

Bringing Kokkos Performance Portability to BigDFT's Poisson Solver

a Fortran/C++ Interoperability Journey

Thibault Cimic — L_Sim (CEA Grenoble) / Maison de la Simulation (CEA Saclay)

Kokkos Tea Time — 21 October 2026



kokkos

Introduction

- Context
- BigDFT
- The PSolver module and its backends

Context

L_Sim
Grenoble



PTC

Programme Transverse de
Compétences

Maison de la Simulation
Saclay



YEAR 1 · THIS TALK

Implement a Kokkos backend in BigDFT's Fortran Poisson solver



The **psolver** module and its backends

psolver :

A generalized **Poisson solver** for various BC

→ **computational heart of BigDFT**

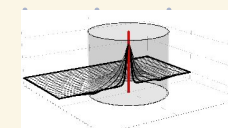
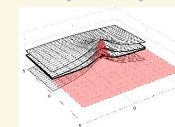
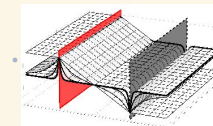
- Orchestrated in **Fortran** 
- Distributed via **MPI** 
- Optimised on **GPUs**   

Screened Poisson equation

$$(\nabla^2 - \mu^2) V(x, y, z) = -4\pi \rho(x, y, z)$$

Generalized Poisson equation

$$\nabla \cdot (\epsilon(x, y, z) \nabla V(x, y, z)) = -4\pi \rho(x, y, z)$$



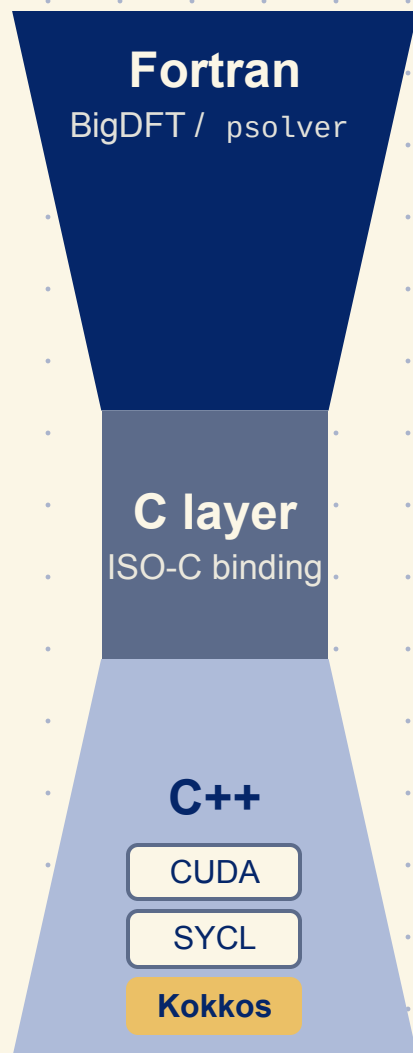
Backends :

Backend	Language / model	Hardware
SGFFT	Fortran / OpenMP	any CPU
ONEMATHCPU	SYCL + oneMKL (TBB)	x86 CPUs; used mostly on AMD hardware
ONEMATHGPU	SYCL + cuFFT / rocFFT / oneMKL	NVIDIA, AMD, Intel GPUs; used mostly on AMD
DBFFTCPU / DBFFT	SYCL (bbfft, OpenCL-JIT callbacks)	Intel GPUs, and CPUs through OpenCL
CUDA	pure CUDA (cuFFT)	NVIDIA GPUs only
KOKKOSCPU / KOKKOSGPU	Kokkos	any CPU and any GPU, one implementation

1. It works, how and why

- 1.1 Fortran $\leftrightarrow$ C++ interoperability: hourglass interface
- 1.2 Autotools $\leftrightarrow$ CMake interoperability
- 1.3 Validation
- 1.4 Exact exchange
- 1.5 The FFT algorithm

1.1 Fortran ↔ C++ interoperability: hourglass interface

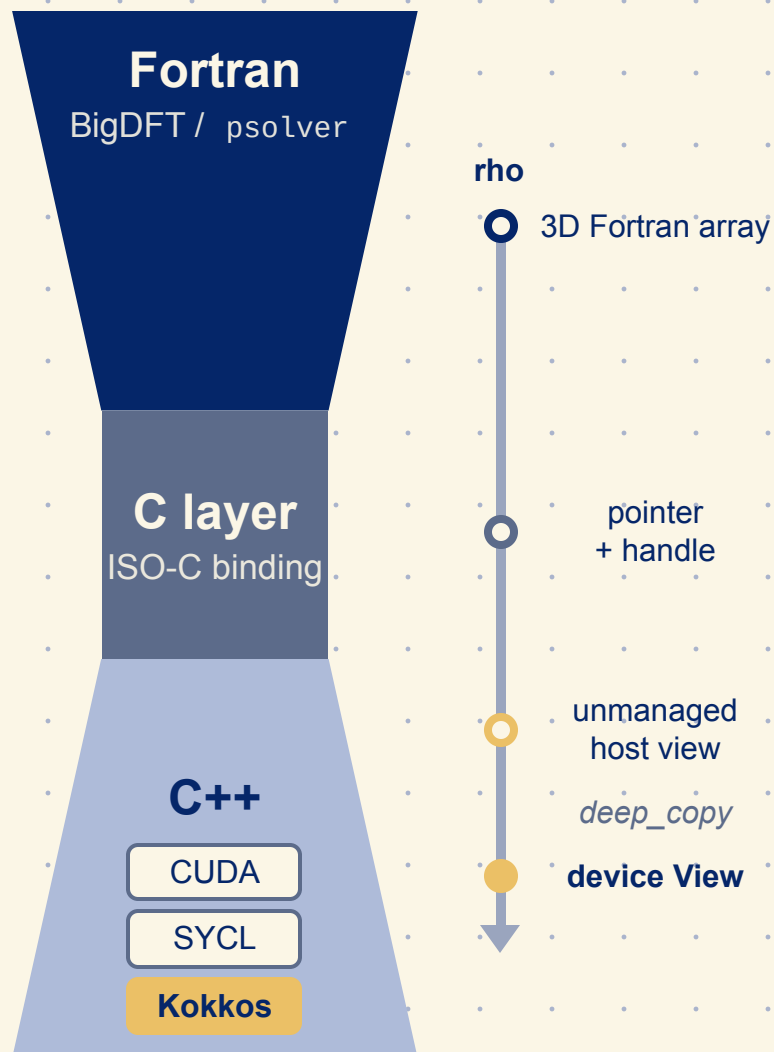


```
subroutine kokkos_HtoD(nelems, h_data, d_data) bind(C, name='kokkos_HtoD')
  integer(kind=c_int), intent(in), value :: nelems
  real(kind=c_double), intent(in) :: h_data(*)
  integer(f_address), intent(in), value :: d_data
end subroutine kokkos_HtoD
```

```
extern "C" {
void kokkos_HtoD(int nelems, double const *hData, intptr_t dData);
}
```

```
void kokkos_HtoD(int nElems, double const *hData, intptr_t dData) {
  auto *dViewPtr = reinterpret_cast<View1D<double>*>(dData);
  Kokkos::View<const double *, Kokkos::HostSpace,
              Kokkos::MemoryTraits<Kokkos::Unmanaged>>
    hView(hData, nElems);
  auto dSubview = Kokkos::subview(*dViewPtr, std::make_pair(0, nElems));
  Kokkos::deep_copy(dSubview, hView);
}
```

1.1 Fortran ↔ C++ interoperability: hourglass interface

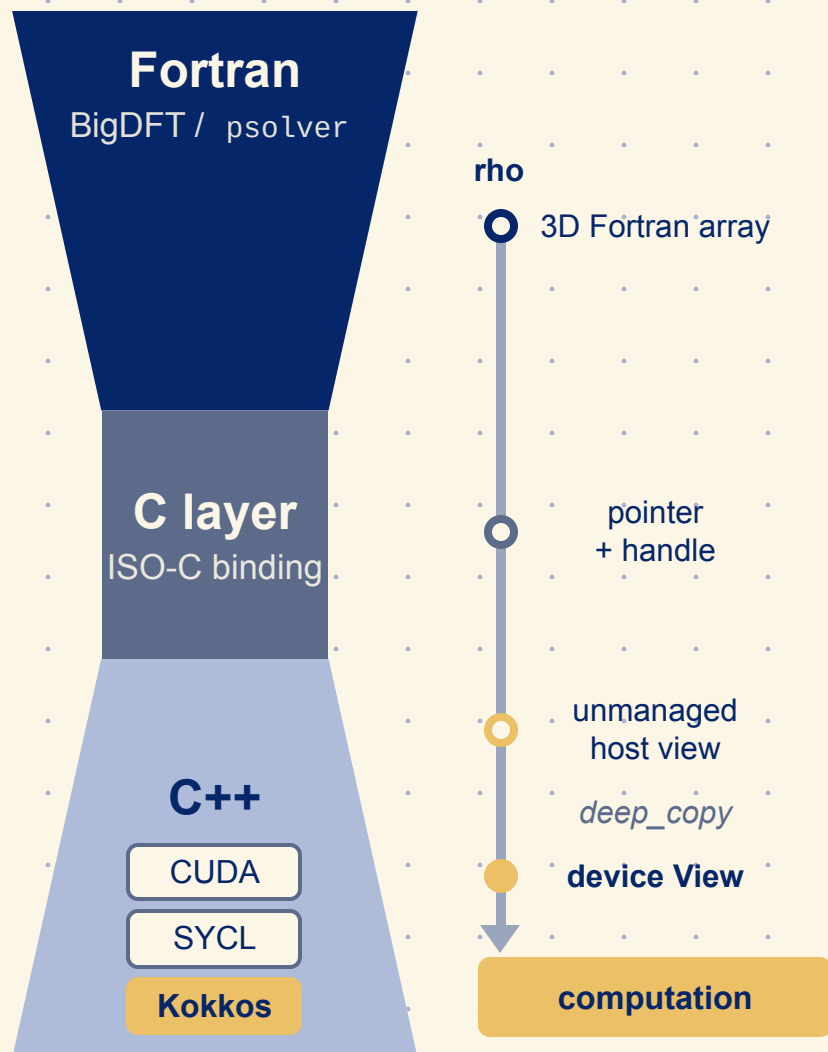


```
subroutine kokkos_HtoD(nelems, h_data, d_data) bind(C, name='kokkos_HtoD')
  integer(kind=c_int), intent(in), value :: nelems
  real(kind=c_double), intent(in) :: h_data(*)
  integer(f_address), intent(in), value :: d_data
end subroutine kokkos_HtoD
```

```
extern "C" {
  void kokkos_HtoD(int nelems, double const *hData, intptr_t dData);
}
```

```
void kokkos_HtoD(int nElems, double const *hData, intptr_t dData) {
  auto *dViewPtr = reinterpret_cast<View1D<double>*>(dData);
  Kokkos::View<const double *, Kokkos::HostSpace,
              Kokkos::MemoryTraits<Kokkos::Unmanaged>>
    hView(hData, nElems);
  auto dSubview = Kokkos::subview(*dViewPtr, std::make_pair(0, nElems));
  Kokkos::deep_copy(dSubview, hView);
}
```

1.1 Fortran ↔ C++ interoperability: hourglass interface

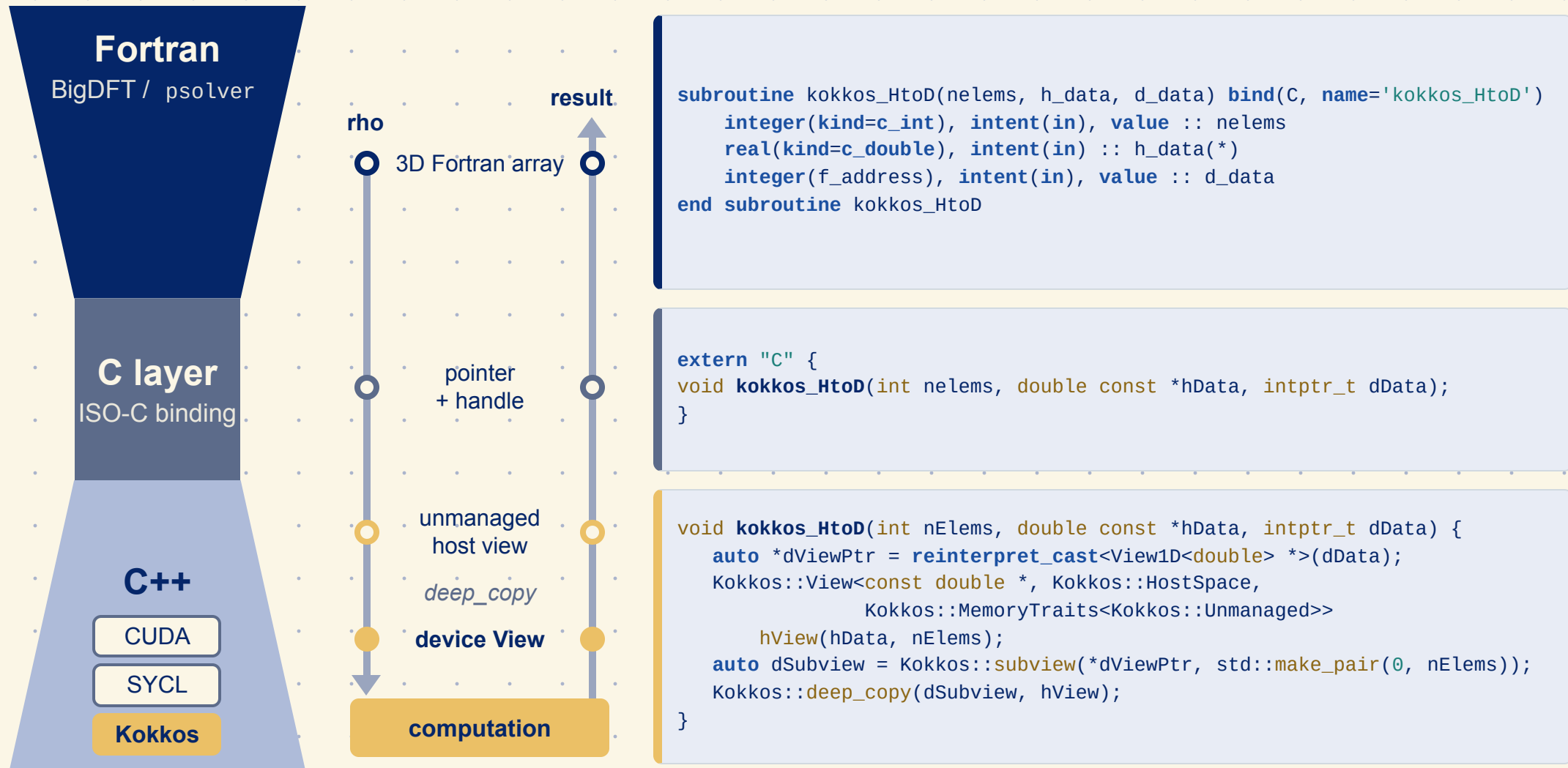


```
subroutine kokkos_HtoD(nelems, h_data, d_data) bind(C, name='kokkos_HtoD')
  integer(kind=c_int), intent(in), value :: nelems
  real(kind=c_double), intent(in) :: h_data(*)
  integer(f_address), intent(in), value :: d_data
end subroutine kokkos_HtoD
```

```
extern "C" {
  void kokkos_HtoD(int nelems, double const *hData, intptr_t dData);
}
```

```
void kokkos_HtoD(int nElems, double const *hData, intptr_t dData) {
  auto *dViewPtr = reinterpret_cast<View1D<double>*>(dData);
  Kokkos::View<const double *, Kokkos::HostSpace,
              Kokkos::MemoryTraits<Kokkos::Unmanaged>>
    hView(hData, nElems);
  auto dSubview = Kokkos::subview(*dViewPtr, std::make_pair(0, nElems));
  Kokkos::deep_copy(dSubview, hView);
}
```

1.1 Fortran ↔ C++ interoperability: hourglass interface



1.2 Autotools ↔ CMake interoperability

CMake

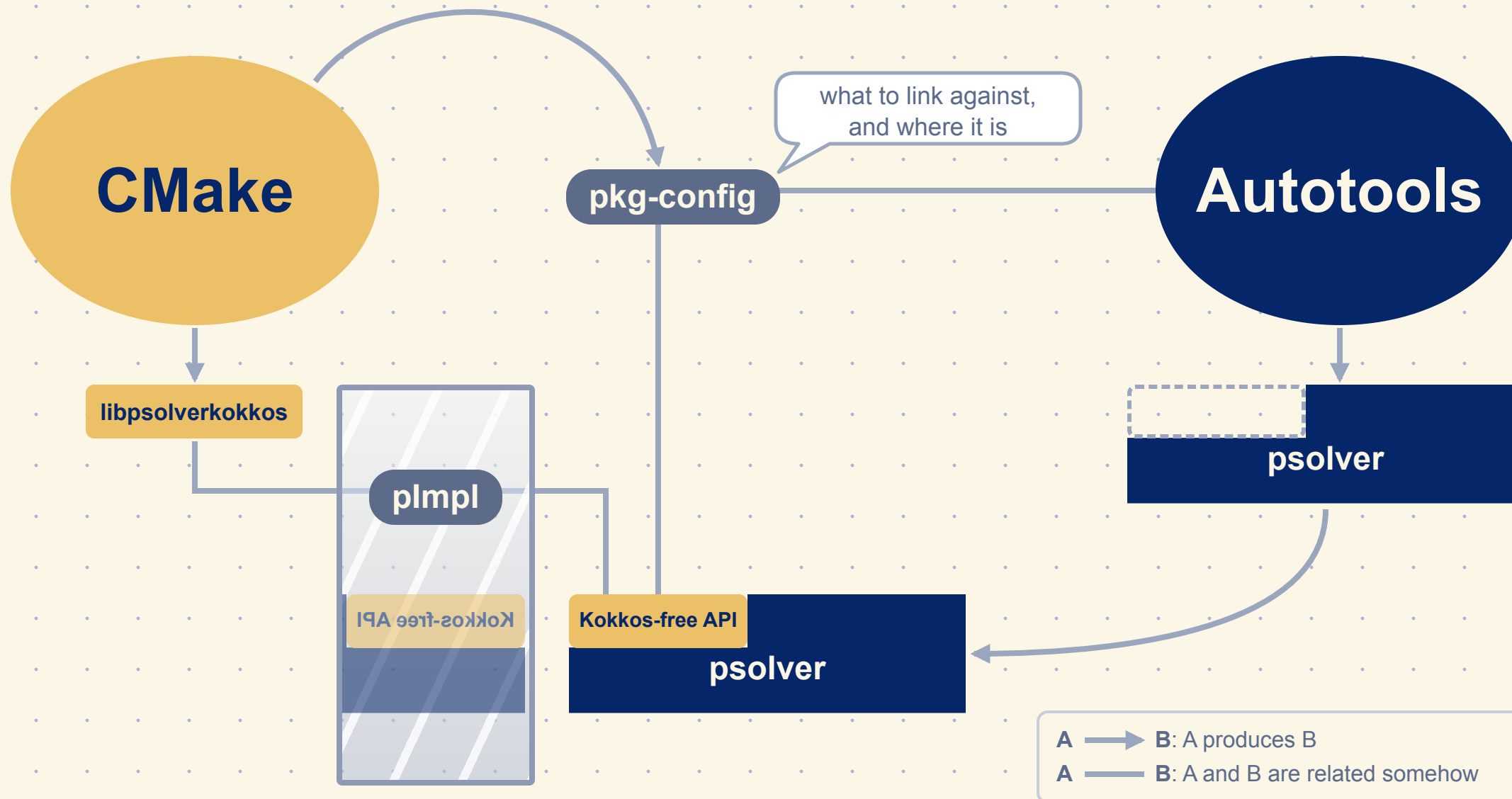
Autotools

1.2 Autotools ↔ CMake interoperability



A → B: A produces B

1.2 Autotools ↔ CMake interoperability



1.3 It works: extensively validated

Level	Test	Covers
libpsolverkokkos	24 public-API tests	memory, transfers, pad / unpad, pre / post, exchange, reductions
	8 internal kernel tests	transposes, spread, fused transpose+spread, multiply
psolver	PS_Program	Solves : $\begin{cases} \nabla^2 V(\mathbf{r}) = -4\pi \rho(\mathbf{r}) \\ \text{BC} : F, W, S, P \end{cases}$
	Fock	Computes $E_x = - \sum_{i,j} f_i f_j \frac{1}{2} \int \rho_{i,j} V_{i,j}$, where $\forall i, j$ $\begin{cases} \rho_{i,j} = \phi_i \phi_j \\ \nabla^2 V_{i,j} = -4\pi \rho_{i,j} \end{cases}$

→ Fock is also the application every benchmark in part 2 runs (explained next)

1.4 Exact exchange

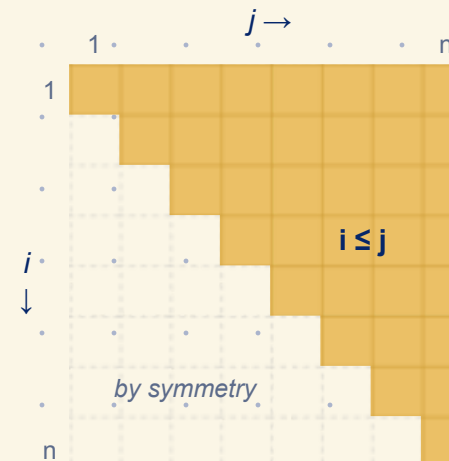
for each exact-exchange evaluation

for i in orbitals:

for j in orbitals, $j \geq i$:

```

pre_computation()     $\rho_{ij} = \phi_i \cdot \phi_j$ 
pad_data()           grid  $\rightarrow$  FFT work array
PoissonSolve()        $\nabla^2 V_{ij} = -4\pi \rho_{ij}$ 
unpad_data()         FFT work array  $\rightarrow$  grid
reduction()           $E_{ij} = \int \rho_{ij} V_{ij}$ 
accumulate_eexctx()   $Ex \leftarrow Ex + \frac{1}{2} f_i f_j E_{ij}$ 
post_computation()    $D\phi_i \leftarrow D\phi_i + f_j V_{ij} \phi_j$ 
  
```



$\frac{n(n+1)}{2}$ Poisson solves per evaluation
 $n = 128$ orbitals (H2O-32): **8256**

■ the FFT algorithm (next slide) ■ Kokkos kernels

the kernels around the solve

```

pre_computation  post_computation  accumulate_eexctx
pad_data        unpad_data
reduction
  
```

element-wise on the grid

data movement: grid $\leftrightarrow$ FFT work array

the energy of the pair, and $\rho_{ij} \leftarrow V_{ij}$

1.5 The FFT algorithm and its kernels

forward

```
for axis in [X,Y,Z]:
    <transpose,Spread>(axis)
    FFT(axis)
```

Fourier space computation

```
multiplyKernel():
    data[i] *= scale * kernel[i]
```

backward

```
for axis in [Z,Y,X]:
    FFTInv(axis)
    <transpose,Spread>Inv(axis)
```

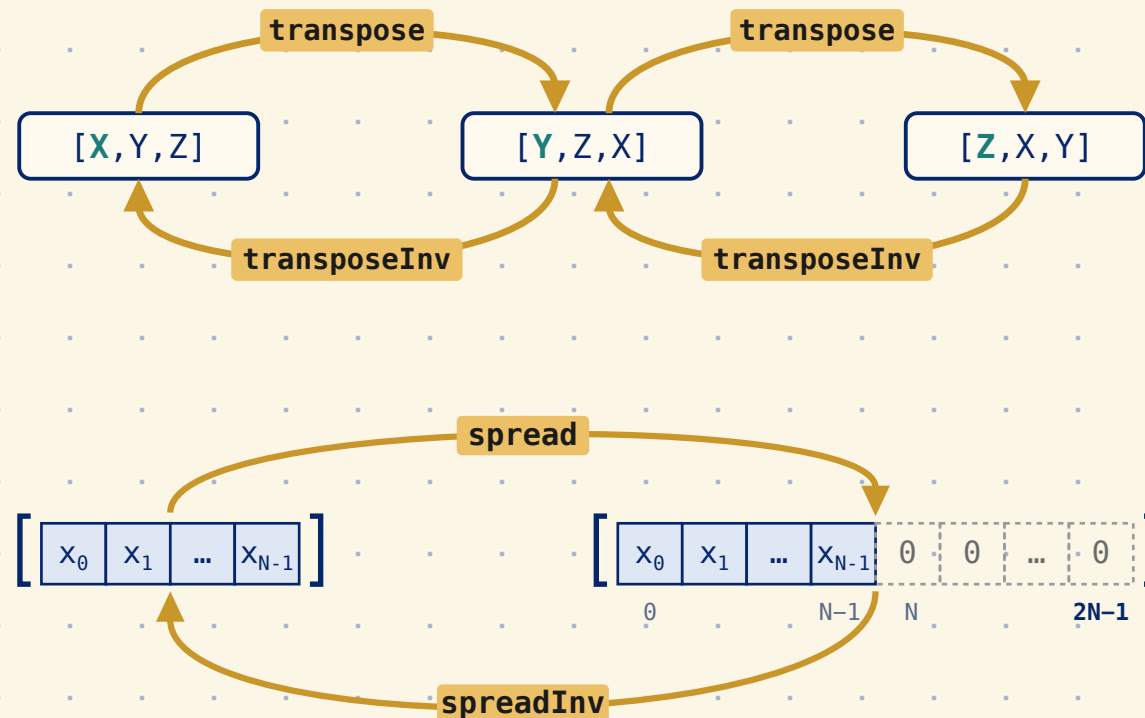
 KokkosFFT  Kokkos kernels

the kernels

spread **spreadInverse** **transpose** **transposeSpread** **transposeSpreadInverse**
multiplyKernel

pure data movement

the only computation



2. Performance

- 2.1 What we measured
- 2.2 What we compared
- 2.3 CPU results
- 2.4 GPU results
- 2.5 Where the gap comes from
- 2.6 Memory bandwidth, and beyond
- 2.7 Performance portability

2.1 What we measured

Software

Kokkos 5.2.2

KokkosFFT 2.0.0

Cluster	CPU	GPU
Ruche	2 × (Intel Xeon Gold 6230 · 20 cores · 2.1 GHz) 4 MPI ranks × 10 OpenMP threads	1 × (NVIDIA A100 · 40 GB HBM2 · 1555 GB/s) CUDA 13.0.2
Adastra	2 × (AMD EPYC 9654 · 96 cores · 2.4 GHz) 12 MPI ranks × 16 OpenMP threads	1 × (GCD AMD MI250X · 64 GB HBM2e · 1638 GB/s) ROCm 6.4.3

System	BC	Grid	Orbitals	Poisson solves
H2O-32	periodic	96 ³	128	8256
Aflatoxin	free	209×191×145	58	1711
Graphene-H	surface	216×91×216	40↑ + 40↓	1640
UO2_2	periodic	108 ³	90↑ + 90↓	8190

Backends

SGFFT

ONEMATHCPU

ONEMATHGPU

CUDA

KOKKOSCPU

KOKKOSGPU

Method

exclusive nodes

5 runs on GPU, 3 on CPU

Metrics

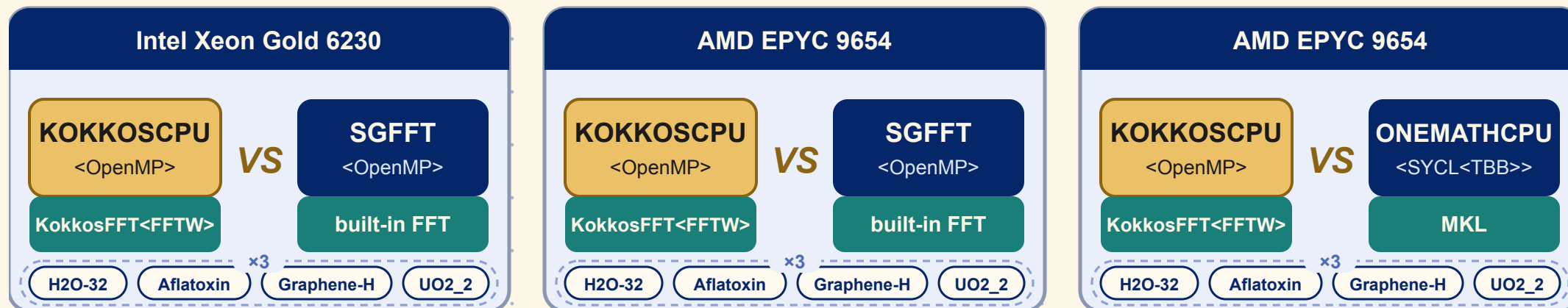
Fock wall time

GPU time

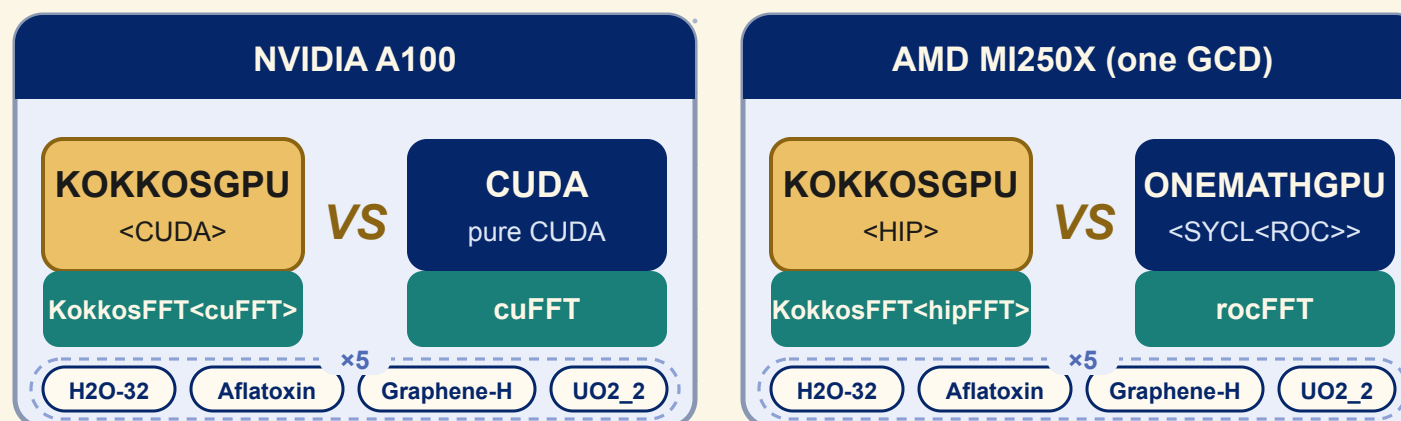
memory consumption

2.2 What we compared

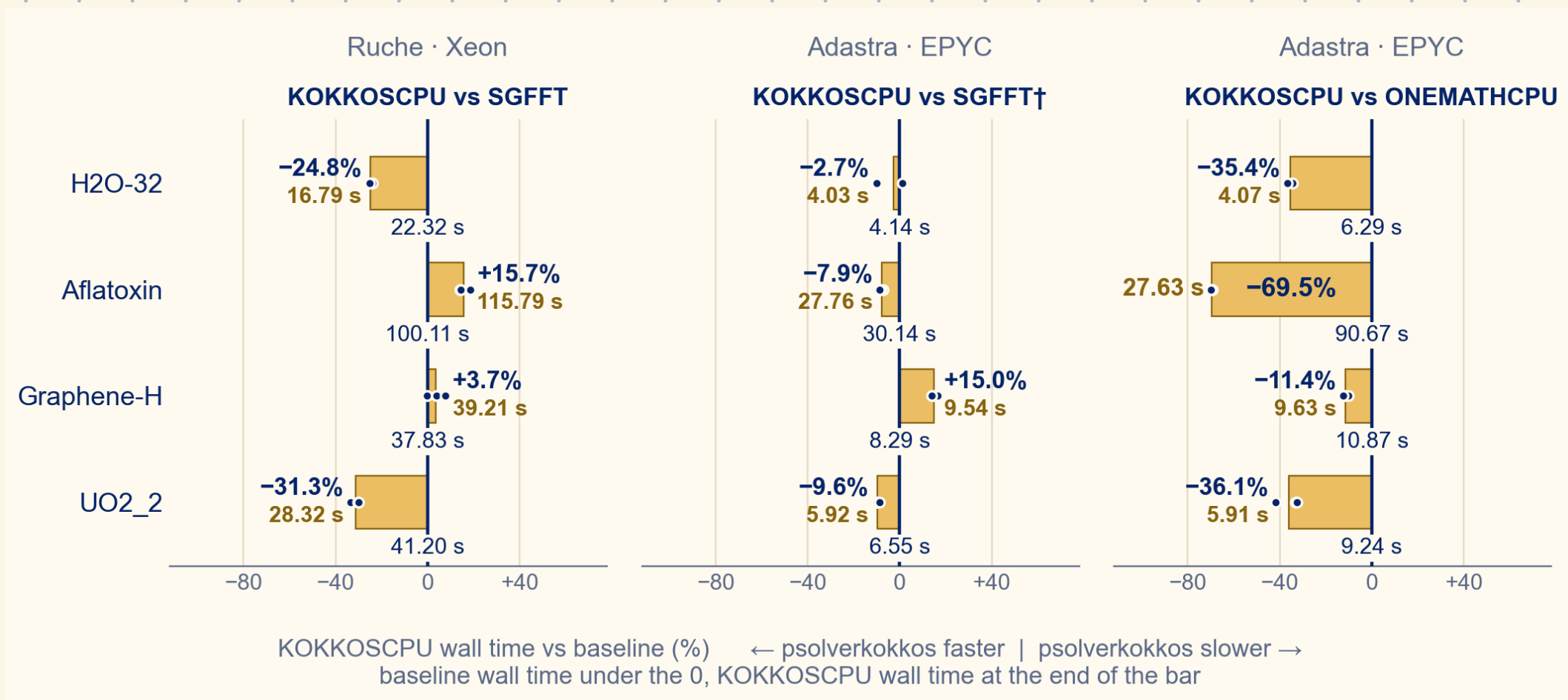
CPU



GPU

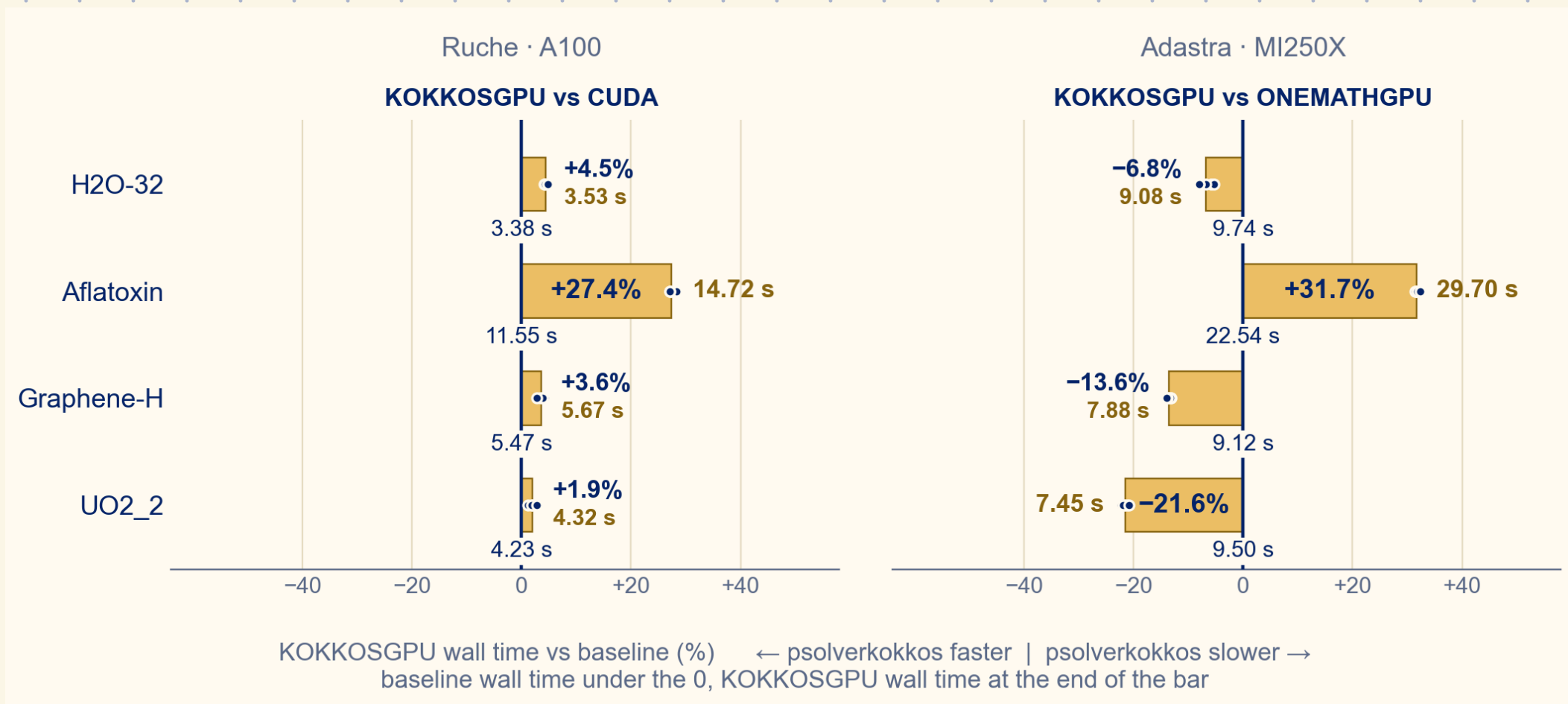


2.3 CPU: indicative results

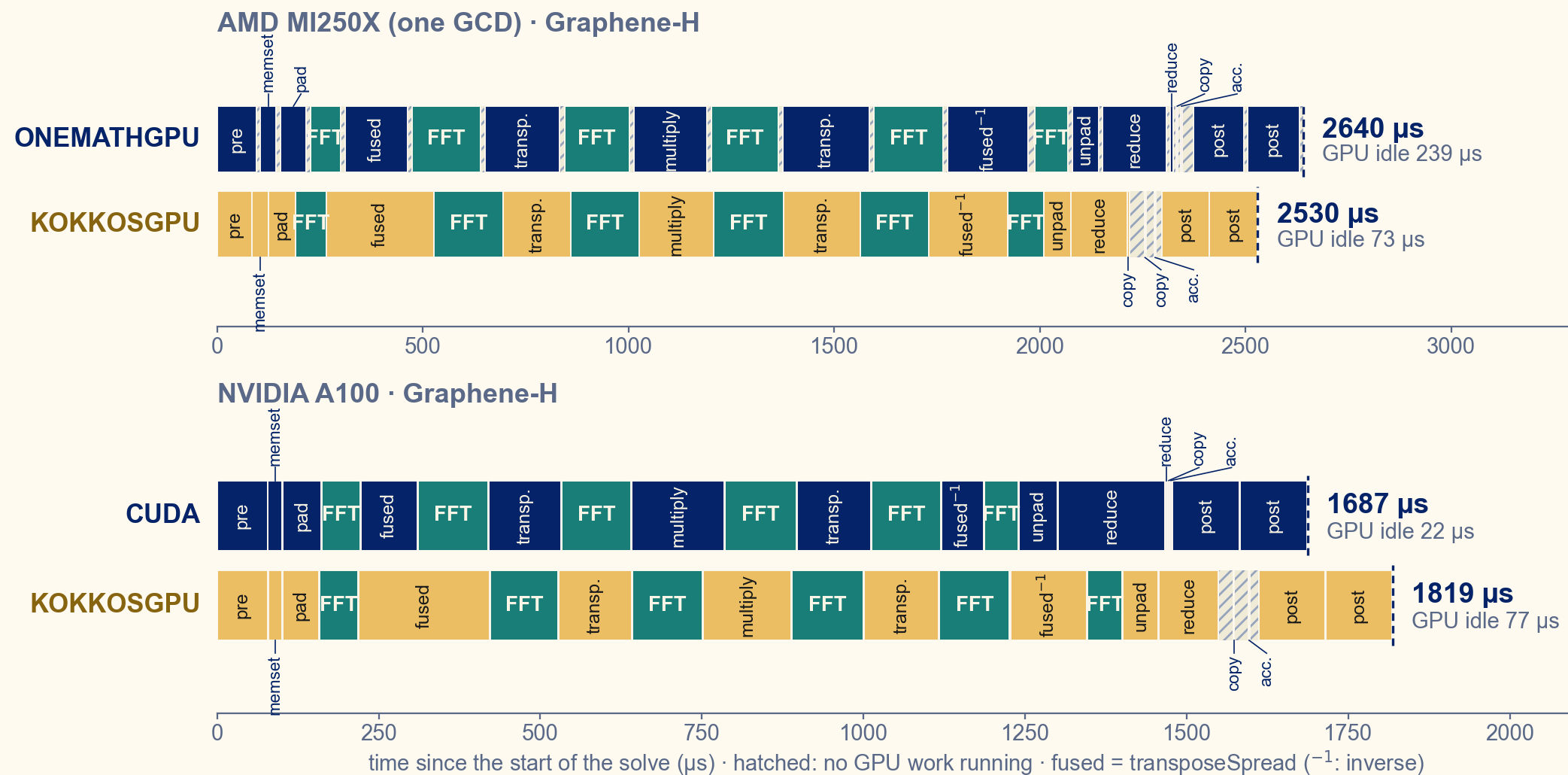


† indicative only: not reproducible against the group's published SGFFT (our build: -O3 -march=znver4, theirs: -O2)

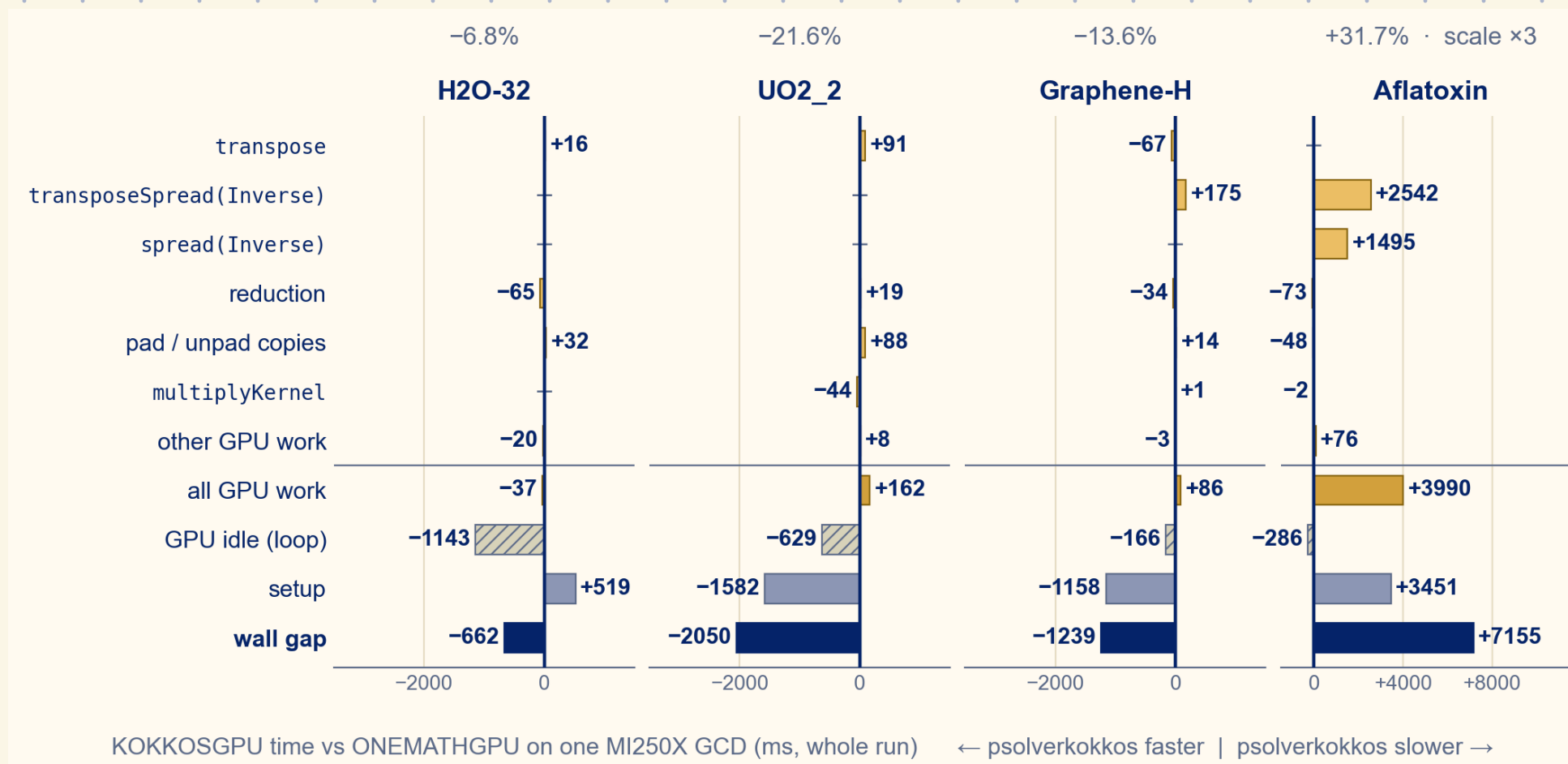
2.4 GPU results



2.4 GPU results: One full Fock iteration



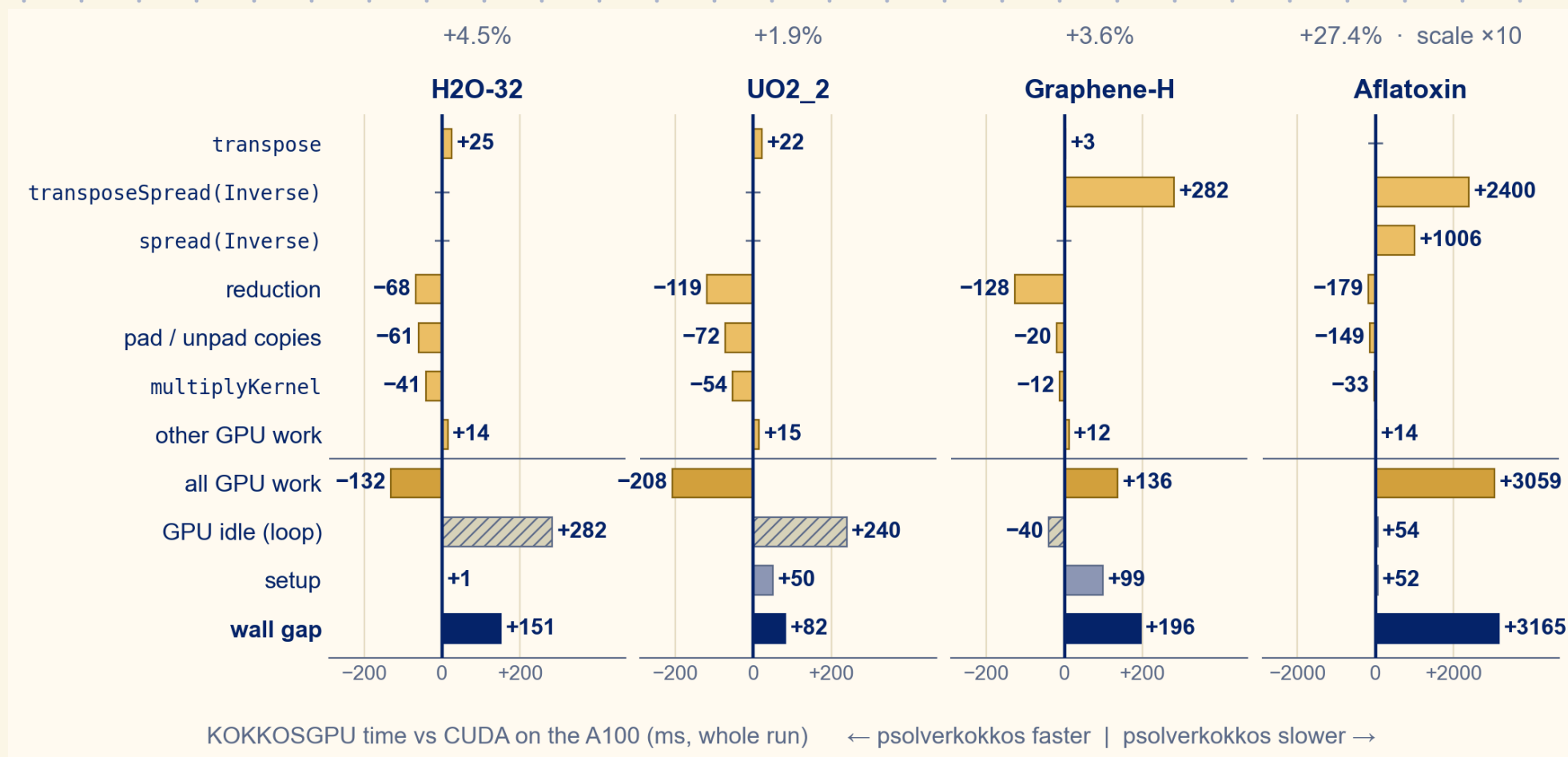
2.5 Where the gap comes from: MI250X



GPU idle (loop) = loop time gap - GPU gap · gaps between the kernels, per solve, mean over the run: ONEMATH $\approx 270 \mu\text{s}$, psolverkokkos $\approx 80 \mu\text{s}$
 setup = wall time outside the loop of solves, mostly FFT plan creation on this GPU (extra slide), not GPU work

Graphene-H, Aflatoxin: fused and spread kernels, no shared-memory tiling yet (inferred)

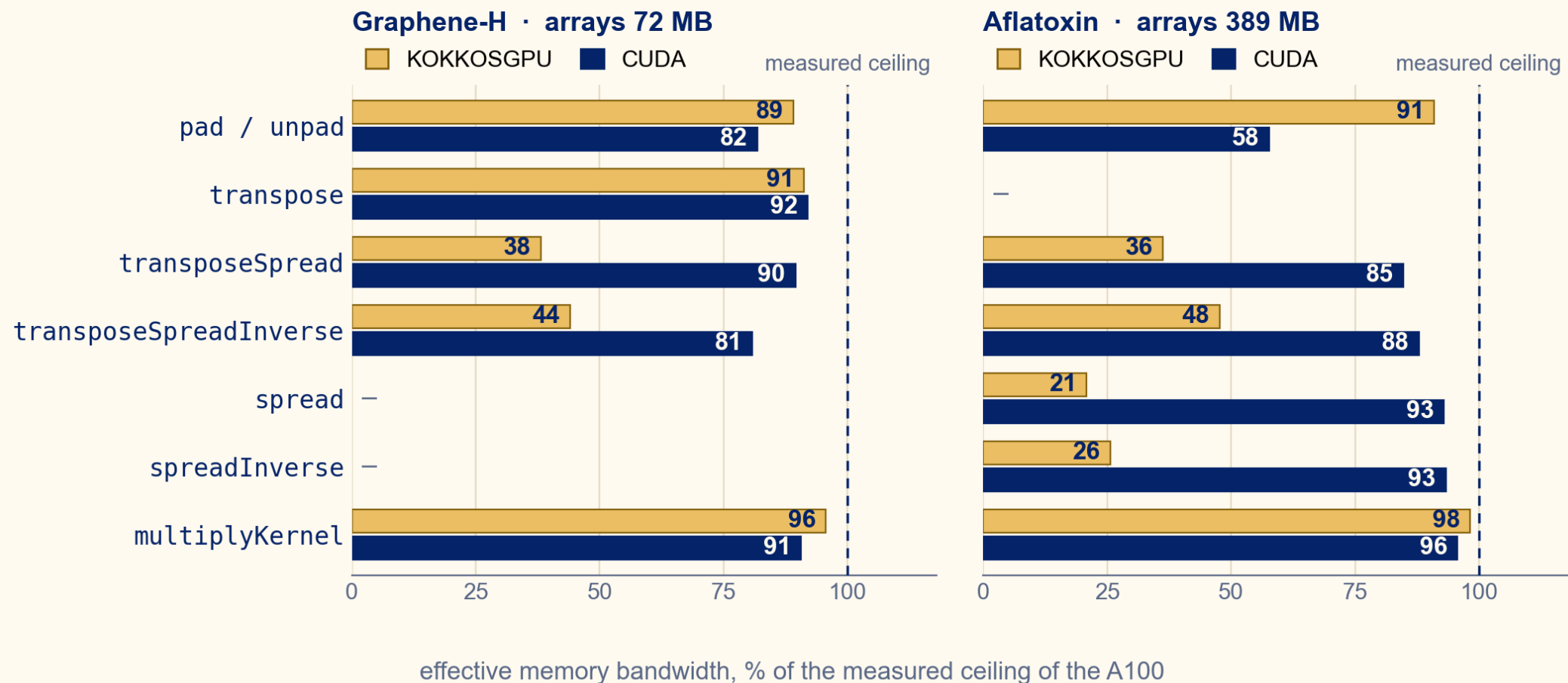
2.5 Where the gap comes from: A100



GPU idle (loop) = loop time gap - GPU gap · gaps between the kernels, per solve, mean over the run: psolverkokkos ≈ 80 μs (fence, scalar copy);
 CUDA ≈ 24 μs · setup = wall time outside the loop of solves (A100: at most 0.1 s), not GPU work

Graphene-H, Aflatoxin: fused and spread kernels, no shared-memory tiling yet (inferred)

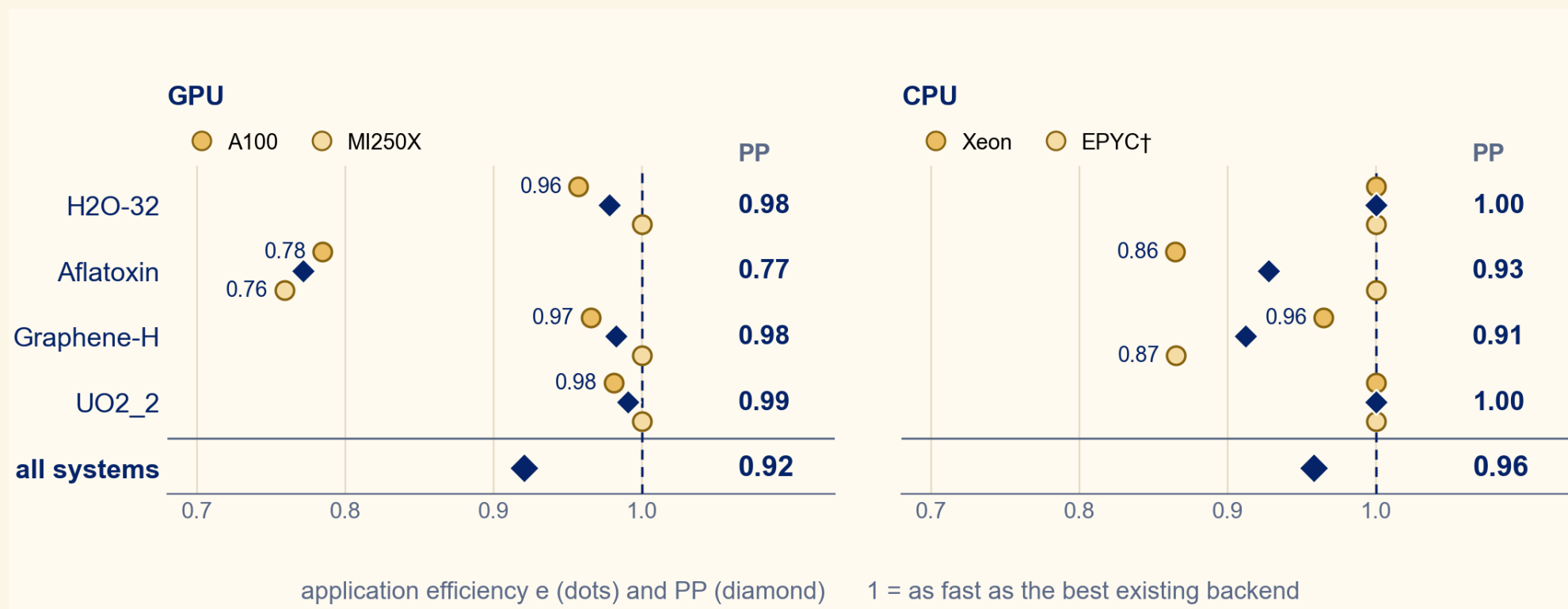
2.6 Memory bandwidth, and beyond



dashed line: bandwidth measured on the same GPU (copy kernel; triad for `multiplyKernel`)

2.7 Performance portability: the Pennycook metric

$$PP(H) = \frac{|H|}{\sum_{i \in H} \frac{1}{e_i}} \quad e_i = \min\left(1, \frac{t_i^{\text{best existing backend}}}{t_i^{\text{Kokkos}}}\right)$$



Pennycook PP, application-efficiency flavour · † EPYC baseline: our own SGFFT build, possibly too slow · all four platforms: PP 0.94

3. Conclusion

Recap and outlook

3.1 Recap

It works

24 + 8

library tests, plus PS_Program and Fock

hourglass interface: Fortran
↔ C ↔ Kokkos

one library, two build
systems

one implementation, 4
platforms

same energies on every
backend

3.1 Recap

It works

24 + 8

library tests, plus PS_Program and Fock

hourglass interface: Fortran
↔ C ↔ Kokkos

one library, two build
systems

one implementation, 4
platforms

same energies on every
backend

It performs

PP 0.92 - 0.96

GPUs · CPUs

A100 vs CUDA, MI250X vs
ONEMATHGPU

Xeon, EPYC vs SGFFT

GPU PP ≥ 0.98 on 3 of 4
systems

3.1 Recap

It works

24 + 8

library tests, plus PS_Program and Fock

hourglass interface: Fortran
↔ C ↔ Kokkos

one library, two build
systems

one implementation, 4
platforms

same energies on every
backend

It performs

PP 0.92 · 0.96

GPUs · CPUs

A100 vs CUDA, MI250X vs
ONEMATHGPU

Xeon, EPYC vs SGFFT

GPU PP ≥ 0.98 on 3 of 4
systems

The gap

Aflatoxin 0.77

GPU PP, the one outlier

fused transposeSpread
and spread kernels

GPU idle time

3.2 Outlook

Close the gap

optimise `transposeSpread` and `spread` kernels

reduce GPU idle time

expected, not measured

3.2 Outlook

Close the gap

optimise `transposeSpread` and `spread` kernels

reduce GPU idle time

expected, not measured

Go beyond

FFT callbacks (KokkosFFT)

by analogy, not measured

3.2 Outlook

Close the gap

optimise `transposeSpread` and `spread` kernels

reduce GPU idle time

expected, not measured

Go beyond

FFT callbacks (KokkosFFT)

by analogy, not measured

Clean up

`psolverkokkos` + `futilekokkos`

a label per View

errors through the interface

Thanks

Questions?

The slides

this Kokkos Tea Time page
downloadable there soon

The repository behind them

gitlab.com/thibaultcemic/kokkos-tea-time
build the slides, and the benchmark and
profiling data

BigDFT psolver papers

Genovese, Deutsch, Neelov, Goedecker, Beylkin, *Efficient solution of Poisson's equation with free boundary conditions*, J. Chem. Phys. 125, 074105 (2006)

Genovese, Deutsch, Goedecker, *Efficient and accurate three-dimensional Poisson solver for surface problems*, J. Chem. Phys. 127, 054704 (2007)

Cerioni, Genovese, Mirone, Sole, *Efficient and accurate solver of the three-dimensional screened and unscreened Poisson's equation with generic boundary conditions*, J. Chem. Phys. 137, 134108 (2012)

Dugan, Genovese, Goedecker, *A customized 3D GPU Poisson solver for free boundary conditions*, Comput. Phys. Commun. 184, 1815 (2013)

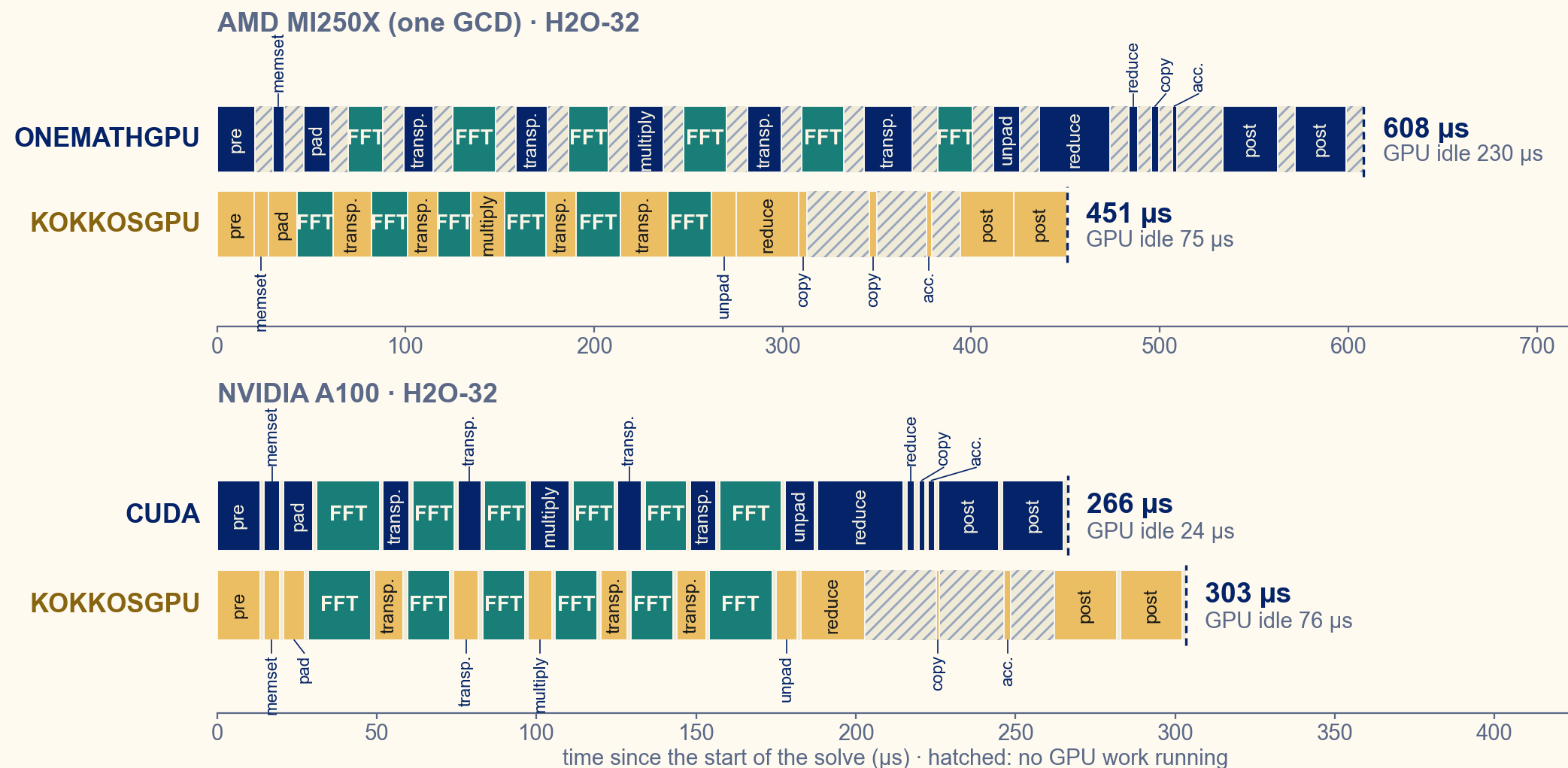
Fisicaro, Genovese, Andreussi, Marzari, Goedecker, *A generalized Poisson and Poisson-Boltzmann solver for electrostatic environments*, J. Chem. Phys. 144, 014103 (2016)

Fisicaro, Genovese, Andreussi, Mandal, Nair, Marzari, Goedecker, *Soft-sphere continuum solvation in electronic-structure calculations*, J. Chem. Theory Comput. 13, 3829 (2017)

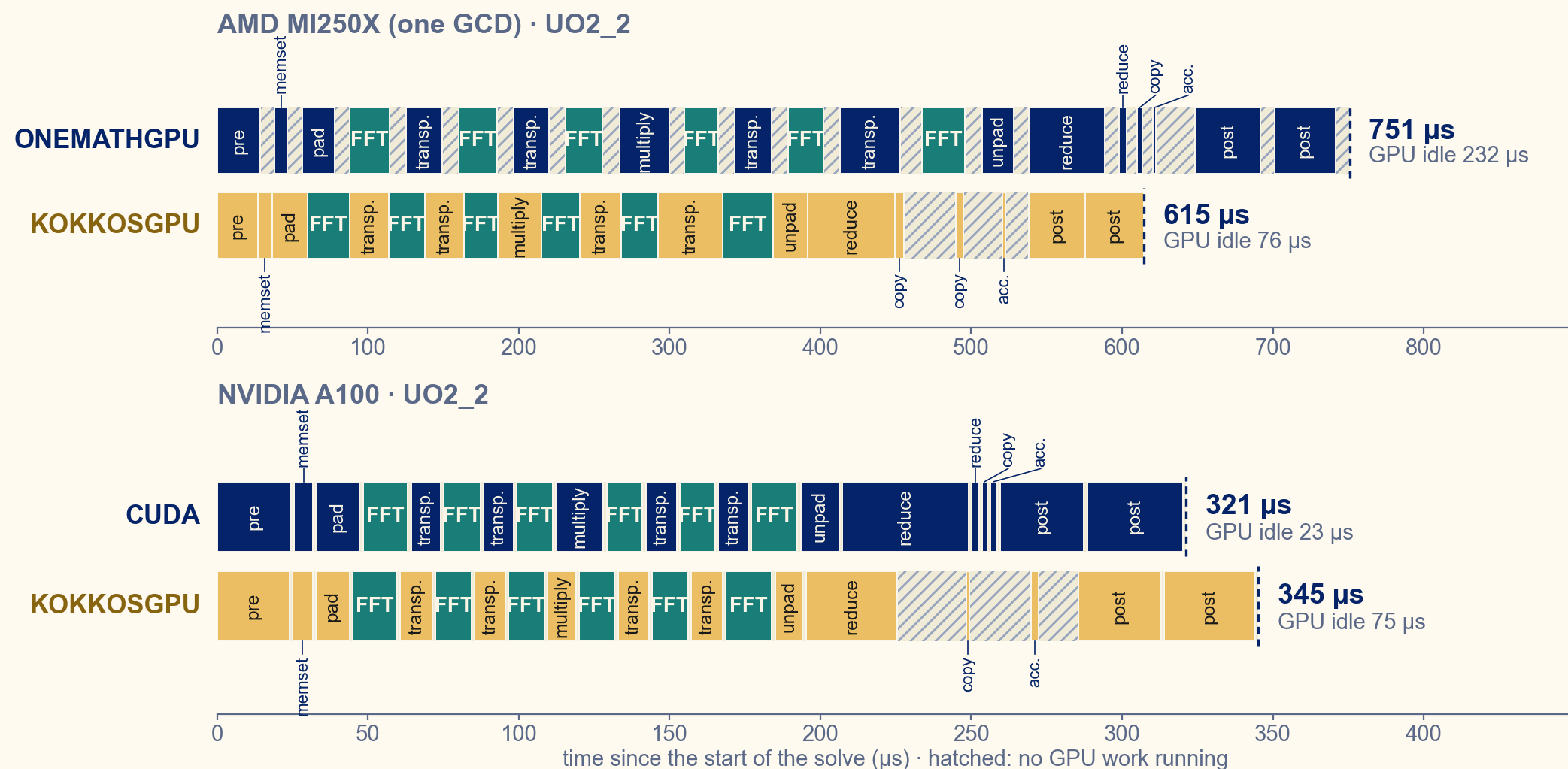
Ratcliff, Degomme, Flores-Livas, Goedecker, Genovese, *Affordable and accurate large-scale hybrid-functional calculations on GPU-accelerated supercomputers*, J. Phys.: Condens. Matter 30, 095901 (2018)



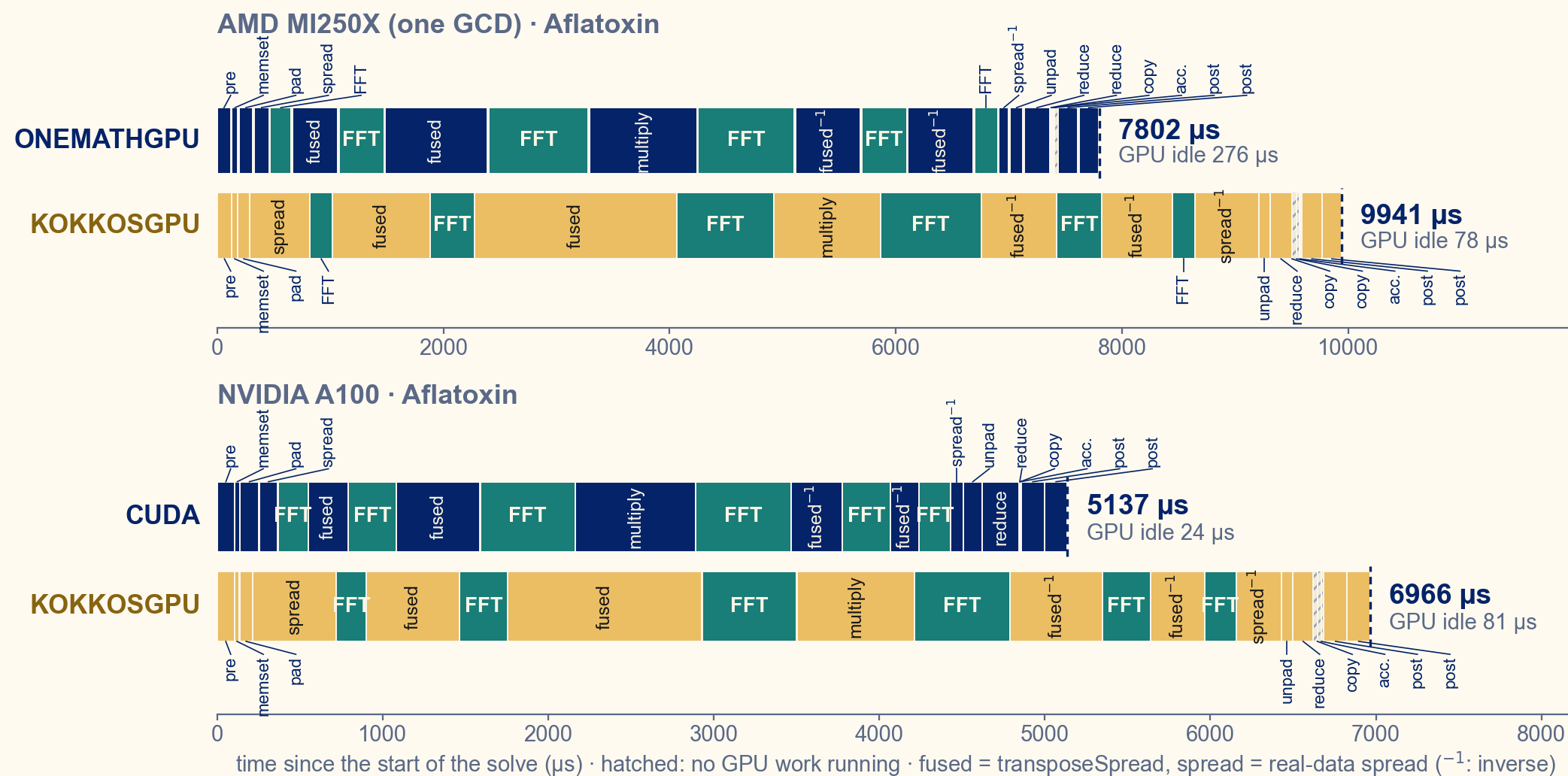
GPU results: One full Fock iteration



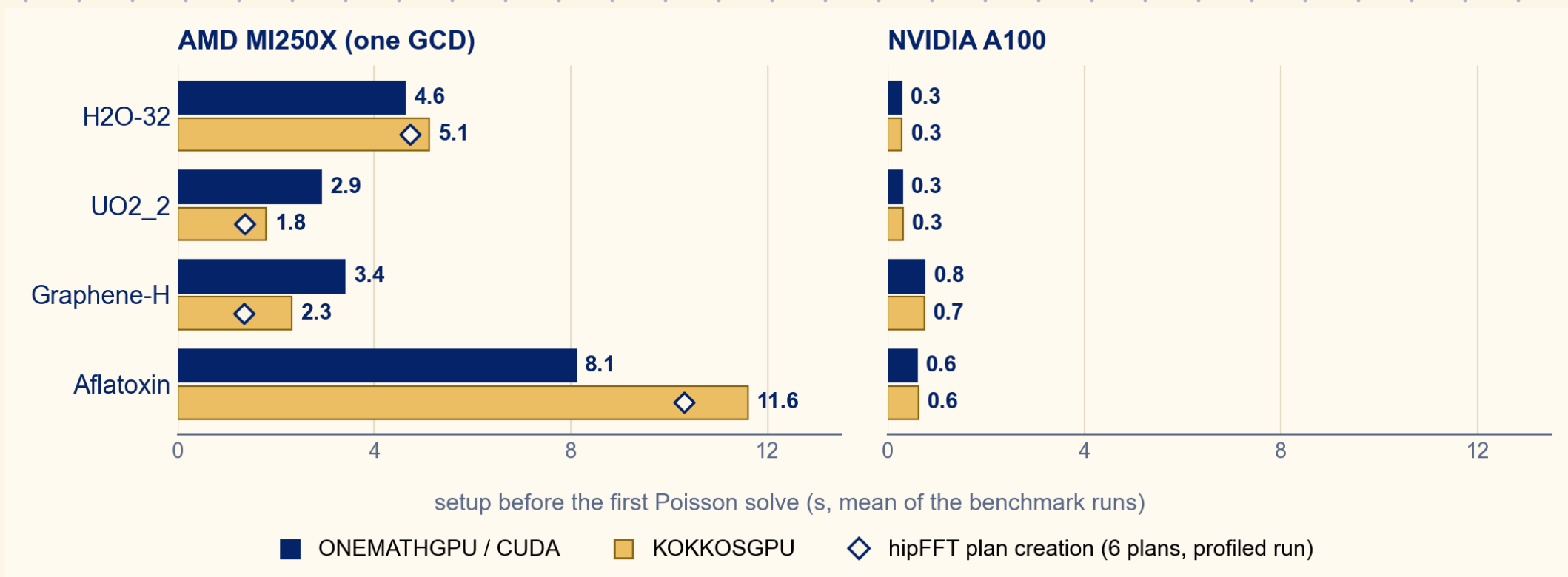
GPU results: One full Fock iteration



GPU results: One full Fock iteration



Setup on the AMD GPU: FFT plan creation



Aflatoxin, MI250X: the same 6 rocFFT kernels in both backends · pauses of about 1 s before the first solve: 7 with ONEMATH, 13 with psolverkokkos