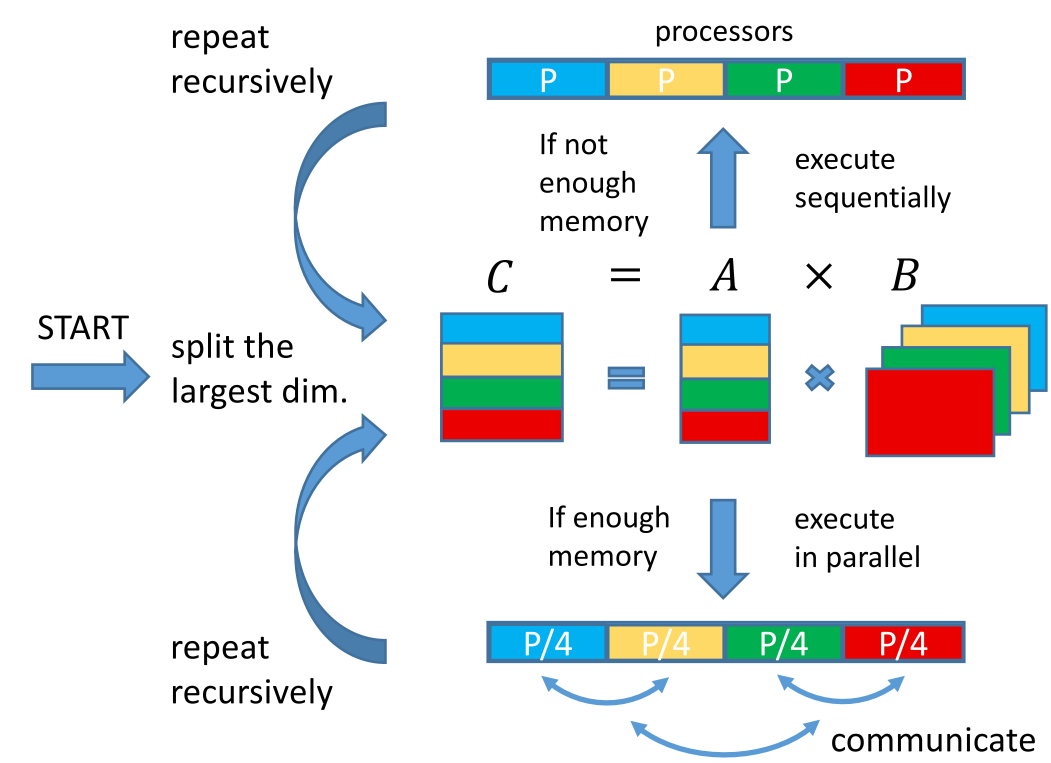
Practical Communication-Optimal Algorithm for Dense Matrix-Matrix Multiplication

Marko Kabić, Raffaele Solcà, Thibault Notargiacomo, Joost VandeVondele

Scientific Software & Libraries Group, Swiss National Supercomputing Center

CARMA Algorithm:

Recursive algorithm developed by James Demmel et al. [1] that is proven to be communication optimal for any matrix sizes and any memory ranges. It adapts to the available memory and enables trading the memory for reducing the expensive communication. Here we consider the *distributed* version of CARMA.



Sequential execution:
All the processors are used to solve all the subproblems sequentially and recursively. No additional memory is used.

Table 1: Asymptotic communication costs for matrix multiplication on P processors with local memory size M and matrix dimensions $d_1 \leq d_2 \leq d_3$ [1].

comm. costs	1 large	2 large	3 large dimensions
Lower Bound	$d_1 d_2$	$\sqrt{\frac{d_1^2 d_2 d_3}{P}}$	$\frac{d_1 d_2 d_3}{P\sqrt{M}} + \left(\frac{d_1 d_2 d_3}{P}\right)^{2/3}$
2D SUMMA	$\sqrt{\frac{d_1^2 d_2 d_3}{P}}$	$\sqrt{\frac{d_1^2 d_2 d_3}{P}}$	$\sqrt{\frac{d_1^2 d_2 d_3}{P}}$
3D SUMMA	$\sqrt{\frac{d_1^2 d_2 d_3}{P}}$	$\sqrt{\frac{d_1^2 d_2 d_3}{P}}$	$\frac{d_1 d_2 d_3}{P\sqrt{M}} + \left(\frac{d_1 d_2 d_3}{P}\right)^{2/3}$
CARMA	$d_1 d_2$	$\sqrt{\frac{d_1^2 d_2 d_3}{P}}$	$\frac{d_1 d_2 d_3}{P\sqrt{M}} + \left(\frac{d_1 d_2 d_3}{P}\right)^{2/3}$

Parallel execution:
Processors are also split into chunks and each chunk solves exactly 1 problem recursively, so that all the subproblems are solved in parallel. Additional memory is necessary since each processor chunk has to own independent copy of the communicated matrix (in this example matrix B).

Main Contributions:

CARMA before:

- **Only powers of 2:** (m, n, k, P) assumed to be powers of 2 or that at each step the number of processor left and the largest dimension share common divisors.
- **Cyclic data base-case layout:** Requires complete data reshuffling after each communication. The corresponding mapper not provided. Compatibility issues with other layouts.
- **Impatient buffers allocations:** every parallel step allocates and deallocates a new buffer. Buffers not reused.
- **Limited division strategies:** Not all division schedules produced correct results.

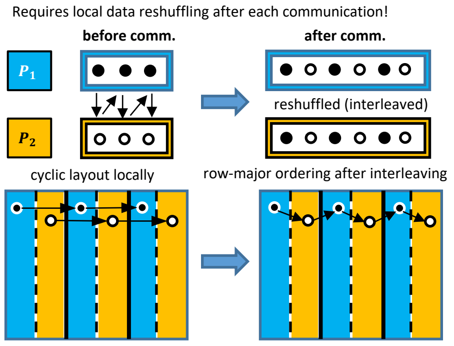
CARMA now:

- **Generalized implementation:** Works for any (m, n, k, P)!
- **Blocked data base-case layout:** Requires local data reshuffling only in some case and only on the level of blocks, instead of single elements.
- **Less memory, more performance:** Buffers carefully allocated and reused throughout the algorithm. Using ~25% less memory in total.
- **Any division strategy available**

New Data Layout:

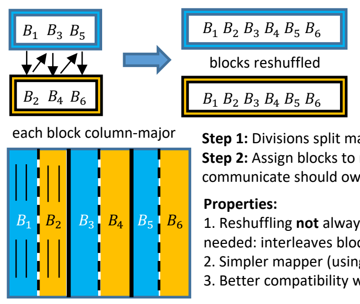
Cyclic Layout (before)

Requires local data reshuffling after each communication!



Blocked Layout (now)

Idea: knowing the communication schedule, assign blocks to processors to minimize the local data reshufflings.



When needed, reshuffle blocks, rather than single elements.

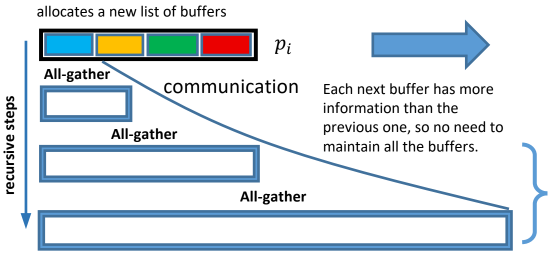
Step 1: Divisions split matrix into blocks
Step 2: Assign blocks to ranks: processors who communicate should own consecutive blocks.

Properties:
1. Reshuffling **not** always needed and if needed: interleaves blocks, not elements.
2. Simpler mapper (using interval trees).
3. Better compatibility with other layouts.

Buffers Reuse:

Our implementation decreases the total amount of memory used and the total number of buffers allocated. All the buffers are allocated just once and are then being reused throughout the application.

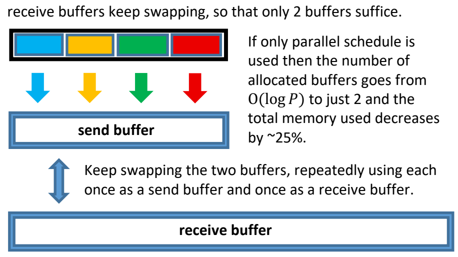
Before: each sequential subproblem allocates a new list of buffers



Each next buffer has more information than the previous one, so no need to maintain all the buffers.

Evolution of a buffer holding local data of a single matrix

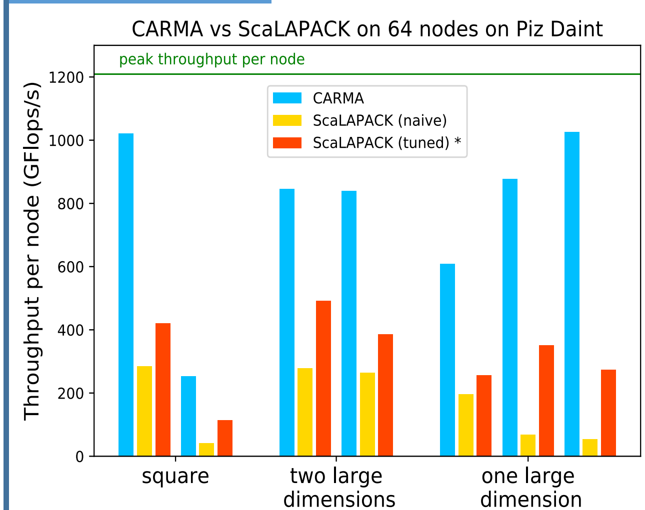
Now: All subproblems reuse the same buffers. Moreover, send and receive buffers keep swapping, so that only 2 buffers suffice.



If only parallel schedule is used then the number of allocated buffers goes from $O(\log P)$ to just 2 and the total memory used decreases by ~25%.

Keep swapping the two buffers, repeatedly using each once as a send buffer and once as a receive buffer.

Performance:



Problem size

(tuned) $*$ in the plot means the configuration (e.g. #ranks/node, #cores/rank,...) which maximizes the TPS was chosen.

	square		two large		one large dimension		
m	64k	8k	64k	64k	64k	8704	17408
n	64k	8k	64k	8k	8k	8704	17408
k	64k	8k	8k	64k	8k	933888	3735552

Configuration

Piz Daint (multicore)	Cray XC40: 2x18-core Broadwell per node
Number of nodes	64 nodes
MPI implementation	Cray MPICH
ScaLAPACK implementation	Intel MKL

References:

[1] Demmel, James, et al. "Communication-optimal parallel recursive rectangular matrix multiplication." *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on.* IEEE, 2013.

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Contact us:

source github.com/eth-cscs/CARMA
email Marko Kabić (marko.kabic@cscs.ch)