25+ years of scheduling at ICL

Yves Robert

ENS Lyon
University of Tennessee Knoxville

Dongarra 70 — July 7, 2021

Overview

- A few results in parallel linear algebra and resilience that were achieved at ICL under Jack's guidance
- The story started in 1996. It continues today

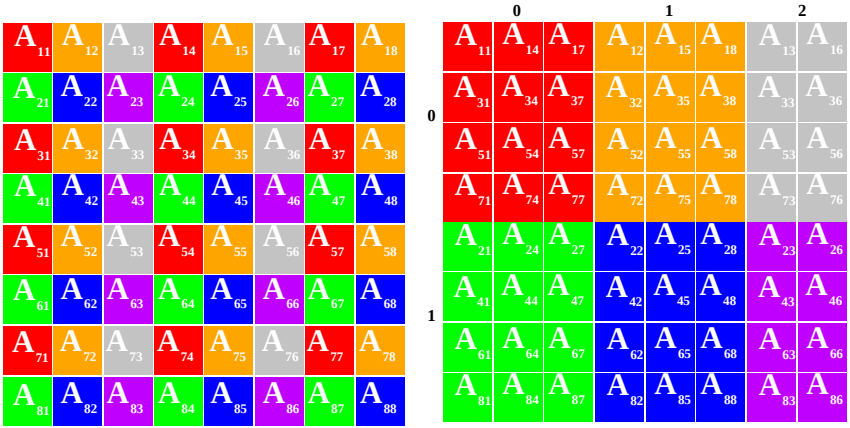
Jack's co-authors according to DBLP

1. Stanimire Tomov (149)
2. Piotr Luszczek (119)
3. George Bosilca (91)
4. Jakub Kurzak (76)
5. Azzam Haidar (69)
6. Peter M. A. Sloot (62)
7. Thomas Hérault (51)
8. Hartwig Anzt (51)
9. Ichitaro Yamazaki (51)
10. Julien Langou (43)
11. Aurelien Bouteiller (43)
12. Yves Robert (41)
13. Graham E. Fagg (40)
14. Mark Gates (32)
15. Ahmad Abdelfattah (32)
16. Hatem Ltaief (31)
17. ...

Outline

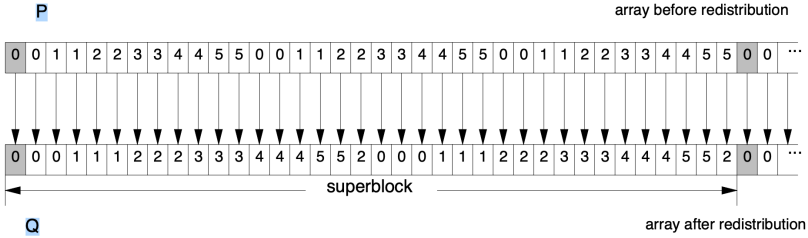
Outline

1996-2000 Redistribution algorithms

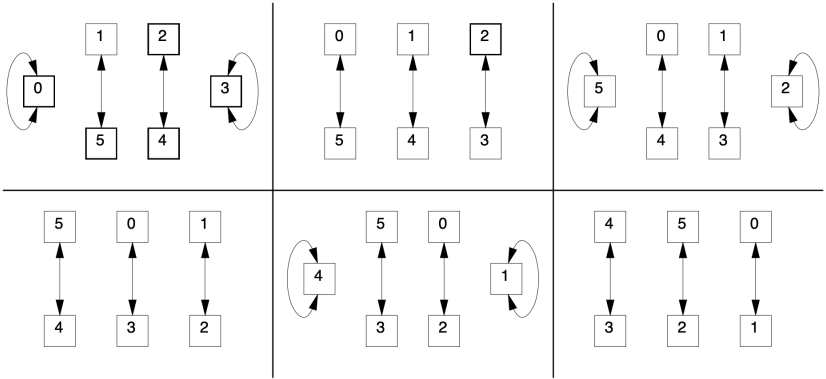


Global (left) and distributed (right) views of matrix

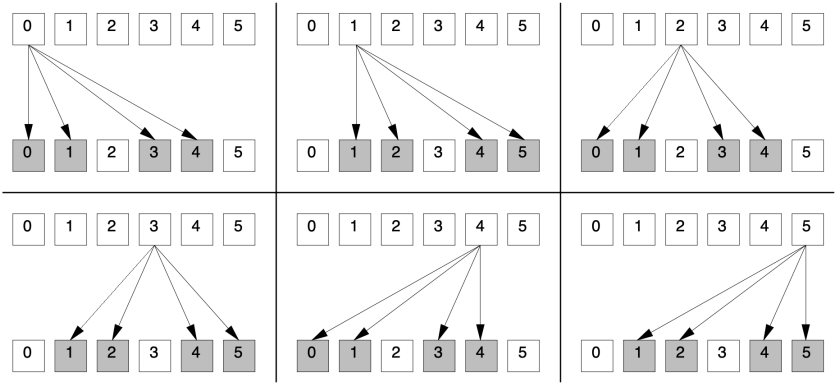
From CYCLIC(2) to CYCLIC(3)



ScaLAPACK caterpillar



From CYCLIC(2) to CYCLIC(3)

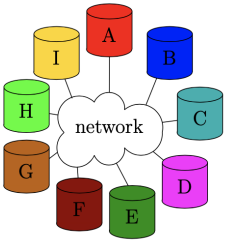


From CYCLIC(2) to CYCLIC(3)

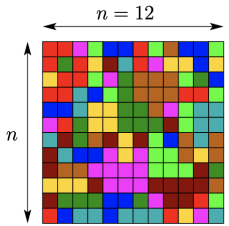
Sender/Recv.	0	1	2	3	4	5	Nbr of msg.
0	2	-	2	-	2	-	3
1	1	1	1	1	1	1	6
2	-	2	-	2	-	2	3
3	2	-	2	-	2	-	3
4	1	1	1	1	1	1	6
5	-	2	-	2	-	2	3
Nbr of msg.	4	4	4	4	4	4	

Message lengths are for a single-slice vector of $L = lcm(Pr, Qs) = 36$ components

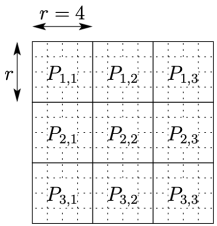
Redistribution followed by a computational kernel



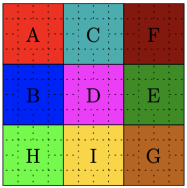
(a) processors holding the data



(b) intial data distribution



(c) target data partition

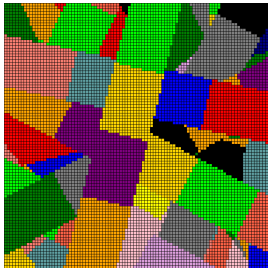


(d) final data distribution

Each color in the data distributions corresponds to a processor, e.g., all red data items reside on processor A.

Redistribution followed by a computational kernel

- Data elements stored on different processors
- Computation kernel must be applied to data

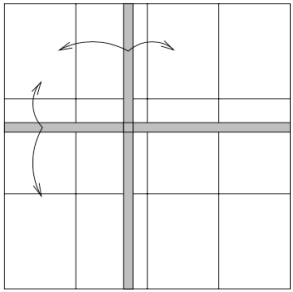


Distribution of a tiled matrice

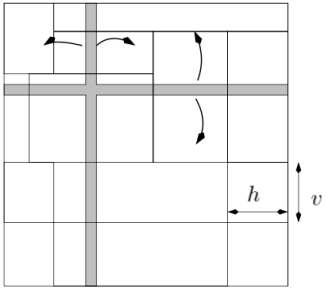
- Initial data distribution may be inefficient for the computation kernel

Outline

1999-2003 Matrix product on heterogeneous platforms

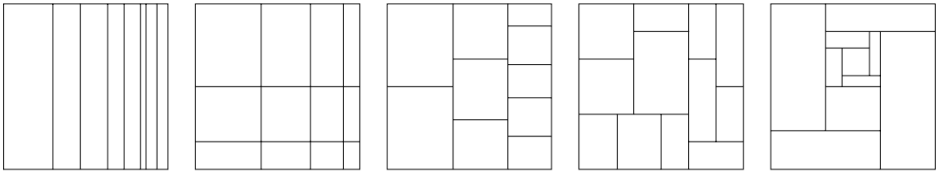


Homogeneous 3×4 grid

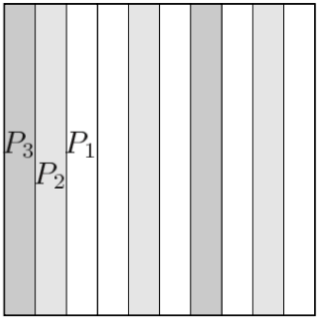


Heterogeneous 3×4 grid

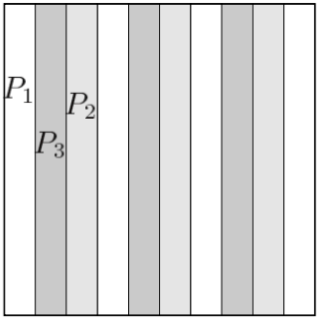
Matrix product on heterogeneous platforms



LU decomposition on heterogeneous platforms



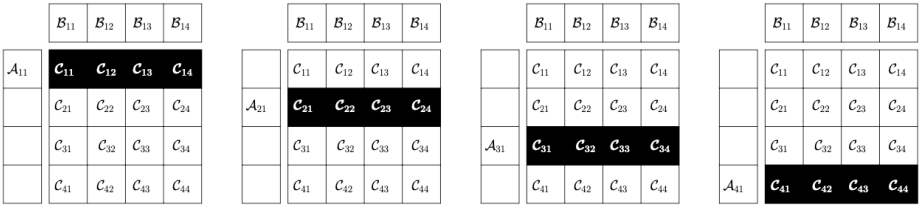
Heterogeneous allocation



Block-cyclic allocation

Cycle-times: $c(P_1) = 3$, $c(P_2) = 5$, $c(P_3) = 8$

2008 Matrix product on master-worker platforms

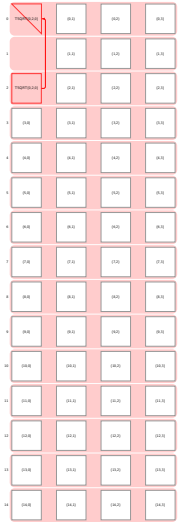


With $1 + \mu + \mu^2$ blocks in memory, improving Toledo's algorithm

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action



2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

2012 QR factorization – Flat Tree in action

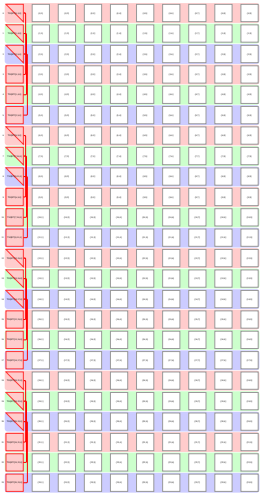
QR factorization – Binary Tree in action

QR factorization – Binary Tree in action

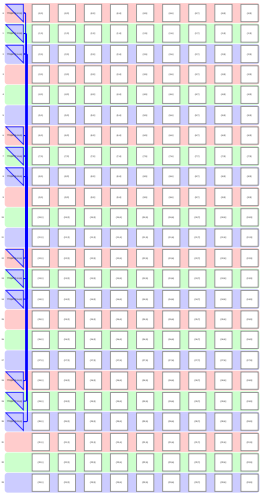
QR factorization – Binary Tree in action

QR factorization – Binary Tree in action

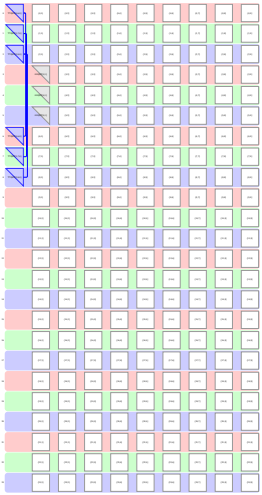
QR factorization – Hierarchical algorithm in action



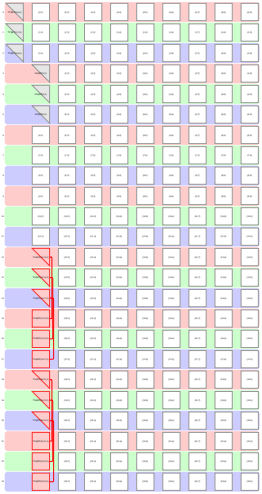
QR factorization – Hierarchical algorithm in action



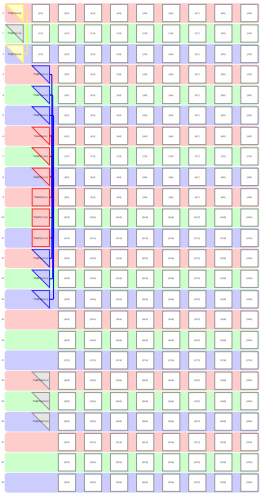
QR factorization – Hierarchical algorithm in action



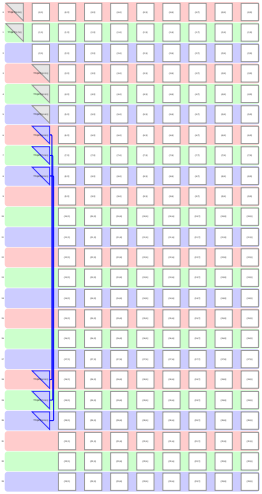
QR factorization – Hierarchical algorithm in action



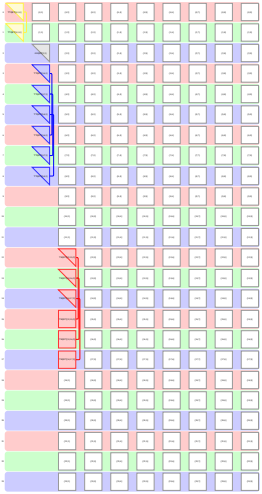
QR factorization – Hierarchical algorithm in action



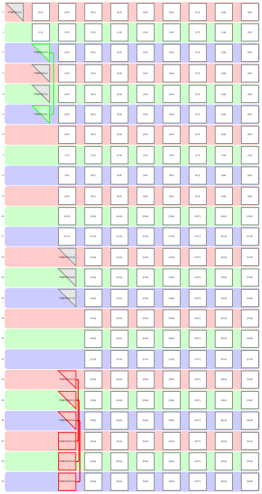
QR factorization – Hierarchical algorithm in action



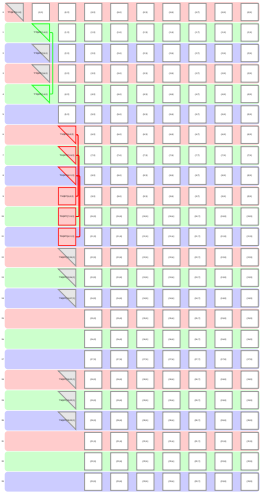
QR factorization – Hierarchical algorithm in action



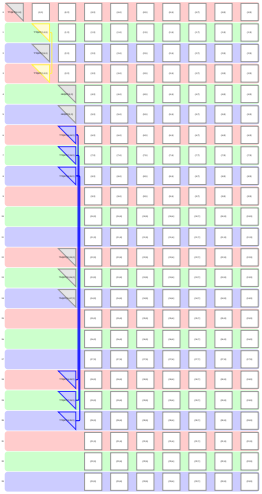
QR factorization – Hierarchical algorithm in action



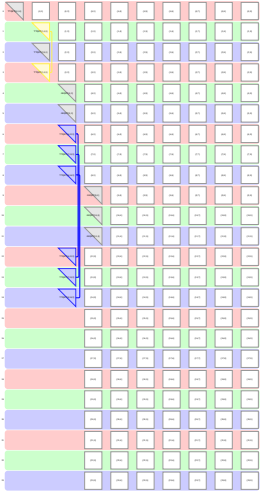
QR factorization – Hierarchical algorithm in action



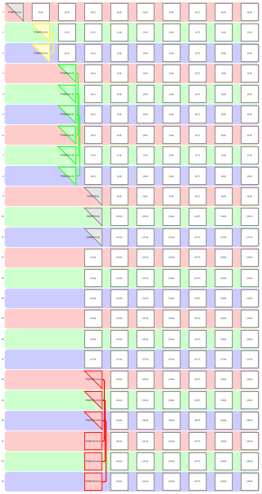
QR factorization – Hierarchical algorithm in action



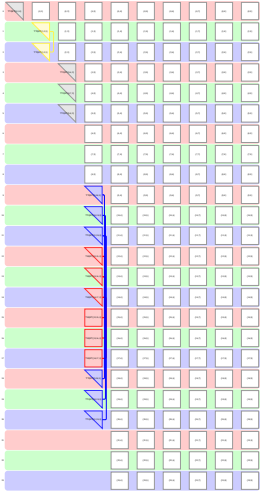
QR factorization – Hierarchical algorithm in action



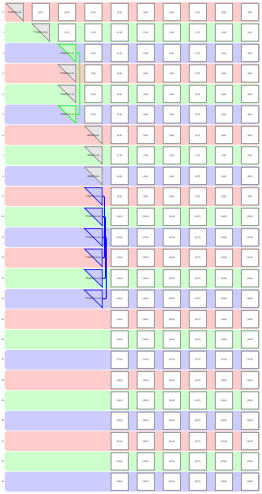
QR factorization – Hierarchical algorithm in action



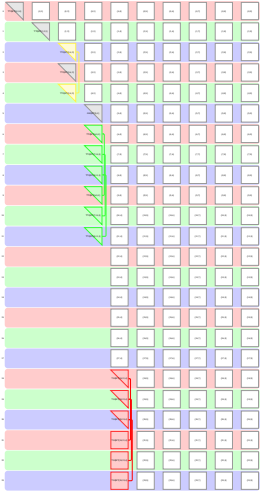
QR factorization – Hierarchical algorithm in action



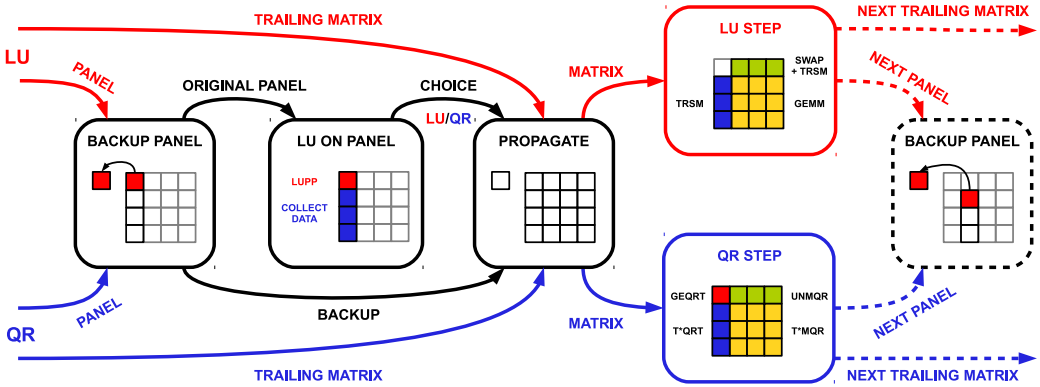
QR factorization – Hierarchical algorithm in action



QR factorization – Hierarchical algorithm in action



2014 Hybrid LU-QR solvers

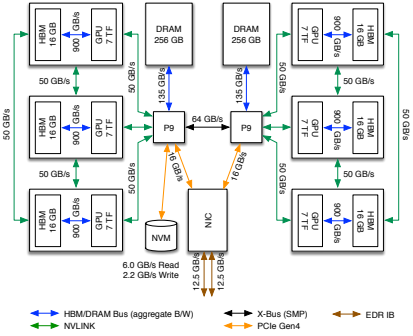


2019-2021 Matrix product on multi-GPU nodes

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

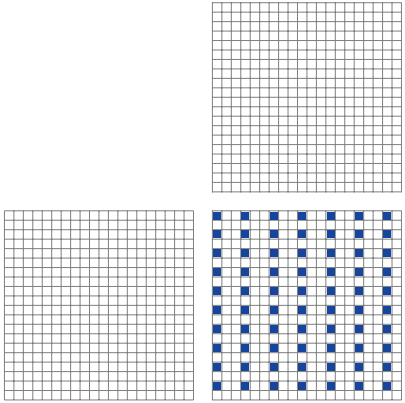


"Summit Interconnection Network Spectrum MPI and InfiniBand", Christopher Zimmer, Dec. 2018

GEMM Algorithm for GPUs

Node-level Task Distribution

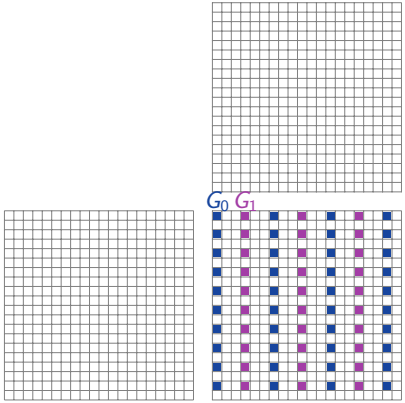
- data affinity
- owner computes heuristic
- 2D block cyclic distribution



GEMM Algorithm for GPUs

GPU-level Task Distribution

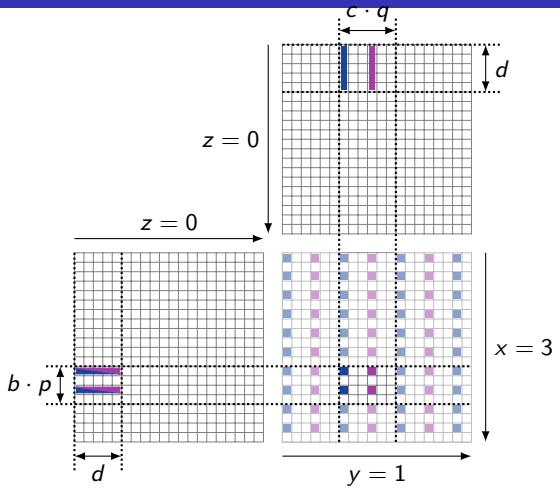
- Fixed, using PaRSEC memory advise API
- Round-robin assignment of tile-columns to the different GPUs



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.

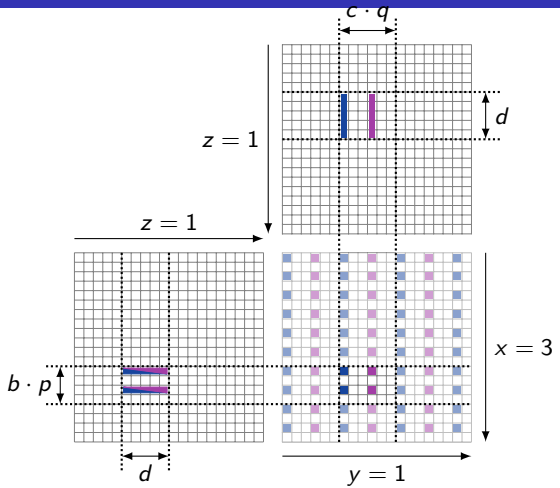


GEMM Algorithm for GPUs

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

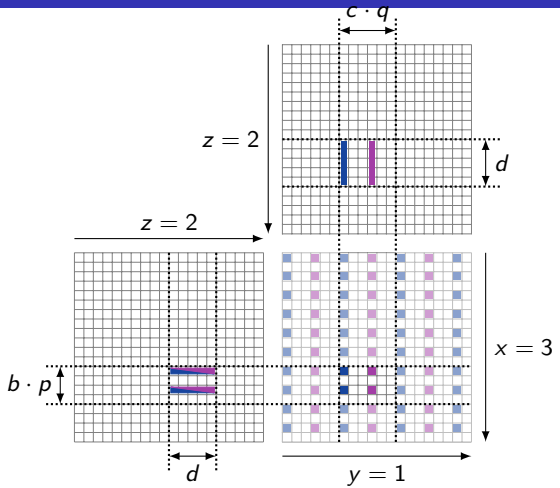


GEMM Algorithm for GPUs

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

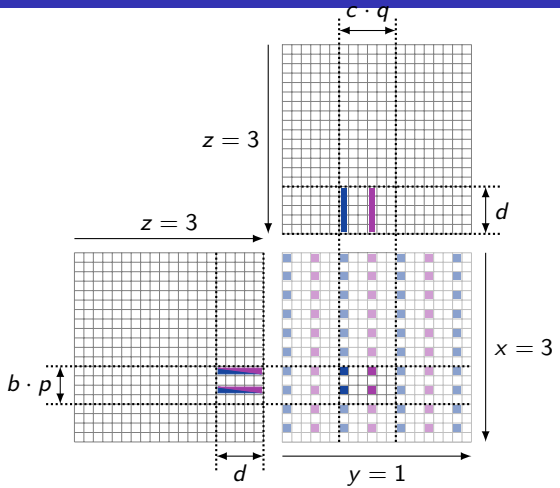


GEMM Algorithm for GPUs

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

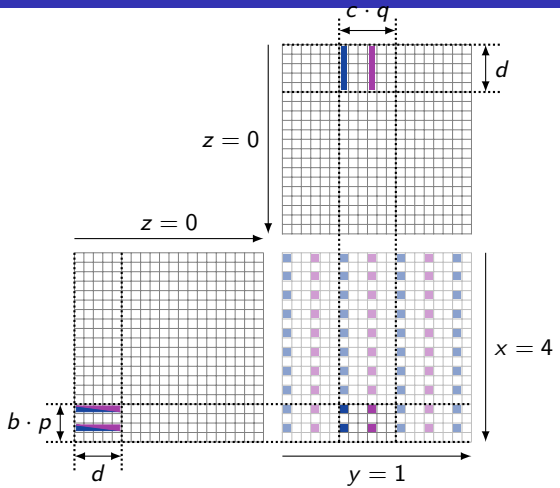


GEMM Algorithm for GPUs

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

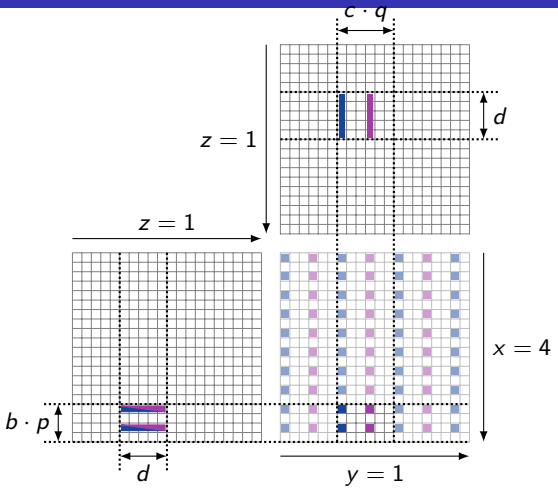


GEMM Algorithm for GPUs

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

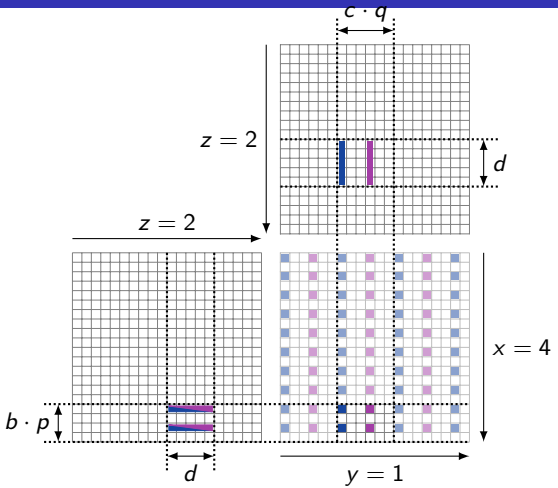


GEMM Algorithm for GPUs

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

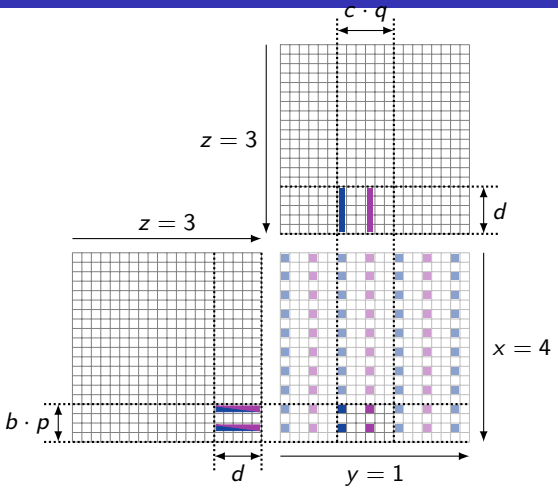


GEMM Algorithm for GPUs

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

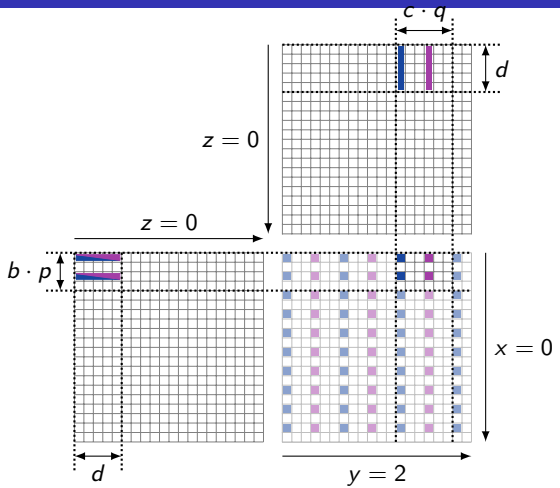


GEMM Algorithm for GPUs

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

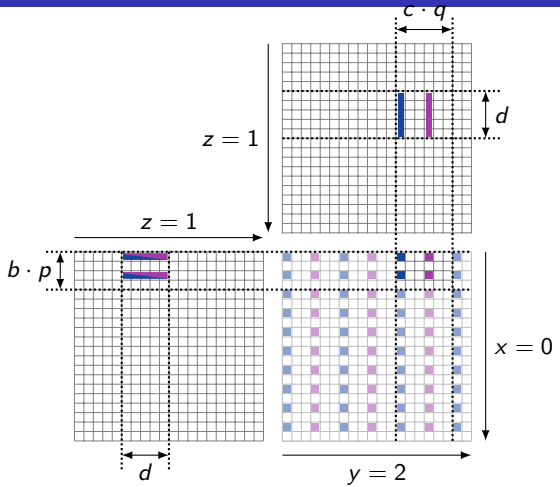


GEMM Algorithm for GPUs

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

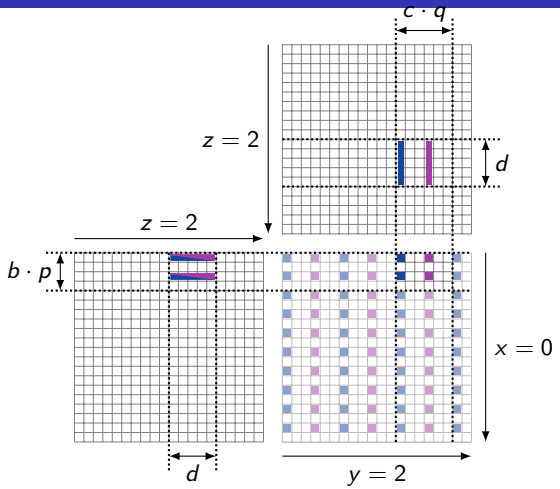


GEMM Algorithm for GPUs

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

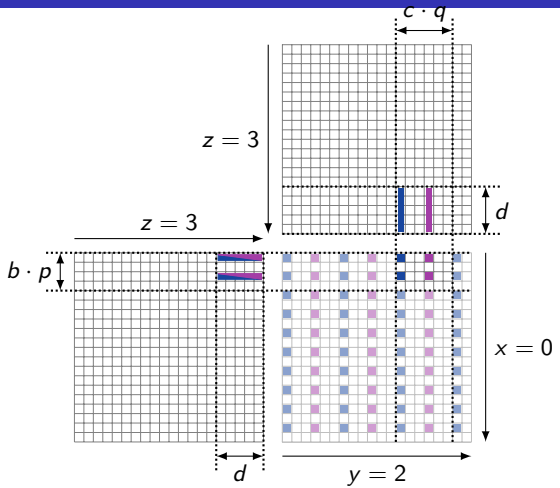


GEMM Algorithm for GPUs

Node-Level Blocking

- Controls the size of the active set on GPUs

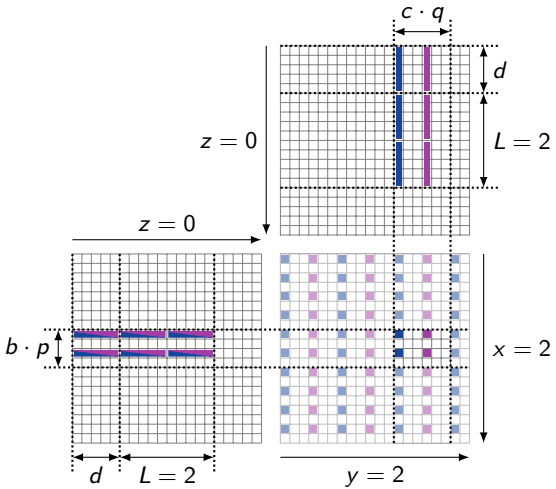
- if $z < MAX_z$:
  $(x, y, z) \longrightarrow (x, y, z + 1)$
- if $z = MAX_z \wedge x < MAX_x$:
  $(x, y, z) \longrightarrow (x + 1, y, 0)$
- if $z = MAX_z \wedge x = MAX_x$:
  $(x, y, z) \longrightarrow (0, y + 1, 0)$



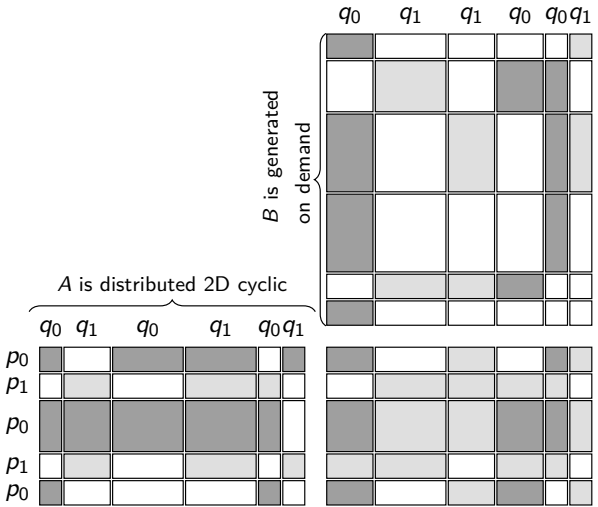
GEMM Algorithm for GPUs

Machine-level Blocking:

- Main RAM is used as a temporary buffer
- Look ahead parameter L: #blocks loaded in advance
- Global synchronizations prevent any node to progress more than L steps than slowest node
  - Prevent overloading a node with download request
  - Control amount of temporary memory



Tensor product

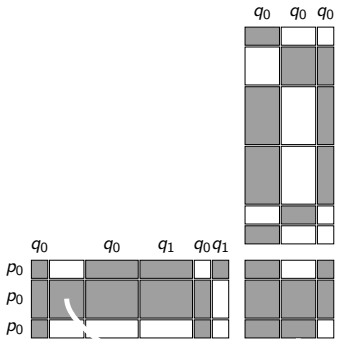


- $M = K >> N$: B is huge in front of A or C.
  - Tiles of B are generated on demand
  - Generating a tile is non trivial

Strategy for B

Tiles of B are generated once, when needed, used then discarded
A single node reads a given tile of B
Load balance flops between nodes

On a node



Distribute B on GPUs

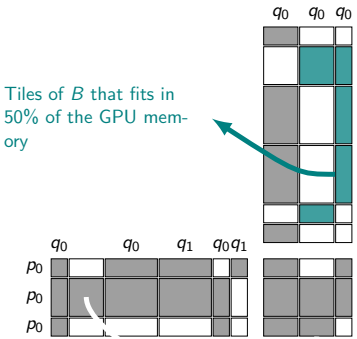
Assign each columns of B to a single GPU on the node
Load balance flops between GPUs

Blocking B for a given GPU

Tiles of B occupy at most 50% of GPU memory

- Sort local columns of B by size
- Split execution in phases
- Greedy algorithm to fill each phase with as many (full) columns of B

On a node



Distribute B on GPUs

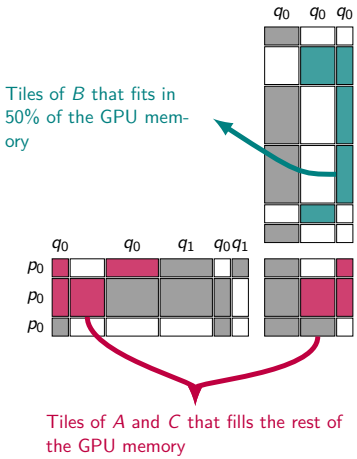
Assign each columns of B to a single GPU on the node
Load balance flops between GPUs

Blocking B for a given GPU

Tiles of B occupy at most 50% of GPU memory

- Sort local columns of B by size
- Split execution in phases
- Greedy algorithm to fill each phase with as many (full) columns of B

On a node



Blocking A and C for a given block phase

Tiles of A and C fill up the rest
Greedy with heuristic: ≈rectangles of A of size d by what fits, and rectangles of C of size d by #columns of B

- Split previous column phases in block-phases
- In each block-phase, assign as many GEMMs as possible
- Such that tiles of A and C fit in the remaining GPU memory
- Looking at tiles of A vertically until at least d rows are selected, then adding only tiles of the same rows, from left to right
- if memory is still available, add a row, iterate.

Outline

2013-2021 Resilience

- In-memory checkpointing
- ABFT and composite strategies
- Failure detection
- Cooperative checkpointing
- Distributed termination
- ...

Outline

1996-1997

- Spent a few months with Jack in IBM ECSEC Rome, 1986-87
- Waited for Bernard Tourancheau and Frédéric Desprez to report on Knoxville ☺
- Well, I liked country music before coming to Tennessee ☺

At Jack's

At Jack's

The dream team

Leaving . . .

A little theorem for the road

Theorem Life is good at ICL

Proof

- By evidence (look at us)
- By enumeration (enough to count the French mafia)

Thanks to Jack and everybody at ICL ☺