CUG2016

London, UK, May 8 - 12

On Enhancing 3D-FFT Performance in VASP

Florian Wende

Martijn Marsman, Thomas Steinke

Zuse Institute Berlin

Outline

Many-core Optimizations in VASP

- From MPI to MPI + (OpenMP) Threading
- Multi-threaded FFT: MKL, FFTW (LibSci)
- 3D-FFT in VASP

How to improve FFT computation in VASP?

- FFTLIB: C++ template library to intercept FFTW calls
  - ❑ Plan Reuse
  - ❑ Composed FFT computation
  - ❑ …
- Some performance numbers

Many-core Optimizations in VASP

Many-core Optimizations in VASP

VASP – Vienna Ab-initio Simulation Package

- Electronic structure code
- MPI-only (latest official release)
- Implements DFT: many FFT computations

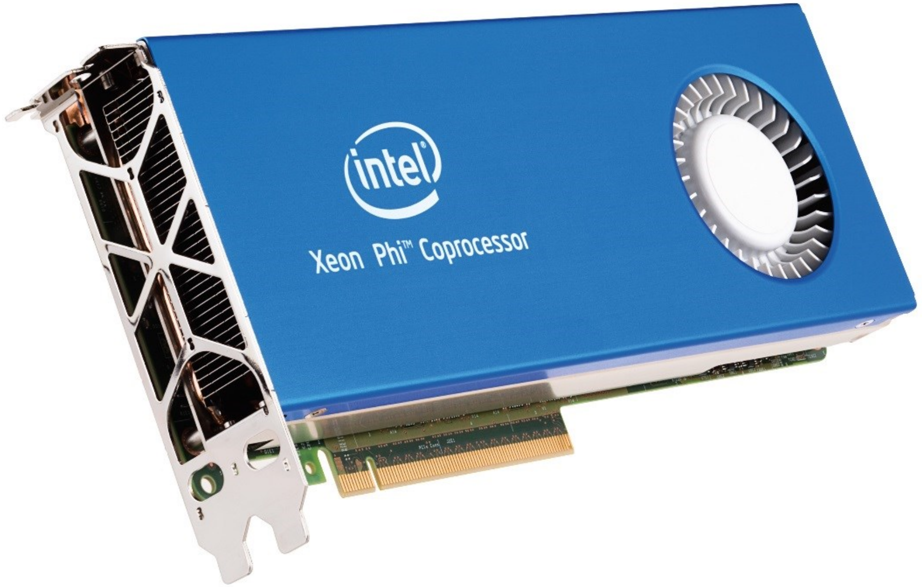
Optimization approach

- Introduce Threading
- Optimize code sections for SIMD (talk at CUG2015)
- Improve library integration/usage: FFT, BLAS/Scalapack, …

Many-core Optimizations in VASP

Many-core processor = lots of (lightweigth) compute cores on a chip

- Intel Xeon Phi KNC/KNL: 60+ cores
- Highly parallel computation
- MPI-only will not work in the majority of cases



MPI + (OpenMP) Threading

- KNL: 4, 8, 16 MPI ranks + lots of (OpenMP) threads
- User function part: code instrumentation via OpenMP compiler directives
- Library functions: multi-threaded context or call to multi-threaded library

Many-core Optimizations in VASP

Current state of VASP* optimization

- MPI + OpenMP: almost fully adapted code base
- SIMD: OpenMP 4.x directives
- Multi-threaded FFT computation
  - 3D-FFT where possible: it is really fast!
  - “Ball ↔ Cube” FFT optimization: composed 1D+1D+1D FFT

*This VASP version is not yet officially available! Will be coming soon ☺

Many-core Optimizations in VASP

3D-FFT in VASP

- FFTW library calls: Intel MKL can be used through its FFTW interface
- Calling scheme in VASP:

```
// create plan
p=fftw_plan_xxx(…)

// execute
fftw_execute(p)

// destroy plan
fftw_destroy_plan(p)
```

This happens again and again

Many-core Optimizations in VASP

3D-FFT in VASP

- Program performance (PdO2: Paladiumdioxid on Paladium surface)

  24 MPI ranks on 4 Cray XC-40 compute nodes (Haswell), T=1,4 threads per rank
  MKL 11.3.2, FFTW (from GitHub and Cray LibSci)

	Setup: PdO2					
	MKL 11.3.2		FFTW		FFTW (LibSci)	
	T=1	T=4	T=1	T=4	T=1	T=4
Total	146.6s	78.0s	162.1s	122.5s	162.3s	121.9s
3D-FFT	23.4s	10.7s	38.2s	43.3s	38.6s	41.9s
+planner	1.0s	1.0s	10.0s	32.9s	9.8s	31.1s
+execute	22.4s	9.7s	28.2s	10.4s	28.8s	10.8s

Many-core Optimizations in VASP

3D-FFT in VASP

- Program performance (PdO2: Paladiumdioxid on Paladium surface)

  24 MPI ranks on 4 Cray XC-40 compute nodes (Haswell), T=1,4 threads per rank
  MKL 11.3.2, FFTW (from GitHub and Cray LibSci)

	Setup: PdO2					
	MKL 11.3.2		FFTW		FFTW (LibSci)	
	T=1	T=4	T=1	T=4	T=1	T=4
Total	146.6s	78.0s	162.1s	122.5s	162.3s	121.9s
3D-FFT	23.4s	10.7s	38.2s	43.3s	38.6s	41.9s
+ planner	1.0s	1.0s	10.0s	32.9s	9.8s	31.1s
+ execute	22.4s	9.7s	28.2s	10.4s	28.8s	10.8s

A lot of time is spent in the planner phase!

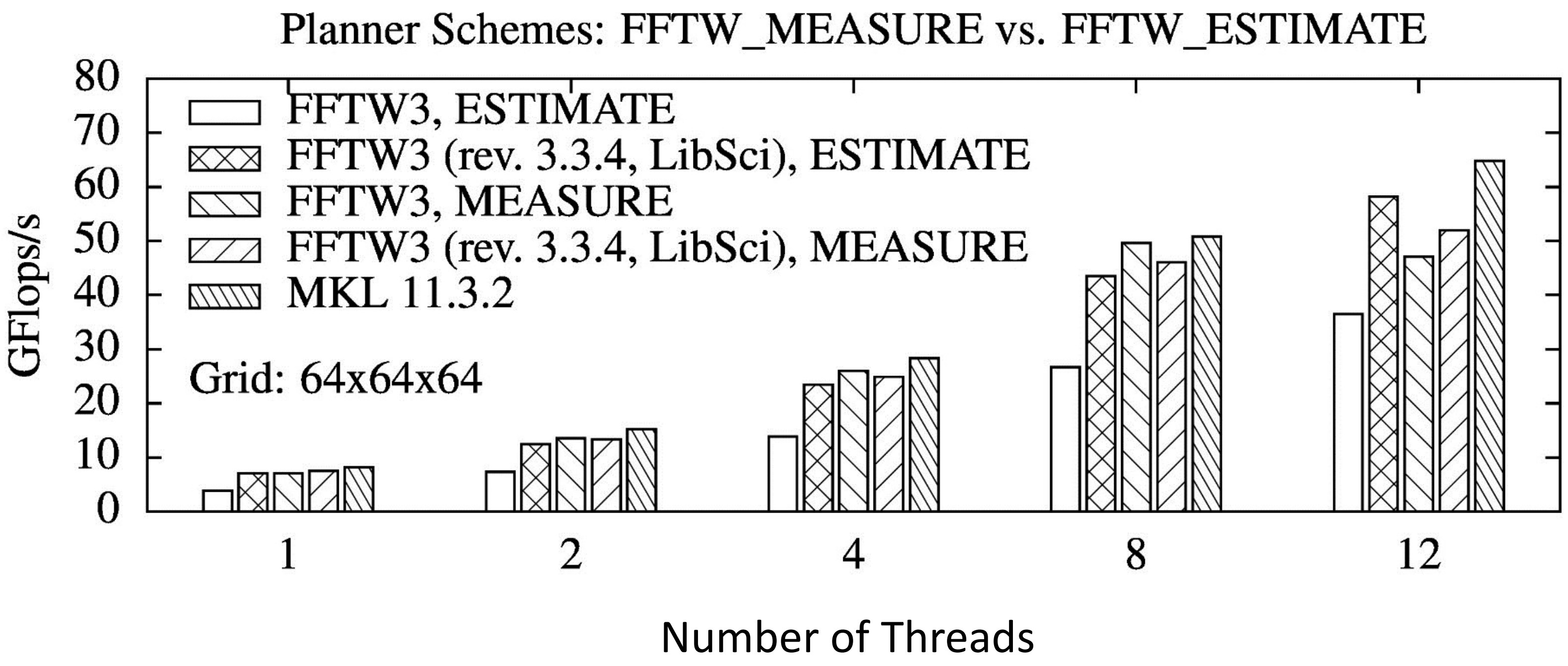
Many-core Optimizations in VASP

3D-FFT in VASP

- FFTW provides different planner schemes: ESTIMATE, MEASURE, …

  ESTIMATE – cheap

  MEASURE – expensive: kind of online-autotuning on different FFT algorithms



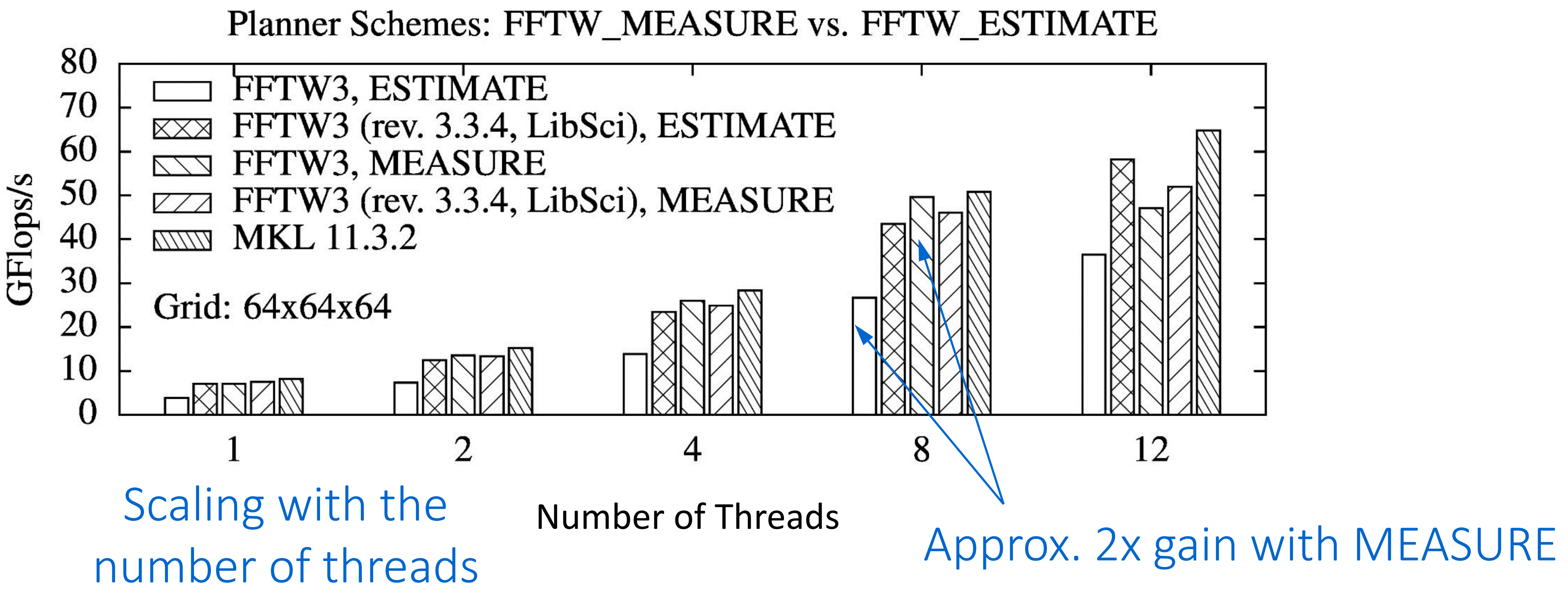
Many-core Optimizations in VASP

3D-FFT in VASP

- FFTW provides different planner schemes: ESTIMATE, MEASURE, …

  ESTIMATE – cheap

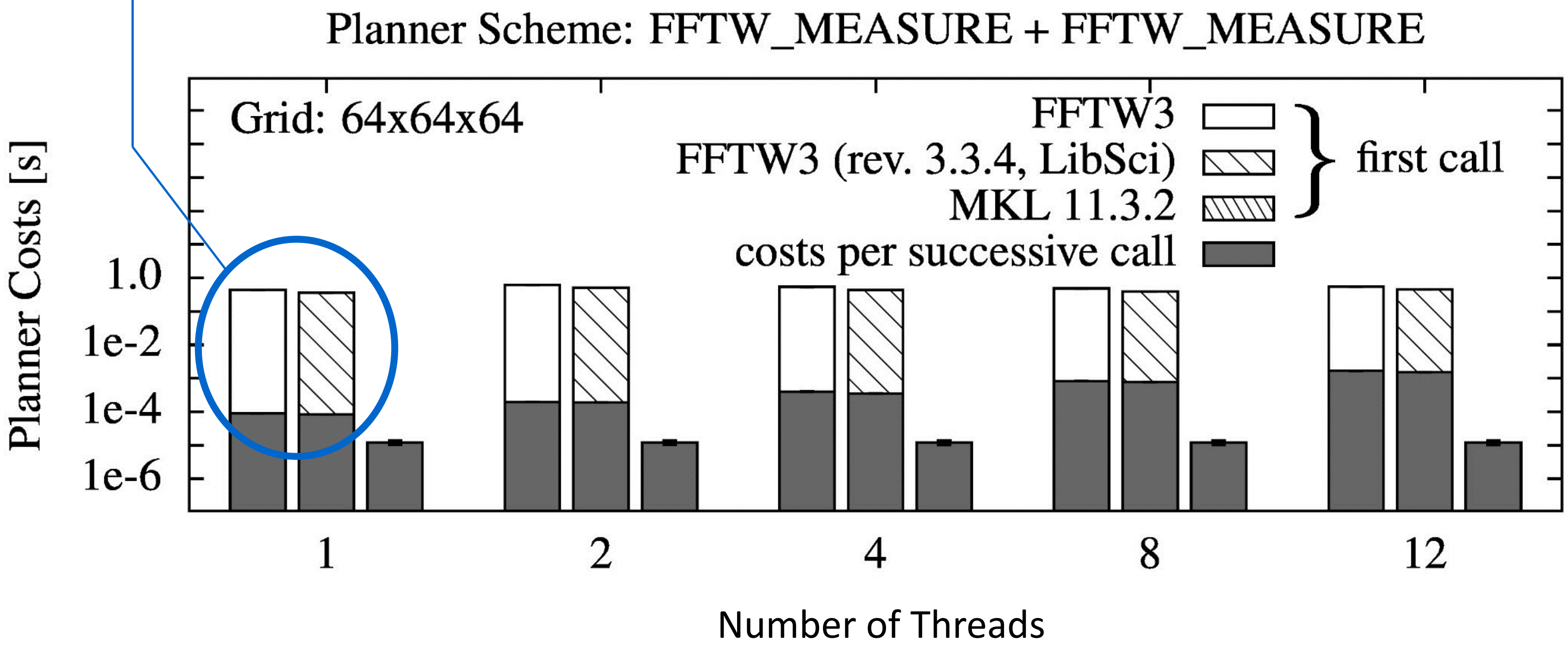
  MEASURE – expensive: kind of online-autotuning on different FFT algorithms



Many-core Optimizations in VASP

3D-FFT in VASP

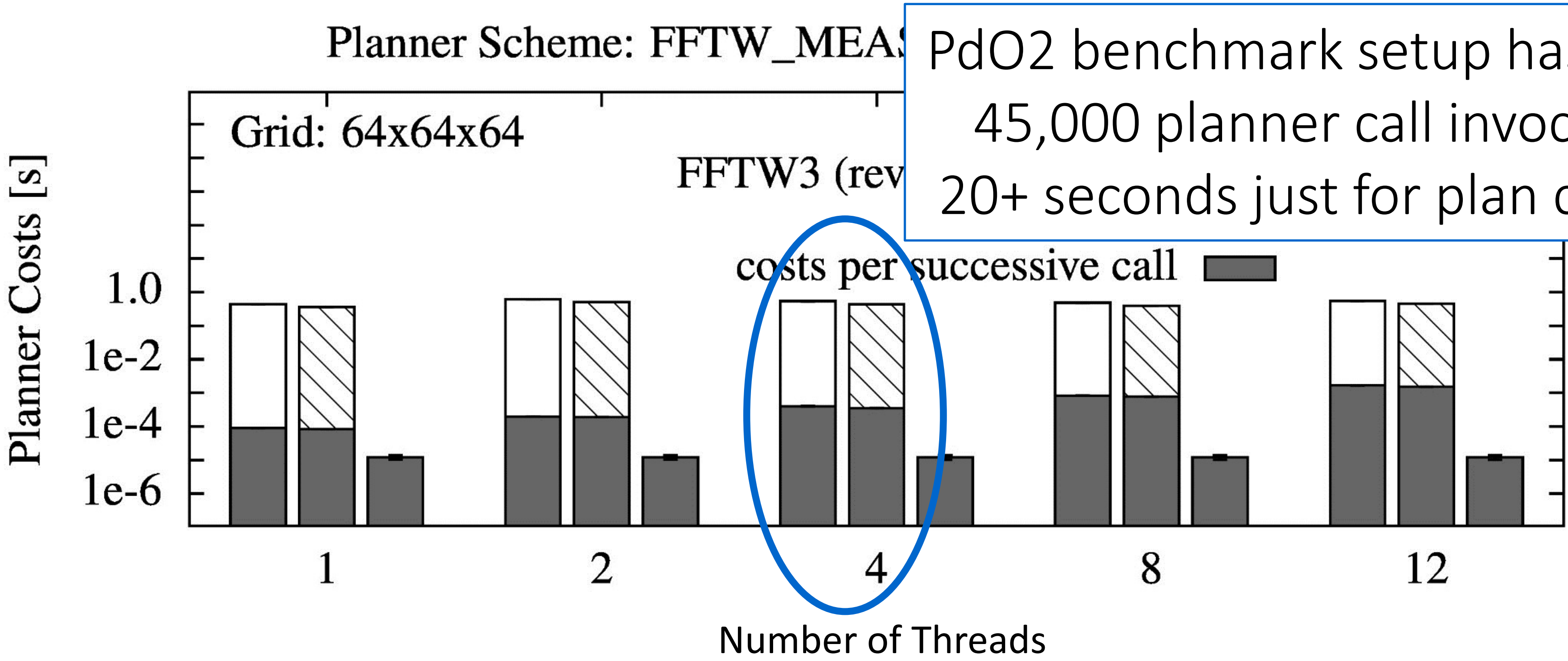
- Costs for planning with MEASURE and using the FFTW Wisdom feature for faster plan creation



Many-core Optimizations in VASP

3D-FFT in VASP

- Costs for planning with MEASURE and with FFTW Wisdom feature for faster plan creation



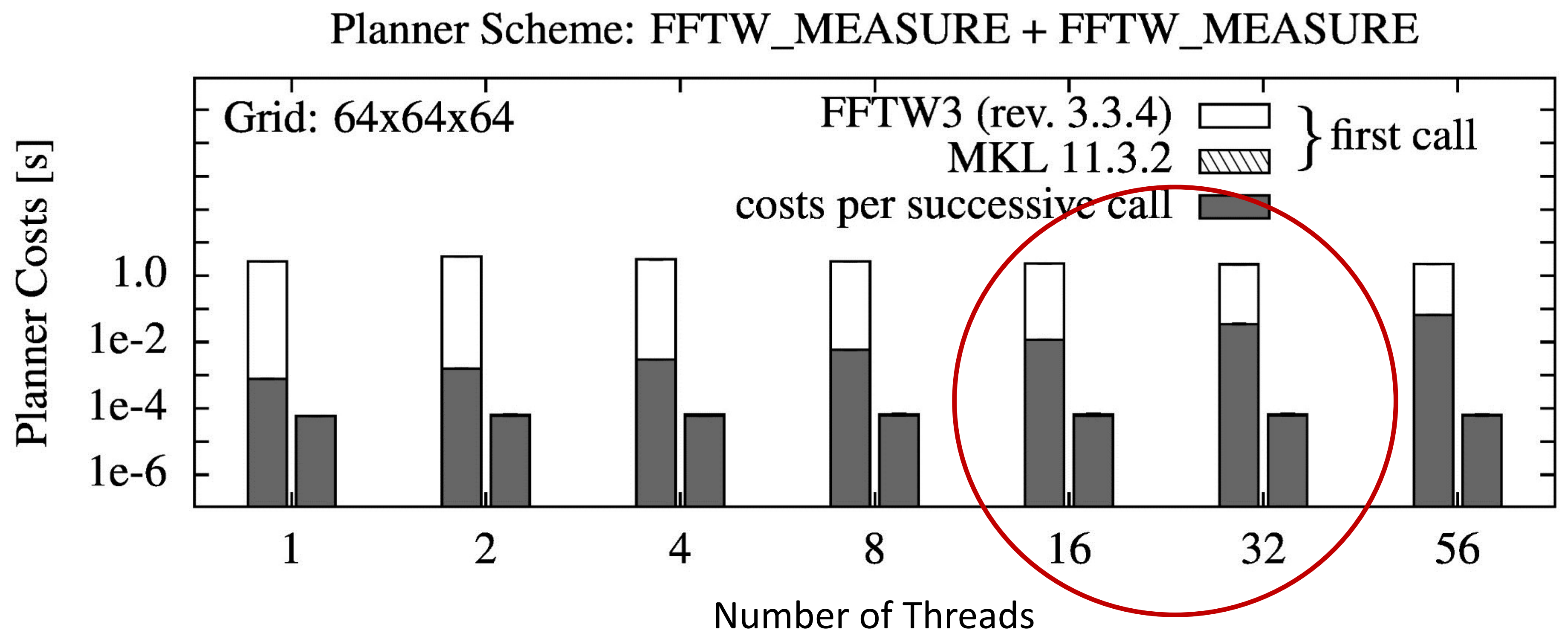
PdO2 benchmark setup has approx. 45,000 planner call invocations: 20+ seconds just for plan creation!

Many-core Optimizations in VASP

3D-FFT in VASP

- Costs for planning with MEASURE and with FFTW Wisdom feature for faster plan creation

Xeon Phi (KNC) Data



Many-core Optimizations in VASP

3D-FFT in VASP

- Costs for plan creation seems to become dominant with increasing number of threads to be used for the computation

Can we do better?

How to Improve FFT Computation in VASP?

How to Improve FFT Computation in VASP?

Main issue when using FFTW: plan creation

- Recommendation on FFTW webpage: “reuse plans as long as possible”
- Plan caching mechanism in the application?
  No, create a library for that purpose!

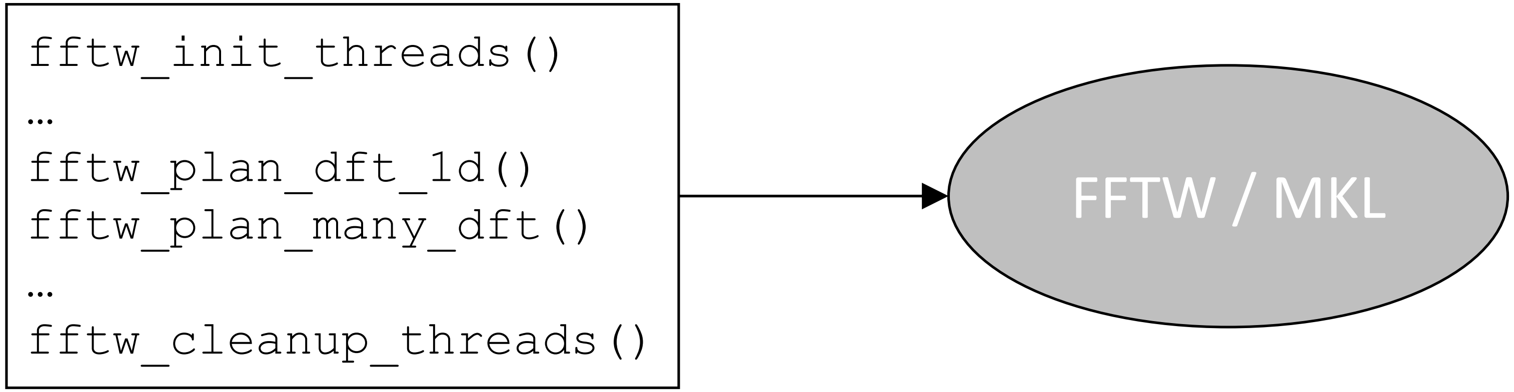
FFTLIB (will be hosted as open source library soon)

- C++ template library encapsulation FFTW and DFTI specifics
- Plan reuse: hash-map + cache
- Composed FFT
  - “Ball ↔ Cube” FFT optimization
  - Skip transpose operation(s)
  - High Bandwidth Memory (preparation for Intel’s Xeon Phi KNL)

How to Improve FFT Computation in VASP?

FFTLIB – Approach

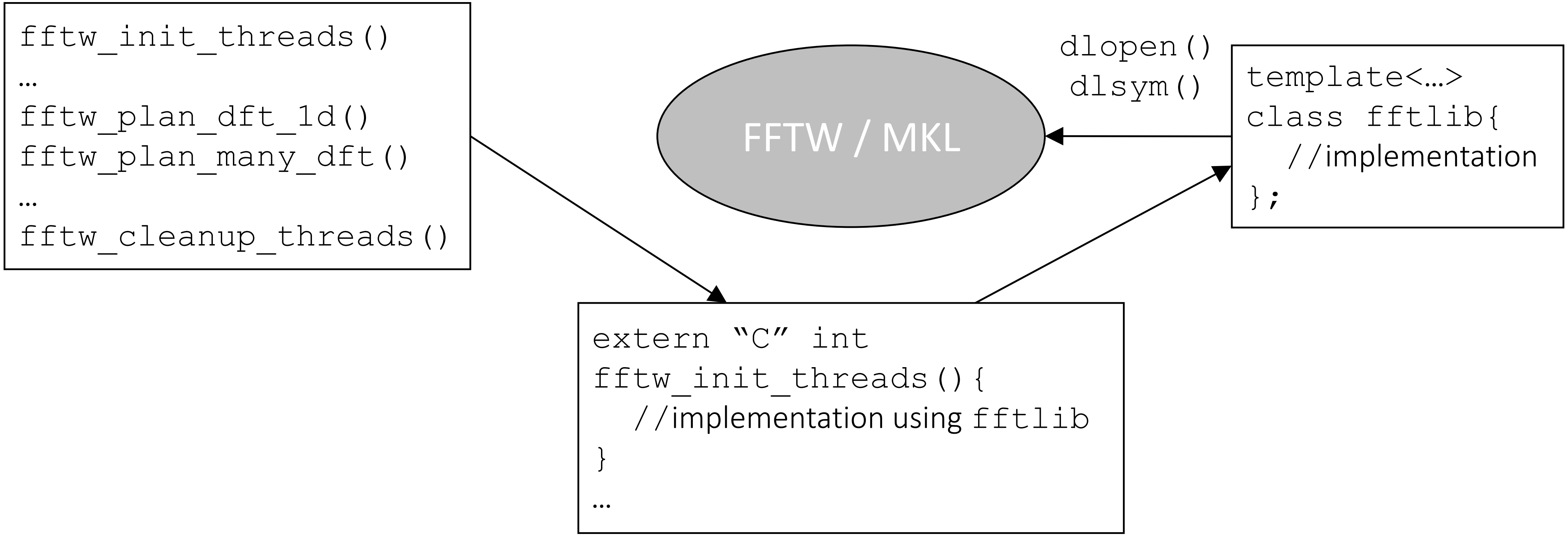
- Intercept FFTW calls + additional features



How to Improve FFT Computation in VASP?

FFTLIB – Approach

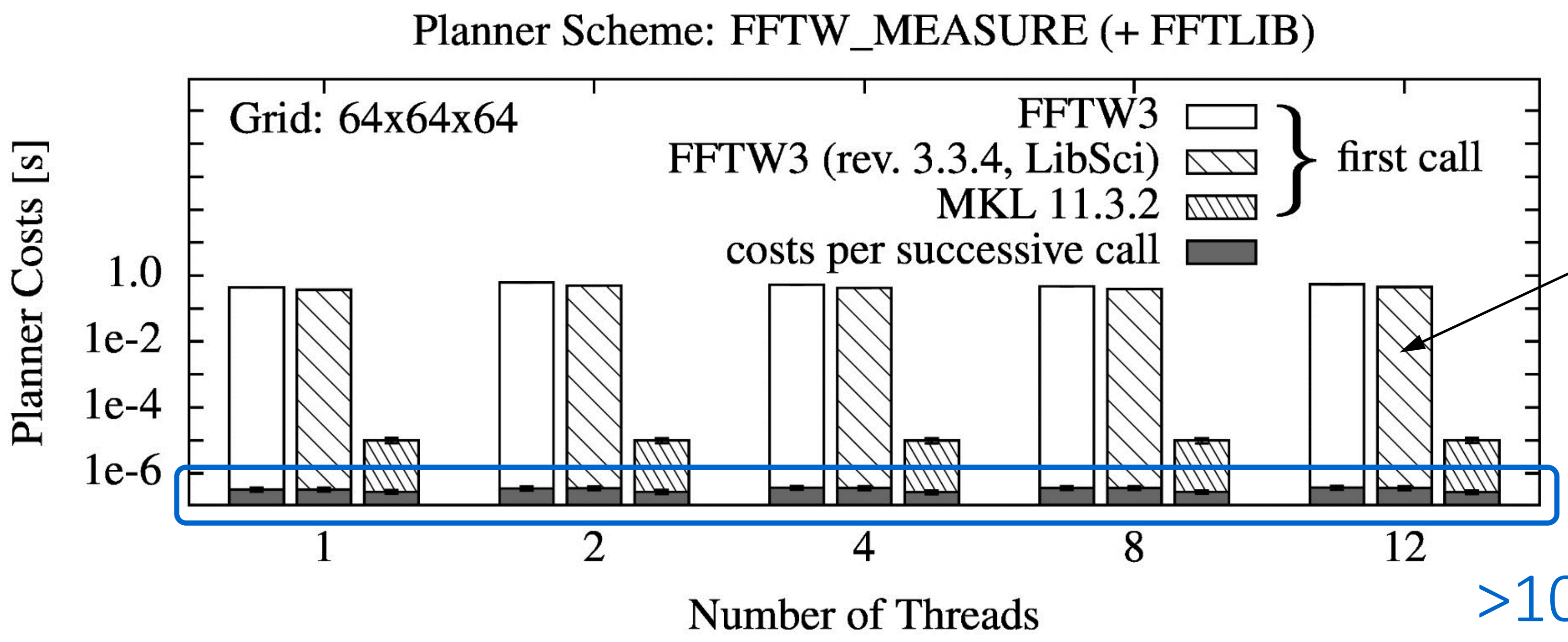
- Intercept FFTW calls + additional features



How to Improve FFT Computation in VASP?

FFTLIB – Plan Reuse

- Plans are stored permanently in a hash-map + cache
  - ❑ First planner call goes to FFTW / MKL for each geometry
  - ❑ Successive planner calls are served by FFTLIB



Remains the same

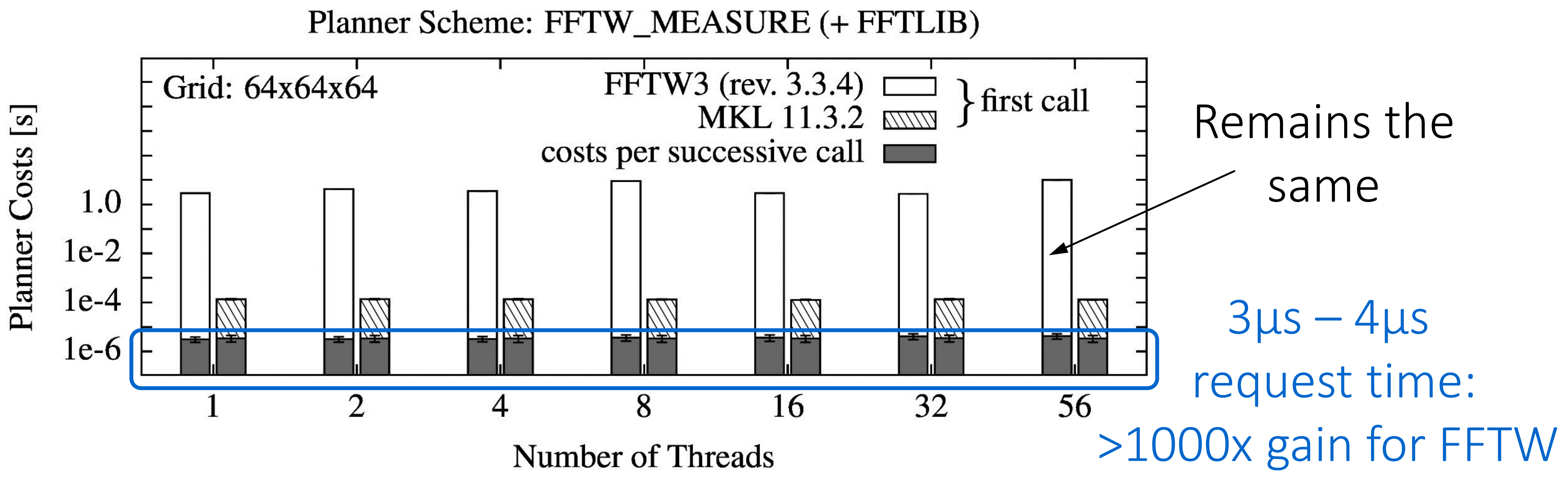
0.3μs – 0.4μs request time: >1000x gain for FFTW

How to Improve FFT Computation in VASP?

FFTLIB – Plan Reuse

- Plans are stored permanently in a hash-map + cache
  - First planner call goes to FFTW / MKL for each geometr
  - Successive planner calls are served by FFTLIB

Xeon Phi (KNC) Data



How to Improve FFT Computation in VASP?

3D-FFT in VASP with FFTLIB

- Program performance (PdO2: Paladiumdioxid on Paladium surface)

  24 MPI ranks on 4 Cray XC-40 compute nodes (Haswell), T=1,4 threads per rank
  MKL 11.3.2, FFTW (from GitHub and Cray LibSci)

	Setup: PdO2					
	MKL 11.3.2		FFTW		FFTW (LibSci)	
	T=1	T=4	T=1	T=4	T=1	T=4
Total	145.5s	84.8s	152.6s	86.5s	153.3s	90.0s
3D-FFT	23.4s	10.0s	29.0s	11.3s	29.6s	11.7s
+ planner	0.3s	0.3s	0.8s	0.9s	0.8s	0.9s
+ execute	22.4s	9.7s	28.2s	10.4s	28.8s	10.8s

Without FFTLIB:
Approx. 32 seconds (T=4) just for plan creation with FFTW

Only the initial planner costs contribute

How to Improve FFT Computation in VASP?

3D-FFT in VASP with FFTLIB

- Program performance (PdO2: Paladiumdioxid on Paladium surface)

  24 MPI ranks on 4 Cray XC-40 compute nodes (Haswell), T=1,4 threads per rank
  MKL 11.3.2, FFTW (from GitHub and Cray LibSci)

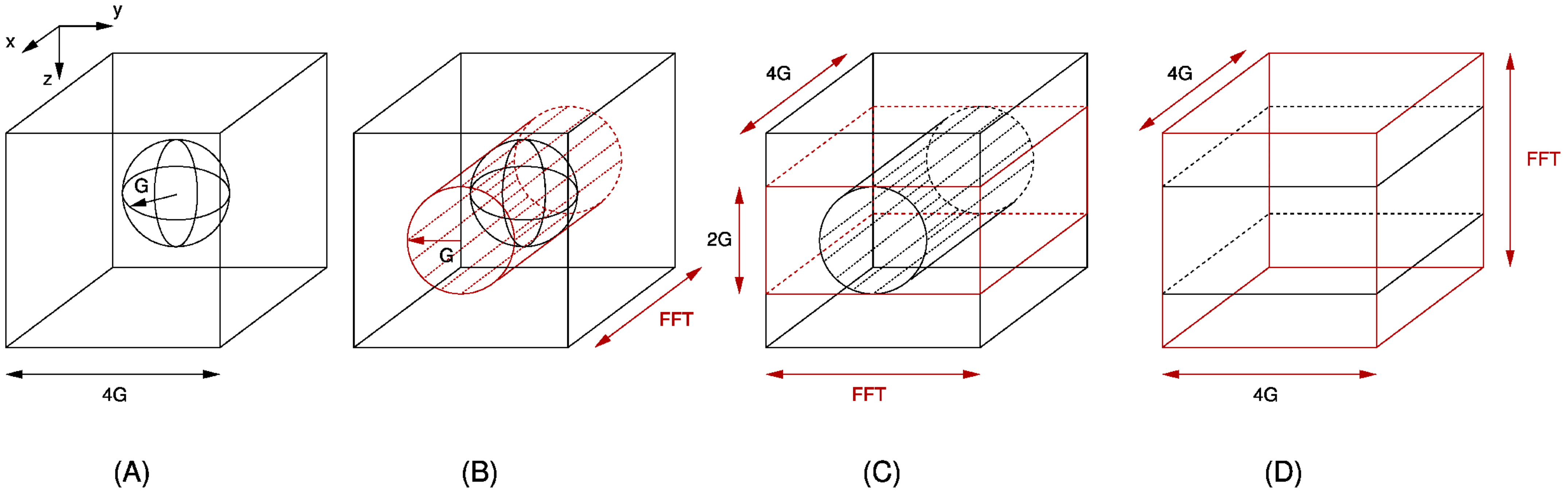
	Setup: PdO2					
	MKL 11.3.2		FFTW		FFTW (LibSci)	
	T=1	T=4	T=1	T=4	T=1	T=4
Total	[illegible] 1.01x	[illegible] 0.92x	[illegible] 1.06x	[illegible] 1.42x	[illegible] 1.06x	[illegible] 1.35x
3D-FFT	23.4s	10.0s	29.0s	11.3s	29.6s	11.7s
+ planner	0.3s	0.3s	0.8s	0.9s	0.8s	0.9s
+ execute	22.4s	9.7s	28.2s	10.4s	28.8s	10.8s

How to Improve FFT Computation in VASP?

3D-FFT with FFTLIB

- Composed FFT: "Ball ↔ Cube" FFT optimization (optional)
  - Reciprocal space vector *G* below a certain cutoff

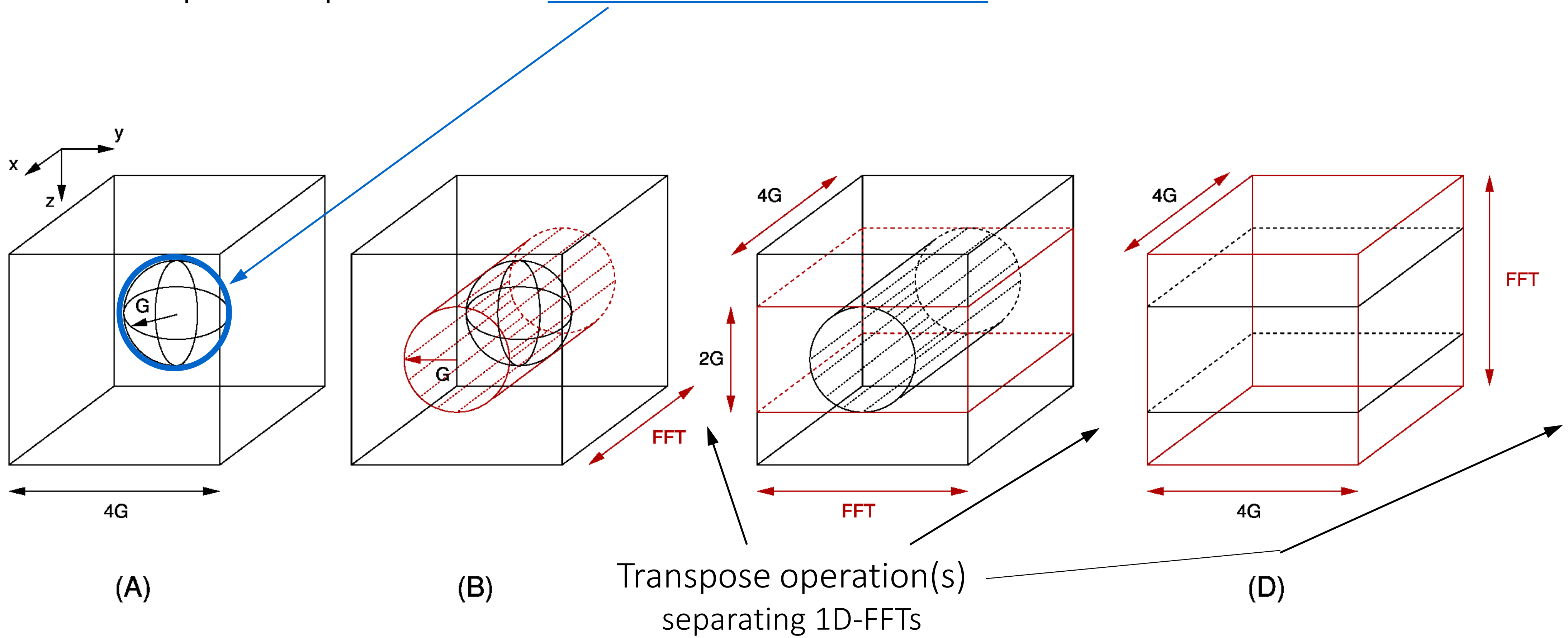
This is what VASP is doing right now



How to Improve FFT Computation in VASP?

3D-FFT with FFTLIB

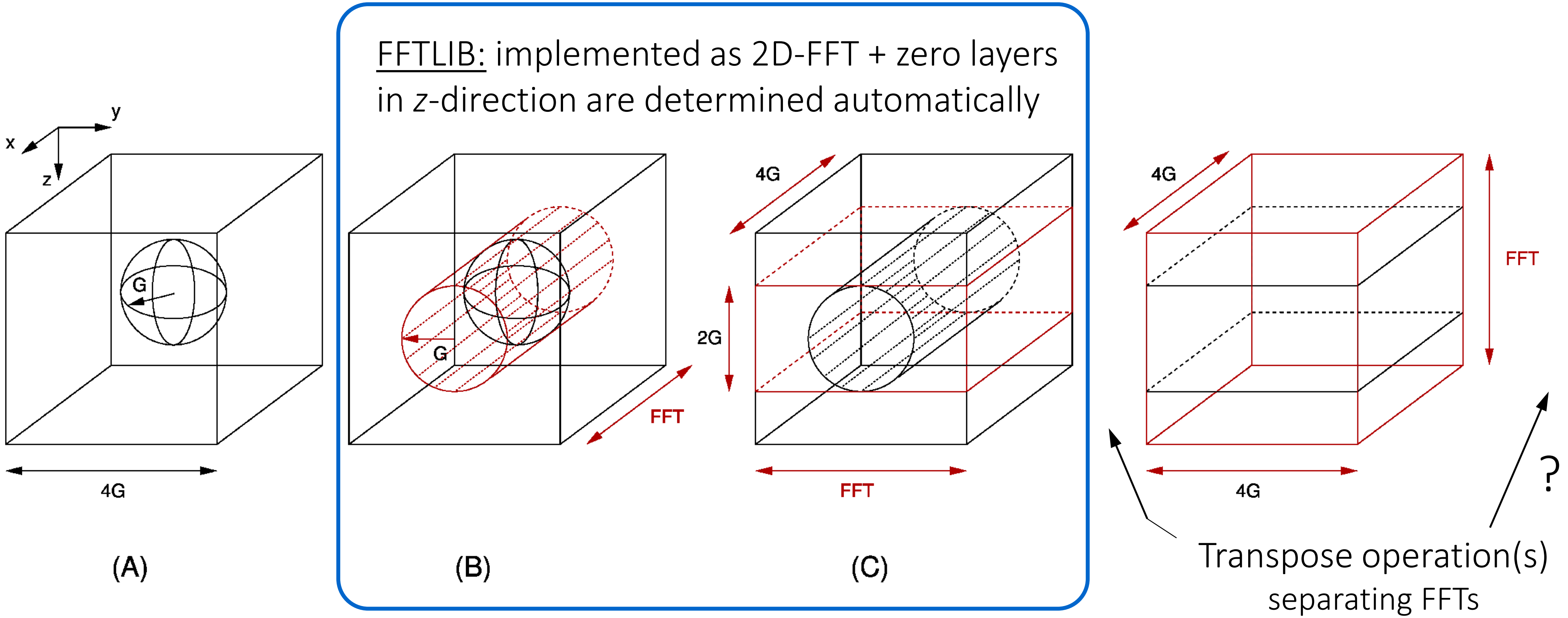
- Composed FFT: “Ball ⟷ Cube” FFT optimization (optional)
  - ❑ Reciprocal space vector *G below a certain cutoff*



How to Improve FFT Computation in VASP?

3D-FFT with FFTLIB

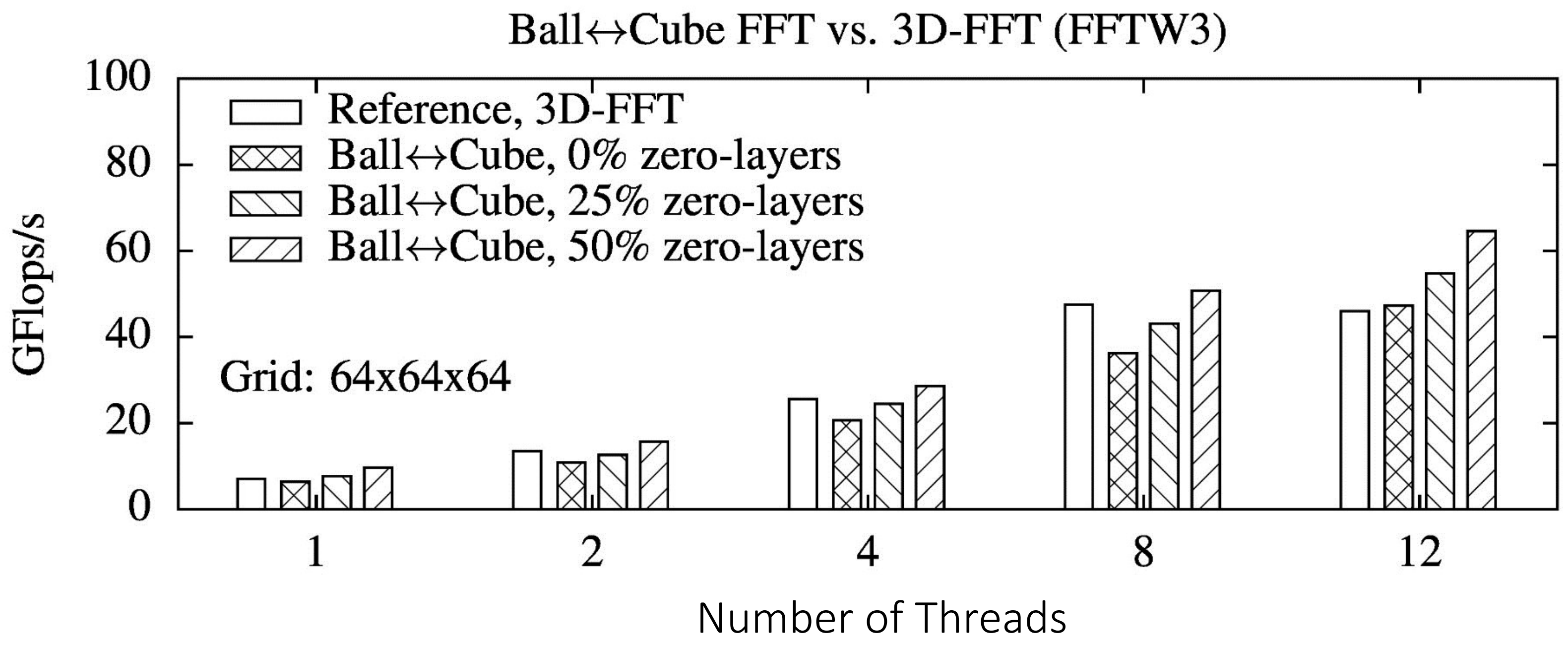
- Composed FFT: “Ball ↔ Cube” FFT optimization (optional)
  - ❑ Reciprocal space vector *G* below a certain cutoff



How to Improve FFT Computation in VASP?

3D-FFT with FFTLIB

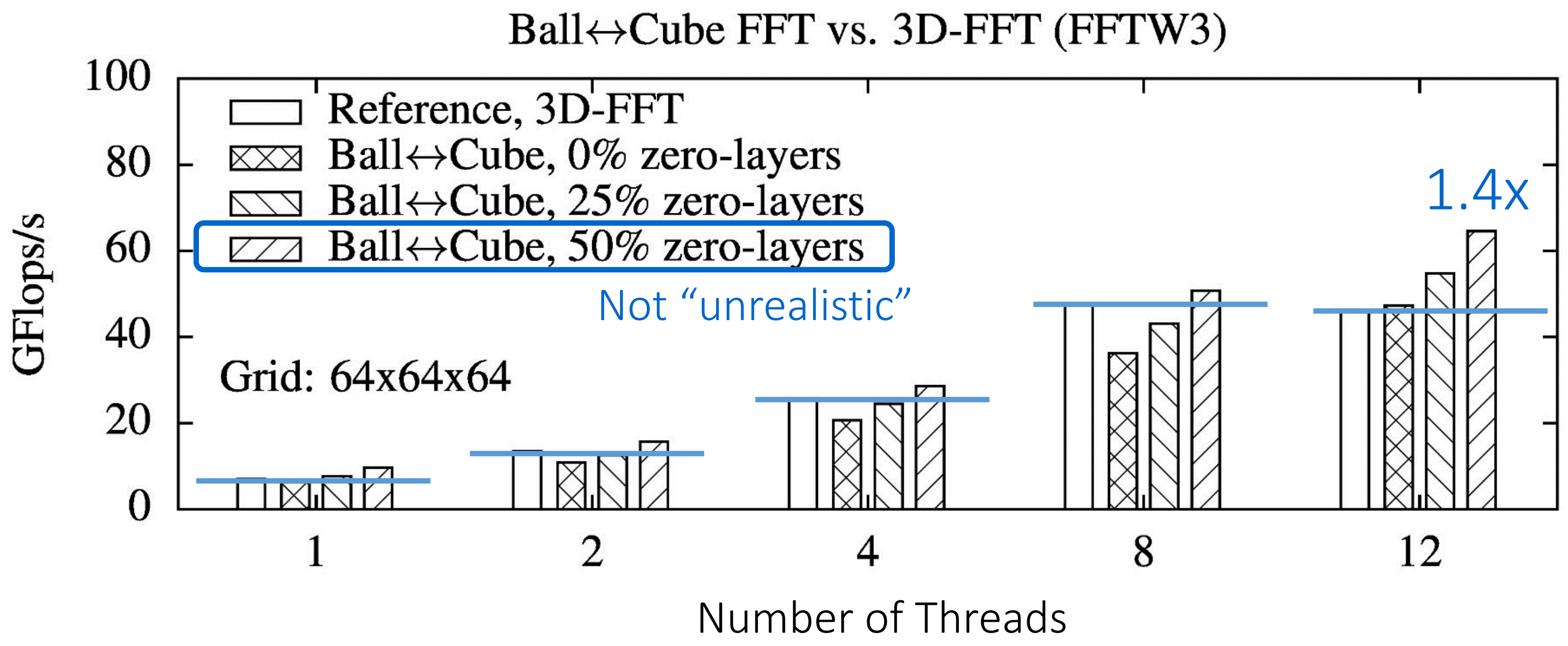
- Composed FFT: “Ball ↔ Cube” FFT optimization + skip last transpose
  - ❑ Synthetic benchmark kernel: here for FFTW, but similar for MKL
  - ❑ Not yet integrated into VASP



How to Improve FFT Computation in VASP?

3D-FFT with FFTLIB

- Composed FFT: "Ball ↔ Cube" FFT optimization + skip last transpose
  - ❑ Synthetic benchmark kernel: here for FFTW, but similar for MKL
  - ❑ Not yet integrated into VASP



Summary

Summary

Many-core optimization in VASP

- MPI + OpenMP, SIMD
- Multi-threaded library calls: 3D-FFT in this talk
  - ❑ Scaling quite acceptable
  - ❑ Issue with plan creation when using FFTW: consumes a lot of time

FFTLIB: C++ template library intercepting FFTW calls

- Plan reuse via hash-map + cache: up to 1.4x for VASP application with FFTW
- Composed 3D-FFT: 1.4x with FFTW when skipping last transpose

Not shown here (but in the paper)

- High bandwidth memory usage (memkind): 10% gain for transpose
- Autotuning within FFTLIB: just an outlook

Acknowledgement

Jeongnim Kim (Intel, US)

Funding:

Research Center for Many-core HPC (IPCC@ZIB)

Partners:

ZIB + VASP developer team (Georg Kresse, Martijn Marsman)

Joint research project ZIB + Cray