# Evaluation of Asynchronous Offloading Capabilities of Accelerator Programming Models for Multiple Devices

Jonas Hahnfeld[1], Christian Terboven[1], James Price[2], Hans Joachim Pflug[1], Matthias S. Müller[1]
1: RWTH Aachen University, Germany, email: {hahnfeld,terboven,pflug,mueller}@itc.rwth-aachen.de
2: University of Bristol, UK, email: j.price@bristol.ac.uk

WACCP 2017: Fourth Workshop on Accelerator Programming Using Directives
Nov. 13th, 2017

High Performance Computing
RWTH AACHEN UNIVERSITY

# Motivation & Agenda

## Asynchronous Offloading: are high-level models inferior to low-level APIs?

- CUDA, OpenCL: APIs
  - low-level
  - full control
- OpenMP, OpenACC: based on pragmas
  - ease of use
  - some abstractions

## Agenda of this talk

1. Asynchronous Offloading Capabilities of Accelerator Models
2. Kernel used for evaluation: Conjugate Gradient Method
3. Findings on NVIDIA GPU
4. Findings on Intel Xeon Phi Coprocessor
5. Summary

# Asynchronous Offloading Capabilities of Accelerator Models

# Comparison of CUDA, OpenCL, OpenACC and OpenMP

| | CUDA | OpenCL | OpenACC | OpenMP |
|---|---|---|---|---|
| Asynchronicity | *streams*: actions in different streams can execute concurrently | command queues: operations in different queues can execute concurrently | Clause: `async` with option argument to select a queue; synchronization via `acc wait` construct | Clause: `nowait` because the `target` construct is a task; synchronization via `taskwait` constr. |
| Unstructured data movement | Yes: implicitly | Yes: implicitly | `acc enter/exit data` construct | `target enter/exit data` construct |
| Asynchronous memory transfer | API: `cudaMemcpyAsync` | Argument: `blocking_write` | Clause: `async` | Clause: `nowait` |
| Page-locked memory | API: `cudaMallocHost` | No, but shared virtual memory | - | - |


i12 High Performance Computing RWTH AACHEN UNIVERSITY

# Offloading to multiple devices (1/2)

## Performance Projection without Overlapping

- Total runtime consists of computation and communication:

$$t_{exec} = t_{comp} + t_{comm} \quad (1)$$

- Communication time: Prediction for data d with bandwidth B:

$$t_{comm} = \frac{d}{B} + t_{overhead} \quad (2)$$

- $t_{overhead}$ accounts for preparational tasks
  - may be significant for the overall communication time $t_{comm}$
  - may depend on data volume d

Hahnfeld, Cramer, Klemm, Terboven, Müller: A Pattern for Overlapping Communication and Computation with OpenMP Target Directives. IWOMP 2017.



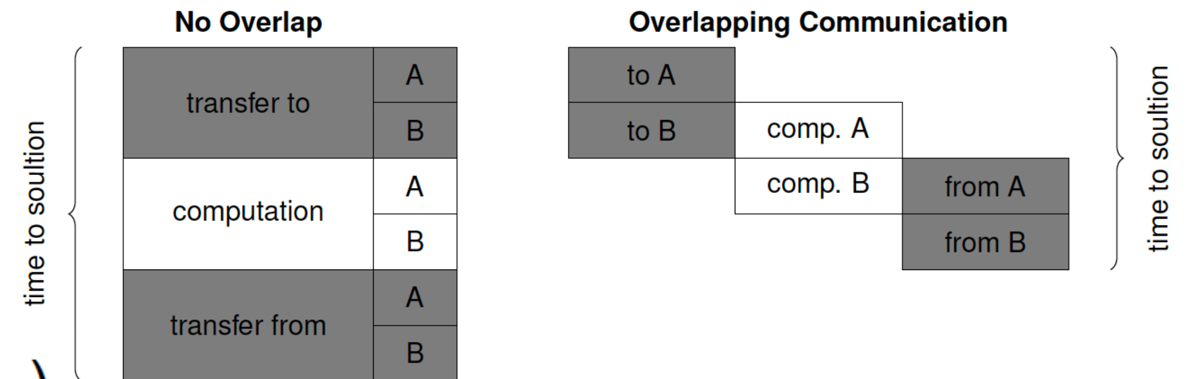
i12 High Performance Computing
RWTH AACHEN UNIVERSITY

# Offloading to multiple devices (2/2)

## Performance Projection with Pipelining Pattern

- Optimal runtime when overlapping computation and communication:

$$t_{pipelined} = max(t_{comp}, t_{comm})$$




- Maximum optimization over runtime without overlapping:

$$o_{max} = \frac{min(t_{comp}, t_{comm})}{t_{exec}} \quad (3)$$

- Approximates a performance increase of $o_{max} = 0.5$
  - if communication and computation time are perfectly balanced

Hahnfeld, Cramer, Klemm, Terboven, Müller: A Pattern for Overlapping Communication and Computation with OpenMP Target Directives. IWOMP 2017.

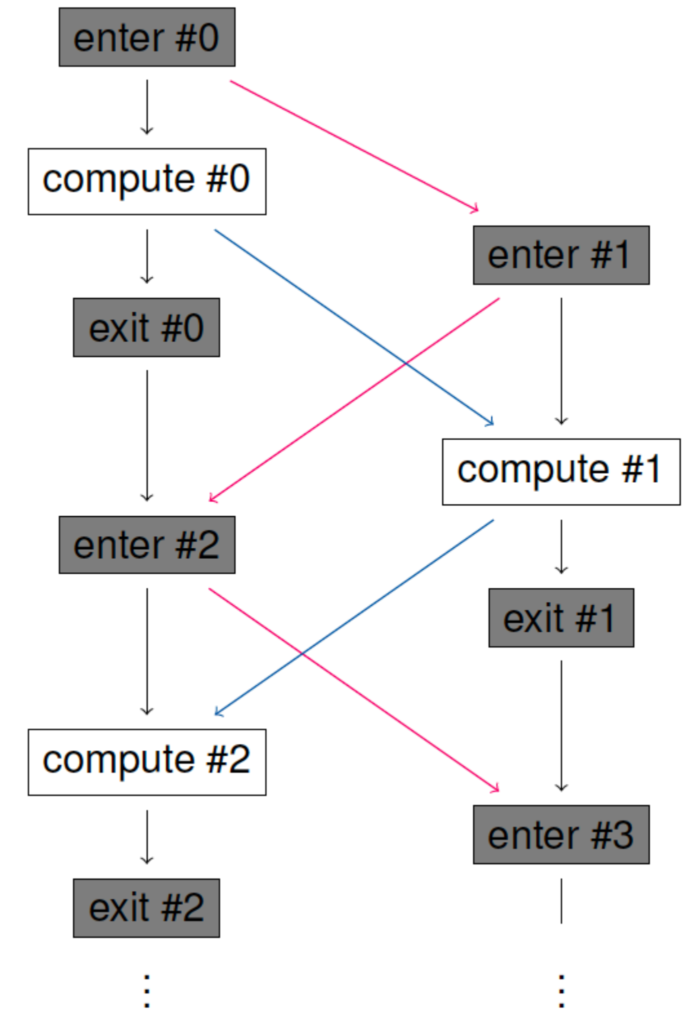


i12 High Performance Computing
RWTH AACHEN UNIVERSITY

# Pipelining Concept for Overlapping Communication

## Implementation with OpenMP 4.5

- Using standalone directives from OpenMP 4.5
  - omp target enter data
  - omp target exit data
- Aynchronous tasks (nowait) and specify dependencies
- Black lines: data dependency
- Red and blue lines: mutual exclusion
  - of enter and compute tasks
  - avoid oversubscription






i12 High Performance Computing
RWTH AACHEN UNIVERSITY

# Kernel used for evaluation: Conjugate Gradient Method

# Evaluation kernel: CG Method (1/3)

## Conjugate Gradients Method on Multiple Devices

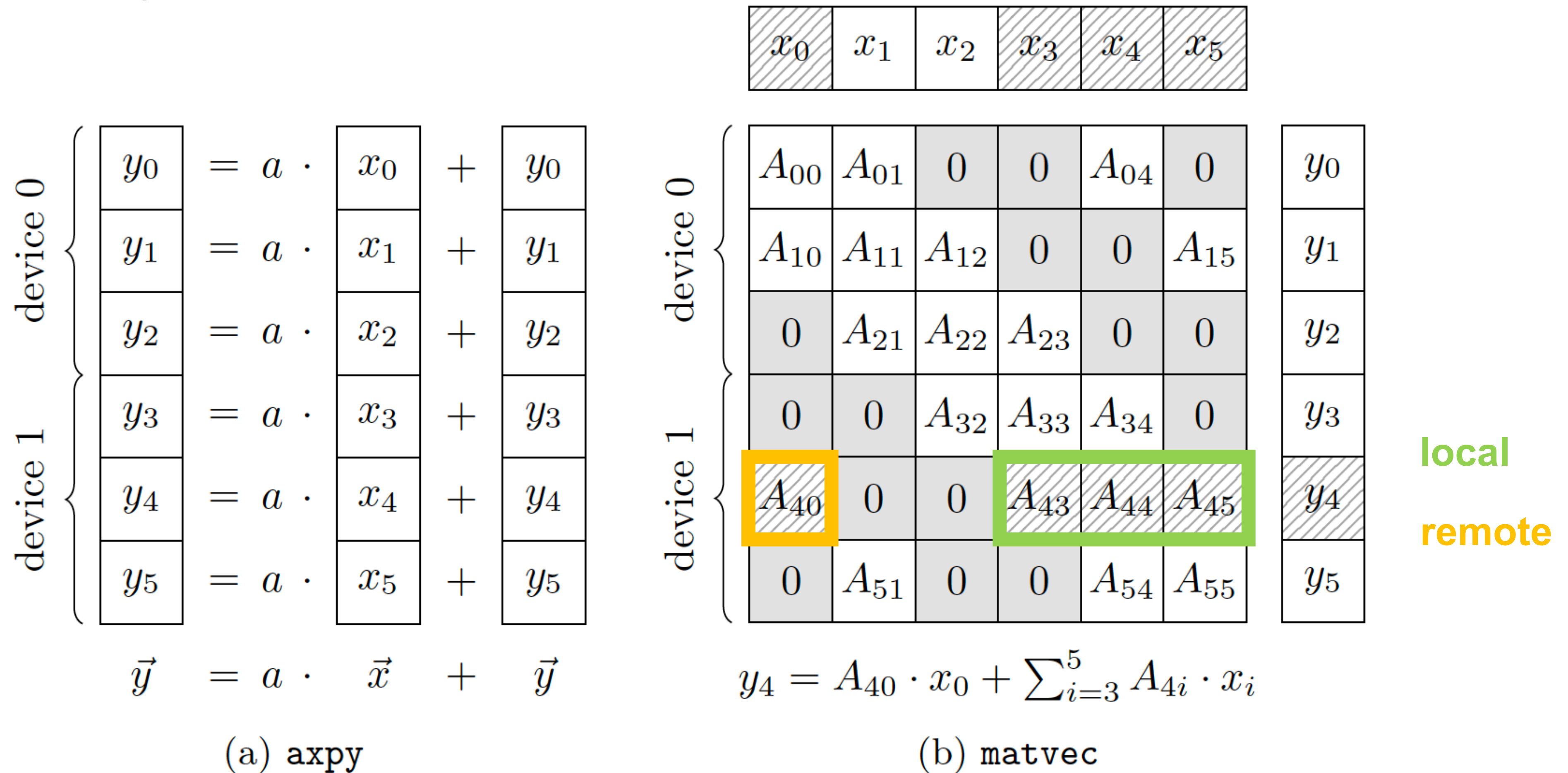
- Iterative solver for linear equation systems:
  - A * x = k
  - widely used for PDE-based problems

- Sparse matrix with regular sparsity pattern: Serena from SuiteSparse Matrix Collection
  - Matrx is spd
  - about 1.4 mio. rows and columns
  - about 780 MB memory consumption in CRS format on host and Xeon Phi
  - about 6.14 GB memory consumption in ELLPACK-R format on GPU

- Division of matrix and each vector into partitions

- Matrix vector multiplication requires data exchange
  - Start computation with local data
  - Apply pipelining concept for data transfer

## Concept for executing kernels on multiple devices

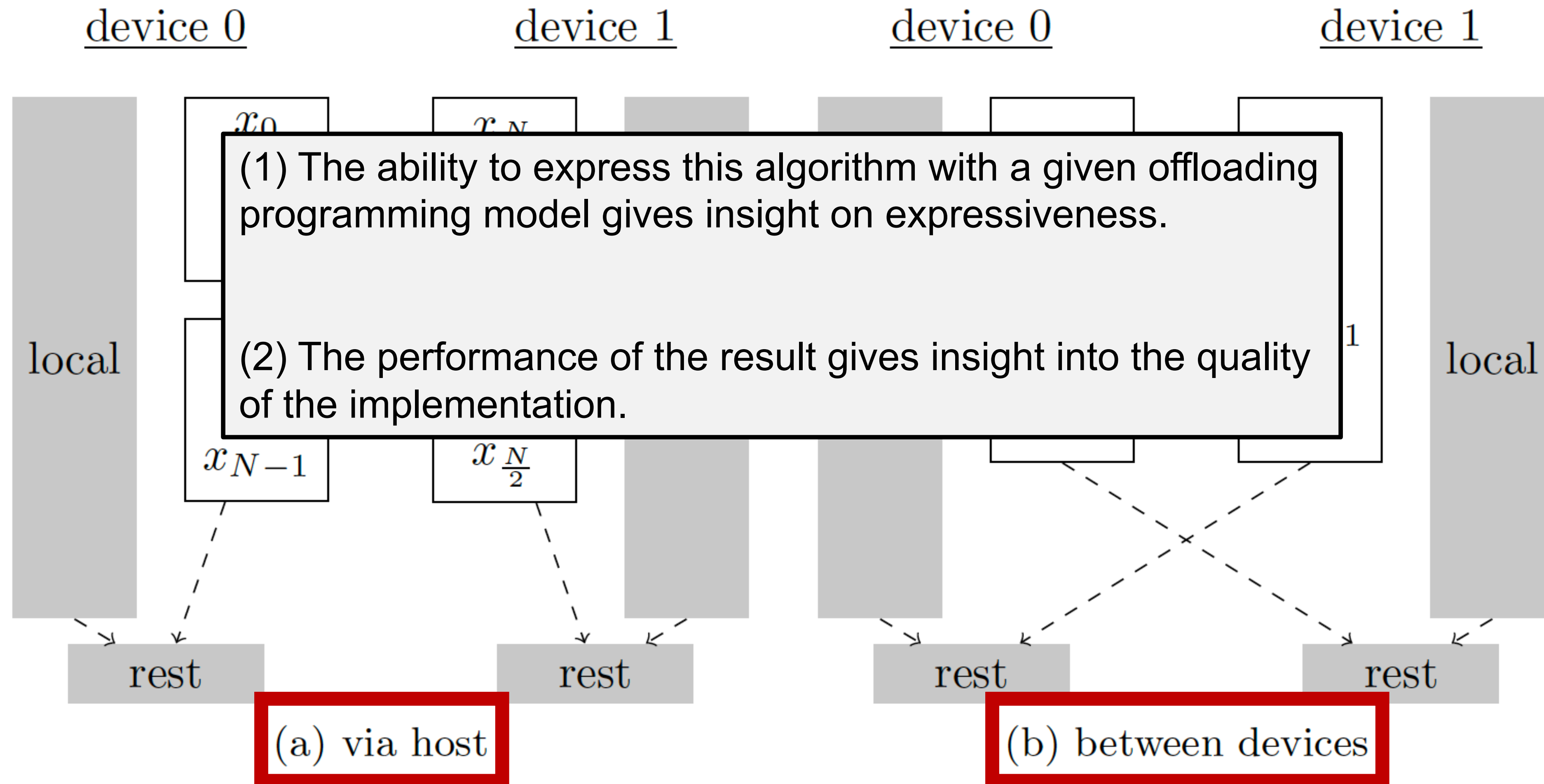




i12 High Performance Computing
RWTH AACHEN UNIVERSITY

## Evaluation kernel: CG Method (3/3)

**Dependencies for overlapping the communication with two devices**




(1) The ability to express this algorithm with a given offloading programming model gives insight on expressiveness.

(2) The performance of the result gives insight into the quality of the implementation.

# Findings on NVIDIA GPU

# Evaluation System

## Technical Specification

- NEC GPU server system
  - 2 NVIDIA Tesla P100, each:
    - 5.3 TFLOP/s dp performance
    - 549 GB/s Triad bandwidth to HBM2 measured with BabelStream
    - 13.2 GB/s achievable transfer rate to host via PCIe
  - NVLink between the GPUs
    - 37 GB/s achievable transfer rate between GPUs
  - 2 Intel Westmere-EP 12-core processors at 2.2 GHz
    - 120 GB/s Triad bandwidth to DDR4 measured with Stream

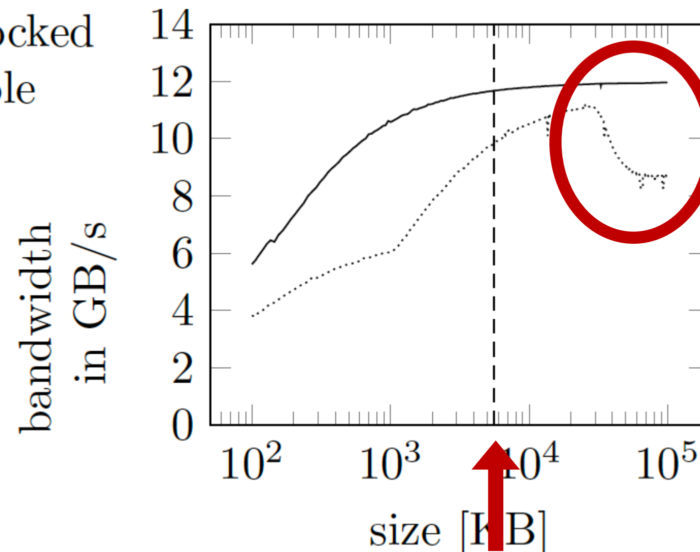
| Programming model | Software stack |
| --- | --- |
| CUDA | GCC 4.8.5 + CUDA 8.0.44 |
| OpenCL | GCC 4.8.5 + CUDA 8.0.44 |
| *pocl* | LLVM 4.0.1 + development version of *pocl* |
| OpenACC | PGI Accelerator Compiler 17.4 |


i12 High Performance Computing
RWTH AACHEN UNIVERSITY

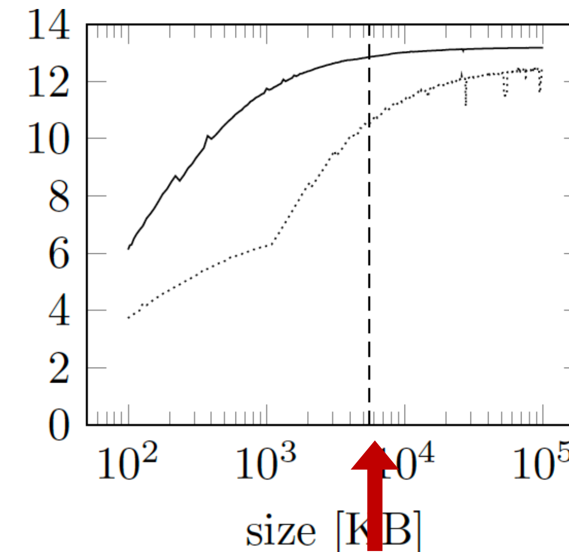
# Data Transfer with Host

## Basis for performance model

- DMA: Direct Memory Access
  - Only possible with memory that is ”page-locked” or “pinned”
    - Necessity for device data transfers
    - By default CUDA (et al.) do a transparent copy
    - In addition, special allocation methods are available



(a) to the device



(b) from the device

- Size of x vector chunks: 5.6 MB
- No explanation for the drop after 32 MB
- CUDA and OpenACC on par pocl about 10% below



i12 High Performance Computing
RWTH AACHEN UNIVERSITY

# Results with a single GPU

## Basis for evaluation

- Initial evaluation on the host and on a single NVIDIA Tesla P100
  - Host: Intel C++ 17.0.4 compiler w/ OpenMP

| | `matvec` (GFLOP/s) | Vector dot products | Iterations | Total Runtime |
|---|---|---|---|---|
| host | 7.11s (17.91) | 0.24s | 985 | 9.17s |
| CUDA | 1.89s (67.44) | 0.20s | 987 | 5.15s |
| OpenCL | 2.23s (57.23) | 0.31s | 986 | 5.72s |
| *pocl* | 2.27s (56.24) | 0.32s | 986 | 5.78s |
| OpenACC | 2.23s (57.22) | 0.29s | 989 | 5.69s |

- Offloading to the GPU pays off (timings include data transfer)

- Number of iterations varies because of reduction operating in vector dot product

- CUDA is fastest
  - Compiler generates better code with added `pragma unroll 1`



i12 High Performance Computing
RWTH AACHEN UNIVERSITY

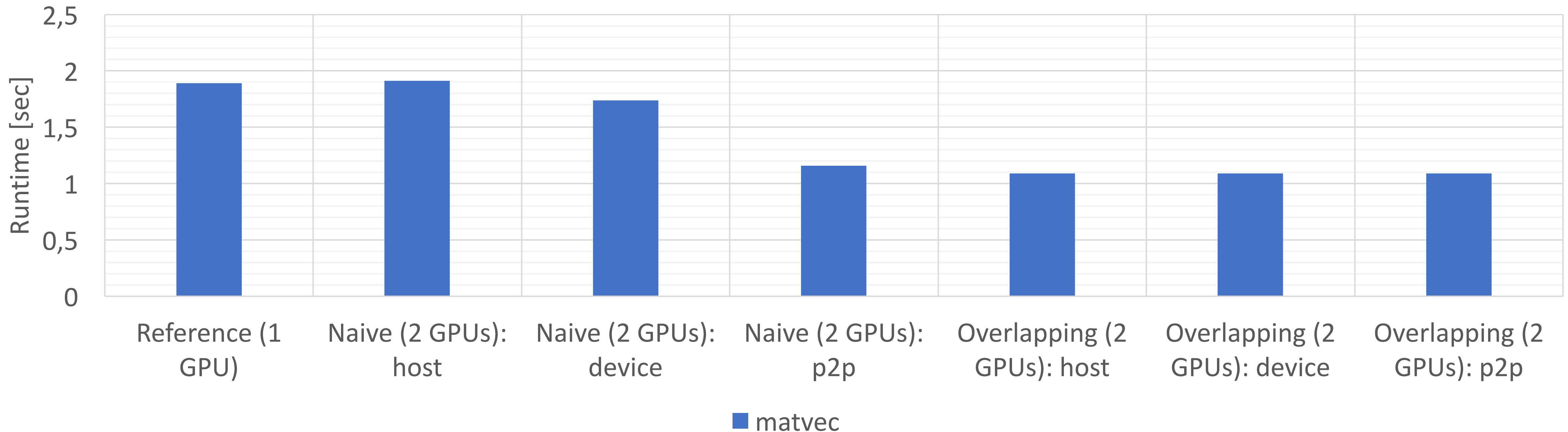
## Options to implement data exchange between devices

- Via host: utilization of PCIe in two successive transfers
  - Plus a temporary buffer on the host
- Between devices: direct communication between devices
  - `cudaMemcpyAsync` with kind `cudaMemcpyDeviceToDevice`
  - Will use PCIe by default
  - Runtime may employ double buffering, or other optimizations
- Peer to peer: utilization of the NVLink
  - `cudaDeviceEnablePeerAccess` on both devices for unidirectional access



i12 High Performance Computing
RWTH AACHEN UNIVERSITY

# CUDA Results (matvec only)




- Performance model prediction: up to 46.91 % optimization
  - Requires two CUDA streams per device: computation & communication

- Remember (3): optimization effect depends on max of $t_{comp}$ and $t_{comm}$
  - Utilizing NVLink reduces the communication time

- Smaller improvement for whole CG as partitioning takes extra time: 5.15s to 4.26s in best case

# OpenCL

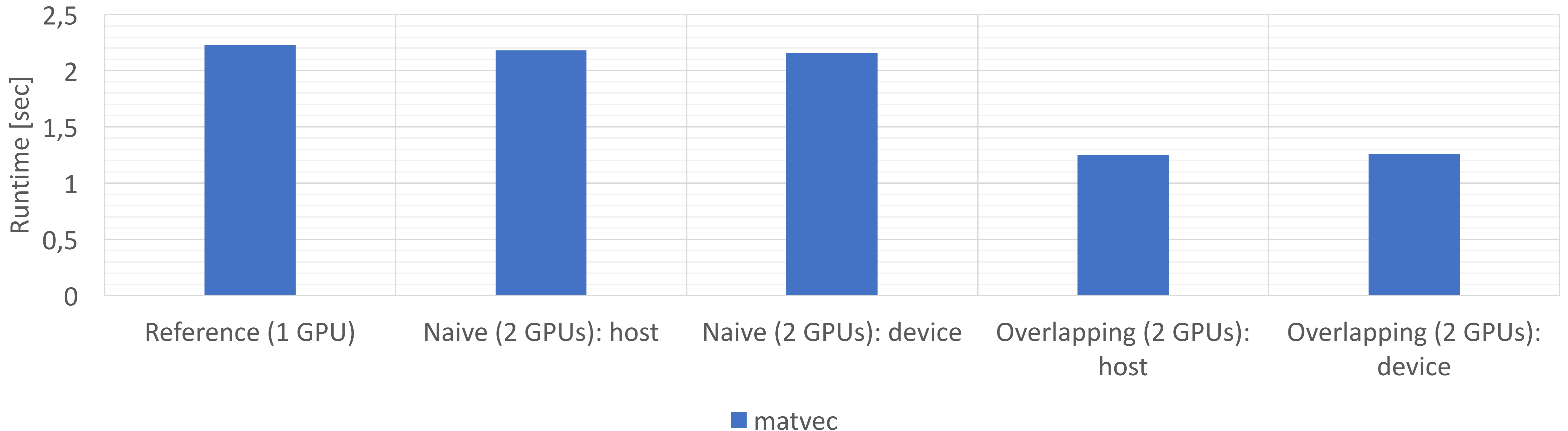
## Quality of the implementation

- NVIDIA‘s own OpenCL …
  - ... does not support page-locked memory
  - ... has a performance bug in the device to device copy

- Therefore we switched to pocl (OpenCL 2.0 implementation)
  - ... and made some improvements which will become available with next release


i12 High Performance Computing
RWTH AACHEN UNIVERSITY

# OpenCL Results (matvec only)

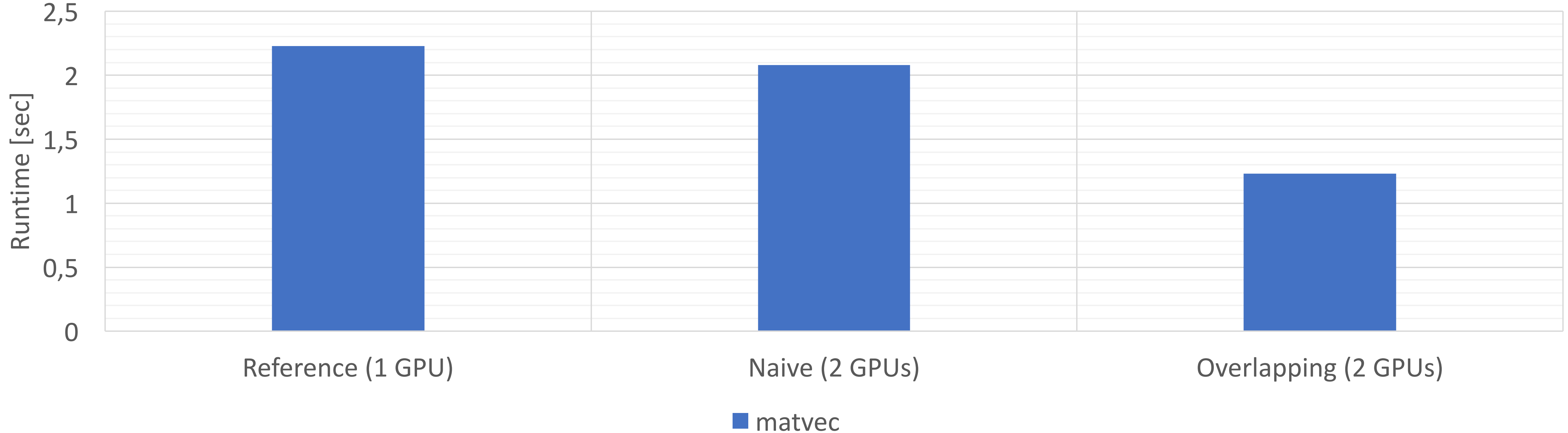



- Performance model prediction: up to 44.34 % optimization
  - Requires two OpenCL command queues per device: computation & communication



i12 High Performance Computing
RWTH AACHEN UNIVERSITY

# OpenACC Results (matvec only) (1/2)

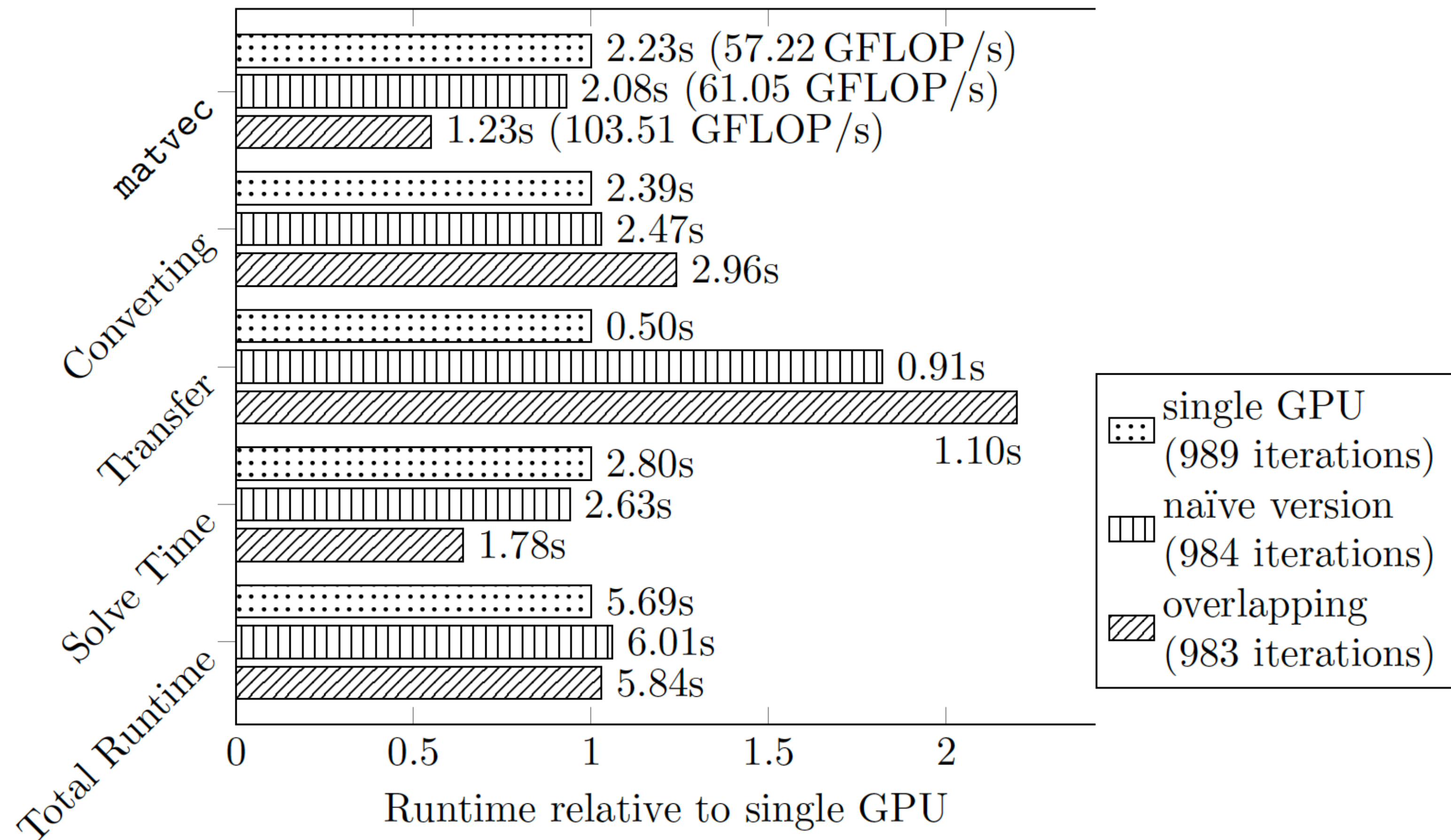



- Performance model prediction: up to 44.05 % optimization
- OpenACC currrently does not allow transfer between two devices without involving the host
- Data transfer time increases
  - PGI‘s implementation cannot transfer matrix from pageable memory asynchronously
  - Issues with the runtime prevented from using threads to parallelize data transfer


i12 High Performance Computing
RWTH AACHEN UNIVERSITY

# OpenACC Results (2/2)

## Not successful considering the total runtime






i12 High Performance Computing
RWTH AACHEN UNIVERSITY

# Findings on the Intel Xeon Phi Coprocessor

# Evaluation System

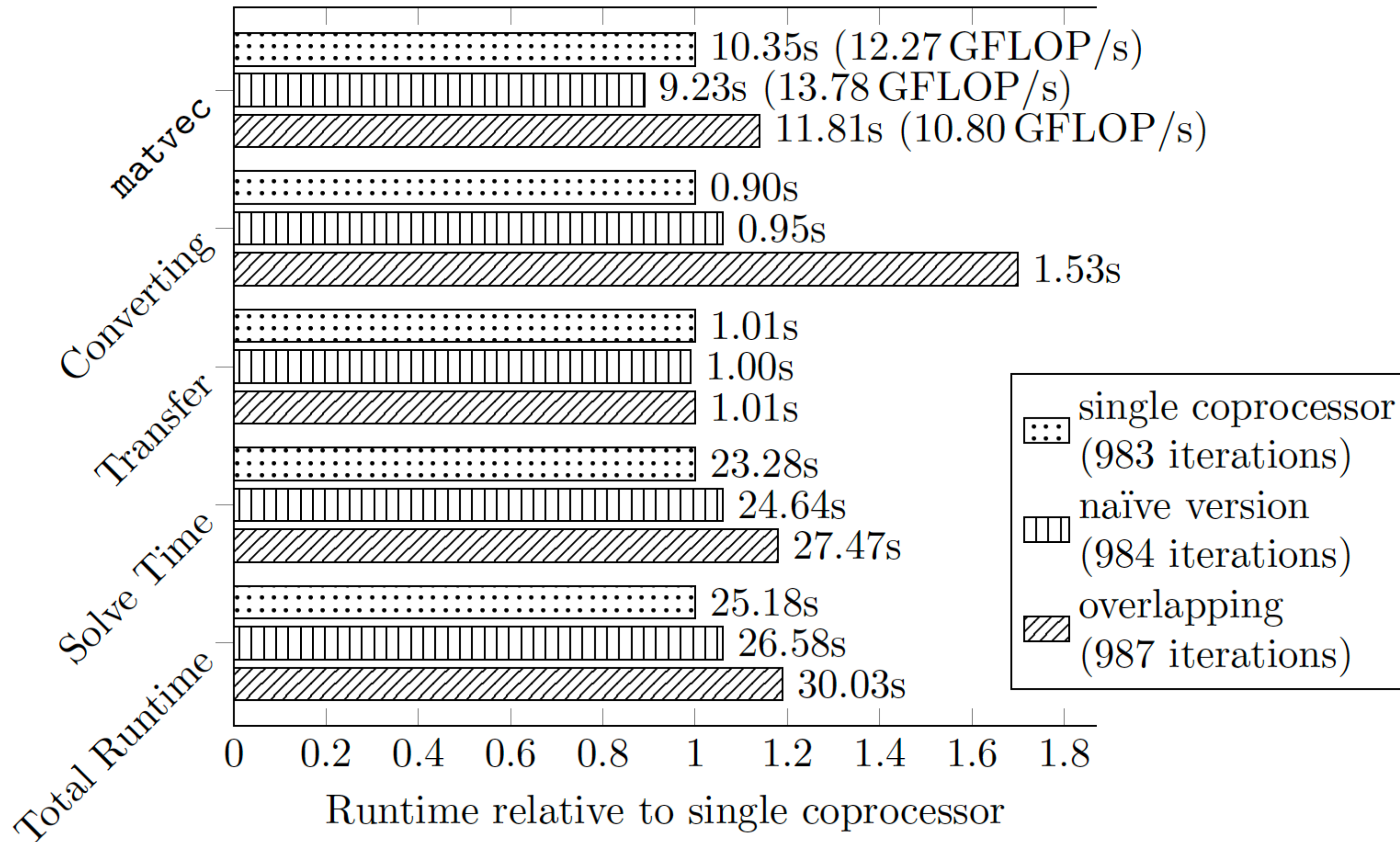
## Technical Specification

- Bull server system
  - 2 Intel Xeon Phi 5110P coprocessors, each:
    - approx. 1 TFLOP/s dp performance
    - 117 GB/s Triad bandwidth to HBM measured with Stream
    - 6.5 GB/s achievable transfer rate to host via PCIe (gen2)
  - 2 Intel SandyBridge-EP 8-core processors at 2.0 GHz
    - 65 GB/s Triad bandwidth to DDR4 measured with Stream

- Software
  - Intel 17.0.2 compilers
    - 17.0.4 contains a performance bug
  - Intel MPSS 3.8
  - Intel OpenCL SDK 14.2

# OpenMP Results

## Not successful considering the total runtime




- Very high overhead for launching the kernels
- Intel 16.0.x compilers show better performance …
- ... but do not provide asynchronous offloading

# Summary

# Summary

## Evaluation of asynchronous offloading

- Asynchronous offloading to multiple devices can deliver the expected performance improvements
- CUDA is perfectly up to the task
- OpenCL 2.0 provides all the necessary ingredients
- OpenACC can be successful
  - Currently, device to device support is missing
  - Issues with the quality of the implementation
- Intel Xeon Phi
  - Implementation issues lead to bad results for OpenMP
  - GCC 7, IBM xlc and LLVM/Clang compilers will soon fully support OpenMP on GPUs
- Programming is challenging: asynchronous offload pattern may ease implementation work
- Code is available at: https://rwth-aachen.sciebo.de/index.php/s/EdjjkEdClHLizyE


i12 High Performance Computing
RWTH AACHEN UNIVERSITY

**Vielen Dank**
**für Ihre Aufmerksamkeit**

**Thank you**
**for your attention**