14th International Workshop on OpenCL and SYCL

IWOCL 2026

Comparing the Performance of Heterogeneous Conjugate Gradient and Cholesky Solvers on Various Hardware Using SYCL

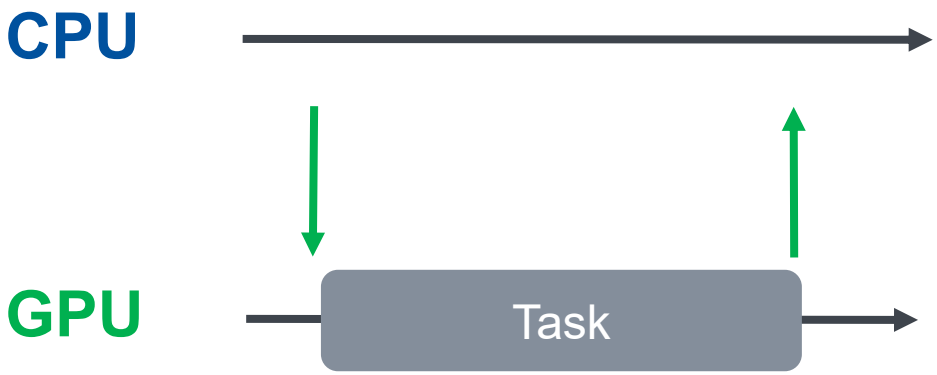
Tim Thüring, University of Stuttgart

Tim Thüring, Alexander Strack, and Dirk Pflüger, University of Stuttgart.

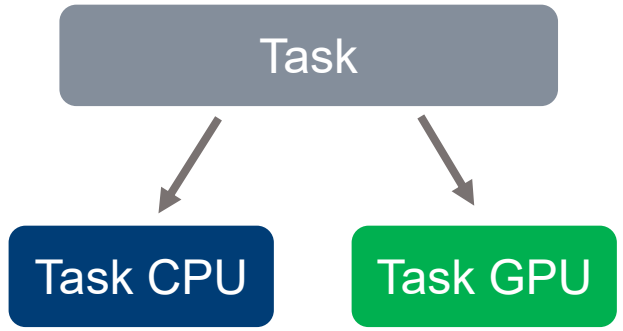
May 6-8, 2026 | Heilbronn, Germany | iwocl.org

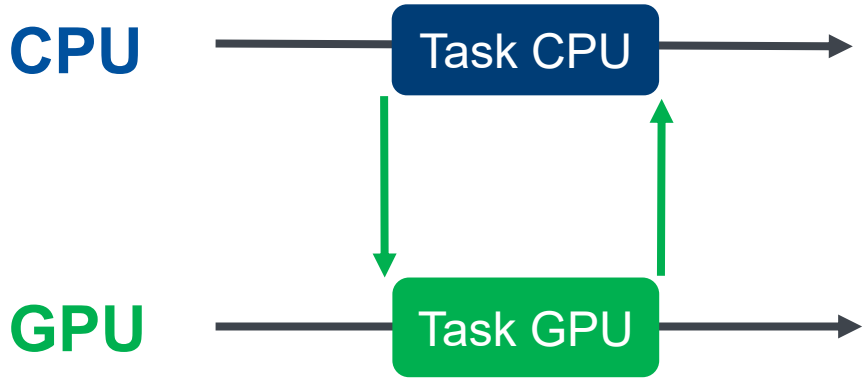
Motivation
Heterogeneous computing

Classic approach



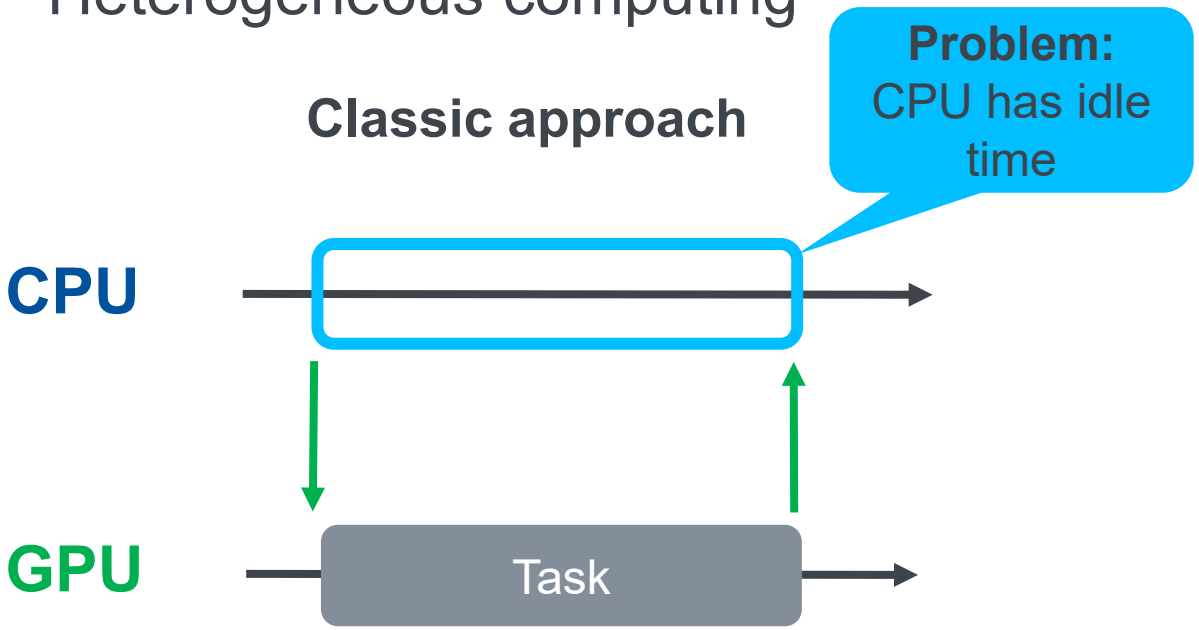
Heterogenous approach



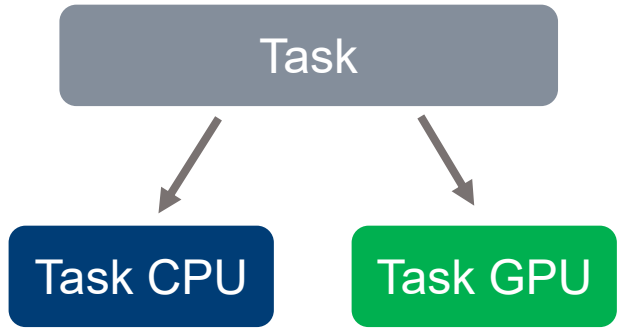


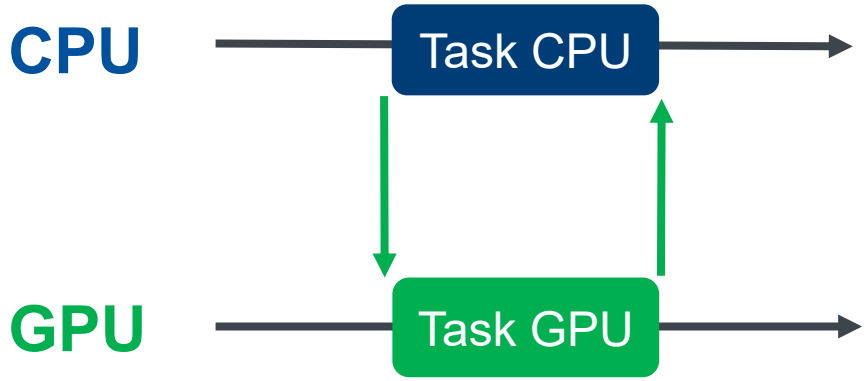
Motivation
Heterogeneous computing

Classic approach



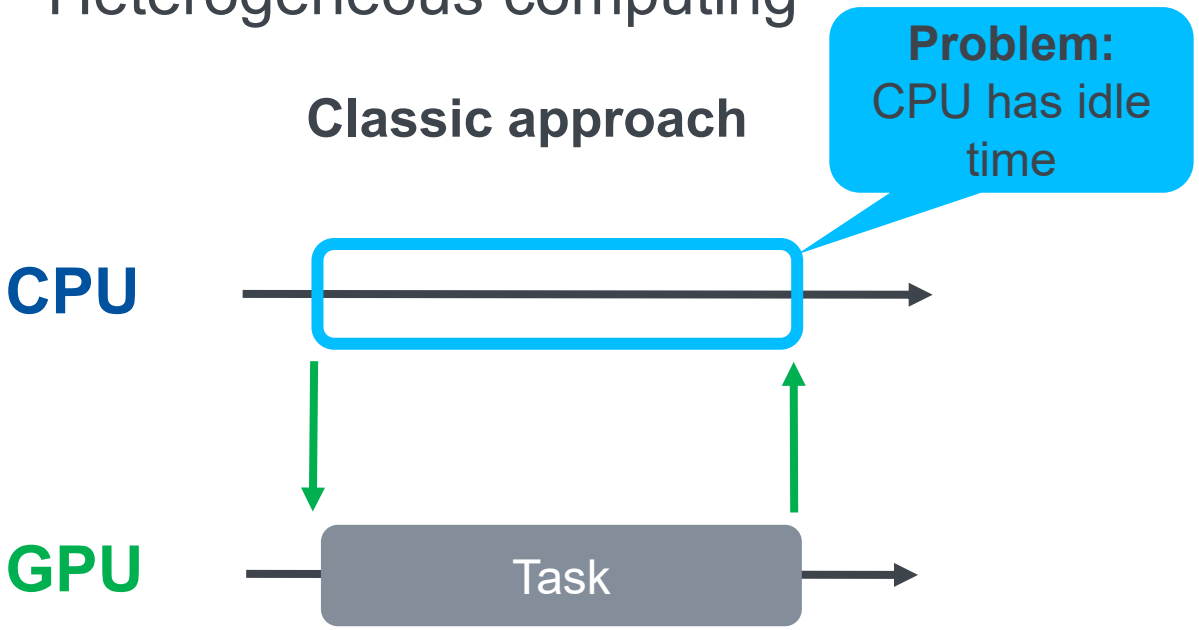
Heterogenous approach



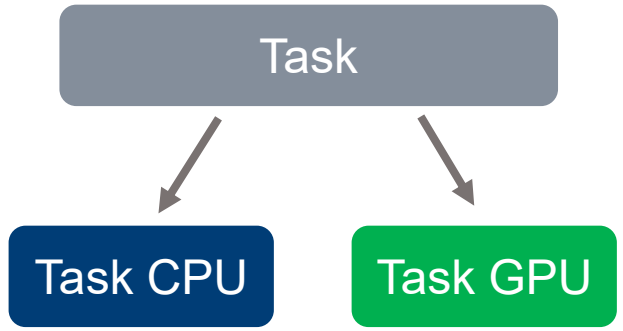


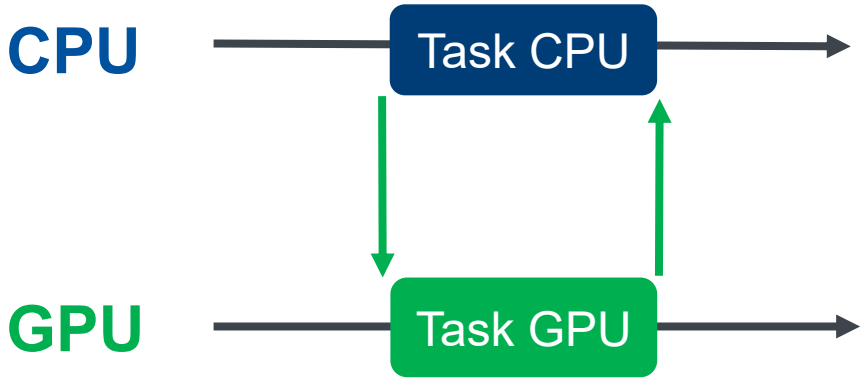
Motivation
Heterogeneous computing

Classic approach



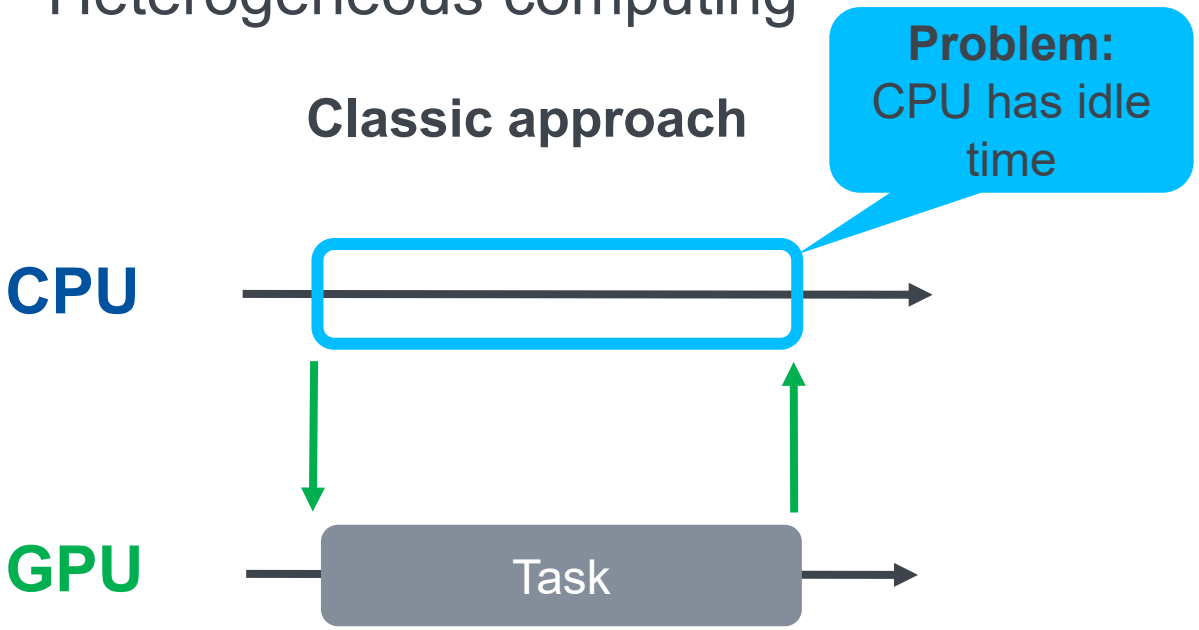
Heterogenous approach



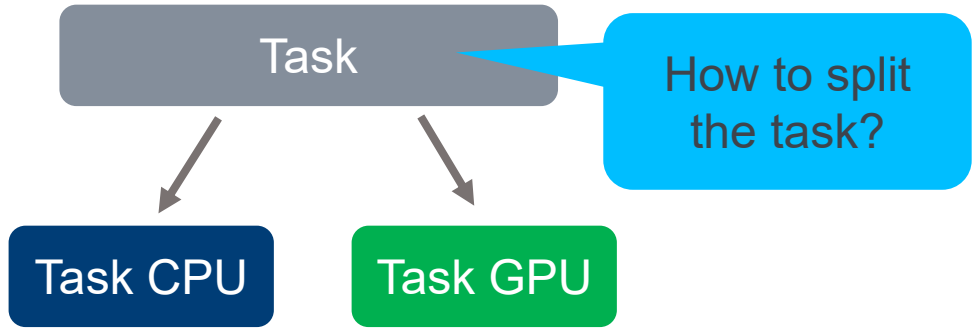


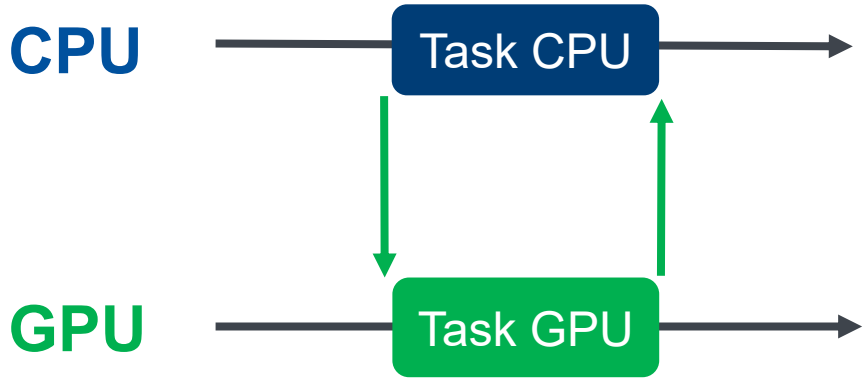
Motivation
Heterogeneous computing

Classic approach

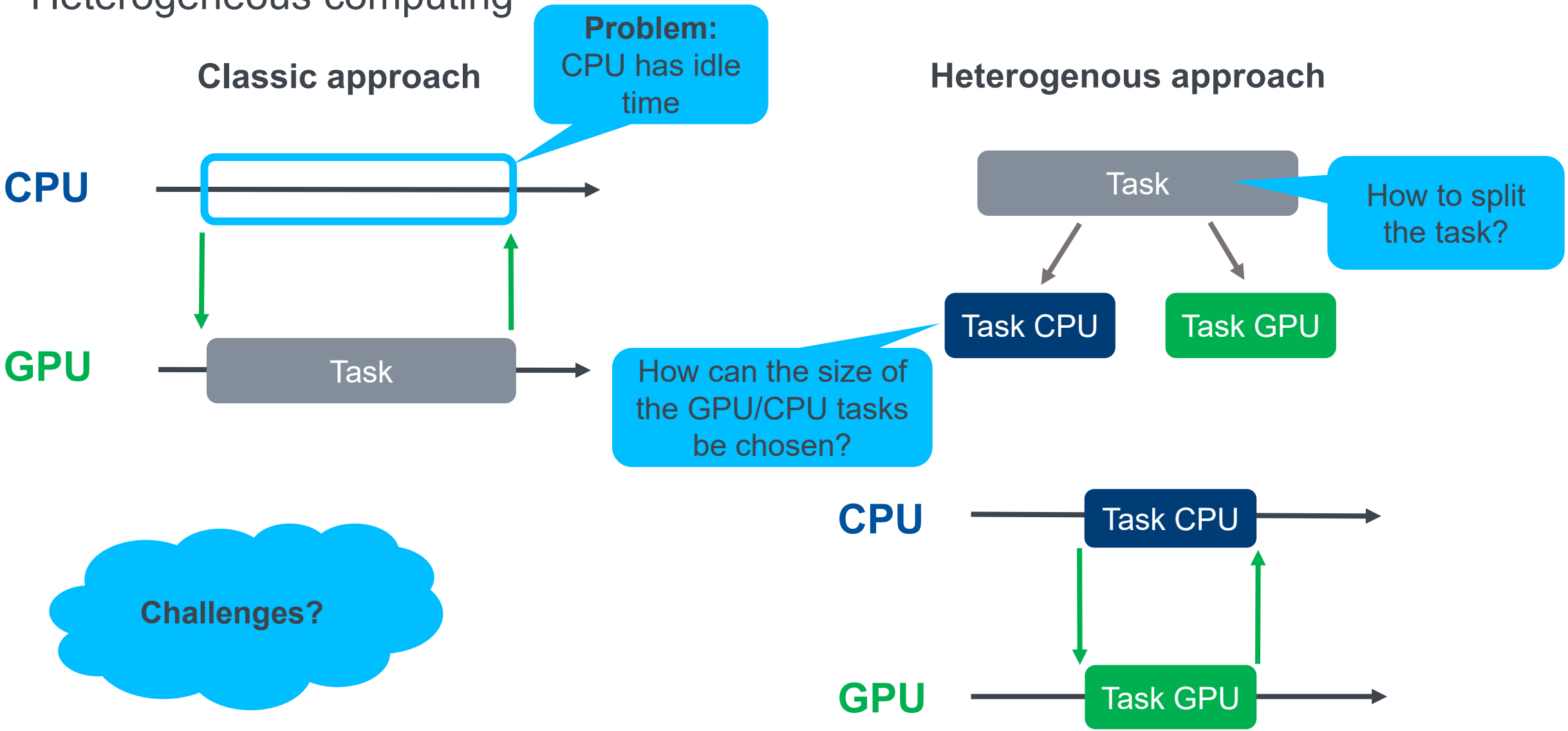


Heterogenous approach

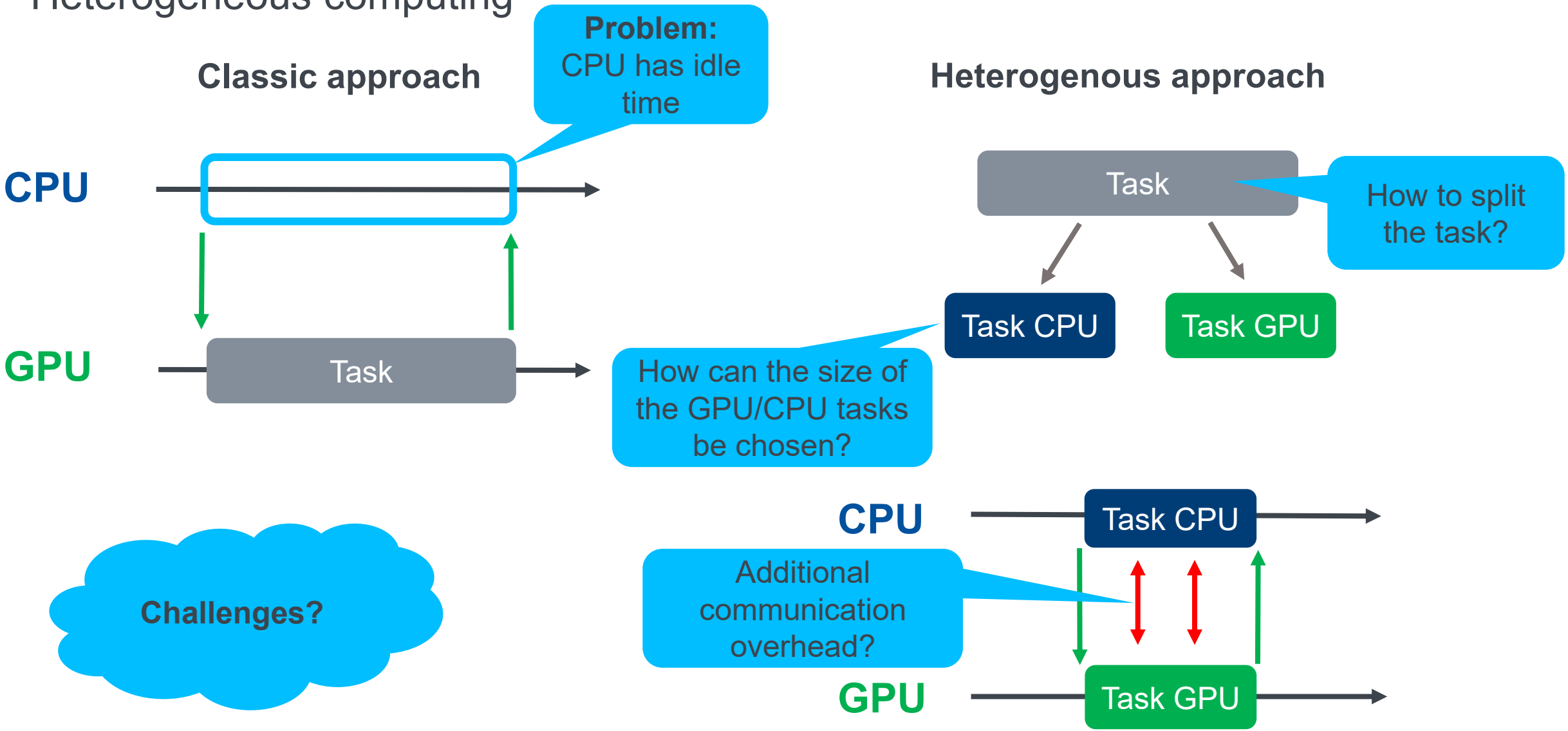




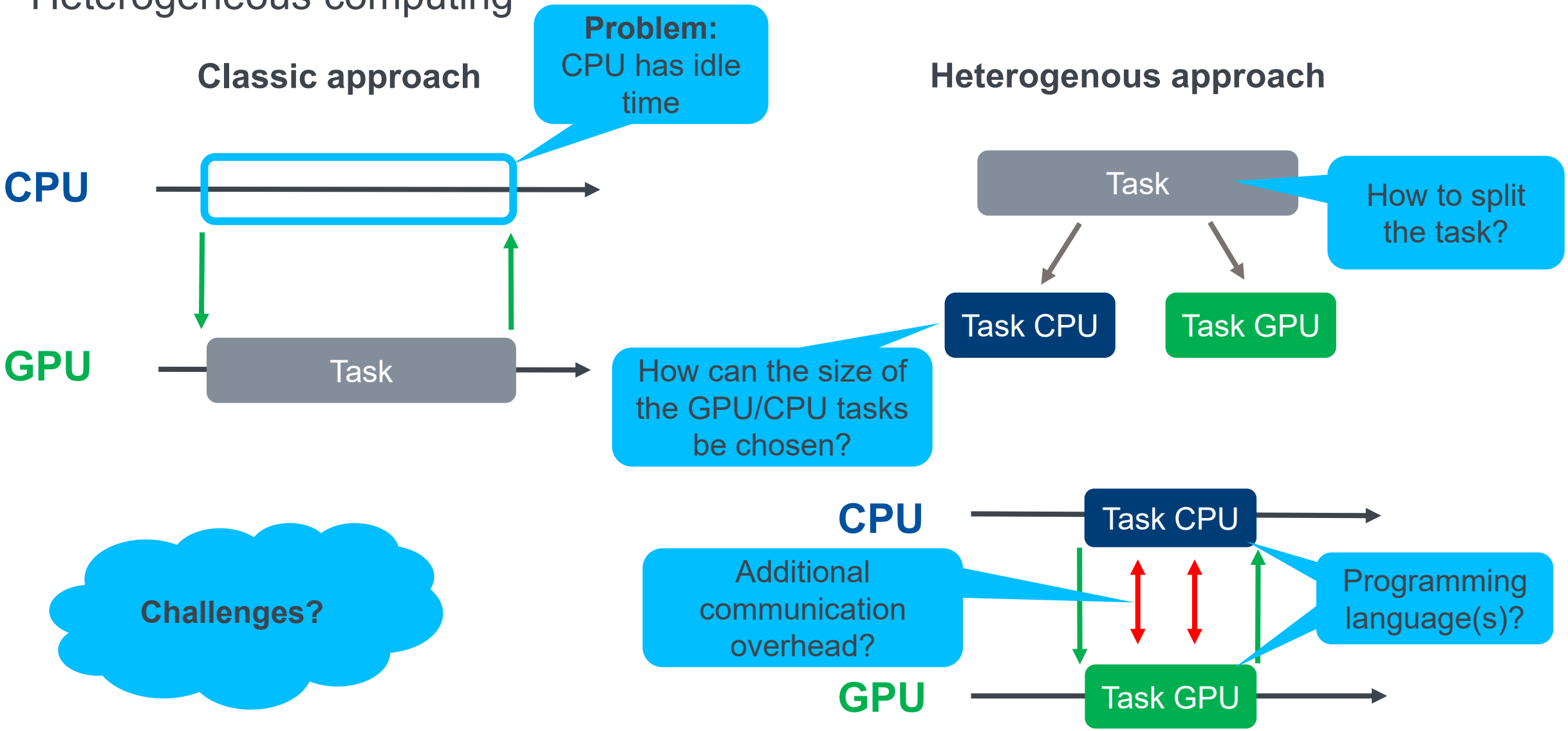
Motivation
Heterogeneous computing



Motivation
Heterogeneous computing



Motivation
Heterogeneous computing



Goal

- Explore heterogeneous computing with CPU-GPU cooperation enabled through SYCL

Goal

- Explore heterogeneous computing with CPU-GPU cooperation enabled through SYCL
- Heterogeneous implementation of solvers for linear systems with SPD matrices using SYCL
  - Conjugate Gradients algorithm
  - Cholesky factorization
  - Runtime comparison with CPU/GPU only implementations on various hardware including **NVIDIA, AMD, and Intel GPUs**

Conjugate Gradient Method

- Iteratively approximates solution of $Ax = b$ for SPD matrices A

While $\delta_{new} > \epsilon^2 \delta_0$ do:

$\mathrm{q} = \mathrm{Ad}$

$\alpha = \frac{\delta_{new}}{d^T \mathrm{q}}$

$\mathrm{x} = \mathrm{x} + \alpha \mathrm{d}$

$\mathrm{r} = \mathrm{b} - \alpha \mathrm{q}$

$\delta_{old} = \delta_{new}$

$\delta_{new} = \mathrm{r}^T \mathrm{r}$

$\beta = \frac{\delta_{new}}{\delta_{old}}$

$\mathrm{d} = \mathrm{r} + \beta \mathrm{d}$

Matrix-Vector operations

Vector-Vector operations

Conjugate Gradient Method

- Iteratively approximates solution of $Ax = b$ for SPD matrices A

While $\delta_{new} > \epsilon^2 \delta_0$ do:

$$q = Ad$$

$$\alpha = \frac{\delta_{new}}{d^T q}$$

$$x = x + \alpha d$$

$$r = b - \alpha q$$

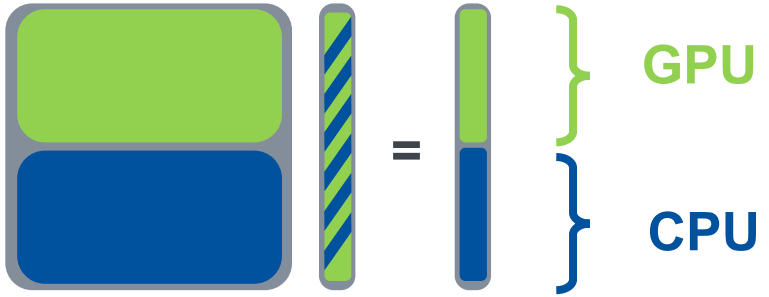
$$\delta_{old} = \delta_{new}$$

$$\delta_{new} = r^T r$$

$$\beta = \frac{\delta_{new}}{\delta_{old}}$$

$$d = r + \beta d$$

Matrix-Vector operations



Vector-Vector operations

[1] Tiwari, Manasi, and Sathish Vadhiyar. "Efficient executions of Pipelined Conjugate Gradient Method on Heterogeneous Architectures." (2021).

Conjugate Gradient Method

- Iteratively approximates solution of $Ax = b$ for SPD matrices A

While $\delta_{new} > \epsilon^2 \delta_0$ do:

$$q = Ad$$

$$\alpha = \frac{\delta_{new}}{d^T q}$$

$$x = x + \alpha d$$

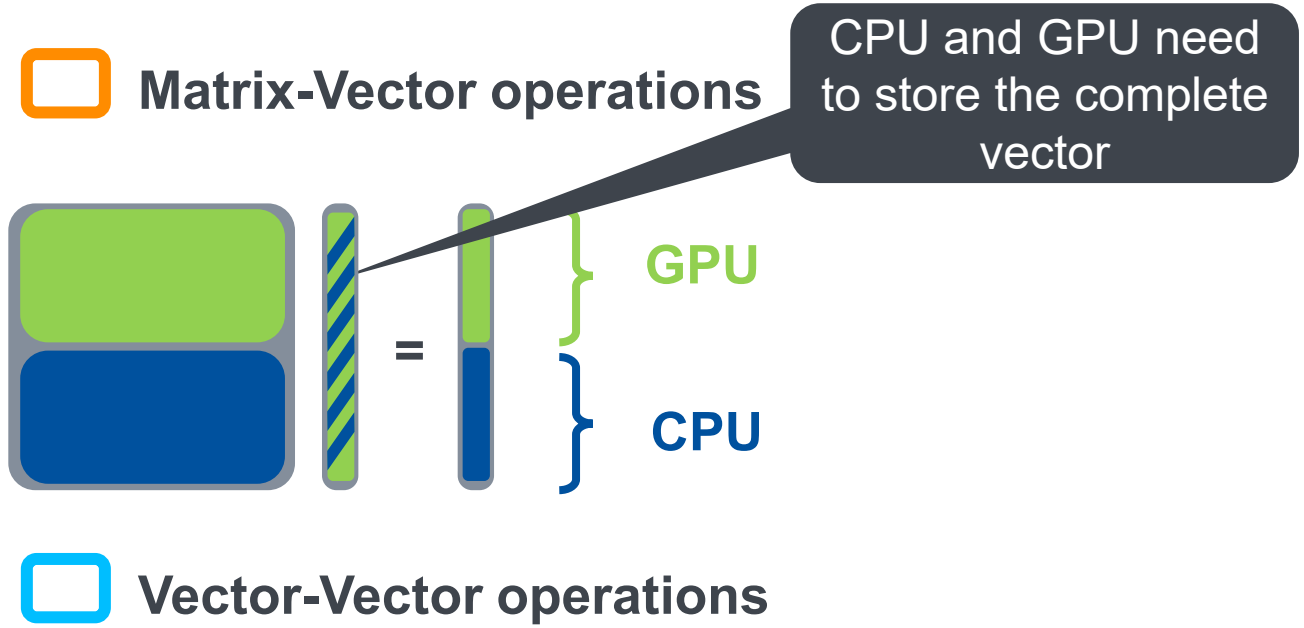
$$r = b - \alpha q$$

$$\delta_{old} = \delta_{new}$$

$$\delta_{new} = r^T r$$

$$\beta = \frac{\delta_{new}}{\delta_{old}}$$

$$d = r + \beta d$$



[1] Tiwari, Manasi, and Sathish Vadhiyar. "Efficient executions of Pipelined Conjugate Gradient Method on Heterogeneous Architectures." (2021).

Conjugate Gradient Method

- Iteratively approximates solution of $Ax = b$ for SPD matrices A

While $\delta_{new} > \epsilon^2 \delta_0$ do:

$$q = Ad$$

$$\alpha = \frac{\delta_{new}}{d^T q}$$

$$x = x + \alpha d$$

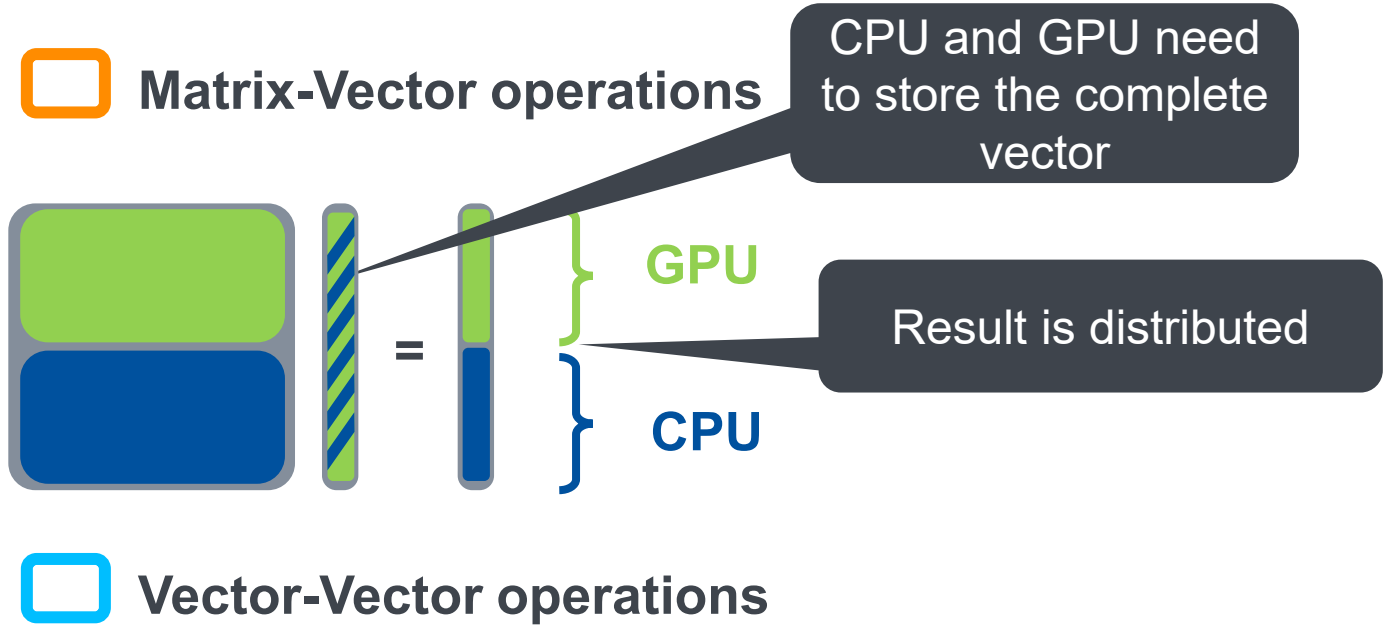
$$r = b - \alpha q$$

$$\delta_{old} = \delta_{new}$$

$$\delta_{new} = r^T r$$

$$\beta = \frac{\delta_{new}}{\delta_{old}}$$

$$d = r + \beta d$$



[1] Tiwari, Manasi, and Sathish Vadhiyar. "Efficient executions of Pipelined Conjugate Gradient Method on Heterogeneous Architectures." (2021).

Conjugate Gradient Method

- Iteratively approximates solution of $Ax = b$ for SPD matrices A

While $\delta_{new} > \epsilon^2 \delta_0$ do:

$$q = Ad$$

$$\alpha = \frac{\delta_{new}}{d^T q}$$

$$x = x + \alpha d$$

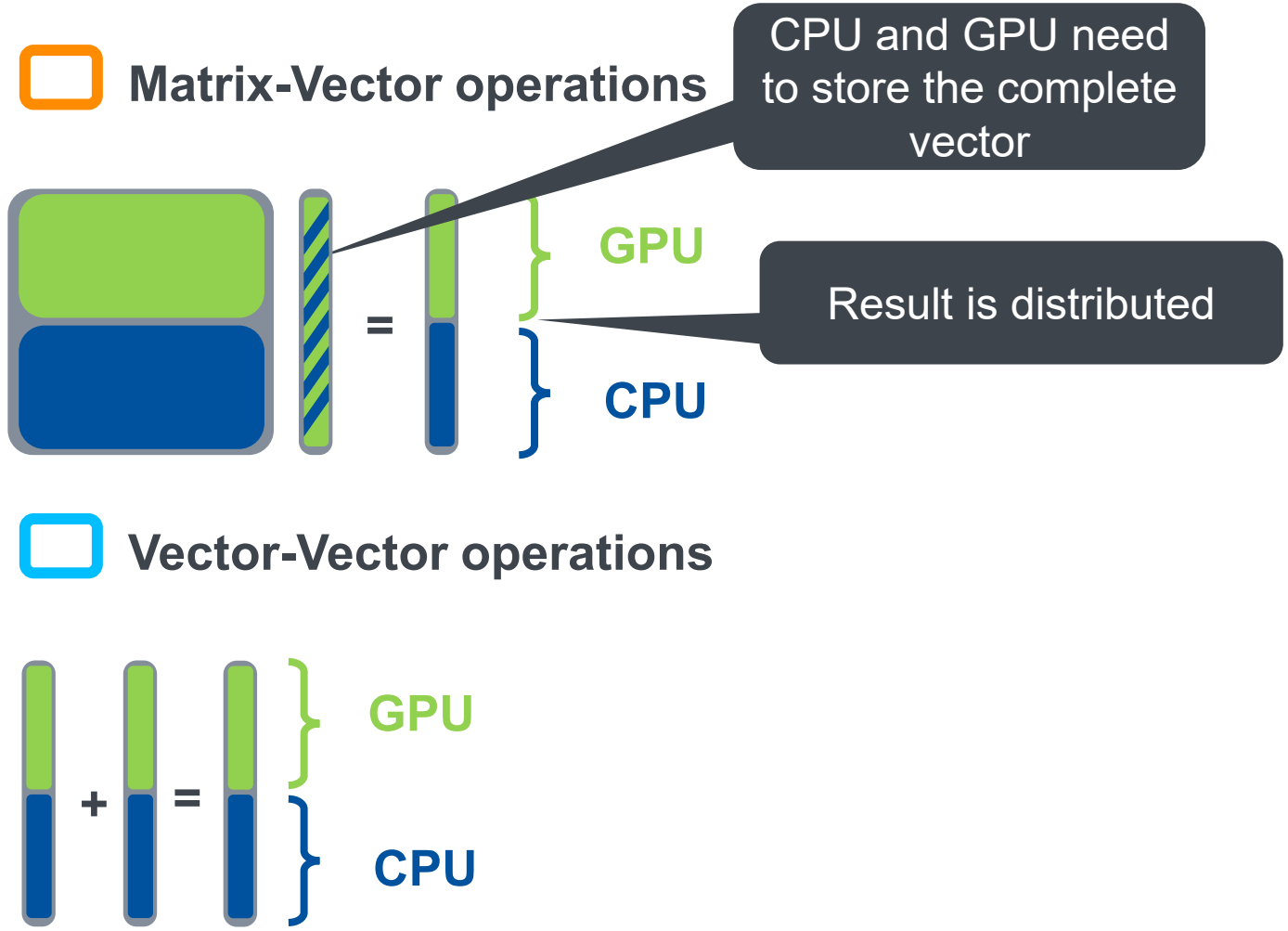
$$r = b - \alpha q$$

$$\delta_{old} = \delta_{new}$$

$$\delta_{new} = r^T r$$

$$\beta = \frac{\delta_{new}}{\delta_{old}}$$

$$d = r + \beta d$$



[1] Tiwari, Manasi, and Sathish Vadhiyar. "Efficient executions of Pipelined Conjugate Gradient Method on Heterogeneous Architectures." (2021).

Conjugate Gradient Method

- Iteratively approximates solution of $Ax = b$ for SPD matrices A

While $\delta_{new} > \epsilon^2 \delta_0$ do:

$$q = Ad$$

$$\alpha = \frac{\delta_{new}}{d^T q}$$

$$x = x + \alpha d$$

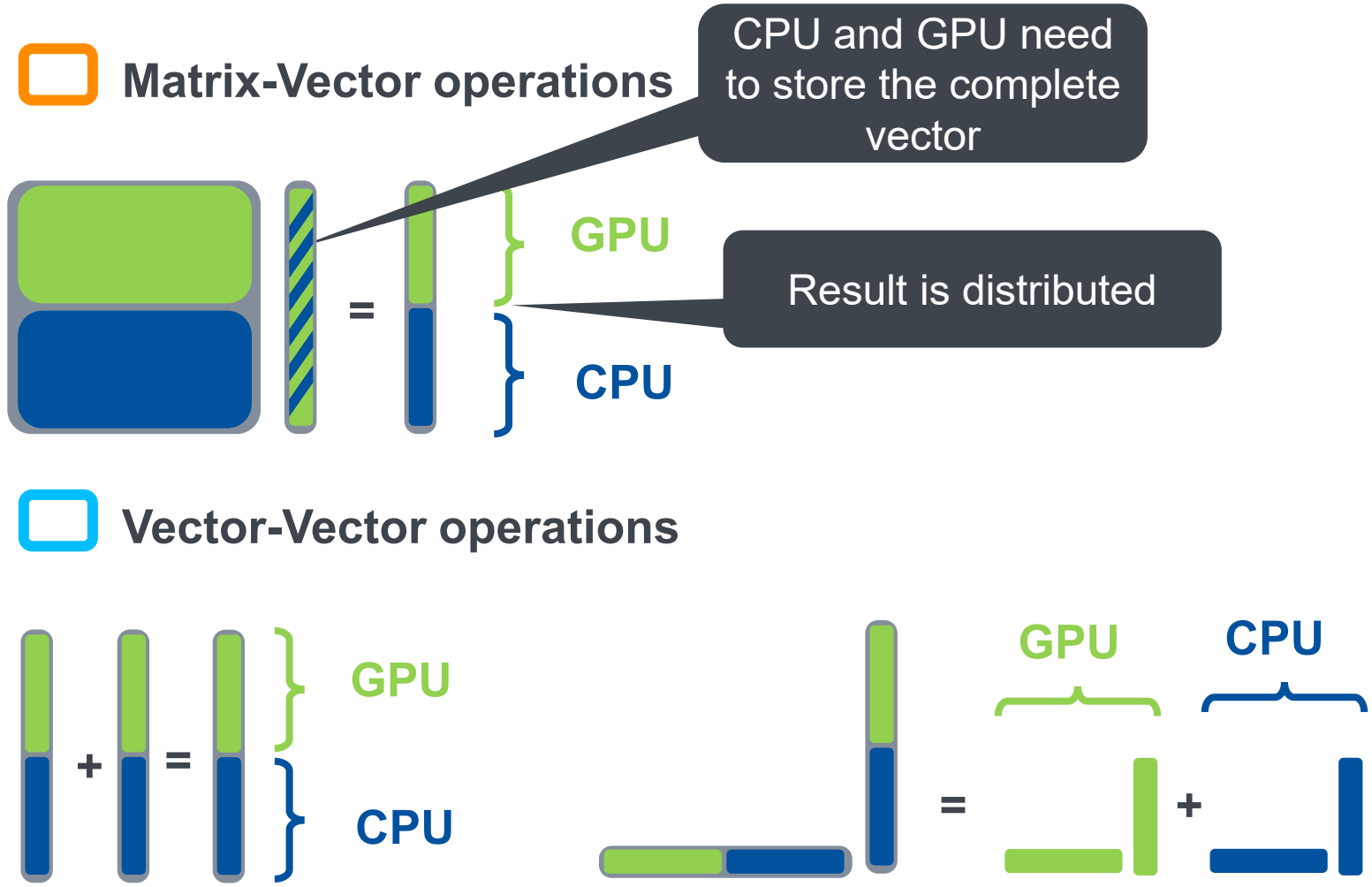
$$r = b - \alpha q$$

$$\delta_{old} = \delta_{new}$$

$$\delta_{new} = r^T r$$

$$\beta = \frac{\delta_{new}}{\delta_{old}}$$

$$d = r + \beta d$$



[1] Tiwari, Manasi, and Sathish Vadhiyar. "Efficient executions of Pipelined Conjugate Gradient Method on Heterogeneous Architectures." (2021).

Conjugate Gradient Method

- Iteratively approximates solution of $Ax = b$ for SPD matrices A

While $\delta_{new} > \epsilon^2 \delta_0$ do:

$$q = Ad$$

$$\alpha = \frac{\delta_{new}}{d^T q}$$

$$x = x + \alpha d$$

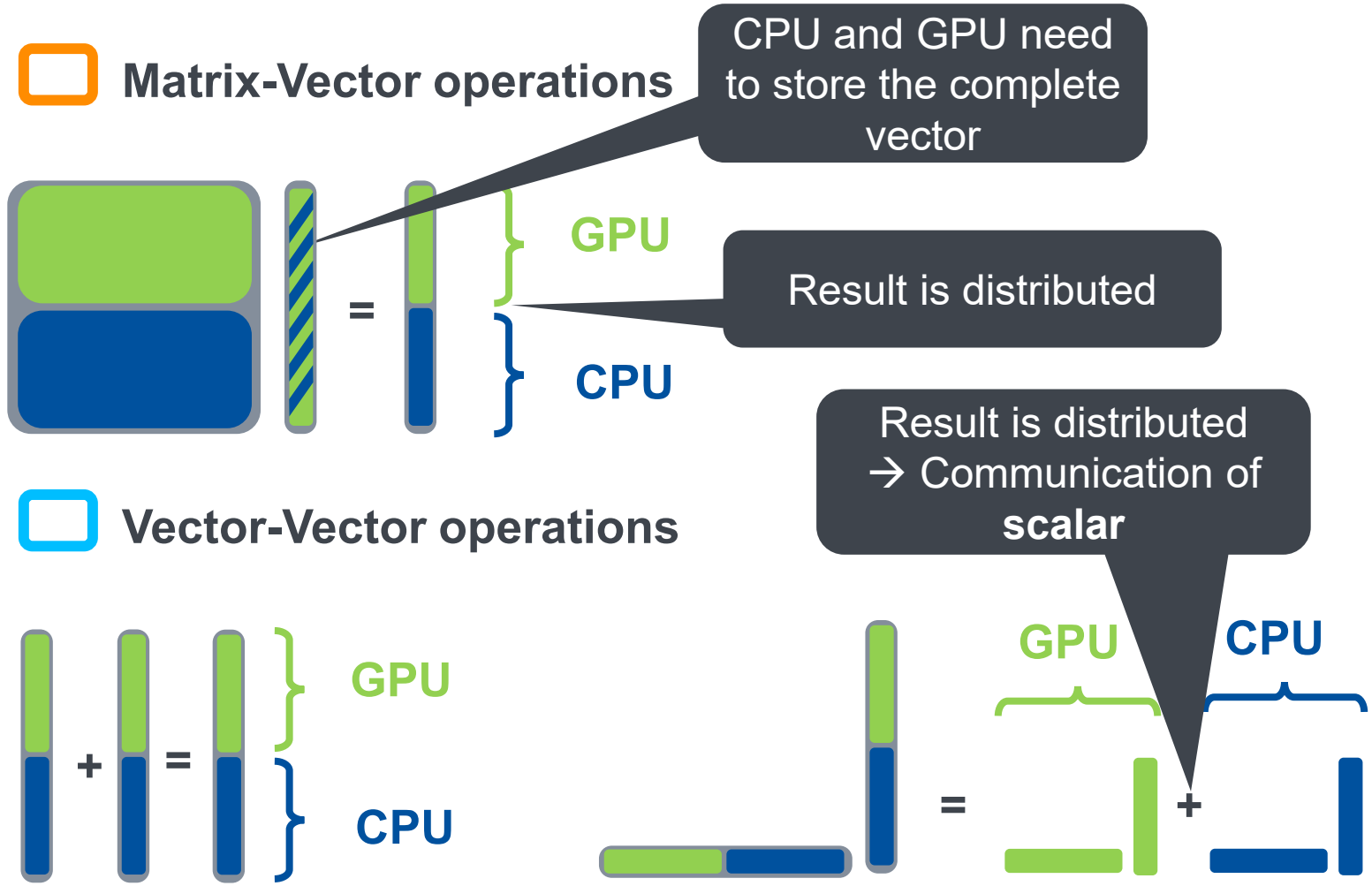
$$r = b - \alpha q$$

$$\delta_{old} = \delta_{new}$$

$$\delta_{new} = r^T r$$

$$\beta = \frac{\delta_{new}}{\delta_{old}}$$

$$d = r + \beta d$$



[1] Tiwari, Manasi, and Sathish Vadhiyar. "Efficient executions of Pipelined Conjugate Gradient Method on Heterogeneous Architectures." (2021).

Conjugate Gradient Method

- Iteratively approximates solution of $Ax = b$ for SPD matrices A

While $\delta_{new} > \epsilon^2 \delta_0$ do:

$q = Ad$

$\alpha = \frac{\delta_{new}}{d^T q}$ **Scalar Com.**

$x = x + \alpha d$

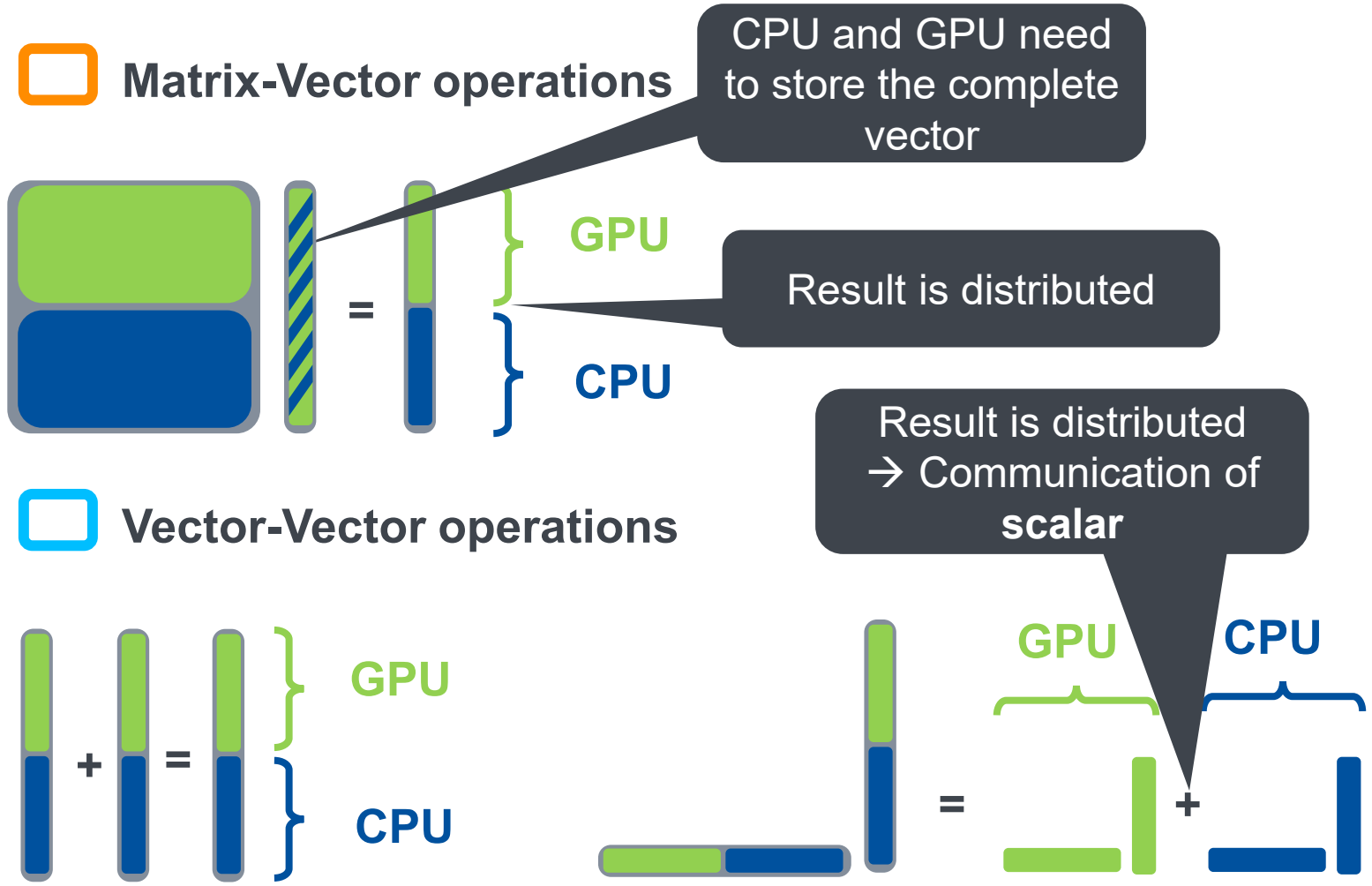
$r = b - \alpha q$

$\delta_{old} = \delta_{new}$

$\delta_{new} = r^T r$ **Scalar Com.**

$\beta = \frac{\delta_{new}}{\delta_{old}}$

$d = r + \beta d$



[1] Tiwari, Manasi, and Sathish Vadhiyar. "Efficient executions of Pipelined Conjugate Gradient Method on Heterogeneous Architectures." (2021).

Conjugate Gradient Method

- Iteratively approximates solution of $Ax = b$ for SPD matrices A

While $\delta_{new} > \epsilon^2 \delta_0$ do:

$$q = Ad$$

$$\alpha = \frac{\delta_{new}}{d^T q}$$ **Scalar Com.**

$$x = x + \alpha d$$

$$r = b - \alpha q$$

$$\delta_{old} = \delta_{new}$$

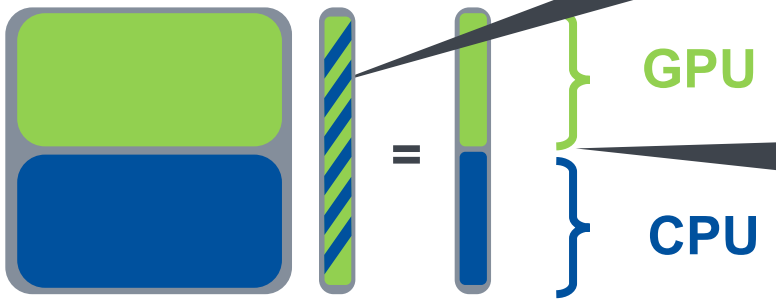
$$\delta_{new} = r^T r$$ **Scalar Com.**

$$\beta = \frac{\delta_{new}}{\delta_{old}}$$

$$d = r + \beta d$$

Vector Com. of d

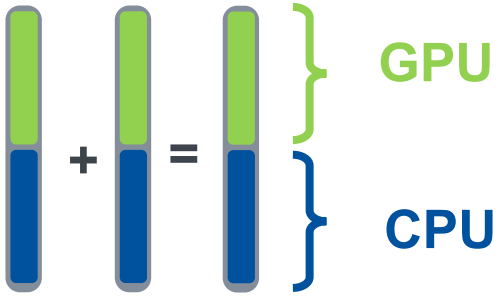
Matrix-Vector operations



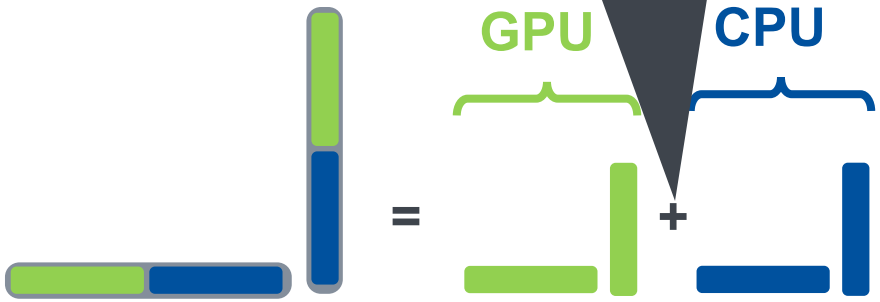
CPU and GPU need to store the complete vector

Result is distributed

Vector-Vector operations



Result is distributed → Communication of **scalar**



[1] Tiwari, Manasi, and Sathish Vadhiyar. "Efficient executions of Pipelined Conjugate Gradient Method on Heterogeneous Architectures." (2021).

Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

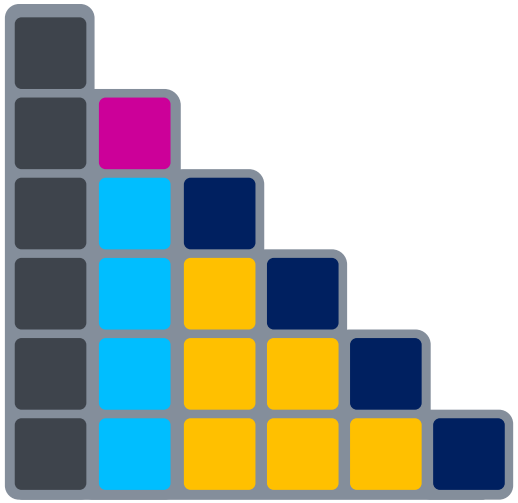
A_{mk} = solve(A_{kk}, A_{mk})

for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

A_{mk} = solve(A_{kk}, A_{mk})

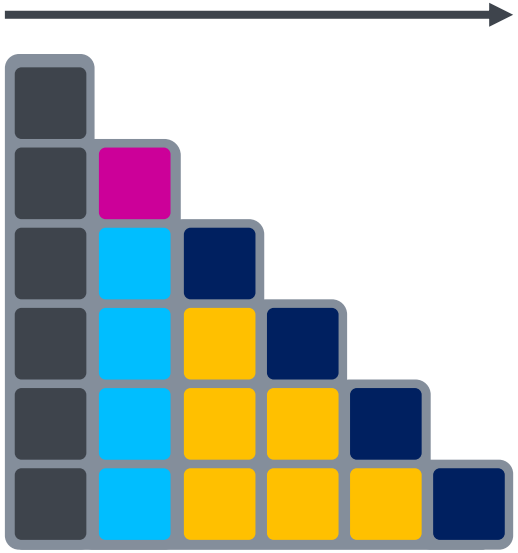
for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$

Right-looking: one column after another



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

A_{mk} = solve(A_{kk}, A_{mk})

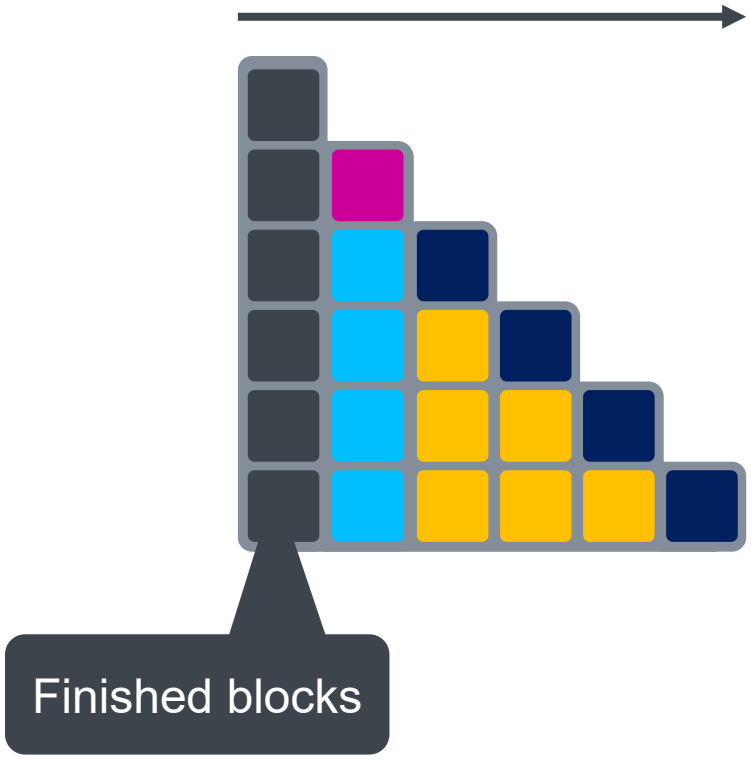
for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$

Right-looking: one column after another



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

Right-looking: one column after another

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

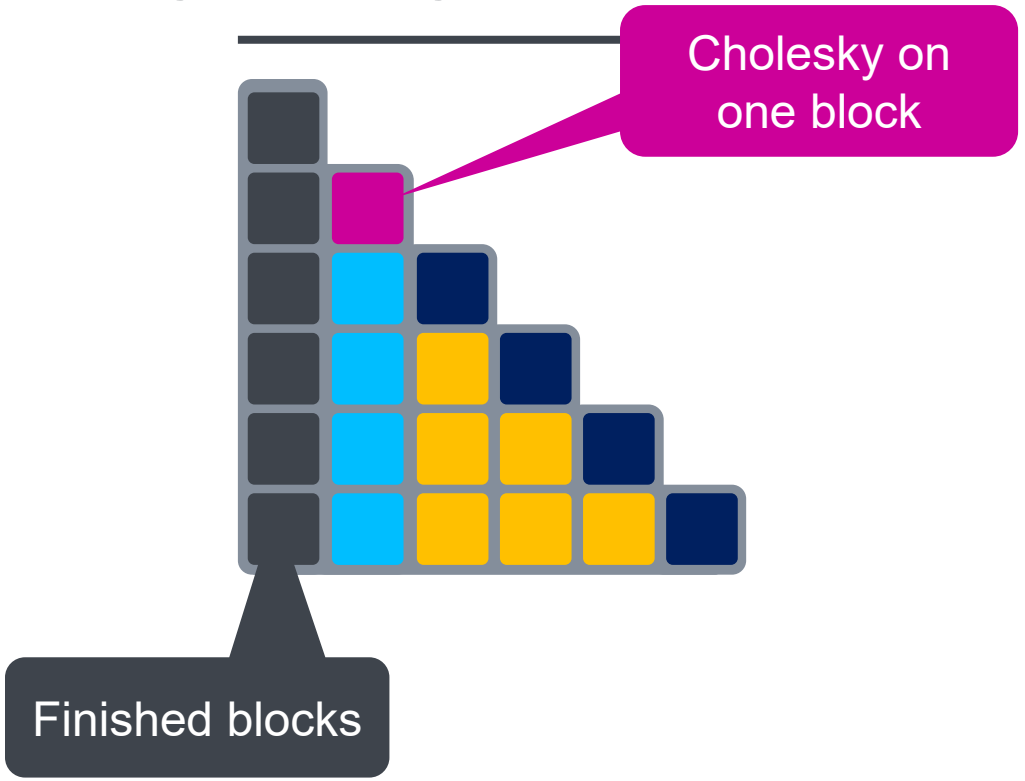
A_{mk} = solve(A_{kk}, A_{mk})

for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

Right-looking: one column after another

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

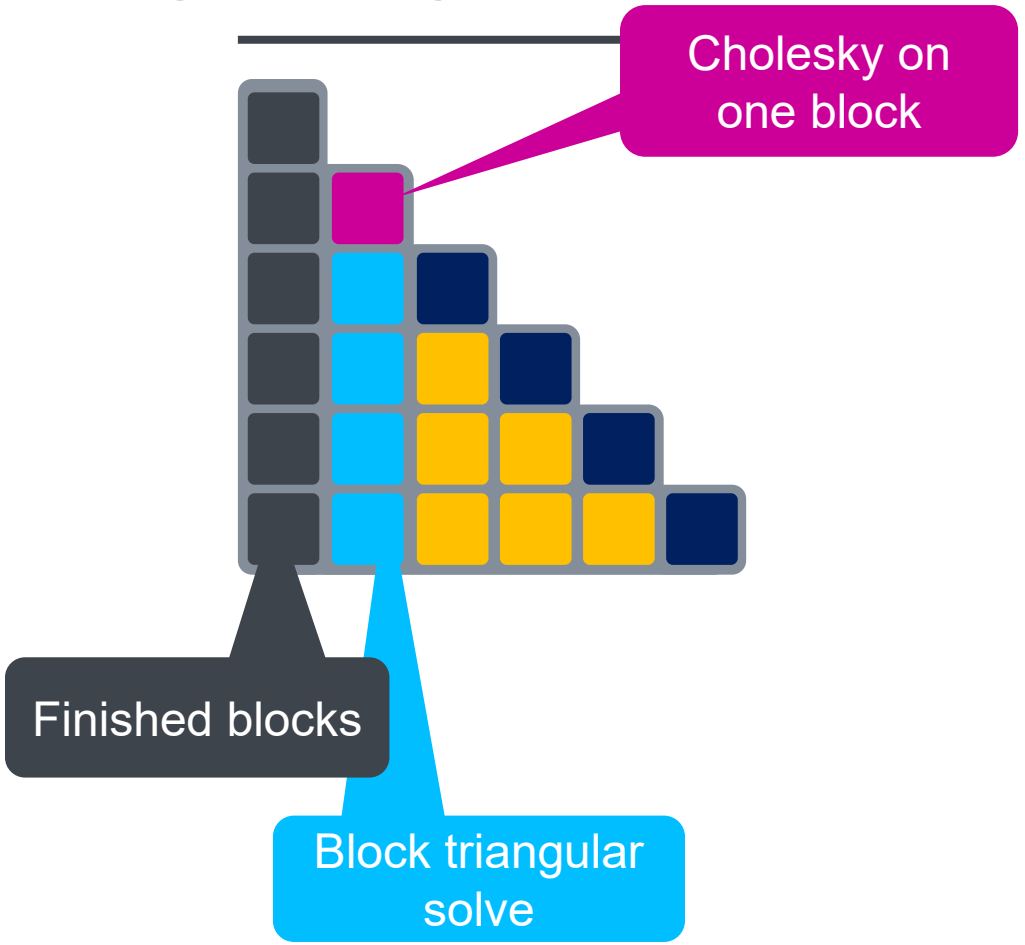
A_{mk} = solve(A_{kk}, A_{mk})

for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

Right-looking: one column after another

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

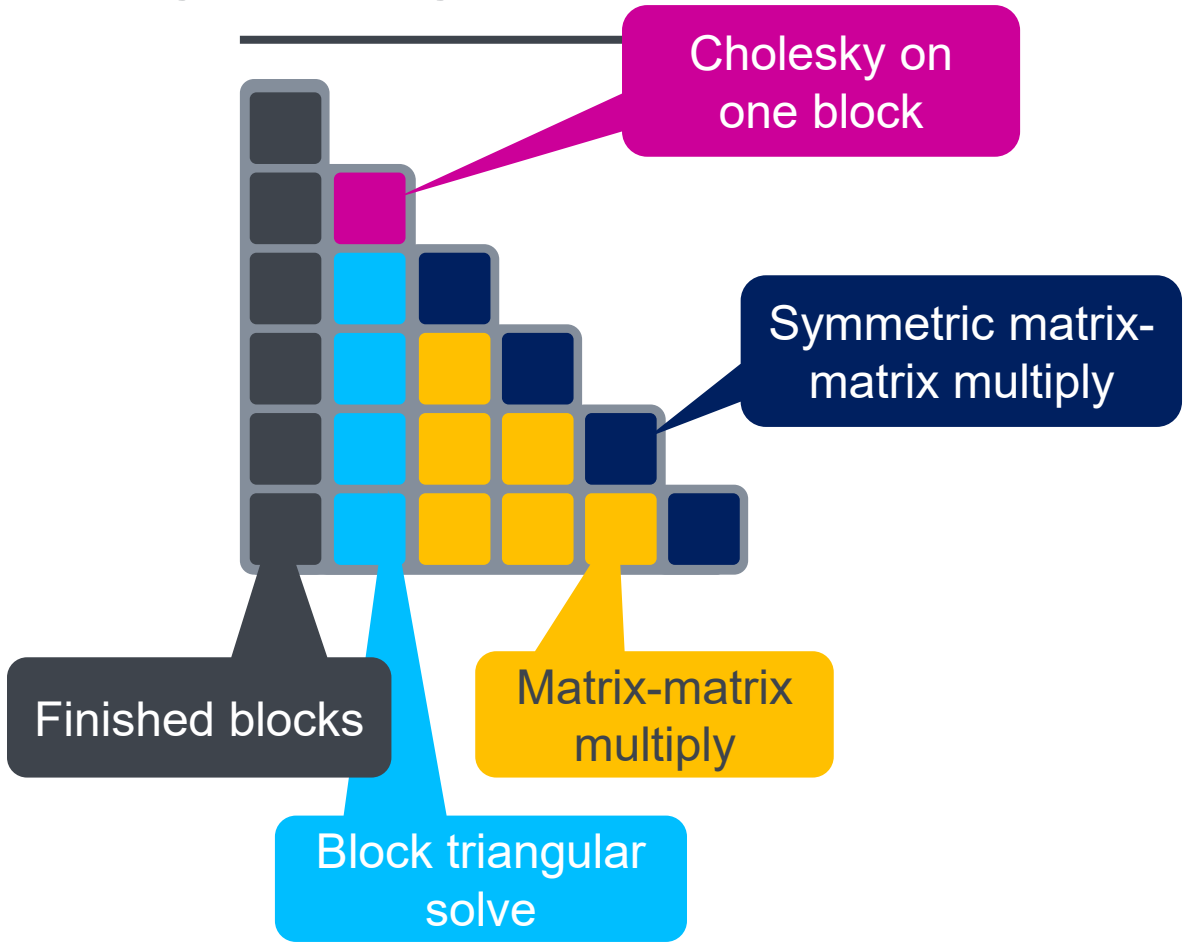
A_{mk} = solve(A_{kk}, A_{mk})

for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

A_{mk} = solve(A_{kk}, A_{mk})

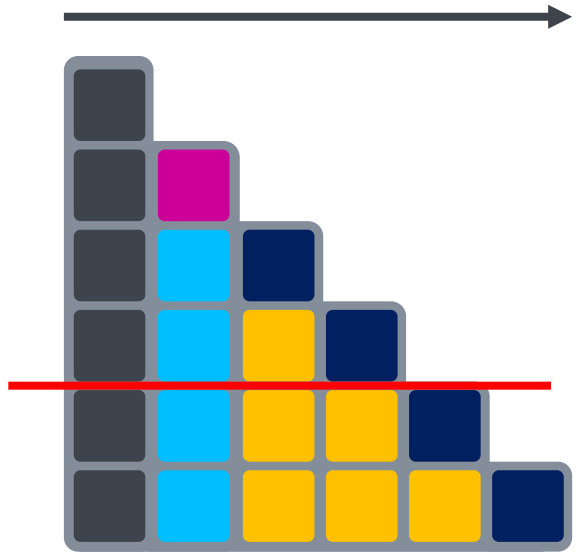
for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$

Right-looking: one column after another



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

A_{mk} = solve(A_{kk}, A_{mk})

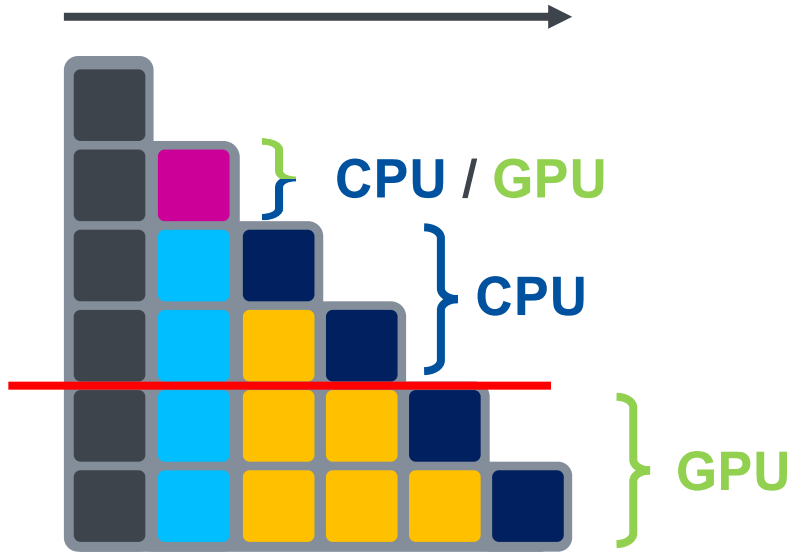
for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$

Right-looking: one column after another



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

A_{mk} = solve(A_{kk}, A_{mk})

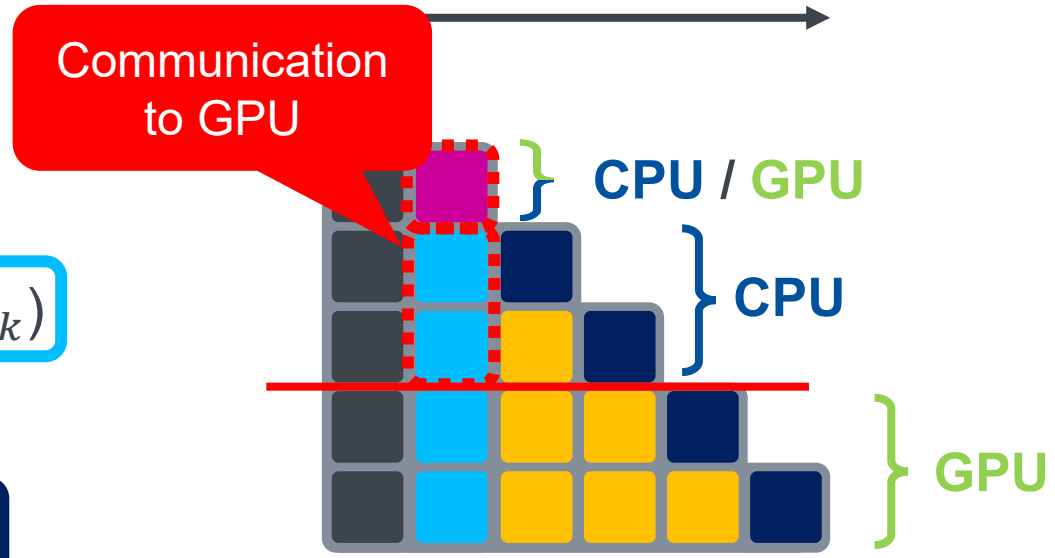
for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$

Right-looking: one column after another



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

A_{mk} = solve(A_{kk}, A_{mk})

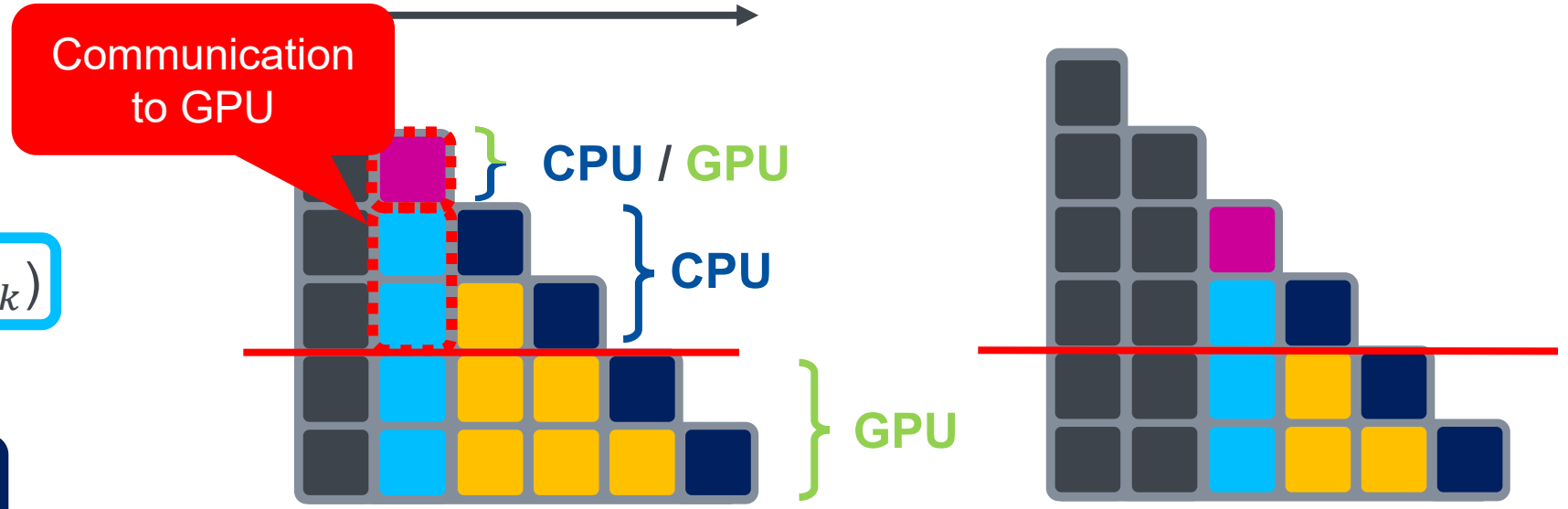
for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\top}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\top}$

Right-looking: one column after another



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

A_{mk} = solve(A_{kk}, A_{mk})

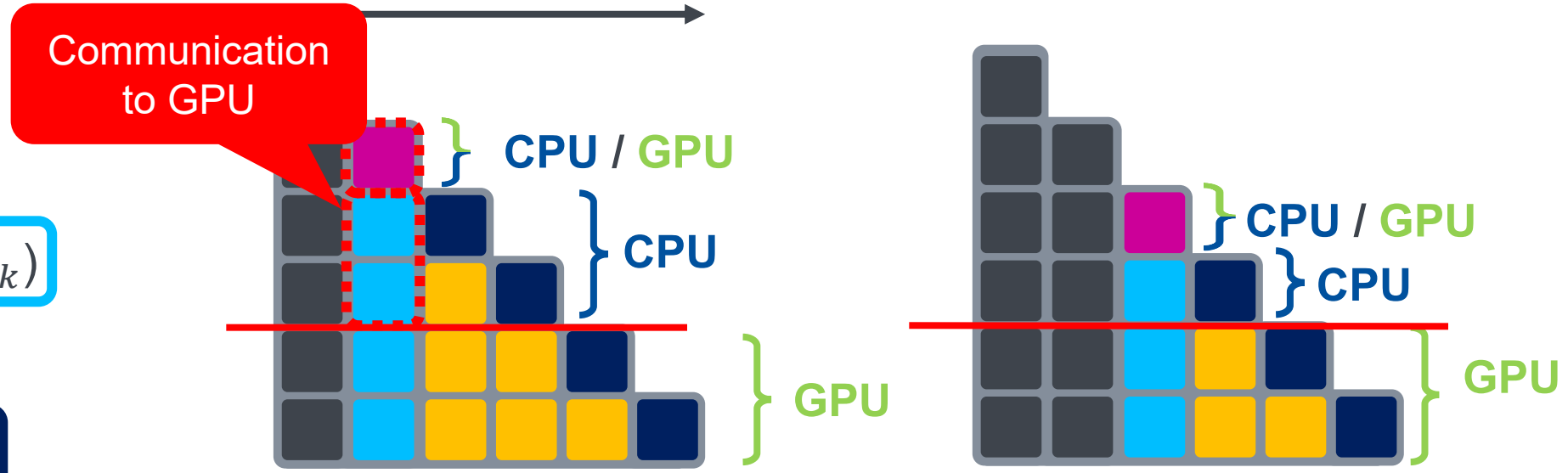
for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$

Right-looking: one column after another



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

A_{mk} = solve(A_{kk}, A_{mk})

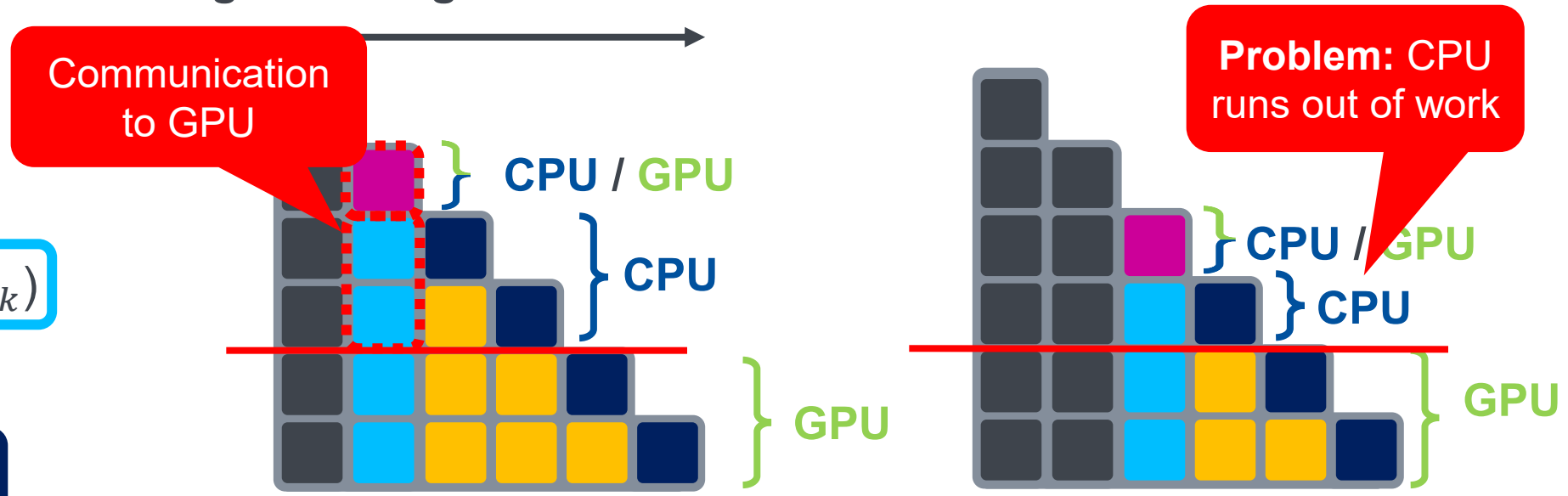
for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$

Right-looking: one column after another



Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

A_{mk} = solve(A_{kk}, A_{mk})

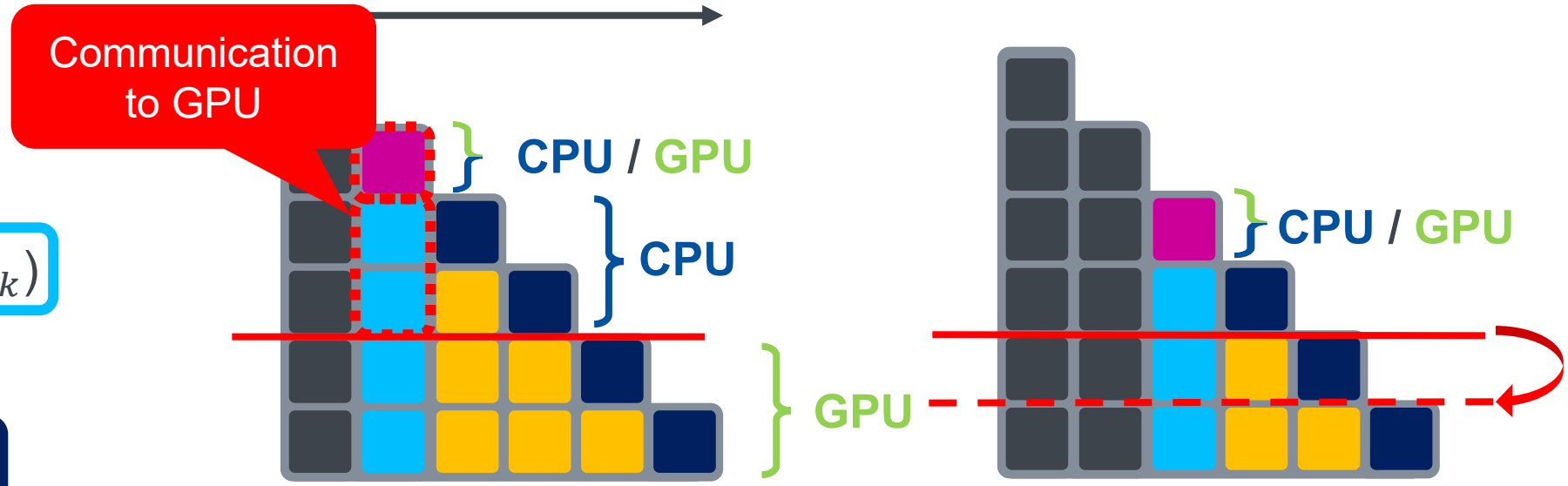
for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\top}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\top}$

Right-looking: one column after another



→ Shift split down every few iterations

Cholesky – blocked algorithm

- Factorizes an SPD matrix A into a lower triangular matrix L with $A = LL^T$

for k = 0…N-1

A_{kk} = Cholesky(A_{kk})

for m = k+1…N-1

A_{mk} = solve(A_{kk}, A_{mk})

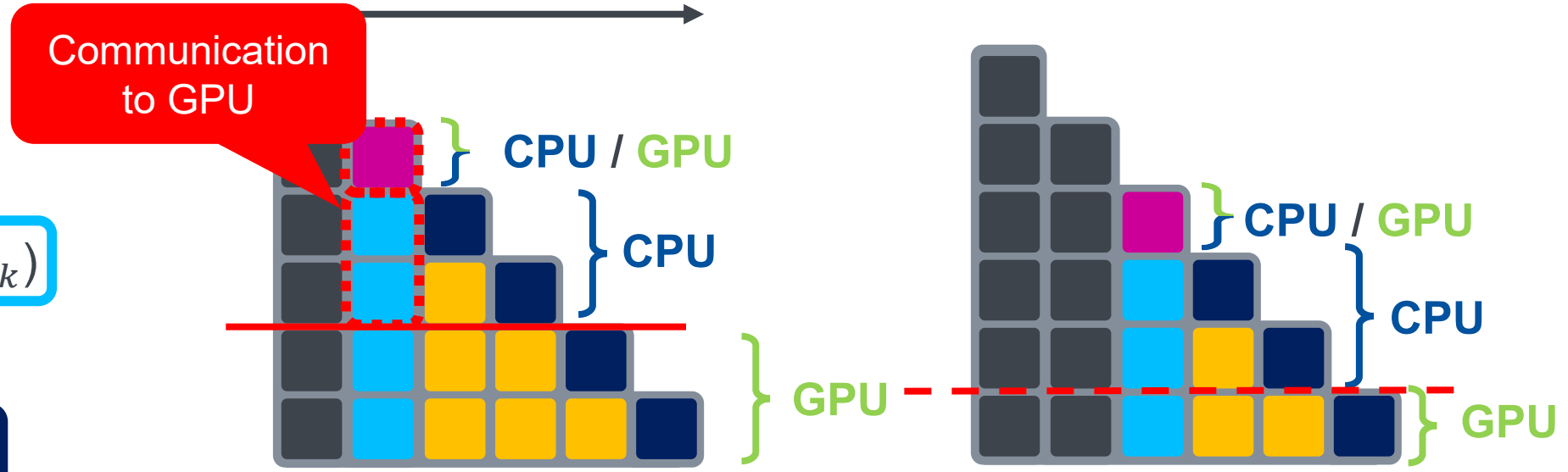
for m = k+1…N-1

$A_{mm} \mathrel{-}= A_{mk} \cdot A_{mk}^{\mathsf{T}}$

for n = k+1…m-1

$A_{mn} \mathrel{-}= A_{mk} \cdot A_{nk}^{\mathsf{T}}$

Right-looking: one column after another



→ Shift split down every few iterations

Conjugate Gradients and Cholesky

CG

Cholesky

Conjugate Gradients and Cholesky

CG

- Iterative solver to approximate the result

Cholesky

- Direct solver that produces exact solution

Conjugate Gradients and Cholesky

CG

- Iterative solver to approximate the result
- Dominated by BLAS-2 operations

Cholesky

- Direct solver that produces exact solution
- Dominated by BLAS-3 operations

Conjugate Gradients and Cholesky

CG

- Iterative solver to approximate the result
- Dominated by BLAS-2 operations
- Constant workload across iterations

Cholesky

- Direct solver that produces exact solution
- Dominated by BLAS-3 operations
- Workload decreases over time

Conjugate Gradients and Cholesky

CG

- Iterative solver to approximate the result
- Dominated by BLAS-2 operations
- Constant workload across iterations

Cholesky

- Direct solver that produces exact solution
- Dominated by BLAS-3 operations
- Workload decreases over time

How does heterogeneous computing affect the runtime of the two algorithms?

Experimental Setup

Experimental Setup

- Solving large systems of linear equations with dense kernel matrices obtained from simulation data for Gaussian Process Regression [2]
  - Sizes $2^{12} \times 2^{12}$ to $2^{16} \times 2^{16}$
- Results for two main Systems are shown:

[2] Helmann, Maksim, Alexander Strack, and Dirk Pflüger. 2025. “Replication Data for: GPRat: Gaussian Process Regression with Asynchronous Tasks.” DaRUS. https://doi.org/doi:10.18419/DARUS-4743.

Experimental Setup

- Solving large systems of linear equations with dense kernel matrices obtained from simulation data for Gaussian Process Regression [2]
  - Sizes $2^{12} \times 2^{12}$ to $2^{16} \times 2^{16}$
- Results for two main Systems are shown:

	System 1	System 2
CPU	**2x AMD EPYC 9274F (48 Cores)**	
CPU FP64 FLOPS	**3.11** TFLOPS	

[2] Helmann, Maksim, Alexander Strack, and Dirk Pflüger. 2025. “Replication Data for: GPRat: Gaussian Process Regression with Asynchronous Tasks.” DaRUS. https://doi.org/doi:10.18419/DARUS-4743.

Experimental Setup

- Solving large systems of linear equations with dense kernel matrices obtained from simulation data for Gaussian Process Regression [2]
  - Sizes $2^{12} \times 2^{12}$ to $2^{16} \times 2^{16}$
- Results for two main Systems are shown:

	System 1	System 2
CPU	**2x AMD EPYC 9274F (48 Cores)**	
CPU FP64 FLOPS	**3.11** TFLOPS	
GPU	**NVIDIA A30**	**AMD MI210**
GPU FP64 FLOPS	**5.2** TFLOPS	**22.6** TFLOPS

[2] Helmann, Maksim, Alexander Strack, and Dirk Pflüger. 2025. “Replication Data for: GPRat: Gaussian Process Regression with Asynchronous Tasks.” DaRUS. https://doi.org/doi:10.18419/DARUS-4743.

Experimental Setup

- Solving large systems of linear equations with dense kernel matrices obtained from simulation data for Gaussian Process Regression [2]
  - Sizes $2^{12} \times 2^{12}$ to $2^{16} \times 2^{16}$
- Results for two main Systems are shown:

	System 1	System 2
CPU	**2x AMD EPYC 9274F (48 Cores)**	
CPU FP64 FLOPS	**3.11** TFLOPS	
GPU	**NVIDIA A30**	**AMD MI210**
GPU FP64 FLOPS	**5.2** TFLOPS	**22.6** TFLOPS

- All computations are performed using double precision

[2] Helmann, Maksim, Alexander Strack, and Dirk Pflüger. 2025. “Replication Data for: GPRat: Gaussian Process Regression with Asynchronous Tasks.” DaRUS. https://doi.org/doi:10.18419/DARUS-4743.

Experimental Setup

- Solving large systems of linear equations with dense kernel matrices obtained from simulation data for Gaussian Process Regression [2]
  - Sizes $2^{12} \times 2^{12}$ to $2^{16} \times 2^{16}$
- Results for two main Systems are shown:

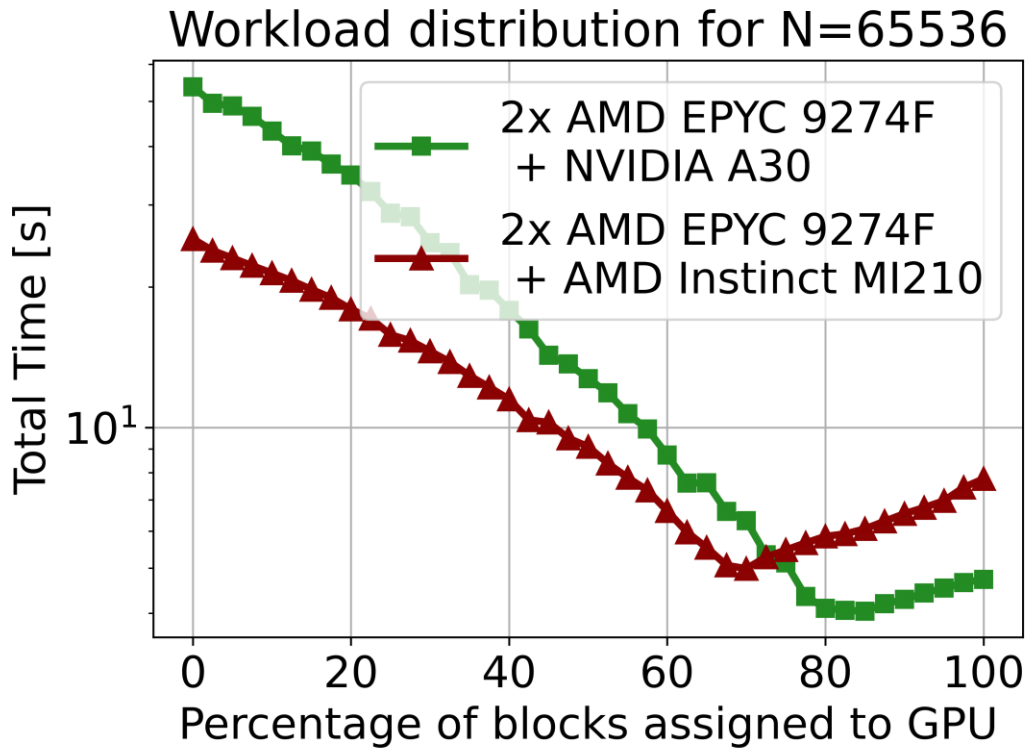
	System 1	System 2
CPU	**2x AMD EPYC 9274F (48 Cores)**	
CPU FP64 FLOPS	**3.11** TFLOPS	
GPU	**NVIDIA A30**	**AMD MI210**
GPU FP64 FLOPS	**5.2** TFLOPS	**22.6** TFLOPS

- All computations are performed using double precision
- AdaptiveCpp is used if not stated otherwise

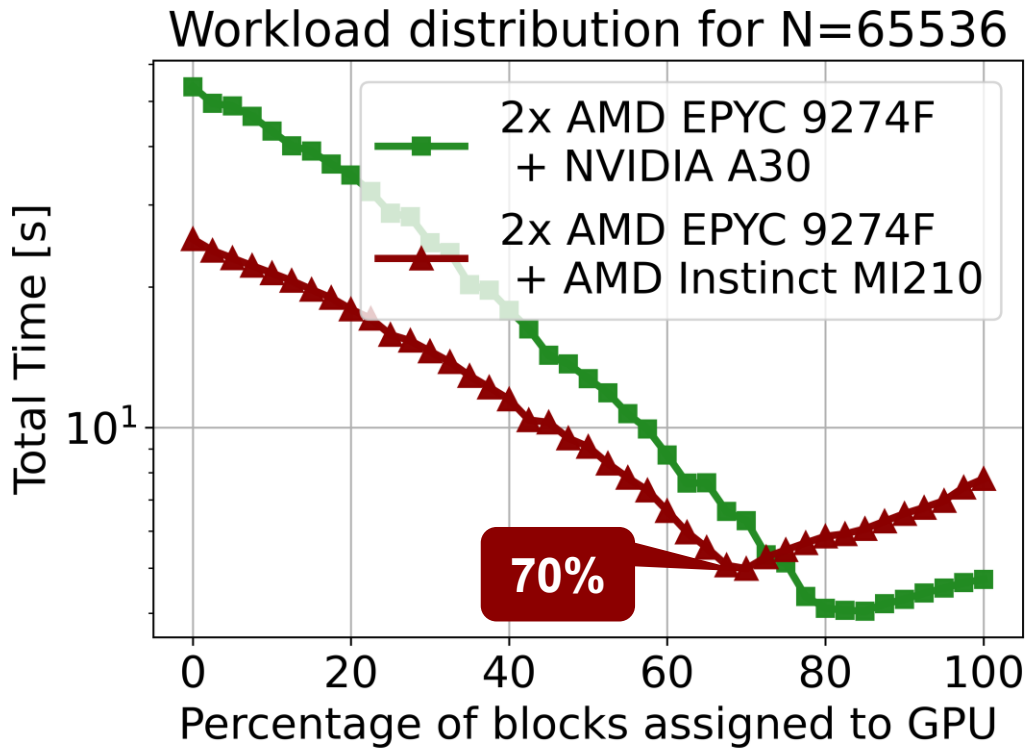
[2] Helmann, Maksim, Alexander Strack, and Dirk Pflüger. 2025. “Replication Data for: GPRat: Gaussian Process Regression with Asynchronous Tasks.” DaRUS. https://doi.org/doi:10.18419/DARUS-4743.

Runtime Evaluation: Heterogeneous CG Algorithm

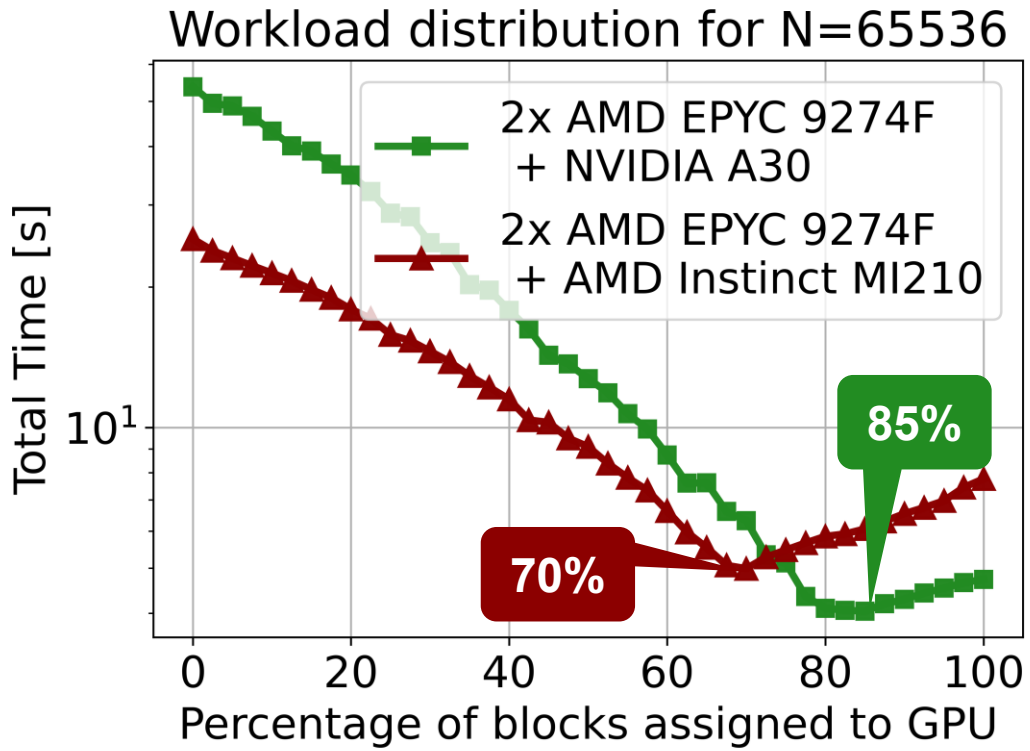
Runtime Evaluation: Heterogeneous CG Algorithm



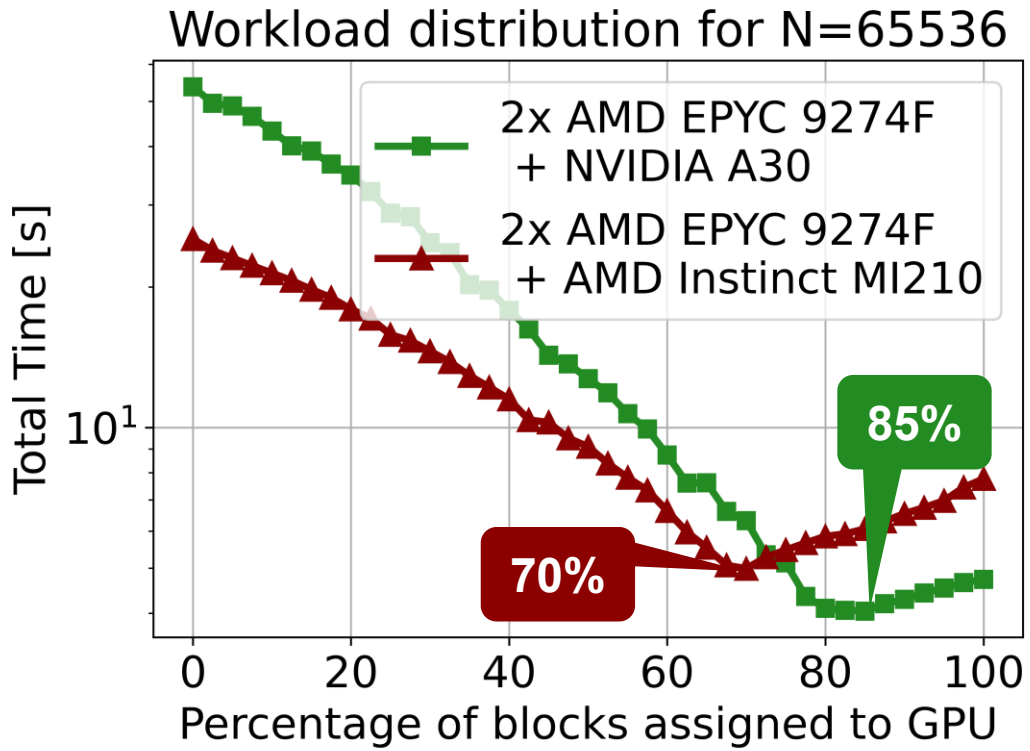
Runtime Evaluation: Heterogeneous CG Algorithm

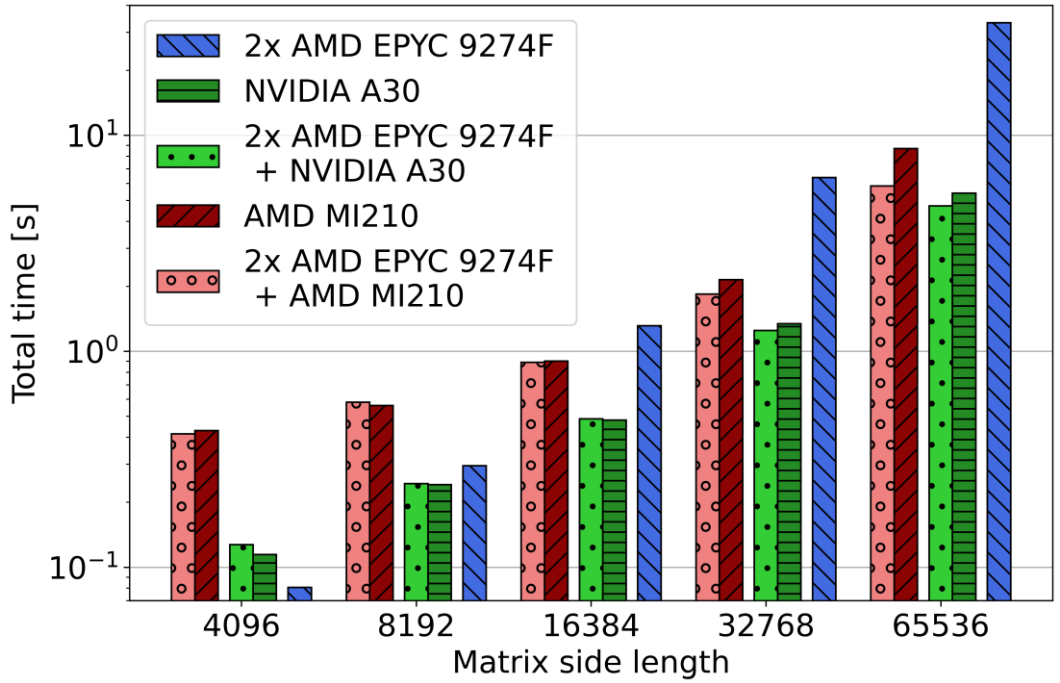


Runtime Evaluation: Heterogeneous CG Algorithm

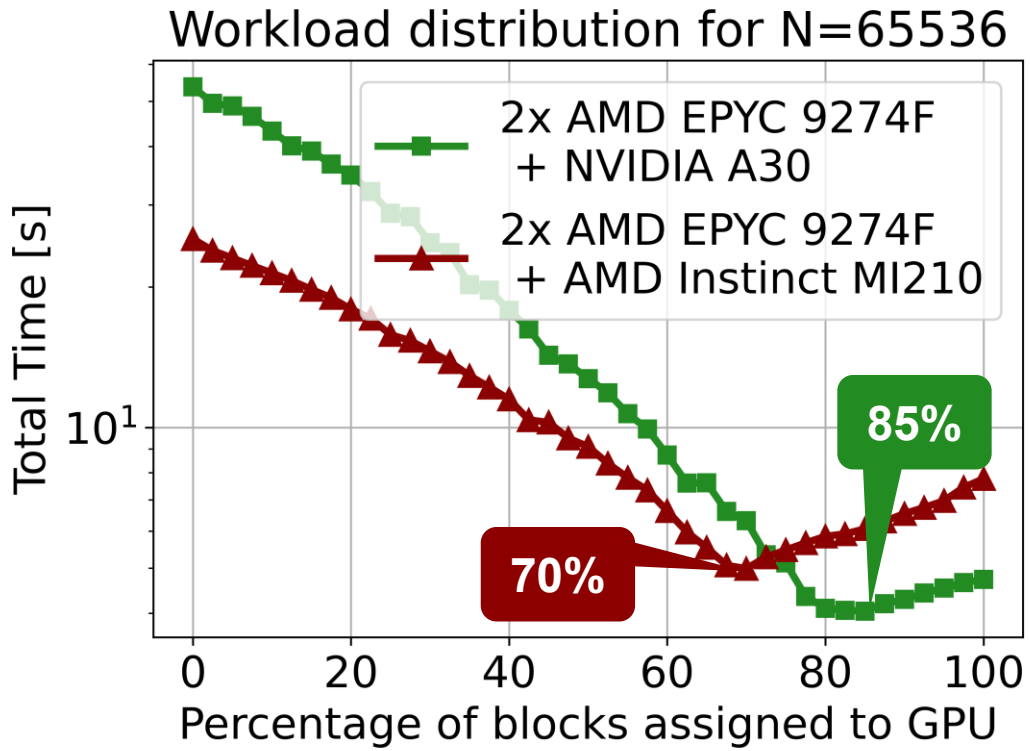


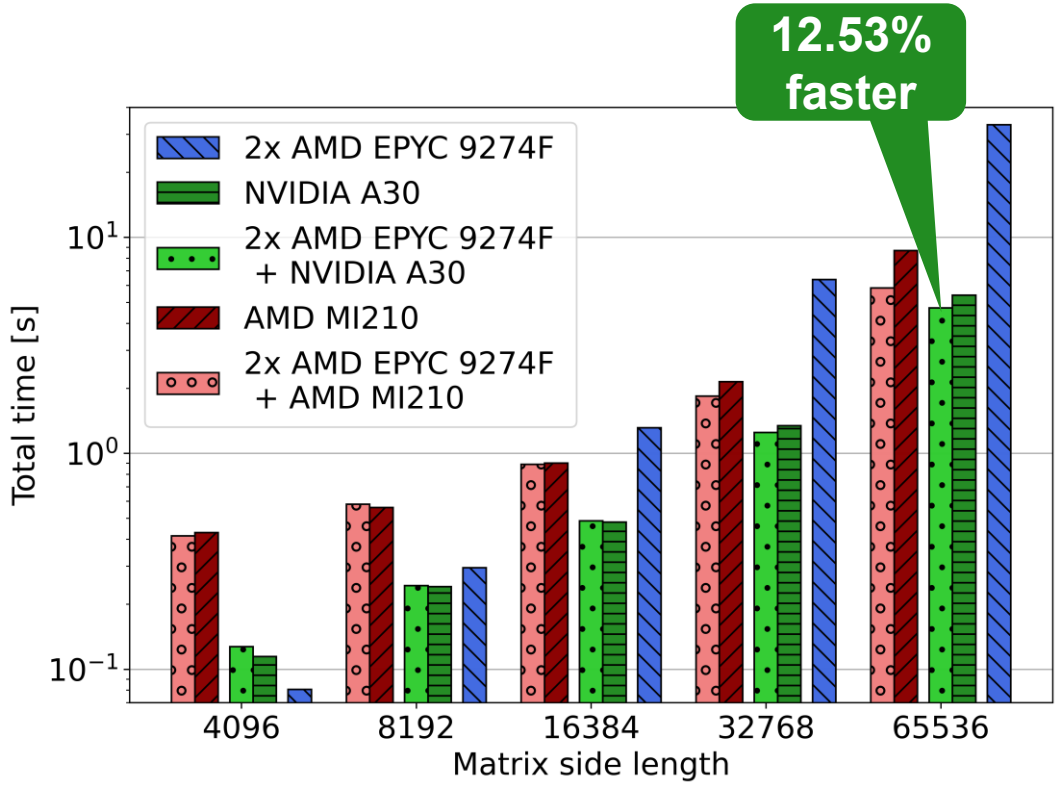
Runtime Evaluation: Heterogeneous CG Algorithm



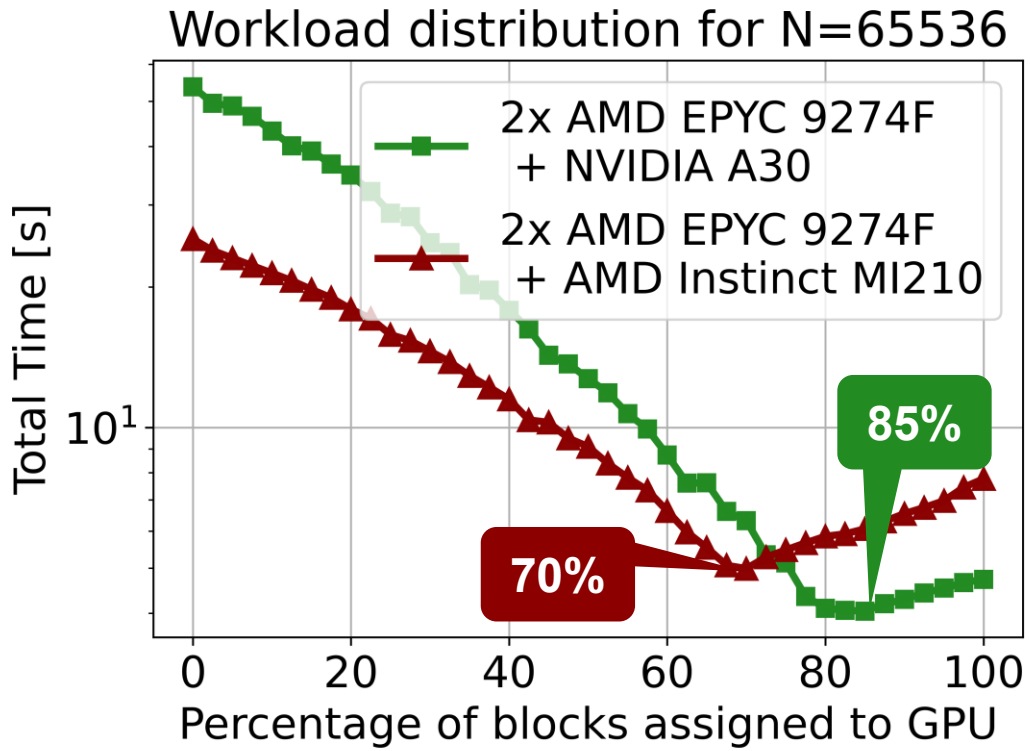


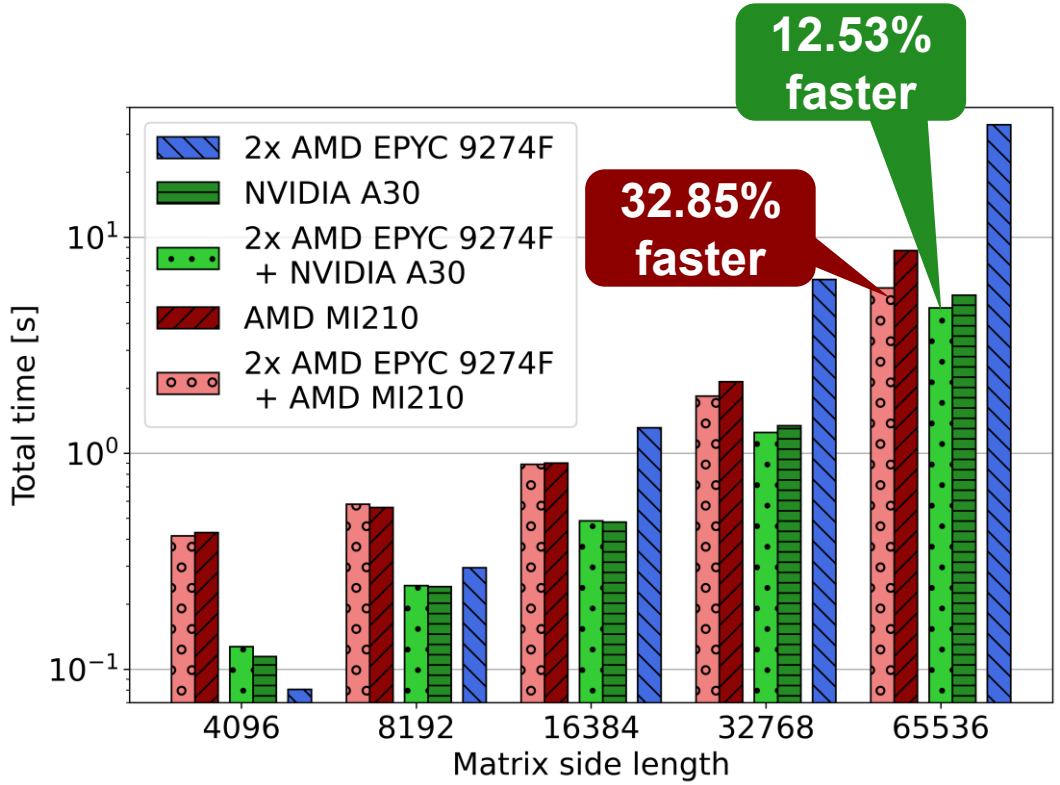
Runtime Evaluation: Heterogeneous CG Algorithm





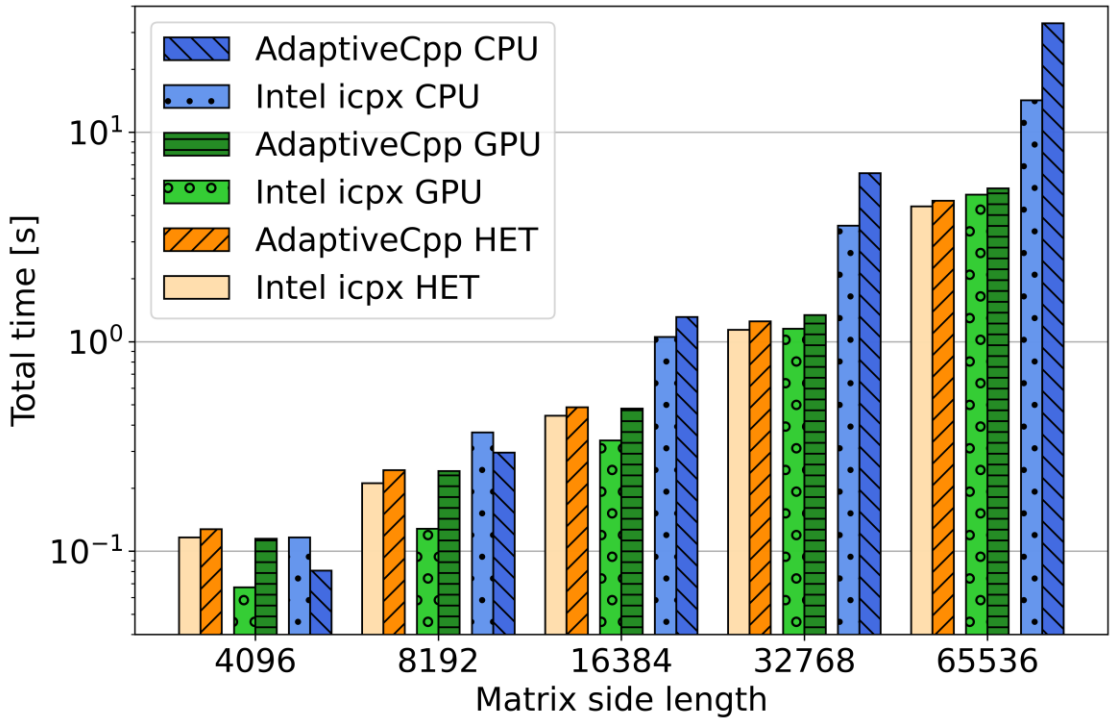
Runtime Evaluation: Heterogeneous CG Algorithm





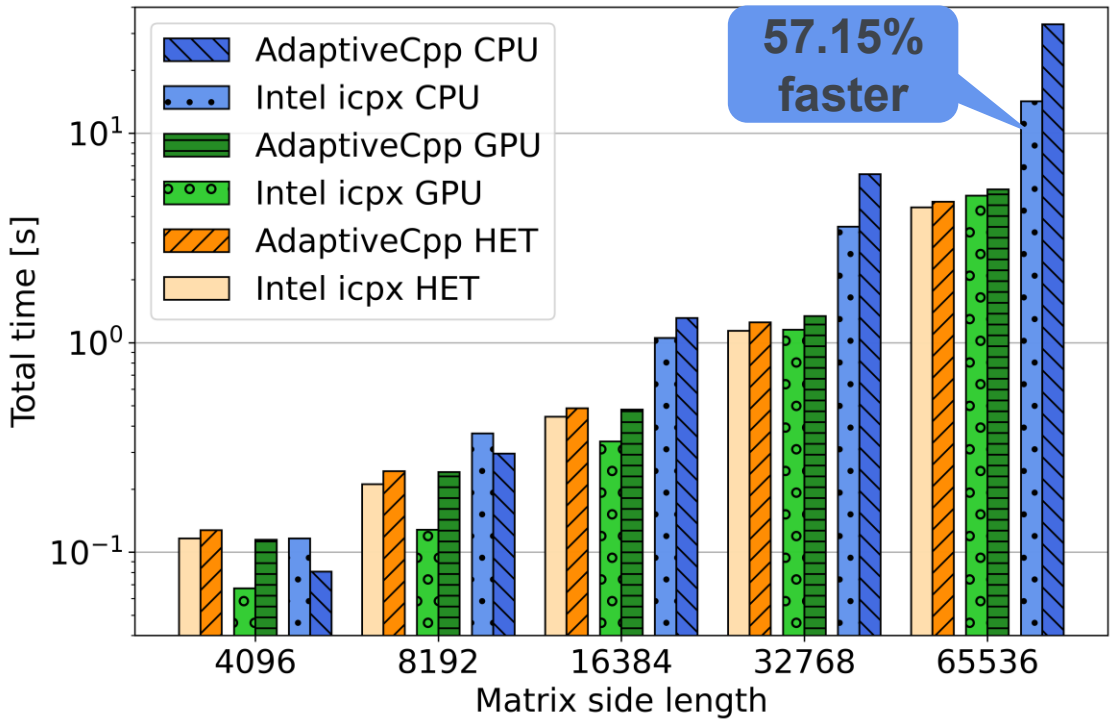
CG Algorithm: Comparison of Intel icpx and AdaptiveCpp

CG Algorithm: Comparison of Intel icpx and AdaptiveCpp



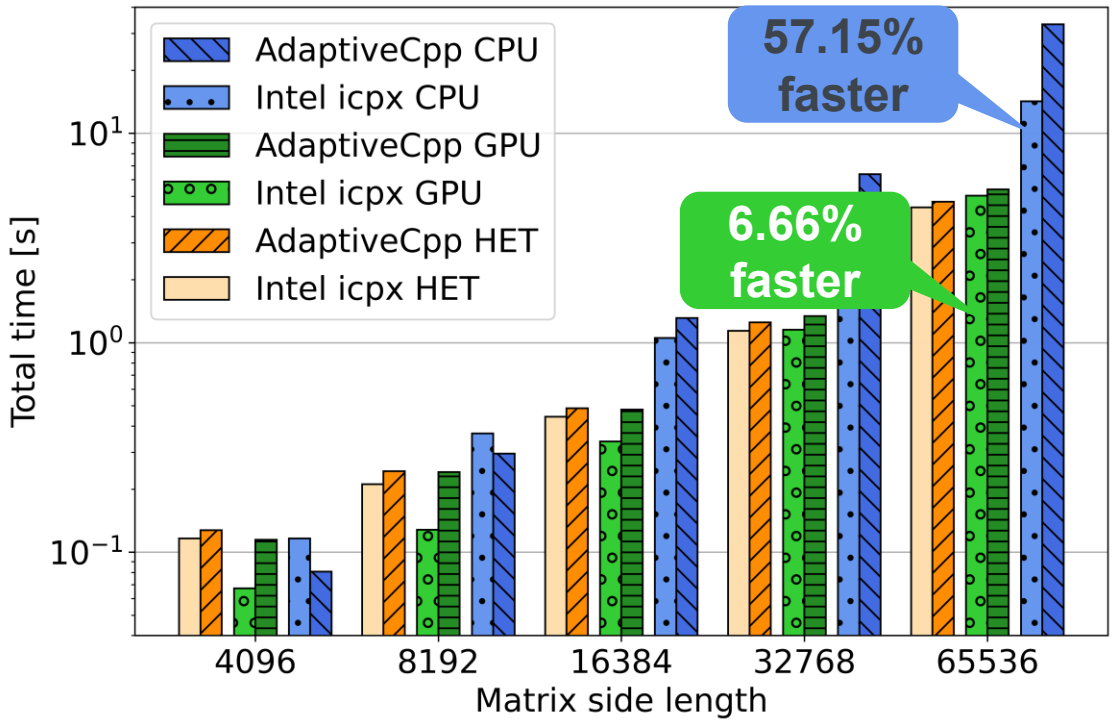
System 1 (NVIDIA A30)

CG Algorithm: Comparison of Intel icpx and AdaptiveCpp



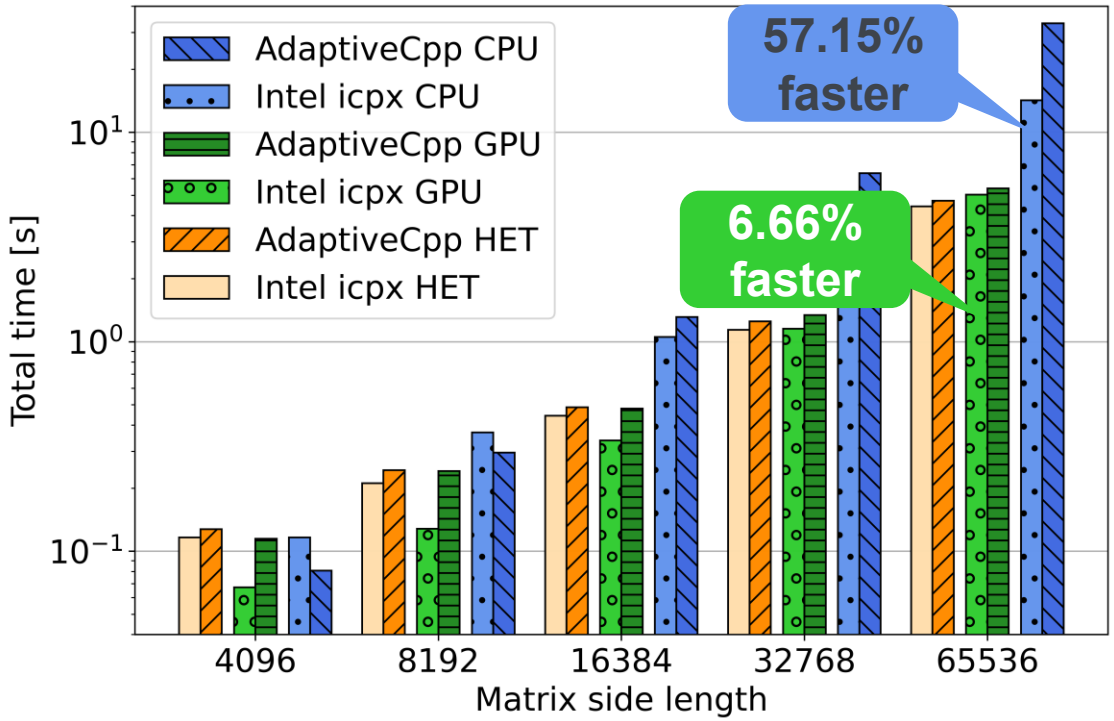
System 1 (NVIDIA A30)

CG Algorithm: Comparison of Intel icpx and AdaptiveCpp

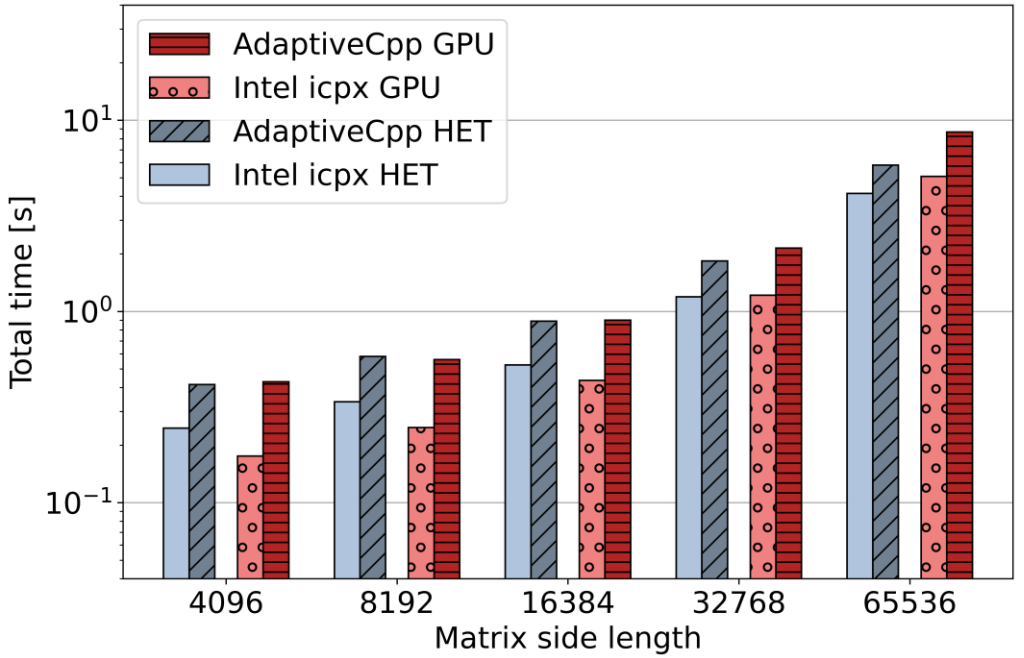


System 1 (NVIDIA A30)

CG Algorithm: Comparison of Intel icpx and AdaptiveCpp

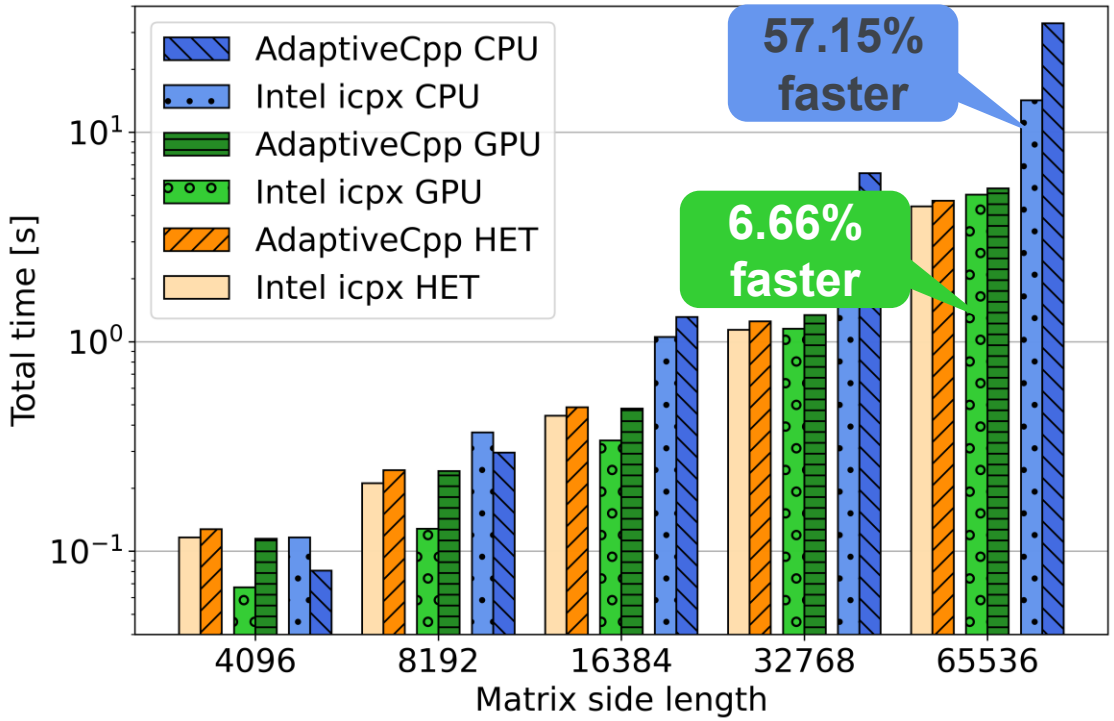


System 1 (NVIDIA A30)

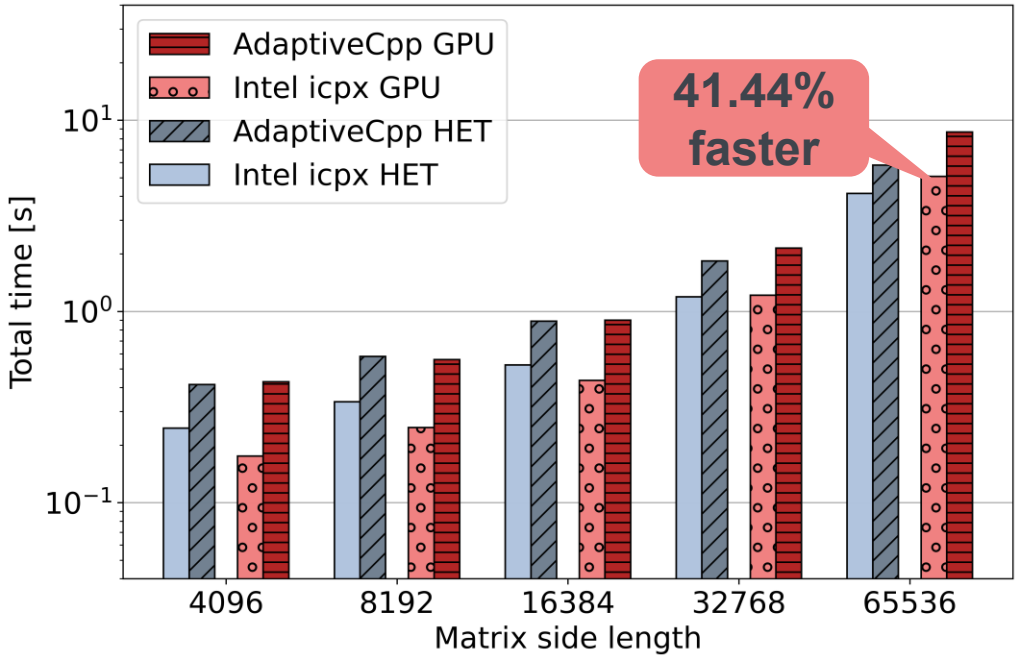


System 2 (AMD MI210)

CG Algorithm: Comparison of Intel icpx and AdaptiveCpp



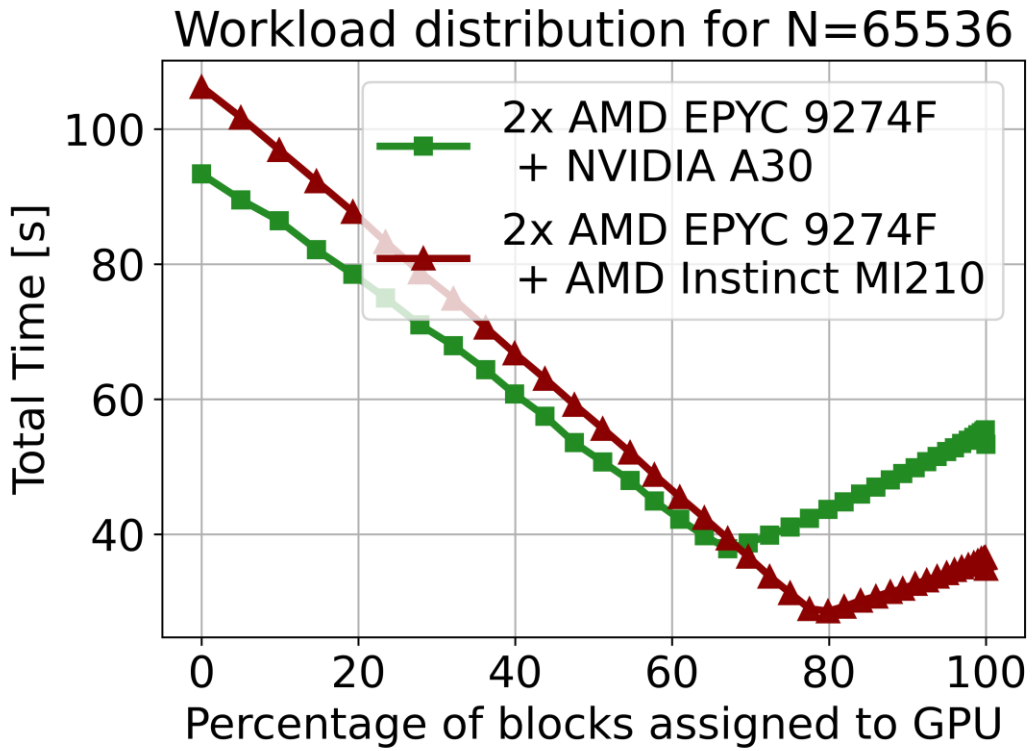
System 1 (NVIDIA A30)



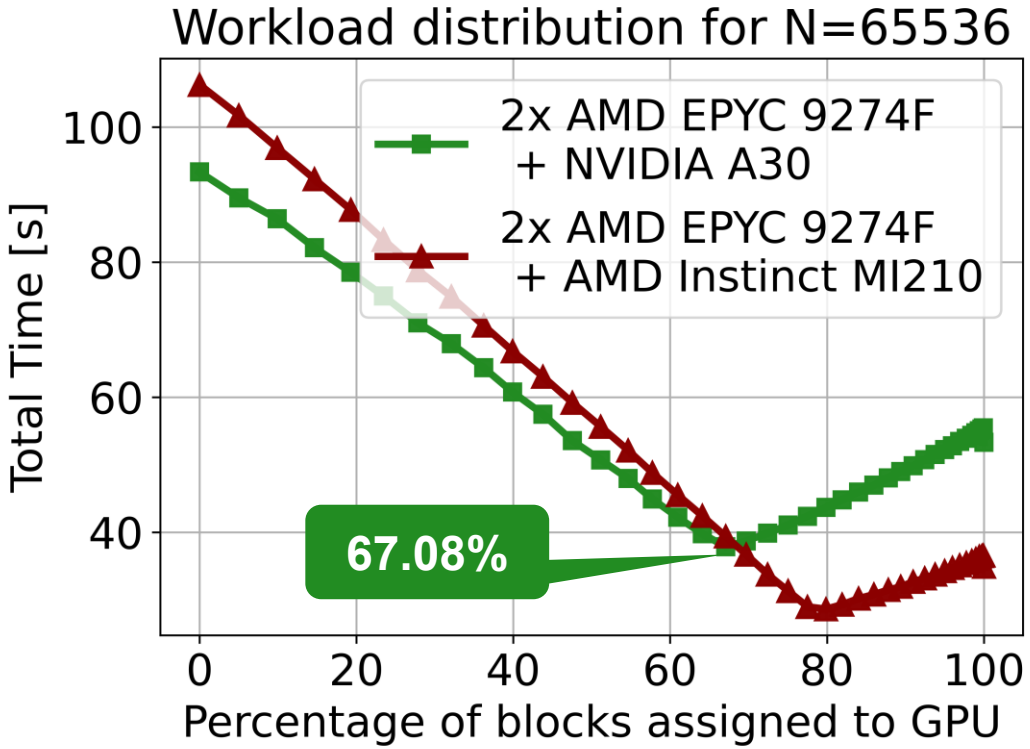
System 2 (AMD MI210)

Runtime Evaluation: Heterogeneous Cholesky Decomposition

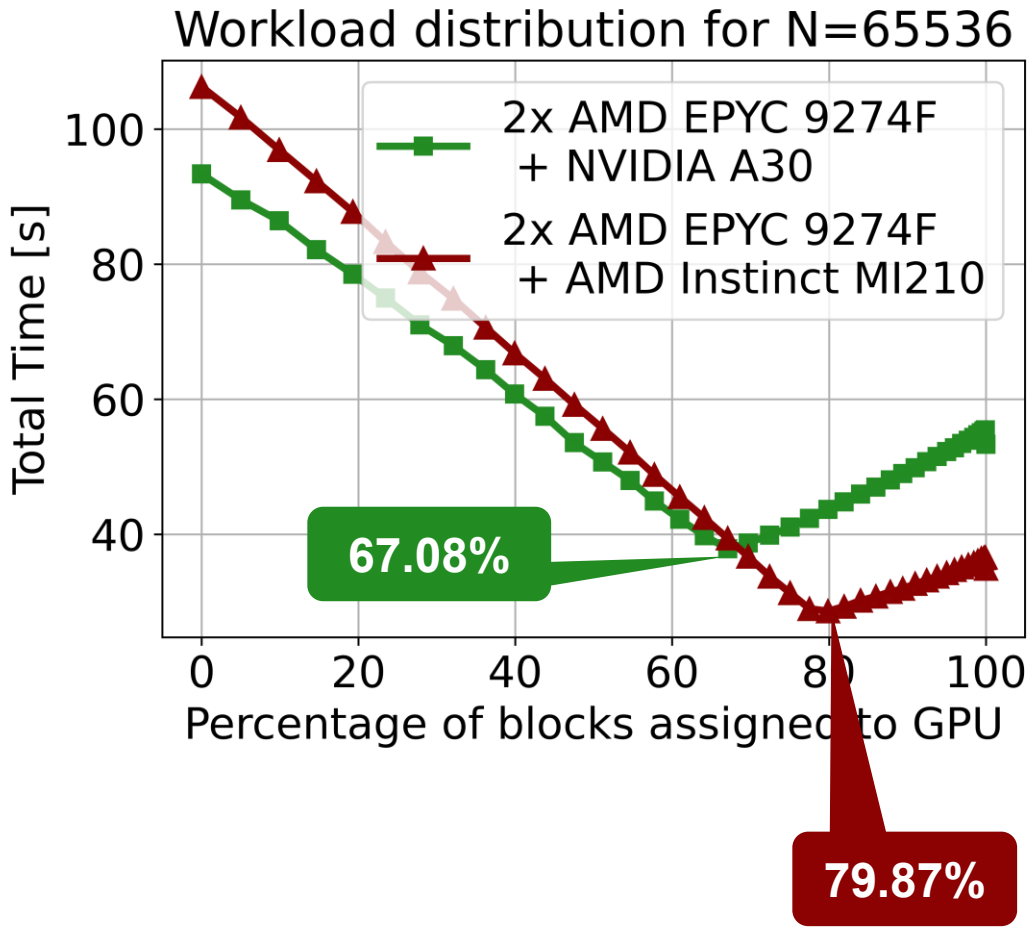
Runtime Evaluation: Heterogeneous Cholesky Decomposition



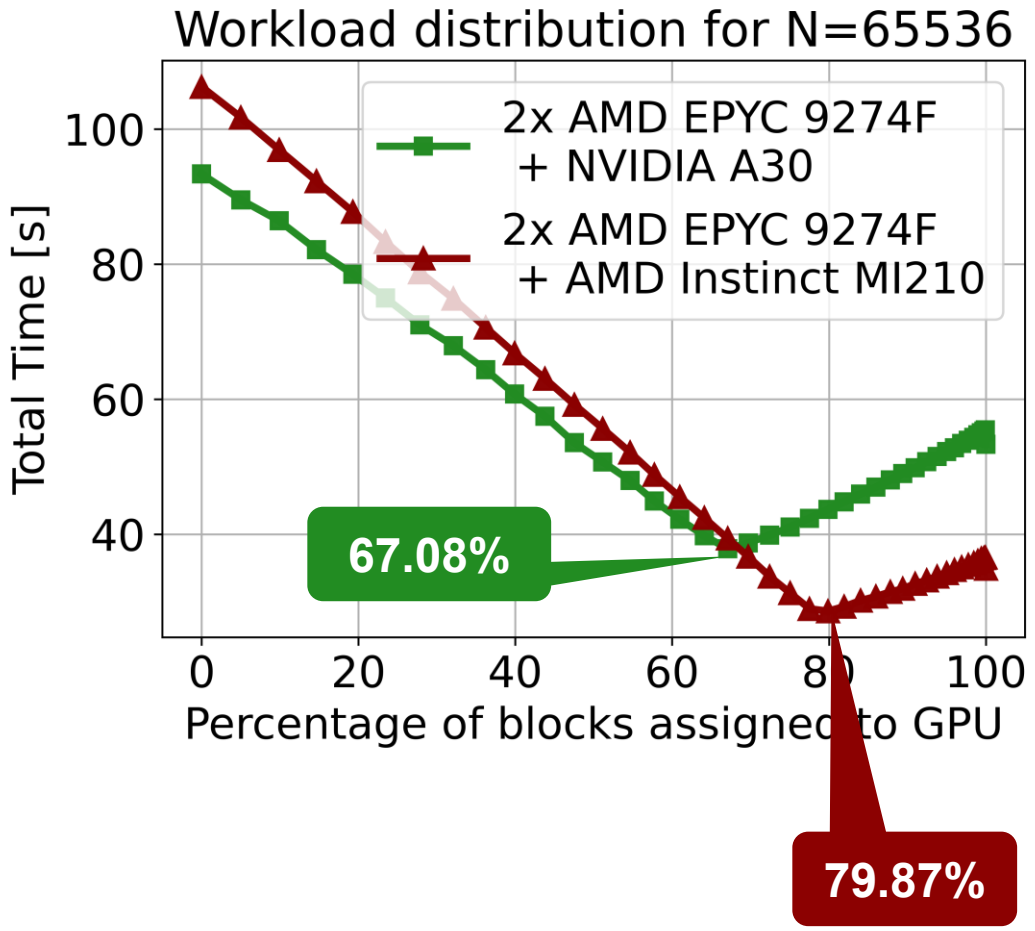
Runtime Evaluation: Heterogeneous Cholesky Decomposition

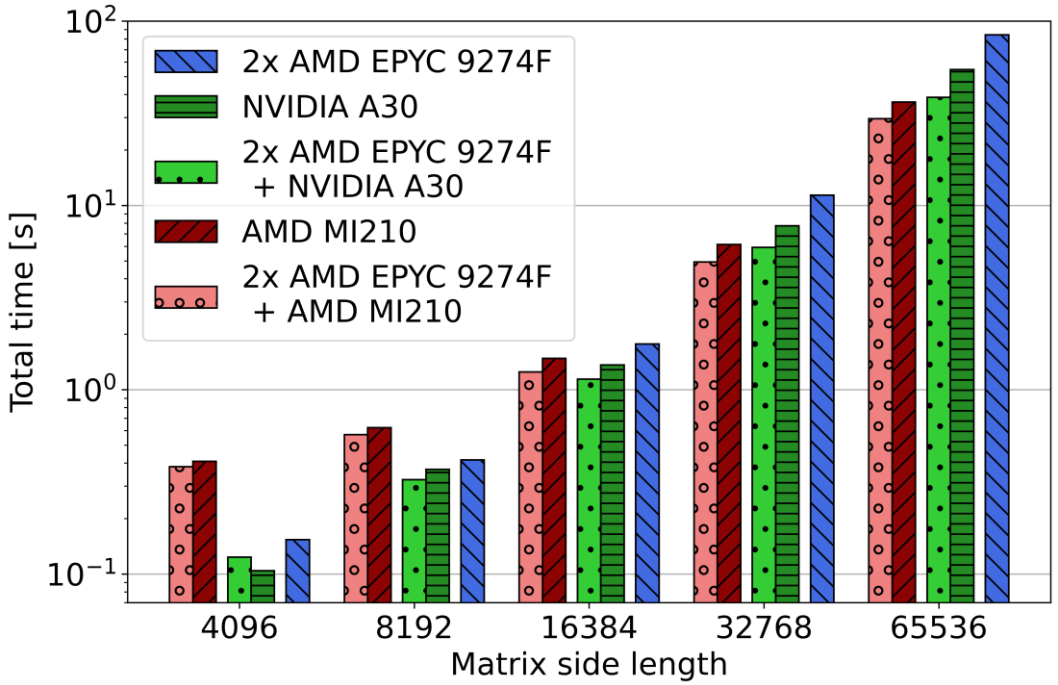


Runtime Evaluation: Heterogeneous Cholesky Decomposition

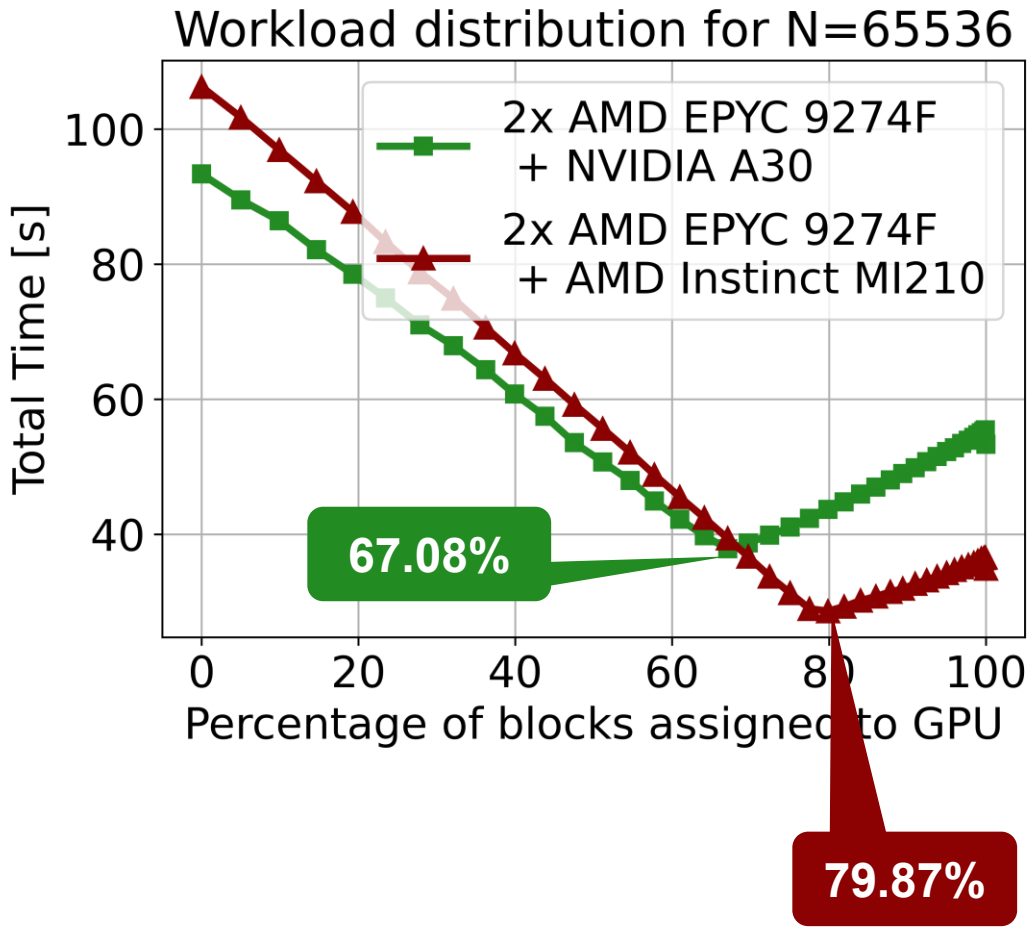


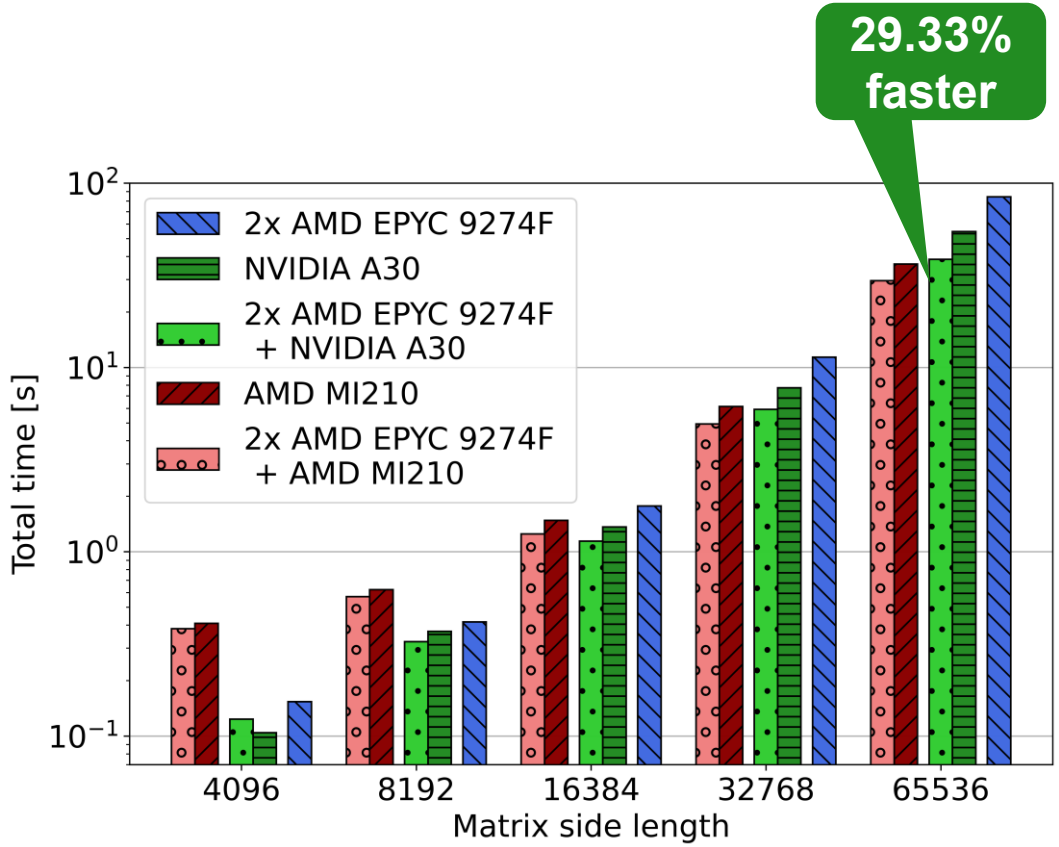
Runtime Evaluation: Heterogeneous Cholesky Decomposition



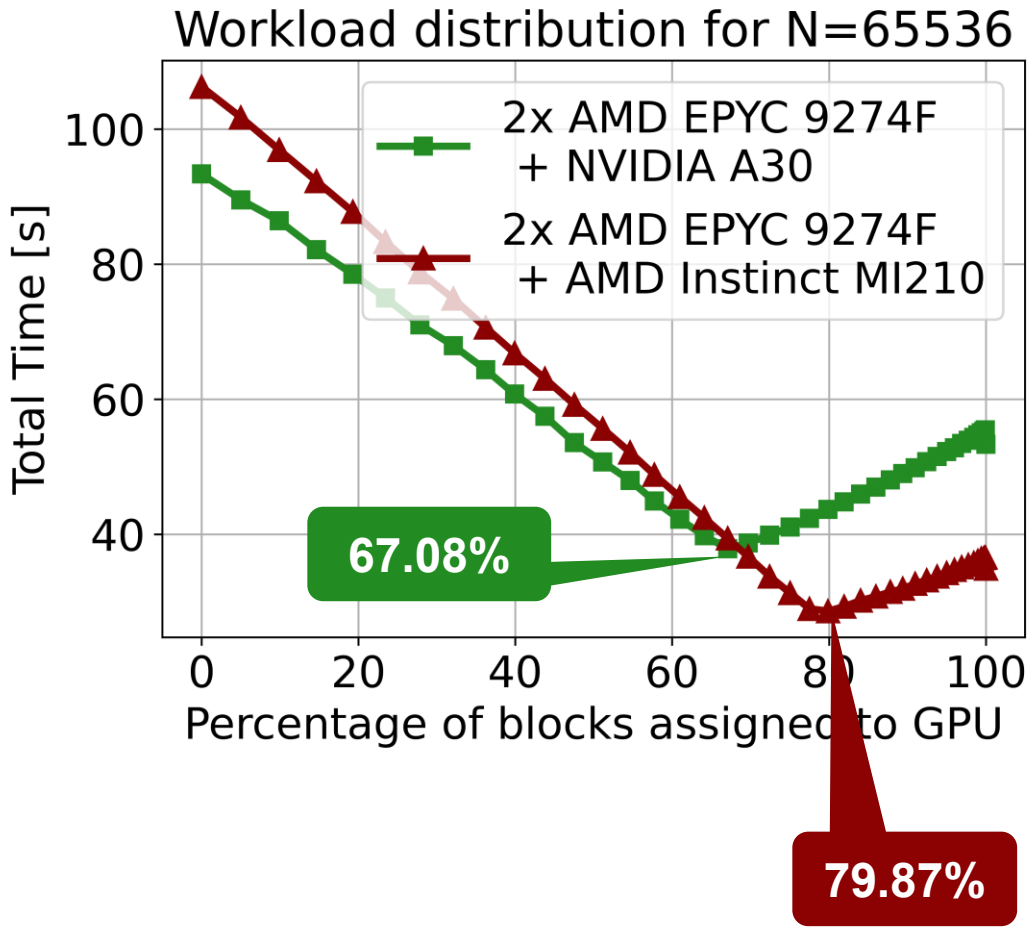


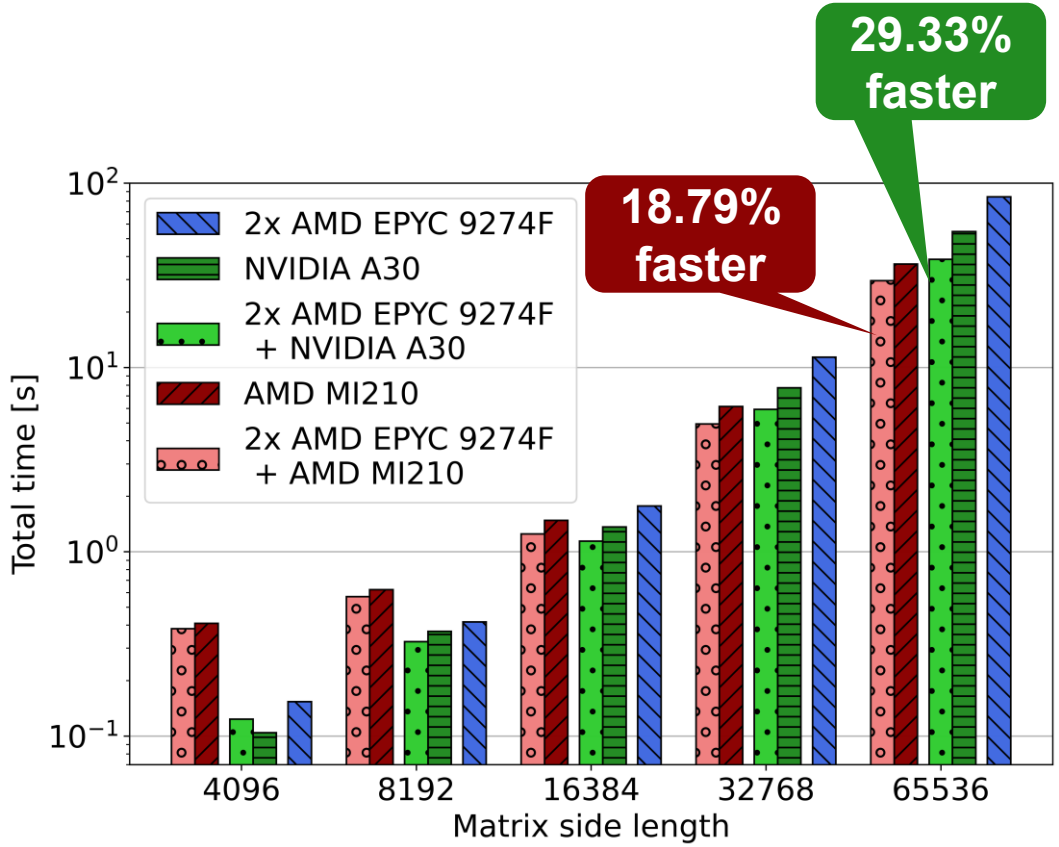
Runtime Evaluation: Heterogeneous Cholesky Decomposition





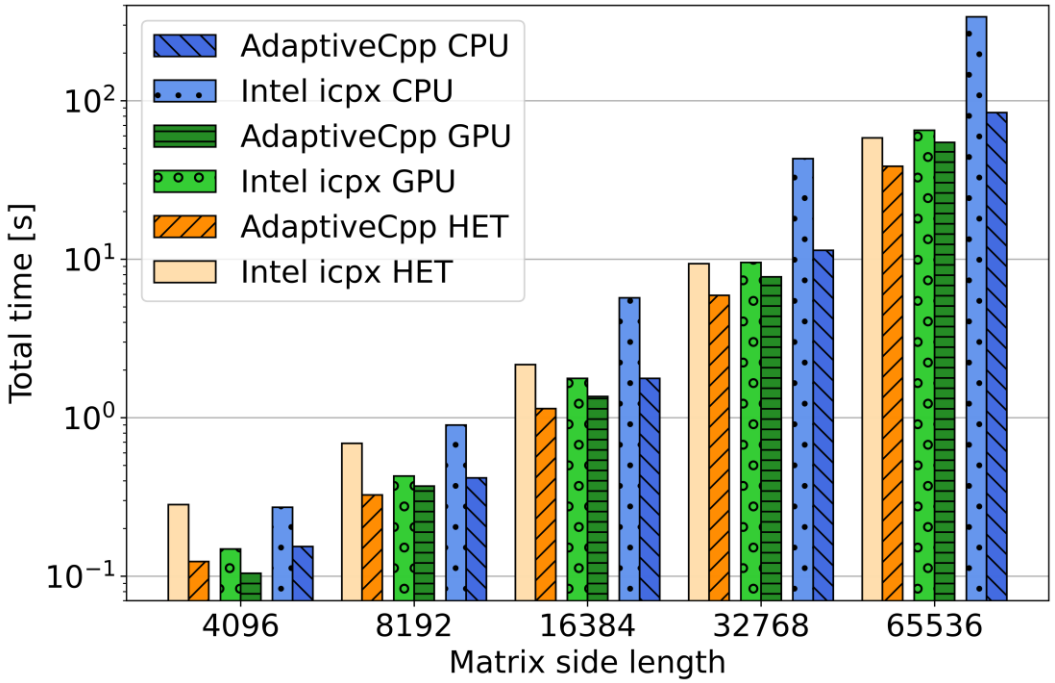
Runtime Evaluation: Heterogeneous Cholesky Decomposition





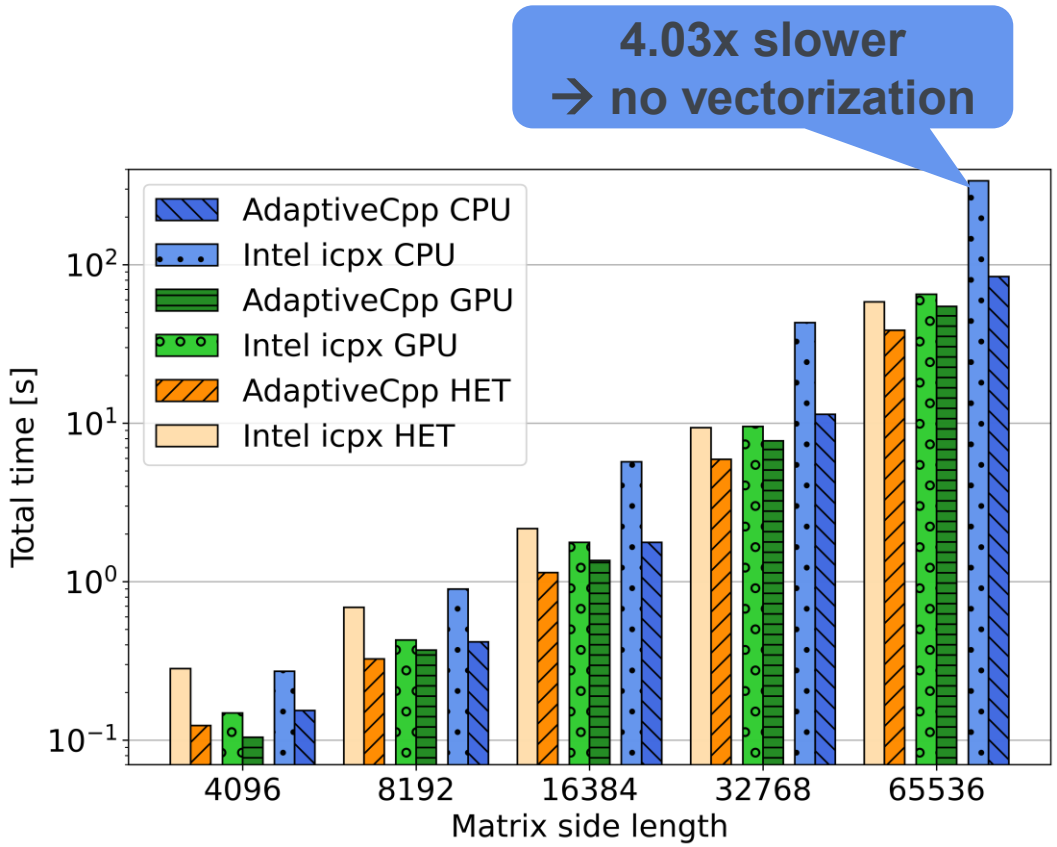
Cholesky Decomposition: Comparison of Intel icpx and AdaptiveCpp

Cholesky Decomposition: Comparison of Intel icpx and AdaptiveCpp



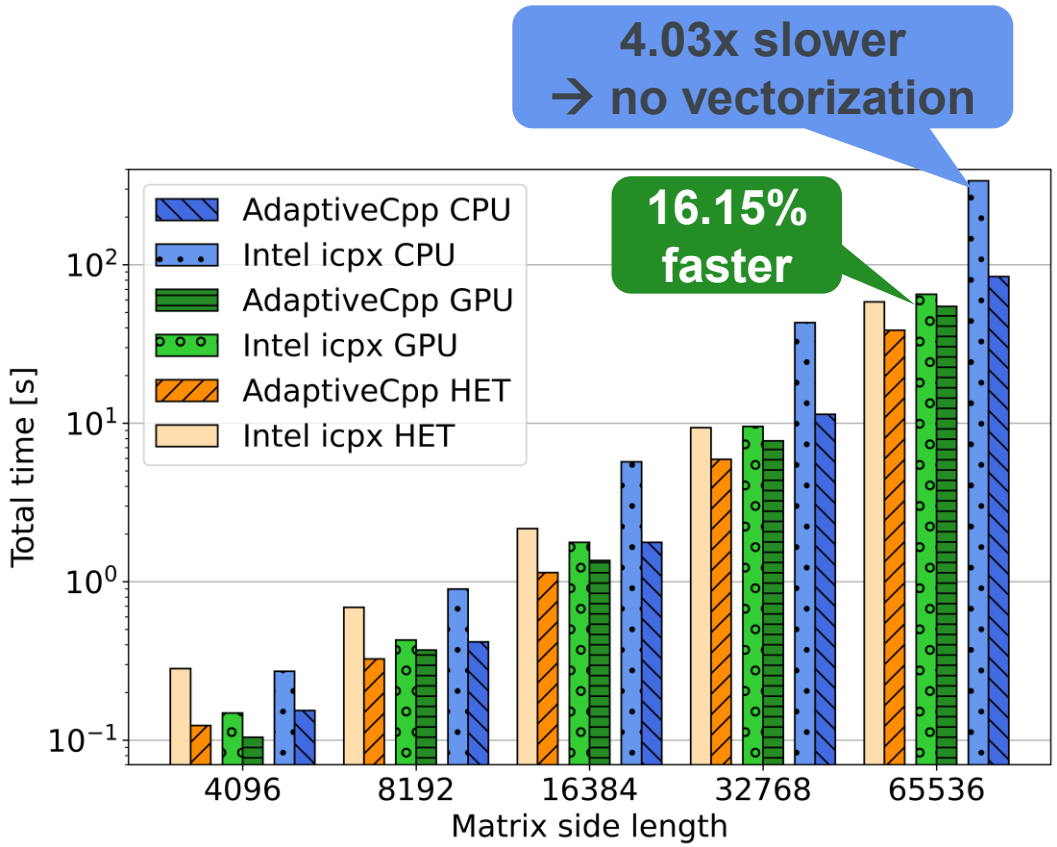
System 1 (NVIDIA A30)

Cholesky Decomposition: Comparison of Intel icpx and AdaptiveCpp



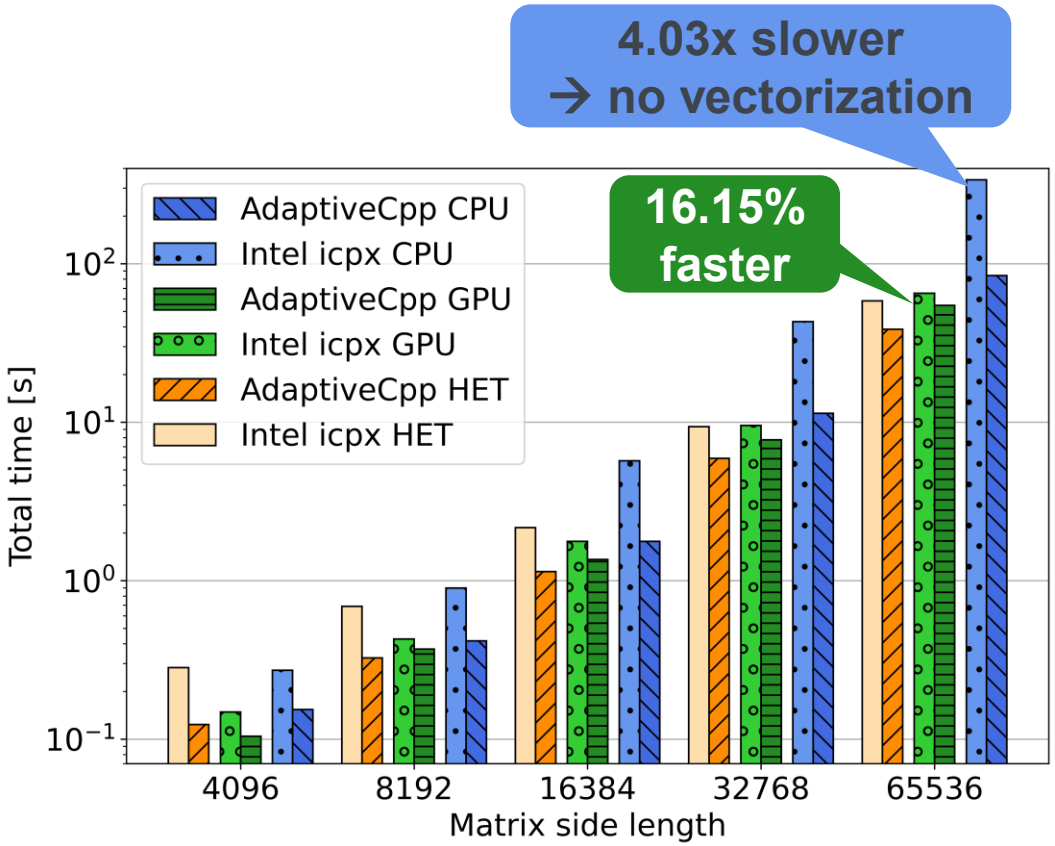
System 1 (NVIDIA A30)

Cholesky Decomposition: Comparison of Intel icpx and AdaptiveCpp

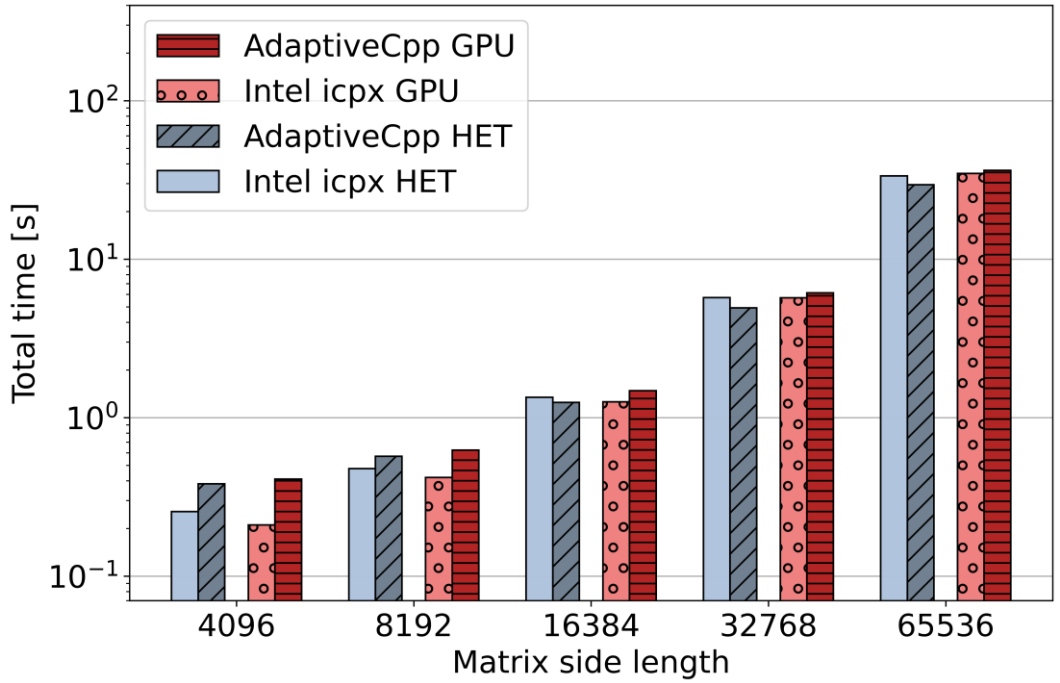


System 1 (NVIDIA A30)

Cholesky Decomposition: Comparison of Intel icpx and AdaptiveCpp

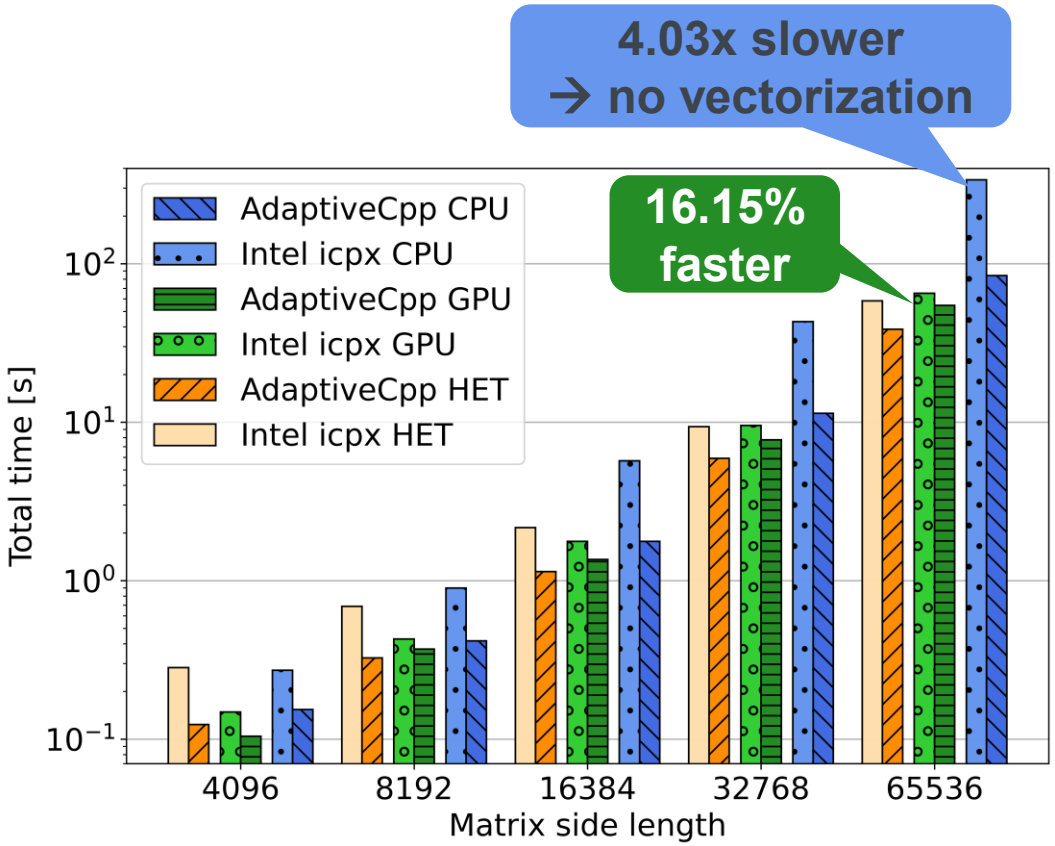


System 1 (NVIDIA A30)

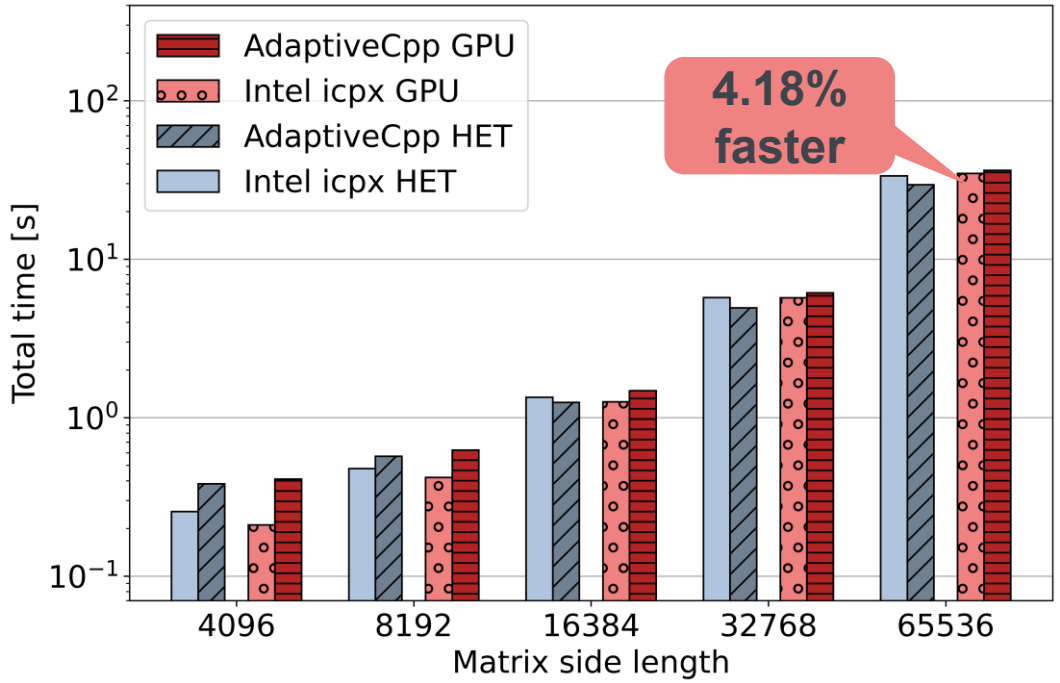


System 2 (AMD MI210)

Cholesky Decomposition: Comparison of Intel icpx and AdaptiveCpp



System 1 (NVIDIA A30)



System 2 (AMD MI210)

Conclusion

Conclusion

- Considerable performance improvements can be achieved with heterogeneous computing

Conclusion

- Considerable performance improvements can be achieved with heterogeneous computing
- Improvements compared to GPU-only on various systems for large matrices:

Conclusion

- Considerable performance improvements can be achieved with heterogeneous computing
- Improvements compared to GPU-only on various systems for large matrices:

	CG	Cholesky
2x AMD EPYC 9274 + **NVIDIA A30**	12.53% (0.68s)	29.33% (15.99s)
2x AMD EPYC 9274 + **AMD MI210**	32.85% (2.85s)	18.79% (6.82s)

Conclusion

- Considerable performance improvements can be achieved with heterogeneous computing
- Improvements compared to GPU-only on various systems for large matrices:

	CG	Cholesky
$2^{16} \times 2^{16}$		
2x AMD EPYC 9274 + **NVIDIA A30**	12.53% (0.68s)	29.33% (15.99s)
2x AMD EPYC 9274 + **AMD MI210**	32.85% (2.85s)	18.79% (6.82s)

Conclusion

- Considerable performance improvements can be achieved with heterogeneous computing
- Improvements compared to GPU-only on various systems for large matrices:

		CG	Cholesky
$2^{16} \times 2^{16}$	2x AMD EPYC 9274 + **NVIDIA A30**	12.53% (0.68s)	29.33% (15.99s)
	2x AMD EPYC 9274 + **AMD MI210**	32.85% (2.85s)	18.79% (6.82s)
$2^{15} \times 2^{15}$	Intel i9-10980XE + **Intel Arc B580**	5% (0.14s)	14.25% (3.27s)
	Intel i9-10980XE + **NVIDIA RTX 3080**	0.67% (0.01s)	12.58% (3.07s)

Conclusion

- Considerable performance improvements can be achieved with heterogeneous computing
- Improvements compared to GPU-only on various systems for large matrices:

		CG	Cholesky
$2^{16} \times 2^{16}$	2x AMD EPYC 9274 + **NVIDIA A30**	12.53% (0.68s)	29.33% (15.99s)
	2x AMD EPYC 9274 + **AMD MI210**	32.85% (2.85s)	18.79% (6.82s)
$2^{15} \times 2^{15}$	Intel i9-10980XE + **Intel Arc B580**	5% (0.14s)	14.25% (3.27s)
	Intel i9-10980XE + **NVIDIA RTX 3080**	0.67% (0.01s)	12.58% (3.07s)

At least 12% on all four systems

Outlook

Outlook

- Evaluation of the heterogeneous implementations on new compute architectures
  - AMD MI300A APU
  - NVIDIA Grace Hopper Superchip

  → CPU and GPU are tightly coupled
- Analysis of energy consumption

University of Stuttgart
Institute for Parallel and Distributed Systems –
Scientific Computing

Thank you!

Tim Thüring

Tim.Thuering@ipvs.uni-stuttgart.de

Alexander Strack

Alexander.Strack@ipvs.uni-stuttgart.de

Dirk Pflüger

Dirk.Pflueger@ipvs.uni-stuttgart.de