Generic matrix multiplication for multi-GPU accelerated distributed-memory platforms over PaRSEC

Thomas Herault, Yves Robert, George Bosilca and Jack Dongarra

Innovative Computing Laboratory, the University of Tennessee
Ecole Normale Superieure de Lyon

November 18, 2019

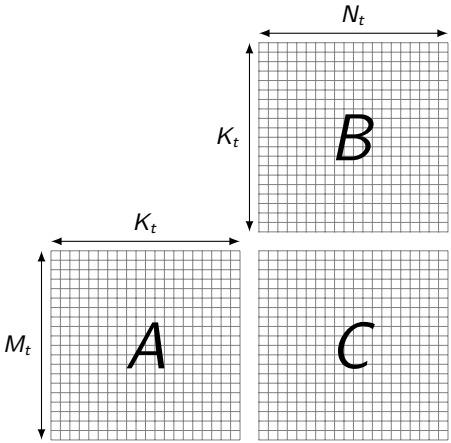
10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems

Problem Definition

Let A, B, and C be matrices of real numbers.
A is of dimension (M, K), B is (K, N), and C is (M, N).
We aim to compute $C = C + A \cdot B$:

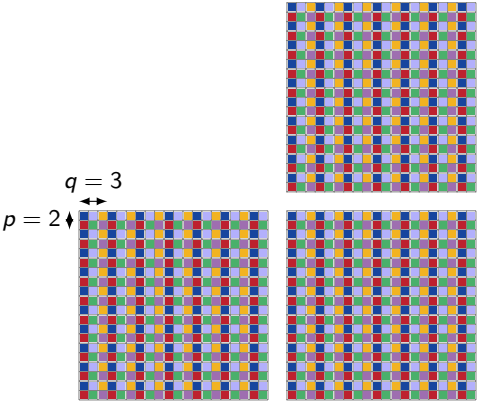
$$C_{ij} = C_{ij} + \Sigma_{k=0}^{K} A_{ik} \cdot B_{kj}$$

A, B, and C are *tiled* in square tiles of size (t, t).
There are $M_t \times K_t$ tiles in A, $K_t \times N_t$ tiles in B, and $M_t \times N_t$ tiles in C
$M_t = \lceil \frac{M}{t} \rceil, N_t = \lceil \frac{N}{t} \rceil, K_t = \lceil \frac{K}{t} \rceil$



Problem Definition

A, B, and C are distributed in a 2D Block Cyclic distribution, with a process grid of $p \times q$ processes.

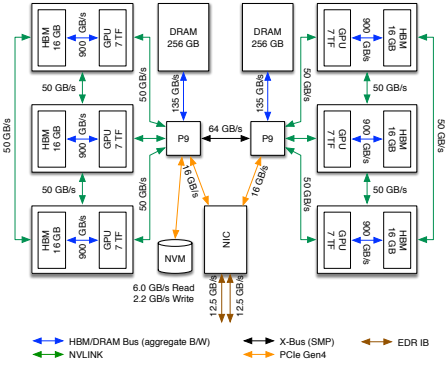


SUMMIT

SUMMIT architecture

- Oak Ridge National Laboratory Leadership Computing Facility
- 9,216 compute nodes
  - POWER9 22-core CPUs with 512 GB / node
  - 6 Nvidia Tesla V100 GPUs with 16GB / card

Heavy nodes with many accelerators per node

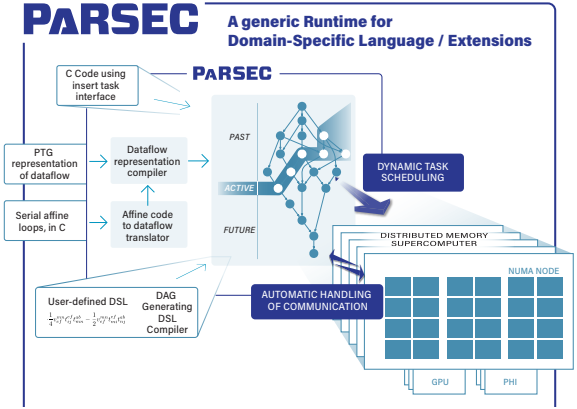


"Summit Interconnection Network Spectrum MPI and InfiniBand", Christopher Zimmer, Dec. 2018

PaRSEC

PaRSEC: Parallel Runtime Scheduling and Execution Controller

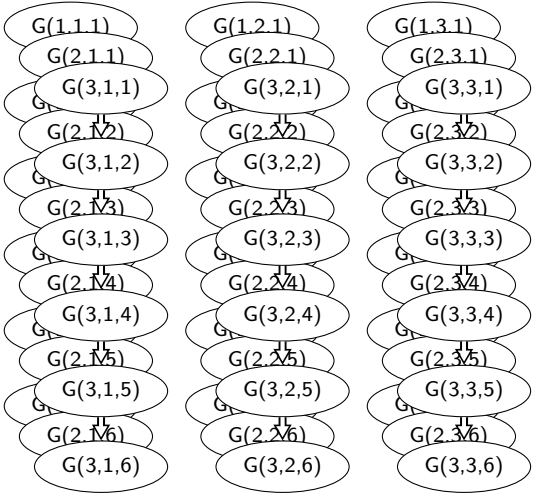
- Innovative Computing Laboratory
- Exascale Computing Project
- Task Based, Distributed, Hybrid Architectures
- Domain Specific Languages
- Runtime System Manages:
  - Data Movement between nodes and between *devices*
  - Computing resource



GEMM over PaRSEC

Parameterized Task Graph

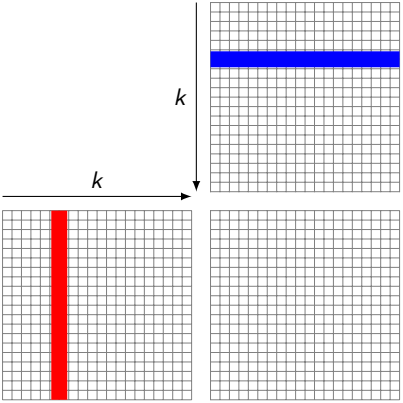
```
GEMM(m, n, k)
  m = 1 .. MT
  n = 1 .. NT
  k = 1 .. KT

:C(m, n)

RW   C <- k==1  ? C(m, n) : C GEMM(m, n, k-1)
       -> k==KT ? C(m, n) : C GEMM(m, n, k+1)
READ A <- A(m, k)
READ B <- B(m, k)

BODY [type=CUDA]
  cublas_dgemm(CUBLAS_OP_N, CUBLAS_OP_N,
               t, t, t, alpha,
               A, t, B, t, beta, C, t);
END
```



Scalable Universal Matrix Multiplication Algorithm over PaRSEC

- Tiles belonging to $A(*, 0 \leq k \leq K_t)$ are sent to nodes that need them in sequence
- Tiles belonging to $B(0 \leq k \leq K_t, *)$ are sent to nodes that need them in sequence
- When a process receives a set of tile from A and B, it updates all the tiles of C that needs them

- Limit the amount of memory needed at any step
- Provides 80% of practical peak on manycore architectures



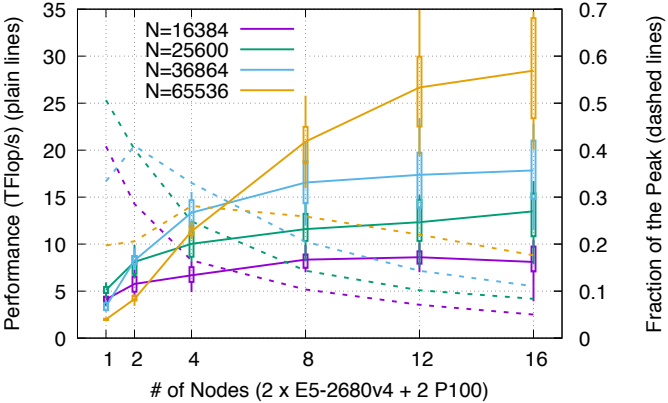
SUMMA over PaRSEC

1 to 16 nodes, 2 x E5-2680v4 + 2 P100
Tiles of 1024×1024
Square problem ($M_t = N_t = K_t$)
Problem size from 16k to 65k
Double precision

- Fraction of the peak bumps up when the amount of memory per node gets under a threshold

⇒ Scheduling is thrashing the GPU memory

⇒ Performance require a tighter control of the active set and data movements

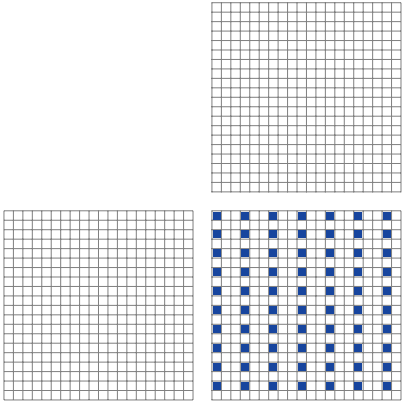


Bosilca, Genet, Harrison, Herault, Javanmard, Peng, Valeev. "Tensor contraction on distributed hybrid architectures using a task-based runtime system", 2018

GEMM Algorithm for GPUs

Node-level Task Distribution

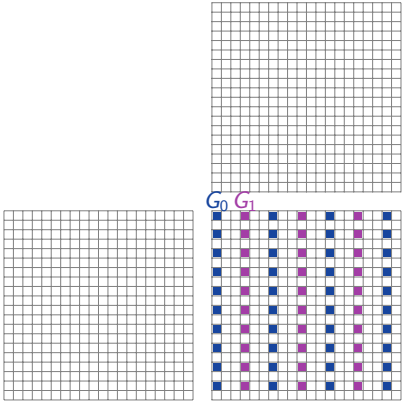
- data affinity
- owner computes heuristic
- 2D block cyclic distribution



GEMM Algorithm for GPUs

GPU-level Task Distribution

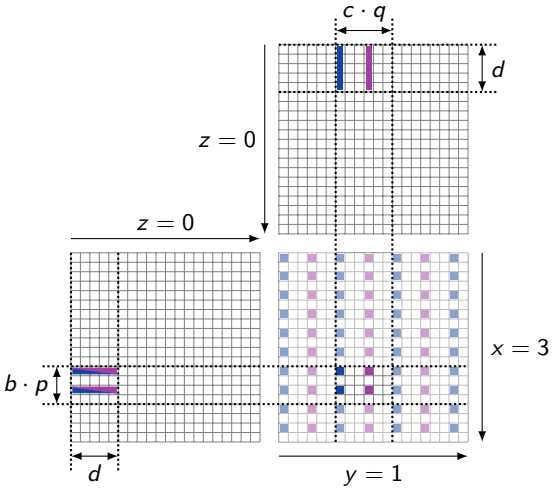
- Fixed, using PaRSEC memory advise API
- Round-robin assignment of tile-columns to the different GPUs



GEMM Algorithm for GPUs

Node-Level Blocking

- Active set of GEMMs is defined locally by a block of coordinate (x, y, z)
- (x, y) defines a block in C, of *local* size $b \times c$
- (x, z) defines a block in A, of *global* size $d \times bp$
- (z, y) defines a block in B, of *global* size $d \times cq$
- Order of progress follows z, then x and y.

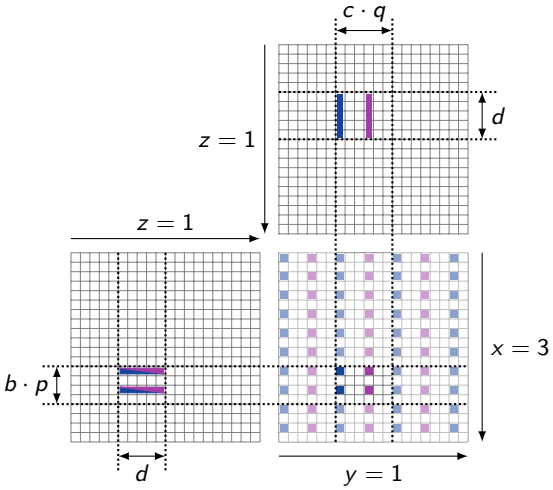


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$

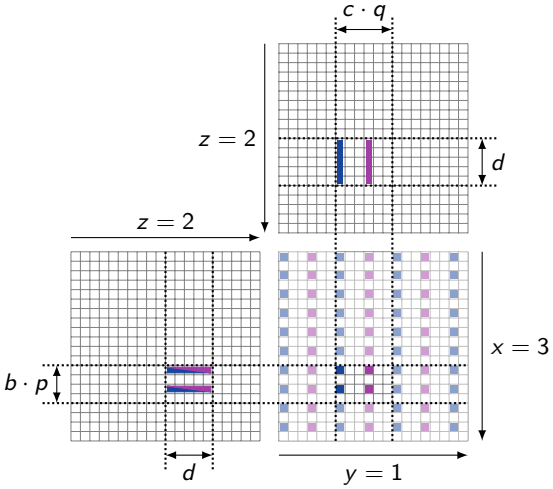


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$

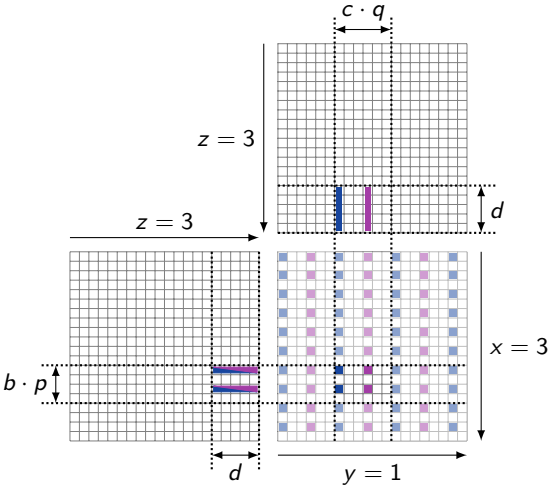


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$

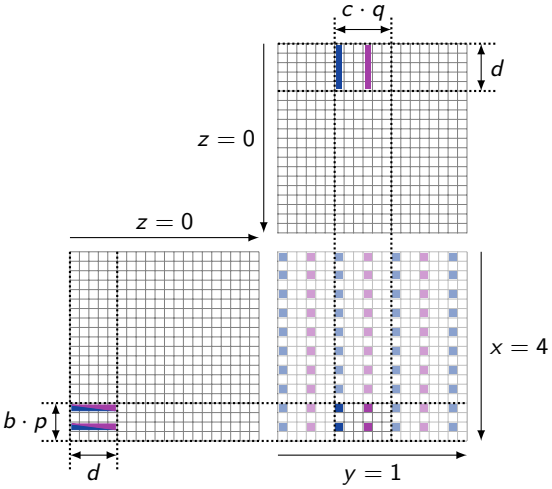


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$

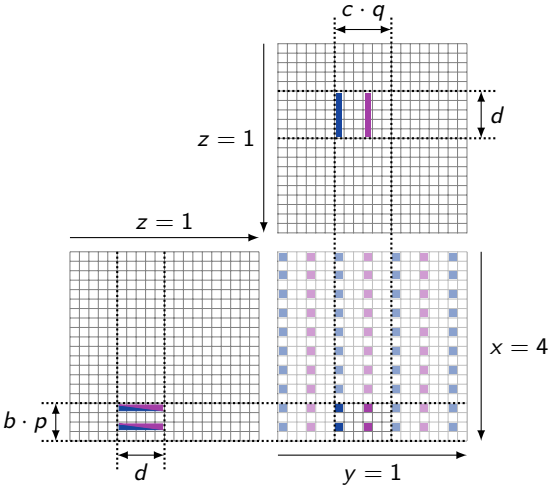


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$

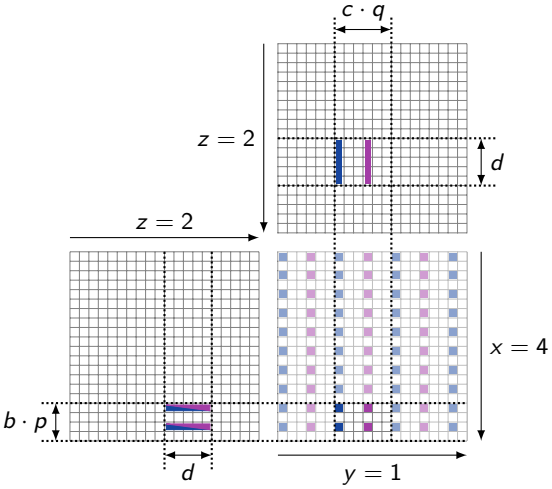


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$

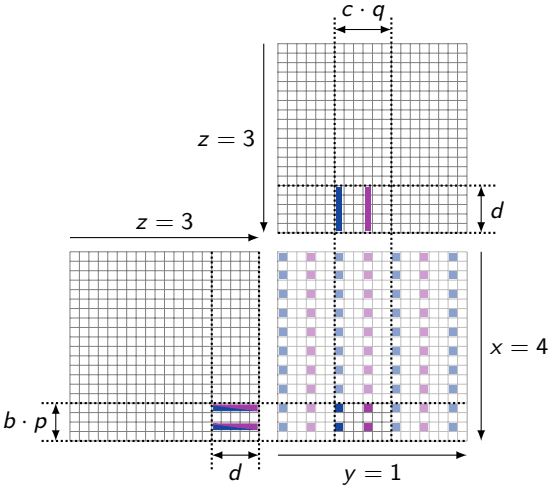


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$

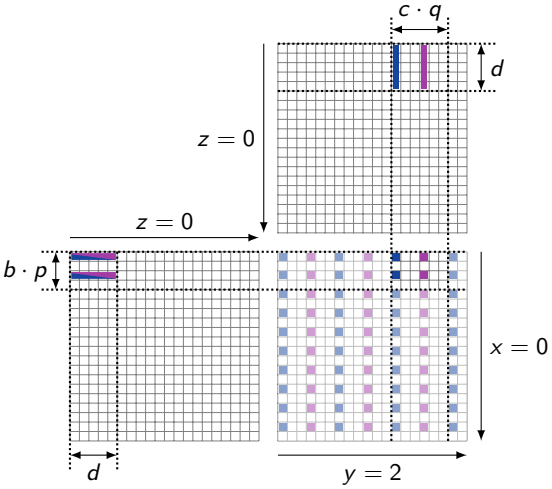


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$

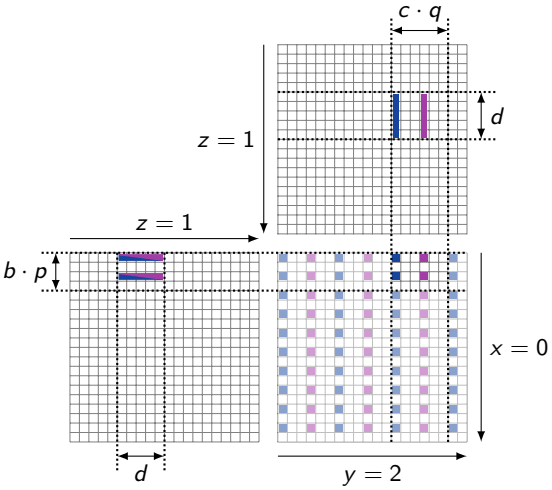


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$

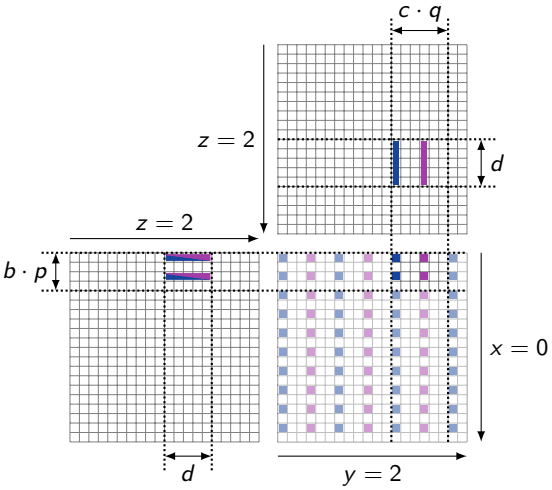


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$

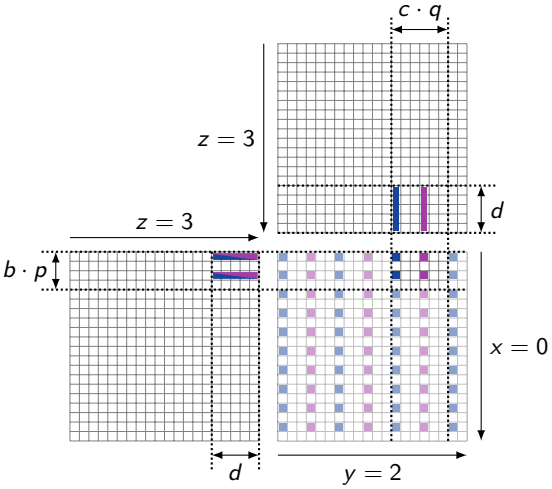


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

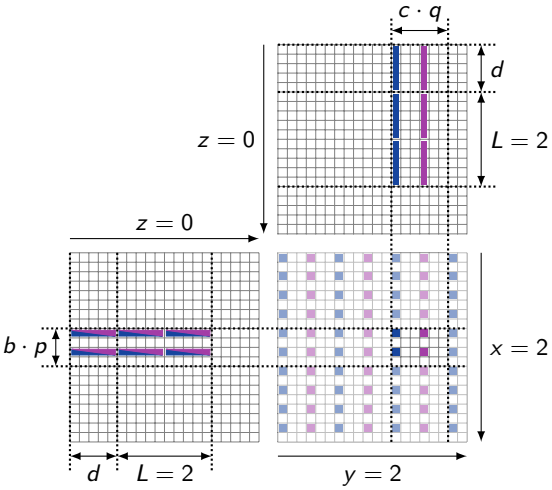
- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$



GEMM Algorithm for GPUs

Machine-level Blocking:

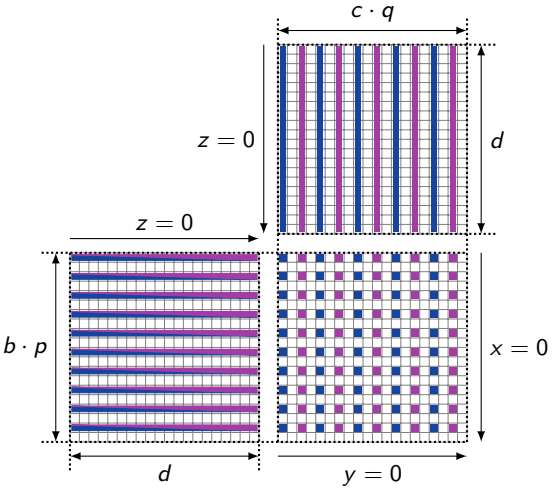
- Main RAM is used as a temporary buffer
- Look ahead parameter L: how many blocks in advance are loaded
- Global Synchronizations prevent a node to progress more than this look ahead step blocks faster than the slowest
  - Prevent overloading a node with download request
  - Control amount of temporary memory



GEMM Algorithm for GPUs – Analysis

$(b, c, d) \times (p, q, g)$ define the number of active tasks.

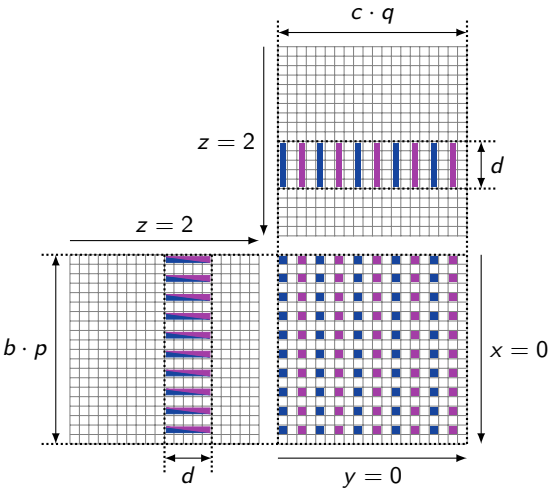
- *Case a)* All local tiles of C, and the required tiles of A and B fit on the memory of the GPUs
- *Case c_1)* All local tiles of C, but only some of the required tiles of A and B fit on the memory of the GPUs
- *Case c_2)* No data can remain stationary



GEMM Algorithm for GPUs – Analysis

$(b, c, d) \times (p, q, g)$ define the number of active tasks.

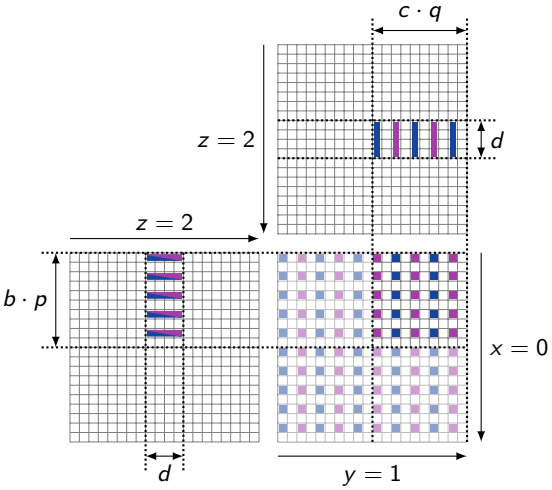
- *Case a)* All local tiles of C, and the required tiles of A and B fit on the memory of the GPUs
- *Case c_1)* All local tiles of C, but only some of the required tiles of A and B fit on the memory of the GPUs
- *Case c_2)* No data can remain stationary



GEMM Algorithm for GPUs – Analysis

$(b, c, d) \times (p, q, g)$ define the number of active tasks.

- *Case a)* All local tiles of C, and the required tiles of A and B fit on the memory of the GPUs
- *Case c_1)* All local tiles of C, but only some of the required tiles of A and B fit on the memory of the GPUs
- *Case c_2)* No data can remain stationary



GEMM Algorithm for GPUs – Analysis

$(b, c, d) \times (p, q, g)$ define the number of active tasks.

- *Case a)* All local tiles of C, and the required tiles of A and B fit on the memory of the GPUs
- *Case c_1)* All local tiles of C, but only some of the required tiles of A and B fit on the memory of the GPUs
- *Case c_2)* No data can remain stationary

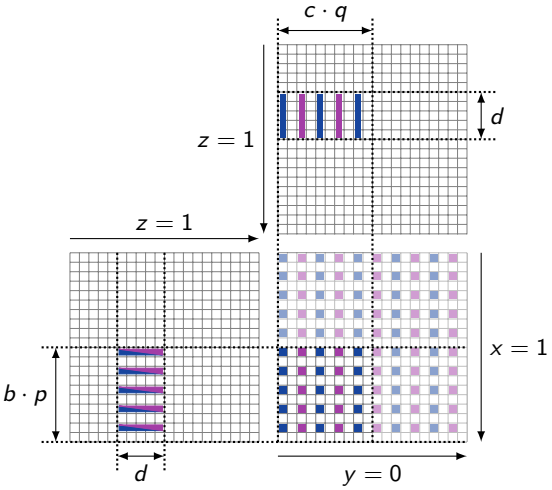
$$Comm_{GPU} = \begin{cases} (a)\ \frac{M_t K_t}{p} + \frac{K_t N_t}{qG} + \frac{M_t N_t}{pqG} \\ \quad \text{if } \frac{M_t K_t}{p} + \frac{K_t N_t}{qG} + \frac{bc}{G} \leq Mem_{GPU} \\ (b)\ \frac{N_t}{cq}\frac{M_t K_t}{p} + \frac{K_t N_t}{qG} + \frac{M_t N_t}{pqG} \\ \quad \text{if } bd + \frac{cK_t}{G} + \frac{bc}{G} \leq Mem_{GPU} \\ (c)\ \frac{N_t}{cq}\frac{M_t K_t}{p} + \frac{M_t}{bp}\frac{K_t N_t}{qG} + \frac{M_t N_t}{pqG} \\ \quad \text{if } bd + \frac{cd}{G} + \frac{bc}{G} \leq Mem_{GPU} \end{cases}$$

GEMM Algorithm for GPUs – Analysis

$(b, c, d) \times (p, q, g)$ define the number of active tasks.

General Strategy:

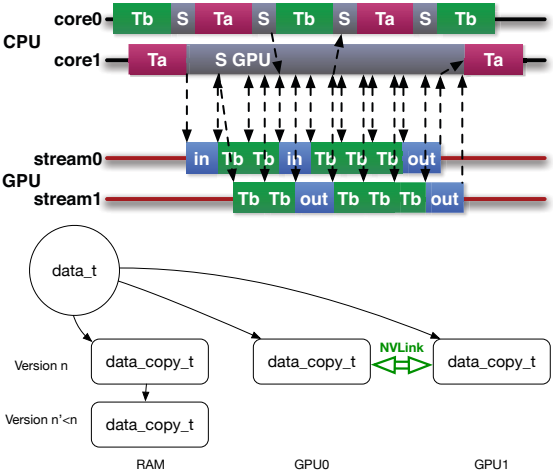
- Select b, c such that
  - the C active set fills at most 75% of GPU memory
  - pb divides M_t and cq divides N_t (best effort)
- Select d such that
  - the C active set plus the required parts of A and B fits the GPU memory
  - d divides K_t (best effort)



Implementation in PaRSEC

PaRSEC Device System and PaRSEC CUDA Device

- PaRSEC abstracts computing resources through *devices*
- The device subsystem has been rewritten as a modular framework
- The CUDA device has been rewritten to support
  - NVLink between GPUs
  - Persistent objects (to support CUBLASv2 and cuDnSolver etc.)



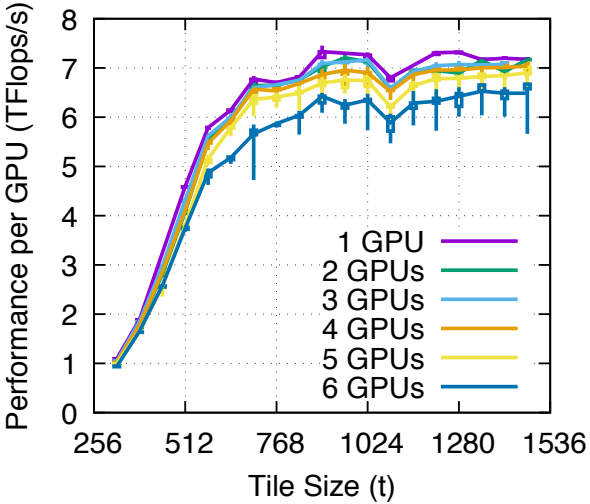
SUMMIT

All Experiments run on Summit (OLCF-4, 9,216 POWER9 22-core CPUs 27,648 Nvidia Tesla V100 GPUs)
PaRSEC Commit e22800d (https://bitbucket.org/icldistcomp/parsec/)
XLC 16.1.1-2, CUDA 9.2.148, Spectrum MPI 10.3.0.0
Double Precision GEMM
All figures show a tuck box plot of 10 measurements
Variability is too low to show on most figures

Single Node Tuning

Single Node Tuning
$t \in [256, 1536], M = K = N = t\lceil \frac{70{,}000}{t} \rceil$
1-3 GPUs: no matrix is stationary
4-6 GPUs: C is statonary

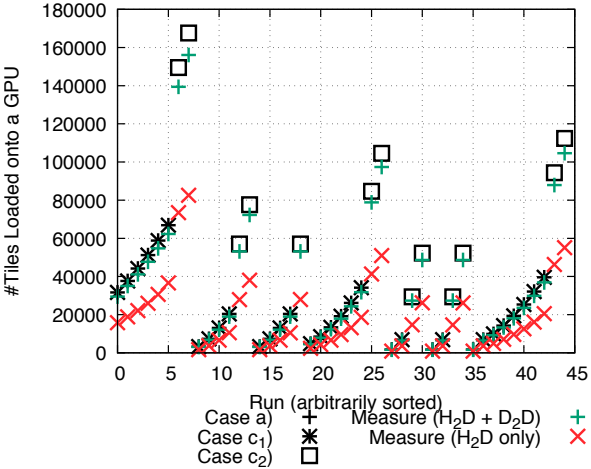
- Strong scaling above 90% efficiency at 1-3 GPUs
- NUMA & Bus sharing effects for 4-6 GPUs
- Minimal Tile Size with Optimal performance: 1024



Bus Activity

PaRSEC profiling tools ⇒ measure # tiles loaded on each GPU, from RAM (H_2D) and from another GPU (D_2D) Compare with $Comm_{GPU}$ for all runs (single node and distributed)

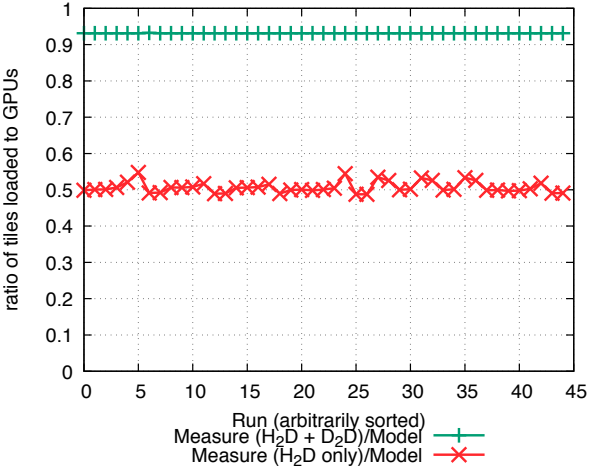
- Overall bus activity consistent with analysis
- About 50% of memory requests are served D_2D
- Independent of deployment case, and distributed or shared memory run



Bus Activity

PaRSEC profiling tools ⇒ measure # tiles loaded on each GPU, from RAM (H_2D) and from another GPU (D_2D) Compare with $Comm_{GPU}$ for all runs (single node and distributed)

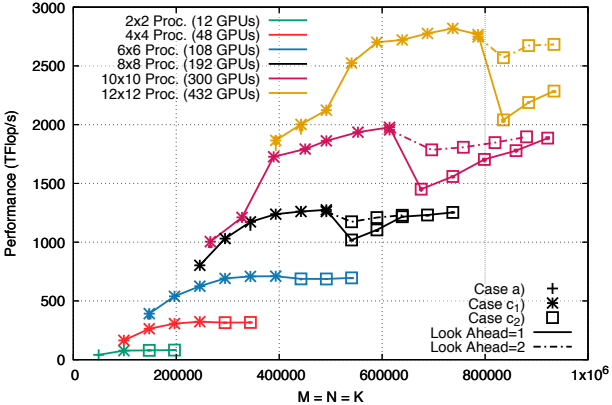
- Overall bus activity consistent with analysis
- About 50% of memory requests are served D_2D
- Independent of deployment case, and distributed or shared memory run



Distributed Performance

Problem and node scaling in distributed
2 Processes/Node: 3 GPU/process (best performance)

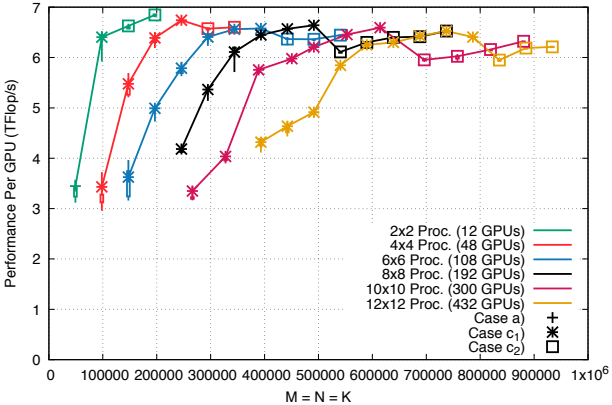
- Up to 108 GPUs (36 proc., 18 nodes), peak reached with constant look ahead
- When the node scale grows, the look ahead needs to be increased to maintain performance
- Preserve high performance despite memory constraints



Distributed Performance

Problem and node scaling in distributed
2 Processes/Node: 3 GPU/process (best performance)

- Up to 108 GPUs (36 proc., 18 nodes), peak reached with constant look ahead
- When the node scale grows, the look ahead needs to be increased to maintain performance
- Preserve high performance despite memory constraints



Conclusion and Future Work

Summary

- Multi-level Blocked Parameterized GEMM targetting parallel machine with multiple GPU/node over PaRSEC
- 3 blocking parameters, 1 look ahead parameter
  - Selection of parameters is driven by memory capacity of the GPUs
- Sustained performance at scale, even when running over the GPU memory capacity

Future Work

- Irregular Dense Tiling
- Irregular Block-Sparse Tiling
- Irregular Tiling with Low-Rank Representation of Block-Sparse Tiles
- Integrate Memory Constraints in GPU Scheduling of the PaRSEC CUDA Device