

Russian Supercomputing Days
Moscow, September 28–29, 2026



Опыт разработки пакета BUFO для расчетов электронной структуры: мультиплатформенность и параллелизм на основе MPI+Kokkos



Iurii Kashpurovich, Vladislav Galigerov,
Alexander Oleynichenko, Leonid Pilugin, Vladimir Stegailov
JIHT RAS – MIPT – PNPI ("Kurchatov Institute") – HSE

September 28–29, 2026







Introduction: Context

$$H\psi = E\psi$$

Introduction: Context

$H\psi = E\psi$ ⟶ How to solve accurately and fast?

Introduction: Context

$H\psi = E\psi$ ⟶ How to solve accurately and fast?

+ *portably* fast

Introduction: Context

$H\psi = E\psi$ ⟶ How to solve accurately and fast?

+ *portably* fast



Approaches

Introduction: Context

$H\psi = E\psi$ ⟶ How to solve accurately and fast?

\+ *portably* fast



Approaches



Library-based Offloading
(ELPA, DBSCR, TAMM, etc)

Introduction: Context

$H\psi = E\psi$ ⟶ How to solve accurately and fast?

+ *portably* fast

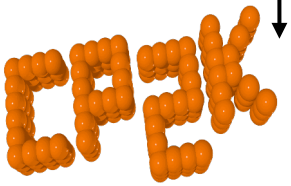
⬇

Approaches

↙

Library-based Offloading
(ELPA, DBSCR, TAMM, etc)

↓





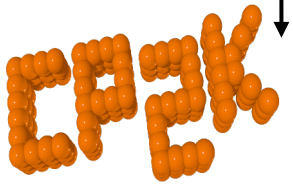
Introduction: Context

$H\psi = E\psi$ → How to solve accurately and fast?
+ *portably* fast

Approaches

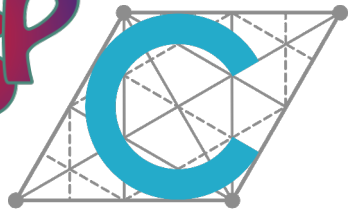
Library-based Offloading
(ELPA, DBSCR, TAMM, etc)

OpenMP/OpenACC
Directive Offloading













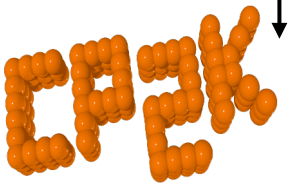


Introduction: Context

$H\psi = E\psi$ → How to solve accurately and fast?
+ *portably* fast

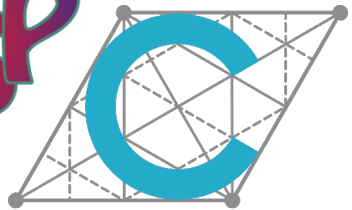
Approaches

- Library-based Offloading (ELPA, DBSCR, TAMM, etc)
- OpenMP/OpenACC Directive Offloading
- GPU-native















Introduction: Context

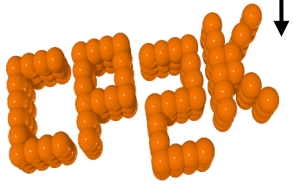
$H\psi = E\psi$ → How to solve accurately and fast?
+ *portably* fast

Approaches

Library-based Offloading
(ELPA, DBSCR, TAMM, etc)

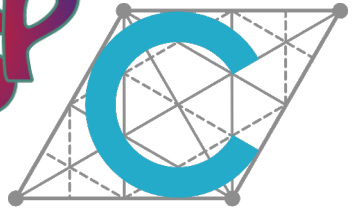
OpenMP/OpenACC
Directive Offloading

GPU-native









EXESS





TeraChem





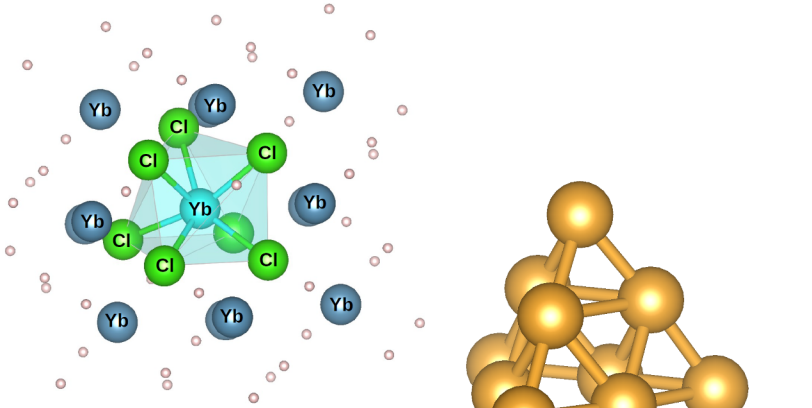


Introduction: BUFO

New code **BUFO** for efficient portable relativistic simulations oriented to model *d*- and *f*-element compounds and based on Gatchina pseudo-potentials

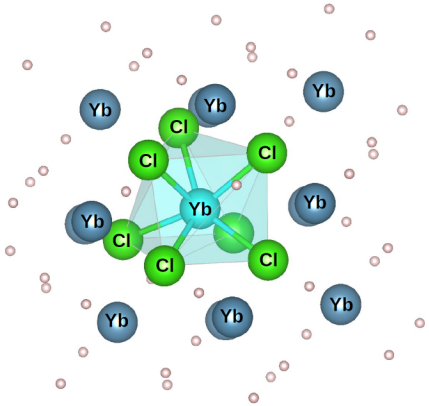
Introduction: BUFO

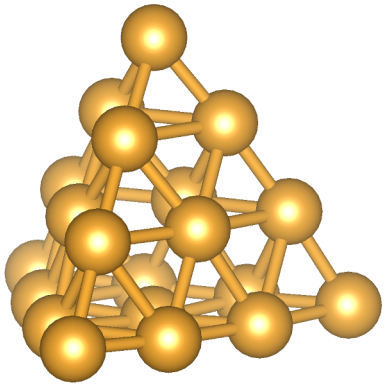
New code **BUFO** for efficient portable relativistic simulations oriented to model *d*- and *f*-element compounds and based on Gatchina pseudo-potentials



Introduction: BUFO

New code **BUFO** for efficient portable relativistic simulations oriented to model *d*- and *f*-element compounds and based on Gatchina pseudo-potentials

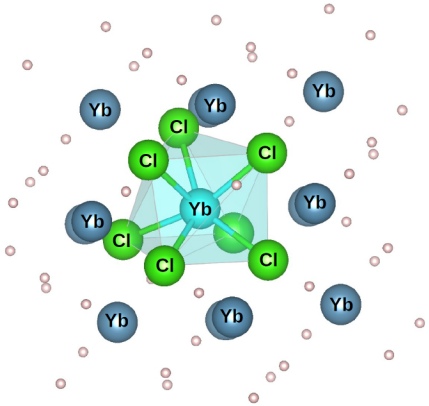


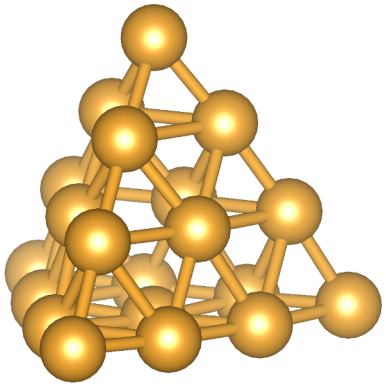


kokkos

Introduction: BUFO

New code **BUFO** for efficient portable relativistic simulations oriented to model *d*- and *f*-element compounds and based on Gatchina pseudo-potentials





kokkos

Implementation and benchmarking of algorithm combining Kokkos with MPI

(The previous BUFO MPI+OpenMP implemenation: https://superfri.org/index.php/superfri/article/view/699)

Plan

1. Introduction

Plan

1. Introduction

2. Resolution-of-the-identity Hartree—Fock method

Plan

1. Introduction

2. Resolution-of-the-identity Hartree—Fock method

3. Algorithm details

Plan

1. Introduction

2. Resolution-of-the-identity Hartree—Fock method

3. Algorithm details

4. Scalability benchmark. Discussion

Plan

1. Introduction
2. Resolution-of-the-identity Hartree—Fock method
3. Algorithm details
4. Scalability benchmark. Discussion
5. Performance benchmark. Discussion

Plan

1. Introduction
2. Resolution-of-the-identity Hartree—Fock method
3. Algorithm details
4. Scalability benchmark. Discussion
5. Performance benchmark. Discussion
6. Mesh generation utility

Plan

1. Introduction
2. Resolution-of-the-identity Hartree—Fock method
3. Algorithm details
4. Scalability benchmark. Discussion
5. Performance benchmark. Discussion
6. Mesh generation utility
7. Conclusions

RI Hartree—Fock Method

$$F_{\mu\nu} = H_{\mu\nu}^{core} + J_{\mu\nu} - K_{\mu\nu}, \quad \mu, \nu = \overline{1, N_{AO}} \tag{1}$$

RI Hartree—Fock Method

$$F_{\mu\nu} = H^{core}_{\mu\nu} + J_{\mu\nu} - K_{\mu\nu}, \;\; \mu, \nu = \overline{1, N_{AO}} \tag{1}$$

$$J_{\mu\nu} = \sum_{P}^{N_{aux}} \sum_{\rho\sigma}^{N_{AO}} (\mu\nu|\widetilde{P})(\widetilde{P}|\rho\sigma) D_{\rho\sigma}, \;\; P = \overline{1, N_{aux}} \tag{2}$$

$$K_{\mu\nu} = \sum_{\rho\sigma}^{N_{AO}} \sum_{P}^{N_{aux}} \sum_{i}^{N_{occ}} (\mu\sigma|\widetilde{P})(\widetilde{P}|\rho\nu) C_{i\sigma} C_{i\rho}, \;\; i = \overline{1, N_{occ}}, \tag{3}$$

RI Hartree—Fock Method

$$F_{\mu\nu} = H^{core}_{\mu\nu} + J_{\mu\nu} - K_{\mu\nu}, \;\; \mu, \nu = \overline{1, N_{AO}} \tag{1}$$

$$J_{\mu\nu} = \sum_{P}^{N_{aux}} \sum_{\rho\sigma}^{N_{AO}} (\mu\nu|\widetilde{P})(\widetilde{P}|\rho\sigma) D_{\rho\sigma}, \;\; P = \overline{1, N_{aux}} \tag{2}$$

$$K_{\mu\nu} = \sum_{\rho\sigma}^{N_{AO}} \sum_{P}^{N_{aux}} \sum_{i}^{N_{occ}} (\mu\widetilde{\sigma}|P)(P|\widetilde{\rho}\nu) C_{i\sigma} C_{i\rho}, \;\; i = \overline{1, N_{occ}}, \tag{3}$$

$$(\mu\widetilde{\nu}|P) = (\nu\widetilde{\mu}|P) \tag{4}$$

$$N_{occ} < N_{AO} < N_{aux} \tag{5}$$

RI Hartree—Fock Method

$$F_{\mu\nu} = H^{core}_{\mu\nu} + J_{\mu\nu} - K_{\mu\nu}, \quad \mu, \nu = \overline{1, N_{AO}} \tag{1}$$

$O(N^3_{AO}) \sim O(N^3)$

$$J_{\mu\nu} = \sum_{P}^{N_{aux}} \sum_{\rho\sigma}^{N_{AO}} (\mu\nu|\widetilde{P})(\widetilde{P}|\rho\sigma) D_{\rho\sigma}, \quad P = \overline{1, N_{aux}} \tag{2}$$

$O(N_{aux} N^2_{AO}) \sim O(N^3)$

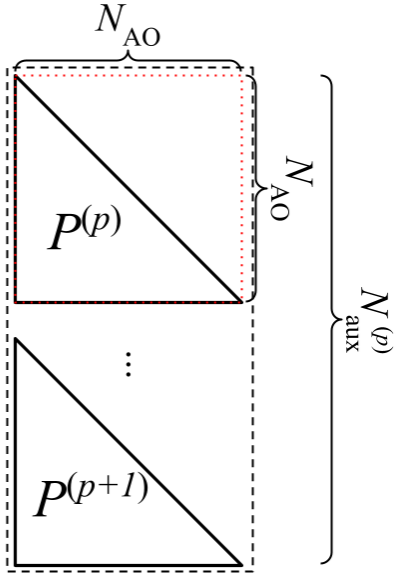
$$K_{\mu\nu} = \sum_{\rho\sigma}^{N_{AO}} \sum_{P}^{N_{aux}} \sum_{i}^{N_{occ}} (\mu\sigma|\widetilde{P})(\widetilde{P}|\rho\nu) C_{i\sigma} C_{i\rho}, \quad i = \overline{1, N_{occ}}, \tag{3}$$

$O(N_{occ} N_{aux} N^2_{AO}) \sim O(N^4)$

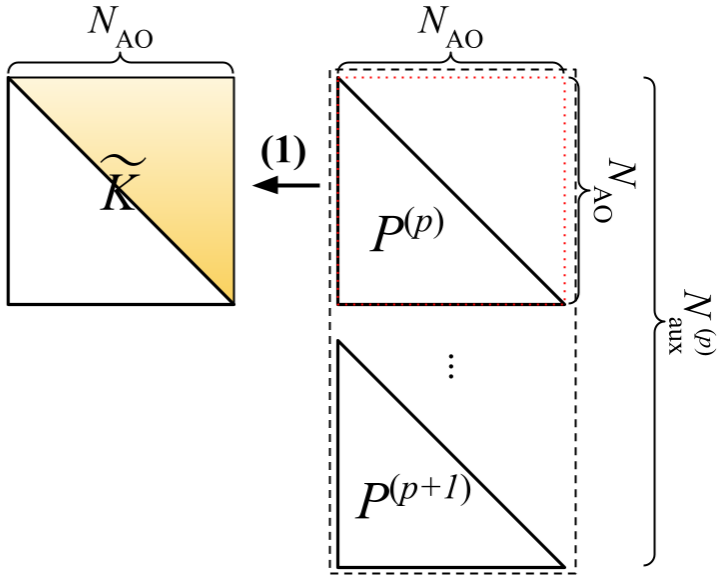
$$(\mu\nu|\widetilde{P}) = (\nu\mu|\widetilde{P}) \tag{4}$$

$$N_{occ} < N_{AO} < N_{aux} \tag{5}$$

Algorithm Scheme

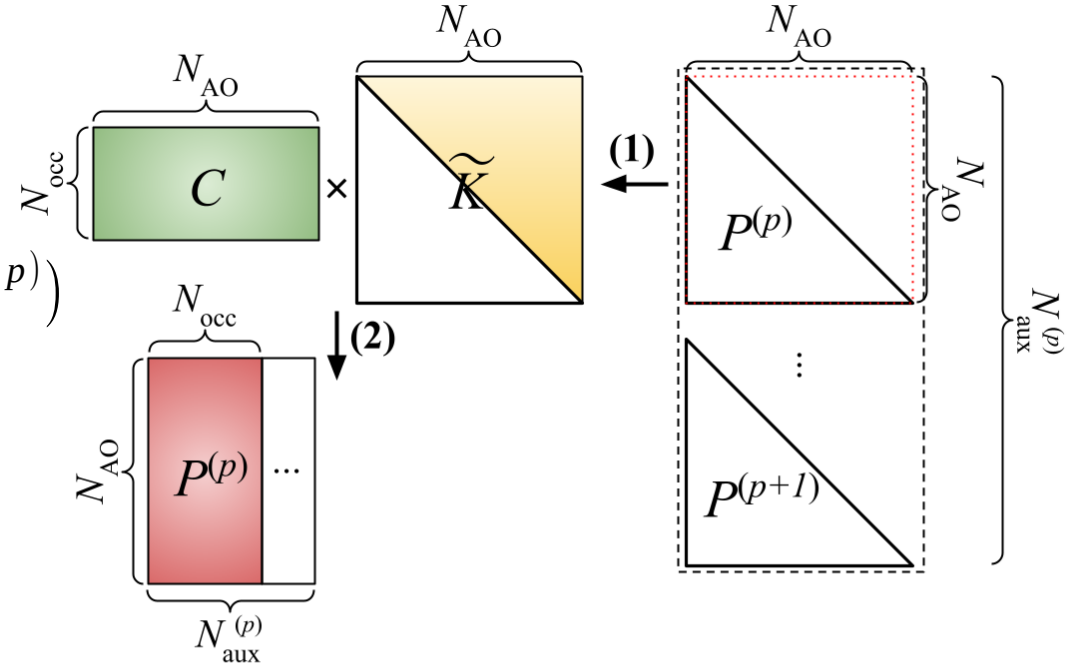


Algorithm Scheme



Algorithm Scheme

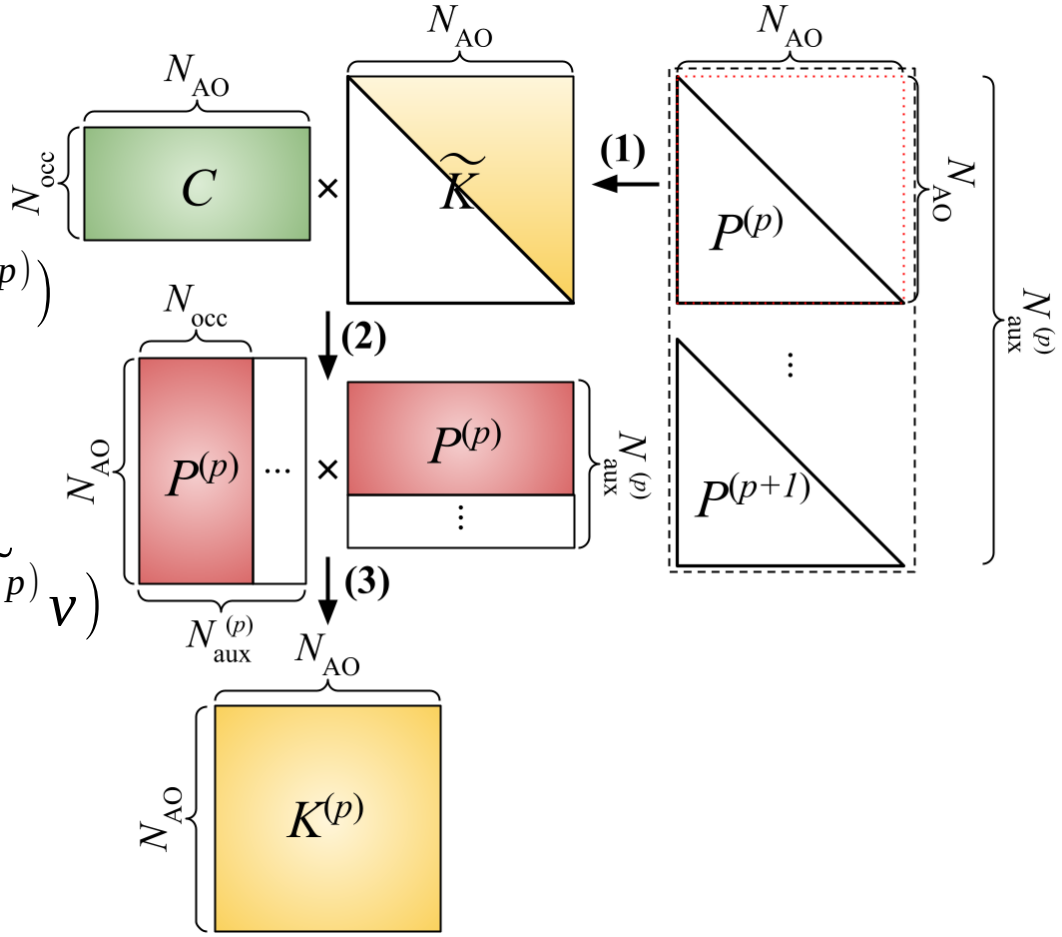
$$\left(\mu i|\widetilde{P}^{(p)}\right)=\sum_{\sigma}^{N_{AO}} C_{i\sigma}\left(\mu\sigma|\widetilde{P}^{(p)}\right)$$



Algorithm Scheme

$$\left(\mu i|\widetilde{P}^{(p)}\right)=\sum_{\sigma}^{N_{AO}} C_{i\sigma}\left(\mu\sigma|\widetilde{P}^{(p)}\right)$$

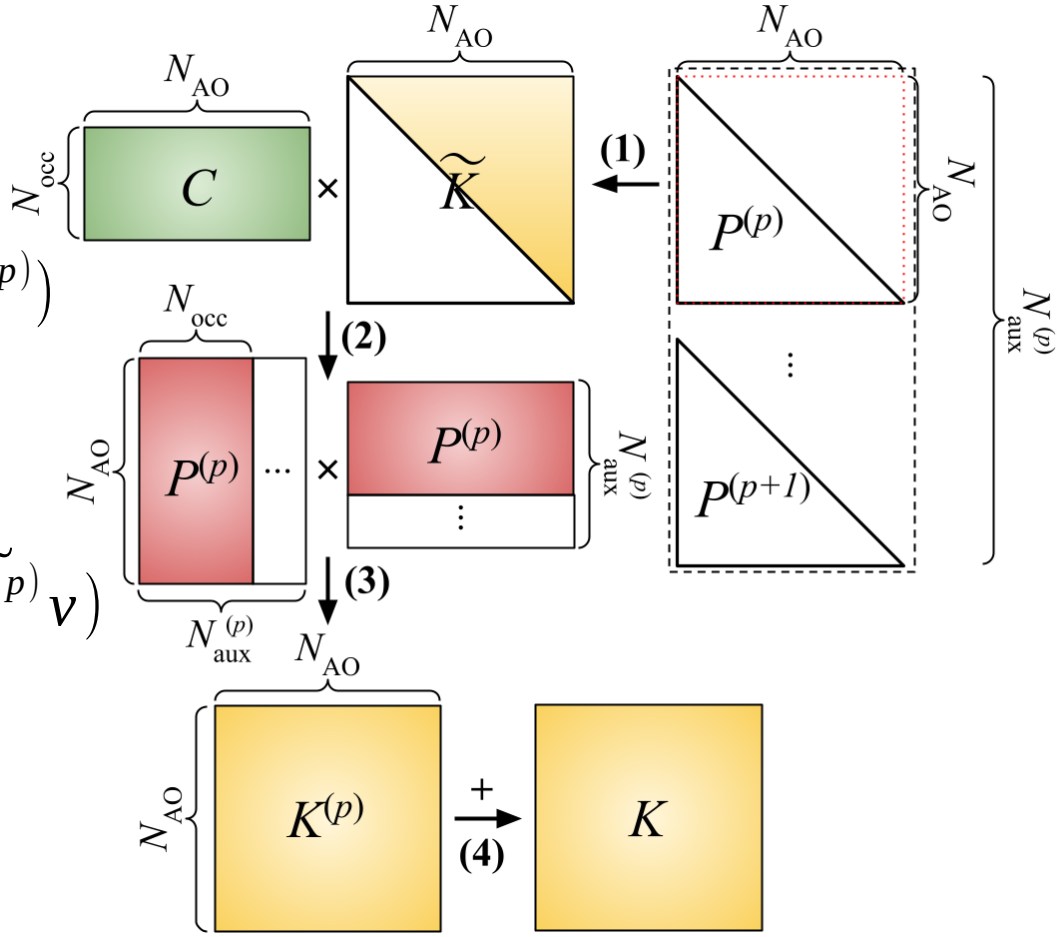
$$K_{\mu\nu}^{(p)}=\sum_{i\times P}^{N_{occ}\times N_{aux}}\left(\mu i|\widetilde{P}^{(p)}\right)\left(i|\widetilde{P}^{(p)}\nu\right)$$



Algorithm Scheme

$$\left(\mu i|\widetilde{P}^{(p)}\right)=\sum_{\sigma}^{N_{AO}} C_{i\sigma}\left(\mu\sigma|\widetilde{P}^{(p)}\right)$$

$$K_{\mu\nu}^{(p)}=\sum_{i\times P}^{N_{occ}\times N_{aux}}\left(\mu i|\widetilde{P}^{(p)}\right)\left(i|\widetilde{P}^{(p)}\nu\right)$$



Algorithm Pseudocode

for $P = \overline{P^{(p)}, \ldots, P^{(p+1)}}$ **do**

  `Kokkos::parallel_for` $[\tilde{K}^{P}_{\mu\sigma} \leftarrow \widetilde{(\mu\sigma|P)},\ \sigma \leq \mu]$

  $\widetilde{(\mu i|P)} \leftarrow$ `KokkosBlas::gemm`$[C_{i\sigma}, \tilde{K}^{P}_{\mu\sigma}]$

end for

$K^{(p)}_{\mu\nu} \leftarrow$ `KokkosBlas::gemm`$[\widetilde{(\mu i|P^{(p)})}, \widetilde{(\nu i|P^{(p)})}]$

$K_{\mu\nu} \leftarrow$ `MPI_Allreduce`$[K^{(p)}_{\mu\nu}]$

Algorithm Pseudocode

for $P = \overline{P^{(p)}, \ldots, P^{(p+1)}}$ **do** **~ (3.4k/p)**

`Kokkos::parallel_for` $[\tilde{K}^P_{\mu\sigma} \leftarrow \widetilde{(\mu\sigma|P)},\ \sigma \leq \mu]$

$\widetilde{(\mu i|P)} \leftarrow$ `KokkosBlas::gemm`$[C_{i\sigma}, \tilde{K}^P_{\mu\sigma}]$

end for

$K^{(p)}_{\mu\nu} \leftarrow$ `KokkosBlas::gemm`$[\widetilde{(\mu i|P^{(p)})}, \widetilde{(\nu i|P^{(p)})}]$

$K_{\mu\nu} \leftarrow$ `MPI_Allreduce`$[K^{(p)}_{\mu\nu}]$

Algorithm Pseudocode

for $P = \overline{P^{(p)}, \ldots, P^{(p+1)}}$ **do** **~ (3.4k/p)**

(1) `Kokkos::parallel_for` $[\tilde{K}^P_{\mu\sigma} \leftarrow \widetilde{(\mu\sigma|P)},\ \sigma \leq \mu]$ **~ (1.4k x 1.4k)**

$\widetilde{(\mu i|P)} \leftarrow$ `KokkosBlas::gemm`$[C_{i\sigma}, \tilde{K}^P_{\mu\sigma}]$

end for

$K^{(p)}_{\mu\nu} \leftarrow$ `KokkosBlas::gemm`$[\widetilde{(\mu i|P^{(p)})}, \widetilde{(\nu i|P^{(p)})}]$

$K_{\mu\nu} \leftarrow$ `MPI_Allreduce`$[K^{(p)}_{\mu\nu}]$

Algorithm Pseudocode

for $P = \overline{P^{(p)}, \ldots, P^{(p+1)}}$ **do** **~ (3.4k/p)**

(1) `Kokkos::parallel_for` $[\tilde{K}^P_{\mu\sigma} \leftarrow \widetilde{(\mu\sigma|P)},\ \sigma \leq \mu]$ **~ (1.4k x 1.4k)**

(2) $\widetilde{(\mu i|P)} \leftarrow$ `KokkosBlas::gemm`$[C_{i\sigma}, \tilde{K}^P_{\mu\sigma}]$

end for **~ (0.2k x 1.4k) @ (1.4k x 1.4k)**

$K^{(p)}_{\mu\nu} \leftarrow$ `KokkosBlas::gemm`$[\widetilde{(\mu i|P^{(p)})}, \widetilde{(\nu i|P^{(p)})}]$

$K_{\mu\nu} \leftarrow$ `MPI_Allreduce`$[K^{(p)}_{\mu\nu}]$

Algorithm Pseudocode

for $P = \overline{P^{(p)}, \ldots, P^{(p+1)}}$ **do** **~ (3.4k/p)**

(1) `Kokkos::parallel_for` $[\tilde{K}^P_{\mu\sigma} \leftarrow \widetilde{(\mu\sigma|P)},\ \sigma \leq \mu]$ **~ (1.4k x 1.4k)**

(2) $\widetilde{(\mu i|P)} \leftarrow$ `KokkosBlas::gemm`$[C_{i\sigma}, \tilde{K}^P_{\mu\sigma}]$

end for **~ (0.2k x 1.4k) @ (1.4k x 1.4k)**

(3) $K^{(p)}_{\mu\nu} \leftarrow$ `KokkosBlas::gemm`$[\widetilde{(\mu i|P^{(p)})}, \widetilde{(\nu i|P^{(p)})}]$

$K_{\mu\nu} \leftarrow$ `MPI_Allreduce`$[K^{(p)}_{\mu\nu}]$ **~ (1.4k x 540k/p) @ (540k/p x 1.4k)**

Algorithm Pseudocode

for $P = \overline{P^{(p)}, \ldots, P^{(p+1)}}$ **do** **~ (3.4k/p)**

(1) `Kokkos::parallel_for` $[\tilde{K}^P_{\mu\sigma} \leftarrow \widetilde{(\mu\sigma|P)},\ \sigma \leq \mu]$ **~ (1.4k x 1.4k)**

(2) $\widetilde{(\mu i|P)} \leftarrow$ `KokkosBlas::gemm`$[C_{i\sigma}, \tilde{K}^P_{\mu\sigma}]$ **~ (0.2k x 1.4k) @ (1.4k x 1.4k)**

end for

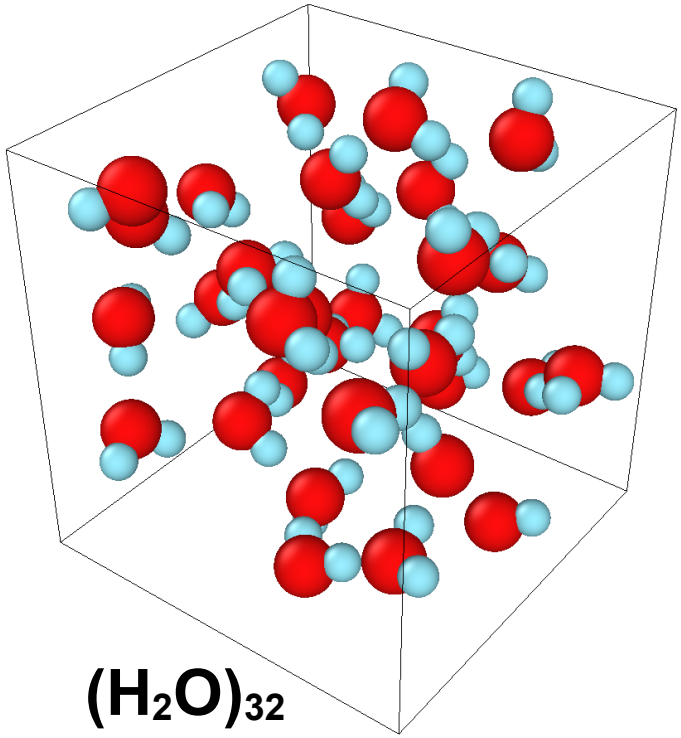
(3) $K^{(p)}_{\mu\nu} \leftarrow$ `KokkosBlas::gemm`$[\widetilde{(\mu i|P^{(p)})}, \widetilde{(\nu i|P^{(p)})}]$

(4) $K_{\mu\nu} \leftarrow$ `MPI_Allreduce`$[K^{(p)}_{\mu\nu}]$ **~ (1.4k x 540k/p) @ (540k/p x 1.4k)**

~ (1.4k x 1.4k)

Computational Details I

Test series I

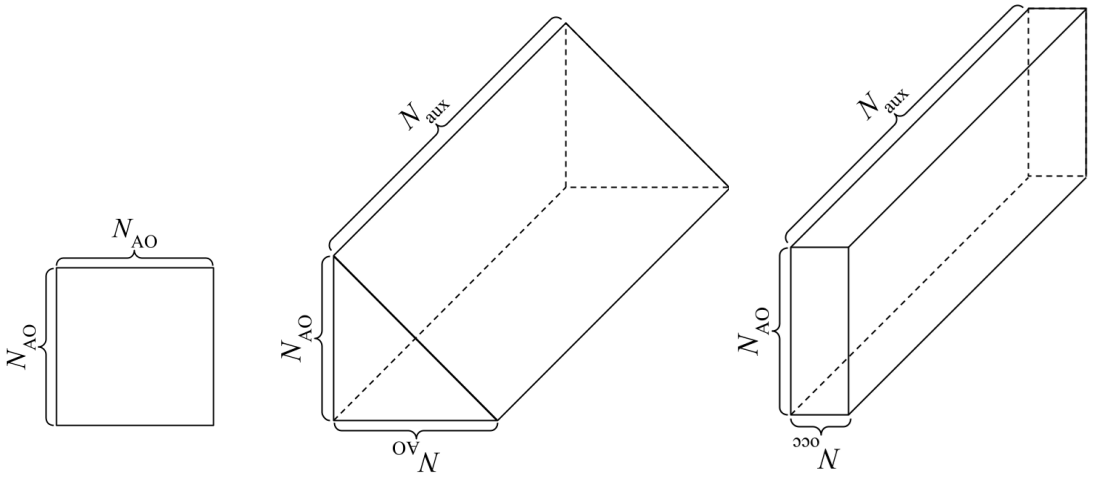


$(H_2O)_{32}$

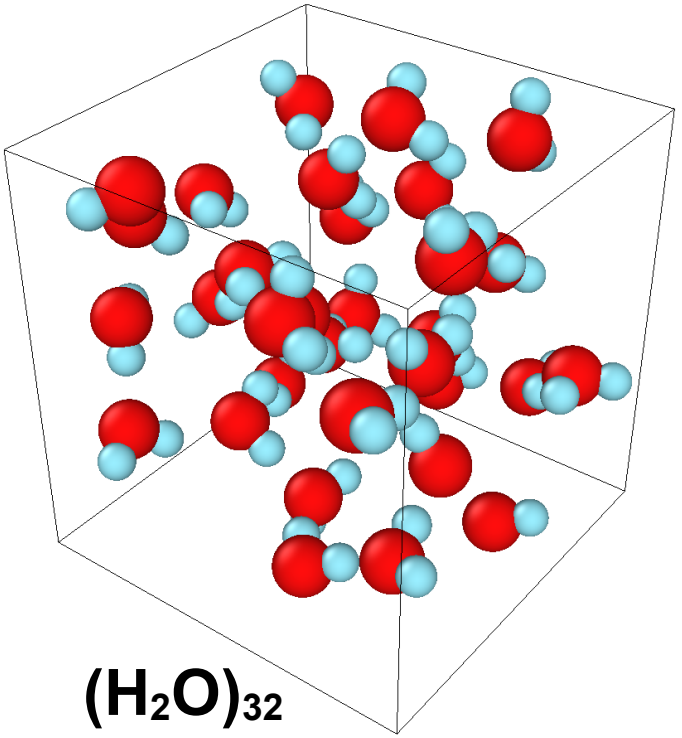
Computational Details I

Test series I

$N_{occ} = 160,\ N_{AO} = 1376,\ N_{aux} = 3392$

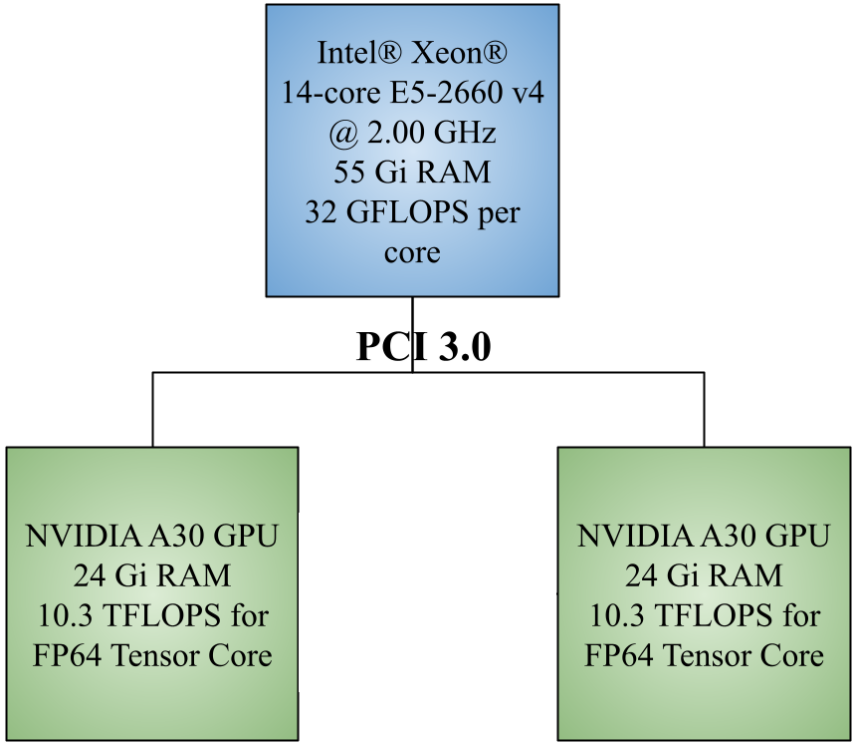


~ 30 Gi



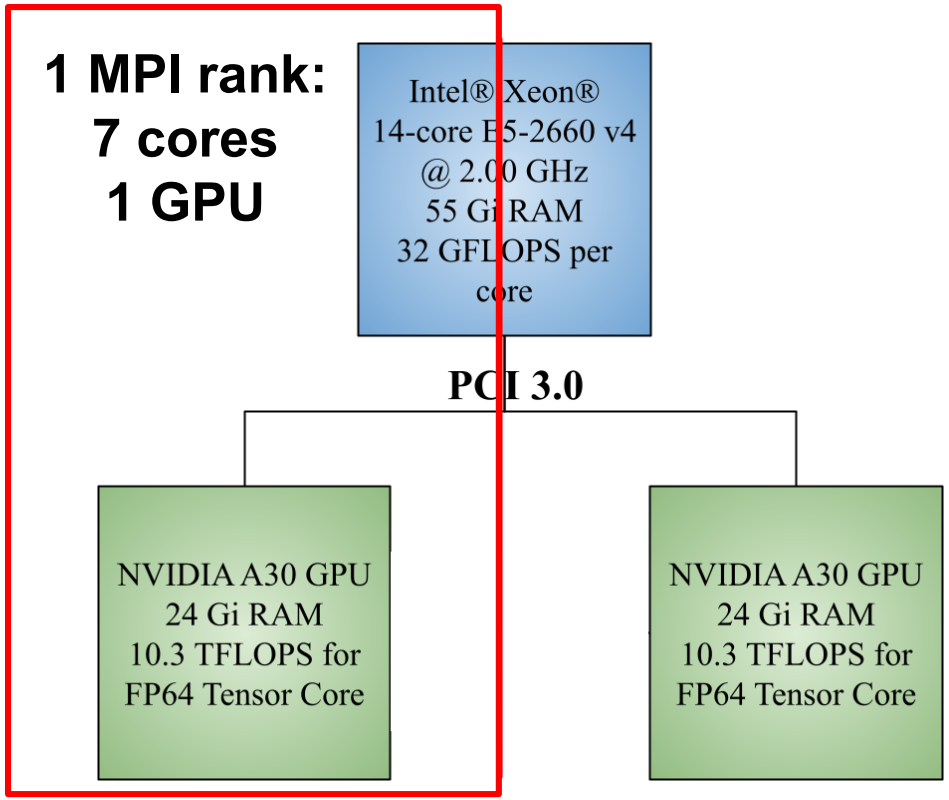
$(H_2O)_{32}$

Computational Details I



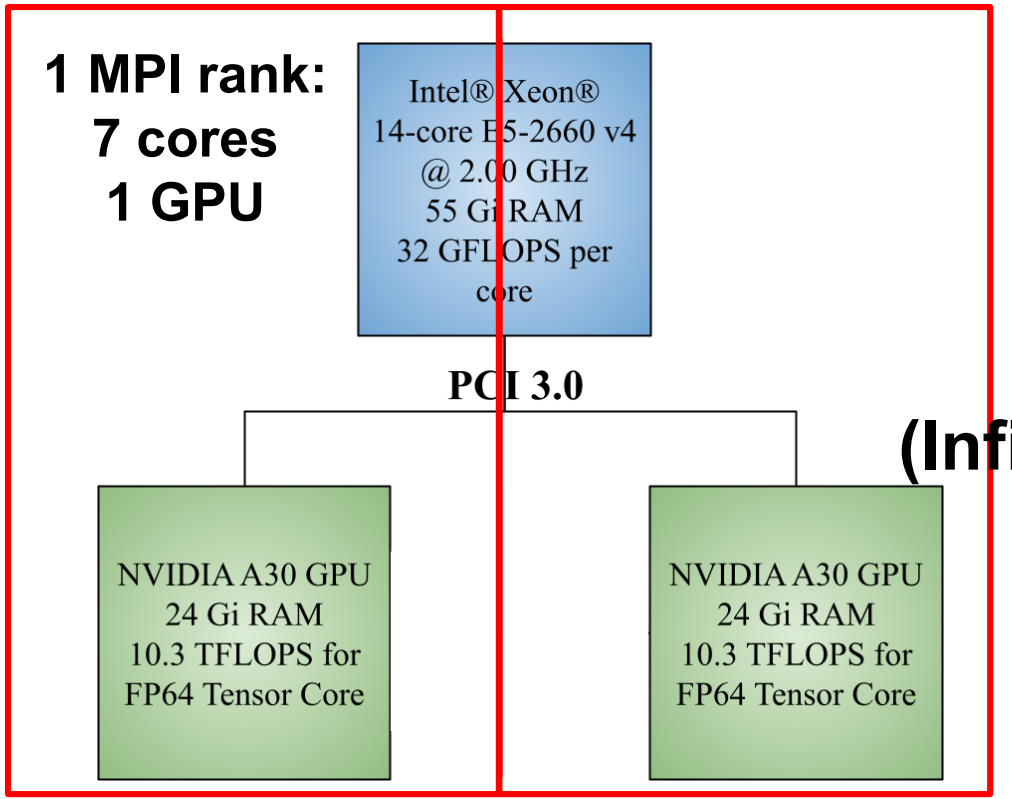
Scheme 1: Intel Xeon E5-2660 v4 host (OpenMP 4.5) +
2x GPU A30 device (CUDA 12.9)

Computational Details I



Scheme 1: Intel Xeon E5-2660 v4 host (OpenMP 4.5) +
2x GPU A30 device (CUDA 12.9)

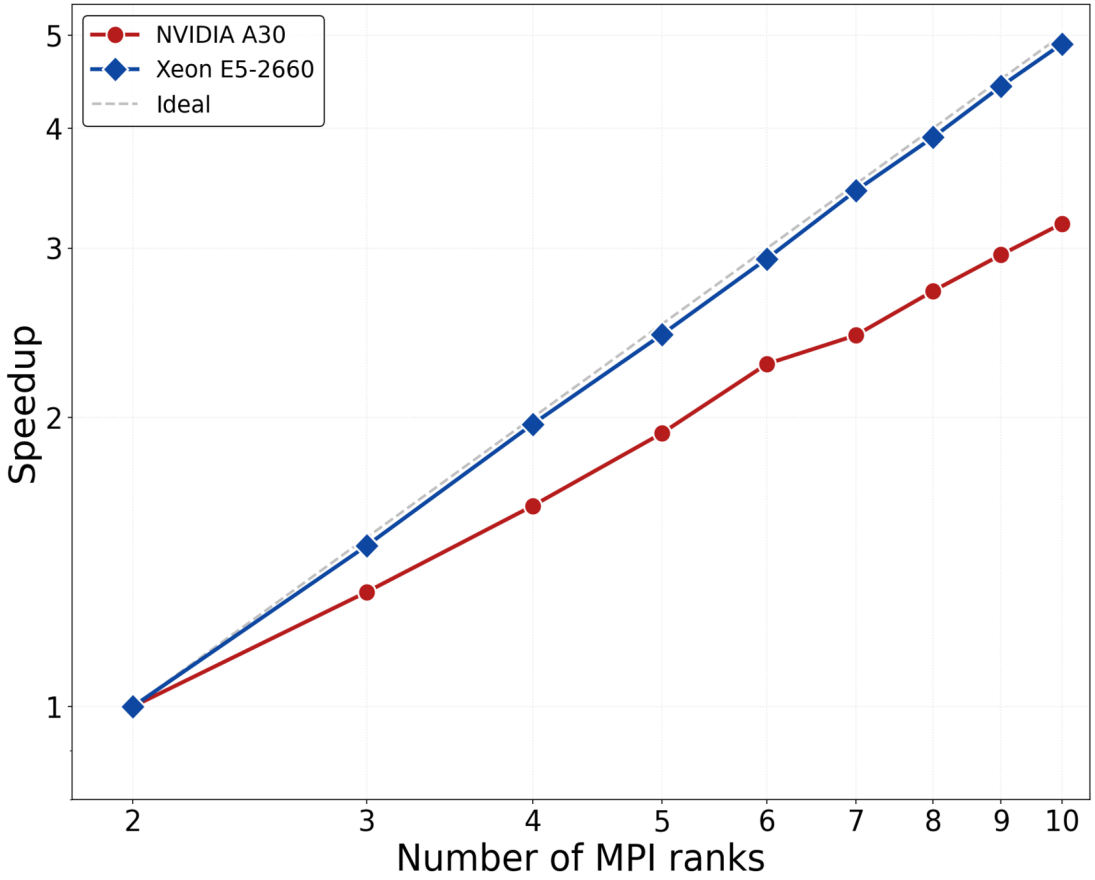
Computational Details I



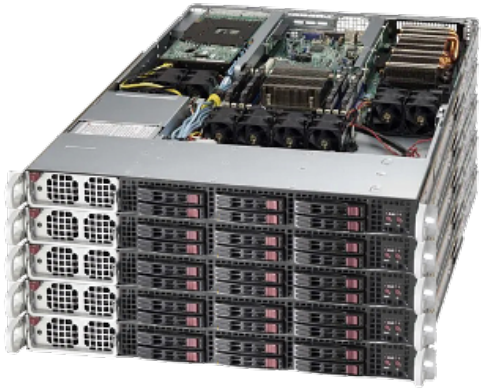
x 5 nodes (InfiniBand FDR)

Scheme 1: Intel Xeon E5-2660 v4 host (OpenMP 4.5) + 2x GPU A30 device (CUDA 12.9)

Benchmark I: Speedup

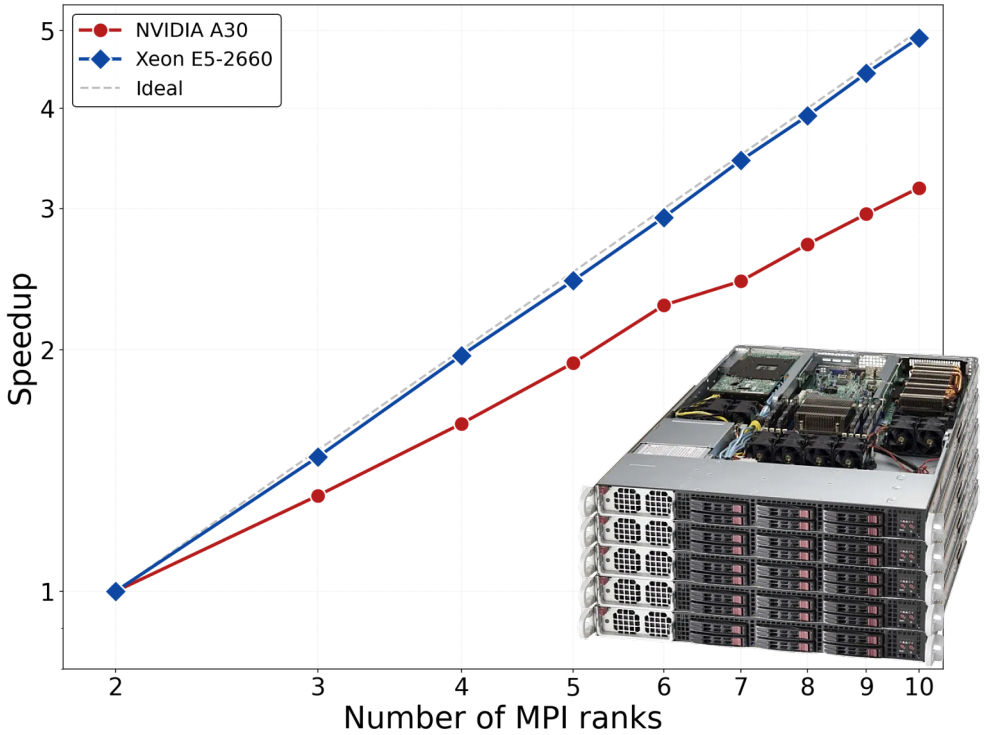


1 node 2 nodes 3 nodes 4 nodes 5 nodes



5 nodes with
Intel Xeon E5-2660 v4 +
2x GPU Nvidia A30

Benchmark I: Speedup

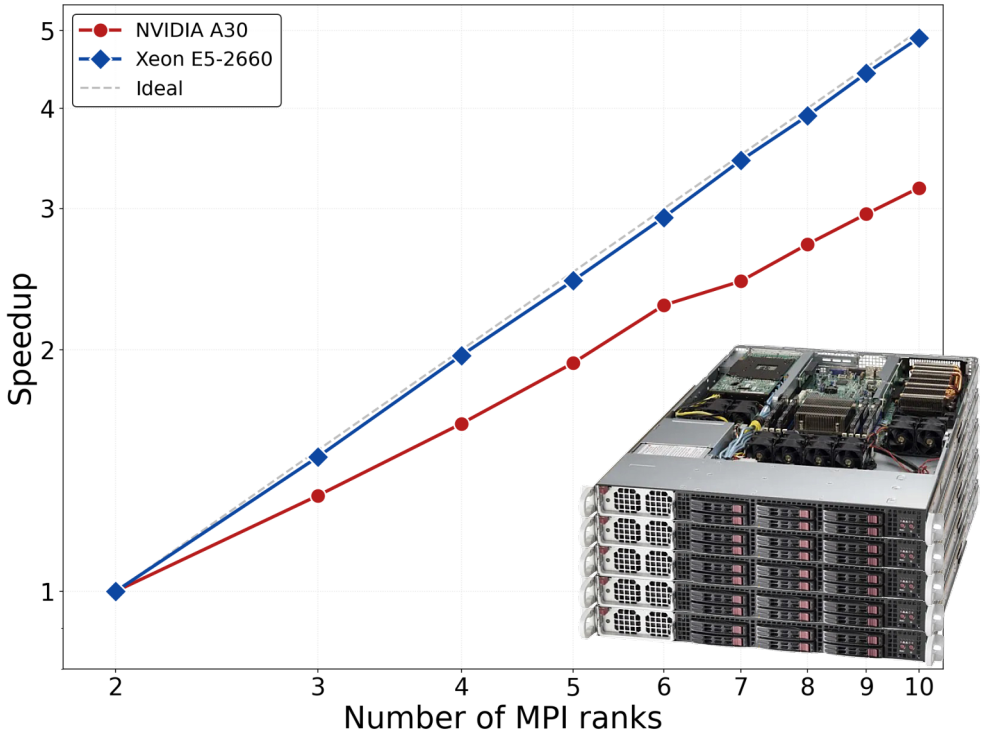


1 node 2 nodes 3 nodes 4 nodes 5 nodes

Approximate percentage distribution of time between steps of the Algorithm

	Intel Xeon E5-2660				NVIDIA A30			
N_p	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
2	34	42	24	< 1.0	35	41	22	1.7
3	35	41	23	< 1.0	33	37	19	11
4	35	41	23	< 1.0	33	38	19	11
5	36	41	23	< 1.0	31	36	18	15
6	36	41	23	< 1.0	32	37	20	11
7	36	41	23	< 1.0	29	34	19	18
8	36	41	23	< 1.0	29	33	20	18
9	35	41	23	< 1.0	28	32	20	20
10	36	41	23	< 1.0	27	32	19	22

Benchmark I: Speedup



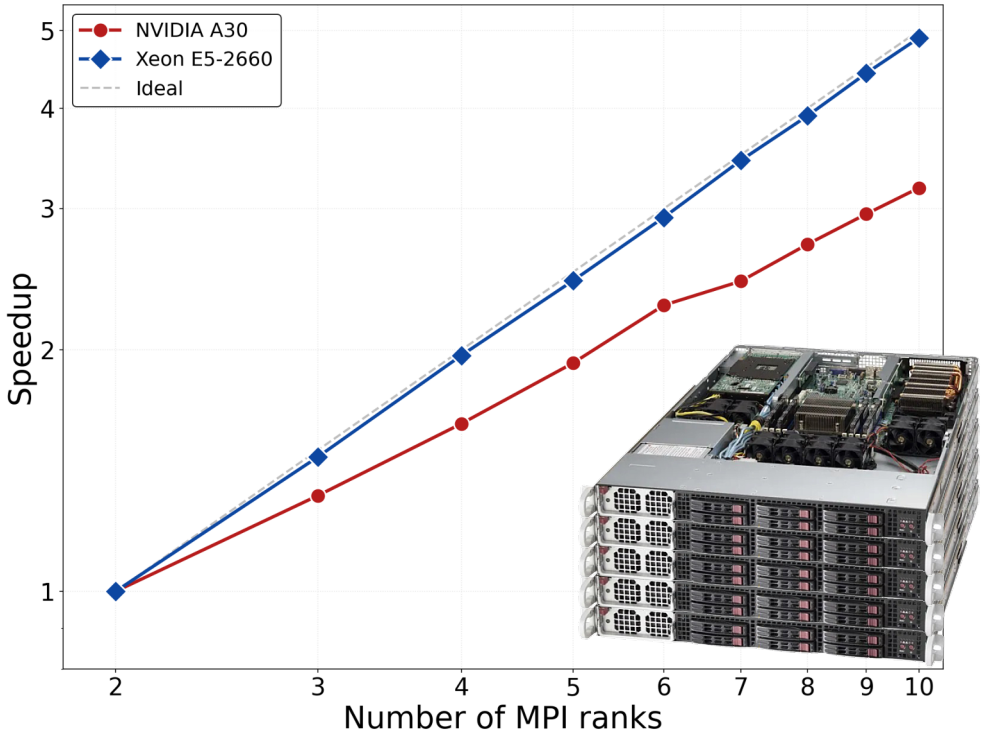
1 node 2 nodes 3 nodes 4 nodes 5 nodes

Approximate percentage distribution of time between steps of the Algorithm

	Intel Xeon E5-2660				NVIDIA A30			
N_p	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
2	34	42	24	< 1.0	35	41	22	1.7
3	35	41	23	< 1.0	33	37	19	11
4	35	41	23	< 1.0	33	38	19	11
5	36	41	23	< 1.0	31	36	18	15
6	36	41	23	< 1.0	32	37	20	11
7	36	41	23	< 1.0	29	34	19	18
8	36	41	23	< 1.0	29	33	20	18
9	35	41	23	< 1.0	28	32	20	20
10	36	41	23	< 1.0	27	32	19	22

Suboptimal unpacking

Benchmark I: Speedup



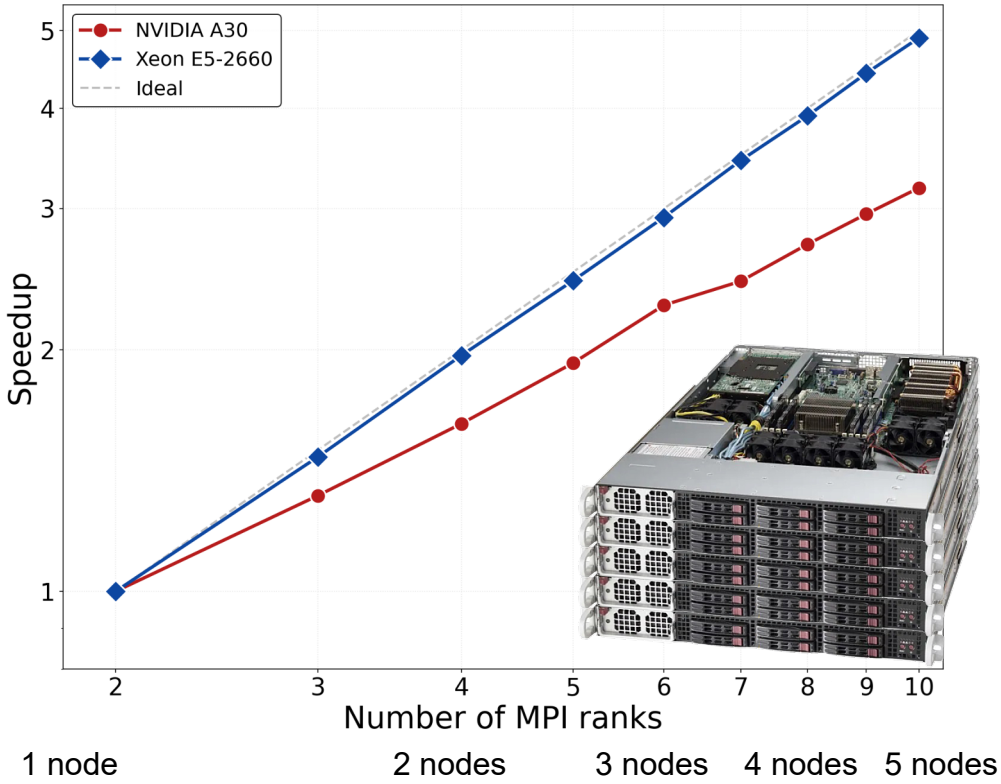
1 node 2 nodes 3 nodes 4 nodes 5 nodes

Approximate percentage distribution of time between steps of the Algorithm

	Intel Xeon E5-2660				NVIDIA A30			
N_p	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
2	34	42	24	< 1.0	35	41	22	1.7
3	35	41	23	< 1.0	33	37	19	11
4	35	41	23	< 1.0	33	38	19	11
5	36	41	23	< 1.0	31	36	18	15
6	36	41	23	< 1.0	32	37	20	11
7	36	41	23	< 1.0	29	34	19	18
8	36	41	23	< 1.0	29	33	20	18
9	35	41	23	< 1.0	28	32	20	20
10	36	41	23	< 1.0	27	32	19	22

Ideally expected to be similar

Benchmark I: Speedup



1 node 2 nodes 3 nodes 4 nodes 5 nodes

Approximate percentage distribution of time between steps of the Algorithm

	Intel Xeon E5-2660				NVIDIA A30			
N_p	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
2	34	42	24	< 1.0	35	41	22	1.7
3	35	41	23	< 1.0	33	37	19	11
4	35	41	23	< 1.0	33	38	19	11
5	36	41	23	< 1.0	31	36	18	15
6	36	41	23	< 1.0	32	37	20	11
7	36	41	23	< 1.0	29	34	19	18
8	36	41	23	< 1.0	29	33	20	18
9	35	41	23	< 1.0	28	32	20	20
10	36	41	23	< 1.0	27	32	19	22

MPI_Allreduce bottleneck

Discussion: MPI_Allreduce Bottleneck

Reason: GPUs are fast enough to early stack with the communication bottleneck within MPI_Allreduce

Discussion: MPI_Allreduce Bottleneck

Reason: GPUs are fast enough to early stack with the communication bottleneck within MPI_Allreduce

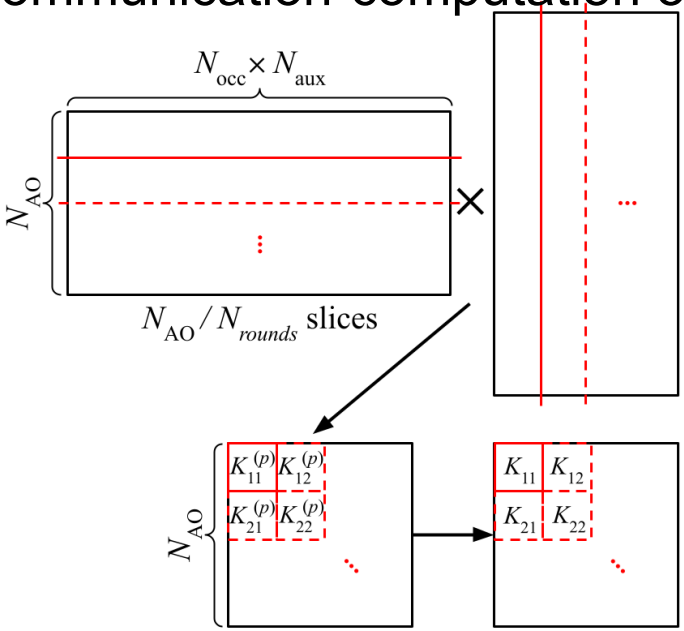


Communication-computation overlap

Discussion: MPI_Allreduce Bottleneck

Reason: GPUs are fast enough to early stack with the communication bottleneck within MPI_Allreduce

Communication-computation overlap

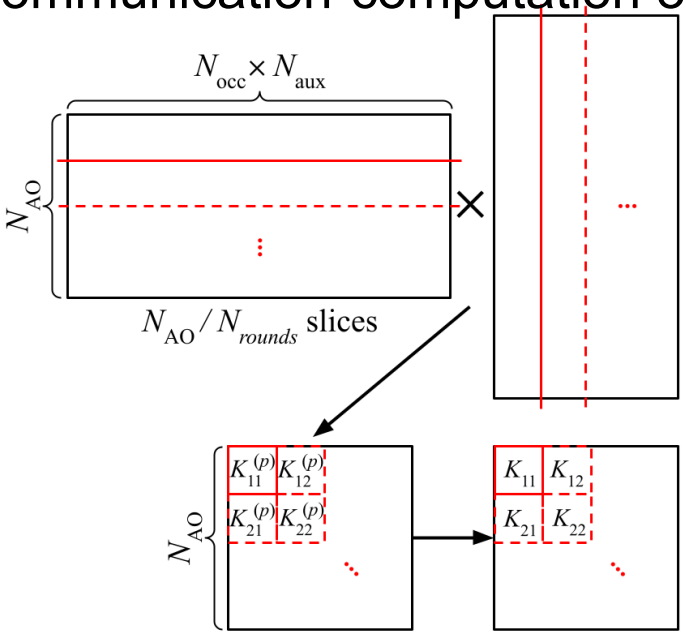


Discussion: MPI_Allreduce Bottleneck

Reason: GPUs are fast enough to early stack with the communication bottleneck within MPI_Allreduce

Communication-computation overlap

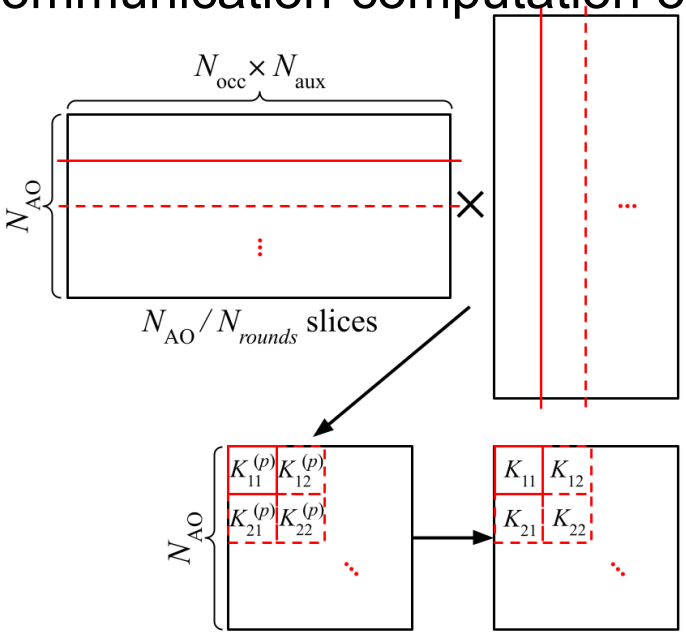
$$T_x \approx \max\left\{\alpha \frac{N^4}{p}, \beta N^2\right\}$$



Discussion: MPI_Allreduce Bottleneck

Reason: GPUs are fast enough to early stack with the communication bottleneck within MPI_Allreduce

Communication-computation overlap



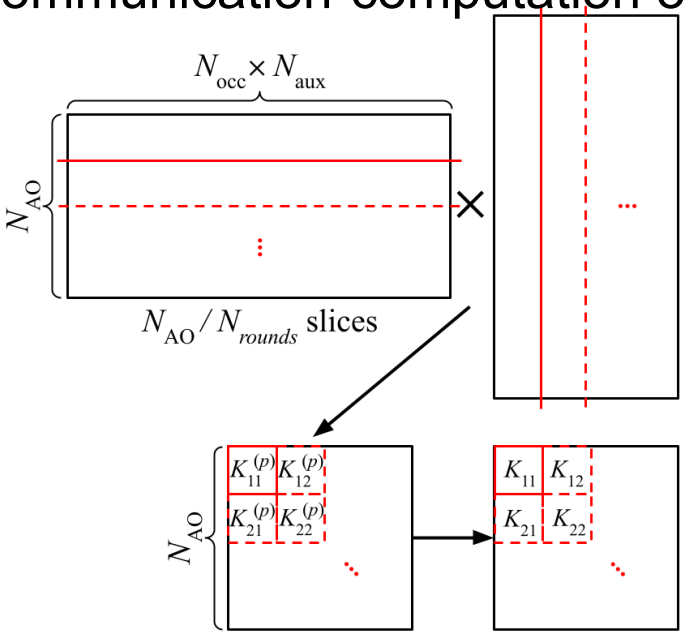
$$T_x \approx \max\left\{\alpha \frac{N^4}{p}, \beta N^2\right\}$$

Other ($\mu\nu|P$) partition between ranks to change communication pattern

Discussion: MPI_Allreduce Bottleneck

Reason: GPUs are fast enough to early stack with the communication bottleneck within MPI_Allreduce

Communication-computation overlap



$$T_x \approx \max\left\{\alpha \frac{N^4}{p}, \beta N^2\right\}$$

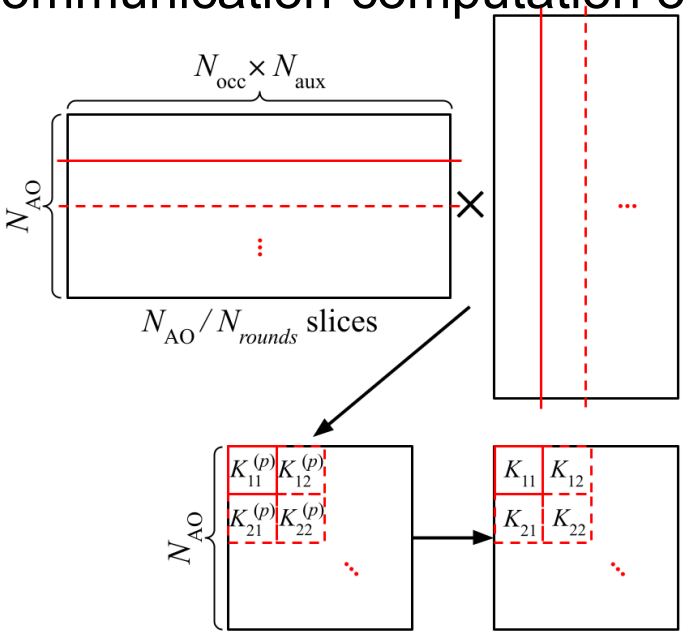
Other ($\mu\nu|P$) partition between ranks to change communication pattern

Requires communicaton model

Discussion: MPI_Allreduce Bottleneck

Reason: GPUs are fast enough to early stack with the communication bottleneck within MPI_Allreduce

Communication-computation overlap



$$T_x \approx \max\left\{\alpha \frac{N^4}{p}, \beta N^2\right\}$$

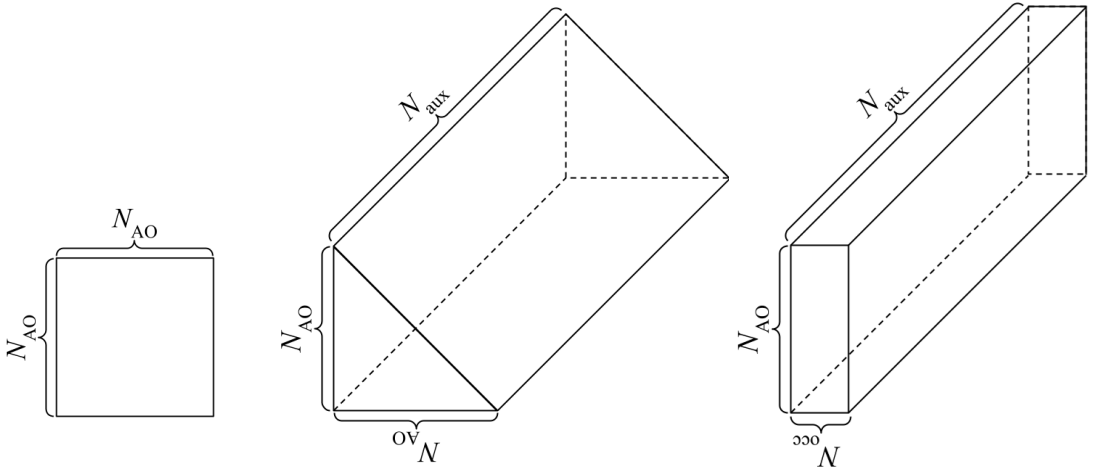
Other ($\mu\nu|P$) partition between ranks to change communication pattern

Requires communicaton model aware of different RI-HF stages as well

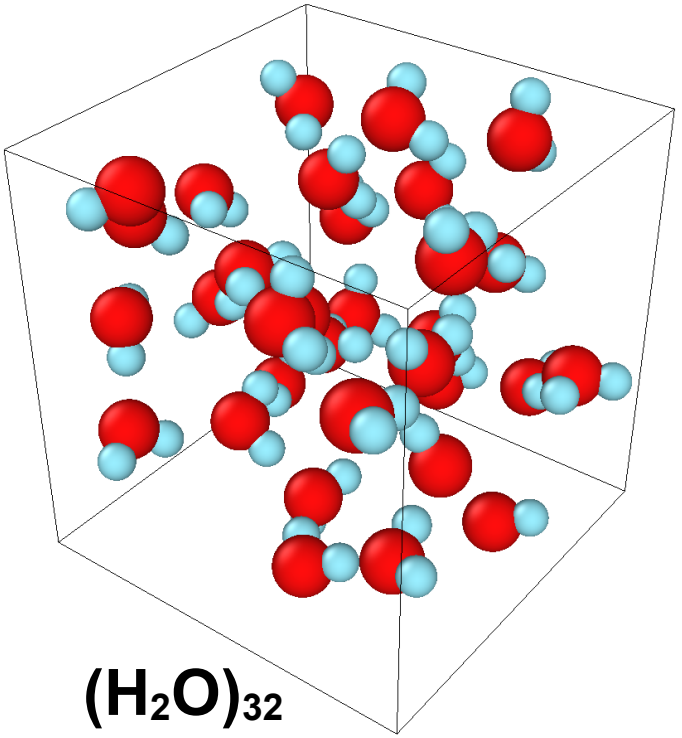
Computational Details II

Test series II

$N_{occ} = 160,\ N_{AO} = 2368,\ N_{aux} = 3616$

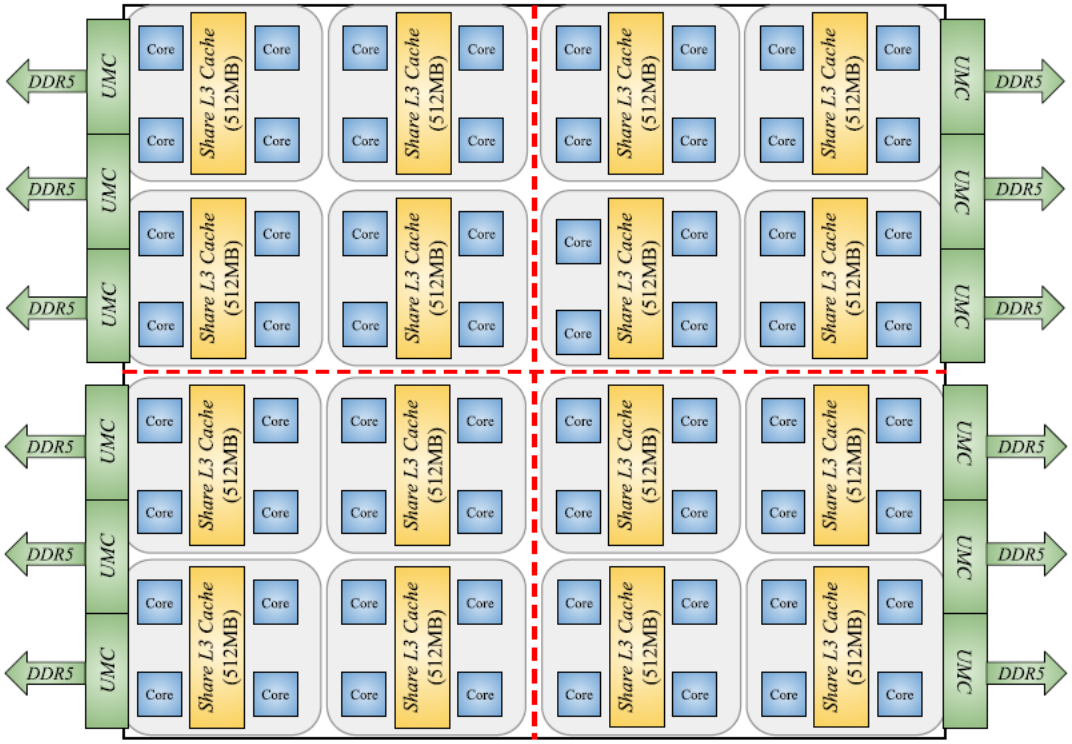


~ 100 Gi



$(H_2O)_{32}$

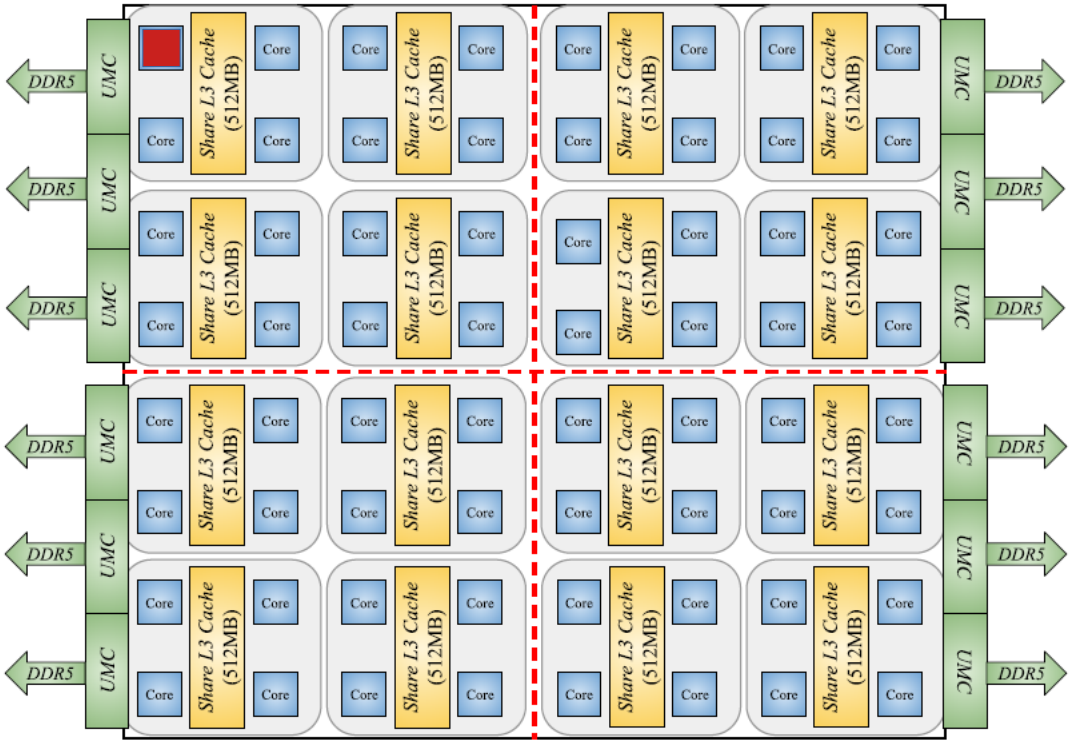
Computational Details II



Scheme 2: 64-core AMD EPYC 9535 processor (OpenMP 4.5), 38.4 GFLOPS per core

Computational Details II

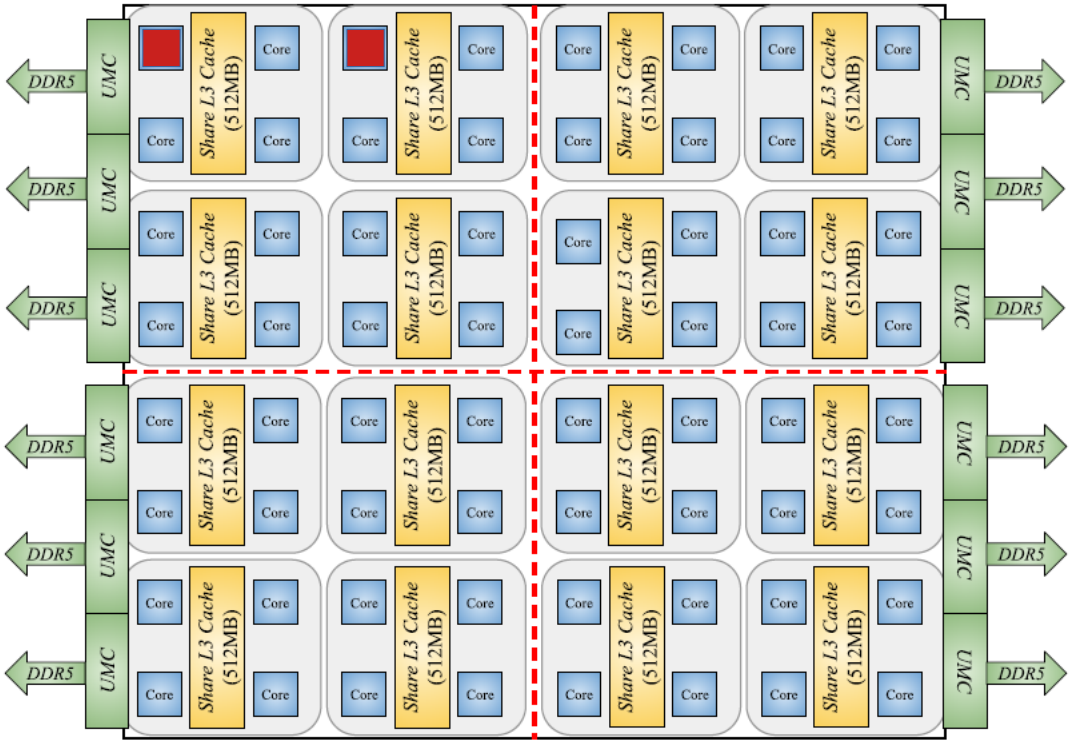
Thread is bound to core



Scheme 2: 64-core AMD EPYC 9535 processor (OpenMP 4.5), 38.4 GFLOPS per core

Computational Details II

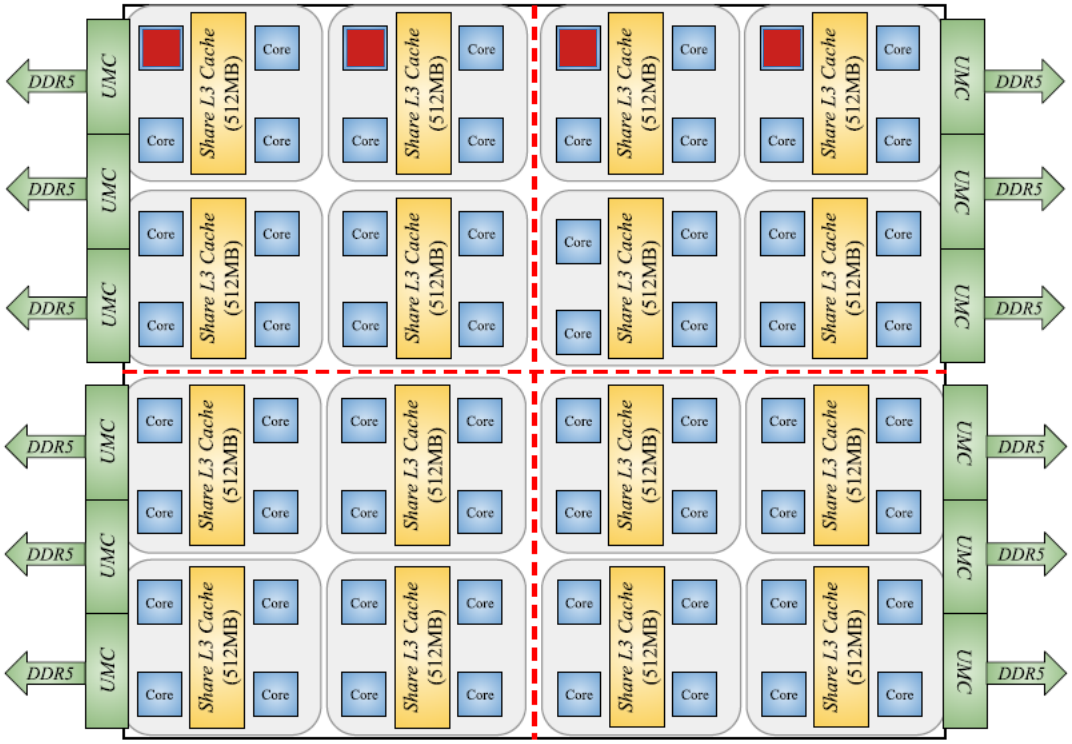
Thread is bound to core



Scheme 2: 64-core AMD EPYC 9535 processor (OpenMP 4.5), 38.4 GFLOPS per core

Computational Details II

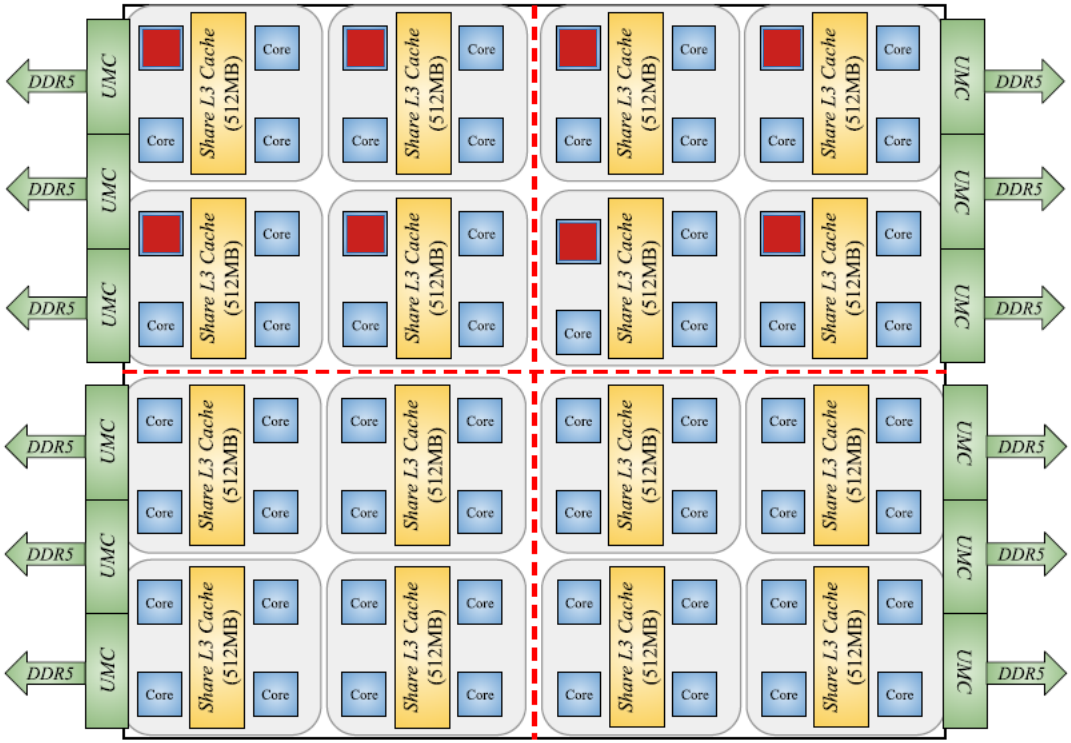
Thread is bound to core



Scheme 2: 64-core AMD EPYC 9535 processor (OpenMP 4.5), 38.4 GFLOPS per core

Computational Details II

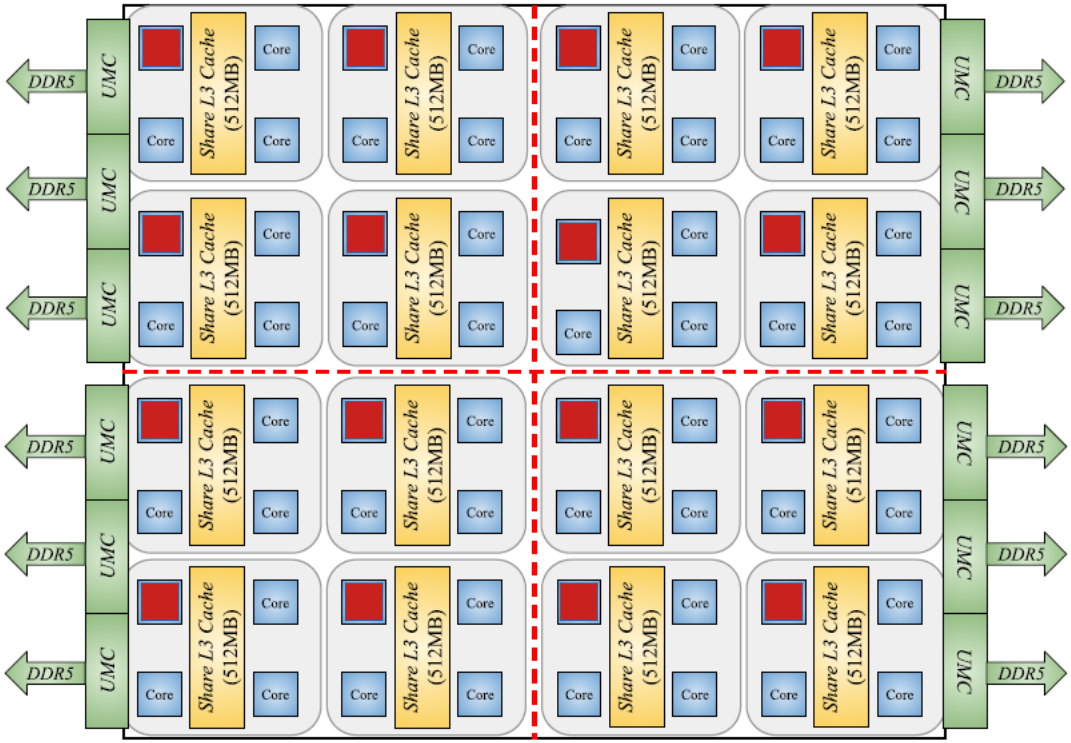
Thread is bound to core



Scheme 2: 64-core AMD EPYC 9535 processor (OpenMP 4.5), 38.4 GFLOPS per core

Computational Details II

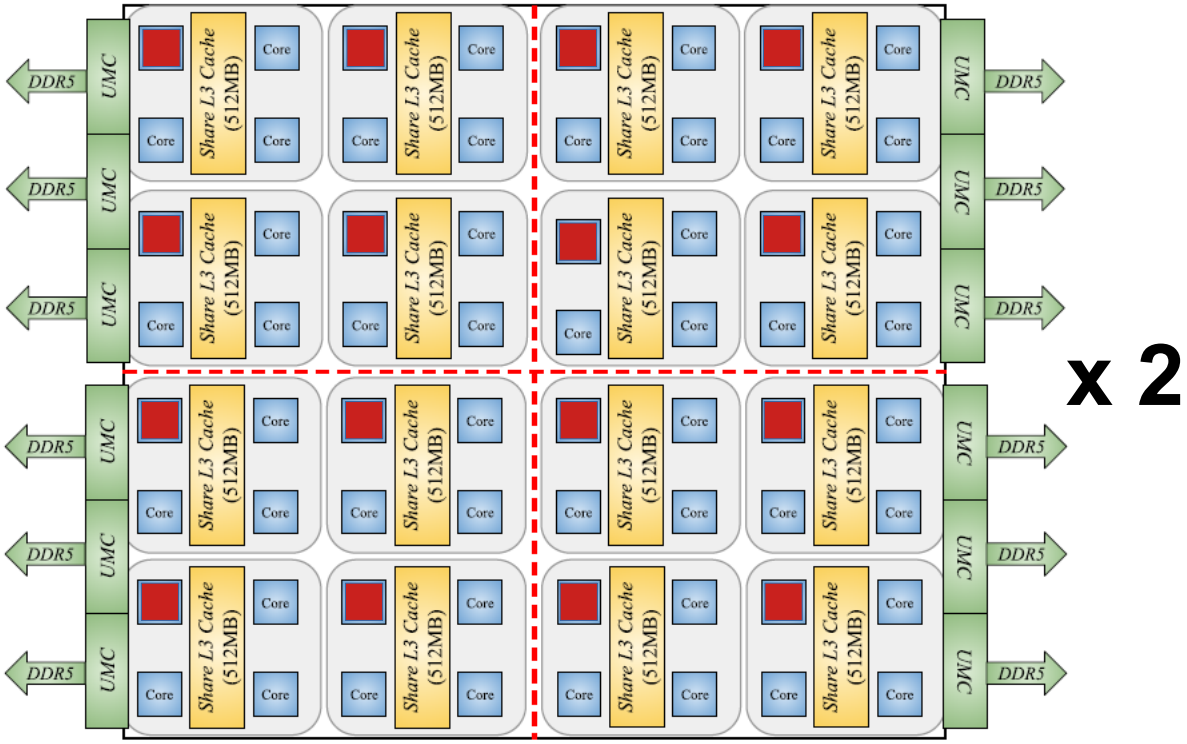
Thread is bound to core



Scheme 2: 64-core AMD EPYC 9535 processor (OpenMP 4.5), 38.4 GFLOPS per core

Computational Details II

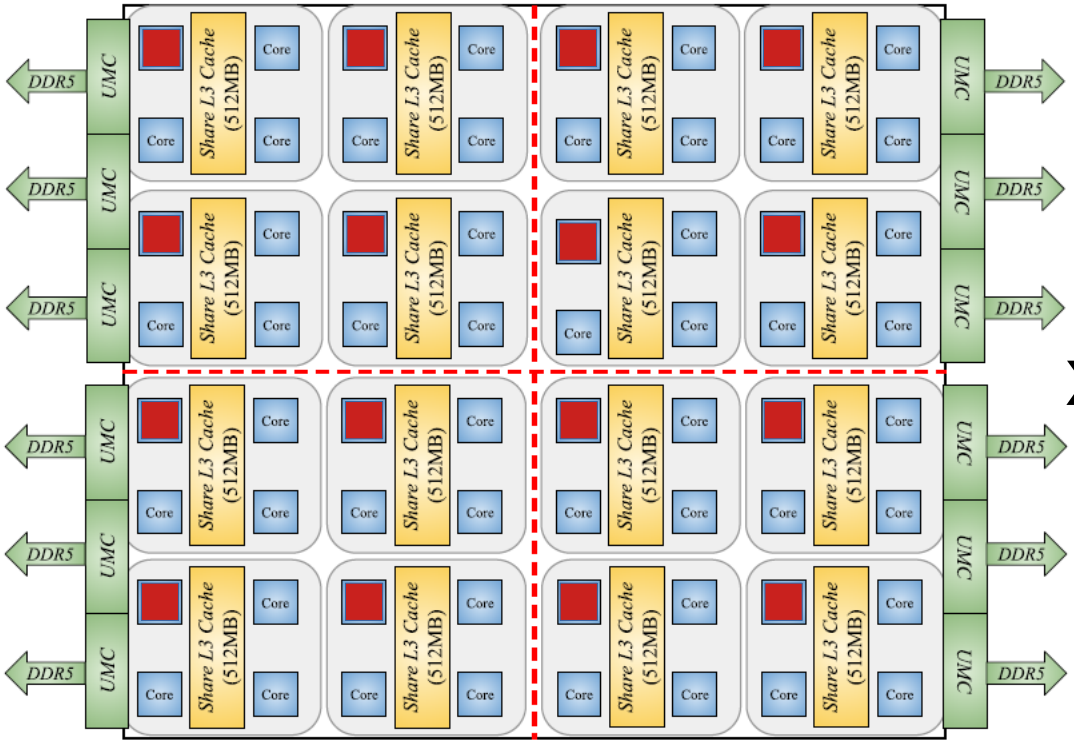
Thread is bound to core



Scheme 2: 64-core AMD EPYC 9535 processor (OpenMP 4.5), 38.4 GFLOPS per core

Computational Details II

Thread is bound to core

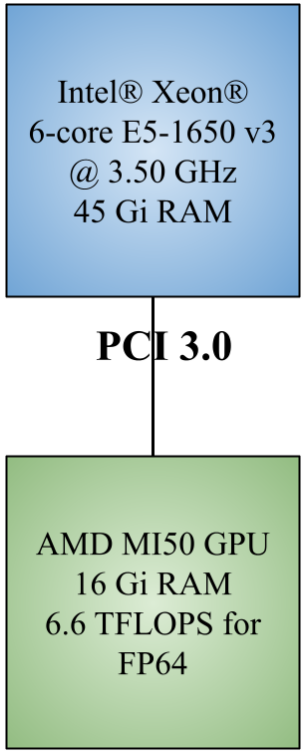


x 2

All L3-caches are involved first, then OpenMP multithreading is enabled

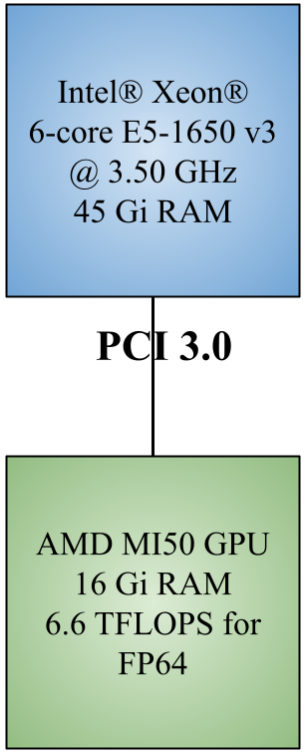
Scheme 2: 64-core AMD EPYC 9535 processor (OpenMP 4.5), 38.4 GFLOPS per core

Computational Details II



Scheme 3: Intel Xeon E5-1650 v3 host +
AMD MI50 device (ROCm 6.3.4)

Computational Details II



x 32 nodes
(InfiniBand FDR)

Scheme 3: Intel Xeon E5-1650 v3 host +
AMD MI50 device (ROCm 6.3.4)

Benchmark II: Performance

Reason: understand and compare efficiency of code execution on CPUs and GPUs

Benchmark II: Performance

Reason: understand and compare efficiency of code execution on CPUs and GPUs

$$\text{overhead} = \frac{R_{peak}\,\tau_{iter}}{N_{FP}}$$

R_{peak} – theoretical peak performance in FLOPS/s

τ_{iter} – *time to evaluate exchange matrix*

N_{FP} – *theoretically estimated number of arithmetic operations*

Benchmark II: Performance

Reason: understand and compare efficiency of code execution on CPUs and GPUs

$$\text{overhead} = \frac{R_{peak}\,\tau_{iter}}{N_{FP}}$$

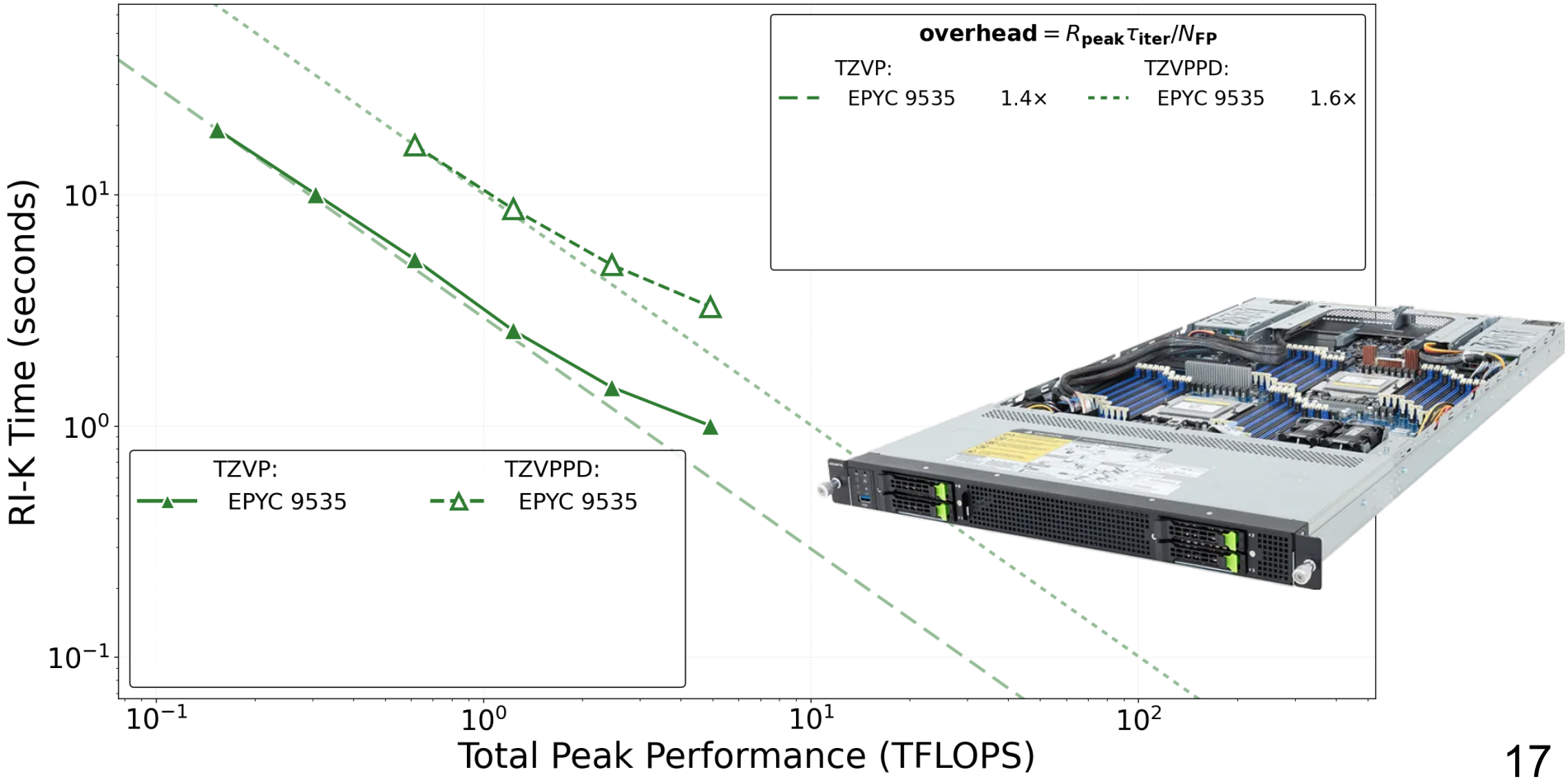
R_{peak} – theoretical peak performance in FLOPS/s

τ_{iter} – *time to evaluate exchange matrix*

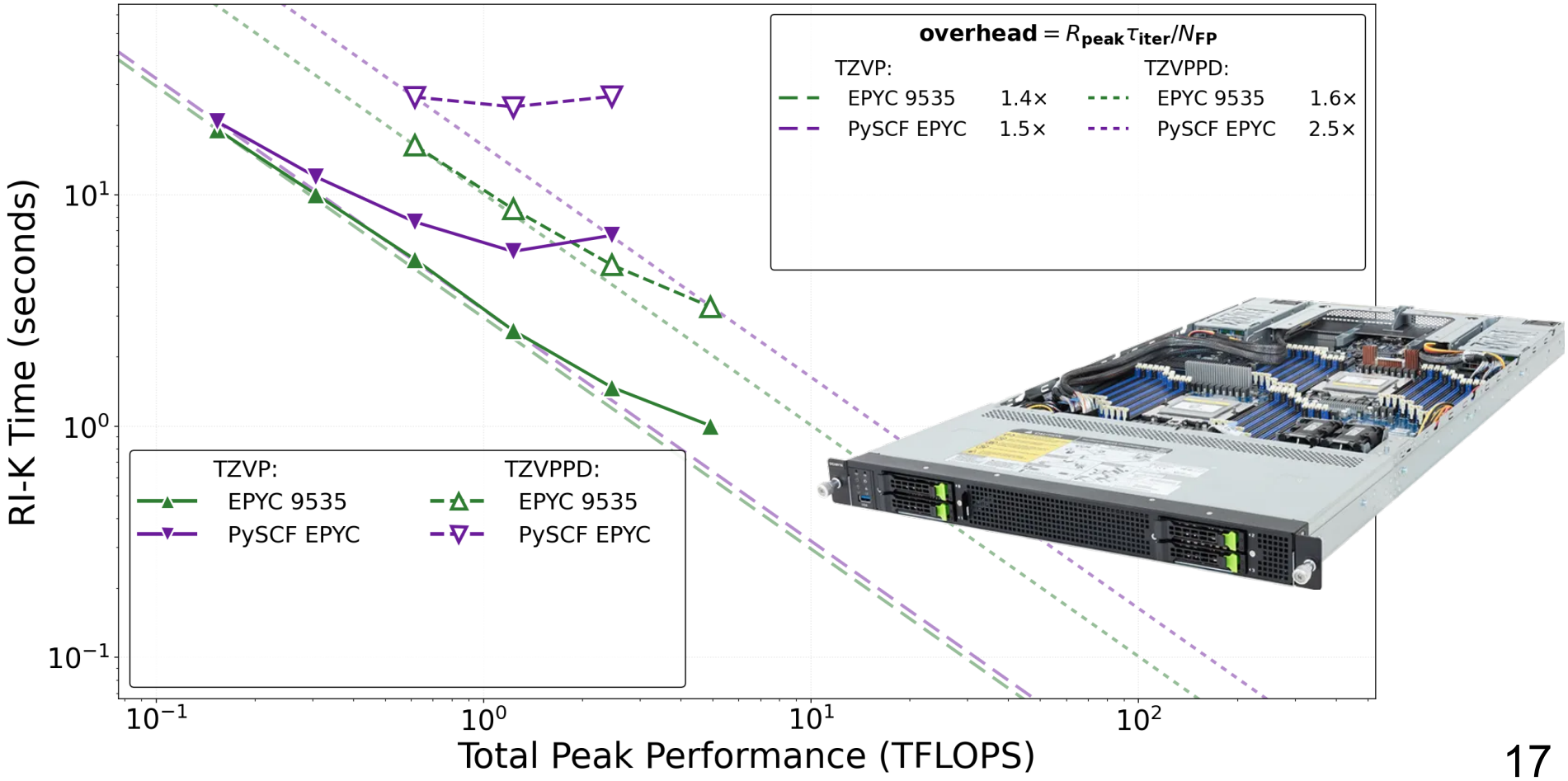
N_{FP} – *theoretically estimated number of arithmetic operations*

$$\text{percentage of total peak} = \frac{N_{FP}}{R_{peak}\,\tau_{iter}} \cdot 100\,\%$$

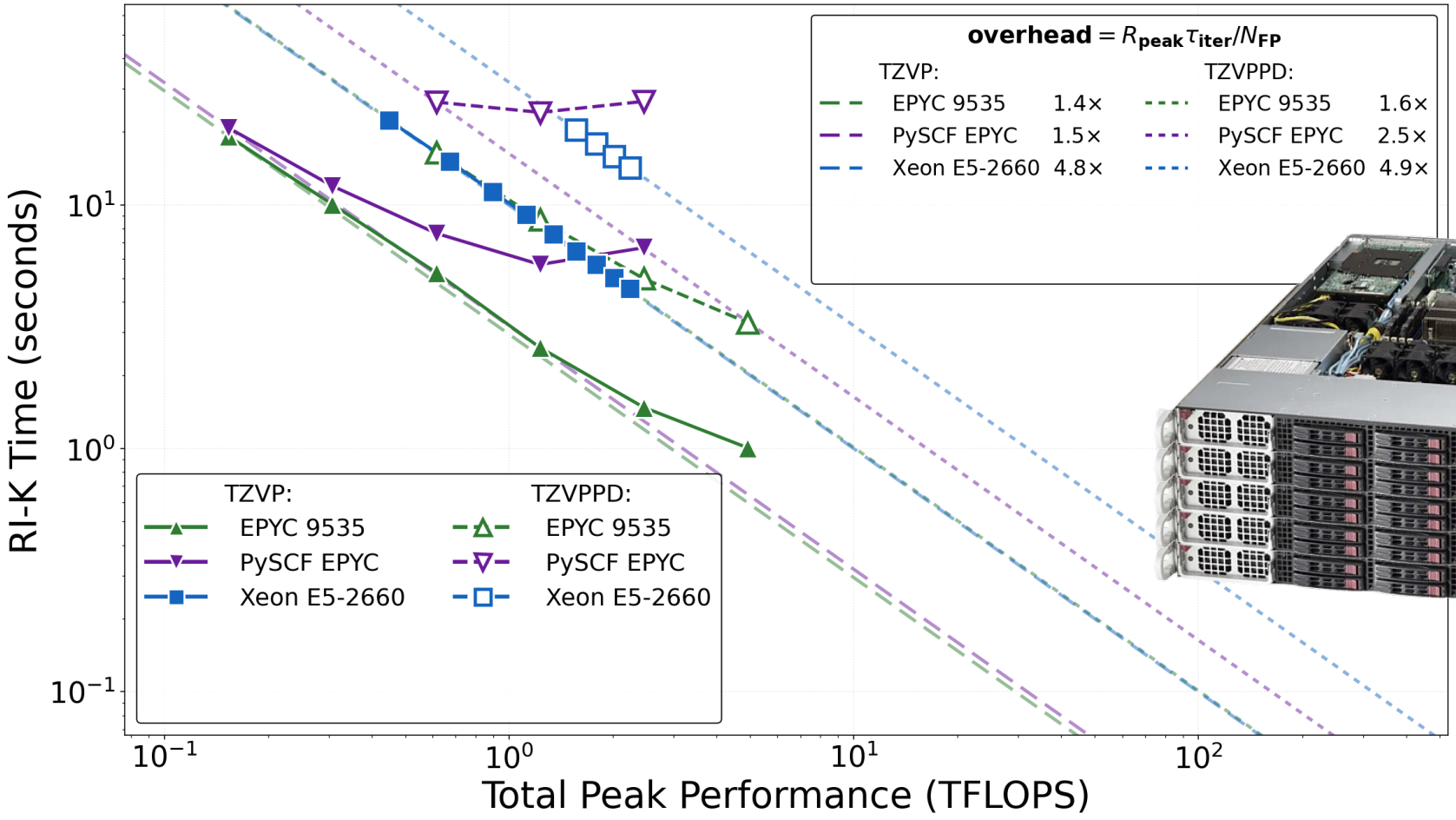
Benchmark II: Performance



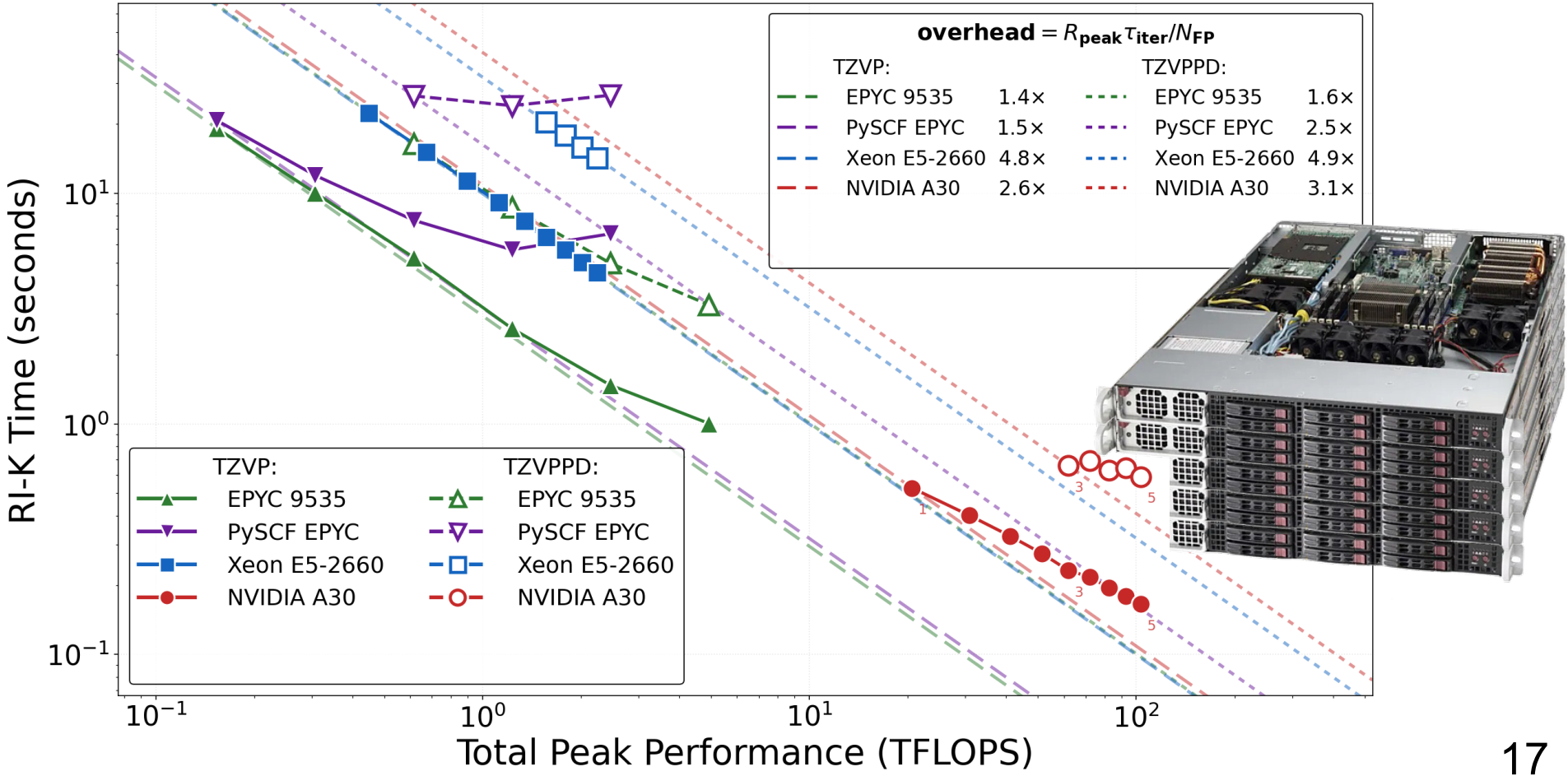
Benchmark II: Performance



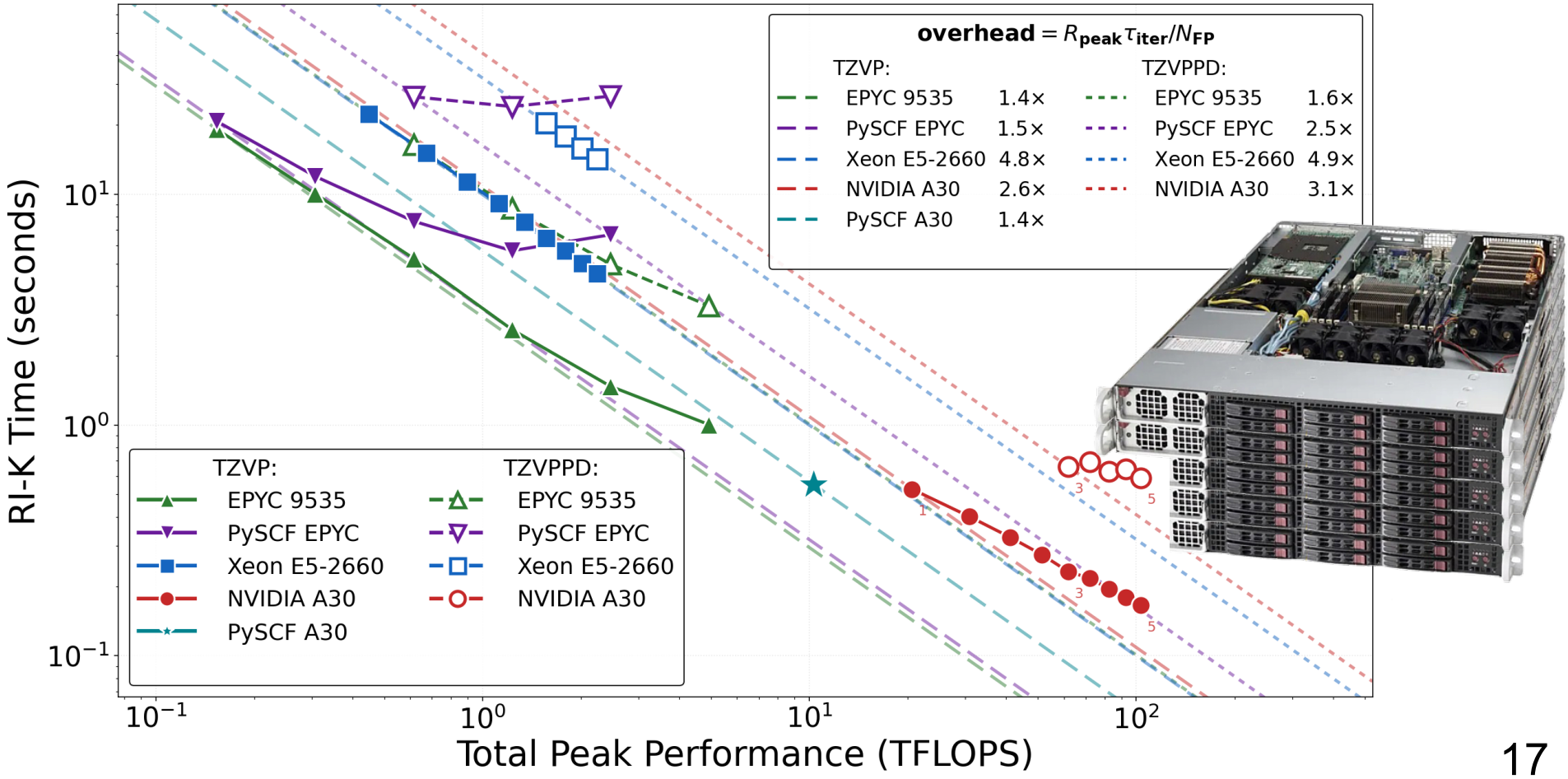
Benchmark II: Performance



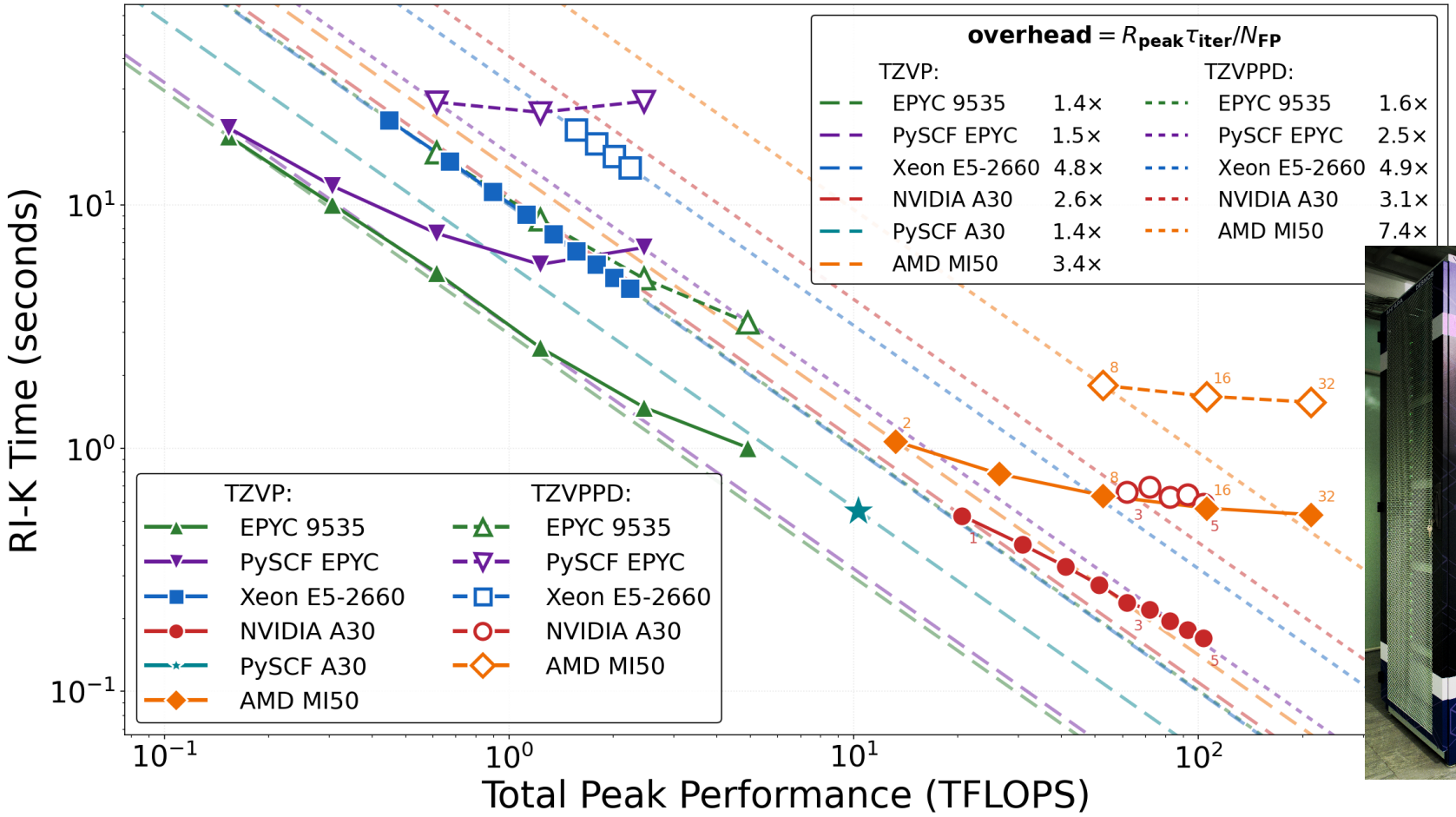
Benchmark II: Performance



Benchmark II: Performance



Benchmark II: Performance





Mesh Generation Utility MeGU

Goal: develop standalone open-source library for performance portable efficient mesh generation and DFT functional evaluation

Mesh Generation Utility MeGU

Goal: develop standalone open-source library for performance portable efficient mesh generation and DFT functional evaluation

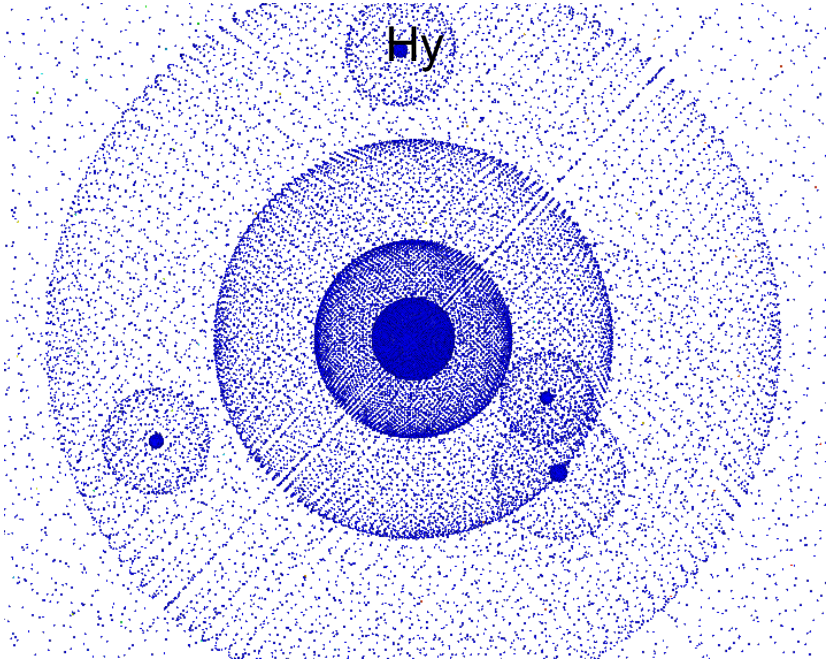
$$V^{xc}_{\mu\nu} = \int \bar{\chi}_\mu(\boldsymbol{r}) V^{xc}[\rho] \chi_\nu(\boldsymbol{r}) d\boldsymbol{r} = \sum_{ij} \omega_{ij} \bar{\chi}_\mu(\boldsymbol{r_{ij}}) V^{xc}[\rho] \chi_\nu(\boldsymbol{r_{ij}})$$

Generation of mesh:

- **<u>1-10%</u>** of SCF computation time;
- most generation time is taken by Becke decomposition: $O(N^2_{atoms})$
- scales with process number linearly

Mesh Generation Utility MeGU

Goal: develop standalone open-source library for performance portable efficient mesh generation and DFT functional evaluation



Mesh for CH_4 molecule

$$V^{xc}_{\mu\nu} = \int \bar{\chi}_\mu(\boldsymbol{r}) V^{xc}[\rho] \chi_\nu(\boldsymbol{r}) d\boldsymbol{r} = \sum_{ij} \omega_{ij} \bar{\chi}_\mu(\boldsymbol{r_{ij}}) V^{xc}[\rho] \chi_\nu(\boldsymbol{r_{ij}})$$

Generation of mesh:

- **<u>1-10%</u>** of SCF computation time;
- most generation time is taken by Becke decomposition: $O(N^2_{atoms})$
- scales with process number linearly

Conclusions

1. BUFO is a novel code for relativistic electronic structure calculations focused at Hartree—Fock and density functional theories.

Conclusions

1. BUFO is a novel code for relativistic electronic structure calculations focused at Hartree—Fock and density functional theories.
2. Mesh generation utility MeGU is an open-source library for performance portable density functional computations.

Conclusions

1. BUFO is a novel code for relativistic electronic structure calculations focused at Hartree—Fock and density functional theories.
2. Mesh generation utility MeGU is an open-source library for performance portable density functional computations.
3. In this work, BUFO has been ported to the MPI+Kokkos framework for performance portability.

Conclusions

1. BUFO is a novel code for relativistic electronic structure calculations focused at Hartree—Fock and density functional theories.
2. Mesh generation utility MeGU is an open-source library for performance portable density functional computations.
3. In this work, BUFO has been ported to the MPI+Kokkos framework for performance portability.
4. Performance has been compared for a server with AMD EPYC Turin CPUs as well as for two multi-node systems with Intel Xeon CPUs, NVIDIA and AMD GPUs.

Conclusions

1. BUFO is a novel code for relativistic electronic structure calculations focused at Hartree—Fock and density functional theories.
2. Mesh generation utility MeGU is an open-source library for performance portable density functional computations.
3. In this work, BUFO has been ported to the MPI+Kokkos framework for performance portability.
4. Performance has been compared for a server with AMD EPYC Turin CPUs as well as for two multi-node systems with Intel Xeon CPUs, NVIDIA and AMD GPUs.
5. Comparisons with the industry-standard PySCF code have been made.

Conclusions

1. BUFO is a novel code for relativistic electronic structure calculations focused at Hartree—Fock and density functional theories.
2. Mesh generation utility MeGU is an open-source library for performance portable density functional computations.
3. In this work, BUFO has been ported to the MPI+Kokkos framework for performance portability.
4. Performance has been compared for a server with AMD EPYC Turin CPUs as well as for two multi-node systems with Intel Xeon CPUs, NVIDIA and AMD GPUs.
5. Comparisons with the industry-standard PySCF code have been made.
6. MPI scalability allows BUFO to speed-up rather effectively on multi-node GPU clusters. MPI_Allreduce limitation has been detected and discussed.