Performance Optimization of CPMD

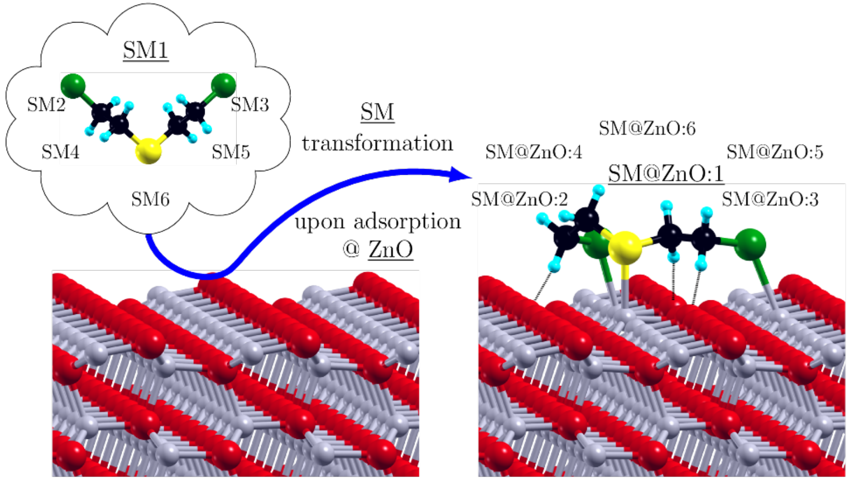
Tobias Klöffel[1,2], Bernd Meyer[1], Gerald Mathias[3]

[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)
Friedrich-Alexander-Universität Erlangen-Nürnberg

[3] Leibniz Supercomputing Centre (LRZ), Garching

Static Simulations: Adsorption on Surfaces



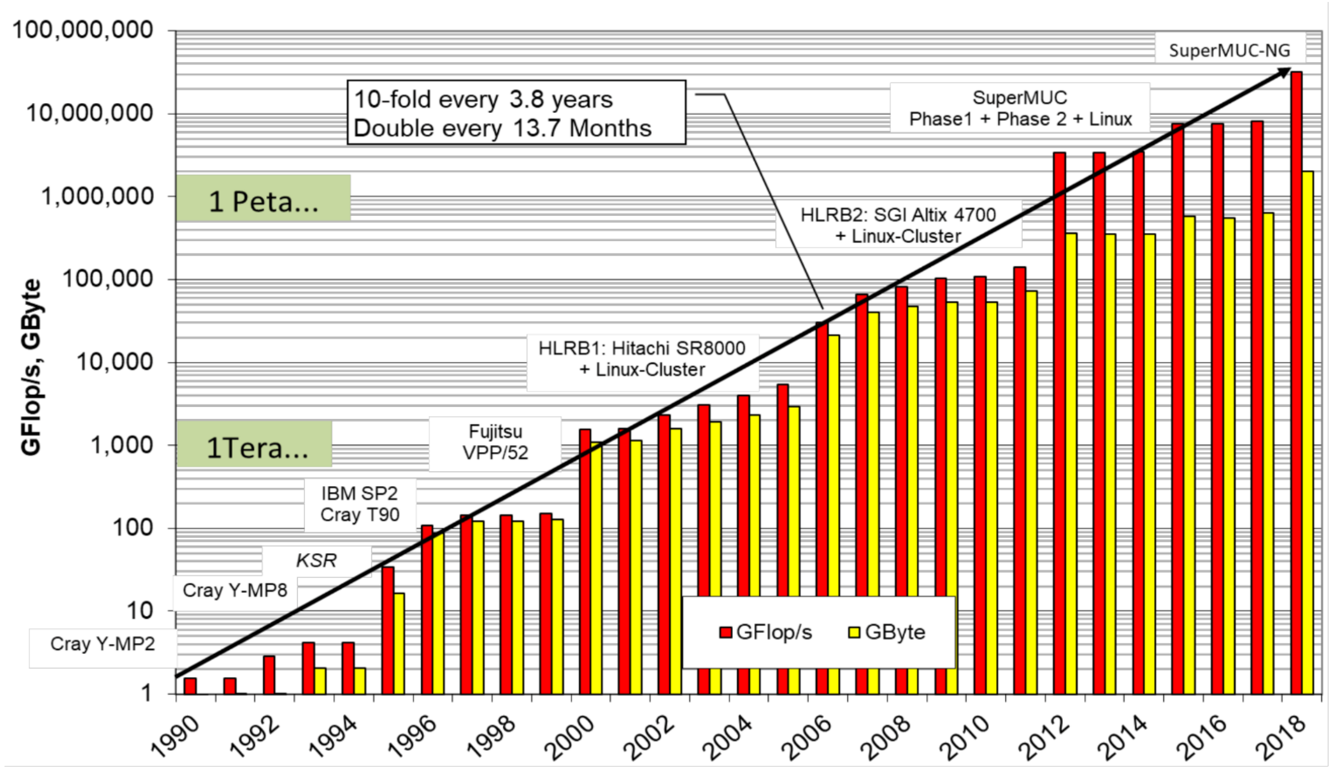
- 6 most stable gas phase conformers probed
- 279 independent geometry optimizations
- 150-250 geometry updates per geometry optimization
- 20 wave-function updates per geometry update

  -> 1E7 wave-function & 1E6 force updates

Dynamic Simulations: Molecules in Solution

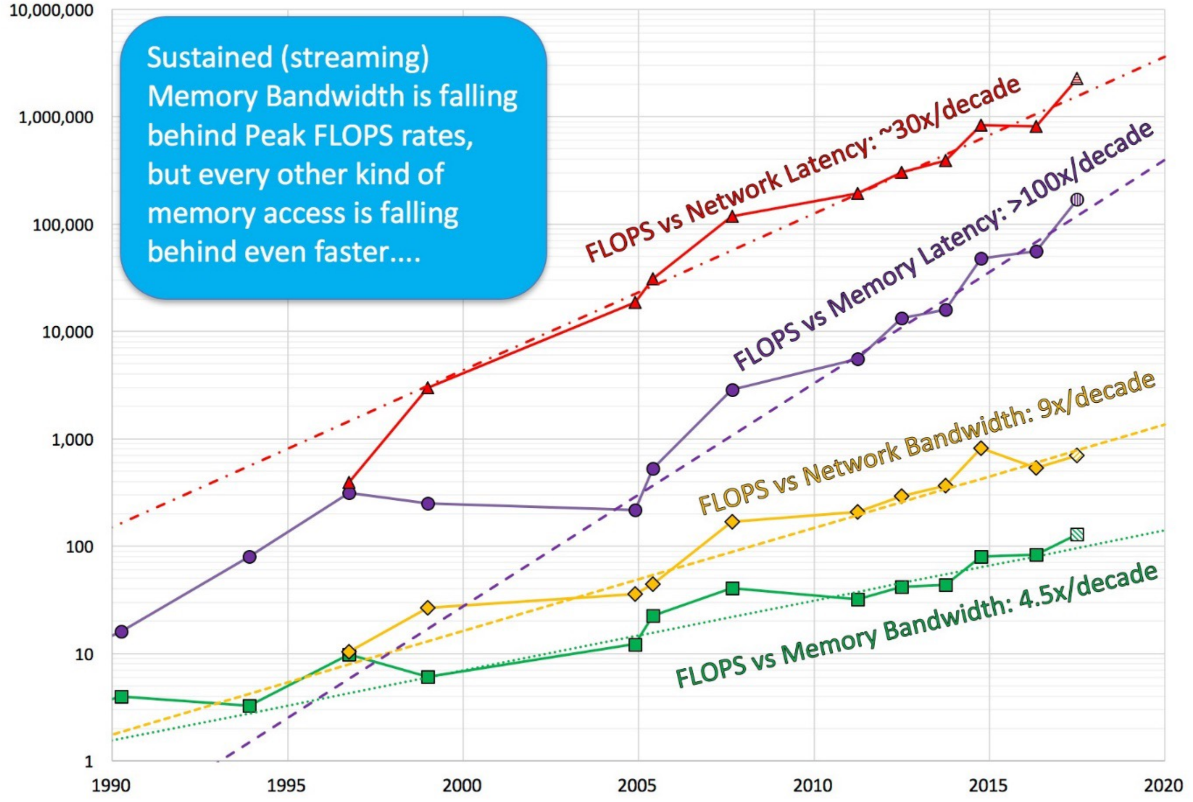
- 50 ps equilibration
- 250 ps simulated time
- 0.145 fs time step -> 20 M force **and** wave-function updates
- Multiple trajectories mandatory!
- System size may increase (cubic scaling DFT!)
- Parallelization of time not possible
- Vastly different computational requirements
  -> Extremely efficient code needed

Moore’s Law @ LRZ



Linpack performance!
~ DGEMM performance

Moore’s Law @ LRZ



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
- **Thouroughly optimized by IBM Research (Rüschlikon)**
- **Dominated by 3D-FFTs**

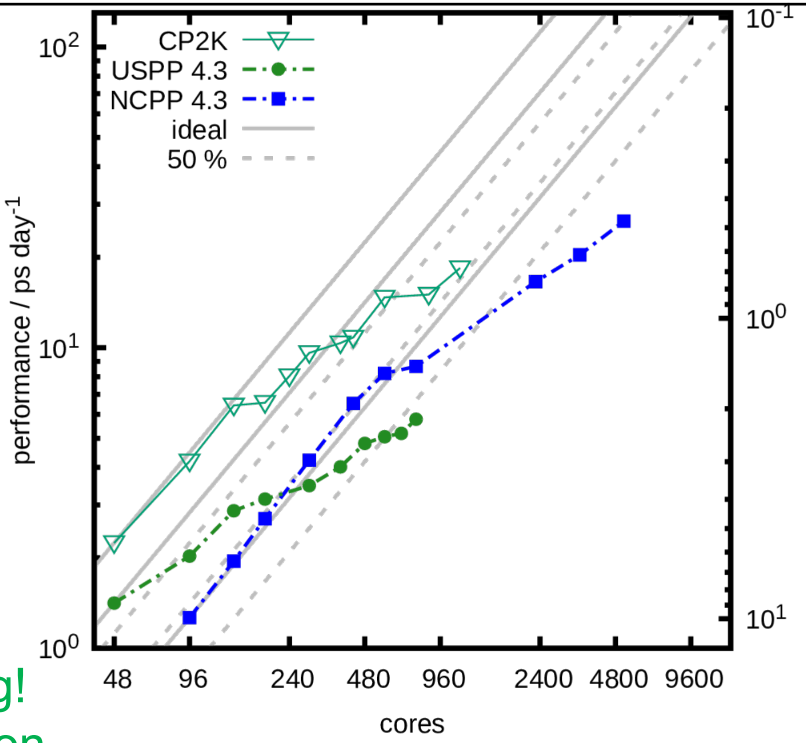
Ultrasoft US-PP Vanderbilt pseudopotentials

- Less plane waves
- Typical 3D-FFT grid size: 100 ... 200
- Approx. 10x less work in 3D-FFT!
- Overhead: $< \Phi \mid \Phi > \; \rightarrow \; < \Phi \mid S \mid \Phi >$!
- Transformation of overhead into DGEMMs?

CPMD Strong Scaling: Starting Point

NC-PP:
- N_β= 256
- FFT: 216^3 (80 Ry)

US-PP:
- N_β= 2560
- FFT: 120^3 (25 Ry)



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
- No OpenMP/MPI hybrid parallelization

CPMD Internal Instrumentation

```
****************************************************
SUBROUTINE CALLS          SELF TIME         TOTAL TIME
                     AVERAGE   MAXIMUM   AVERAGE   MAXIMUM
cpmd             1      0.17      0.18    108.66    108.67
mdpt             1      0.02      0.02    107.96    107.96
mdmain           1      0.38      0.70    107.94    107.94
forcedr         51      0.01      0.01     98.24     98.24
noforce         51      1.87      2.28     98.23     98.24
rnlsm          102      0.00      0.00     37.62     37.74
rnlsm2          51     22.00     22.91     29.63     29.74
rscpot          51      0.01      0.01     21.30     21.32
vpsi            51      3.39      3.43     14.81     14.83
rhoofr          51      1.97      2.04     14.53     14.53
invfftn         51     11.59     11.64     11.59     11.64
fwfftn          51     11.42     11.44     11.42     11.44
nlforce         51      9.05      9.28      9.05      9.28
rnlsm2_b       306      7.63      8.41      7.63      8.41
rnlsm1          51      5.95      6.21      7.99      8.01
vofrho          51      0.00      0.00      5.32      5.33
rotate          85      4.06      4.21      4.06      4.21
ovlap          103      3.95      4.08      3.95      4.08
```

```
=====================================================
COMMUNICATION TASK  AVERAGE MESSAGE LENGTH  NO. CALLS
SEND/RECEIVE                  54689. BYTES     19476.
BROADCAST                    814689. BYTES       595.
GLOBAL SUMMATION            2981743. BYTES      2049.
ALL TO ALL COMM              273449. BYTES     52785.
ALLGATHERV                  2615031. BYTES       409.

                            PERFORMANCE     TOTAL TIME
SEND/RECEIVE                3833.006  MB/S   0.278 SEC
BROADCAST                   2310.022  MB/S   0.210 SEC
GLOBAL SUMMATION            2000.989  MB/S  13.196 SEC
GLOBAL MULTIPLICATION          0.000  MB/S   0.001 SEC
ALL TO ALL COMM             1333.118  MB/S  10.827 SEC
ALLGATHERV                   480.392  MB/S   2.226 SEC
SYNCHRONISATION                              0.510 SEC
```

- Timings excluding/including subroutines
- Communication heavy on global summation and all-to-all communication

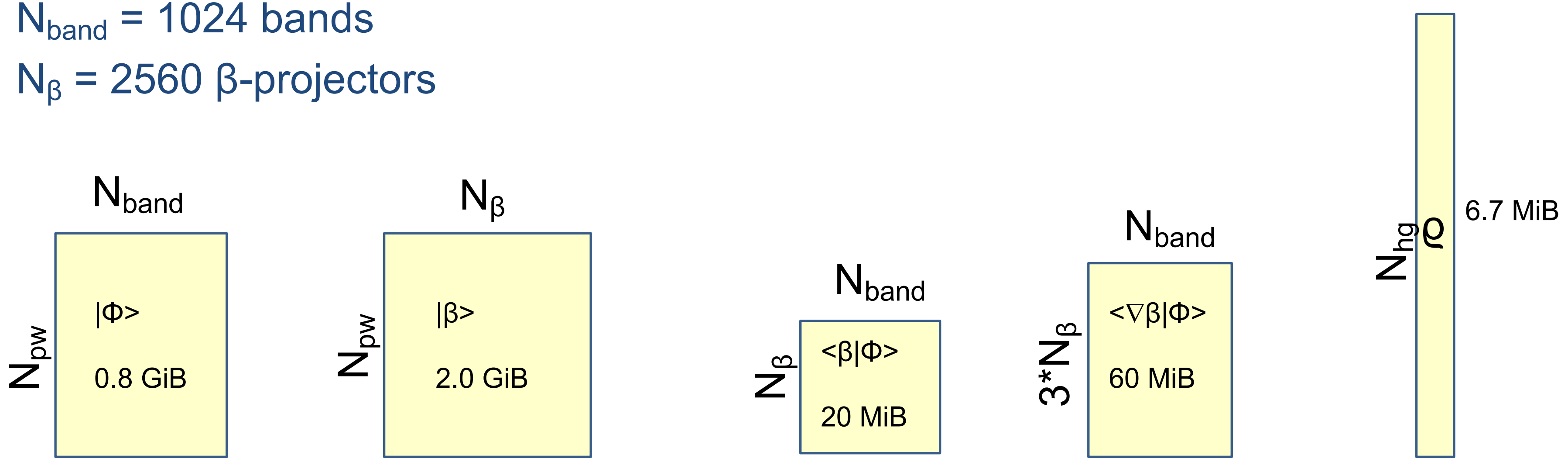
How Can We Improve the US-PP Code Path?

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

Data Structures

256 H_2O molecules, 25 Ry wave-function cutoff, 100 Ry charge-density cutoff, 19.73 $Å^3$

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



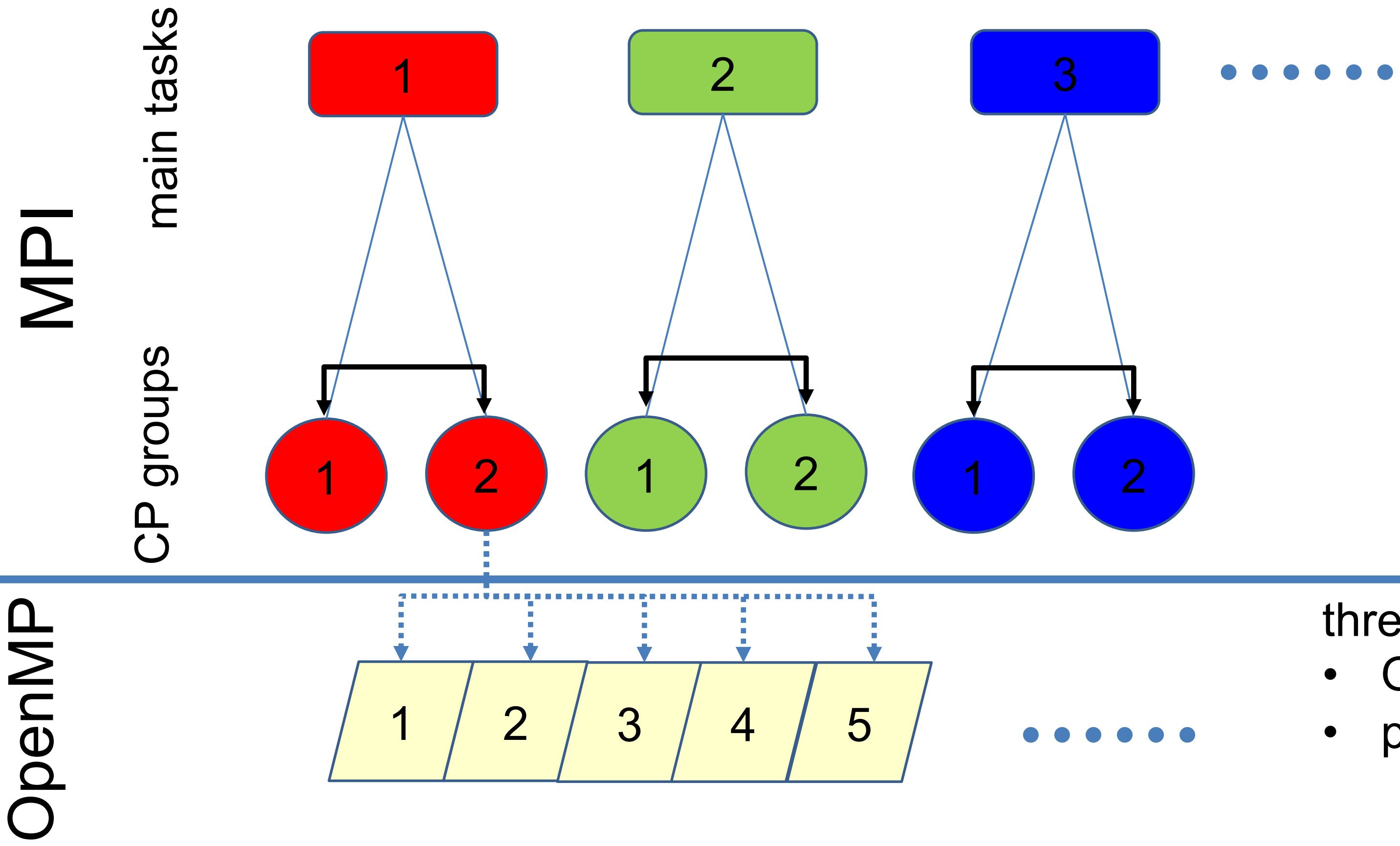
Parallelization Strategies in CPMD

1. Distribution of N_{pw} plain waves (basis functions) across main MPI tasks

2. Second level MPI parallelization with so called cp_groups:
   - 2 (or few) communicators with replicated data
   - parallelization over N_{band} electronic states
   - parallelization over N_{β} projectors
   - implemented only for selected routines along the main code path

3. Thread parallelization with OpenMP and threaded performance libraries
   - efficient only within NUMA domains.
   - implemented only for selected routines along the main code path

Parallelization Layers of CPMD



distribution of

- pw coefficients
- FFT grid planes

work sharing:

- same data set within a group
- synchronization necessary

thread level

- OpenMP
- performance libraries

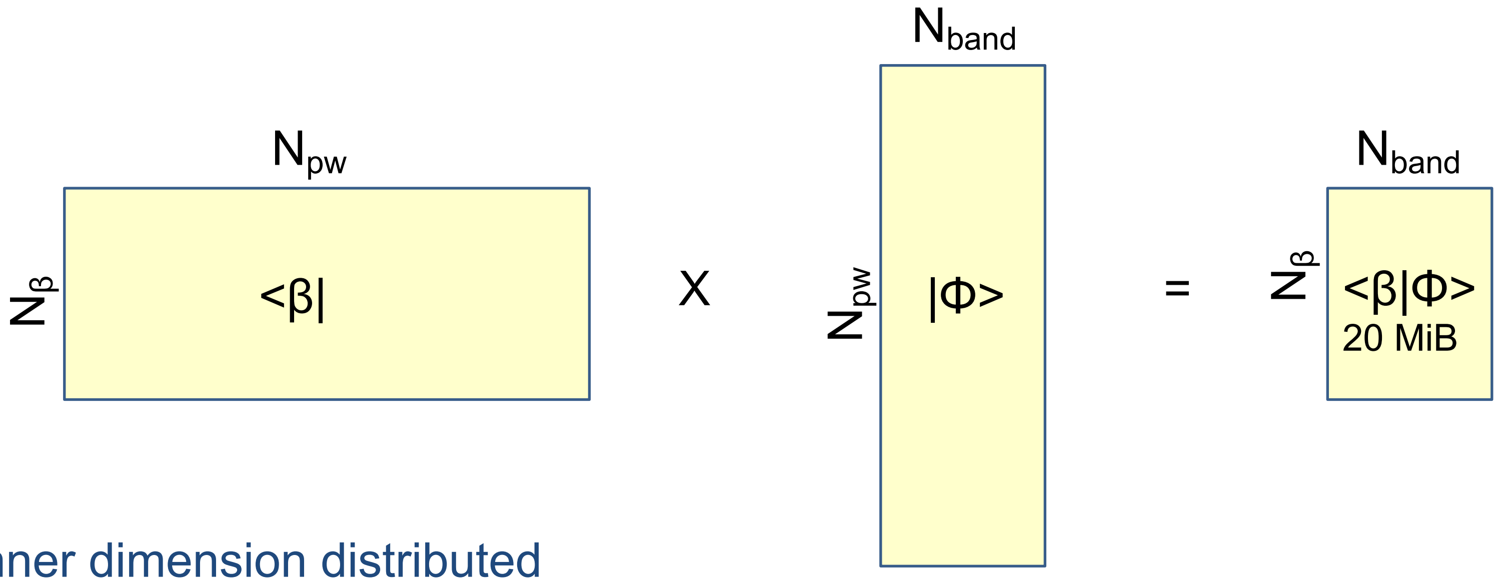
Distribution of Plane Wave Coefficients

1. **Distributed Matrix Matrix Multiplication ($<\beta|\Phi>$ and $< \nabla\beta|\Phi>$)**

2. Distributed 3D-FFT transformation ($|\Phi>_G \rightarrow |\Phi>_R$)

Distributed Matrix Matrix Multiplication

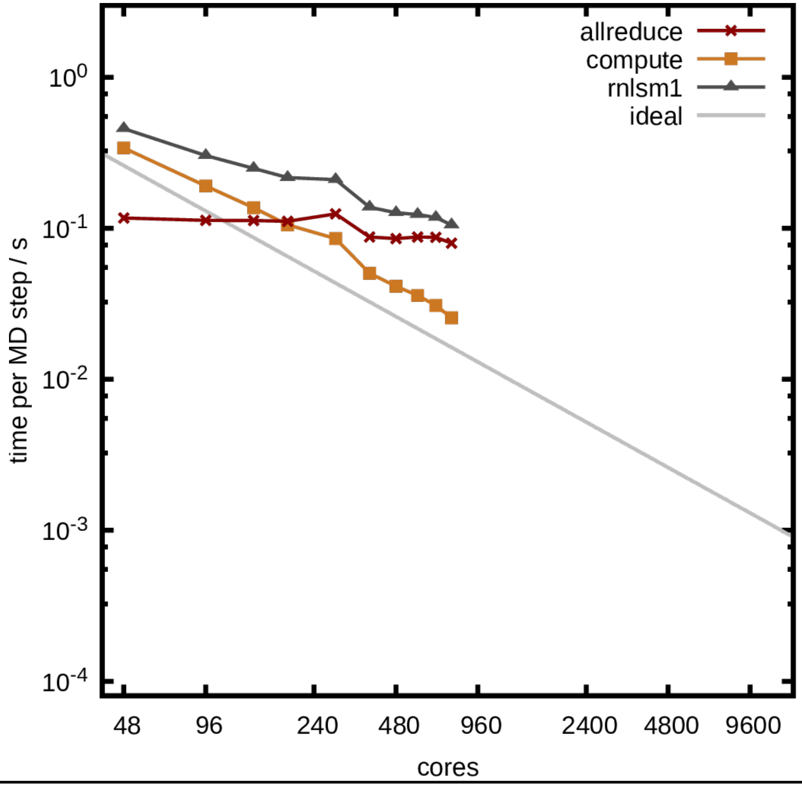
Calculation of <β|Φ>



- Inner dimension distributed
- <β|Φ> is replicated at each MPI task!
- Local DGEMMs + MPI_allreduce of 20 MiB

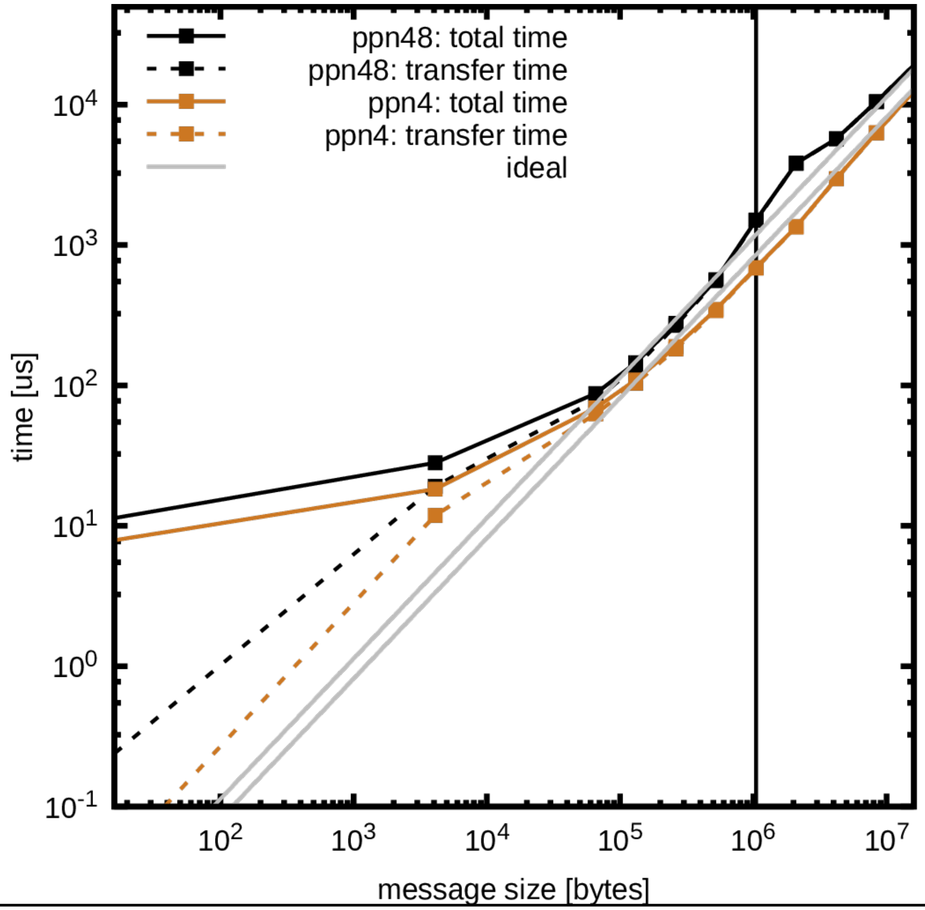
Calculation of <β|Φ>

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: 0.080 s/MD
  - Compute: 0.026 s/MD
- MPI_allreduce could benefit from larger message size
- DGEMMs should be kept as big as possible (and as quadratic as possible)

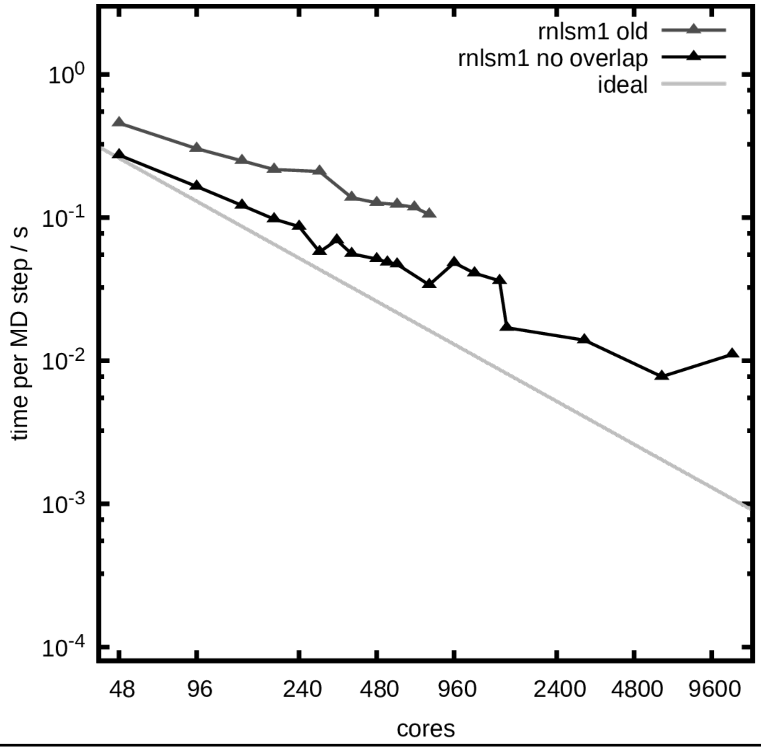
MPI Allreduce Performance @ 16 Nodes



- Almost no benefit from using fewer MPI ranks
- Allreduce size should be >512 KiB

Distributed Matrix Matrix Multiplication (<β|Φ>)

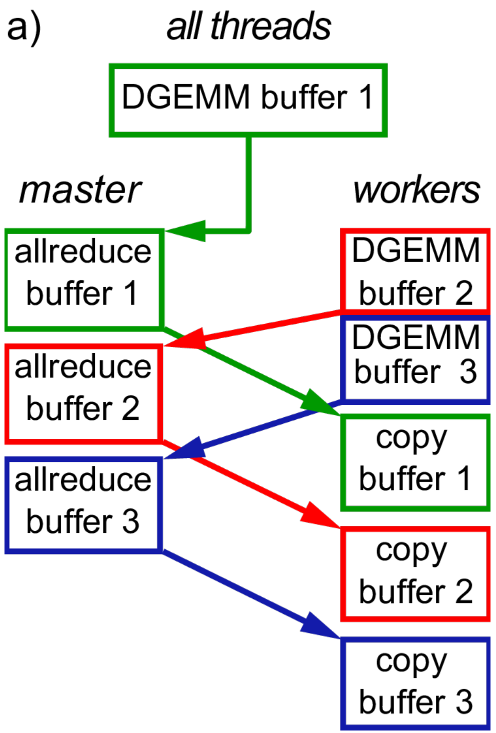
Idea 1: Use a single DGEMM + MPI_allreduce (all species all projectors)



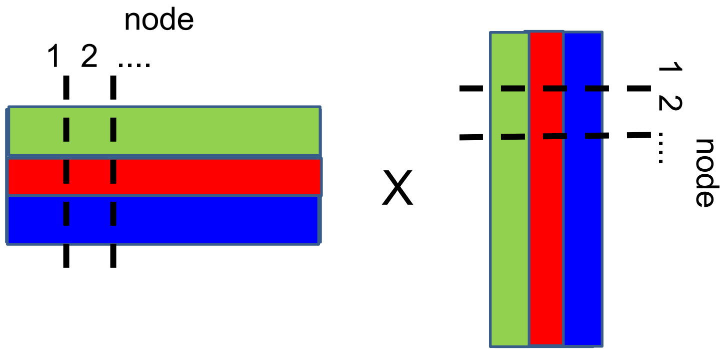
- Number of MPI tasks / OpenMP threads according to overall best performance!
- Total time at 16 nodes: 0.034 s/MD step
- Distribute β-projectors across cp_groups
- cp_groups active at >= 1536 cores
- cp_groups overhead not shown!

Distributed Matrix Matrix Multiplication (<β|Φ>)

Idea 2: Overlap of communication and computation



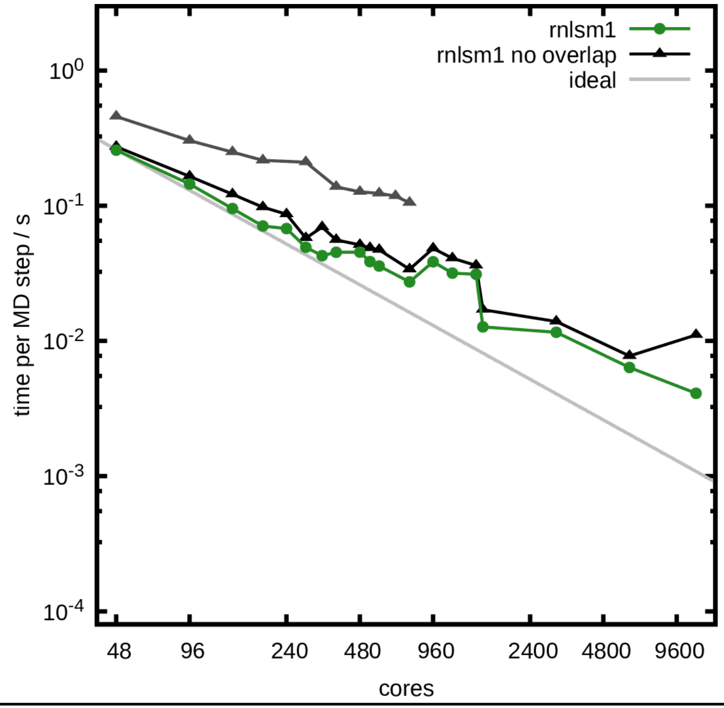
- Split DGEMM into few smaller parts
- OpenMP master thread used for communication
- OpenMP threads 2:n for remaining DGEMMs (nested OpenMP parallelism!)



Distributed Matrix Matrix Multiplication (<β|Φ>)

Idea 1: Use a single DGEMM + MPI_allreduce (all species all projectors)

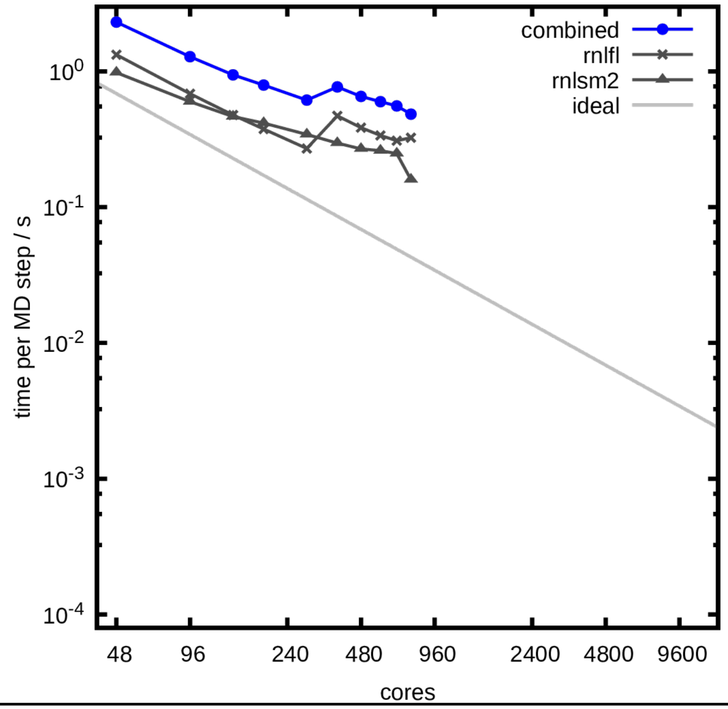
Idea 2: Overlap of communication and computation



- Total time at 16 nodes: 0.034 s/MD step
- Total time (overlap) at 16 nodes: 0.027 s/MD step
- Speedup at 16 nodes (old vs new + overlap): 3.9

Distributed Matrix Matrix Multiplication (< ∇β|Φ>)

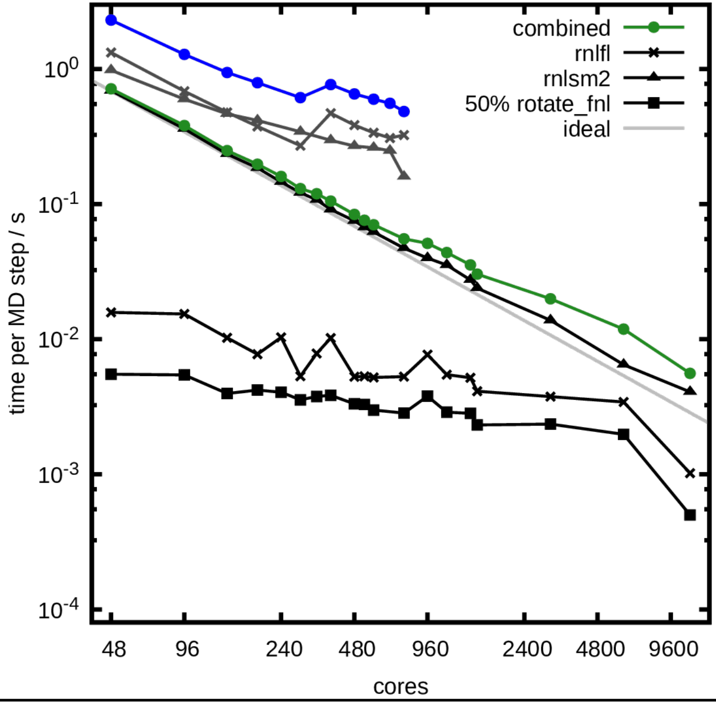
Idea 1: apply same optimizations as for <β|Φ>
+ optimization of rnlfl (hidden DGEMM (<β|Φ> x < Φ|H|Φ>)



- No OpenMP inside rnlfl
- Only rnlfl needs <∇β|Φ> (ionic forces)

Distributed Matrix Matrix Multiplication (< ∇β|Φ>)

Idea 1: apply same optimizations as for <β|Φ>
+ optimization of rnlfl (hidden DGEMM (<β|Φ> x < Φ|H|Φ>)
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 ($<\beta|\Phi>$ and $< \nabla\beta|\Phi>$)

2. **Distributed 3D-FFT transformation ($|\Phi>_G \rightarrow |\Phi>_R$)**

Distributed 3D-FFT - Parallelization

- 3D-FFT for each of the N_{band} electronic states (~ 120^3 grid each)
  N_{pw} plane wave coefficients distributed over the MPI task

- Distribute planes in real space
  scaling limited to number of planes, here 120 MPI tasks
  48 cores per node @ LRZ, 128 cores per node @ HLRS

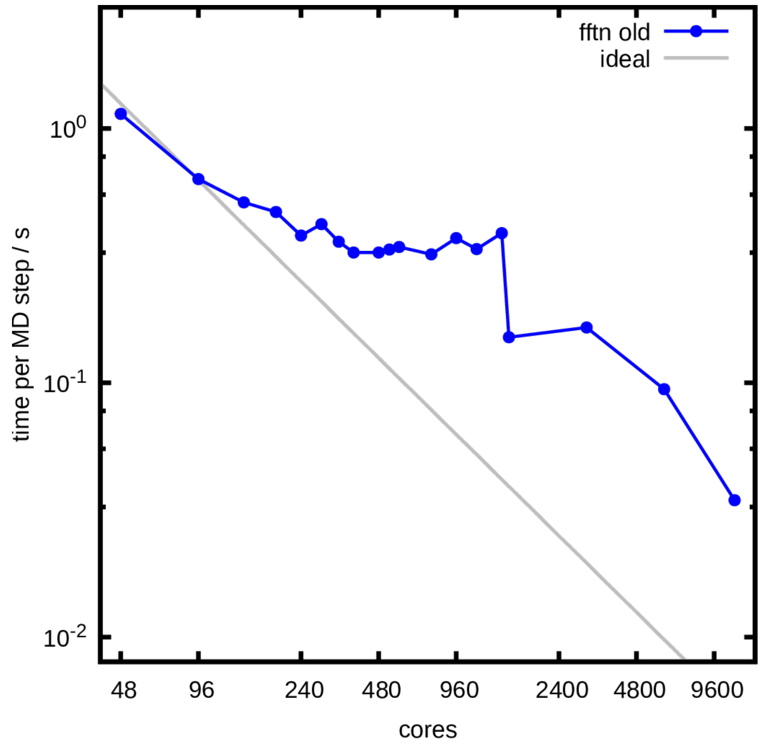
- Add more resources to a single MPI task for the actual 1D-FFT computations:
  hybrid parallelization (MPI + X, X = OpenMP, accelerators, …)

- cp_group parallelization:
  distribute electronic states among cp_groups
  data replication + synchronization

Distributed 3D-FFT MPI + OpenMP

Already implemented in CPMD

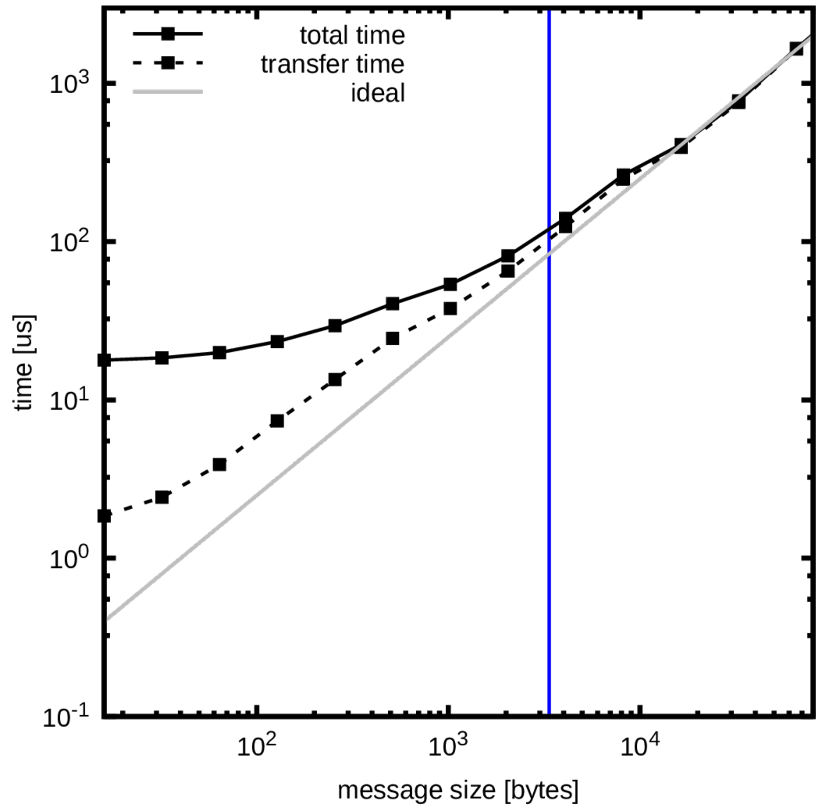
Performance of new US-PP implementation with old 3D-FFT routines



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

MPI Alltoall Performance

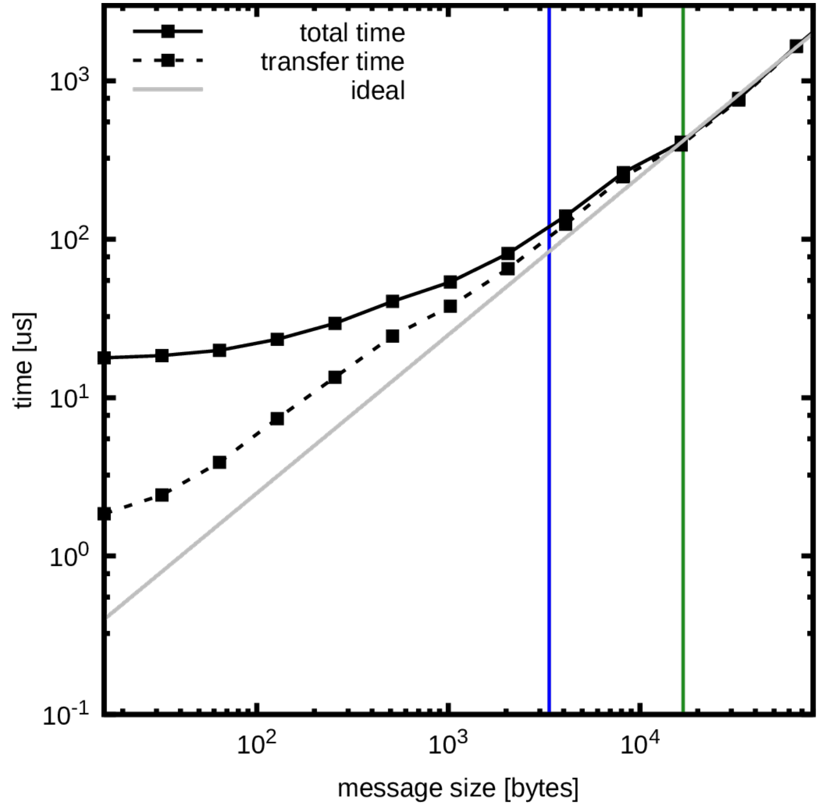
5 nodes, 8 MPI tasks per node



- All to all latency bound!
- FFT All-to-all message size 3360 bytes (3 planes x 70 rays)
- Message size will decrease with increasing MPI tasks

MPI Alltoall Performance

Idea 1: Combine A2A communication



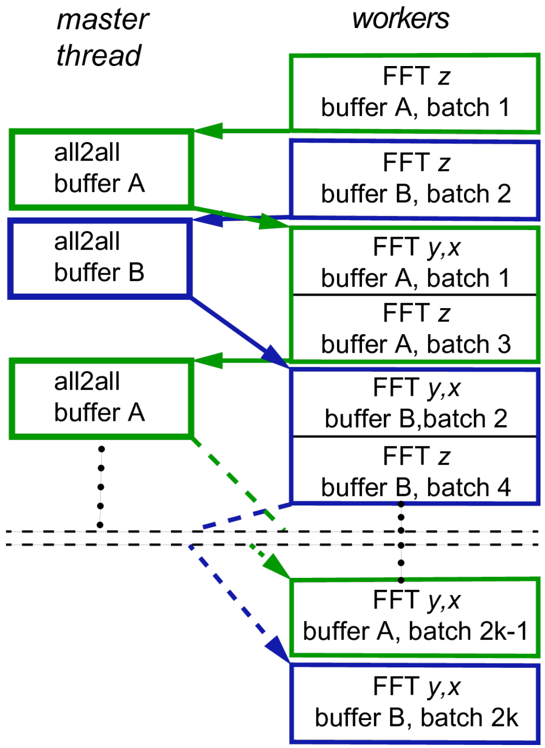
- Combine several all-to-all calls → pack/unpack state information

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

- total time decreases up to sweet spot (5 states)
- larger effect for higher node count expected

MPI Alltoall Performance

Idea 2: Work on two batches to hide communication

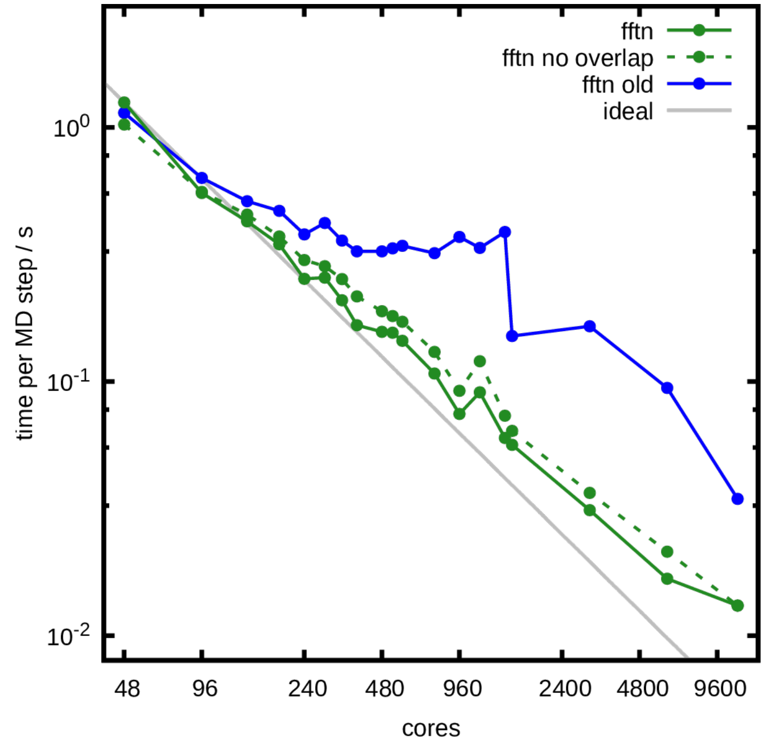


master thread
workers
FFT *z* buffer A, batch 1
all2all buffer A
FFT *z* buffer B, batch 2
all2all buffer B
FFT *y,x* buffer A, batch 1
FFT *z* buffer A, batch 3
all2all buffer A
FFT *y,x* buffer B,batch 2
FFT *z* buffer B, batch 4
FFT *y,x* buffer A, batch 2k-1
FFT *y,x* buffer B, batch 2k

States A2A	Msg size [bytes]	Time [ms]	Time [ms] (overlap)
1	3360	514	504
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

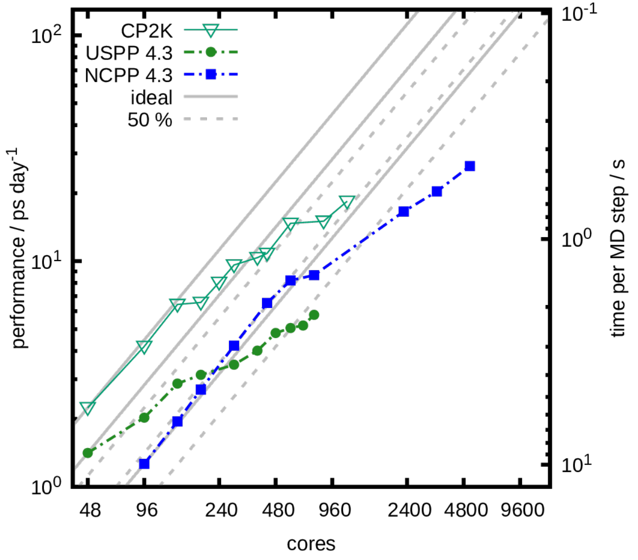
Performance of new US-PP implementation with new 3D-FFT routines



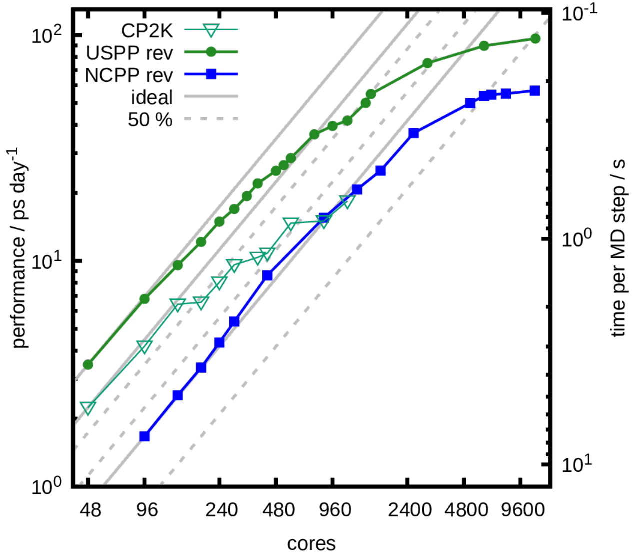
- Number of MPI tasks / OpenMP threads according to overall best performance!
- Scaling of FFT in hybrid parallelization: > 4800 cores
- No performance benefit of using cp_group parallelization at 1536!

CPMD US-PP > 15,000 Codes Lines Changed

old



new



NC-PP

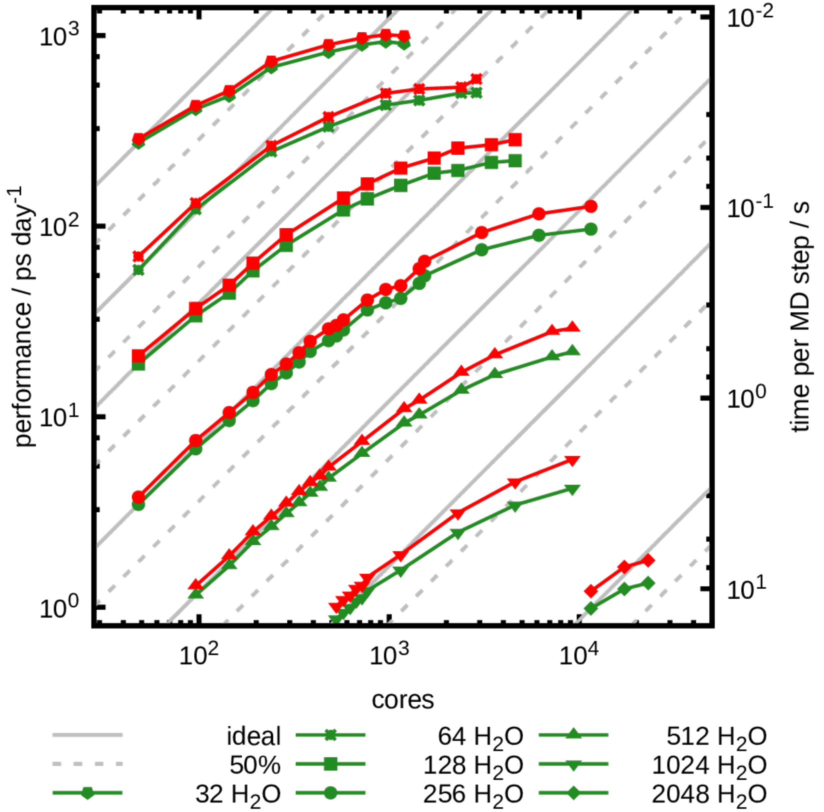
- 1.2x-1.3x speedup
- 50ps per day
- Outperforms CP2K

US-PP

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

Strong Scaling Benchmark

32 – 2048 H20 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 H2O molecules, 16384 electrons! Code not even optimized

Take Home Messages: Node Level Optimization

- Check if you can map inner loops to BLAS calls.
  → let the performance library do the work for you
- Thread parallelization is included
- Overhead to locally rearrange data (e.g. matrix buildup) is often acceptable (for BLAS level 2 or 3)
- Check, if you can combine smaller matrices to a larger one.
  → better vectorization and less overhead
- Large BLAS operations ready for offloading
- Variants: Check batched BLAS (MKL), libxsmm (small matrices)

Take Home Messages: Communication

- Check ,if you are running into latency bound regimes on scale out
- Combine communication calls to stay in bandwidth bound regime
- Check, if communication can be avoided
  - Is the information really necessary (in all cases)?
  - Is MPI_allreduce needed or is MPI_reduce sufficient?
  - Is there a faster node local algorithm?
- Overlapping communication and computation can give you the last bleeding edge. (max. gain is a factor of 2)

Read more ...

- Tobias Klöffel, Gerald Mathias, Bernd Meyer,
Integrating state of the art compute, communication, and autotuning strategies to multiply the performance of ab initio molecular dynamics on massively parallel multi-core supercomputers, Computer Physics Communications, Volume 260, 2021, 107745, https://doi.org/10.1016/j.cpc.2020.107745.
(https://www.sciencedirect.com/science/article/pii/S0010465520303684)

- Integrating State of the Art Compute, Communication, and Autotuning Strategies to Multiply the Performance of the Application Programm CPMD for Ab Initio Molecular Dynamics Simulations
https://arxiv.org/abs/2003.08477

Acknowledgements

Tobias Klöffel

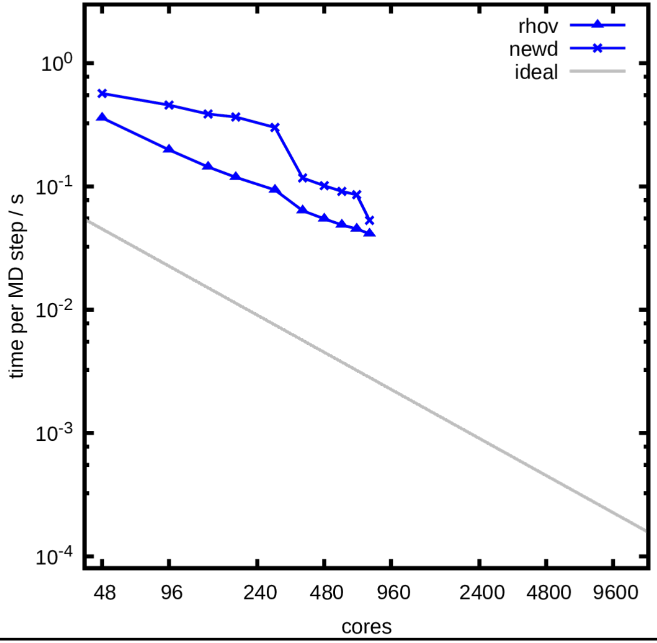
Bernd Meyer

Georg Hager

Backmatter

Calculation of Augmentation Charges & New D

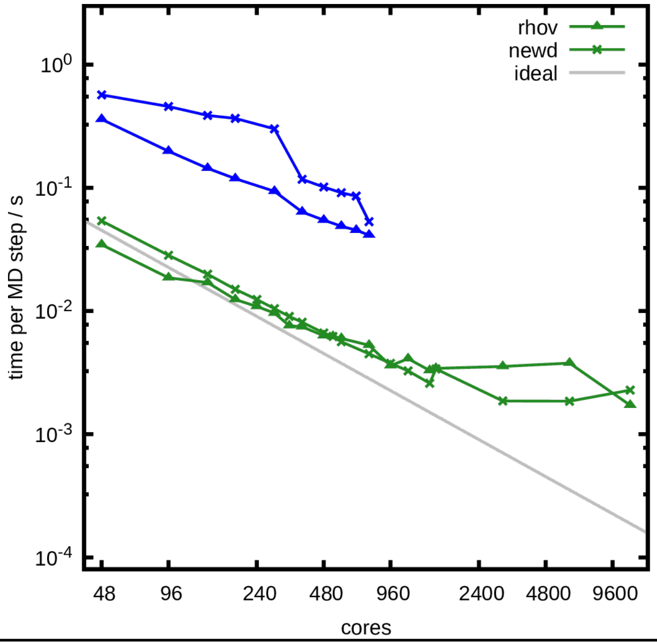
- Rhov: calculation of augmentation charges
  Newd: calculation of D, Q and ionic forces
- Recalculation of Q-function at every call in both routines
- Calculation of becsum in both routines, differently implemented, not parallelized



- 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)
  MPI summation of sparse array deeq

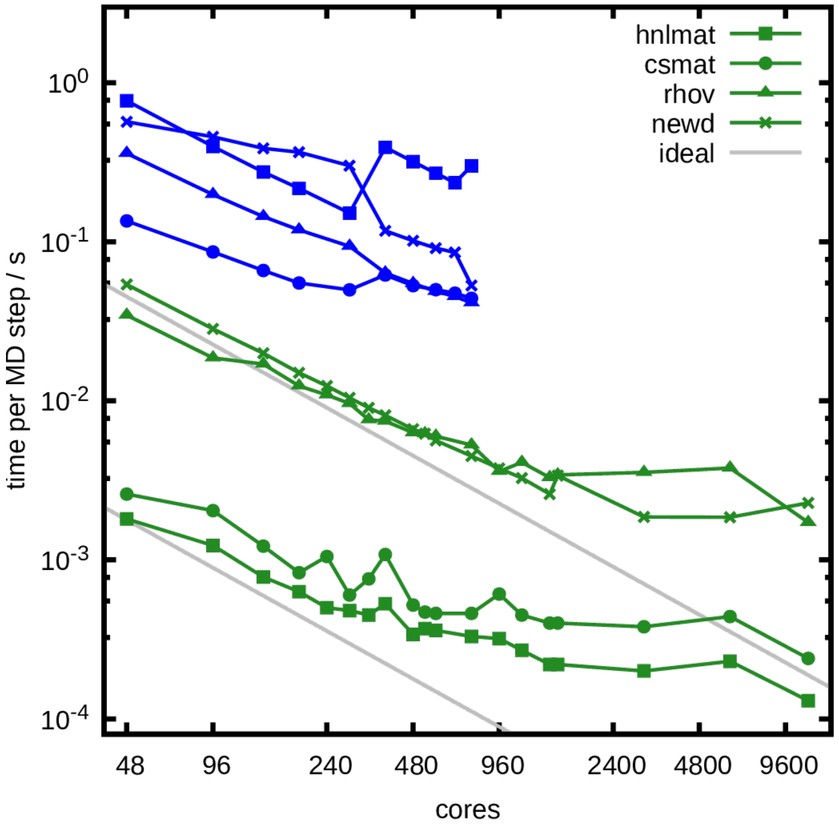
Calculation of Augmentation Charges & New D

- Save 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

Hnlmat & Csmat



- Sophisticated loopnest (7) -> loopnest (6) + DGEMM
- Reworked MPI parallelization