Evaluation of two topology-aware heuristics on level-3 BLAS library for multi-GPU platforms

Thierry Gautier (INRIA, CNRS, ENS Lyon, UCBL, France)
João Vicente Ferreira Lima (UFSM, Brazil)

Introduction

- Dense linear algebra are fundamental to scientific applications
- BLAS library makes easy the development of HPC applications
  - A standard building block in HPC
  - Many implementations: Intel MKL, AMD ACML, OpenBLAS, ATLAS, ...
  - It ensures performance portability
- BLAS on multi-GPUs imposes a tradeoff
  - Heavy code refactoring
  - Overhead from drop-in replacement libraries
- Previous work: XKBlas library [PDP 2020]
  - Drop-in library with LAPACK data layout, PLASMA algorithms
  - Asynchronous BLAS calls
  - On top of XKaapi multi-GPUs runtime engine [IPDPS 2013, Parco 2015]

Background - DGX-1

- Target architecture = Highly Density Computer with (a lot of) GPUs
  - Between 6 to 16 GPUs per node
  - DGX-1, DGX-A100, DGX-2, … Summit node, Frontier / El Capitan node
- Main issues - Links with heterogeneous performance
- Example: NVIDIA DGX-1
  - CPU <-> GPU: PCIe GEN3
  - GPU <-> GPU: NVLink high-bandwidth
    - 2 Levels of performance
    - Single NVLink
    - Aggregated 2 NVLinks
  - PCIe on others

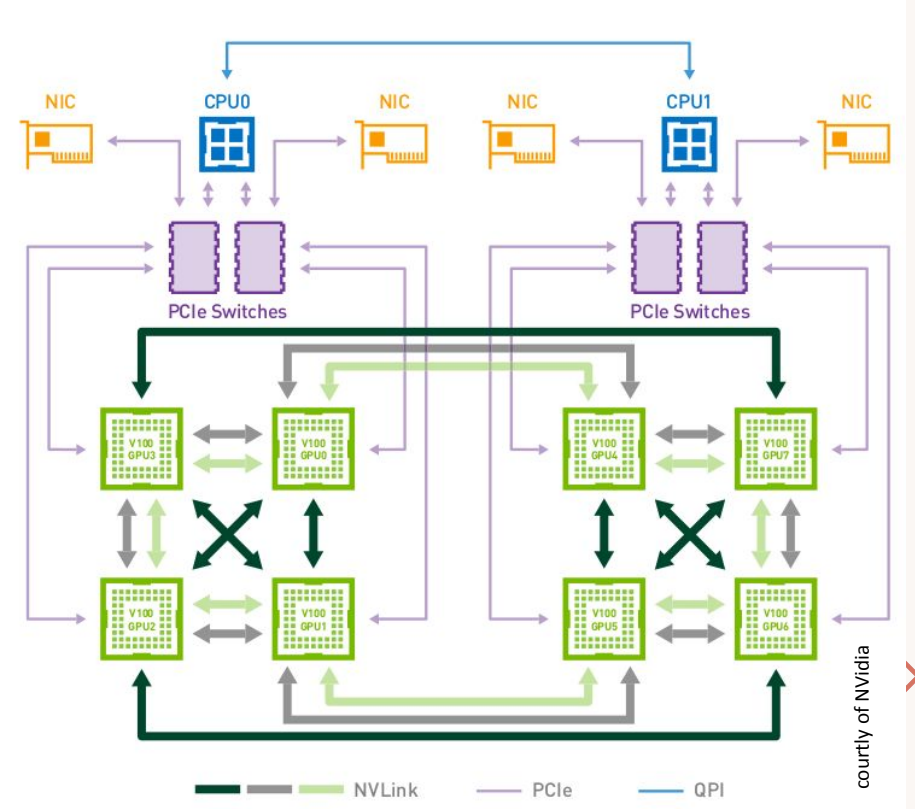


Illustration: DGX-1

D\D	0	1	2	3	4	5	6	7
0	744.05	48.37	48.39	96.49	96.45	17.11	17.74	17.97
1	48.38	750.48	96.50	48.38	16.98	96.44	17.32	16.97
2	48.34	96.28	750.48	96.47	17.62	16.93	48.39	17.75
3	96.26	48.34	96.28	750.48	17.58	17.22	17.60	48.39
4	96.46	16.98	17.65	17.53	746.89	48.39	48.40	96.49
5	16.94	96.42	16.88	17.21	48.39	745.47	96.51	48.40
6	17.65	16.90	48.40	17.51	48.34	96.47	750.48	96.47
7	17.80	16.91	17.77	48.39	96.28	48.38	96.28	747.61

xx : local GPU
xx : 2 NVLinks
xx : 1 NVLink
xx : PCIe

Peer-to-Peer GBytes/s transfer ratio

Our contribution

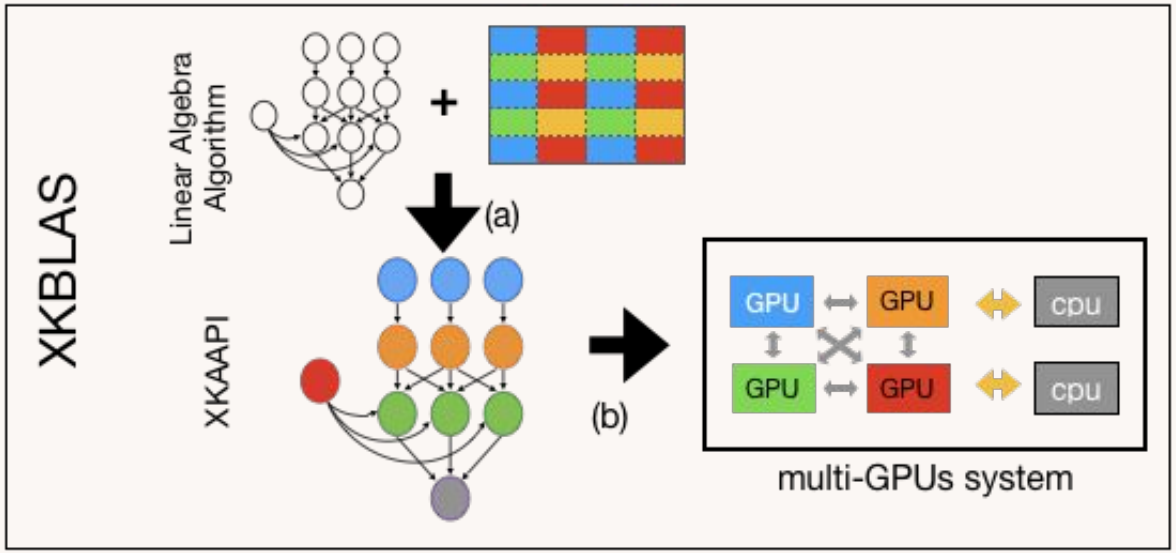
- Two topology-aware heuristics in order to
  - An **optimistic heuristic** to reduce communications between CPUs and GPUs
    - by optimistically wait and already in-transfers data to a GPUs
  - A **topology-aware heuristic** to favor high performance links in the topology

- Evaluation on level-3 BLAS kernels
  - XKBlas

- Comparison against state of the art BLAS libraries
  - CUBLAS-XT, Slate, DPLASMA/Parsec, Chameleon/StarPU, BLASX
  - Not all libraries provides the 9 BLAS kernels

XKBlas Overview

XKBlas Overview



- **XKBlas** has a set of task-based BLAS-3 tile algorithms with data dependencies
  - Tile algorithms from PLASMA/Chameleon
- **XKaapi** runtime unrolls a data flow graph (DFG) and schedules tasks with data dependencies over GPUs
  - Extended by the proposed heuristics

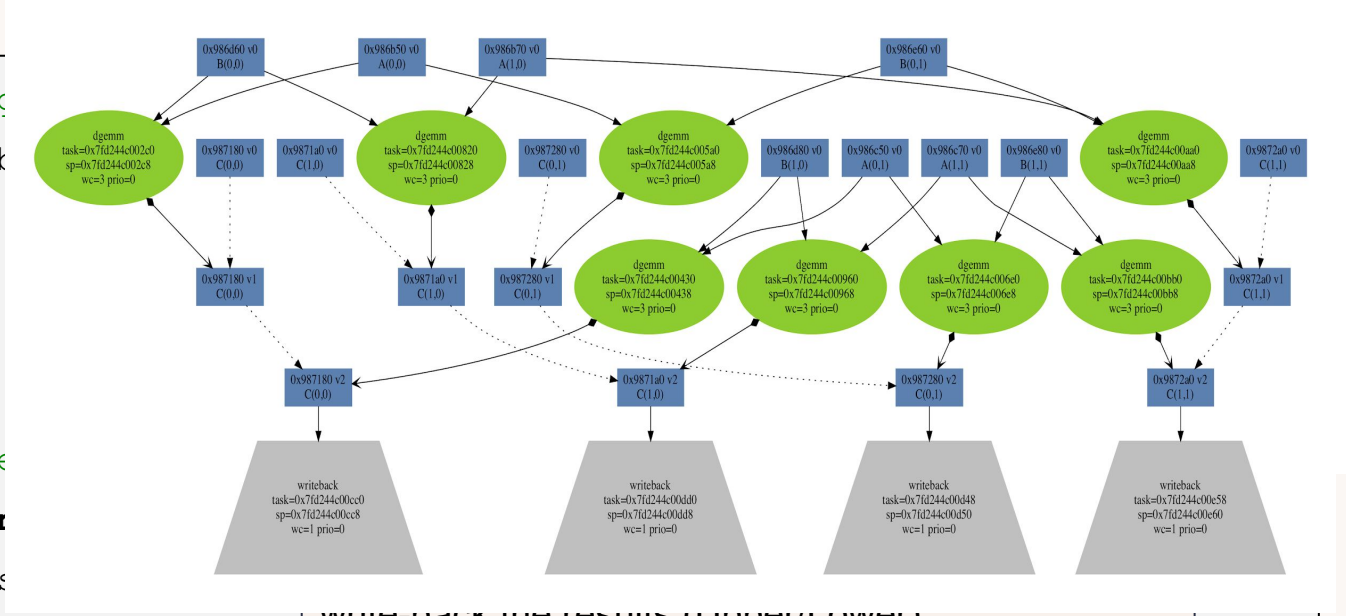
API design

```
// (1) no blocking dgemm call
xkblas_dgemm_async(CblasNoTrans, CblasNoTrans, m, n, k,
    &alpha, A, lda,
            B, ldb,
    &beta,  C, ldc);

// (2) no blocking request to make coherent full C matrix
xkblas_memory_coherent_async(
    CblasLower | CblasUpper,
    m, n, C, ldc, sizeof(double)
 );
 // (3) wait completion of previo
xkblas_sync();
```

xkblas_memory_coherent_async is mandatory to write-back the results (Upper/Lower).

xkblas_sync waits for all asynchronous operations.

```
// (1) no blocking dg
xkblas_dgemm_async(Cb
    &alpha, A, lda,
            B, ldb,
    &beta,  C, ldc);

// (2) no blocking re
xkblas_memory_coheren
    CblasLower | Cblas
    m, n, C, ldc, sizeof(double)
 );
 // (3) wait completion of previo
xkblas_sync();
```

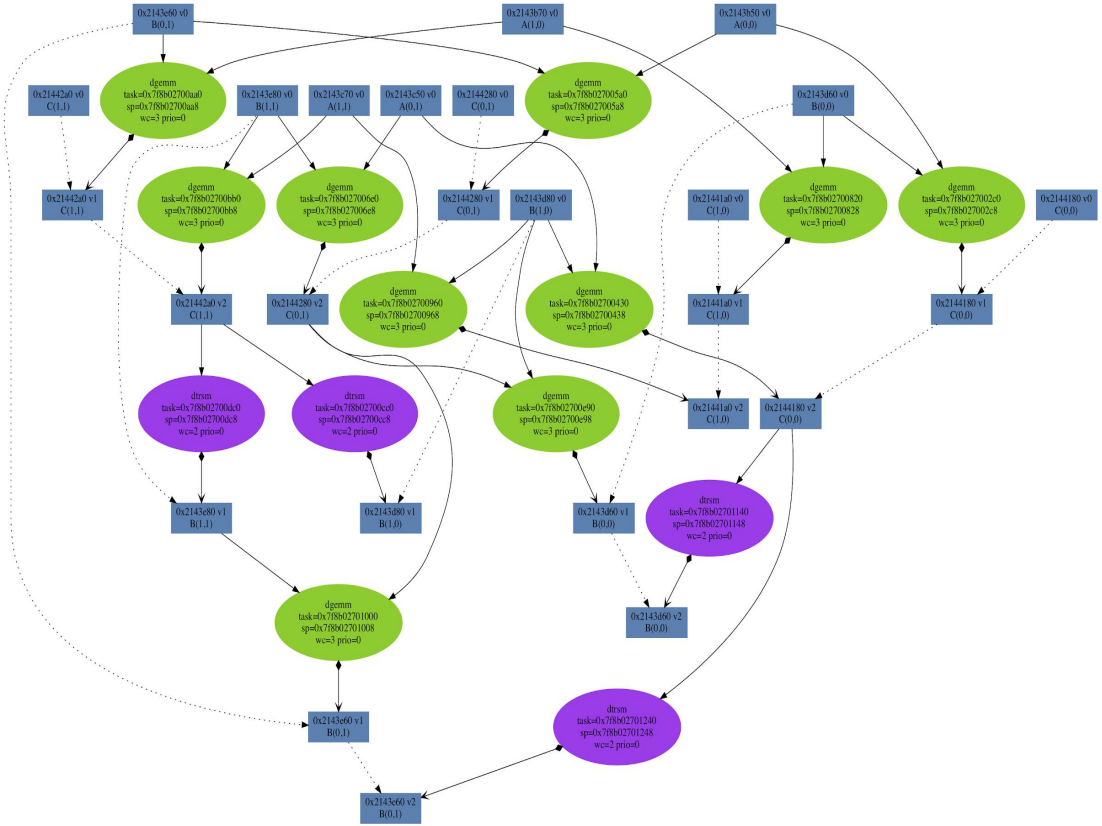


write-back the results (Upper/Lower).

xkblas_sync waits for all asynchronous operations.

Composing GEMM + TRSM

```
// (1) no blocking dgemm call
xkblas_dgemm_async(CblasNoTrans, CblasNoTrans, n, n, n, &alpha, A, lda,
            B, ldb, &beta,  C, ldc);

// (2) no blocking dtrsm call
xkblas_dtrsm_async( CblasLeft, CblasUpper, CblasNoTrans, CblasUnit,
    n, n, &alpha, C, n, B, n);

// (3) no blocking request to make coherent full C matrix
xkblas_memory_coherent_async( CblasLower | CblasUpper,
   m, n, C, ldc, sizeof(double) );

// (4) wait completion of previous asynchronous operations
xkblas_sync();
```



Composition: point to point dependencies between tasks

RSM

```
n, n, &alpha, A, lda,

NoTrans, CblasUnit,

C matrix
Upper,

operations
```

Topology-Aware Heuristics

Topology-Aware Heuristics

- XKBlas has two topology-aware heuristics for hierarchical multi-GPU systems
- Both give priority to transfers from GPUs rather than Host main memory

- **Optimistic heuristic for device-to-device data transfer**
  - Wait an in-progress transfer to GPU(s) before request data from Host

- **Topology-aware device-to-device data transfer**
  - Transfers begins from a GPU with the fastest performance link (if available)

Optimistic heuristic for device-to-device transfer

- The heuristic is called “*optimistic*” because it hopefully delay the transfer to receive data faster!

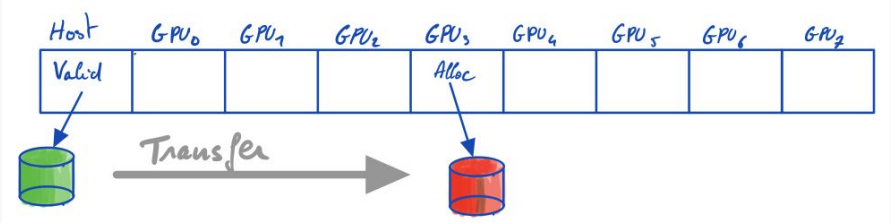
- Very interesting when all the data are initially located on the Host
  - Prevent transfers over PCIe , slower than NVLink

- Extension of XKaapi runtime
  - Adding the state Xfer (In transfer) attached to copies managed by the DSM
  - Adding the capacity to add callback on some internal event
    - The event is “data is received” which decrements the counter on the task waiting for its input data

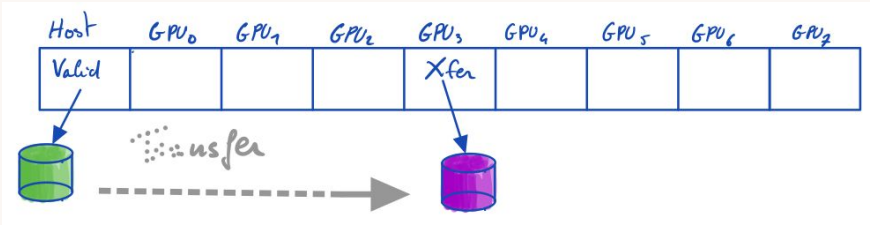
Optimistic heuristic for device-to-device transfer

- Assume GPU_3 needs a data on the host (that stores a valid copy)

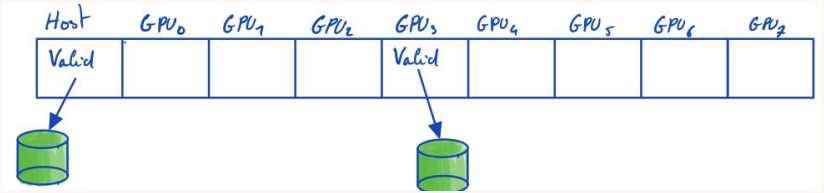
(1) Sender: the Host
Receiver: GPU_3



(2) copy on GPU_3 is in-transfer

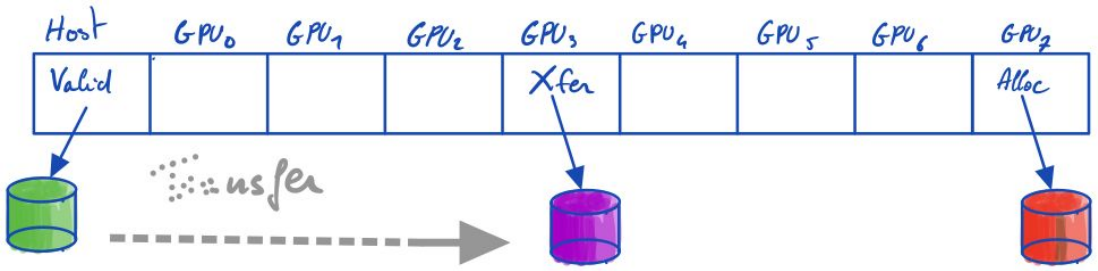


(3) in-transfer is done, the copy is marked as "valid"



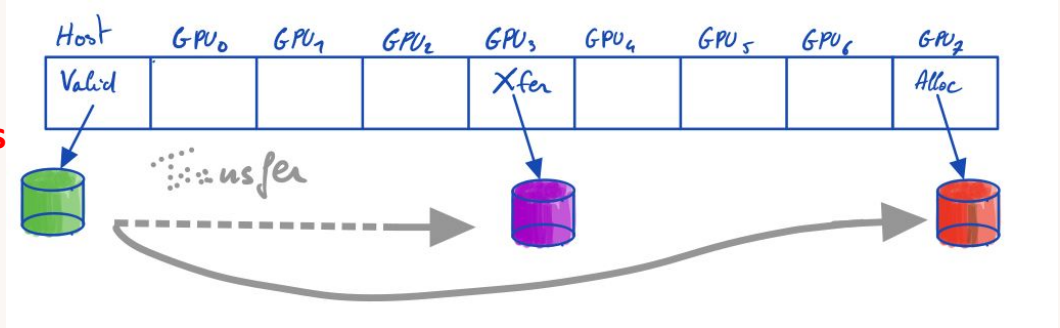
Optimistic heuristic for device-to-device transfer

- Now assume GPU_7 requires the data



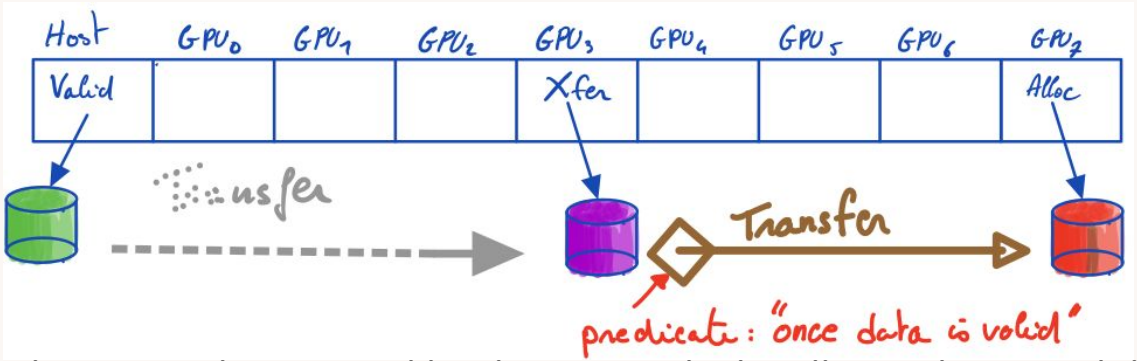
- Without our heuristic, data is send by the Host holding the only valid copy

2x host transfers

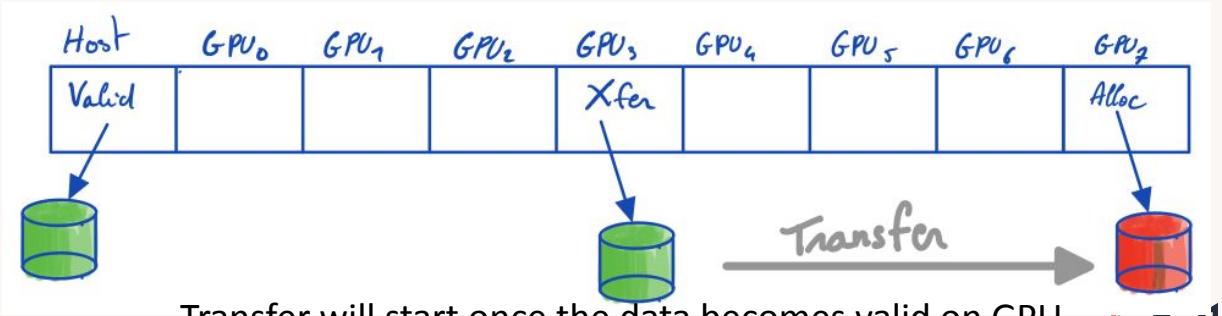


Optimistic heuristic for device-to-device transfer

- Now assume GPU_7 requires the data

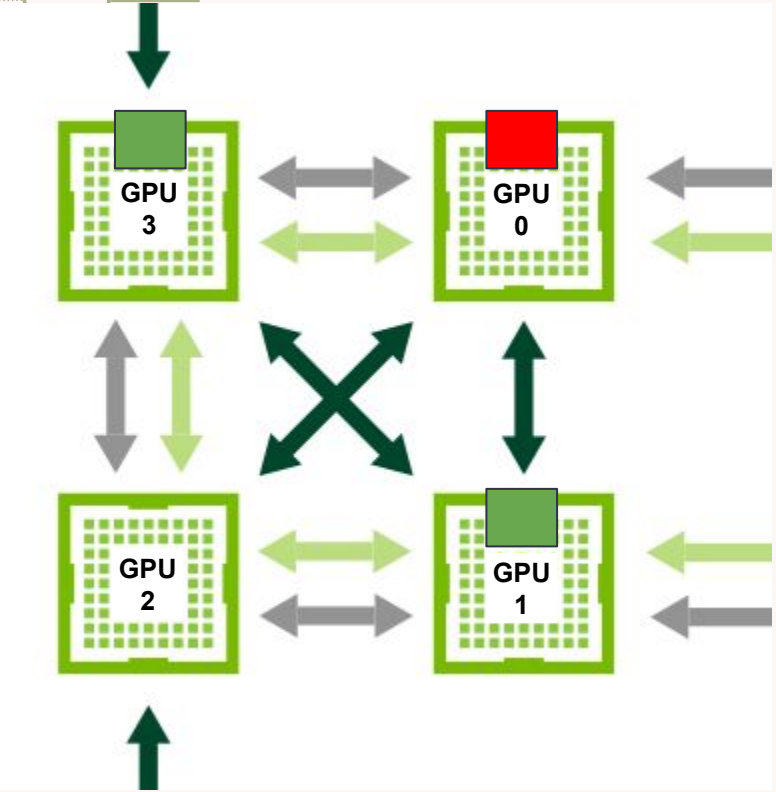


- With our heuristic, data is send by the GPU_3 which will soon host a valid copy



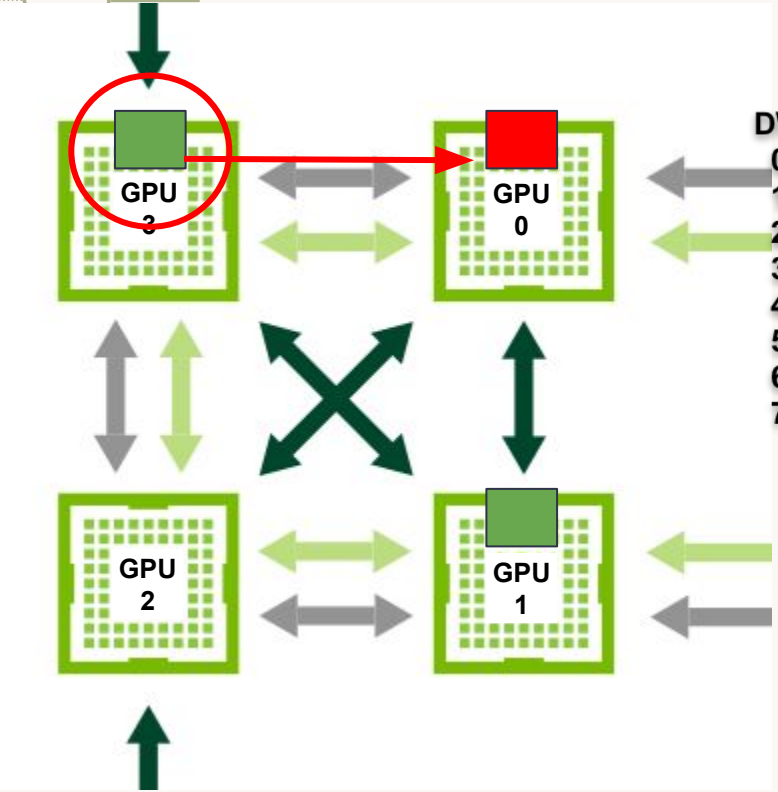
Transfer will start once the data becomes valid on GPU_3

Topology-aware device-to-device transfer



- Favor high speed links
  - if several valid copies exist, then always select the link with the fastest performance
- GPU-0 needs a valid copy
  - GPU-1 and GPU-3 have valid copies
    - GPU-1 has 1 NVlink to GPU-3 (48 GB/s)
    - GPU-3 has 2 NVlinks to GPU-3 (96 GB/s)
  - Host transfer would require PCIe (17GB/s)
  - The runtime will transfer from GPU-3

Topology-aware device-to-device transfer



D\D	0	1	2	3	4	5	6	7
0	744.05	48.37	48.39	96.49	96.45	17.11	17.74	17.97
1	48.38	750.48	96.50	48.38	16.98	96.44	17.32	16.97
2	48.34	96.28	750.48	96.47	17.62	16.93	48.39	17.75
3	96.26	48.34	96.28	750.48	17.58	17.22	17.60	48.39
4	96.46	16.98	17.65	17.53	746.89	48.39	48.40	96.49
5	16.94	96.42	16.88	17.21	48.39	745.47	96.51	48.40
6	17.65	16.90	48.40	17.51	48.34	96.47	750.48	96.47
7	17.80	16.91	17.77	48.39	96.28	48.38	96.28	747.61

xx : local GPU
xx : 2 NVLinks
xx : 1 NVLink
xx : PCIe

Experimental Results

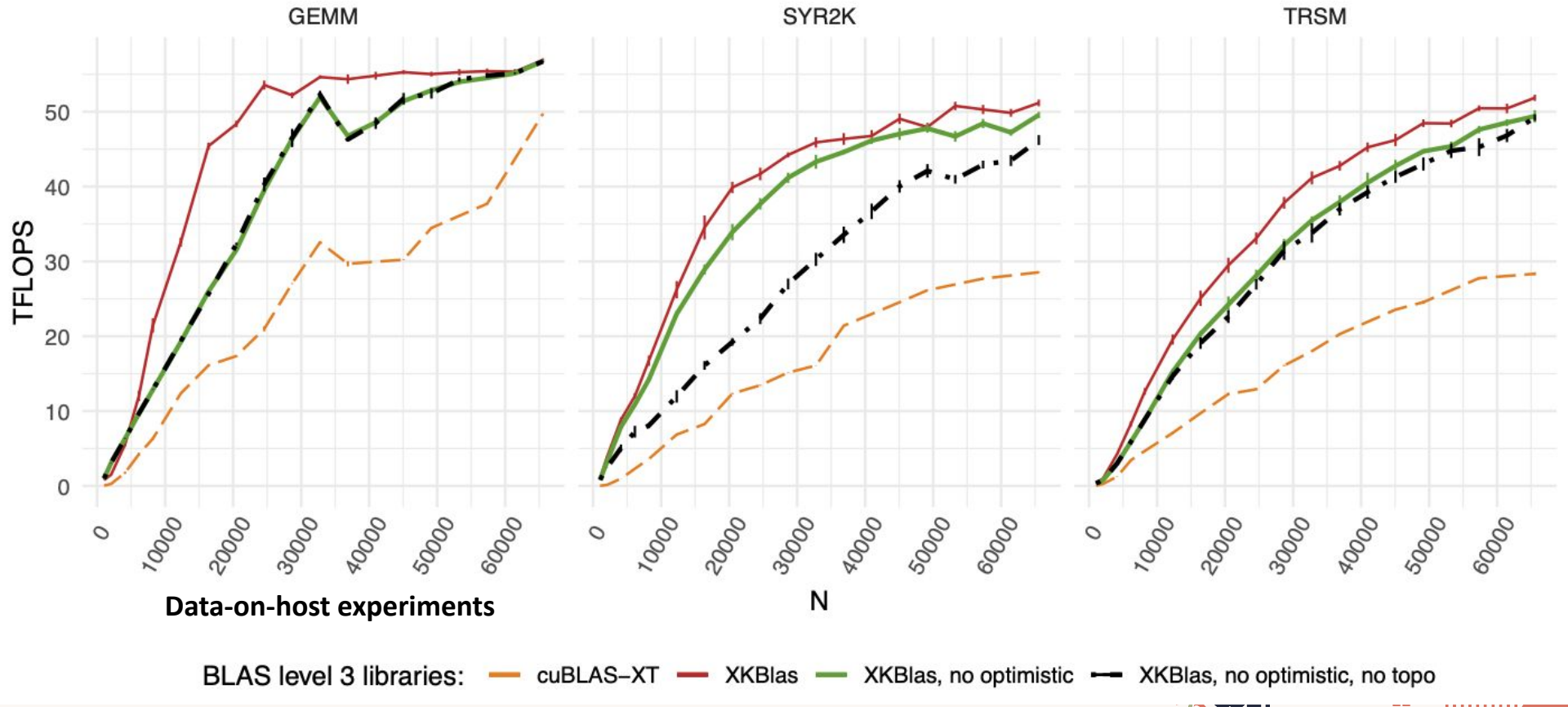
Experimental results

- Impact of XKBlas topology-aware heuristics on performance

- **Data-on-host** - base case
  - Operands and results are mainly on the host memory

- **Data-on-device** - only device transfers
  - Matrix tiles are initially on GPUs by a 2D-block cyclic distribution

Evaluation of the XKBlas-heuristics



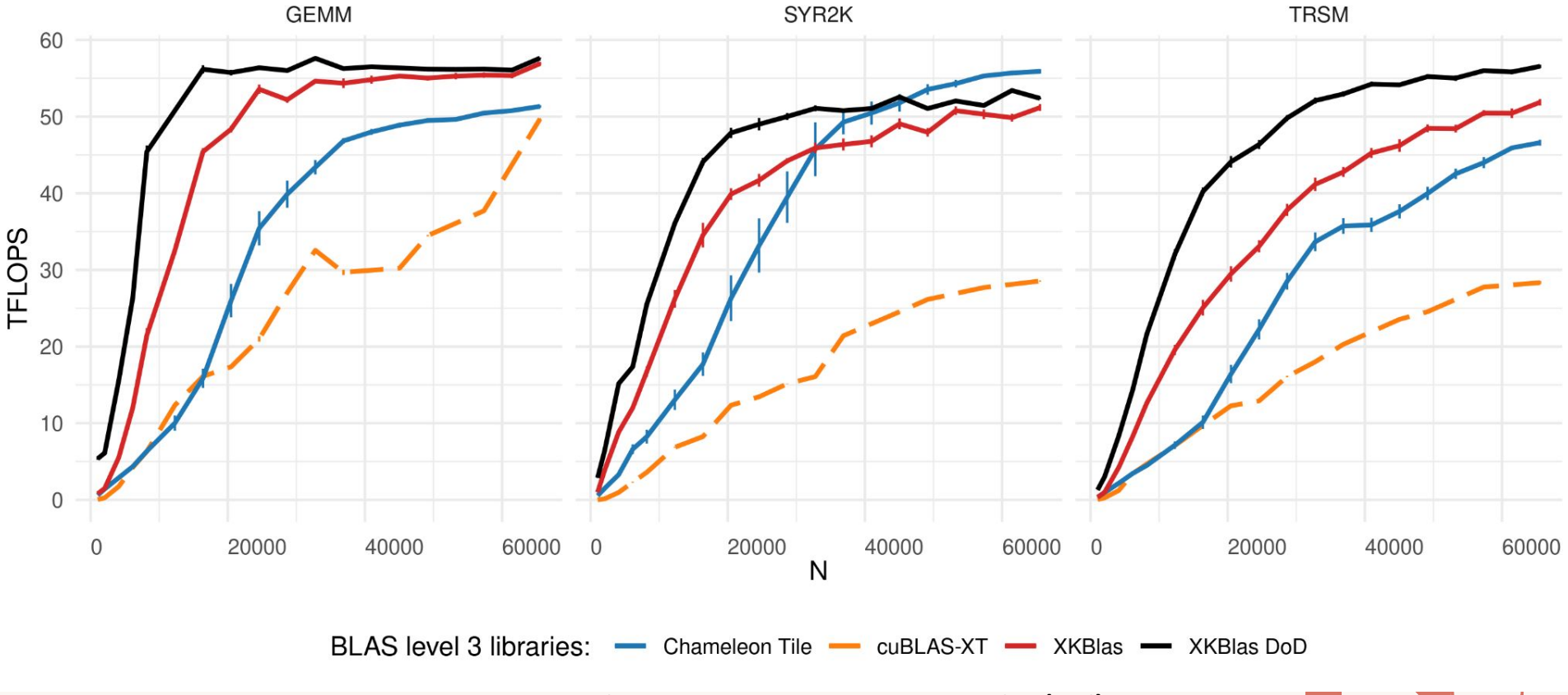
XBlas Topology Heuristics - Performance Overview

Loss/gain of performance over XKBlas disabling topology-aware features (> 30K dimension).

Kernel	No optimistic heuristic	No heuristic/topology
GEMM	-13.96%	-14.81%
SYR2K	-8.02%	-34.05%
TRSM	-13.74%	-17.88%

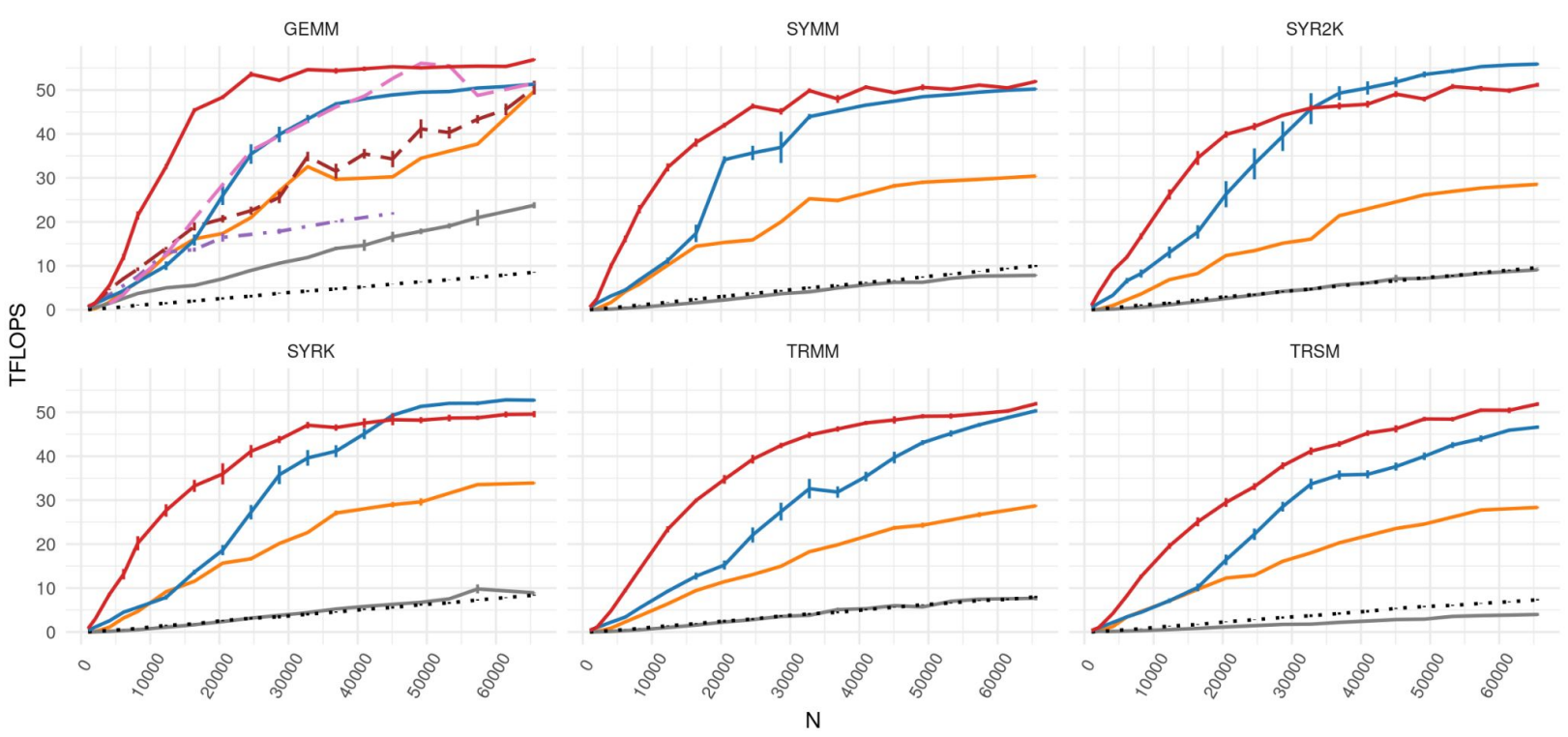
- GEMM benefits mostly from the optimistic heuristic
- SYR2K had significant gains with the topology heuristic
- TRSM gained with both heuristics

Data on device - DGX1



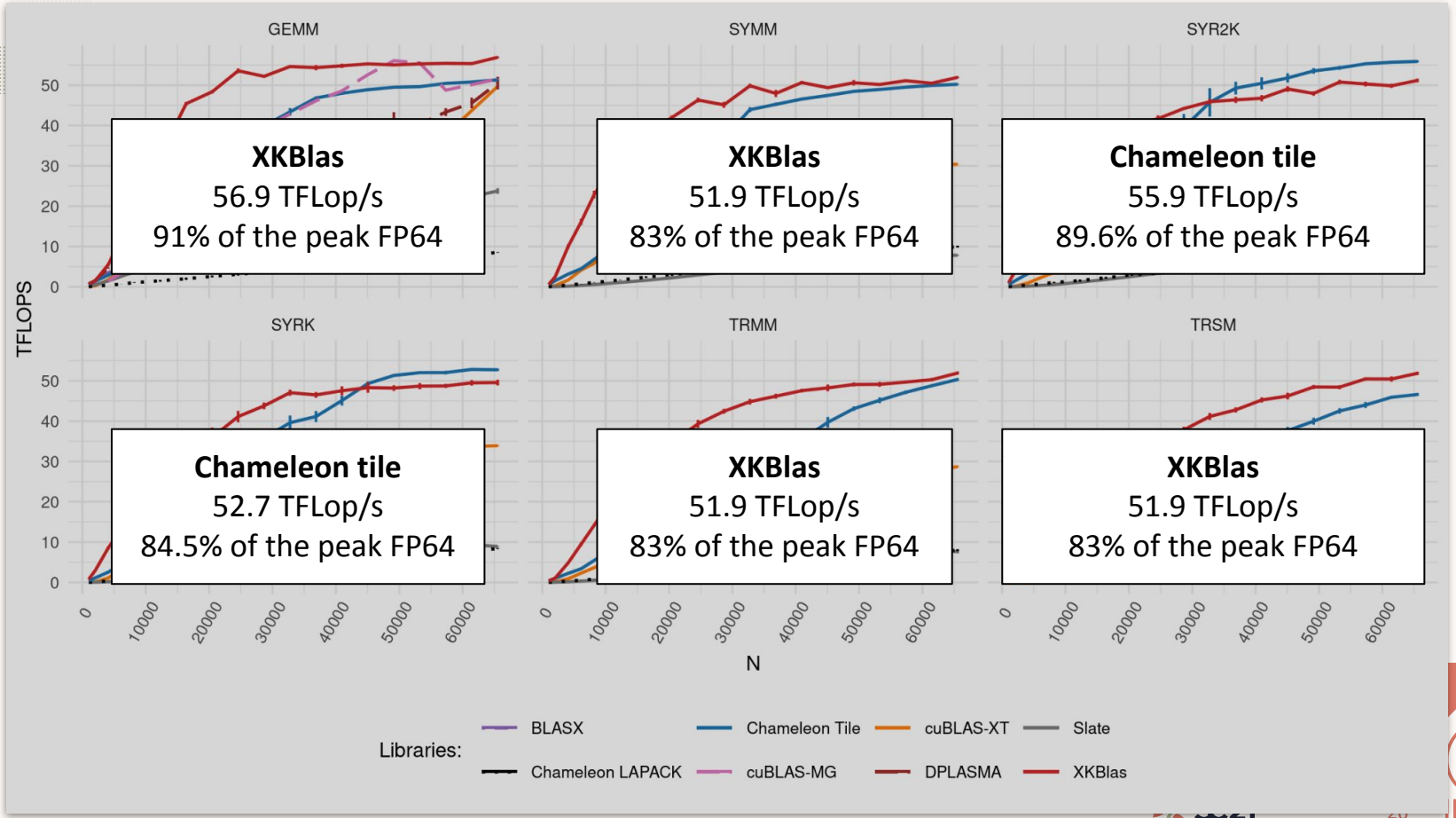
Data-on-device experiments, except XKBlas (red)

Performances of 6 BLAS kernels- DGX1 - FP64

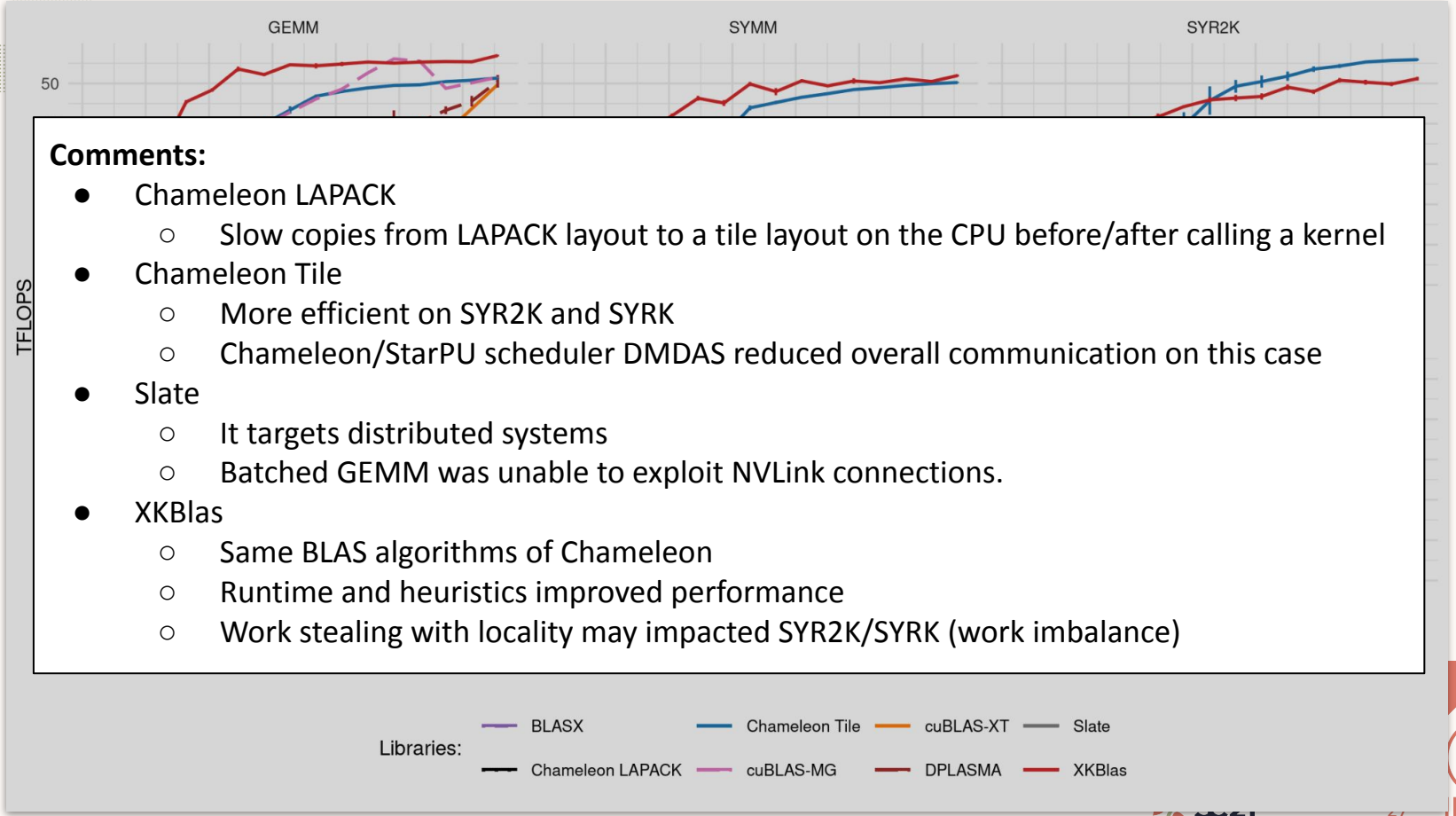


Data-on-host experiments

Performances of FP64 BLAS kernels- DGX1 - 8 GPUs



Performances of FP64 BLAS kernels- DGX1



Comments:

- Chameleon LAPACK
  - Slow copies from LAPACK layout to a tile layout on the CPU before/after calling a kernel
- Chameleon Tile
  - More efficient on SYR2K and SYRK
  - Chameleon/StarPU scheduler DMDAS reduced overall communication on this case
- Slate
  - It targets distributed systems
  - Batched GEMM was unable to exploit NVLink connections.
- XKBlas
  - Same BLAS algorithms of Chameleon
  - Runtime and heuristics improved performance
  - Work stealing with locality may impacted SYR2K/SYRK (work imbalance)

Conclusion and Perspectives

- Our two topology-aware heuristics were able to improve performance
  - GEMM with 91% of the FP64 peak on 8 GPUs machine
  - Outperformed related work libraries
    - Even with respect to a priori tile representation as Chamelon-Tile/StarPU
    - LAPACK matrix layout for better support of legacy application
  - Better on GEMM, SYMM, TRMM, TRSM
  - Work stealing imbalance on SYR2K/SYRK

https://gitlab.inria.fr/xkblas

- Future perspectives
  - Flat multi-GPU architectures (DGX-2, etc)
  - Improving scheduling heuristics
  - Expand to new application domains
    - Fluid dynamics, deep learning, etc
  - Ongoing collaboration with MUMPS Tech
    - Robust Sparse Linear Solver, http://mumps-solver.org

Evaluation of two topology-aware heuristics on level-3 BLAS library for multi-GPU platforms

Thierry Gautier (INRIA, CNRS, ENS Lyon, UCBL, France)
João Vicente Ferreira Lima (UFSM, Brazil)