JÜLICH
FORSCHUNGSZENTRUM

Efficient parallel implementation of the ChASE library on distributed CPU-GPU architectures

JLESC, Kobe, December 1st | **E. Di Napoli**, A. Schleife

Outline

Motivation

The eigensolver: Chebyshev Accelerated Subspace Iteration (ChASE)

Distributed CPU/GPU: a simple and efficient parallelization

Experimental tests and outlook

Topic

Motivation

The eigensolver: Chebyshev Accelerated Subspace Iteration (ChASE)

Distributed CPU/GPU: a simple and efficient parallelization

Experimental tests and outlook

Two-particle excitation and Bethe-Salpeter Eq. (BSE)

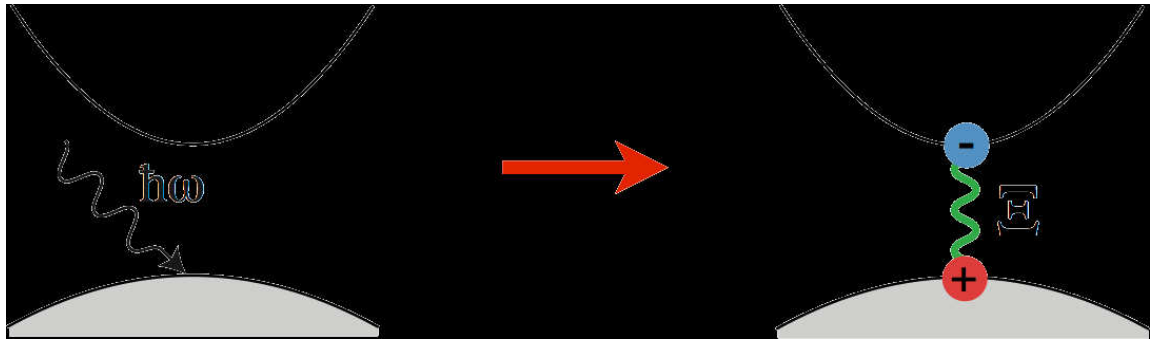


Figure: Optical absorption

- Electron from valence band excited into conduction band
- Electron-hole attraction (screened Coulomb potential) Ξ
- Macroscopic dielectric function: Local-field effects

Bethe-Salpeter equation for optical polarization

$$P = P_0 + P_0 \left(2\bar{v} - \Xi\right) P$$

BSE is an eigenvalue equation

Eigenvalue equation

$$H(\mathbf{k})\, X(\mathbf{k}) = E\, X(\mathbf{k})$$

Excitonic effects: Solution of the Bethe-Salpeter equation

- Leads to dense eigenvalue problem (*excitonic Hamiltonian*)
- Nested **k**-point grids for different energy ranges
- Computationally challenging: LARGE matrices, Size ~ $\mathcal{O}(100k)$ (e.g. $n = 360k$ for In_2O_3 i.e. up to about 1 TB)
- Size of matrices are inversely proportional to number of **k**-points and energy cut-off
- Excellent description of the optical properties of the oxides

$\Longrightarrow$ Predictive power (e.g. for In2O3, Ga2O3, . . .)

BSE is an eigenvalue equation

Eigenvalue equation

$$H(\mathbf{k})\, X(\mathbf{k}) = E\, X(\mathbf{k})$$

- Needed: $\mathcal{O}(100)$ lowest eigenvalues (exciton binding energies);
- Current eigensolver is based on Kalkreuther-Simma Conjugate-Gradient (KSCG) algorithm;
- Parallelized for distributed memory (MPI);
- Needed: increase parallel efficiency, scalability and performance;
- Desired: exploit many-core platforms (e.g. GPUs on Blue Waters)

A computational example

k-points	size (n)	nnz	CPU time	Memory	nodes
10945	82499	$6.8\ 10^9$	1.5 hours	50.7 GiB	8
12713	96399	$9.3\ 10^9$	2 hours	69.2 GiB	8
16299	124281	$1.5\ 10^{10}$	2 hours	115.1 GiB	16
25367	195281	$3.8\ 10^{10}$	5.5 hours	284.1 GiB	16

A Convergence test for exciton-binding energy w.r.t. number of k-points

- only four atoms in a unit cell;
- calculations run on BlueWater;
- cost increases enormously as k-points number increases;
- however, we just barely achieve convergence.

Topic

Motivation

The eigensolver: Chebyshev Accelerated Subspace Iteration (ChASE)

Distributed CPU/GPU: a simple and efficient parallelization

Experimental tests and outlook

Eigenproblems and Eigesolvers

$$AX = X\Lambda \quad ; \quad X = (x_1, \ldots, x_n) \quad \Lambda = (\lambda_1, \ldots, \lambda_n)$$

Direct solvers.

Iterative solvers.

Eigenproblems and Eigesolvers

$$AX = X\Lambda \quad ; \quad X = (x_1, \ldots, x_n) \quad \Lambda = (\lambda_1, \ldots, \lambda_n)$$

Direct solvers.

Iterative solvers.

Eigenproblems and Eigesolvers

$$AX = X\Lambda \quad ; \quad X = (x_1, \ldots, x_n) \quad \Lambda = (\lambda_1, \ldots, \lambda_n)$$

Direct solvers.

Iterative solvers.

$$|\lambda_1| > |\lambda_2| > |\lambda_3| > \ldots$$

$$Ax_j = \lambda_j x_j$$

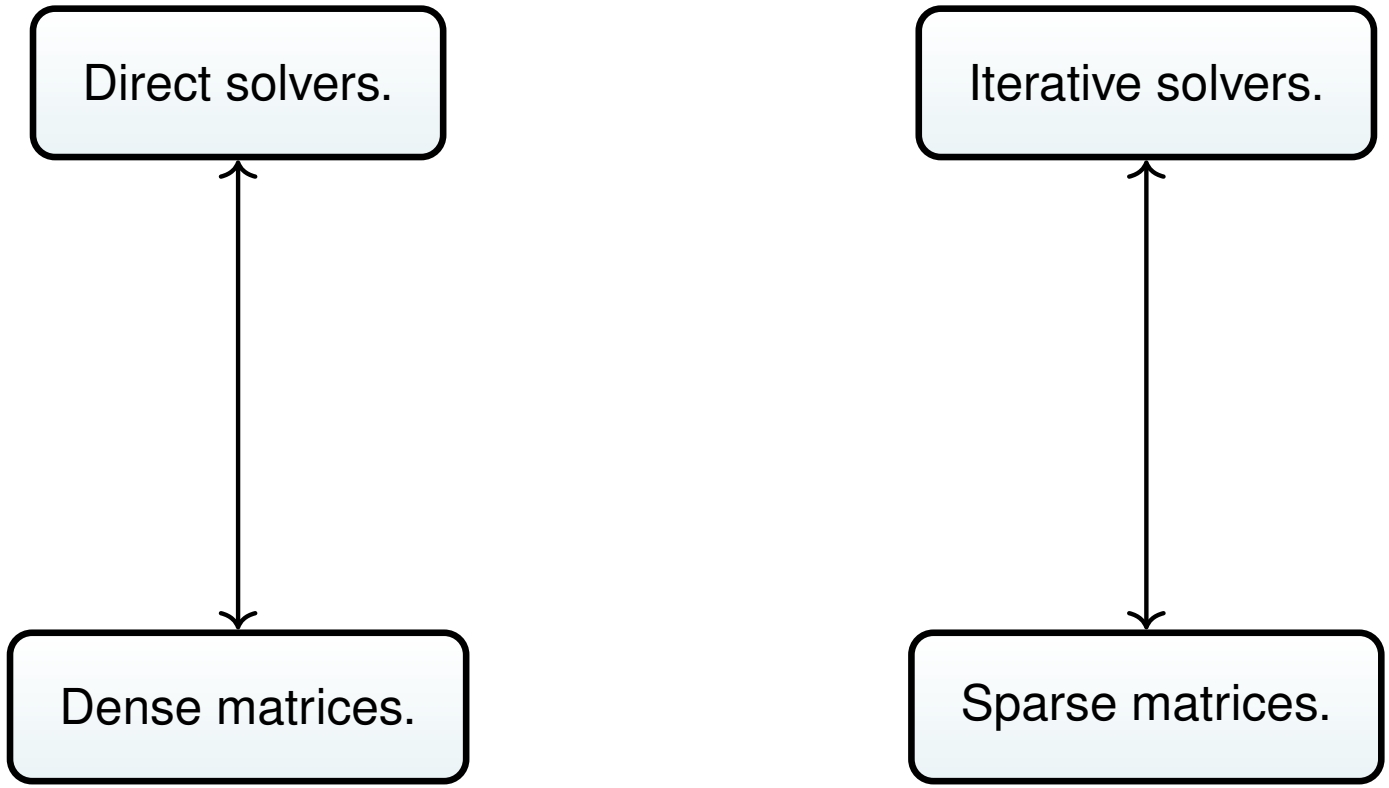
$$v = \sum_j \gamma_j x_j$$

$$Av = \sum_j \lambda_j \gamma_j x_j \Rightarrow A^k v = \sum_j \lambda_j^k \gamma_j x_j = \lambda_1 \left[x_1 + \sum_{j \geq 2} \frac{\lambda_j}{\lambda_1} x_j \right]$$

Rate of convergence $\rightarrow$ magnitude of $\left| \frac{\lambda_j}{\lambda_1} \right|$

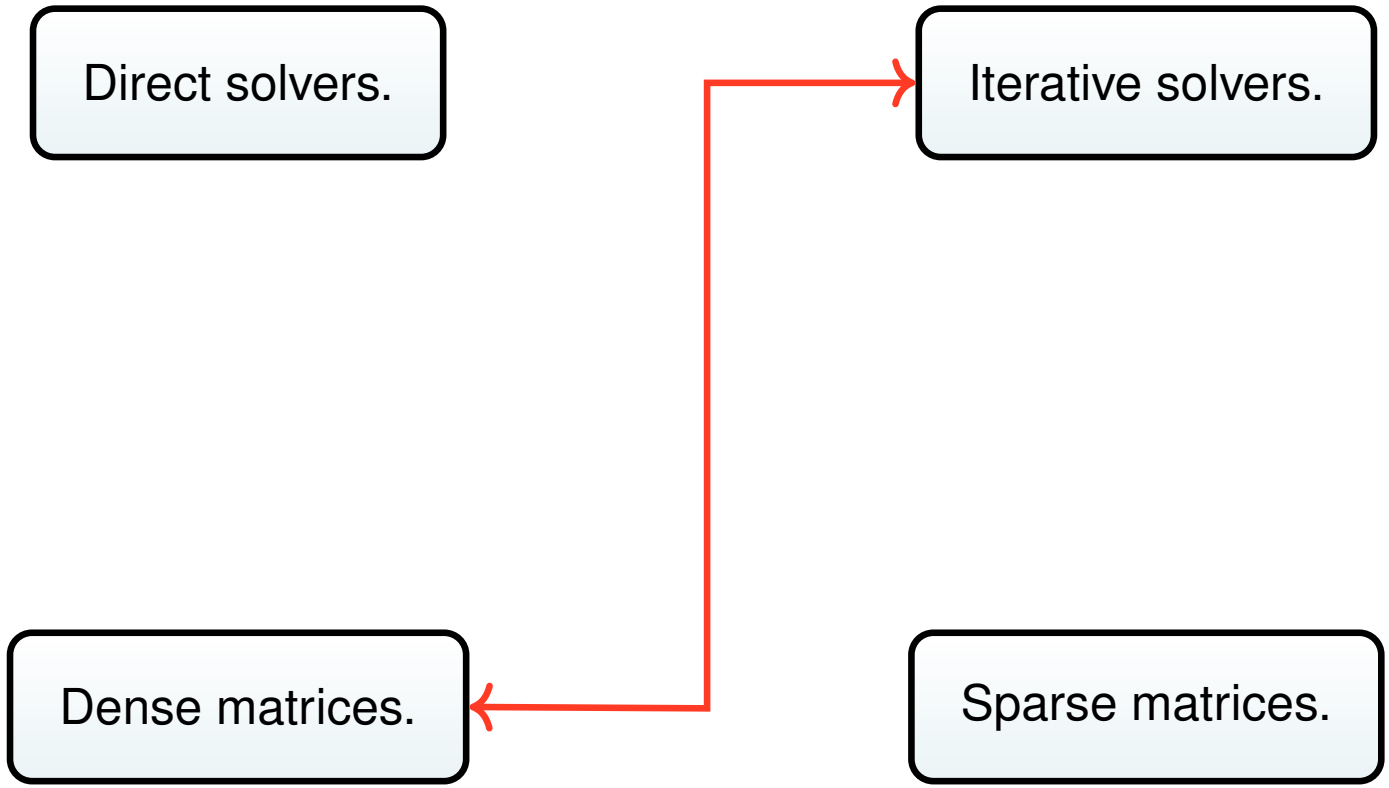
Eigenproblems and Eigesolvers

$$AX = X\Lambda \quad ; \quad X = (x_1, \ldots, x_n) \quad \Lambda = (\lambda_1, \ldots, \lambda_n)$$



Eigenproblems and Eigesolvers

$$AX = X\Lambda \quad ; \quad X = (x_1, \ldots, x_n) \quad \Lambda = (\lambda_1, \ldots, \lambda_n)$$



ChASE

Subspace iterations with Rayleigh-Ritz

- Choose an initial system of vectors $X^0 = [x_1, \ldots, x_m]$.
- Perform successive multiplication $X^k := AX^{k-1}$.
- Every once in a while orthonormalize column-vectors in X^k.
- Compute Rayleigh-Ritz quotient
- Solve reduced problem

ChASE Eigensolver

- Substitute $A^k X \longrightarrow p(A)X$.
- Chebyshev filter improves the rate of convergence.

ChASE pseudocode

INPUT: Hamiltonian H, TOL, DEG — OPTIONAL: approximate eigenvectors Z_0, extreme eigenvalues $\{\lambda_1, \lambda_{\text{NEV}}\}$.
OUTPUT: NEV wanted eigenpairs (Λ, W).

1. *Lanczos DoS step.* Identify the bounds for the **eigenspectrum interval** corresponding to the wanted eigenspace.

REPEAT UNTIL CONVERGENCE:

2. *Chebyshev filter.* **Filter** a block of vectors $W \longleftarrow Z_0$.
3. Re-orthogonalize the vectors outputted by the filter; $W = QR$.
4. Compute the **Rayleigh quotient** $G = Q^{\dagger} H Q$.
5. Compute the primitive Ritz pairs (Λ, Y) by solving for $GY = Y\Lambda$.
6. Compute the approximate Ritz pairs $(\Lambda, W \leftarrow QY)$.
7. *Check* which one among the Ritz vectors *converged*.
8. *Deflate* and *lock* the converged vectors.

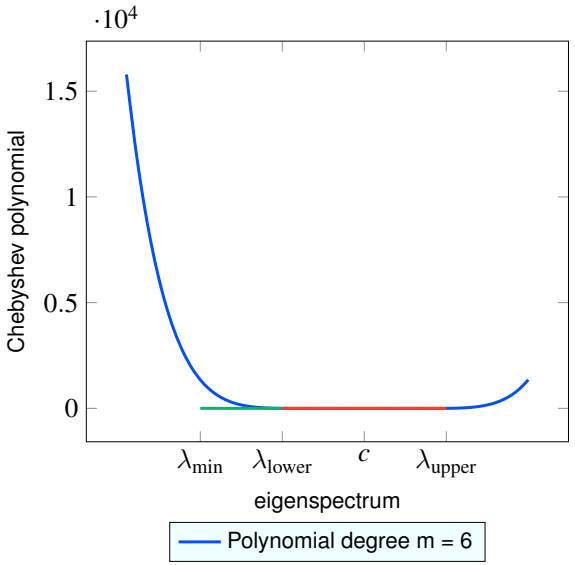
END REPEAT

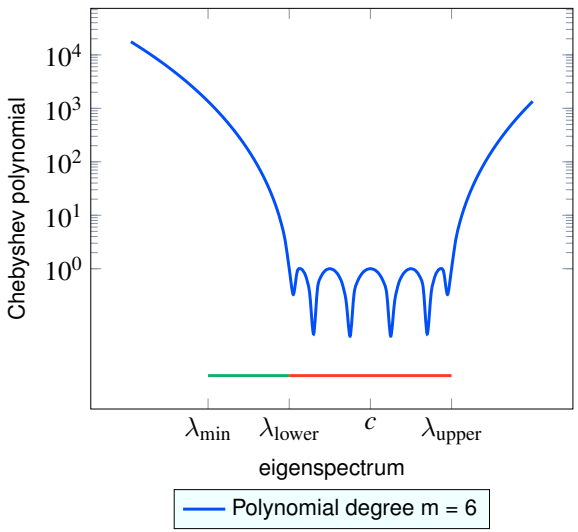
The core of the algorithm: Chebyshev filter

Chebyshev polynomials

A generic vector $v = \sum_{i=1}^{n} s_i x_i$ is very quickly aligned in the direction of the eigenvector corresponding to the extremal eigenvalue λ_1

$$
\begin{aligned}
v^m = p_m(H)v \quad &= \quad \sum_{i=1}^{n} s_i\, p_m(H) x_i = \sum_{i=1}^{n} s_i\, p_m(\lambda_i) x_i \\
&= \quad s_1 x_1 + \sum_{i=2}^{n} s_i \frac{C_m\left(\frac{\lambda_i - c}{e}\right)}{C_m\left(\frac{\lambda_1 - c}{e}\right)} x_i \quad \sim \quad \boxed{s_1 x_1}
\end{aligned}
$$





The core of the algorithm: Chebyshev filter

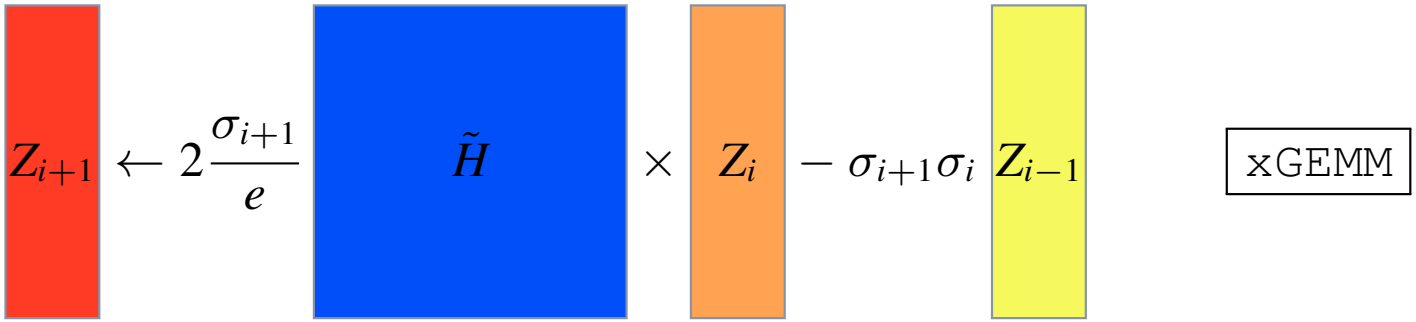
In practice

Three-terms recurrence relation

$$C_{m+1}(t) = 2xC_m(t) - C_{m-1}(t); \qquad m \in \mathbb{N}, \quad C_0(t) = 1, \quad C_1(t) = x$$

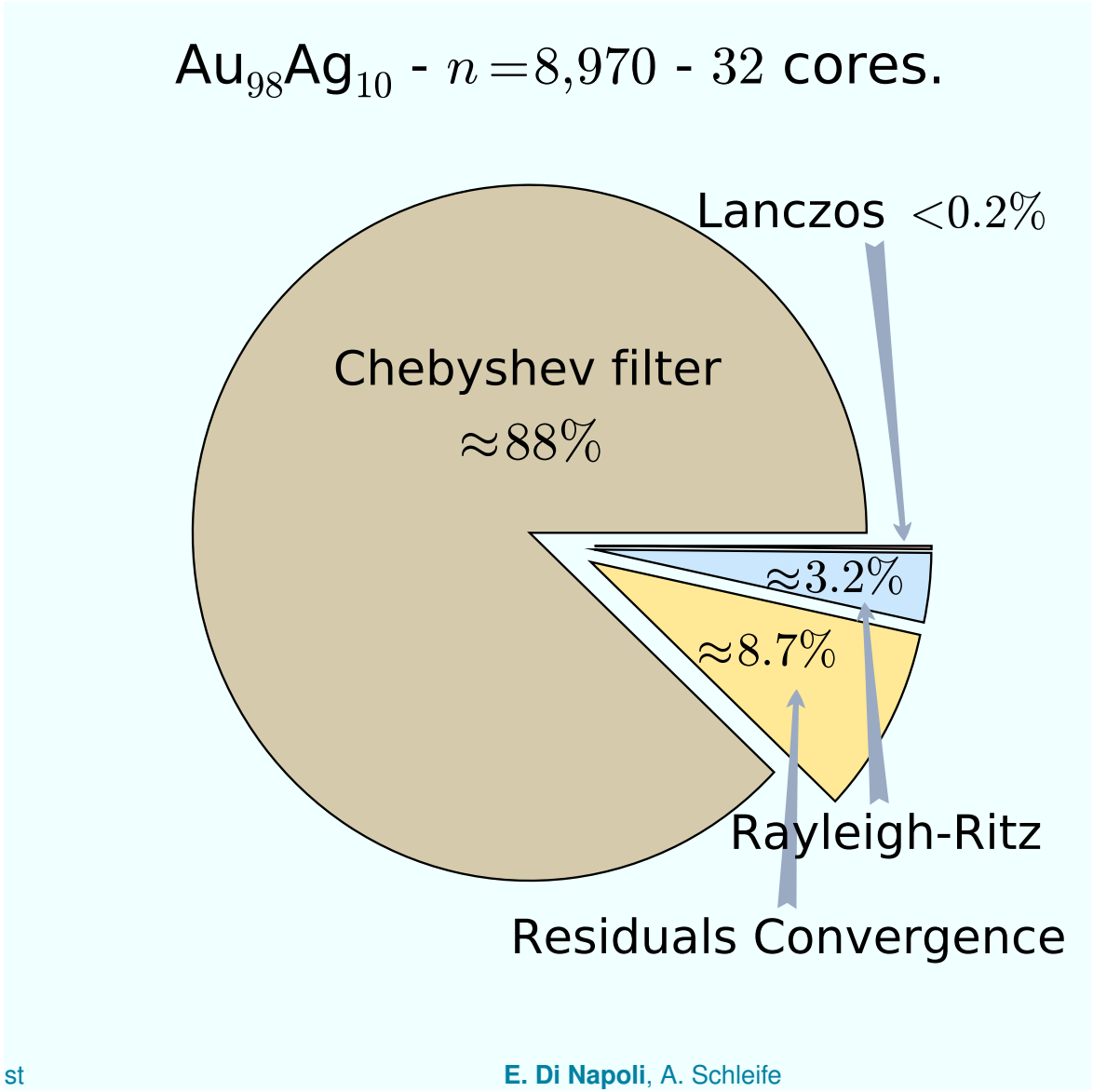
$$Z_m \doteq p_m(\tilde{H})\, Z_0 \quad \text{with} \quad \tilde{H} = H - cI_n$$

FOR: $i = 1 \rightarrow \text{DEG} - 1$



END FOR.

ChASE time profile



Topic

Motivation

The eigensolver: Chebyshev Accelerated Subspace Iteration (ChASE)

Distributed CPU/GPU: a simple and efficient parallelization

Experimental tests and outlook

Parallelization of the Chebyshev filter

Targets

- A simple and efficient scheme for data distribution and communication using MPI
- An economic paradigm that successively performs

$$C \leftarrow \alpha AB + \beta C, \quad B \leftarrow \alpha AC + \beta B. \tag{1}$$

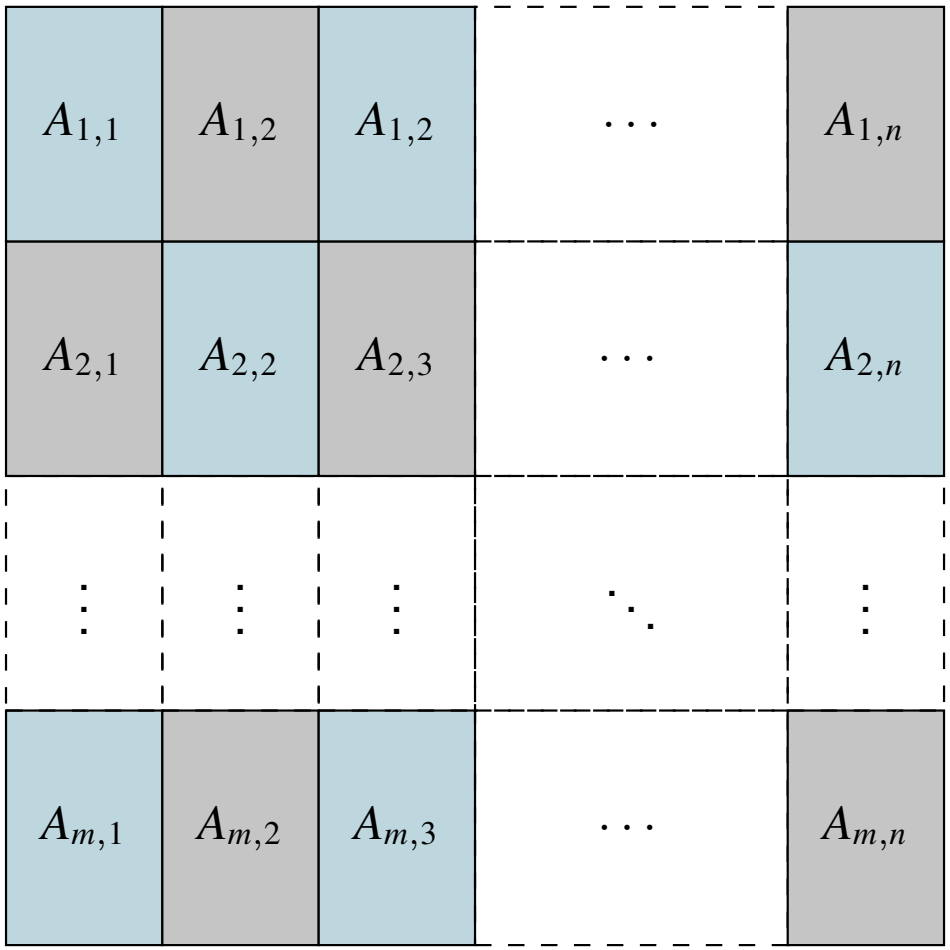
  using CuBLAS on multiple GPUs

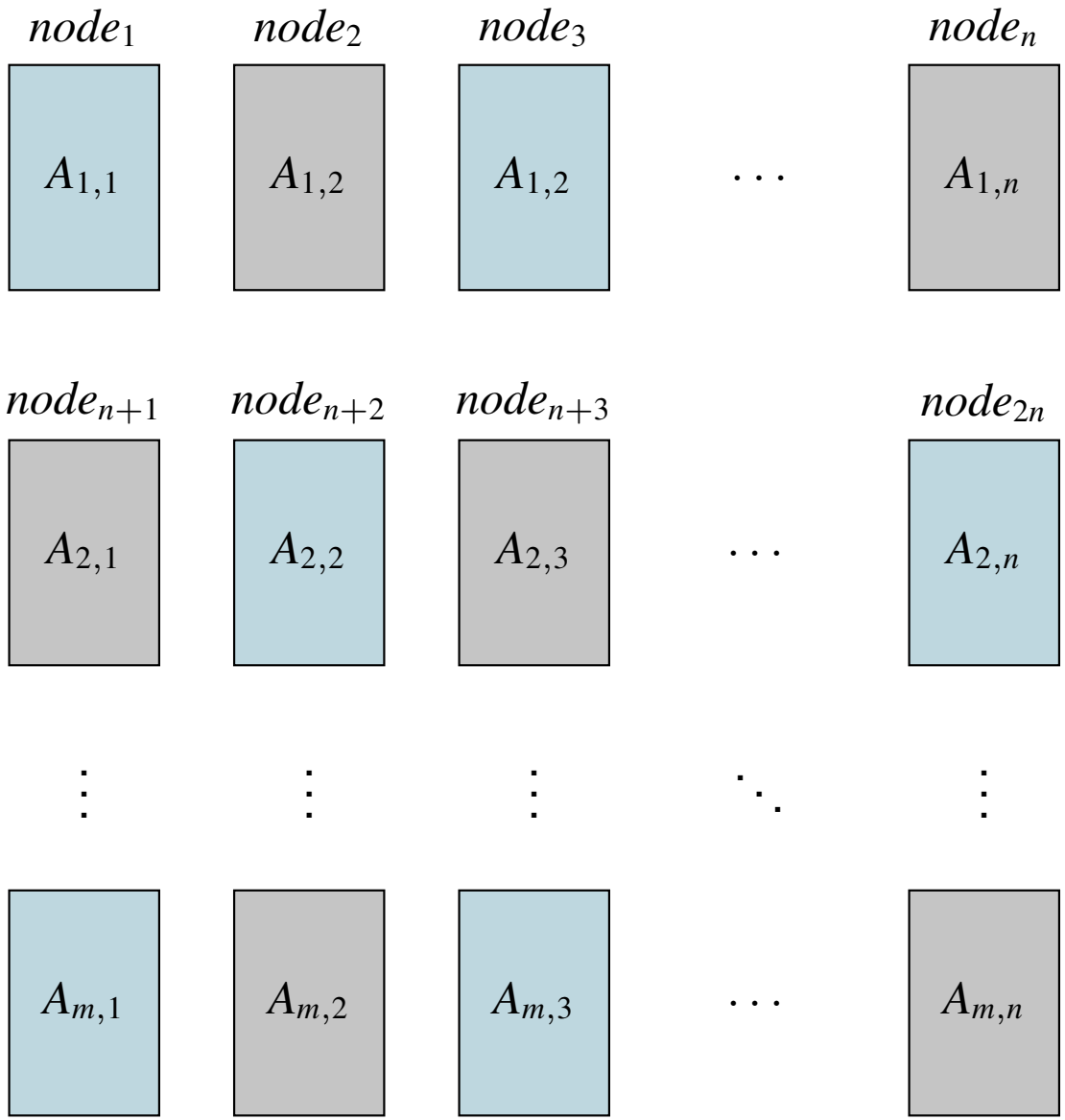
Desired features

- Develop a scheme for parallelization of the 3-terms recurrence relation Chebyshev filter.
- It would be nice to harness the power of GPUs.
- Limited GPU memory $\Rightarrow$ multiple GPU nodes
- Minimize communication and redistribution of data.

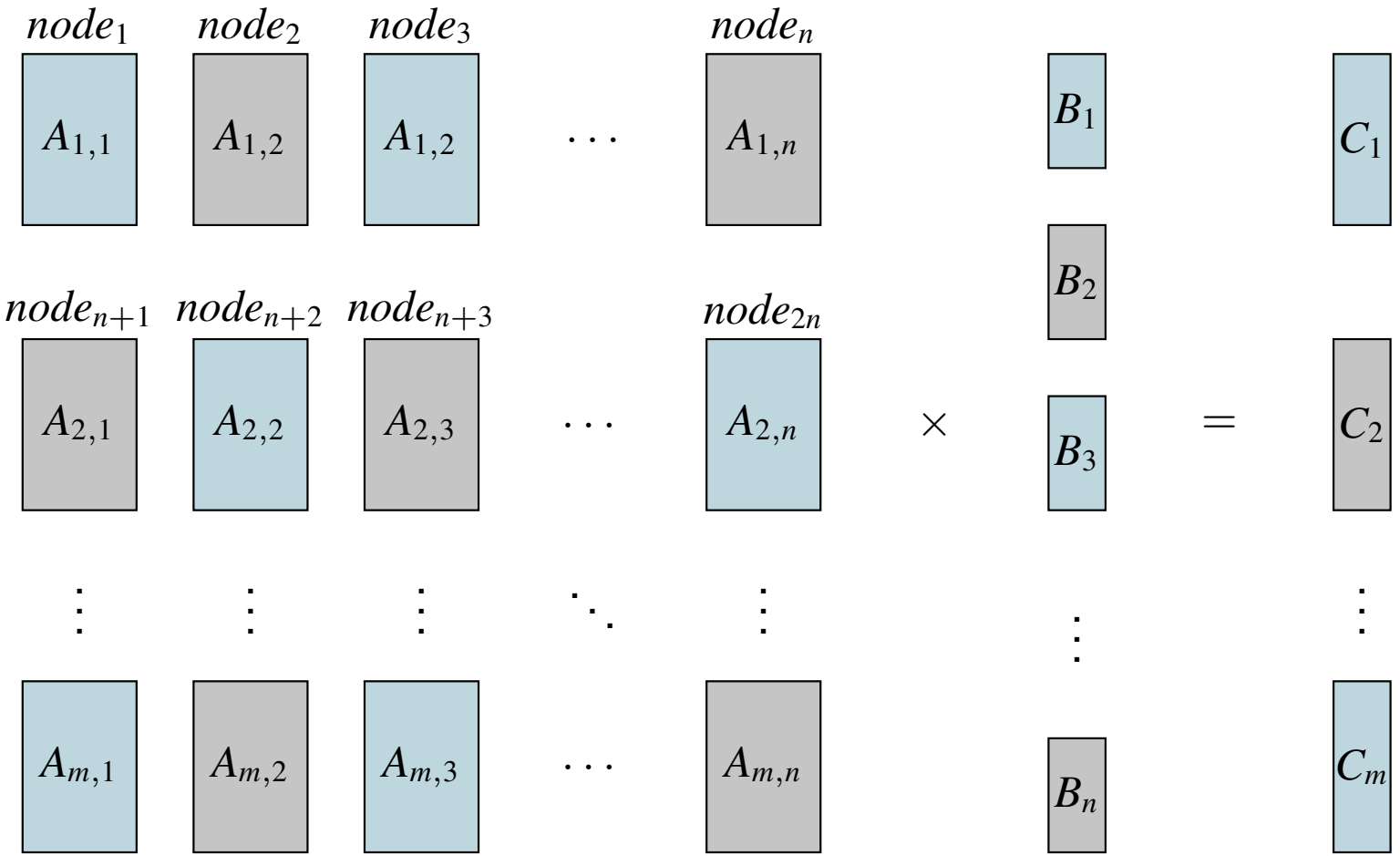
Matrix distribution

The matrix A is tiled and distributed among computing nodes.

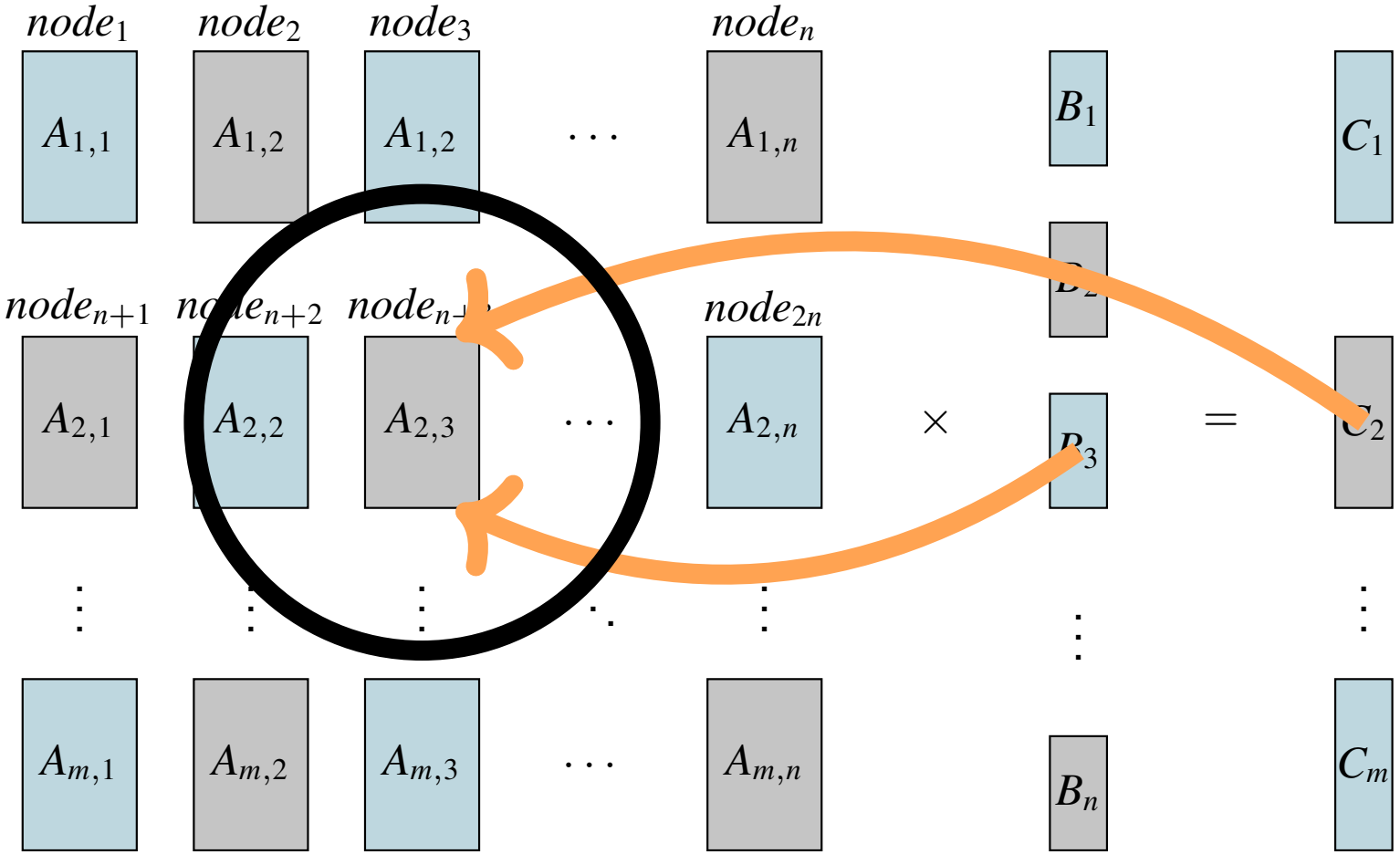



$node_1$
$node_2$
$node_3$
$node_n$
$A_{1,1}$
$A_{1,2}$
$A_{1,2}$
$A_{1,n}$
$node_{n+1}$
$node_{n+2}$
$node_{n+3}$
$node_{2n}$
$A_{2,1}$
$A_{2,2}$
$A_{2,3}$
$A_{2,n}$
$A_{m,1}$
$A_{m,2}$
$A_{m,3}$
$A_{m,n}$

- Each node gets the appropriate part of C and B.



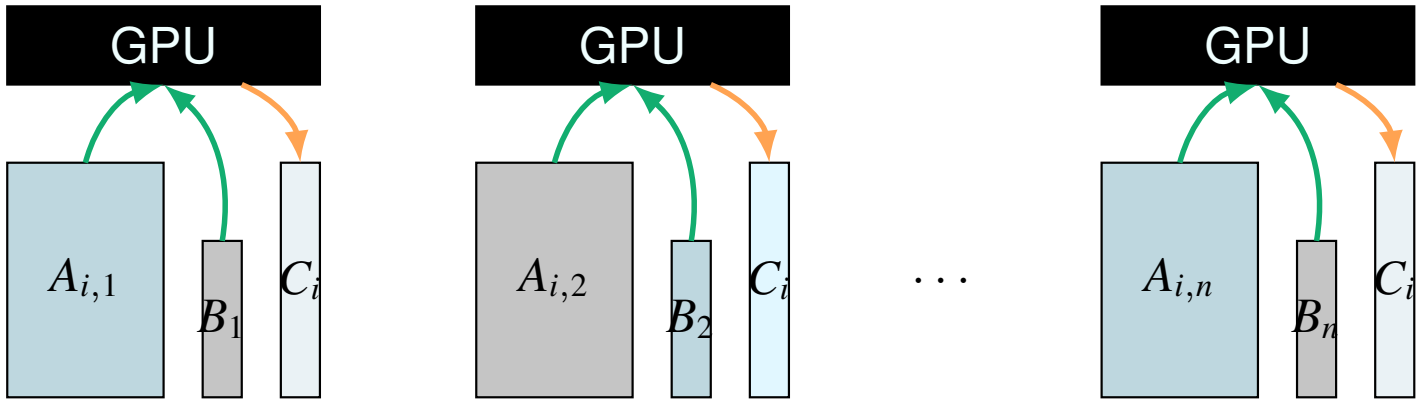
- Each node gets the appropriate part of C and B.



MPI scheme

Step 1

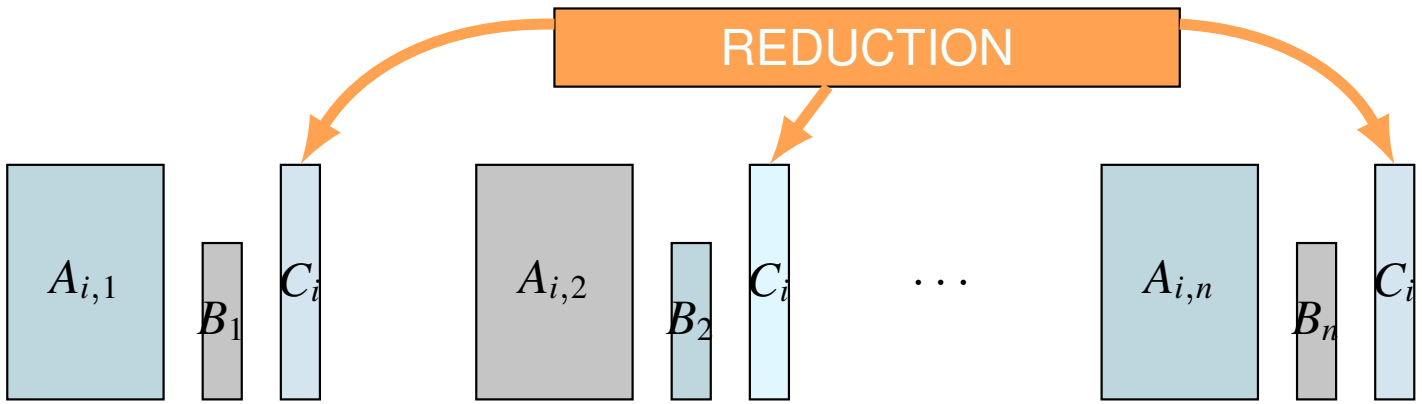
Calculate AB on the GPU, return it to CPU and save in temporary C_{tmp}.



MPI scheme

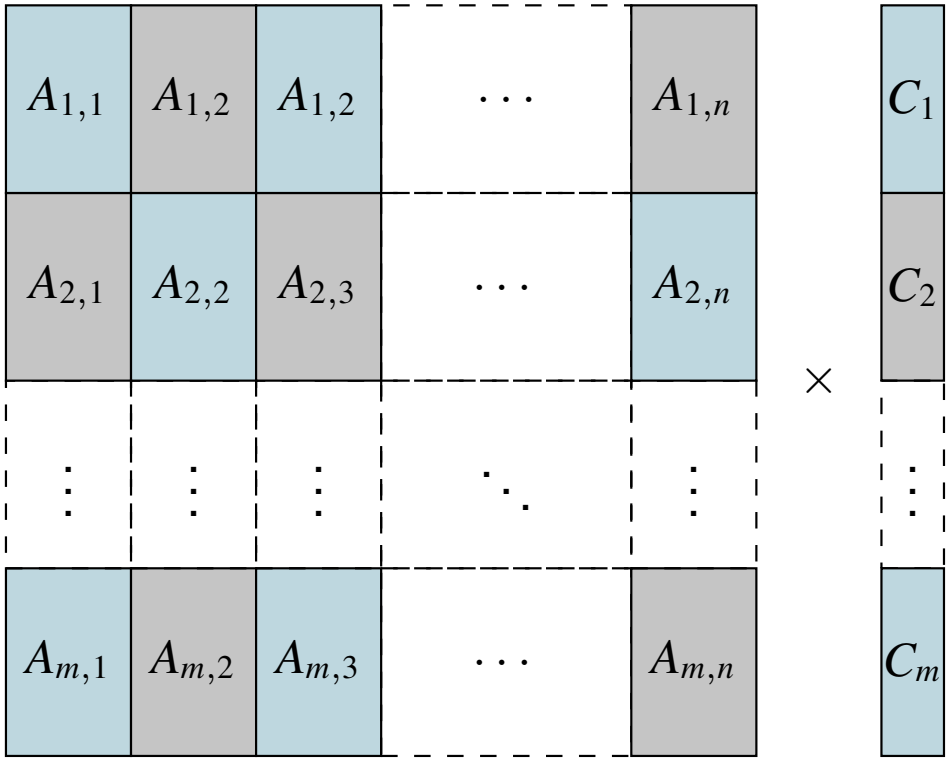
Step 2

Perform reduction (summation) on nodes in each row. Then save $\alpha C_{tmp} + \beta C$ in C.



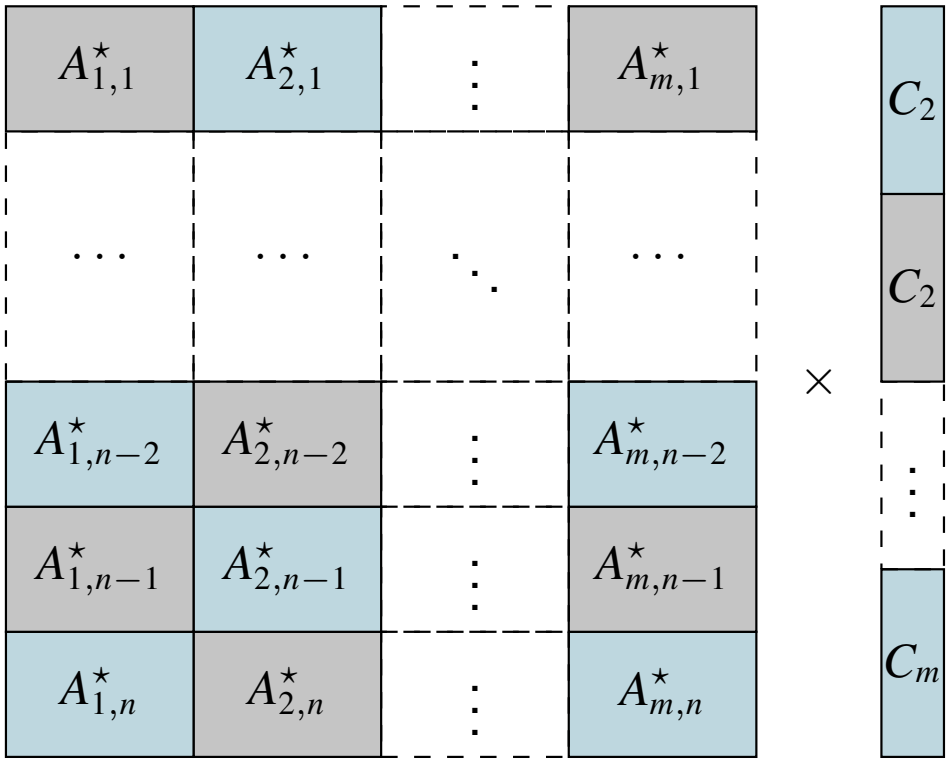
Next step:

Repeat the previous steps for $\alpha AC + \beta B \Longrightarrow$ requires redistribution of C



Next step:

Redistribution of C is avoided thanks to the simple observation that $A = A^H$

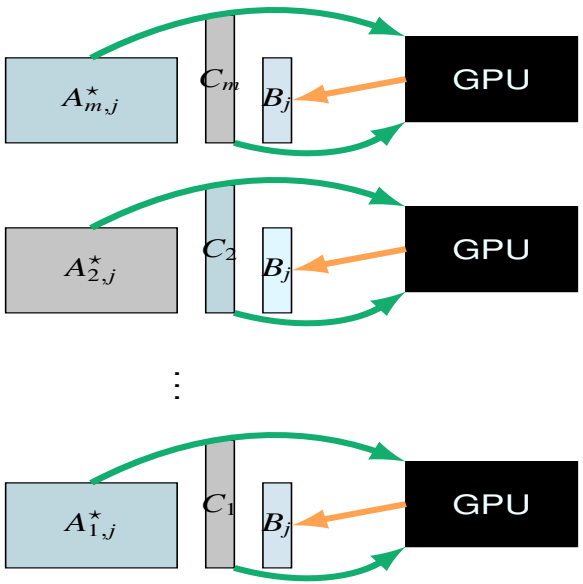


Repeat the previous steps for $\alpha A^H C + \beta B$

MPI scheme

Step 3

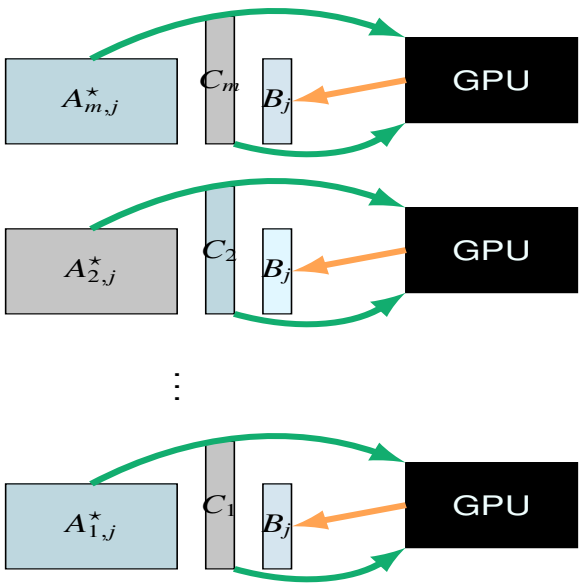
Calculate AC on the GPU, return it to CPU and save in temporary B_{tmp}.



MPI scheme

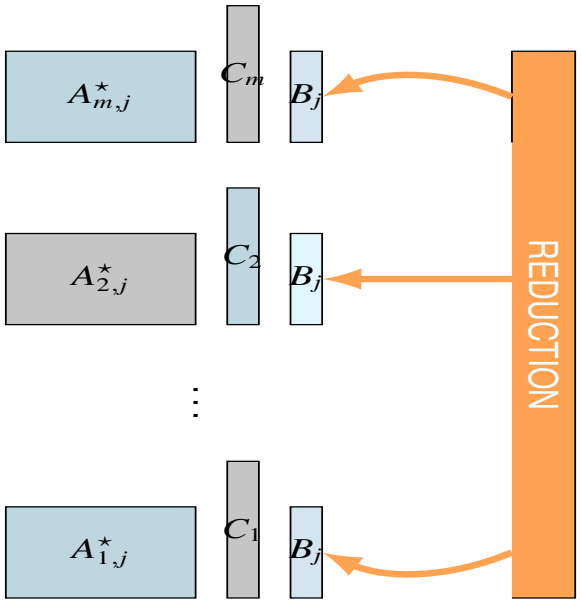
Step 3

Calculate AC on the GPU, return it to CPU and save in temporary B_{tmp}.



Step 4

Perform reduction on nodes in each column. Then save $\alpha B_{tmp} + \beta B$ in B.



MPI scheme: recap

- Steps 1-4 describe two cycles of Chebyshev iteration.
- Performing 3-terms recurrence relation within the Chebyshev iterations relies on alternating between both kinds of cycles.
- Cycle 1: Perform $A \times B$, and then reduce across every row of the processing grid.
- Cycle 2: Perform $A^{\star} \times C$, and then reduce on every column of the processing grid.
- Most of the communication is spent in a `MPI_Allreduce`.

Multi-GPU matrix multiplication schemes

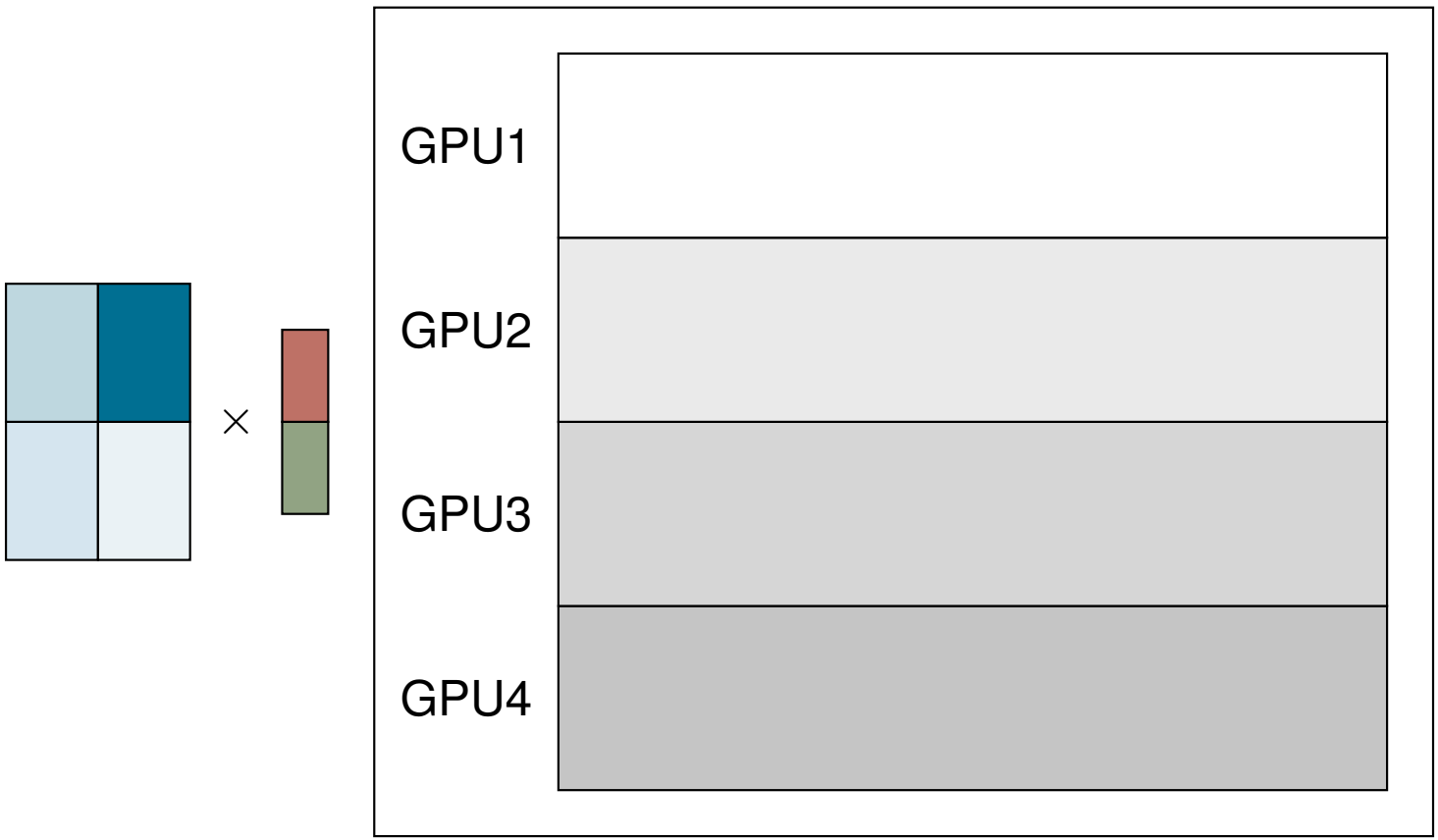
Guiding principle

- The distribution of $A_{i,j}$ on GPUs plays a guiding role
- The distribution of B_j and C_i is a result of the distribution of $A_{i,j}$.

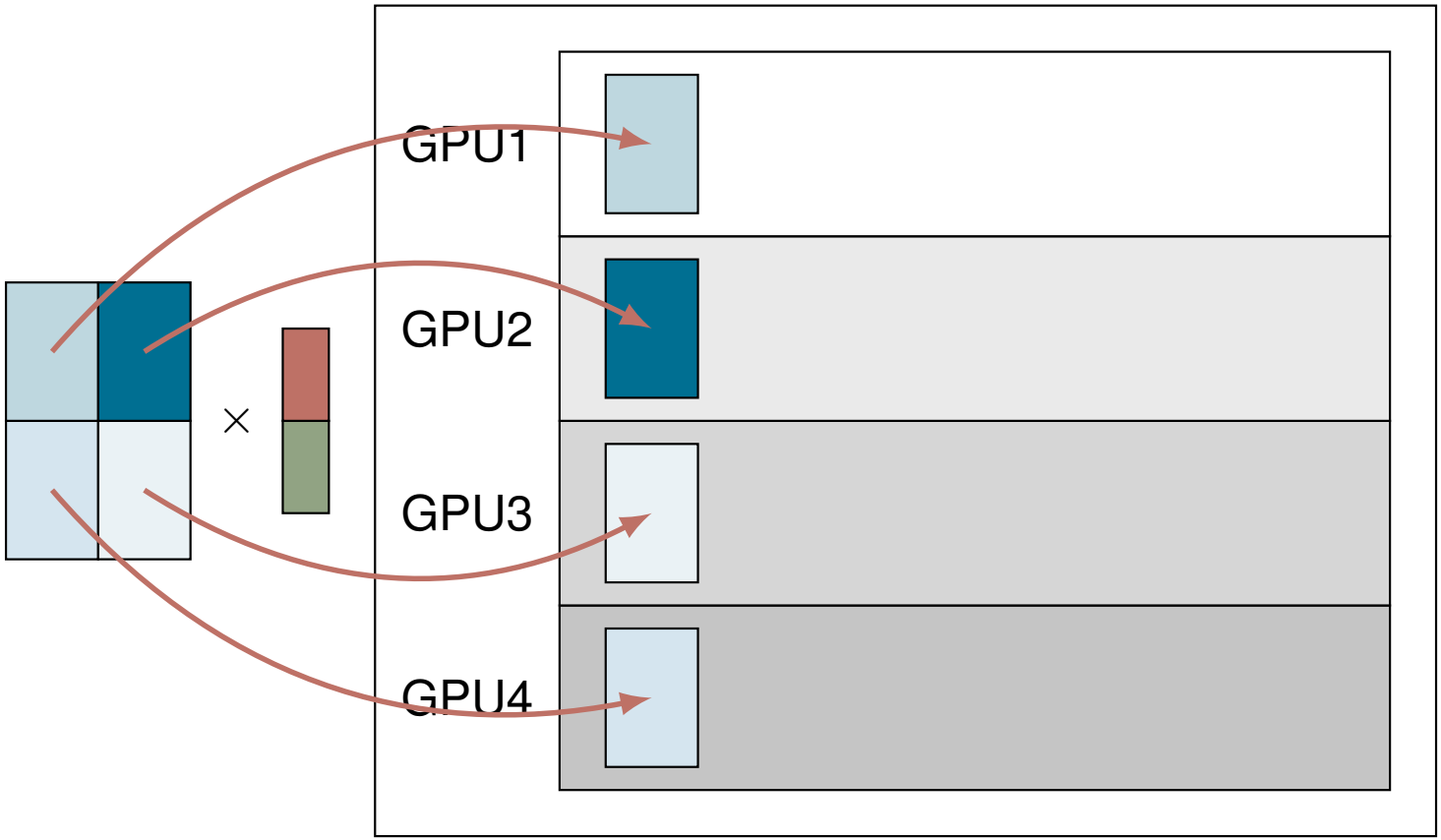
Example: 4 devices on one computing node
There are 3 simple distribution schemes for $A_{i,j}$:

- Vertical distribution (VER)
- Horizontal distribution (HOR)
- Mixed distribution (HV)

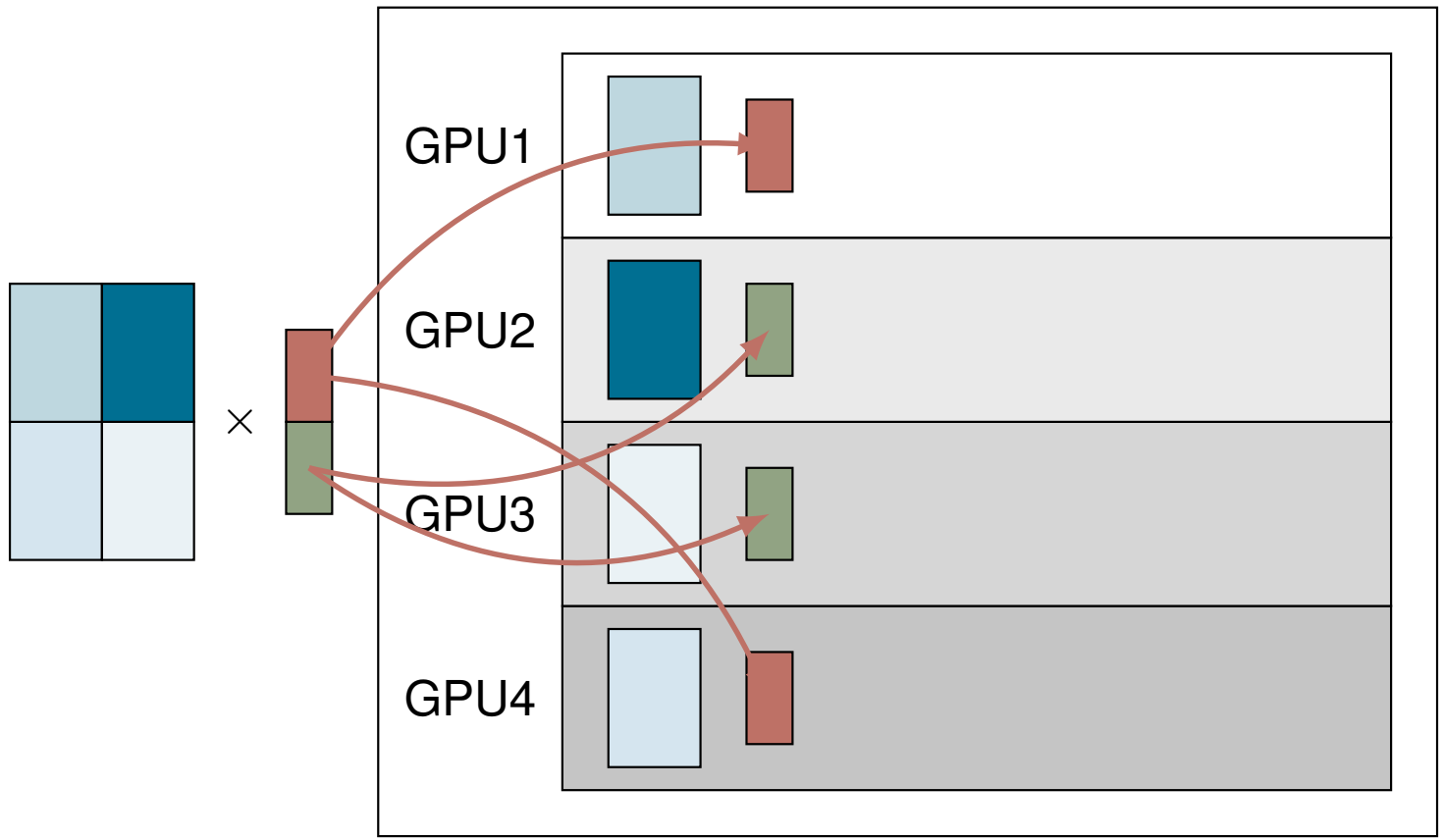
HV Scheme



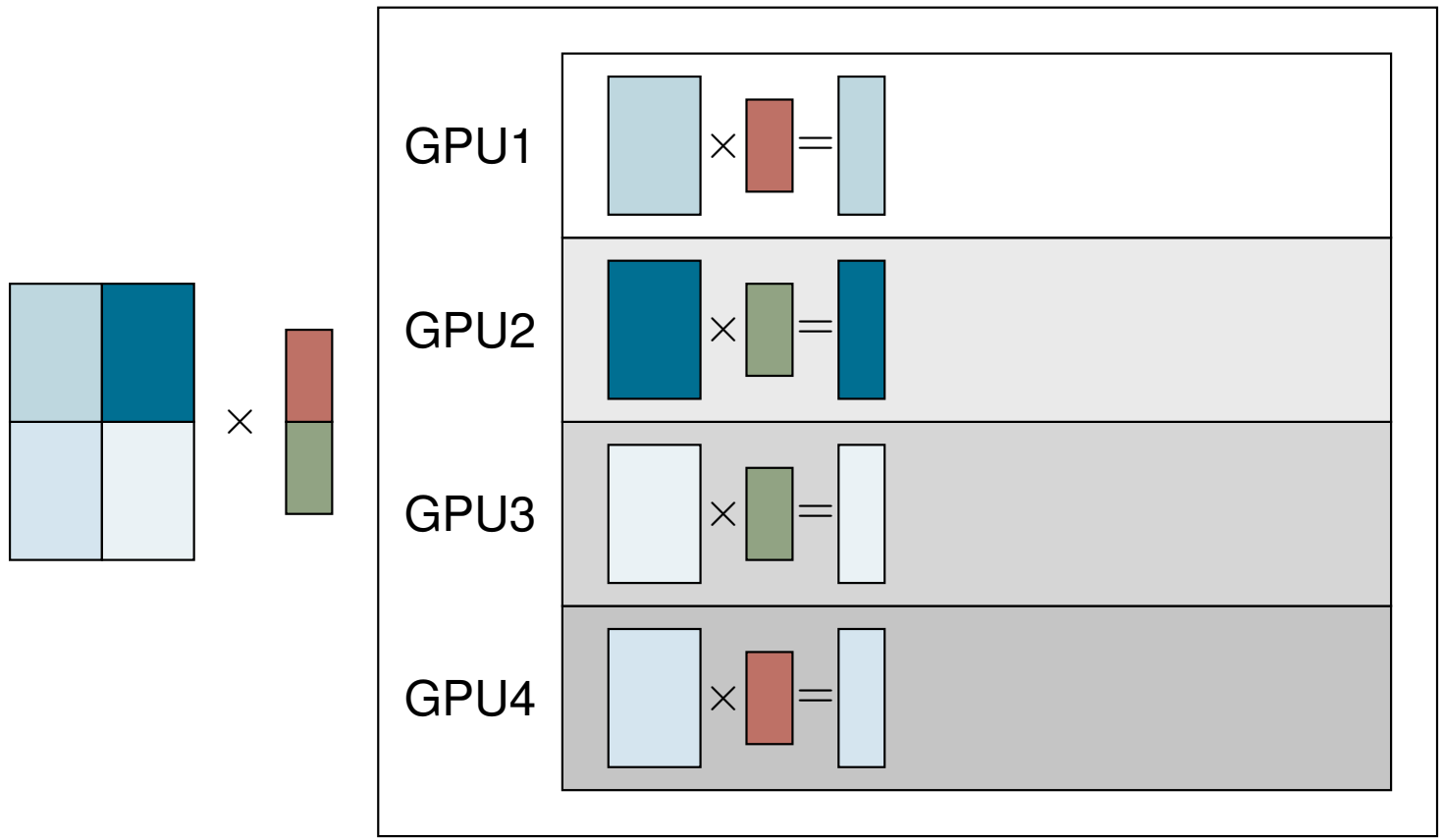
HV Scheme



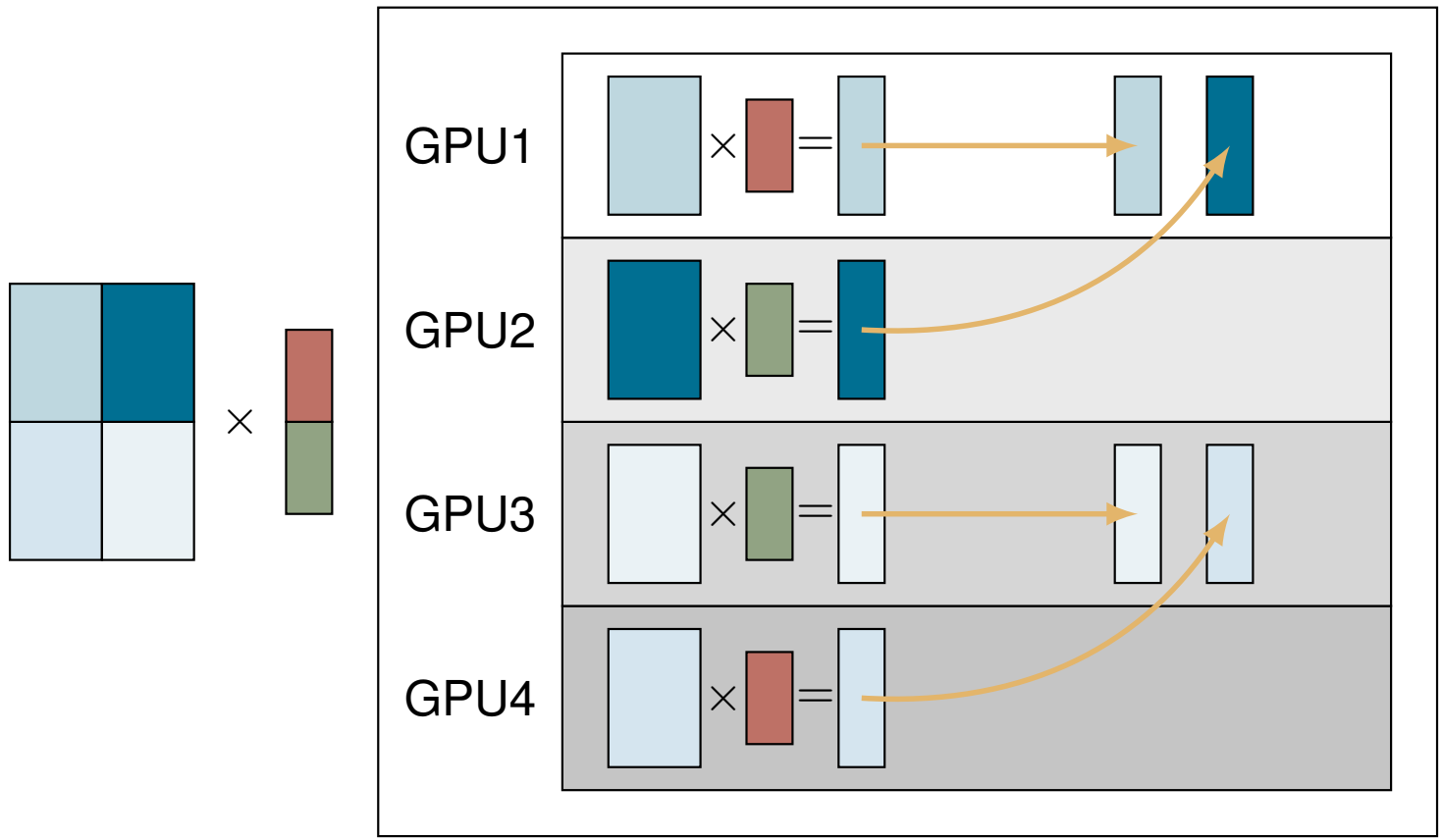
HV Scheme



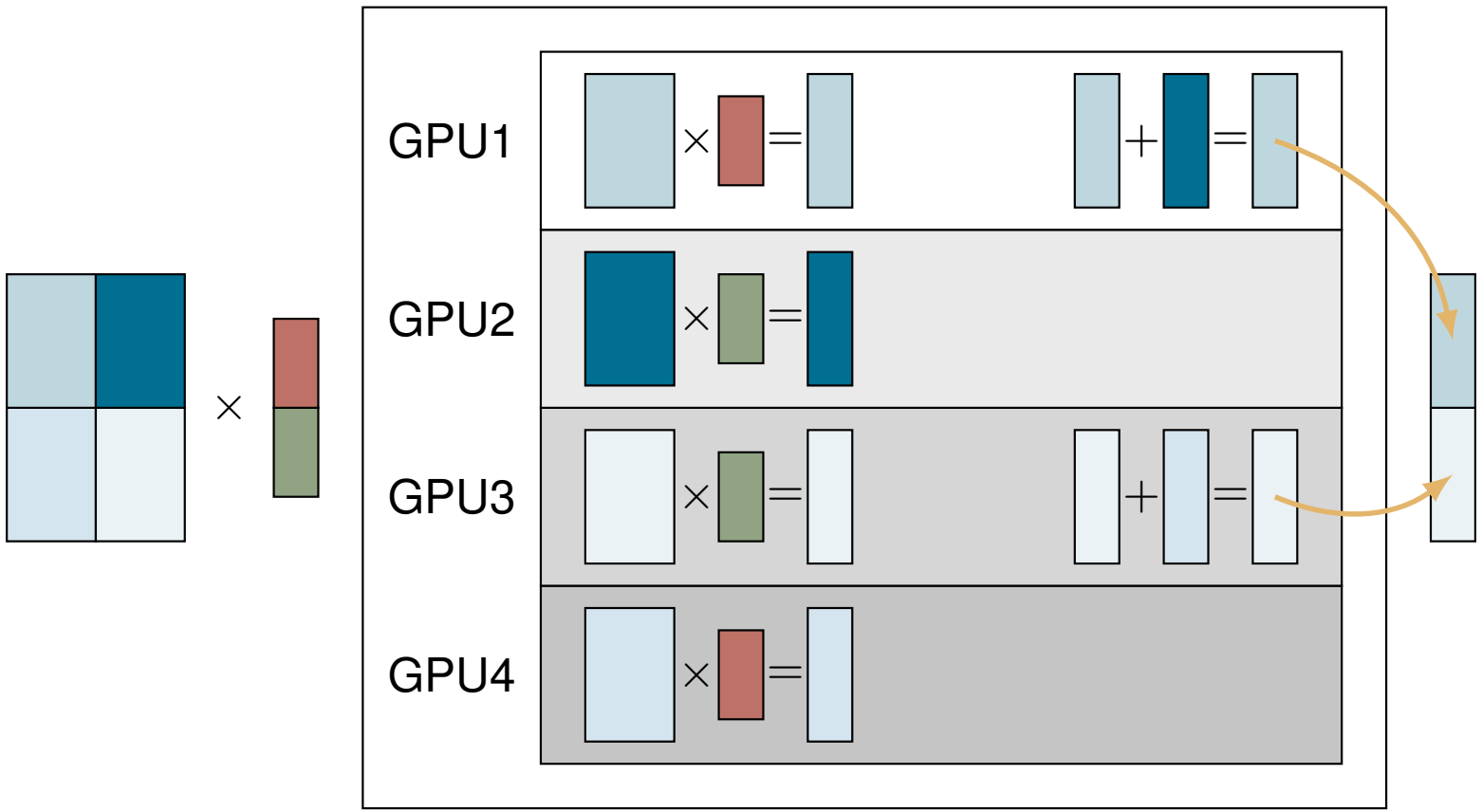
HV Scheme



HV Scheme



HV Scheme



Topic

Motivation

The eigensolver: Chebyshev Accelerated Subspace Iteration (ChASE)

Distributed CPU/GPU: a simple and efficient parallelization

Experimental tests and outlook

Experimental tests setup

Existing C++ implementation of ChASE

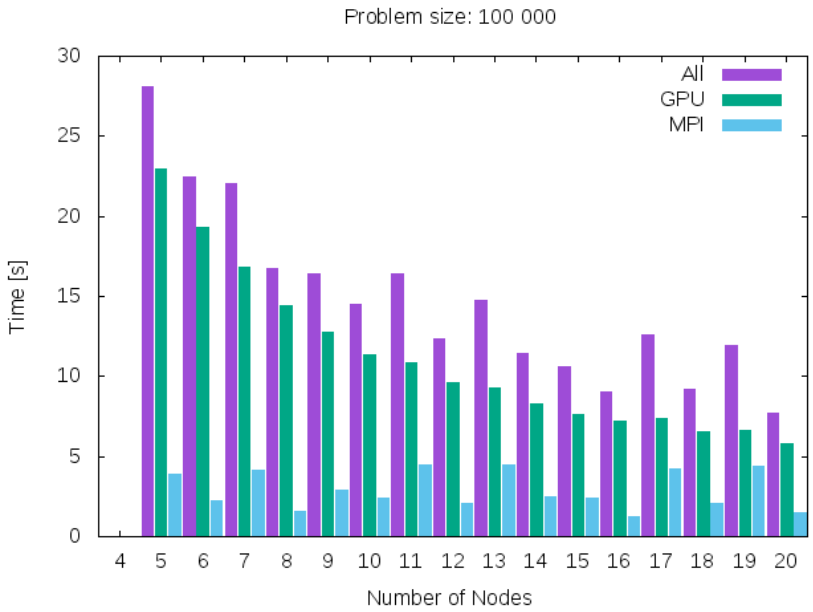
- EleChASE – Elemental (MPI) parallelization for distributed memory
- MTBChASE – Simple multi-threaded parallelization for shared memory
- CUChASE – CUDA parallelization to one GPUs (full or filter offload)
- BLASX+ChASE – Parallelization to multiple GPUs (per node)

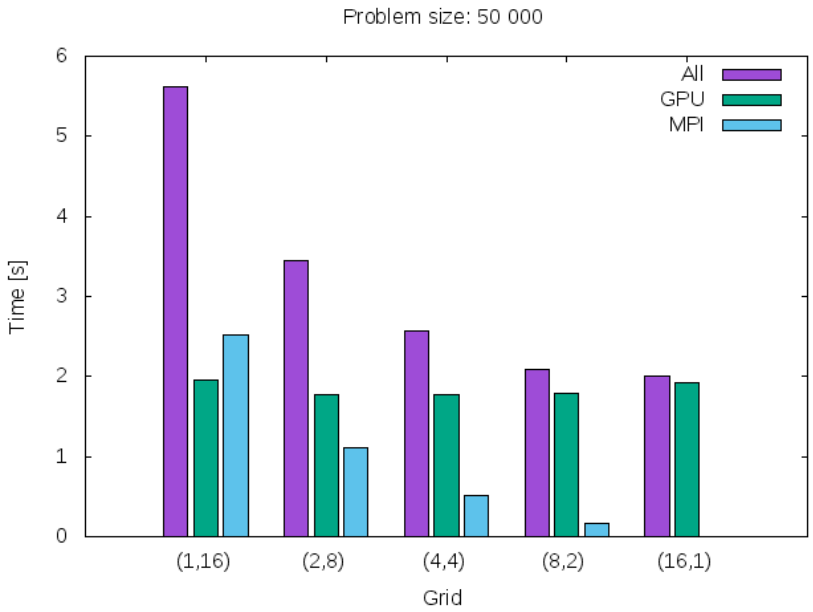
Tests were performed on the JURECA cluster for only the Chebyshev filter.

- 2 Intel Xeon E5-2680 v3 Haswell – Up to $0.96 \div 1.92$ TFLOPS DP$\div$ SP;
- 2 x NVIDIA K80 (four devices) – Up to $2 \times 2.91 \div 8.74$ TFLOPS DP $\div$ SP.
- 4 GB of GDDR5 memory (12 GB per GPU);
- 480 GB/sec memory bandwidth per board;
- Artificial matrices generated on the fly for benchmark purposes

Communication vs computation

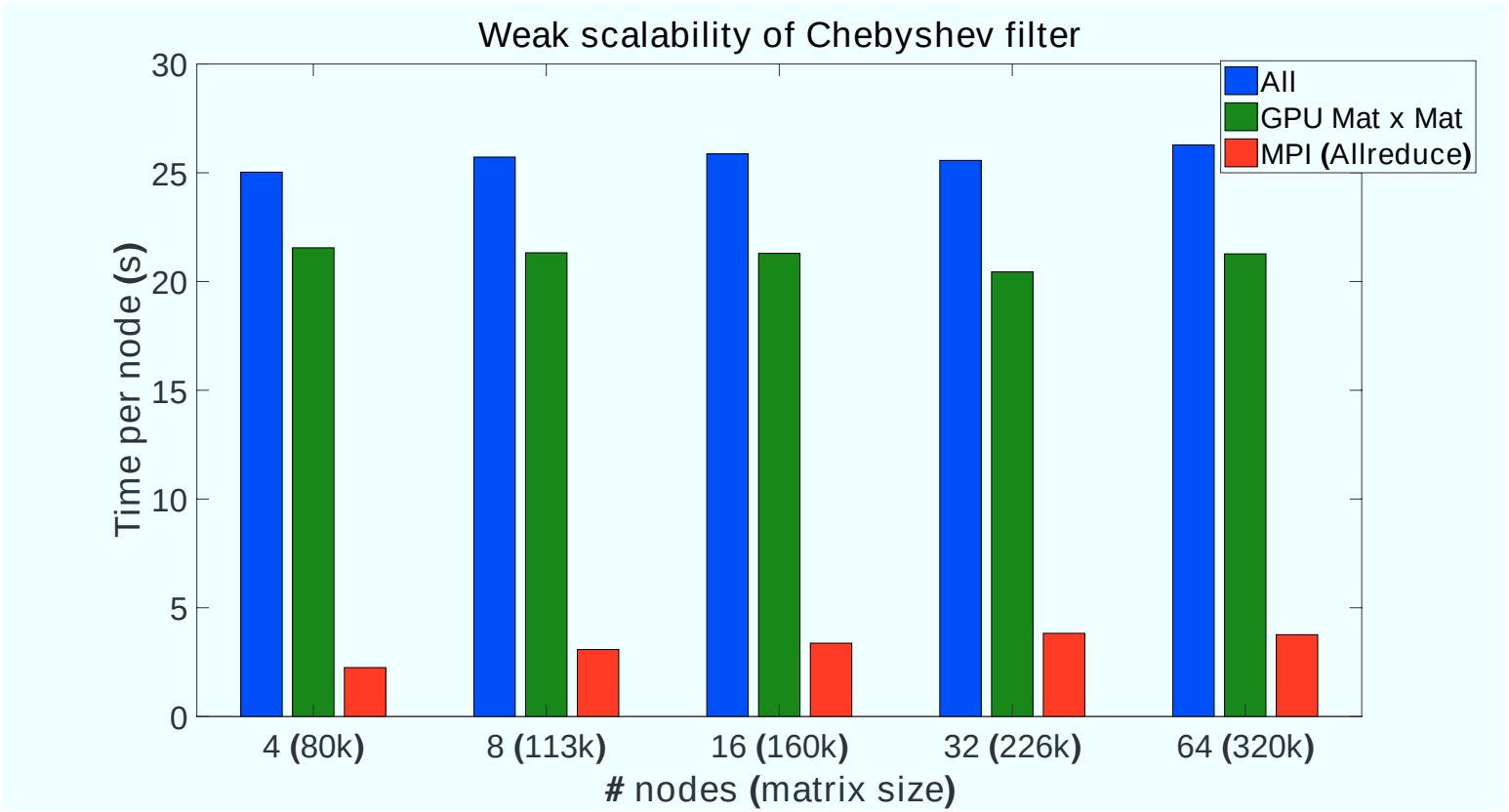
Computing node geometry





- MPI communication is heavily influenced (expected) by computing grid;
- Binary and (especially) squared grids are preferred.

Weak scalability test



- Volume of memory per GPU device occupied by $A_{i,j}$ fixed ($\sim$ 10 GB);
- Communication increases only as $\log(\#\text{nodes})$.

Some observations

Very simple parallelization

1. Using only GPUs for filter
2. Once integrated CPU cores could execute some other ChASE overlapping tasks;
3. B and C are very tall and skinny matrices: tiling or cyclic block distribution could improve performance at the cost of having to redistributed across filter iterations;
4. BLASX could (theoretically) be used at the node level in order to use concurrently both GPUs and CPUs

Some noticeable advantages for ChASE

- Compute bound
- Performance portable (need optimizing for few linear algebra kernels)
- Templating ChASE for SP $\Rightarrow$ up to 4 times the performance on GPUs

Outlook

Next steps

- Templating ChASE filter for SP (the rest of ChASE is already templated);
- MPI can be tweaked to reduce latency.
- Reconfigure VASP BSE package to initialize matrices in DP;
- Refine node-level parallelism with multiple GPUs together with CPU cores $\Longrightarrow$ modify BLASX;
- Implementing a distributed CPU/GPU parallelization for the remaining ChASE inner functions (QR, Rayleigh-Ritz, etc.);
- Computing Lanczos DoS step redundantly on each computing node..

For more information

e.di.napoli@fz-juelich.de

http://www.jara.org/hpc/slai