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
  - NVIDIA DGX-2: 16 GPUs V100 / node

<table>
<thead>
<tr>
<th>Rank</th>
<th>System</th>
<th>Core</th>
<th>Rmax [TFlop/s]</th>
<th>Rpeak [TFlop/s]</th>
<th>Power [kW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>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</td>
<td>2,414,592</td>
<td>148,600.0</td>
<td>200,794.9</td>
<td>10,096</td>
</tr>
<tr>
<td>2</td>
<td>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</td>
<td>1,572,480</td>
<td>94,640.0</td>
<td>125,712.0</td>
<td>7,438</td>
</tr>
<tr>
<td>3</td>
<td>Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China</td>
<td>10,649,600</td>
<td>93,014.6</td>
<td>125,435.9</td>
<td>15,371</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>Rank</th>
<th>System</th>
<th>Cores</th>
<th>Rmax GFlop/s</th>
<th>Rpeak GFlop/s</th>
<th>Power kW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BlueGene/L beta-System - BlueGene/L DD2 beta-System [0.7 GHz PowerPC 440], IBM IBM/DOE United States</td>
<td>32,768</td>
<td>70,720.0</td>
<td>91,750.0</td>
<td></td>
</tr>
</tbody>
</table>

• 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

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

DGX-1 Gemini GB/s
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 $\geq 2.X$
  - 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 kept

• XKaapi = runtime for multi-CPU / multip-GPU developed 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
  \texttt{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)**

```c
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), ldc);
            }
    ...
}
```
Standard BLAS dgemm

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

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

/* 2/ asynchronous 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 complementatry part in the caches
Data flow graph at runtime

• unfold nested loops over the tiles and create tasks
dgemm+trsm+memory coherence

```c
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 if not present
  – write back when needed

• 2 levels of concurrency
  – between data transfers and GPU computation
  – between GPU computations

: thanks to CudaEvent + FiFo stream order
What kind of performance to consider?

• Peak or real performances?

• raw performance \(\approx\) 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 \(\approx\) 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 generally amortized
Principe de la mesure

- Temps pour effectuer l'opération GEMM
  - ne pas prendre en compte le temps pour enregistrer la mémoire

\[ m = n = k = N \]

\[ T_0^1 = T_1^1 - T_0^1 \]

delay case 1

CUDA Host Register

CUDA Host Unregister

DGX-1

Données transférées du Host vers les Devices

Données transférées des Devices vers le Host
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 [VTK]

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

![Diagram withTick representation of a matrix showing logical view, address space, and LAPACK row major representation.]
Result [11/2018] GEMM FP64

![Graph showing performance comparison between different matrix dimensions and runtime options.](image-url)

- Peak NVIDIA P100 SMX2 x4
- Peak NVIDIA V100 PCIe x2

**RUNTIME**
- Chameleon/Tile
- Parsec
- XKBlas

**Axes:**
- Y-axis: TFlops/s
- X-axis: Matrix dimension
Composition GEMM+TRSM

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

```c
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();
```
Result GEMM+TRSM / Blaise FP64

RUNTIME
- cham_tile
- cublasxt
- xkblas

TFlops/s

Matrix dimension - 4 GPUs Blaise
Result GEMM+TRSM / Blaise

Communication volume:  
- **DIR**  
- **D2D**  
- **D2H**  
- **H2D**
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

© P. Amestoy, J.-Y. L’Excellent - CC Meeting 2019
## CALMIP Olympe computer, timings in seconds

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

<table>
<thead>
<tr>
<th>MechaStruct8M, symmetric SPD, N = 8M, NNZ = 363M</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>#cores</strong></td>
</tr>
<tr>
<td>MUMPS +</td>
</tr>
<tr>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MUMPS + with pinning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>#cores</strong></td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MUMPS + with pinning and improved multithreading of non-GPU kernels(^8)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>#cores</strong></td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>18</td>
</tr>
</tbody>
</table>

\(^8\) 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))

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

Only preliminary work ... → 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

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