Performance Optimization of CPMD

Tobias Klöffel[1,2], Gerald Mathias[3], Bernd Meyer[1]

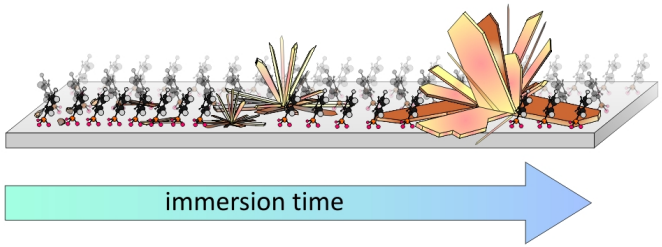
QE Dev-Meeting 2020, Trieste, January 16th 2020

[1]Interdisciplinary Center for Molecular Materials (ICMM)
Computer Chemistry Center (CCC)
Friedrich-Alexander-Universität Erlangen-Nürnberg

[2]High Performance Computing group
at Erlangen Regional Computing Center (RRZE)

[3]Leibniz Supercomputing Centre (LRZ), Garching

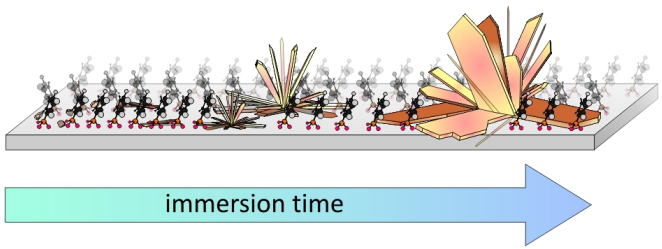
Chemical Reactions @ Solid-Liquid Interfaces



A. Ostapenko, T. Klöffel, J. Eußner, K. Harms, S. Dehnen, B. Meyer, and G. Witte,
ACS Appl. Mater. Interfaces **8**,13472 (2016).

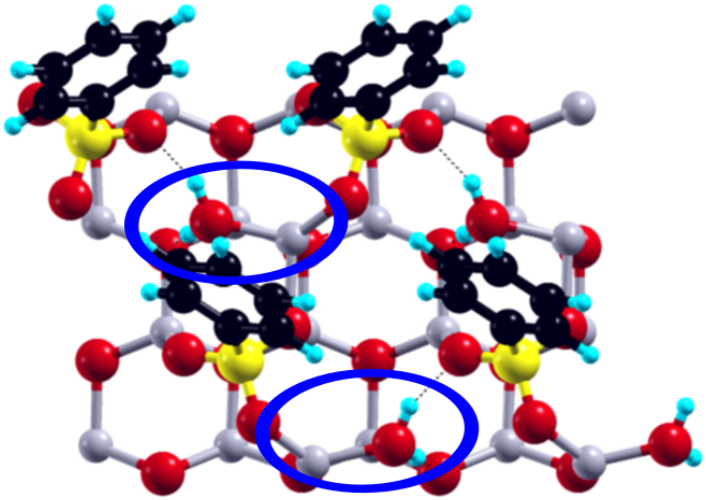
A. Ostapenko, T. Klöffel, B. Meyer, and G. Witte,
Langmuir **32**, 5029 (2016).

Chemical Reactions @ Solid-Liquid Interfaces

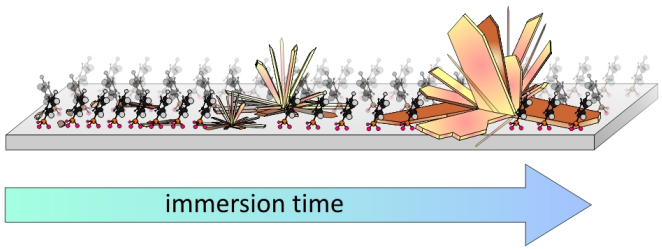


A. Ostapenko, T. Klöffel, J. Eußner, K. Harms, S. Dehnen, B. Meyer, and G. Witte,
ACS Appl. Mater. Interfaces **8**,13472 (2016).

A. Ostapenko, T. Klöffel, B. Meyer, and G. Witte,
Langmuir **32**, 5029 (2016).

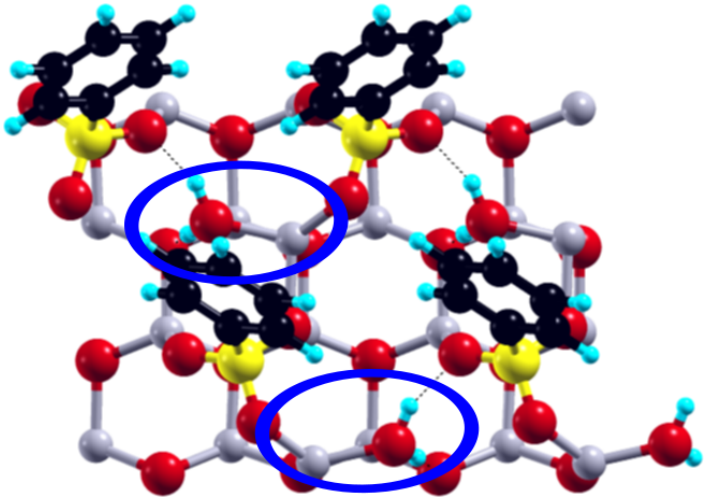


Chemical Reactions @ Solid-Liquid Interfaces

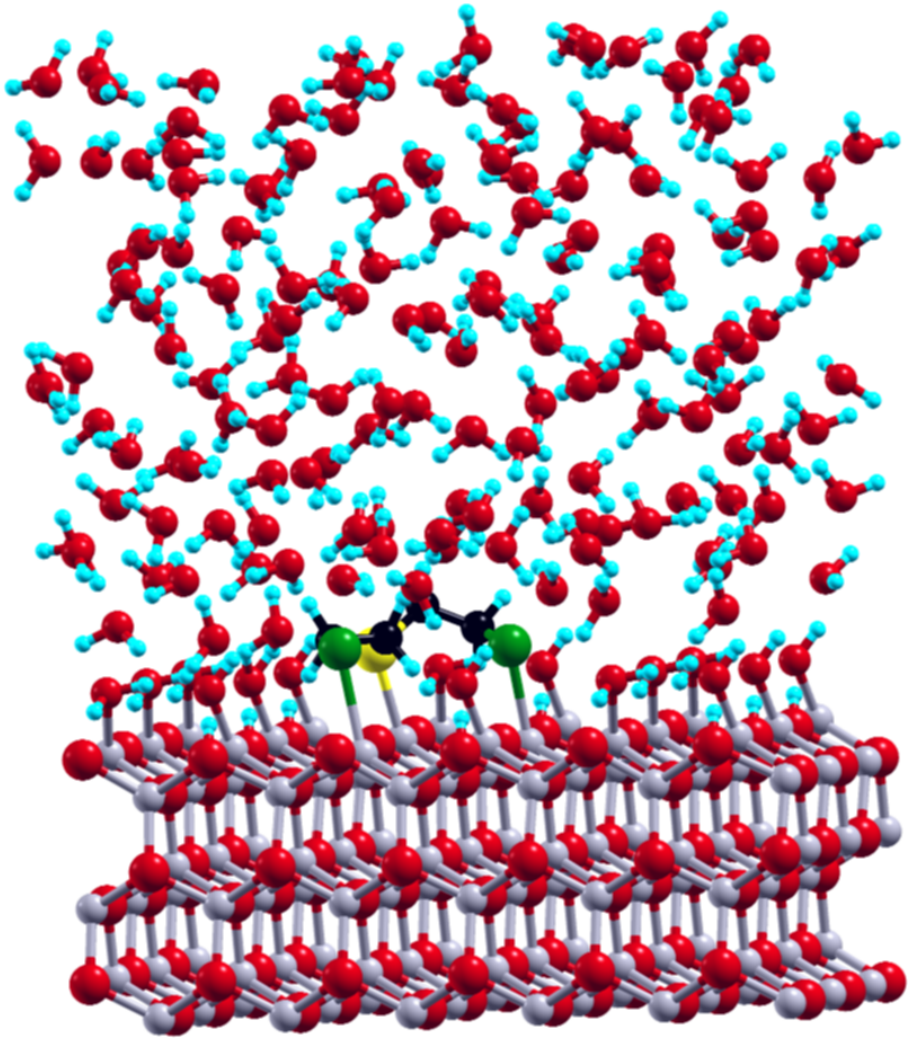


A. Ostapenko, T. Klöffel, J. Eußner, K. Harms, S. Dehnen, B. Meyer, and G. Witte,
ACS Appl. Mater. Interfaces **8**,13472 (2016).

A. Ostapenko, T. Klöffel, B. Meyer, and G. Witte,
Langmuir **32**, 5029 (2016).

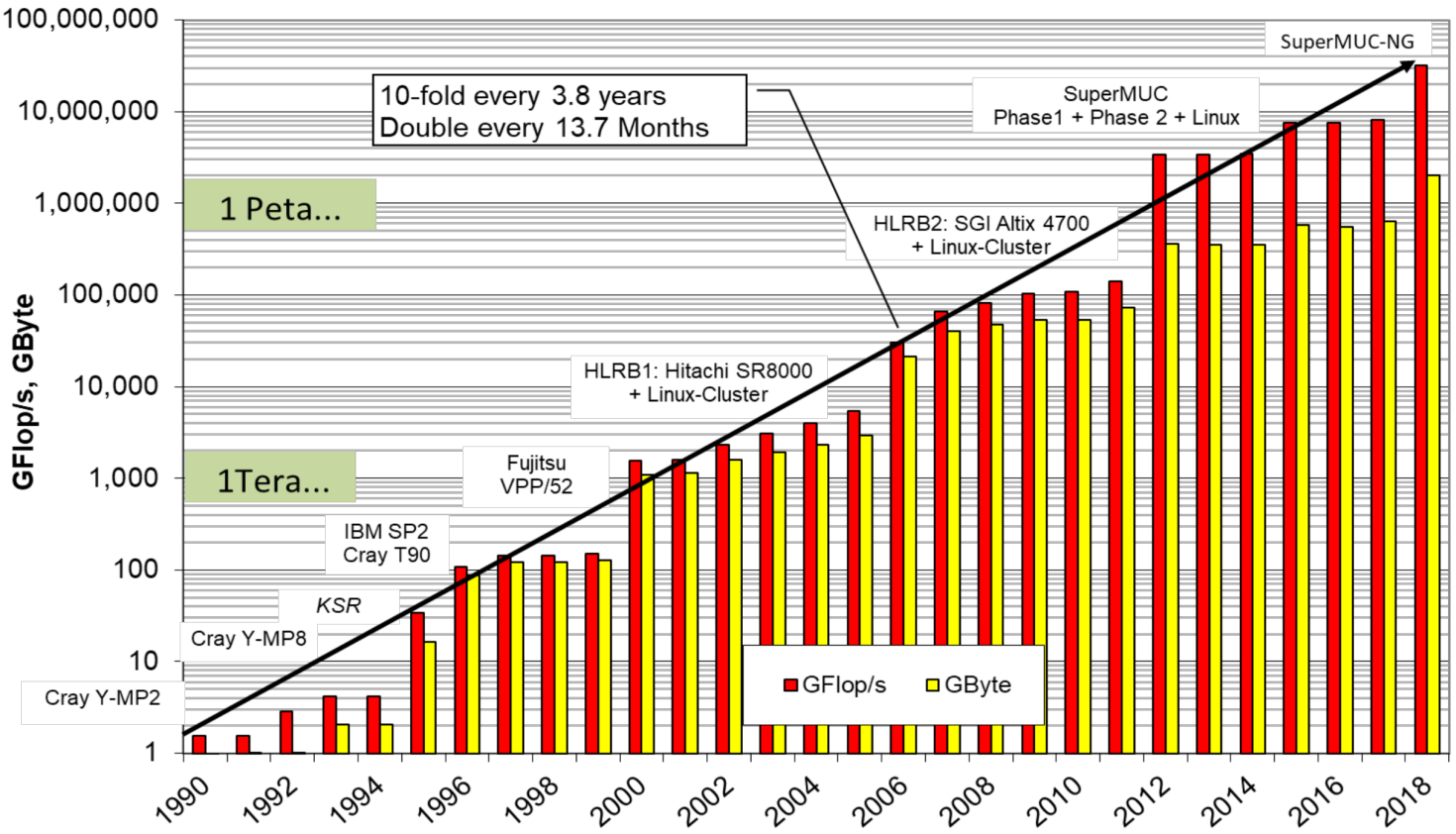


735 atoms, 3324 electrons

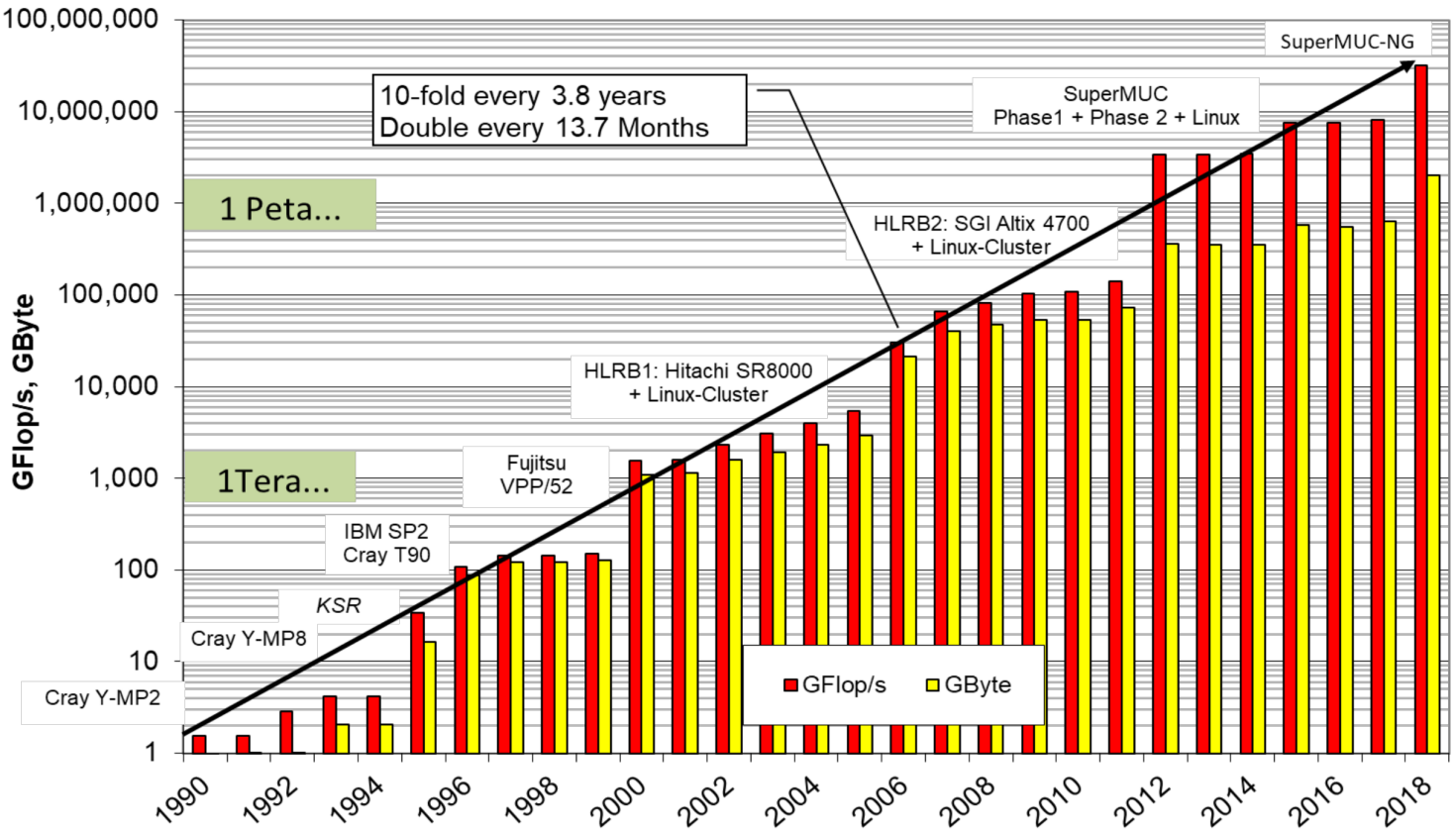


Excursion: High Performance Computing: SuperMUC-NG @ LRZ

Excursion: “Moore’s Law” @ LRZ

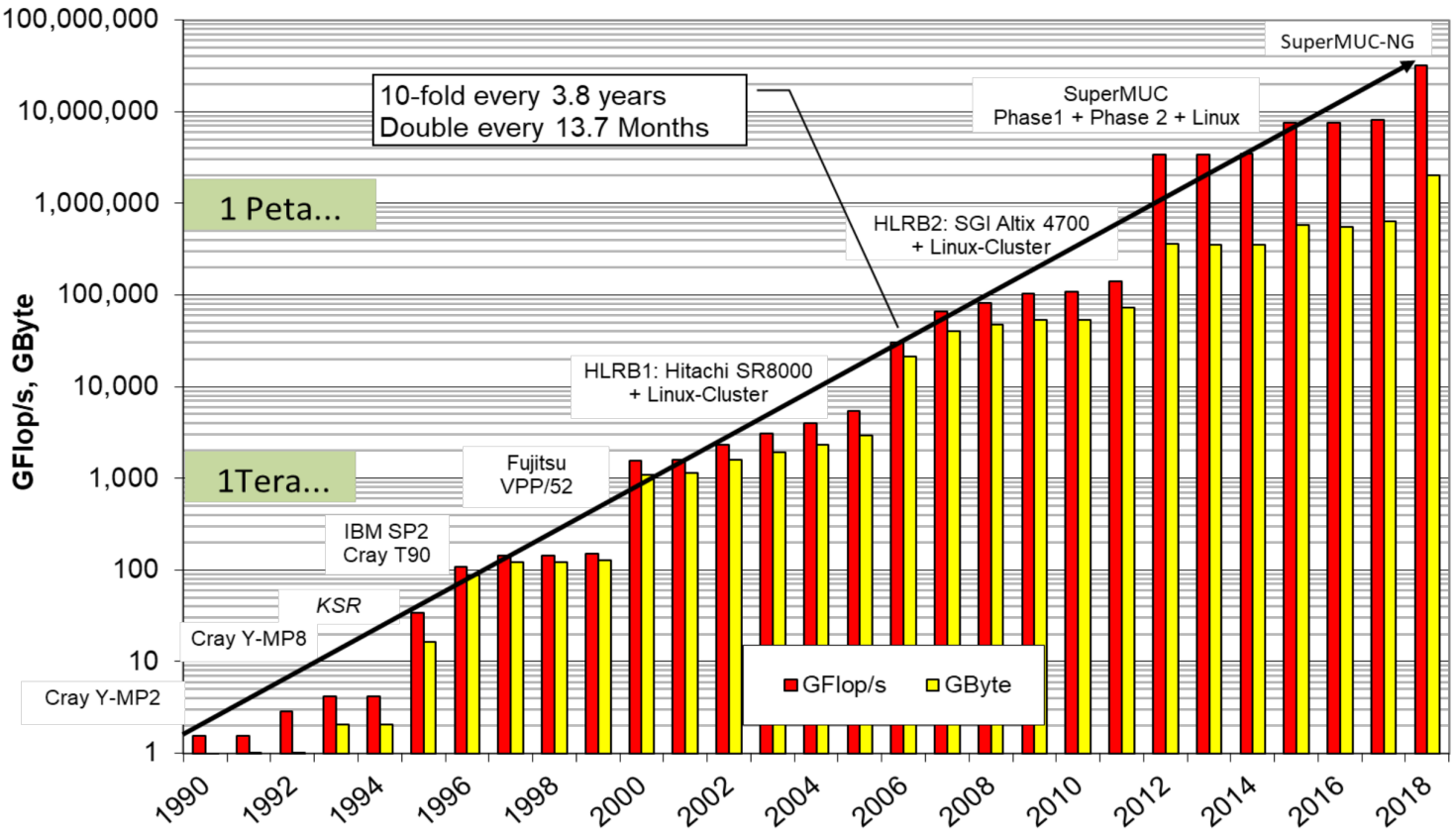


Excursion: “Moore’s Law” @ LRZ



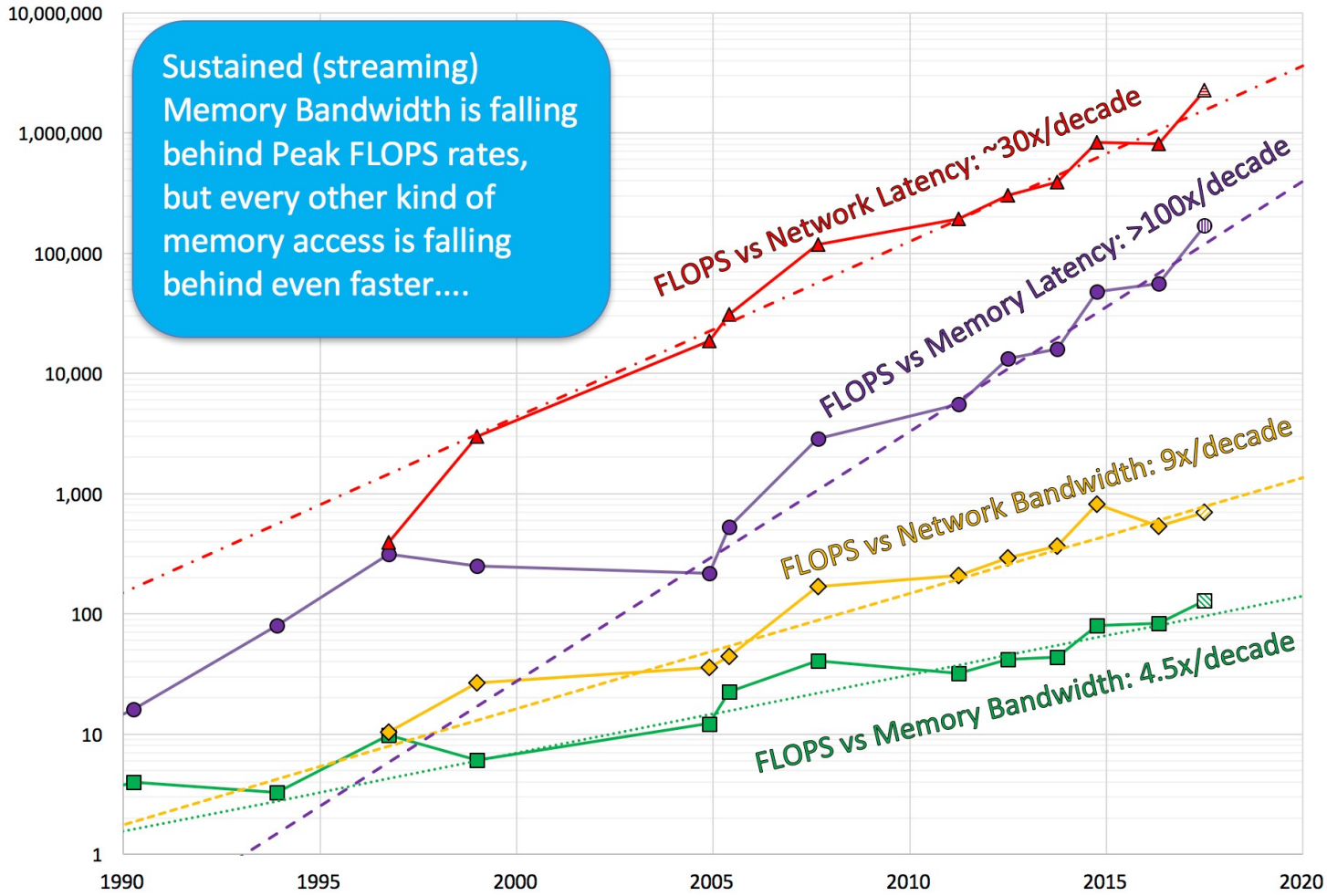
- can you really run your last year’s simulation now in about 50% of the time?!

Excursion: “Moore’s Law” @ LRZ



- can you really run your last year’s simulation now in about 50% of the time?!
- HPC = high Linpack performance!
  DGEMM performance = matrix matrix multiplications

Have You Seen This?



McCalpin, SC16: http://sc16.supercomputing.org/wp-content/uploads/2016/10/McCalpin.jpg

Car-Parrinello Molecular Dynamics (CPMD) Code

Schrödinger equation in the framework of Density Functional Theory

basis set: plane waves + pseudopotentials

pros:

- no Pulay forces
- no basis set superposition errors
- single parameter to tune basis set size
- periodic
- **FFTs for G/R space transformations**

cons:

- isolated systems
- expensive vacuum
- core electrons

Pseudopotential Approach

normconserving NC-PP pseudopotentials

- many plane waves
- typical 3D-FFT grid size: 200 ... 400
- **heavily optimized by IBM Research (Rüschlikon)**
- **dominated by 3D-FFTs**

ultrasoft US-PP Vanderbilt pseudopotentials

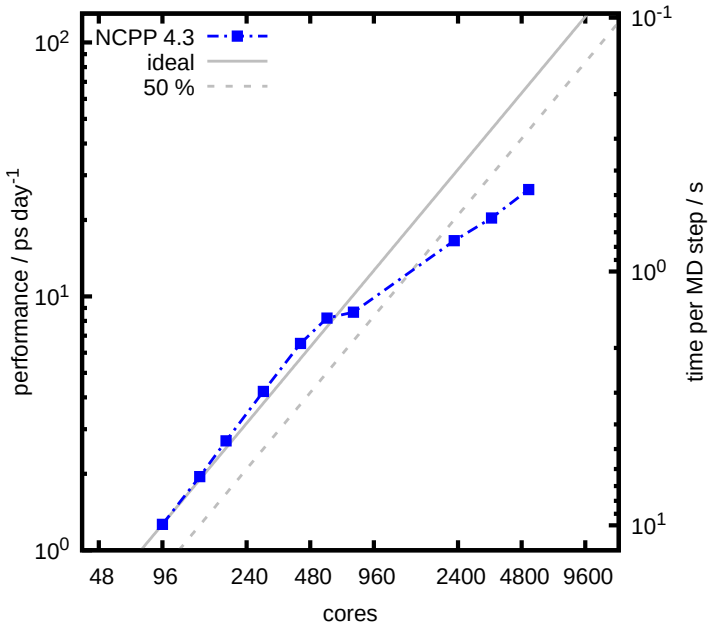
- less plane waves
- typical 3D-FFT grid size: 100 ... 200
- approx. 10x less work in 3D-FFT!
- overhead: $< \Phi | \Phi > \rightarrow < \Phi | S | \Phi >$!
- transformation of overhead into DGEMMs?

CPMD: Status Quo
256 H_2O, 2048 Electrons, SuperMUC-NG

NC-PP:

- $N_\beta = 256$
- FFT: 216^3 (80 Ry)

CPMD old



SuperMUC-NG:

- Intel® Skylake Xeon Platinum 8174 (48 cores / node)
- Fully nonblocking fat tree Intel® OmniPath

NC-PP

- **superlinear scaling!**

CPMD: Status Quo
256 H_2O, 2048 Electrons, SuperMUC-NG

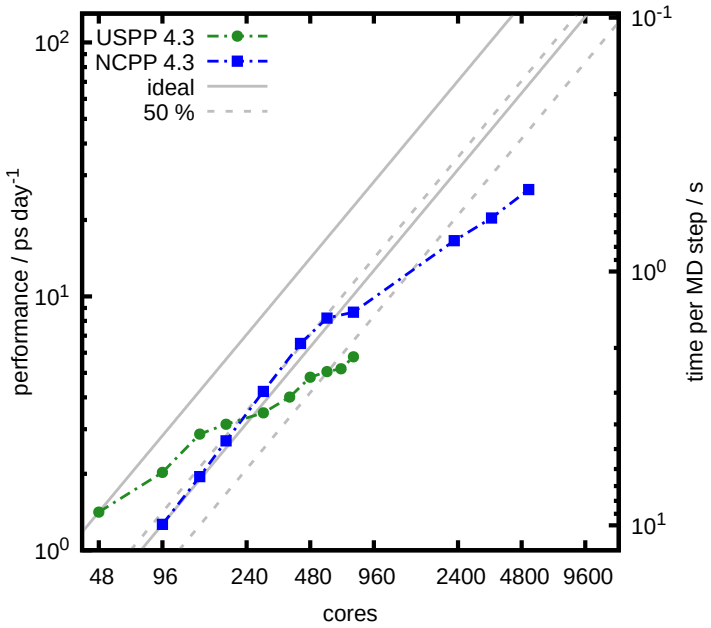
NC-PP:

- $N_\beta = 256$
- FFT: 216^3 (80 Ry)

US-PP:

- $N_\beta = 2560$
- FFT: 120^3 (25 Ry)

CPMD old



SuperMUC-NG:

- Intel® Skylake Xeon Platinum 8174 (48 cores / node)
- Fully nonblocking fat tree Intel® OmniPath

NC-PP

- **superlinear scaling!**
- **best time to solution!**

US-PP

- high performance at low core counts
- OpenMP-MPI hybrid not implemented?

CPMD: Status Quo
256 H_2O, 2048 Electrons, SuperMUC-NG

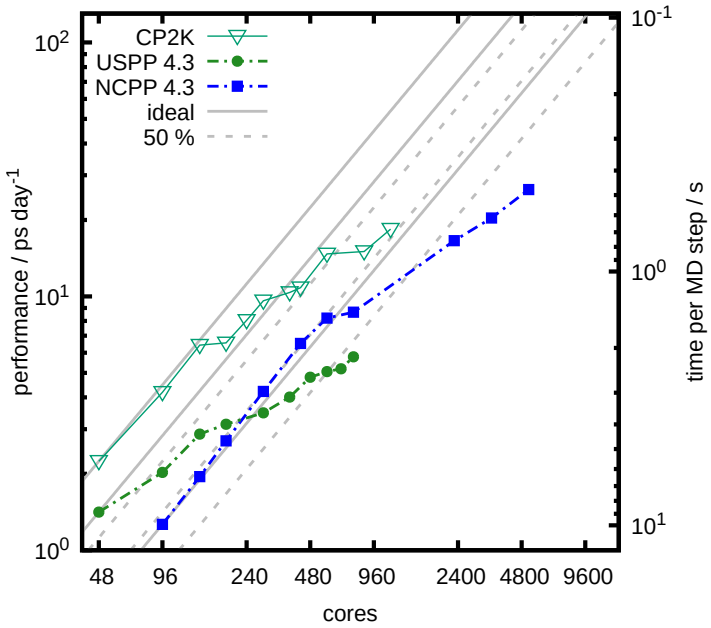
NC-PP:

- $N_\beta = 256$
- FFT: 216^3 (80 Ry)

US-PP:

- $N_\beta = 2560$
- FFT: 120^3 (25 Ry)

CPMD old



SuperMUC-NG:

- Intel® Skylake Xeon Platinum 8174 (48 cores / node)
- Fully nonblocking fat tree Intel® OmniPath

NC-PP

- **superlinear scaling!**
- **best time to solution!**
- **CP2K much more efficient!**

US-PP

- high performance at low core counts
- OpenMP-MPI hybrid not implemented?

How can we improve?

- Understand data structures

How can we improve?

- Understand data structures
- Understand node level performance

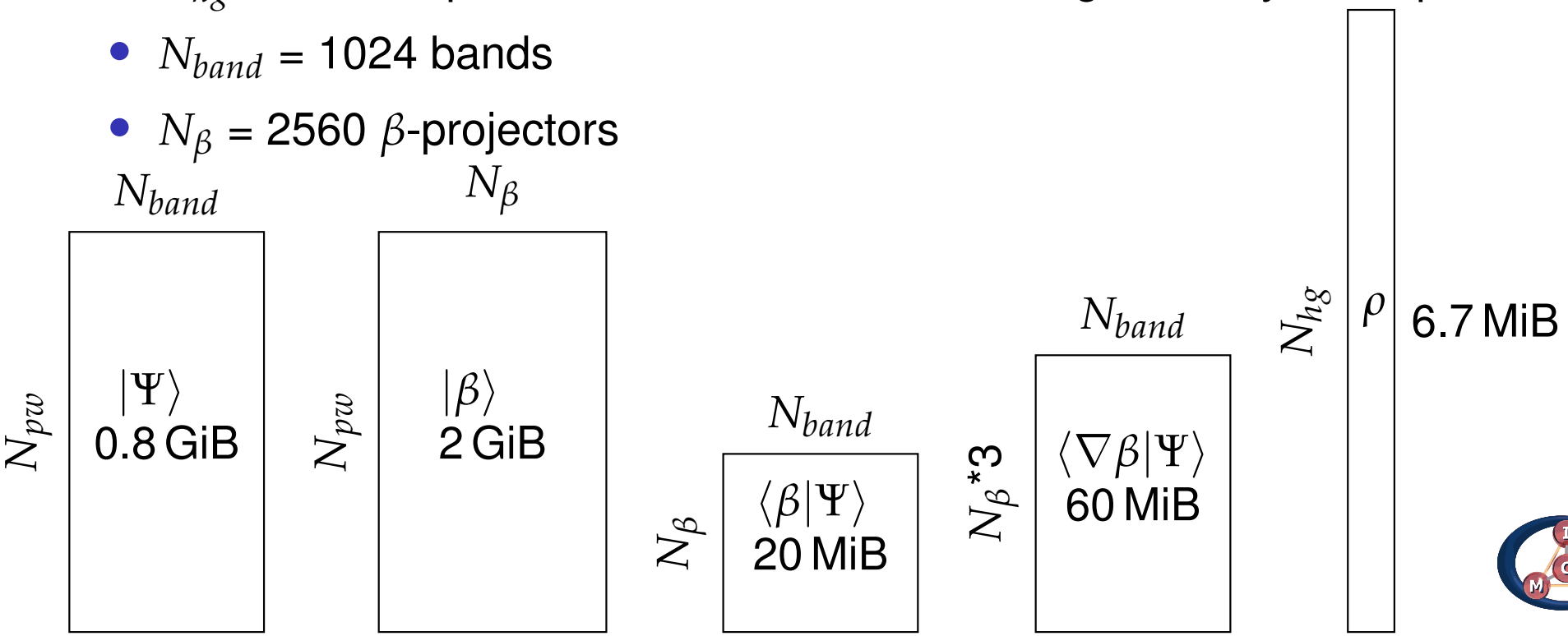
How can we improve?

- Understand data structures
- Understand node level performance
- Understand MPI performance

Data structures

256 H_2O molecules, 25 Ry wavefunction cutoff, 100 Ry charge-density cutoff, 19.734^3 Å^3

- 120^3 FFT grid
- N_{pw} = 54564 plane wave coefficients for wavefunctions in G-space
- N_{hg} = 437792 plane wave coefficients for charge-density in G-space
- N_{band} = 1024 bands
- N_β = 2560 β-projectors



Parallelization Strategies

- Band parallelization: Distribution of N_{band}

Parallelization Strategies

- Band parallelization: Distribution of N_{band}
- Plane wave parallelization: Distribution of N_{pw}

Parallelization Strategies

- Band parallelization: Distribution of N_{band}
- Plane wave parallelization: Distribution of N_{pw}
- Mixture of both: Distribution N_{pw}, work (sometimes) parallelized over N_{band}

Parallelization Strategies

- Band parallelization: Distribution of N_{band}
- Plane wave parallelization: Distribution of N_{pw}
- Mixture of both: Distribution N_{pw}, work (sometimes) parallelized over N_{band}

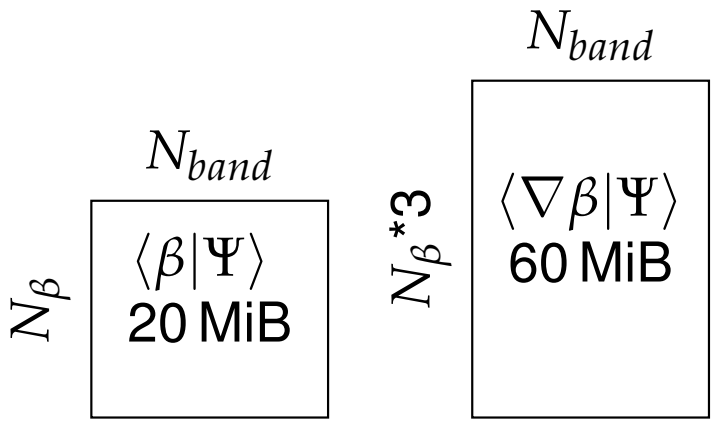
  + Additional: cp_group parallelization: 2 communicators with replicated data
    work (sometimes) parallelized over N_{β}

Distribution of Plane Wave Coefficients

1. Distributed Matrix Matrix Multiplication ($\langle\beta|\Psi\rangle$ and $\langle\nabla\beta|\Psi\rangle$)
2. Distributed 3D-FFT transformation ($|\Psi\rangle_G \rightarrow |\Psi\rangle_R$)

Distributed Matrix Matrix Multiplication

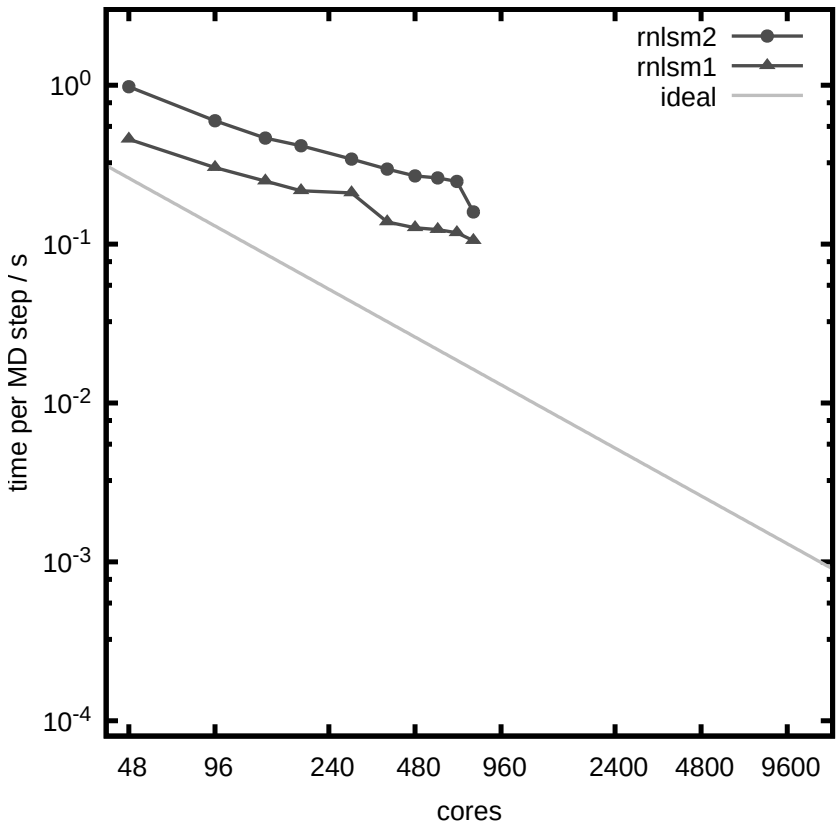
Calculation of $\langle\beta|\Psi\rangle$ and $\langle\nabla\beta|\Psi\rangle$



- Inner dimension distributed (N_{pw})
- $\langle\beta|\Psi\rangle$ and $\langle\nabla\beta|\Psi\rangle$ are replicated at each MPI task!
- Local DGEMMs + MPI allreduce of 20 MiB and 60 MiB

Calculation of $\langle\beta|\Psi\rangle$ and $\langle\nabla\beta|\Psi\rangle$

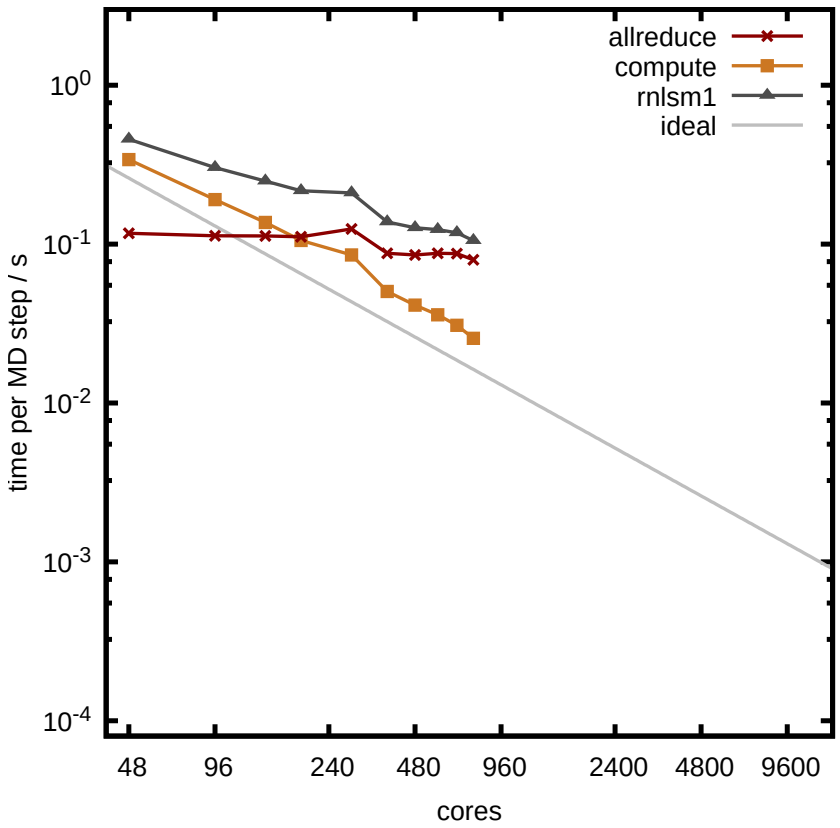
One DGEMM/MPI allreduce call for each atomic species and each β-projector



- Number of MPI tasks / OpenMP threads according to overall best performance!

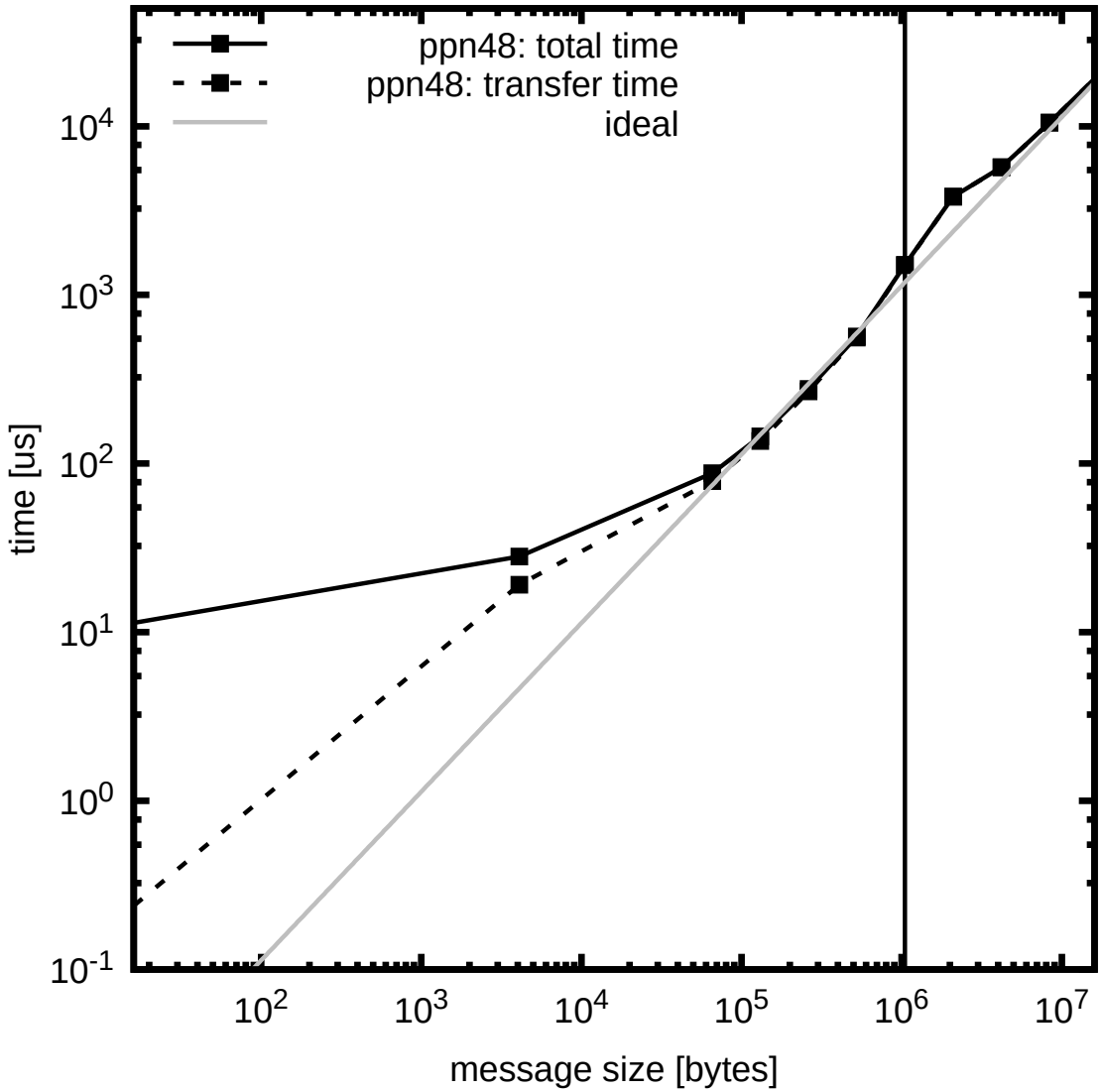
Calculation of $\langle\beta|\Psi\rangle$ and $\langle\nabla\beta|\Psi\rangle$

One DGEMM/MPI allreduce call for each atomic species and each β-projector

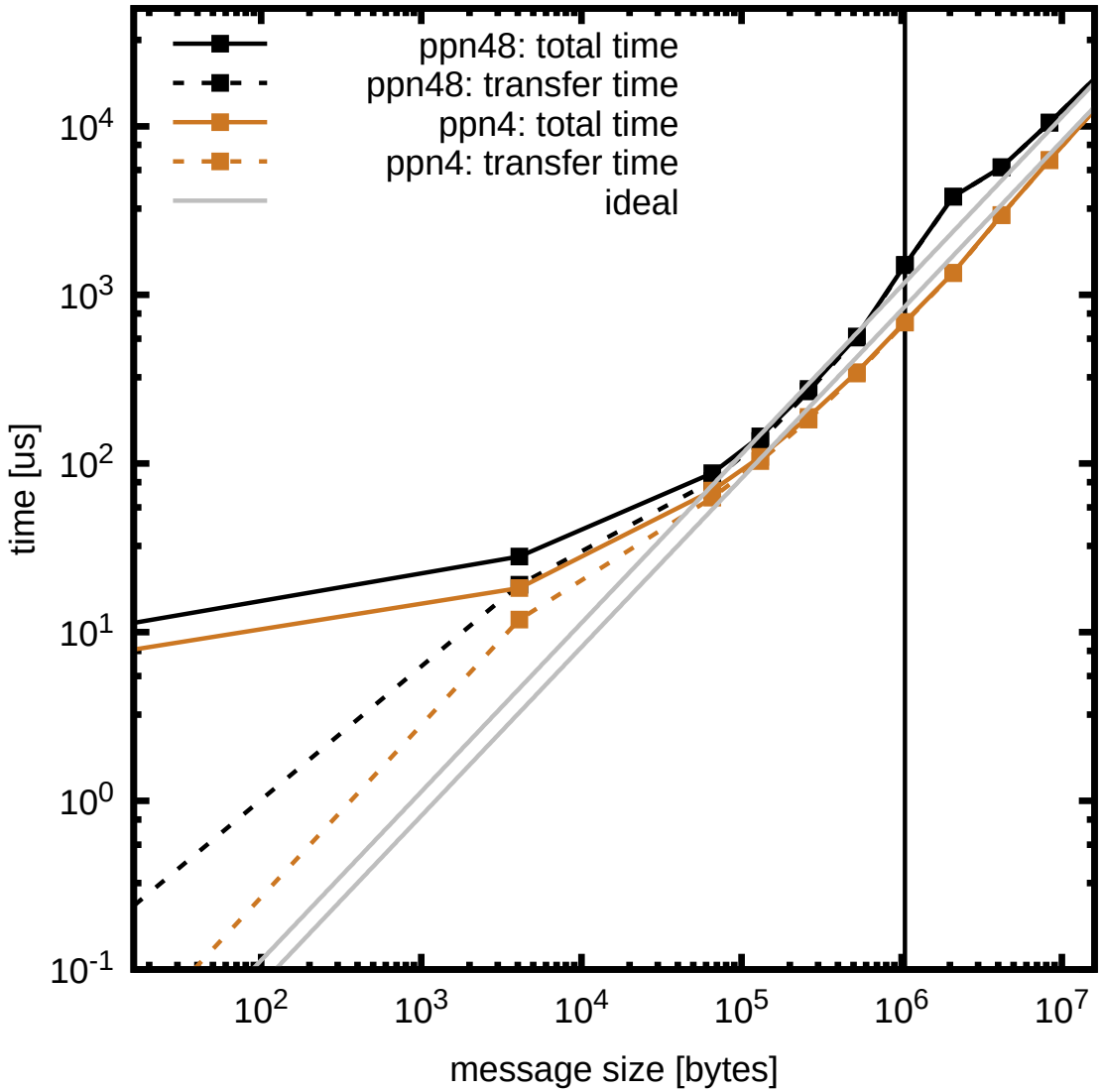


- Number of MPI tasks / OpenMP threads according to overall best performance!
- At 16 nodes (ppn8, 768 cores):
  - MPI comm: 8.0E-2 s/MD
  - Compute: 2.6E-2 s/MD
- MPI allreduce could benefit from larger message size
- DGEMMs should be kept as big as possible (and as quadratic as possible)

MPI Allreduce Performance 16 Nodes



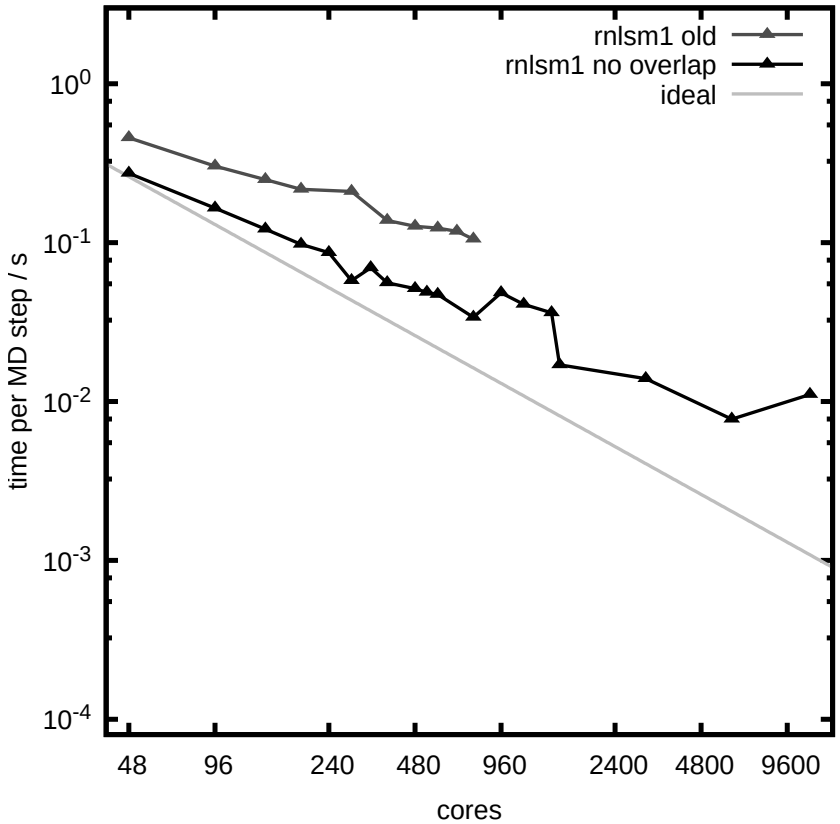
MPI Allreduce Performance 16 Nodes



- Almost no benefit from using fewer MPI ranks
- Allreduce size should be >512 KiB
- Performance target 1.4E-2 s/MD step

Distributed Matrix Matrix Multiplication

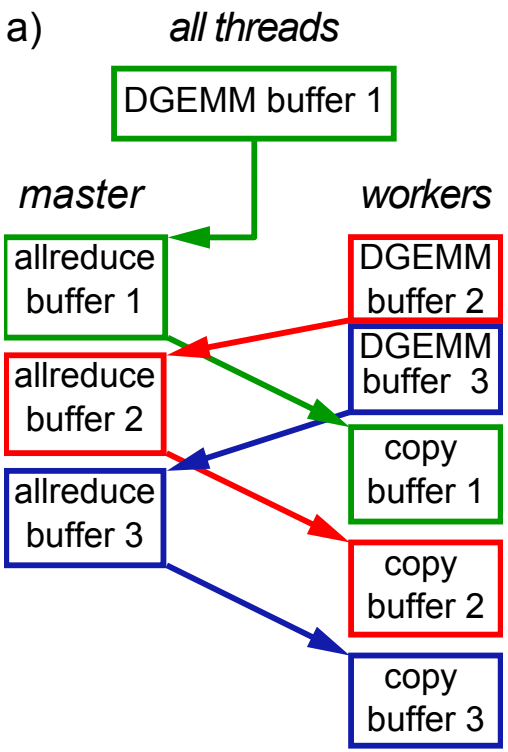
Idea 1: Use a single DGEMM/MPI call



- Number of MPI tasks / OpenMP threads according to overall best performance!
- Total time at 16 nodes: 3.4E-2 s/MD step
- Distribute β-projectors across cp_groups
  cp_groups active at >=1536 cores
  cp_group overhead not shown!

Distributed Matrix Matrix Multiplication

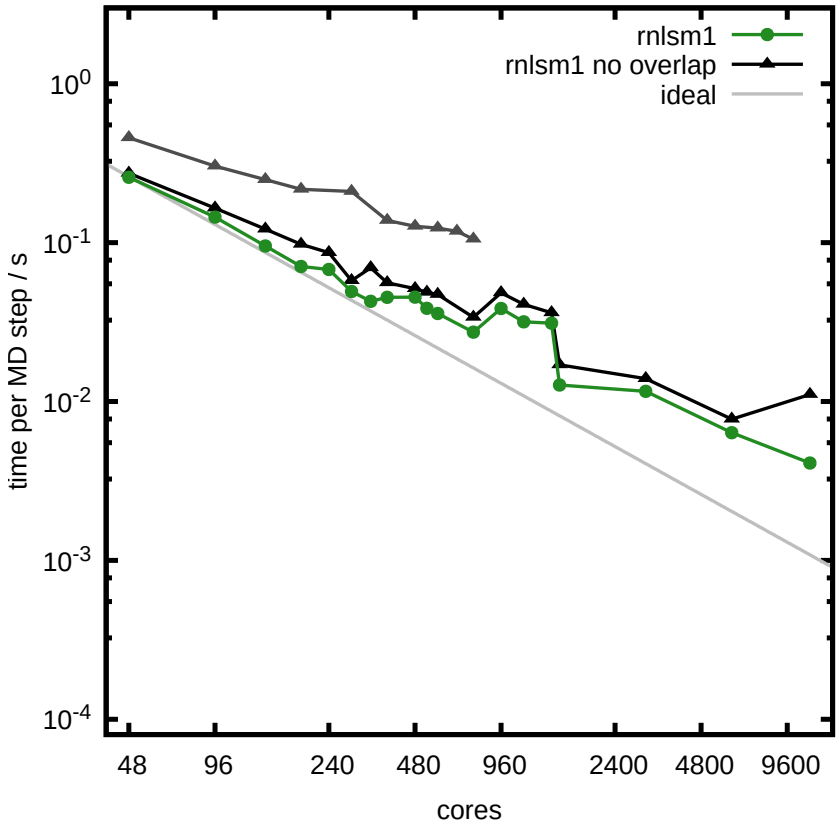
Idea 1: Use a single DGEMM/MPI call
Idea 2: Overlap of communication and computation



- Split DGEMMs into smaller parts
- OpenMP master thread for communication
- OpenMP threads 2:n for DGEMMs (nested OpenMP parallelism!)

Distributed Matrix Matrix Multiplication

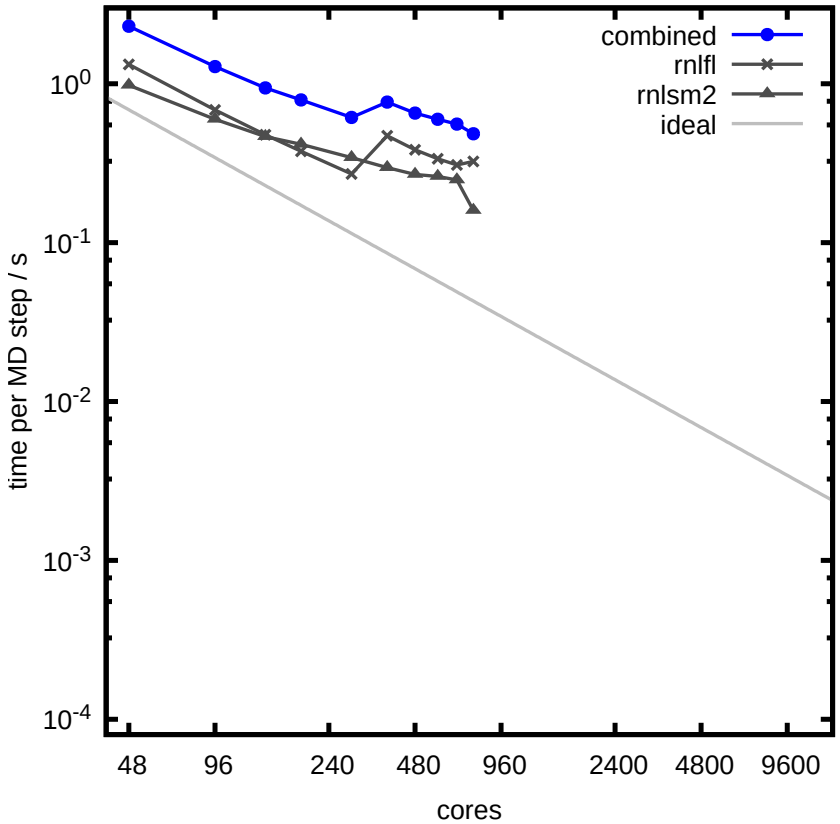
Idea 1: Use a single DGEMM/MPI call
Idea 2: Overlap of communication and computation



- Total time at 16 nodes: 3.4E-2 s/MD step
- Total time (overlap) at 16 nodes: 2.7E-2 s/MD step
- Speedup at 16 Nodes: 3.9 old vs new+overlap

Distributed Matrix Matrix Multiplication

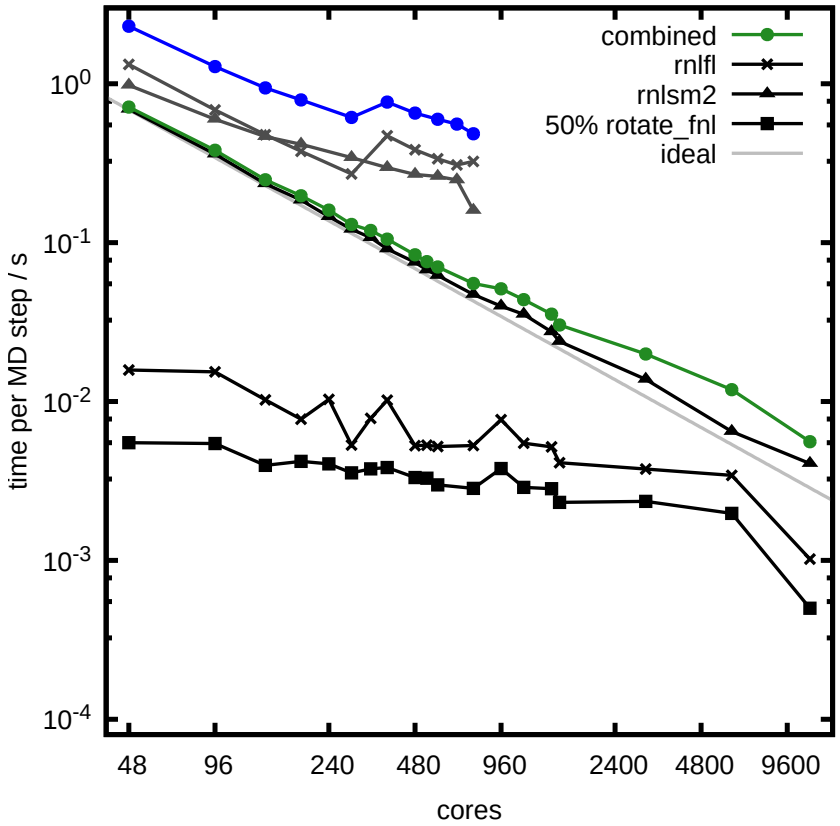
Idea 1: apply same optimizations as for $\langle\beta|\Psi\rangle$
+ optimization of rnlfl (hidden DGEMM $\langle\beta|\Psi\rangle\, x\, \langle\Psi|H|\Psi\rangle$)



- No OpenMP inside rnlfl
- Only rnlfl needs $\langle\nabla\beta|\Psi\rangle$ (ionic forces)

Distributed Matrix Matrix Multiplication

Idea 1: apply same optimizations as for $\langle\beta|\Psi\rangle$
+ optimization of rnlfl (hidden DGEMM $\langle\beta|\Psi\rangle\, x\, \langle\Psi|H|\Psi\rangle$)
Idea 2: Discard MPI allreduce



- Optimized rnlfl discards parallelization
- rottr_fnl (DGEMM) parallelized at node level only
- Discard MPI allreduce
- Almost ideal scaling!

Distribution of Plane wave Coefficients

1. Distributed Matrix Matrix Multiplication ($\langle\beta|\Psi\rangle$ and $\langle\nabla\beta|\Psi\rangle$)
2. Distributed 3D-FFT transformation ($|\Psi\rangle_G \rightarrow |\Psi\rangle_R$)

Distributed 3D-FFT

3D-FFT to transform $|\Psi\rangle$ to real space

$\rightarrow$ redistribution of data and local 3D-FFT (e.g. VASP)
$\rightarrow$ distributed 3D-FFT (e.g. QE, CPMD)

Distributed 3D-FFT - Parallelization

- Distribute planes in real space (xz in QE, yz in CPMD)
  scalability limited to number of planes, here 120 MPI tasks
  (remember: 48 cores per node (LRZ Munich: SuperMUC-NG)
  128 cores per node (HLRS Stuttgart: HAWK))

Distributed 3D-FFT - Parallelization

- Distribute planes in real space (xz in QE, yz in CPMD)
  scalability limited to number of planes, here 120 MPI tasks
  (remember: 48 cores per node (LRZ Munich: SuperMUC-NG)
  128 cores per node (HLRS Stuttgart: HAWK))

- Taskgroup / cp_group parallelization
  work on more states in parallel
  on the fly redistribution (taskgroups) / data replication +
  synchronization (cp_groups)

Distributed 3D-FFT - Parallelization

- Distribute planes in real space (xz in QE, yz in CPMD)
  scalability limited to number of planes, here 120 MPI tasks
  (remember: 48 cores per node (LRZ Munich: SuperMUC-NG)
  128 cores per node (HLRS Stuttgart: HAWK))

- Taskgroup / cp_group parallelization
  work on more states in parallel
  on the fly redistribution (taskgroups) / data replication +
  synchronization (cp_groups)

- Distribute partial planes in real space (QE (??))

Distributed 3D-FFT - Parallelization

- Distribute planes in real space (xz in QE, yz in CPMD)
  scalability limited to number of planes, here 120 MPI tasks
  (remember: 48 cores per node (LRZ Munich: SuperMUC-NG)
  128 cores per node (HLRS Stuttgart: HAWK))

- Taskgroup / cp_group parallelization
  work on more states in parallel
  on the fly redistribution (taskgroups) / data replication +
  synchronization (cp_groups)

- Distribute partial planes in real space (QE (??))

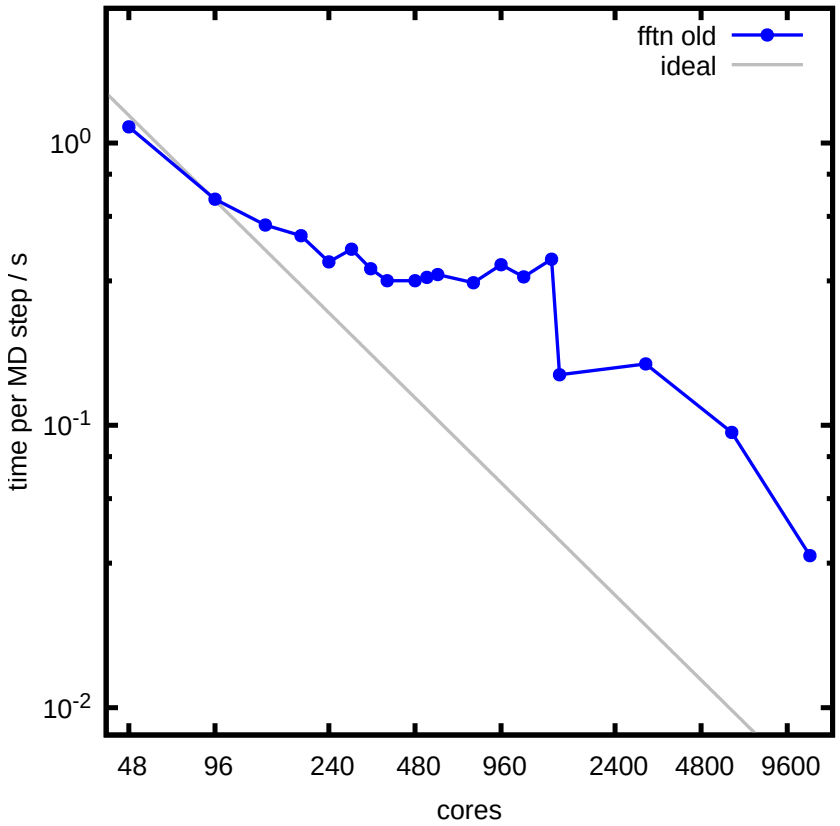
- Add more resources to a single MPI task hybrid parallelization
  (MPI + X, X = OpenMP, accelerators, ...)
  somewhat similar to partial plane distribution

Distributed 3D-FFT MPI + OpenMP

Computer Chemie Centrum

Already implemented in CPMD

Performance of new implementation with old 3D-FFT routines

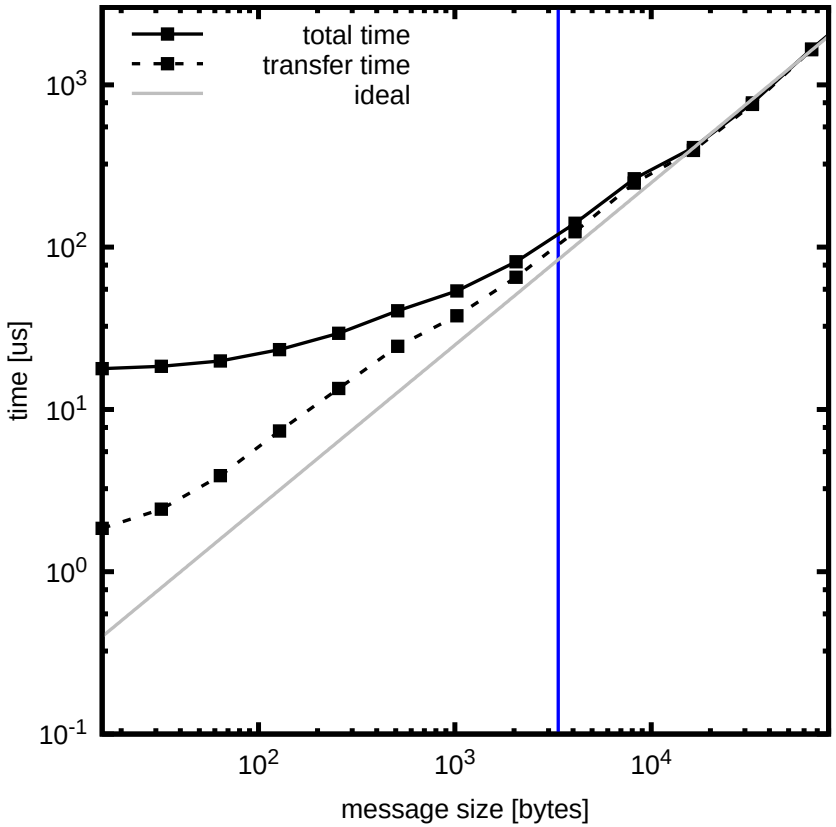


- Number of MPI tasks / OpenMP threads according to overall best performance!
- Scalability of FFT in hybrid parallelization: 240 cores
- Large performance benefit of using cp_group parallelization at 1536 cores!
  (cp_group overhead not shown!)

→ overall unsatisfying scalability

MPI All to All Performance

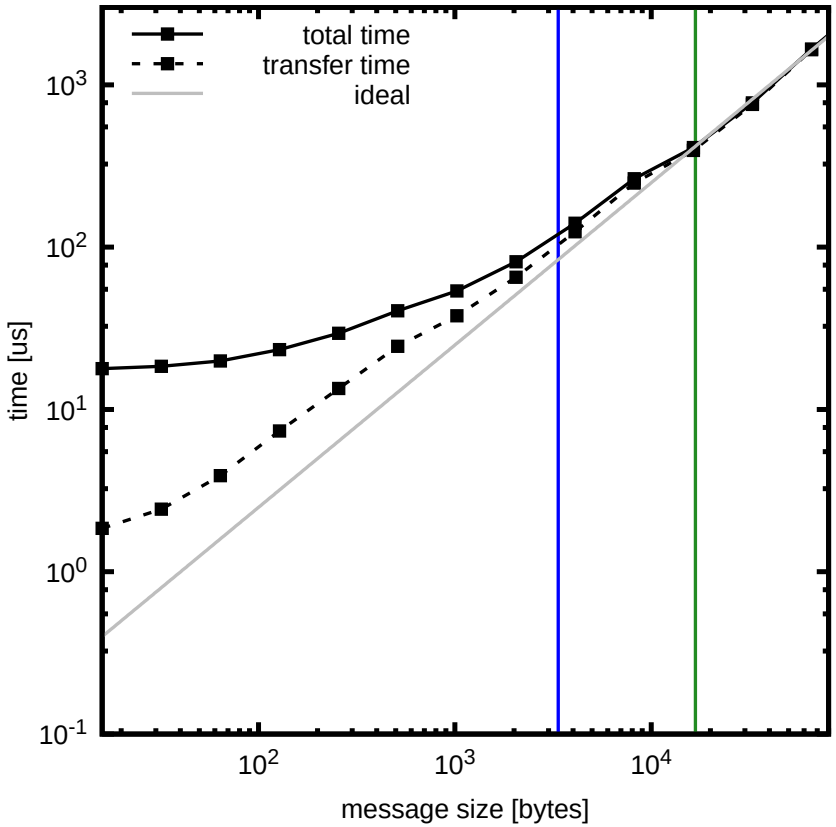
5 nodes, 8 MPI tasks per node



- All to all message size at 240 cores: 3360 bytes (3 planes * 70 rays)
- All to all latency bound!
- Message size will decrease with increasing MPI tasks

New batched 3D-FFT

Idea 1: Combine A2A communication



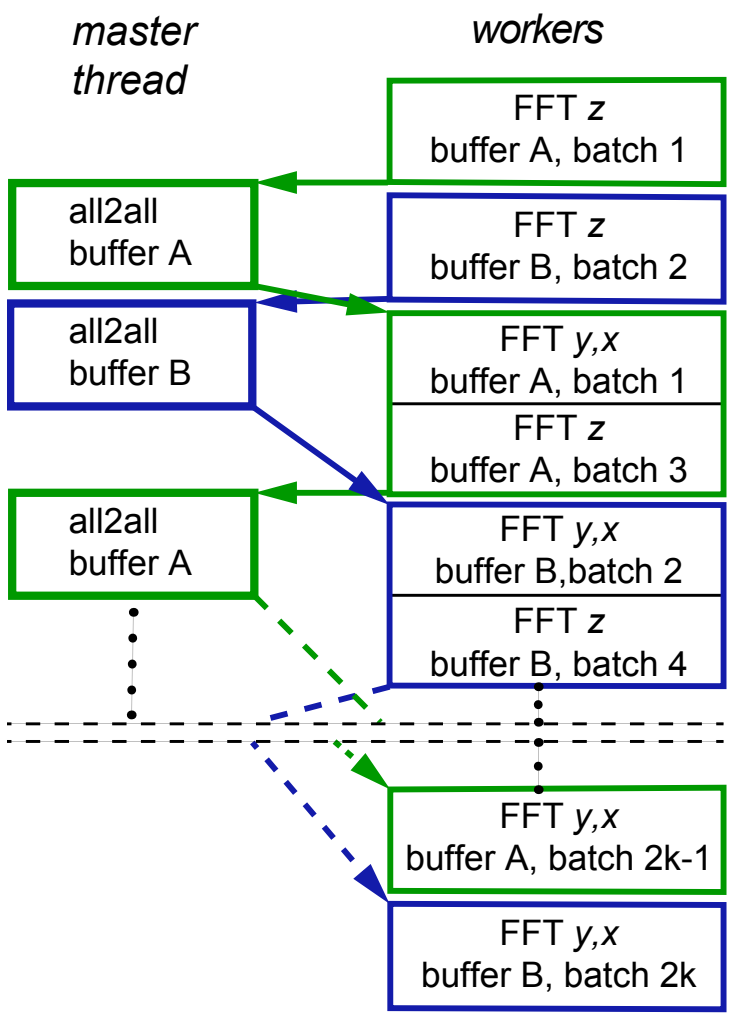
states/A2A	1	2	3	4
time [ms]	514	457	428	418
	5	6	7	8
	404	423	417	423

- 1D-FFT performance might go down with increasing dataset!
- Autotuning necessary!

New batched 3D-FFT

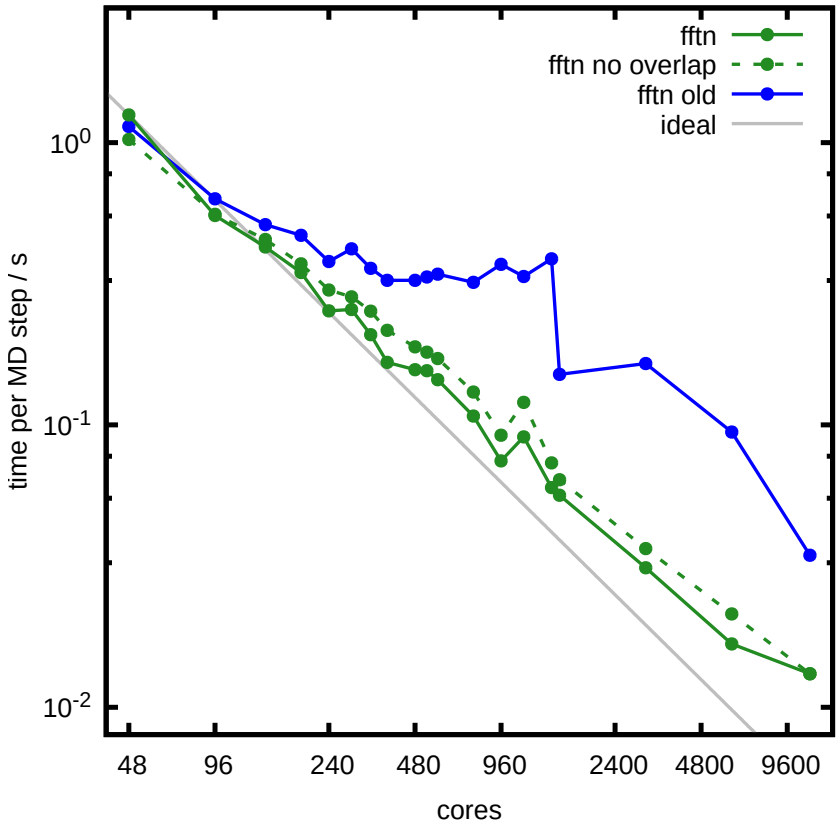
Idea 1: Combine A2A communication
Idea 2: Work on two batches to hide communication

states A2A	msg size [bytes]	time [ms]	time [ms] (overlap)
1	3360	514	508
2	6720	457	379
3	10080	428	358
4	13440	418	352
5	16800	404	371
6	20160	423	376
7	23520	417	403
8	26880	423	399
9	30240	439	407
10	33600	462	410



New Batched 3D-FFT

Performance of new implementation with new batched 3D-FFT routines

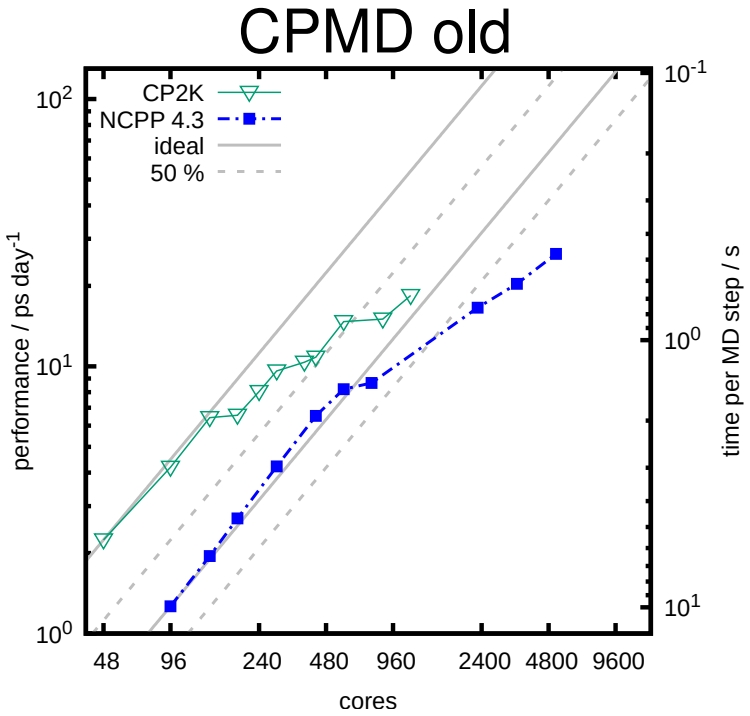


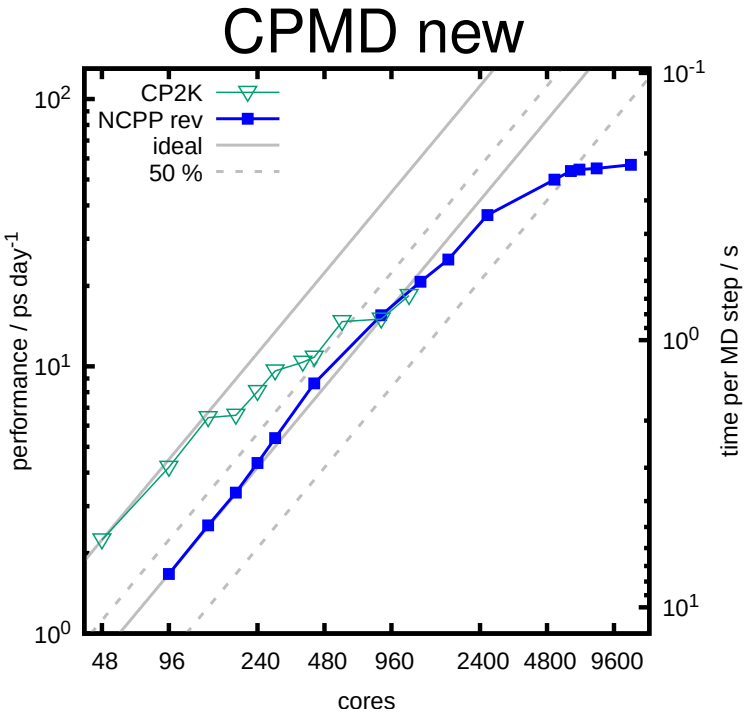
- Number of MPI tasks / OpenMP threads according to overall best performance!
- Scalability of FFT in hybrid parallelization: > 4800 cores
- Very small performance benefit of using cp_group parallelization at 1536 cores!
  (cp_group overhead not shown!)

→ satisfying scalability

CPMD US-PP > 15.000 Code Lines Changed

256 H_2O, 2048 Electrons, SuperMUC-NG



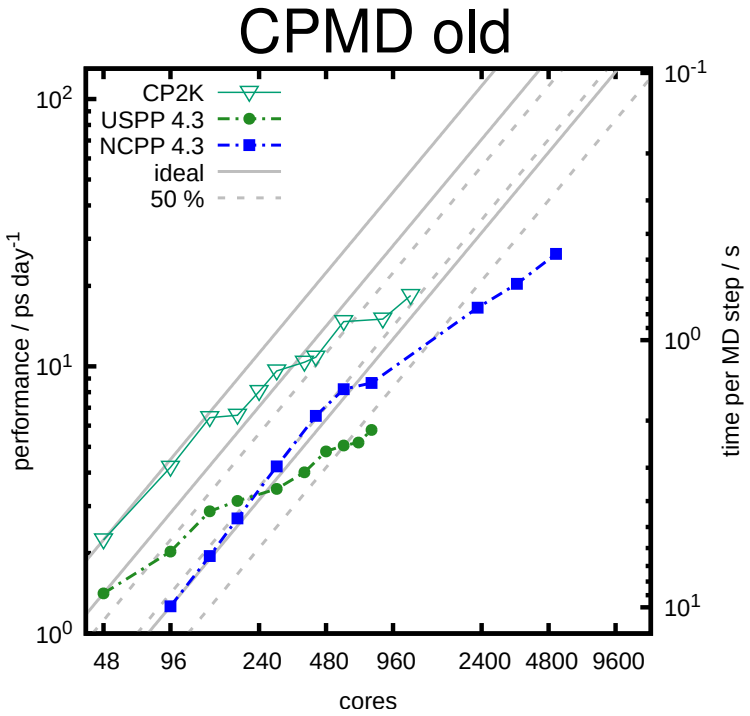


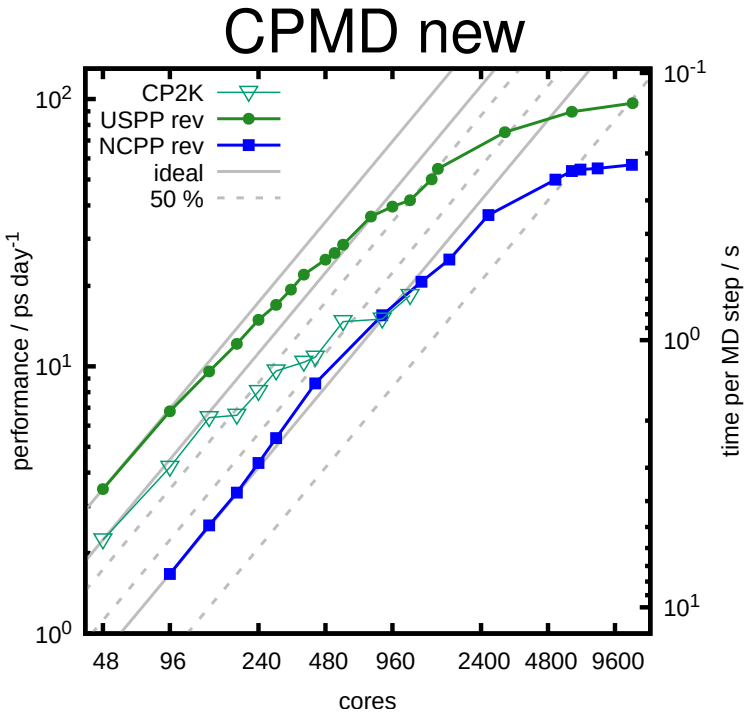
NC-PP

- **1.2x-1.3x speedup**
- **50 ps per day**
- **Outperforms CP2K**

CPMD US-PP > 15.000 Code Lines Changed

256 H_2O, 2048 Electrons, SuperMUC-NG





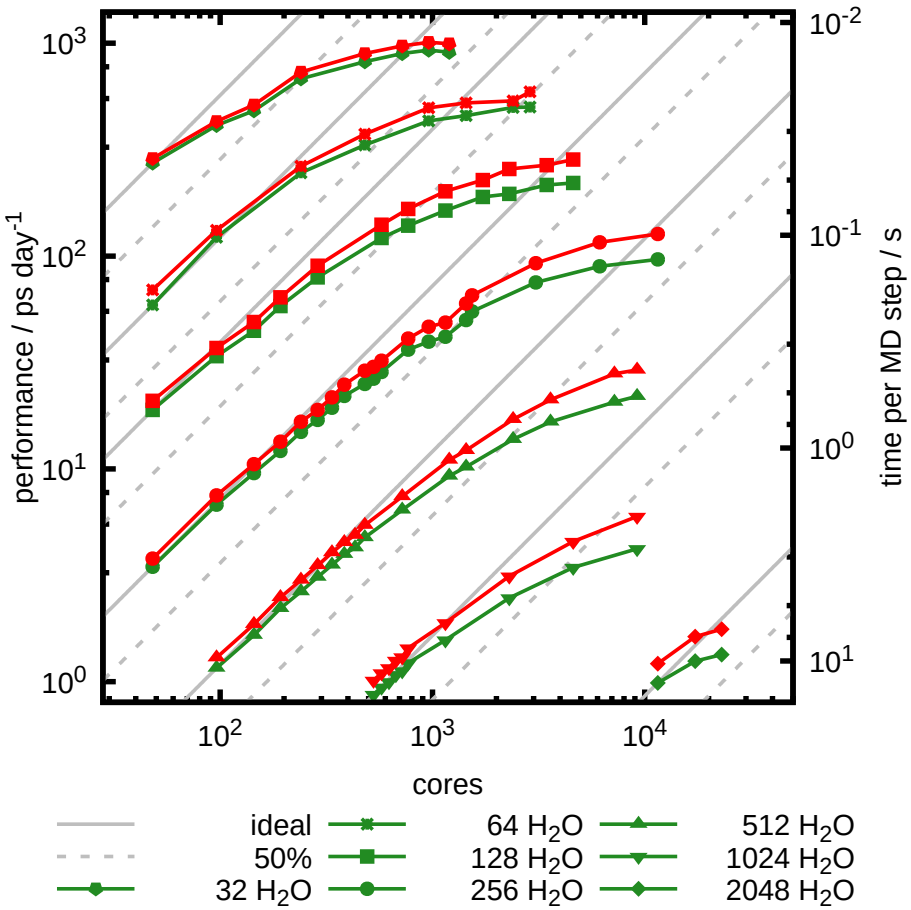
NC-PP

- **1.2x-1.3x speedup**
- **50 ps per day**
- **Outperforms CP2K**

US-PP

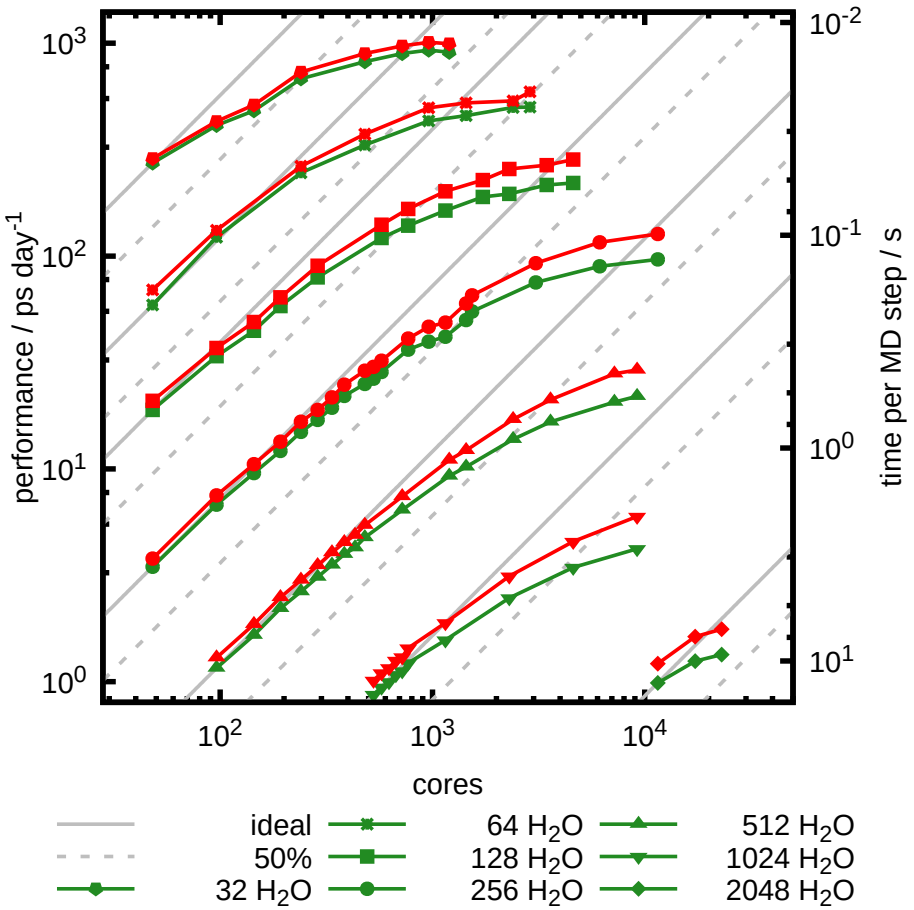
- **> 2.0x speedup**
- **> 70 ps per day**
- **Best time to solution!**
- **Most efficient!**

Strong Scaling Benchmark: 32-2048 H_2O Molecules @SuperMUC-NG



- Optimized for 1400-3000 electrons

Strong Scaling Benchmark: 32-2048 H_2O Molecules @SuperMUC-NG



- Optimized for 1400-3000 electrons
- Excellent performance also for tiny systems! More than **950 ps/day** -> QM/MM simulations
- If you really want to: affordable DFT calculation with 2048 H_2O molecules, 16384 electrons! Code not even optimized...

Conclusions

- Overlapping of communication and computation
  - MPI_IAllreduce did not work
  - Nested OpenMP parallelization can be complicated
  - Large message sizes are more important

Conclusions

- Overlapping of communication and computation
  - MPI_IAllreduce did not work
  - Nested OpenMP parallelization can be complicated
  - Large message sizes are more important
- Highly optimized OpenMP Lapack routines (eigenvalues, matrix inversion, Cholesky factorization) much better than ScaLapack/ELPA (for sizes up to 2000)

Conclusions

- Overlapping of communication and computation
  - MPI_IAllreduce did not work
  - Nested OpenMP parallelization can be complicated
  - Large message sizes are more important
- Highly optimized OpenMP Lapack routines (eigenvalues, matrix inversion, Cholesky factorization) much better than ScaLapack/ELPA (for sizes up to 2000)
- Avoiding MPI communication is always desirable

Conclusions

- Overlapping of communication and computation
  - MPI_IAllreduce did not work
  - Nested OpenMP parallelization can be complicated
  - Large message sizes are more important
- Highly optimized OpenMP Lapack routines (eigenvalues, matrix inversion, Cholesky factorization) much better than ScaLapack/ELPA (for sizes up to 2000)
- Avoiding MPI communication is always desirable
- Large DGEMMs/communication calls $\rightarrow$ large allocations!
  - Allocation of memory (very) expensive
  - Save arrays
  - Use workspace management $\rightarrow$ Fortran pointers
    - Use small functions to get rid of pointer attribute
    - Library will be soon at github.com/RRZE-HPC/flexi-scratch

Project

Prof. Dr. Bernd Meyer

PD Dr. Gerald Mathias

lrz Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

PRACE

Regionales RechenZentrum Erlangen
Der IT-Dienstleister der FAU

KONWIHR III

Calculation of Augmentation Charges & New D

rhov: calculation of augmentation charges
newd: calculation of D, Q and ionic forces due to position decency of Q

Calculation of Augmentation Charges & New D

rhov: calculation of augmentation charges
newd: calculation of D, Q and ionic forces due to position decency of Q

- Recalculation of Q-function at every call in both routines

Calculation of Augmentation Charges & New D

rhov: calculation of augmentation charges
newd: calculation of D, Q and ionic forces due to position decency of Q

- Recalculation of Q-function at every call in both routines
- Calculation of becsum in both routines, not parallelized

Calculation of Augmentation Charges & New D

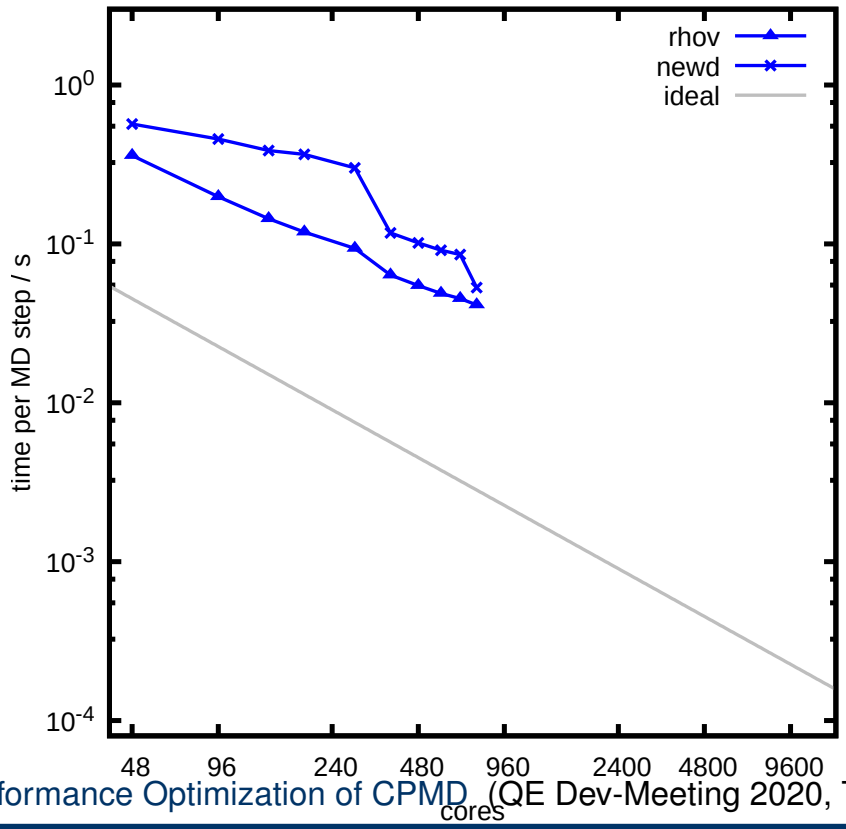
rhov: calculation of augmentation charges
newd: calculation of D, Q and ionic forces due to position decency of Q

- Recalculation of Q-function at every call in both routines
- Calculation of becsum in both routines, not parallelized
- Structure factors for charge-density (and wavefunctions) are precalculated and saved

Calculation of Augmentation Charges & New D

rhov: calculation of augmentation charges
newd: calculation of D, Q and ionic forces due to position decency of Q

- Recalculation of Q-function at every call in both routines
- Calculation of becsum in both routines, not parallelized
- Structure factors for charge-density (and wavefunctions) are precalculated and saved



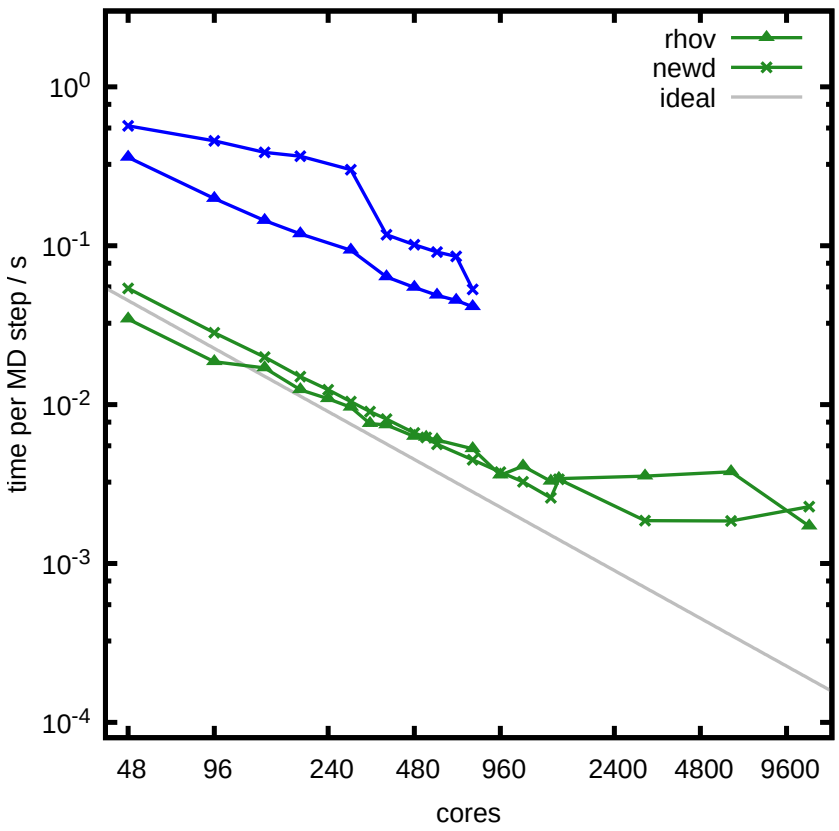
- rhov
  - DGEMV for each β-projector combination for each atomic species (37)
- Newd
  - DGEMM for each atomic species (2)
    separate DGEMM for ionic forces (37)
  - Summation of sparse array deeq

Calculation of Augmentation Charges & New D

- Saving Q function at initialization
- Parallelization and optimization of becsum calculation
- Blocking of N_{hg}

Calculation of Augmentation Charges & New D

- Saving Q function at initialization
- Parallelization and optimization of becsum calculation
- Blocking of N_{hg}



- rhov
  - DGEMM for each species(2)
- Newd
  - Merge DGEMM if ionic forces are needed (2)
  - Summation of packed array