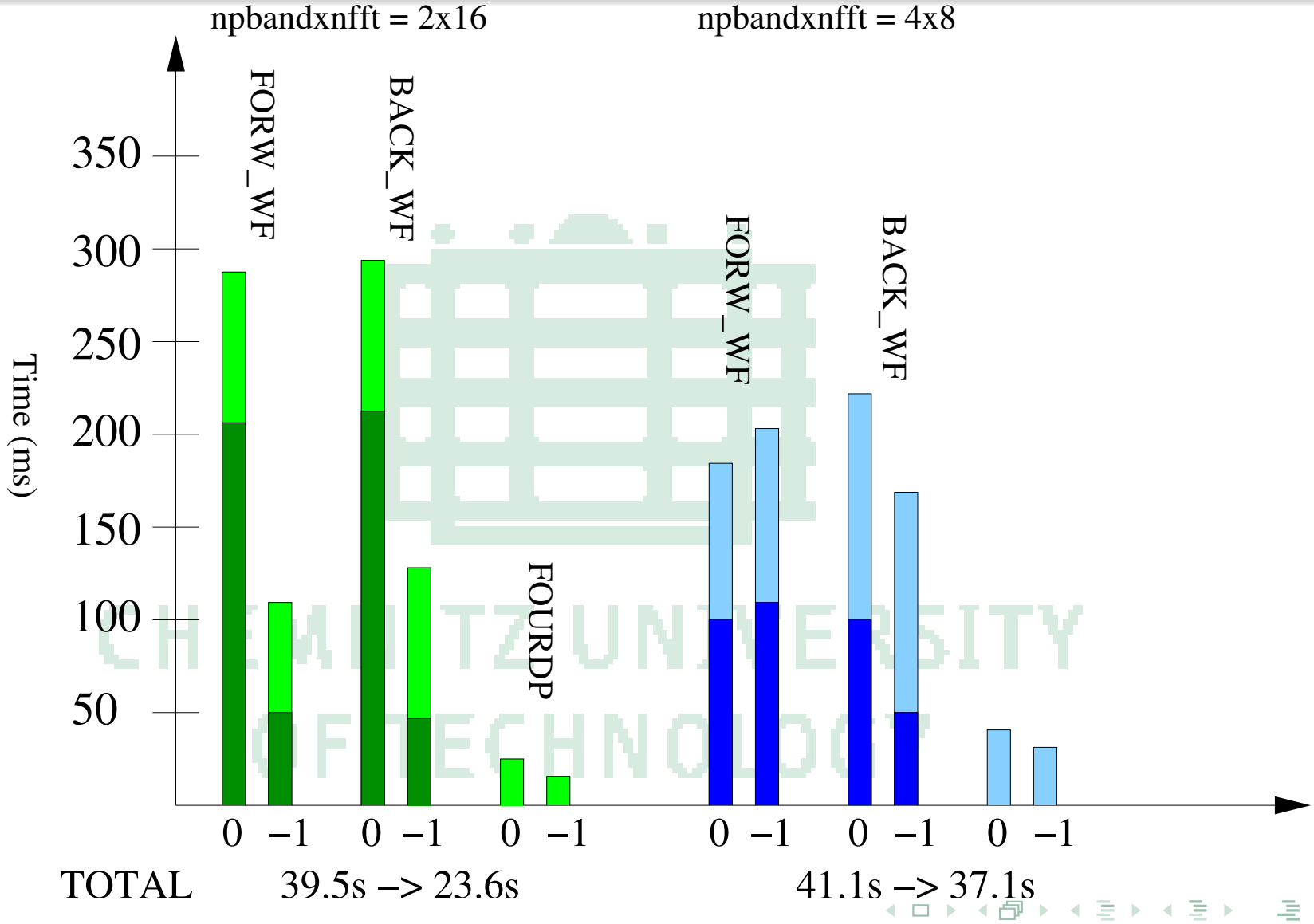
## Autotuning

- automatically determine them at runtime
- each planes parameter is benchmarked (after warmup round)
- fastest is chosen automatically
- relatively accurate but problems with jitter

# Microbenchmarks



# ABINIT - Si, 60 bands, $128^3$ FFT



# Conclusions & Future Work

## Conclusions

- applying NBC requires some effort
- NBC can improve parallel efficience
- cache usage vs. overlap potential

## Future Work

- tune FFT further (reduce serial overhead)
- better automatic parameter assessment (?)
- parallel model for 3d-FFT
- use NBC for parallel orthogonalization
- apply NBC at higher level (LOBPCG?)

# Conclusions & Future Work

## Conclusions

- applying NBC requires some effort
- NBC can improve parallel efficience
- cache usage vs. overlap potential

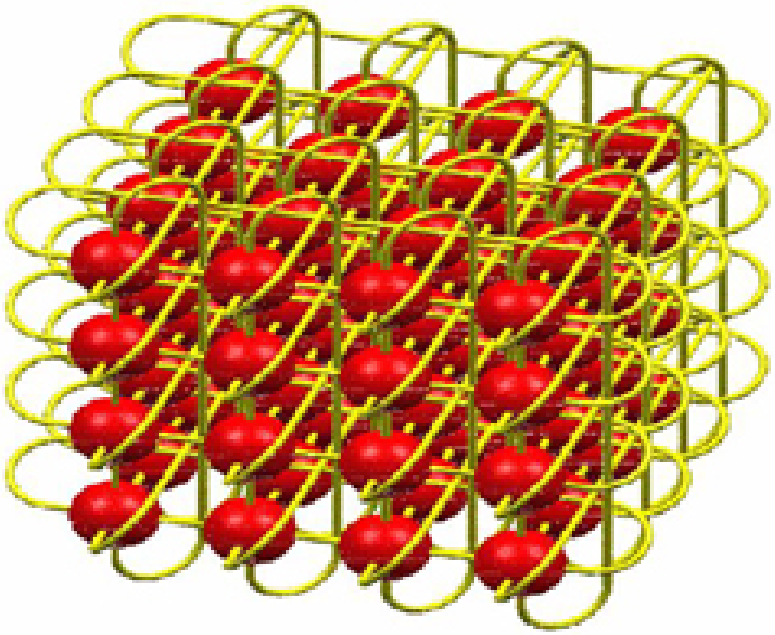
## Future Work

- tune FFT further (reduce serial overhead)
- better automatic parameter assessment (?)
- parallel model for 3d-FFT
- use NBC for parallel orthogonalization
- apply NBC at higher level (LOBPCG?)

# Discussion

ABINIT patch (soon):

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



Thanks to the CEA/DAM for support of this work and you for your attention!