XKBLAS

thierry.gautier@inrialpes.fr
CR INRIA, EPI AVALON
joão vicente ferreira lima,
jvlima@inf.ufsm.br

Outline

- Motivation for BLAS library on top of multi-GPUs
- Architecture consideration
- XKBLAS, a descriptive presentation
- Some preliminary performances
  - 4 NVidia P100 NVLink 1
  - 2 NVidia V100 NVLink 2
  - Gemini
- Preliminary results of XKBLAS with MUMPS
- Conclusion

Current trends in HPC & IA

- «High density computer»
  - several GPUs (≥ 4)
  - Sumit: 6 GPUs GV100 / node
  - NIVIDIA DGX-2: 16 GPUs V100 / node

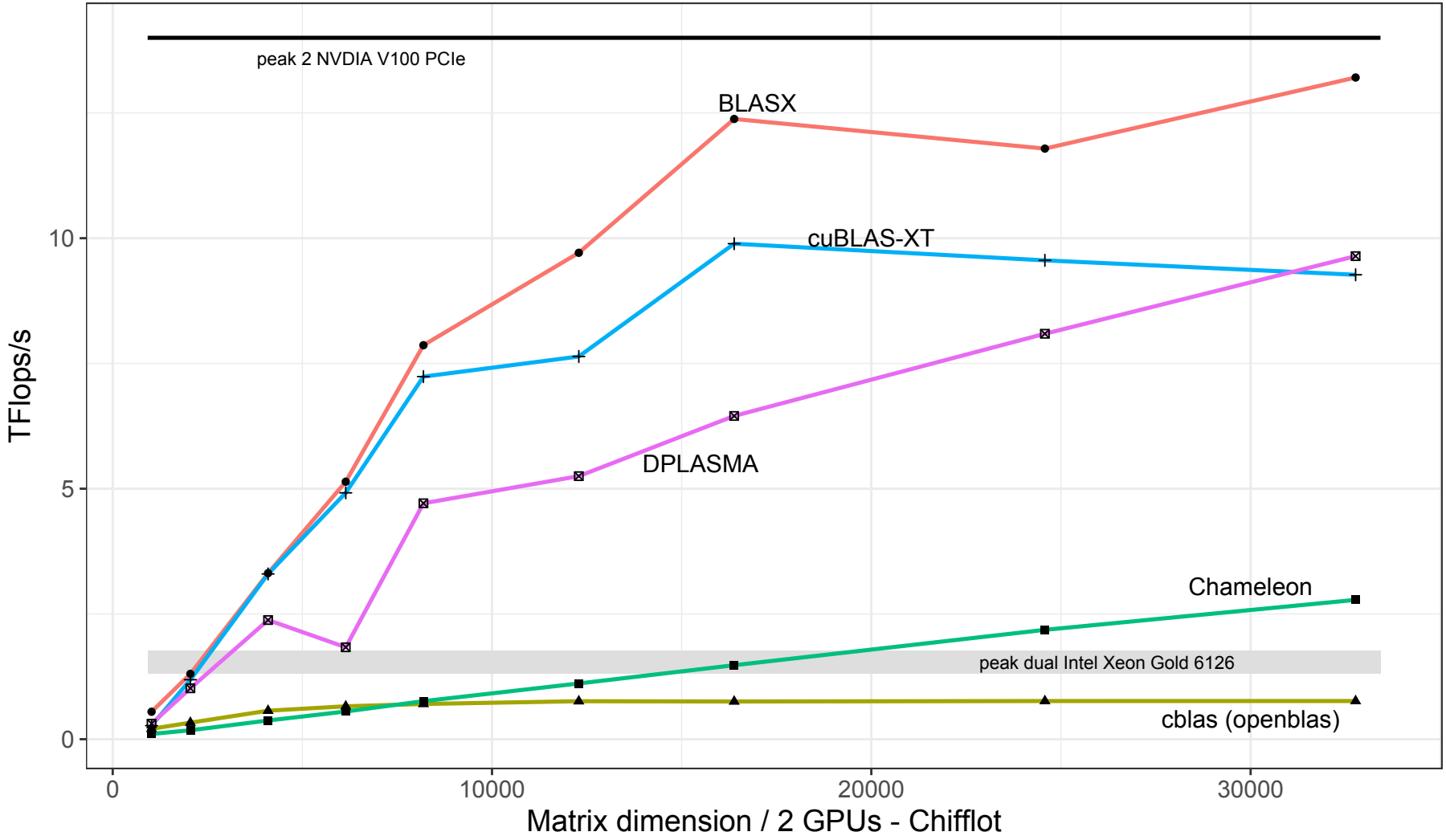
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371

High Density Computer

- Several GPUs (or accelerator) per node
  - NVidia DGX-1 (8 GPUs), DGX-2 (16 GPUs with high speed switch)
  - SuperMicro -> up to 16 GPUs
  - …
- What is expected performances for such machine ?
  - Preliminary experiments in autumn 2018 on BLAS routines
    - "legacy application performance" ~ **performance including data transfers**
    - strong impact on application: ex MUMPS

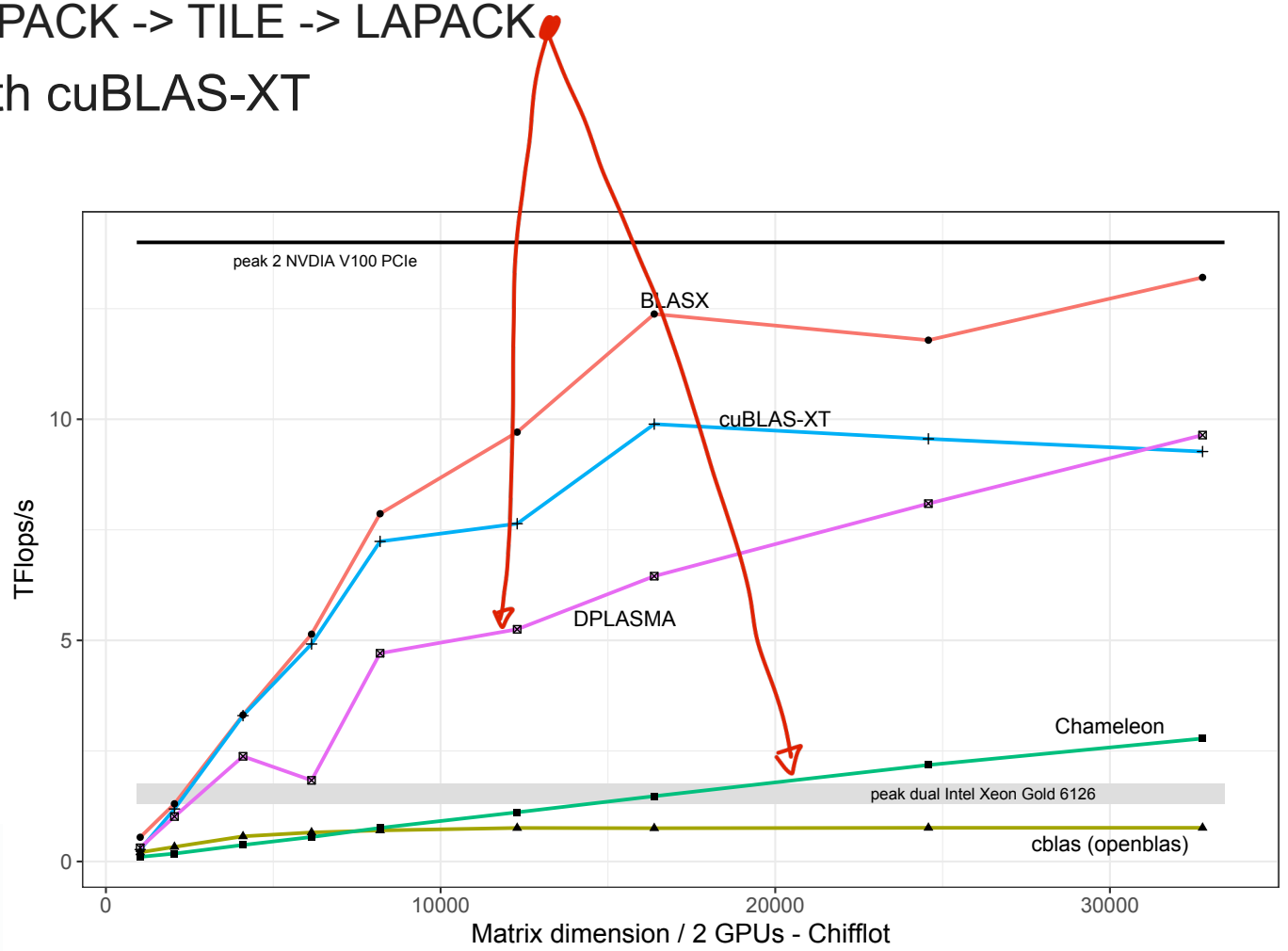
Preliminary runs [11/2018, 2-V100 PCIe]

- GEMM + data transfers.
  - data on host, LAPACK matrix
  - 2 NVidia V100 PCIe + 2 Intel Xeon, chifflot7/8.grid5K.lille.fr



Preliminary conclusion

- No full compliant BLAS multi-GPUs library exists
  - unreasonable performance drop
    - DPLASMA (~PLASMA+PaRSEC) or Chameleon (~ PLASMA + StarPU)
    - matrix conversion LAPACK -> TILE -> LAPACK
  - scalability problem with cuBLAS-XT



Impacts on large multi-GPUs?

- Scalability of BLAS library
  - what will be the performance of “BLAS” on larger multi-GPUs system?
    - Gemini, up to 8 GPUs
  - how to improve efficiency?
    - take into account architecture (NVLINK between GPUs, concurrency, …)
- Mixed-precision arithmetic

Gemini cluster

- Gemini [G5K@ENS Lyon] : 2 Nvidia DGX-1, 8 GPUs / node
  - 1GPU = 7.8 TFlop/s (double), 15.7 TFlop/s (single), 125 TFlop/s (FP16?)
  - per node = **62.4 TFlop/s** (double), **125.6 TFlop/s** (single), **1PFlop/s** (FP16)

Top500 November 2004

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	**BlueGene/L beta-System** - BlueGene/L DD2 beta-System (0.7 GHz PowerPC 440) , IBM IBM/DOE United States	32,768	70,720.0	91,750.0	

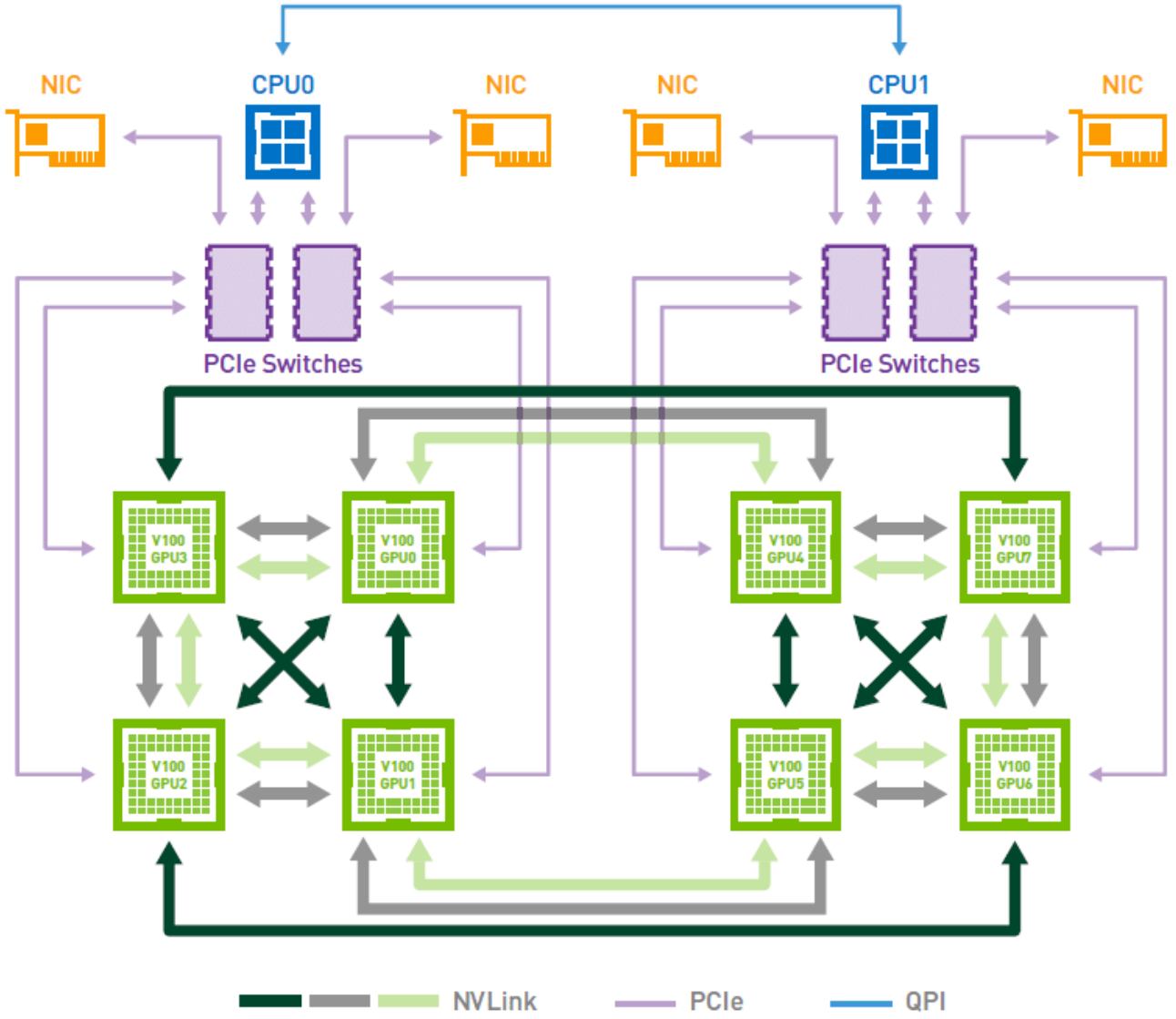
Top500 June 2004

Rank	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	Earth-Simulator , NEC Japan Agency for Marine-Earth Science and Technology Japan	5,120	35,860.0	40,960.0	3,200

- Gemini (1 node)
  - certainly 1st in Top500 list of June 2004!
  - 3kW versus 3200kW

Architecture Gemini / DGX-1

- Internal communication links between CPUs (Xeon) /GPUs (NVidia V100)



Topology DGX-1 Gemini

- 1 NVLink= 25GB/s each direction, 50GB/s per link
- Bidirectional benchmark matches peak perf & topology
  - NVidia Sample / p2pBandwidthLatencyTest

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

DGX-1 Gemini GB/s

750.48 : local GPU
96.44 : 2 NVLinks
48.39 : 1 NVLink
17.75 : PCIe

Asynchronous concurrent executions

- CUDA exposes the following operations as independent tasks that can operate concurrently with one another:
  - Computation on the host;
  - Computation on the device;
  - Memory transfers from the host to the device;
  - Memory transfers from the device to the host;
  - Memory transfers within the memory of a given device;
  - Memory transfers among devices.
- Asynchronous execution thanks to cudaStream concept
  - fifo per stream, concurrent between streams
  - Launch concurrent computation on device to increase occupancy
    - compute capability >= 2.X
  - data transfers can overlap together, with CPU or GPU computations
    - number of “asyncEngine” (6 on V100)
- Host data have to be « pinned » (non pageable) to let asyncEngine(s) to work best

BLAS library

- Basic Linear Algebra Subroutine
  - 9 kernels
    - GEMM / TRMM / SYMM / HEMM
    - TRSM
    - SYRK / SYR2K
    - HERK / HER2K

- Why BLAS ?
  - performance portability (of dense linear algebra subroutine)
  - for different hardwares and memory hierarchies

- Lot of related works but few “drop in replacement” libraries on multi-GPU
  - NVBLAS, BLASX (gemm only),

XKBLAS = BLAS over XKAAPI

- Algorithm extracted from PLASMA
  - tile algorithms
  - modified to consider **LAPACK representation** in place of Tile representation
  - only asynchronous invocations have been keept

- XKaapi = runtime for multi-CPU / multip-GPU developped in MOAIS Team [2005-2015] Grenoble
  - simple & fine grain
  - task with data flow dependencies
  - scheduling by work stealing + heuristics

- XKBLAS = compliant BLAS API + extended API (for the expert)
  - separation of concern data movement / computations
    - better composition of kernels
  - up-to-date exploitation of concurrency levels on multi-GPUs thanks to XKaapi RT

Managing GPU memory in XKBLAS

- Inherited from XKaapi
  - GPU memories store copies of data managed by XKaapi
  - Distributed Shared Memory
    - follows « dag consistency » introduced by Cilk [IPPS96]

- User has to explicitly invoke operation to make CPU memory coherent with respect to data on GPUs

```
xkblas_memory_coherent_async
```

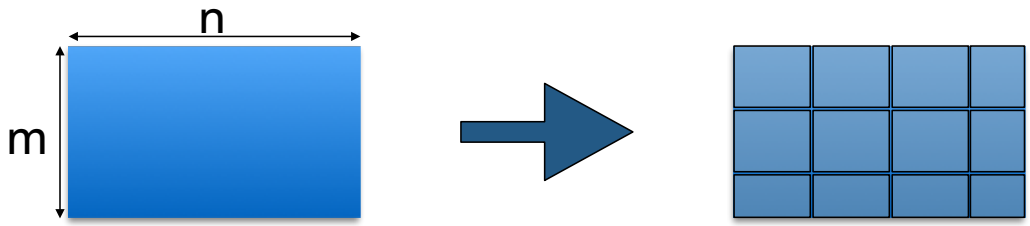
Data transfers in XKBLAS

- Inherited from XKaapi [SBAC/PAD 2012, IPDPS 2013] and compatible with NVIDIA way of exploiting concurrent executions
  - multiple streams per GPU device
    - 1 stream for Host -> Device transfers
    - 1 stream for Device -> Host transfers
    - k streams for Cuda Kernel launch

- Device to device communication (Peer2Peer access)
  - several copies may reside in GPU memories
  - make copy from the GPU with fastest communication first

Tile algorithm in XKBLAS

- BLAS = tile algorithms from PLASMA [UTK]



- Skeleton of modified PLASMA tile algorithm (here gemm)

```
int xkblas_zgemm_async(
  int transA, int transB, int M, int N, int K,
  const Complex64_t* alpha, const Complex64_t *A, int LDA,
  const Complex64_t *B, int LDB,
  const Complex64_t* beta,  Complex64_t *C, int LDC )
{
  ...
  /* NoTrans, NoTrans case */
  for (m = 0; m < Cmt; m++)
    for (n = 0; n < Cnt; n++)
      for (k = 0; k < Ant; k++)
      {
          zbeta = k == 0 ? *beta : 1.0;
          INSERT_TASK_zgemm(
              CblasNoTrans, CblasNoTrans,
              *alpha, A(m, k), ldam,
                      B(k, n), ldbk,
              zbeta, C(m, n), ldcm);
      }
  ...
}
```

Standard BLAS dgemm

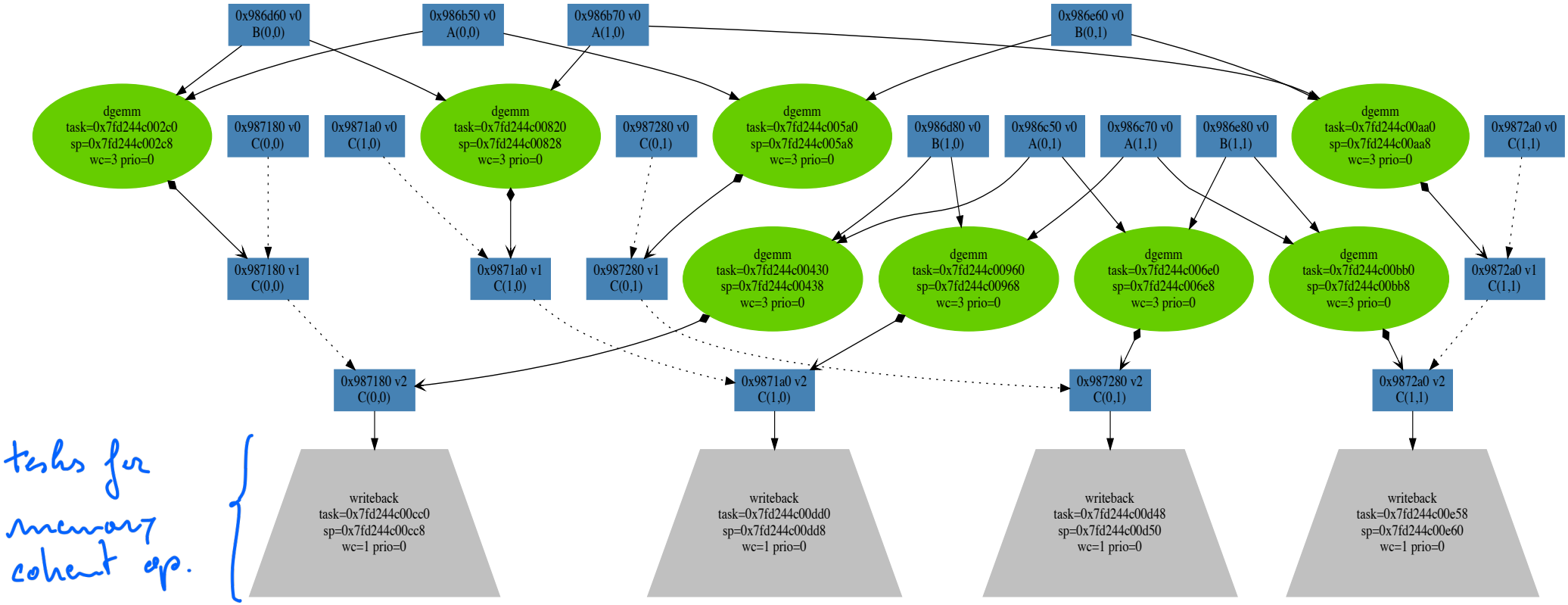
- BLAS [C/Fotran] dgemm == calls to XKBlas extended API

```
/* 1/ asynchronus invocation: dgem */
xkblas_dgemm_async( CblasNoTrans, CblasNoTrans, M, N, K,
  &alpha, A, lda,
          B, ldb,
  &beta,  C, ldc);

/* 2/ asynchronus invocation:coherent update of C on the host */
xkblas_memory_coherent_async(uplo, memflag, M, N, C, ldc, sizeof(double));

/* 3/ wait completion of previous asynchronous operations */
xkblas_sync();
```

- `uplo` = `CblasUpper` xor `CblasLower` xor (`0== CblasLower| CblasUpper`)
- `memflag` = 1 invalidate complementaty part in the caches

Data flow graph at runtime

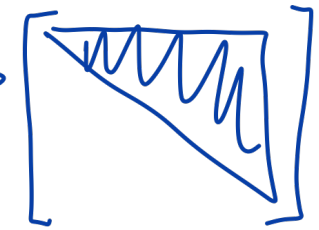
- unfold nested loops over the tiles and create tasks



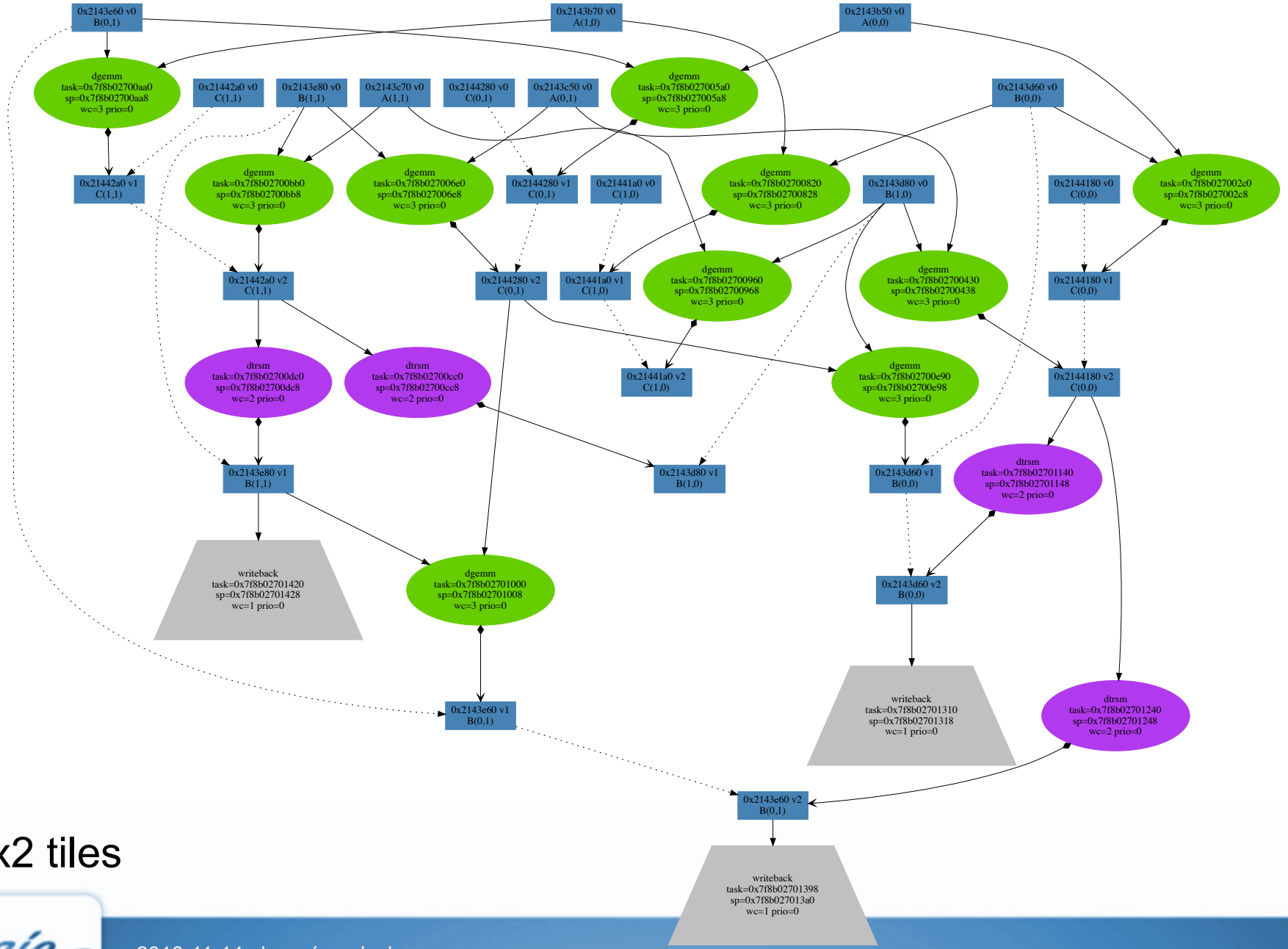
2x2 tiles matrix

dgemm+trsm+memory coherence

```
xkblas_dgemm_async( CblasNoTrans, CblasNoTrans, n, n, n,
  &alpha, A, n,
          B, n,
  &beta,  C, n);

xkblas_dtrsm_async( CblasLeft, CblasUpper, CblasNoTrans, CblasUnit,
  n, n,
  &alpha, C, n,
          B, n);

xkblas_memory_coherent_async(
   CblasUpper,
   0,
   n, n,
   B, n, sizeof(double)
);
```



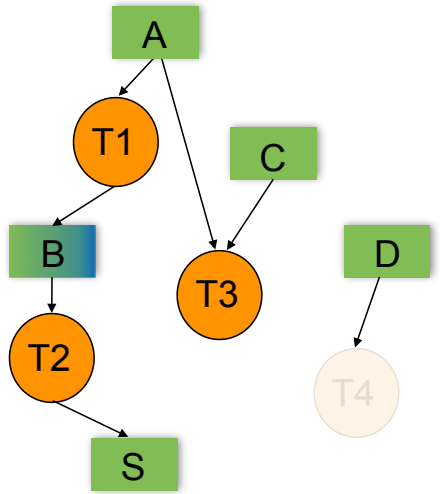
dgemm+trsm+memory coherence



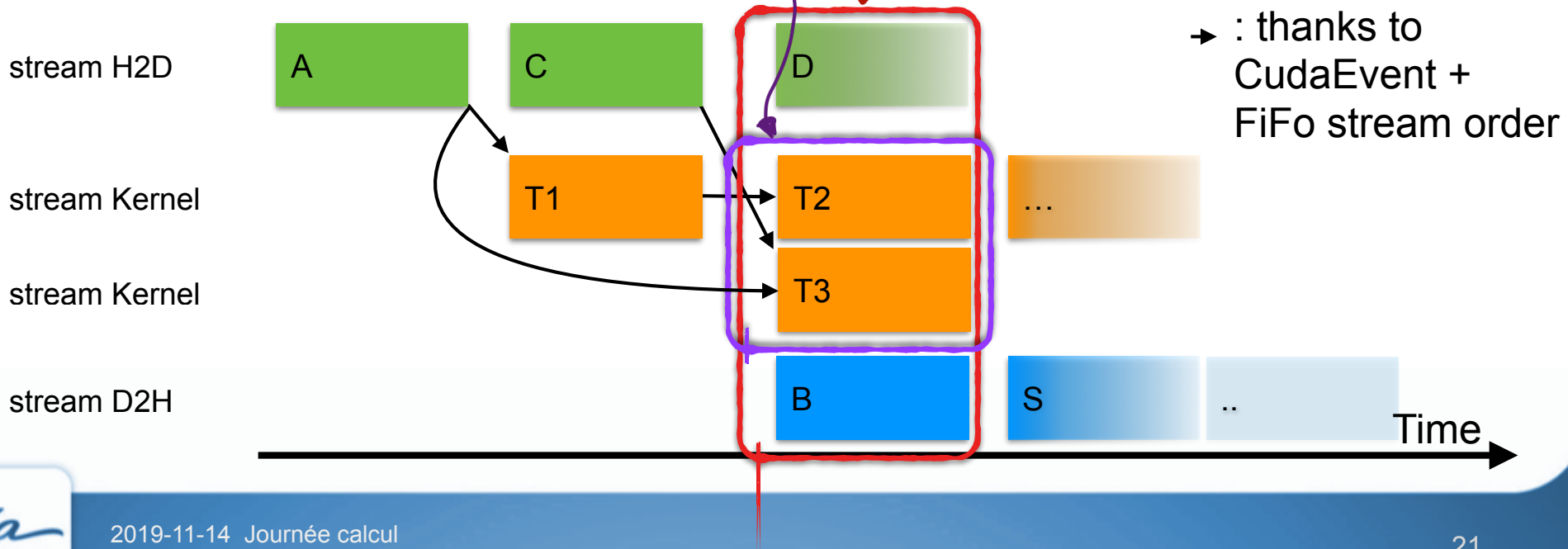
- 2x2 tiles

Concurrent GPU Operations

- Task graph execution
  - fetch input data on GPU_i if not present
  - write back when needed



- 2 levels of concurrency
  - between data transfers and GPU computation
  - between GPU computations



What kind of performance to consider?

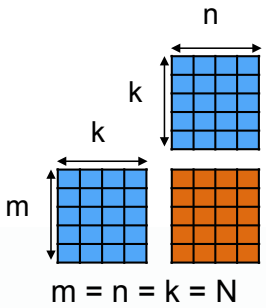
- Peak or real performances ?

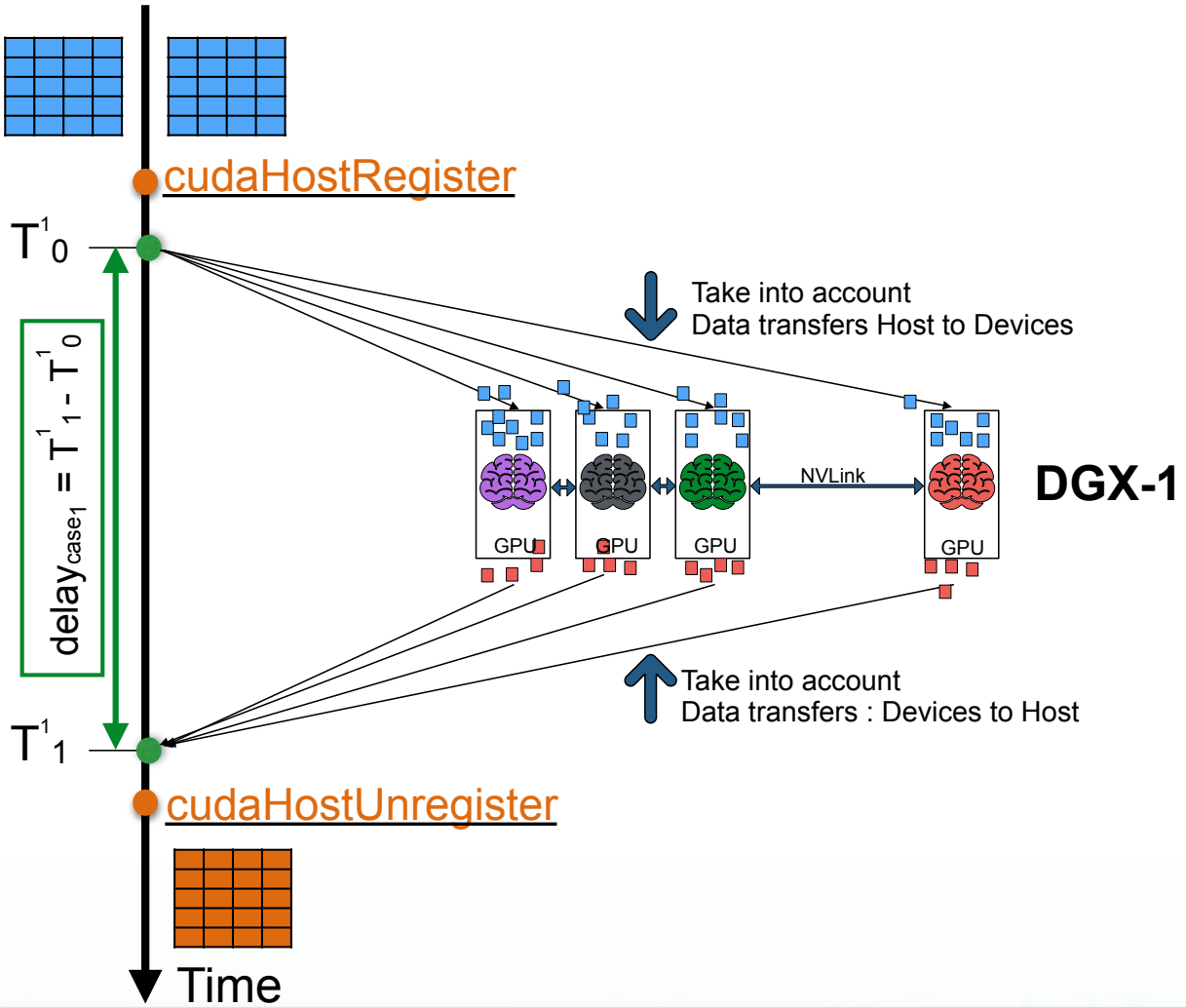
- raw performance ≈ performance **without** considering:
  - initial/final communication Host->Device(s) and Device(s) -> Host
  - pining host memory
  - subquestion: how fast are those operations ?

- legacy application performance ≈ performance when only changing BLAS library of the application
  - take into account initial/final communication H2D, D2H
  - does not take into account pining that could be generaly amortized

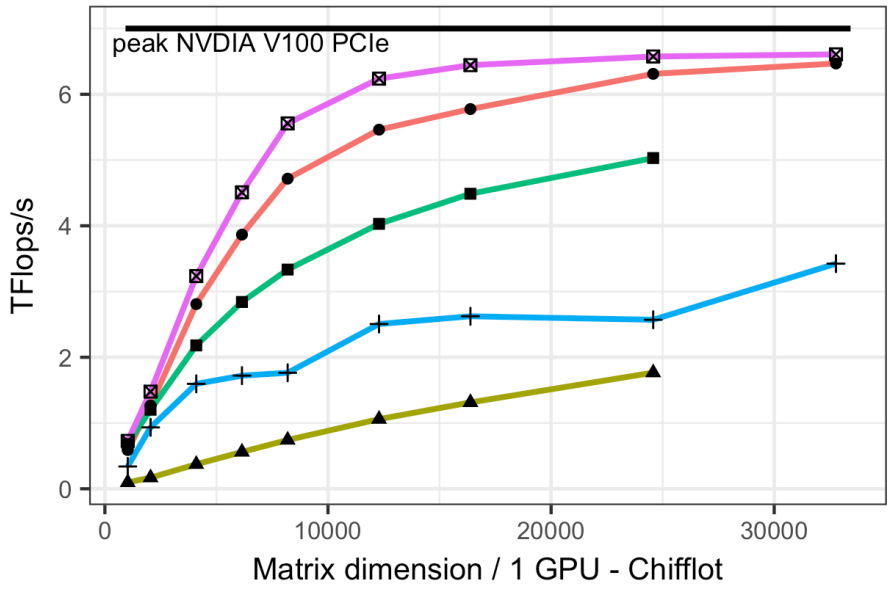
Principe of the measure

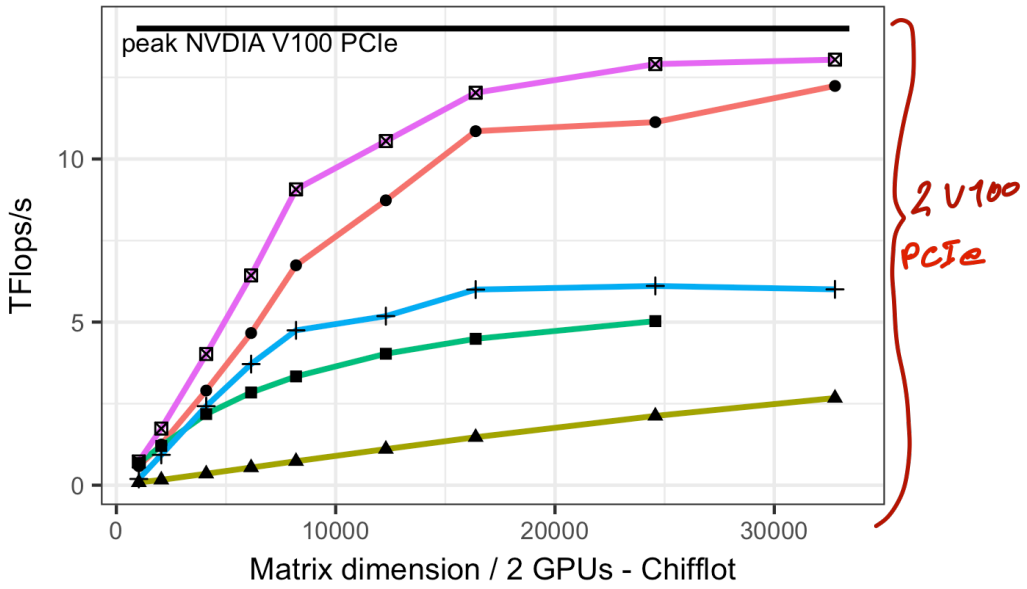
- Time to perform GEMM operation
  - do not take into account time to register memory

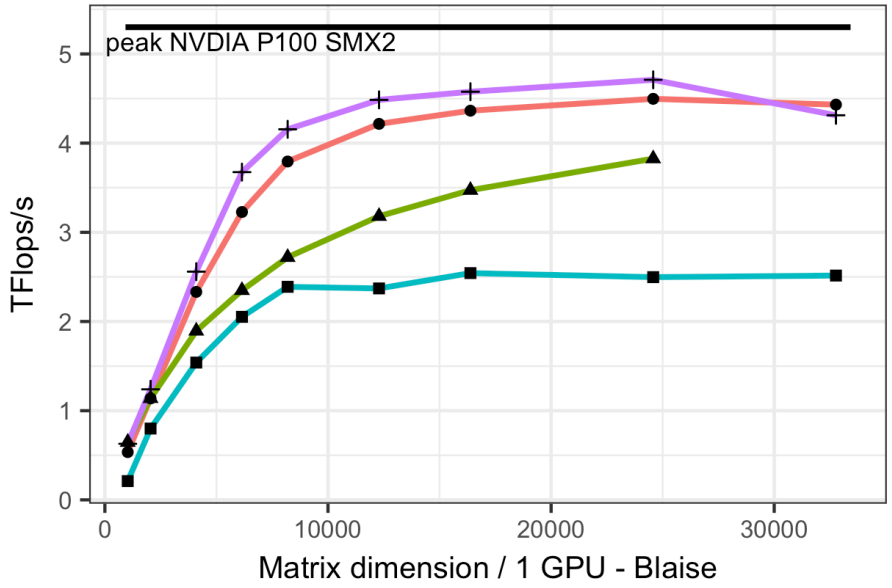


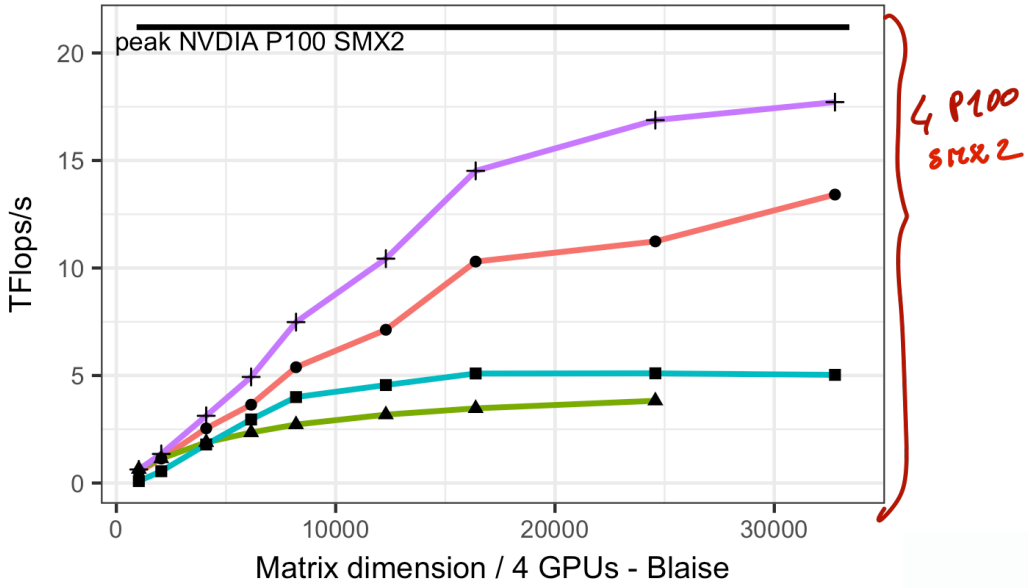


DGEMM Chifflot@G5K, Blaise@UFRGS





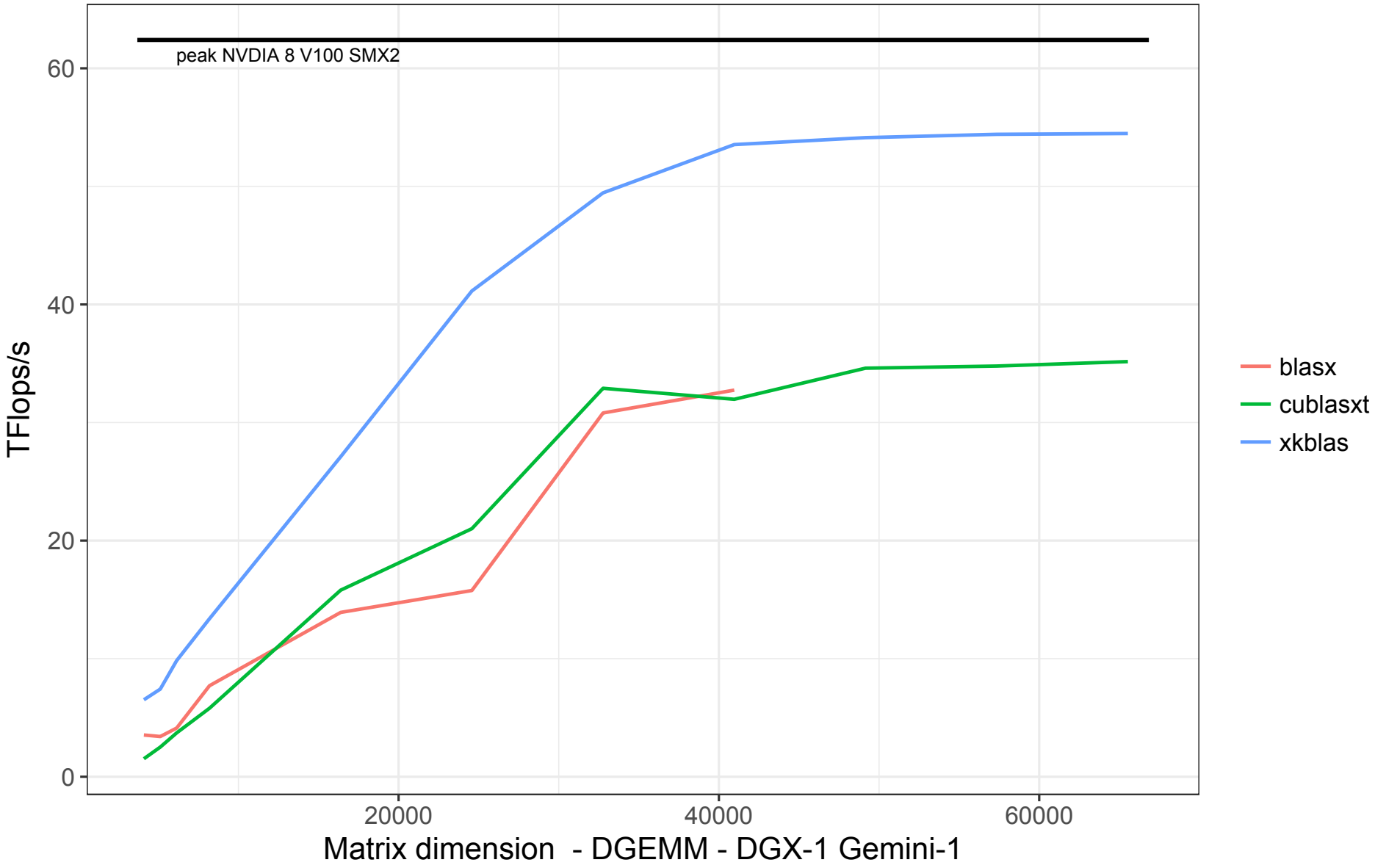




RUNTIME — BLASX — Chameleon — cuBLAS — cuBLASxt — XKBlas

GEMM on Gemini

- xkblas ~ 54TFlop/s FP64, ~ 106TFlop/s FP32

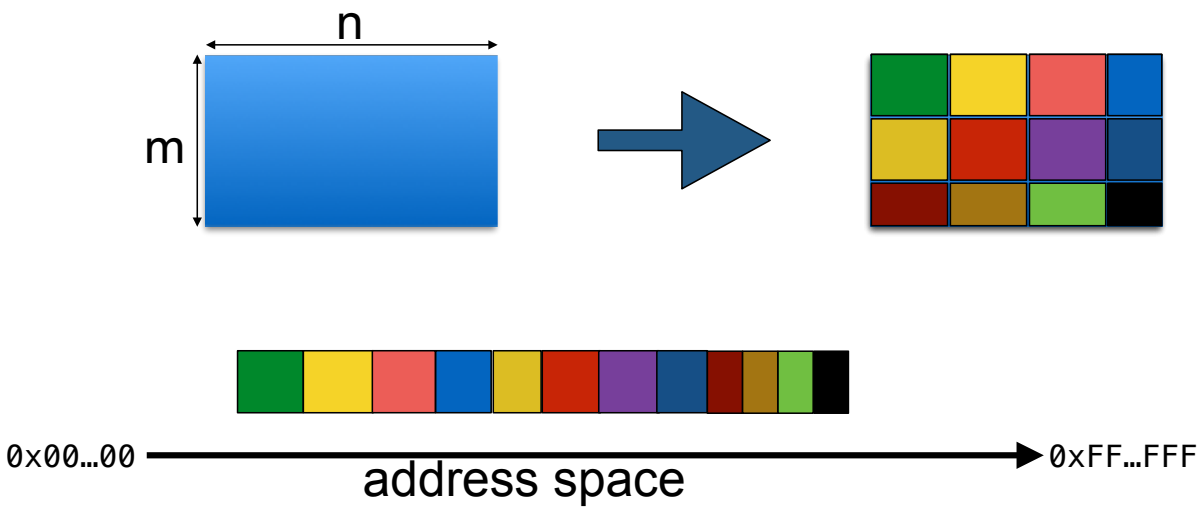


Comparisons with Tile based libraries

- DPLASMA / PaRSEC
- Chameleon / StarPU

} Same origin : PLASMA [UTK]

- chifflot@Grid5K.lille.fr: 2 NVidia V100 PCIe
- blaise@gppd-hpc.inf.ufrgs.br: 4 NVidia P100 NVLink 1

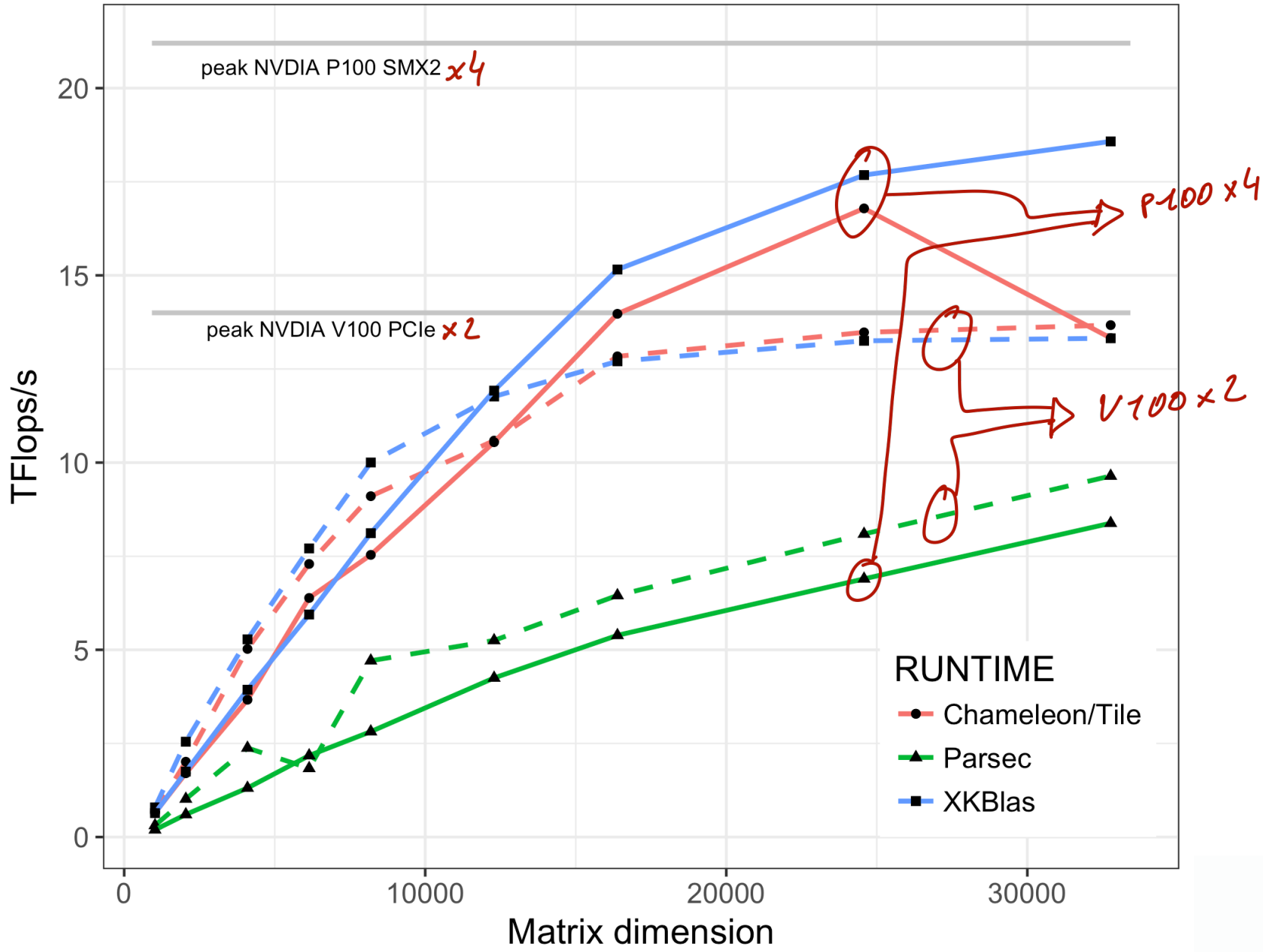


logical view

Tile representation

LAPACK
Row major representation

Result [11/2018] GEMM FP64

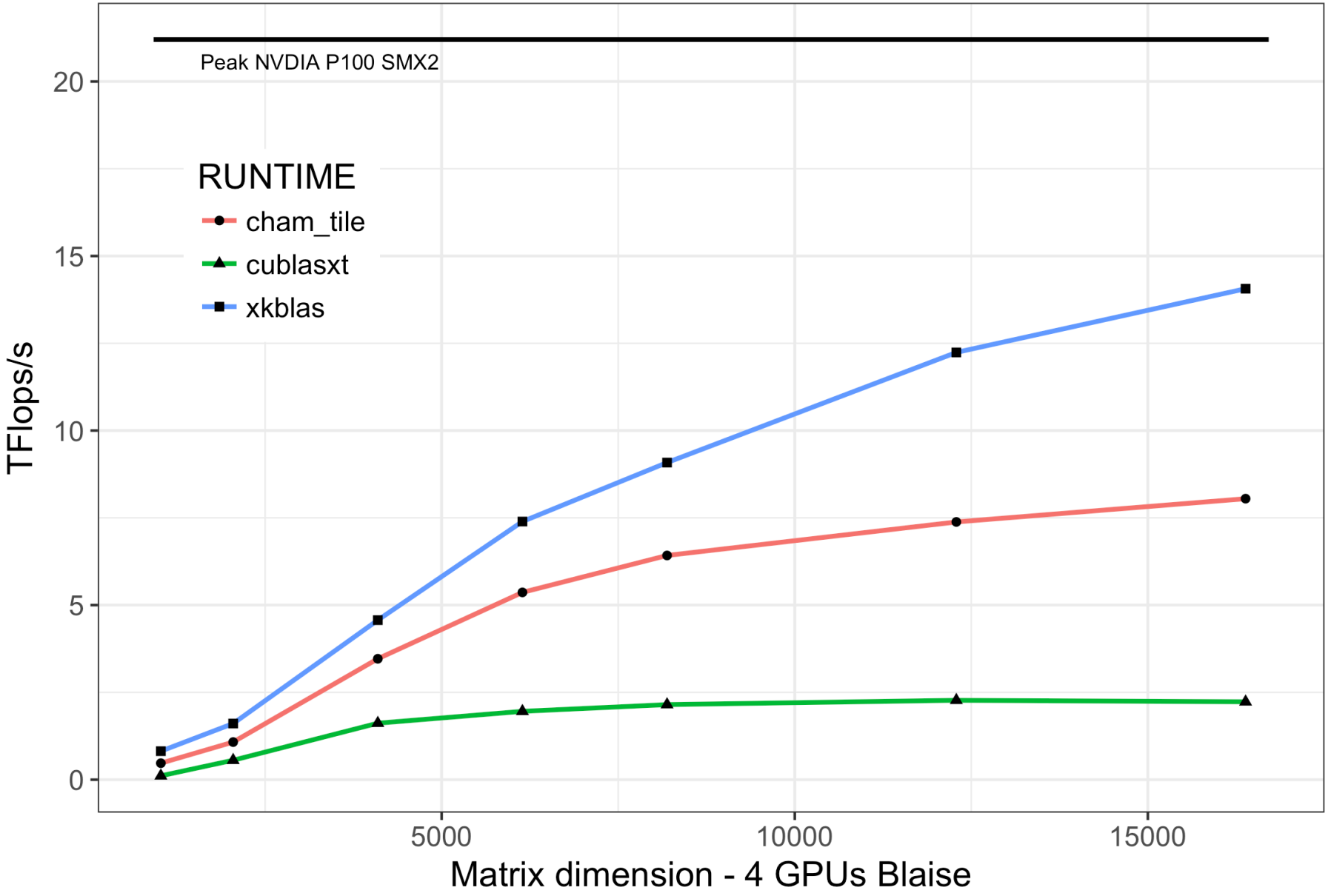


Composition GEMM+TRSM

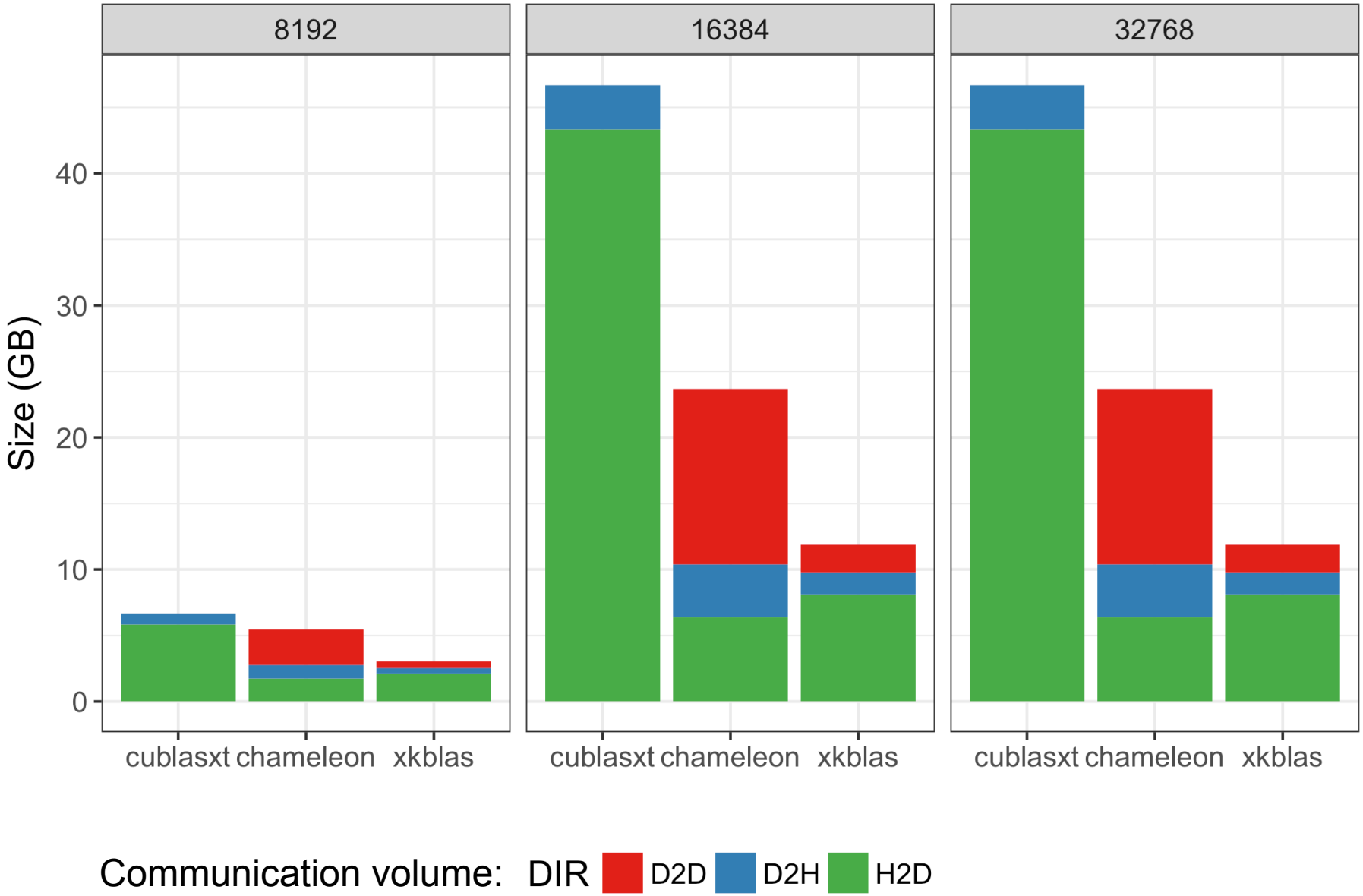
```
xkblas_dgemm_async( CblasNoTrans, CblasNoTrans, n, n, n,
  &alpha, A, n,
          B, n,
  &beta,  C, n);

xkblas_dtrsm_async( CblasLeft, CblasUpper, CblasNoTrans, CblasUnit,
  n, n,
  &alpha, C, n,
          B, n);

xkblas_memory_coherent_async(
   CblasUpper,
   0,
   n, n,
   B, n, sizeof(double)
);

xkblas_sync();
```

- Goal: avoid synchronization between BLAS invocations
- Blaise: 4 GPUs NVidia P100 + NVLink 1

Result GEMM+TRSM / Blaise FP64



Result GEMM+TRSM / Blaise



Experimentation with MUMPs

- Typical of several scientific HPC applications
  - Linear algebra with LAPACK matrix representation
  - Data-On-Host: performance should take into account data transfers

- Autumn 2018 [M. Durand, équipe ROMA]
  - NVBLAS not satisfactory
  - Port on cuBLAS-XT
  - Port on XKBLAS using synchronous invocation
  - Essentially one thread made calls to BLAS offloaded to GPUs
  - Optimization of the threshold between CPU / GPU

GEMM, TRSM
GEMMT
} few kernels

Olympe - SPD MechaStruct8M- Performance evolution

15 000s #core=1

CALMIP Olympe computer, timings in seconds (2x18 Intel Skylake 6140 @2.3 Ghz - Nvidia Volta (V100 - 7,8 Tflops DP))				
MechaStruct8M, symmetric SPD, N = 8M, NNZ = 363M				
#cores	(#GPUs)	CPU	cublasXt	XKBlas
MUMPS +				
18	(1)	1 167	1 451	864
MUMPS + with pinning				
#cores	(#GPUs)	CPU	cublasXt	XKBlas
18	(1)	1 167	937	727
18	(2)		932	703
18	(4)		788	700
MUMPS + with pinning and improved multithreading of non-GPU kernels[8]				
#cores	(#GPU)	CPU	cublasXt	XKBlas
18	(1)	886	734	511
18	(2)		743	494
18	(4)		604	426

[8] includes improved copy-scale algorithm and GEMMT with XKBlas

On medium-size problems also from SuiteSparse collection

CALMIP Olympe computer, timings in seconds

(2x18 Intel Skylake 6140 @2.3 Ghz - Nvidia Volta (V100 - 7,8 Tflops DP))

Matrix	CPU only (18 cores)	CPU (18 cores) + 1 GPU cublasXt	CPU (18 cores) + 1 GPU XKBlas
Symmetric Positive Definite test cases			
Serena (N=1,3M)	50	51	44
3D Laplacian (N=4M)	218	206	162
Bump 2911 (N=2,9M)	244	223	163
Queen 4147 (N=4,1M)	334	307	225
MechaStruct3M (N=2,8M)	125	124	101
Symmetric Indefinite test cases			
perf008d (N=1.9M)	162	162	142
perf008ar (N=3.9M)	507	474	395
Unsymmetric test cases			
RM07R (N=0,4M)	36	35	30

Only preliminary work ... $\rightarrow$ much work to be done

Ongoing work with MUMPS

- Collaboration AVALON / MUMPS Technology [2020]
  - Evaluate performance of MUMPS on top of XKBLAS
    - how asynchronous invocations can help to overlap data transfers / GPU kernel by CPU computation?
      - how kernels composition can help in reducing communication?
    - how the performance varies when multiple threads “flood” GPUs?

Conclusion

- Multi-GPUs allow very high performance
  - Memory pinning is always expensive, incredibly expensive
    - cost increase with number of GPUs
  - take care of communication topology between GPUs

- Except cuBLAS-XT, BLASX, XKBLAS,
  - MAGMA, PaRSEC, Chameleon/StarPU, KBLAS

  require extensive changes on the legacy code

- XKBLAS based on simple concepts
  - Data flow execution engine
  - Asynchronous invocations
  - Separation of data movement / distribution and computation

} composition of kernels

https://gitlab.inria.fr/xkblas or thierry.gautier@inrialpes.fr

Questions ?