CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Domain specific libraries

Material science codes on innovative HPC architectures

Anton Kozhevnikov, CSCS

December 5, 2016

MAX DRIVING THE EXASCALE TRANSITION

Part 1: Introduction

Kohn-Shame equations

Eigen-value problem

$$\left(-\frac{1}{2}\Delta + v_{eff}(\mathbf{r})\right)\psi_j(\mathbf{r}) = \varepsilon_j \psi_j(\mathbf{r})$$

Density generation

$$\rho^{new}(\mathbf{r}) = \sum_j |\psi_j(\mathbf{r})|^2$$

Density mixing

$$\rho(\mathbf{r}) = \alpha\rho^{new}(\mathbf{r}) + (1-\alpha)\rho^{old}(\mathbf{r})$$

Effective potential construction

$$v_{eff}(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}'-\mathbf{r}|} d\mathbf{r}' + v_{XC}[\rho](\mathbf{r}) + v_{ext}(\mathbf{r})$$

Output:

- wave-functions $\psi_j(\mathbf{r})$ and eigen energies ε_j
- charge density $\rho(\mathbf{r})$ and magnetization $\mathbf{m}(\mathbf{r})$
- total energy E_{tot} and atomic forces $\mathbf{F}_\alpha$

Derived objects:

- Spectral functions
- Linear response functions
- Phonons, thermodynamic functions
- Parameters of lattice models

Basis expansion

- Expand Kohn-Sham wave functions in a convenient basis set:

$$\psi_j(\mathbf{r}) = \sum_\mu C_{\mu j}\phi_\mu(\mathbf{r})$$

- Convert differential equation to an eigen-value problem:

$$\mathbf{HC}_j = \varepsilon_j \mathbf{OC}_j$$

where

$$H_{\mu\mu'} = \int \phi^*_\mu(\mathbf{r})\Big(-\frac{1}{2}\Delta + v_{eff}(\mathbf{r})\Big)\phi_{\mu'}(\mathbf{r})d\mathbf{r}$$

$$O_{\mu\mu'} = \int \phi^*_\mu(\mathbf{r})\phi_{\mu'}(\mathbf{r})d\mathbf{r}$$

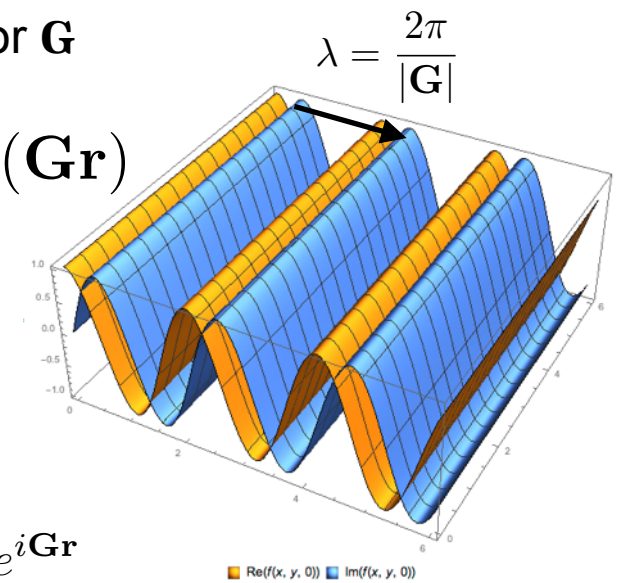
Electronic structure codes

Code	Basis set
Quantum ESPRESSO VASP ABINIT PEtot CPMD QBOX CASTEP	plane-waves
SIESTA CRYSTAL FHI-AIMS FPLO OpenMX TB-LMTO	localized atom-centered orbitals
CP2K	gaussians + plane-waves
BigDFT	wavelets
WIEN2k FLEUR Exciting Elk	linarized augmented plane-waves
GPAW Octopus	real space grid / plane waves / atomic orbitals

Plane-wave basis

Plane-wave with the wave-vector $\mathbf{G}$

$$e^{i\mathbf{G}\mathbf{r}} = e^{i(xG_x + yG_y + zG_z)} = \cos(\mathbf{G}\mathbf{r}) + i\sin(\mathbf{G}\mathbf{r})$$



- Complete orthogonal basis
- Eigen-functions of the Laplace operator

$$\left(\frac{\partial^2}{\partial^2 x} + \frac{\partial^2}{\partial^2 y} + \frac{\partial^2}{\partial^2 z}\right) e^{i\mathbf{G}\mathbf{r}} = -G^2 e^{i\mathbf{G}\mathbf{r}}$$

- Fourier transformations can be rapidly evaluated with FFT
- Real-space integrals between periodic functions are represented as sums

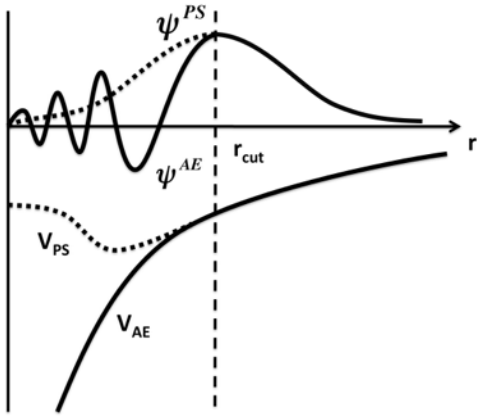
$$\int f^*(\mathbf{r}) g(\mathbf{r}) d\mathbf{r} = \frac{\Omega}{N} \sum_j f^*(\mathbf{r}_j) g(\mathbf{r}_j) = \sum_{\mathbf{G}} \tilde{f}^*(\mathbf{G}) g(\mathbf{G})$$

Pseudopotential

- Plane-wave basis requires a pseudopotential

- Unlike the true (all-electron) potential, pseudopotential has an additional non-local term:

$$\hat{V}_{\mathrm{PS}} = V_{loc}(\mathbf{r}) + \sum_{\alpha} \sum_{\xi\xi'} |\beta_{\xi}^{\alpha}\rangle D_{\xi\xi'}^{\alpha} \langle \beta_{\xi'}^{\alpha}|$$



- Action of the pseudopotential on the wave-function:

$$\hat{V}_{\mathrm{PS}}\psi(\mathbf{r}) = V_{loc}(\mathbf{r})\psi(\mathbf{r}) + \sum_{\alpha} \sum_{\xi\xi'} \beta_{\xi}^{\alpha}(\mathbf{r}) D_{\xi\xi'}^{\alpha} \int \beta_{\xi'}^{\alpha}(\mathbf{r})\psi(\mathbf{r}) d\mathbf{r}$$

Eigen-value problem

$$\mathbf{H}\tilde{\psi}_j = \varepsilon_j \tilde{\psi}_j$$ in case of norm-conserving pseudopotential

$$\mathbf{H}\tilde{\psi}_j = \varepsilon_j \mathbf{S}\tilde{\psi}_j$$ in case of ultrasoft pseudopotential

- **H** and **S** are dense Hermitian matrices of large dimension (~1000 PW / atom).

Eigen-value problem

$$\mathbf{H}\tilde{\psi}_j = \varepsilon_j \tilde{\psi}_j$$ in case of norm-conserving pseudopotential

$$\mathbf{H}\tilde{\psi}_j = \varepsilon_j \mathbf{S}\tilde{\psi}_j$$ in case of ultrasoft pseudopotential

- **H** and **S** are dense Hermitian matrices of large dimension (~1000 PW / atom).
- Iterative subspace diagonalziation methods are used.
  - Conjugate gradient
  - Davidson algorithm
  - RMM-DIIS
  - LOBPCG
  - Chebyshev filering method
  - …

Hamiltonian operator application

- Any iterative solver requires $\hat{H}|\psi\rangle$ operation

Hamiltonian operator application

- Any iterative solver requires $\hat{H}|\psi\rangle$ operation

- Result of Hamiltonian application

$$\tilde{h}_{\psi_j} = \mathbf{H}\tilde{\psi}_j = \int e^{-i\mathbf{G}\mathbf{r}} \hat{H}\psi_j(\mathbf{r})d\mathbf{r}$$

Hamiltonian operator application

- Any iterative solver requires $\hat{H}|\psi\rangle$ operation

- Result of Hamiltonian application

$$\tilde{h}_{\psi_j} = \mathbf{H}\tilde{\psi}_j = \int e^{-i\mathbf{G}\mathbf{r}} \hat{H}\psi_j(\mathbf{r})d\mathbf{r}$$

- Hamiltonian operator with pseudopotential

$$\hat{H} = -\frac{1}{2}\Delta + v_{eff}(\mathbf{r}) + \sum_{\alpha}\sum_{\xi\xi'} |\beta_{\xi}^{\alpha}\rangle D_{\xi\xi'}^{\alpha} \langle\beta_{\xi'}^{\alpha}|$$

Hamiltonian operator application

Application of the local part of potential (V_{loc} kernel)

$$\tilde{\psi}_j(\mathbf{G}) \xrightarrow{FFT^{-1}} \psi_j(\mathbf{r}) \rightarrow v_{eff}(\mathbf{r})\psi_j(\mathbf{r}) \xrightarrow{FFT} \tilde{h}_{\psi_j}(\mathbf{G})$$

Hamiltonian operator application

Application of the local part of potential (V_{loc} kernel)

$$\tilde{\psi}_j(\mathbf{G}) \xrightarrow{FFT^{-1}} \psi_j(\mathbf{r}) \rightarrow v_{eff}(\mathbf{r})\psi_j(\mathbf{r}) \xrightarrow{FFT} \tilde{h}_{\psi_j}(\mathbf{G})$$

```
do j = 1, num_psi
  ! transform psi to real-space domain
  call inverse_fft(psi(:,j), psi_of_r(:))
  ! multiply by effective potential
  do ir = 1, num_r_points
    psi_of_r(ir) = psi_of_r(ir) * veff(ir)
  end do
  ! tranform back to PW domain
  call forward_fft(psi_of_r(:), hpsi(:,j))
end do
```

Two FFTs for each wave-function!

Hamiltonian operator application

Application of Laplace operator

$$-\frac{1}{2}\Delta\psi_j(\mathbf{r}) = \sum_{\mathbf{G}}\Big(-\frac{1}{2}\Delta e^{i\mathbf{G}\mathbf{r}}\Big)\tilde{\psi}_j(\mathbf{G}) = \sum_{\mathbf{G}} e^{i\mathbf{G}\mathbf{r}}\frac{G^2}{2}\tilde{\psi}_j(\mathbf{G})$$

Hamiltonian operator application

Application of Laplace operator

$$-\frac{1}{2}\Delta\psi_j(\mathbf{r}) = \sum_{\mathbf{G}}\Big(-\frac{1}{2}\Delta e^{i\mathbf{G}\mathbf{r}}\Big)\tilde{\psi}_j(\mathbf{G}) = \sum_{\mathbf{G}} e^{i\mathbf{G}\mathbf{r}}\frac{G^2}{2}\tilde{\psi}_j(\mathbf{G})$$

```
do j = 1, num_psi
  ! add kinetic energy contribution
  do ig = 1, num_gvec
    hpsi(ig, j) = hpsi(ig, j) + pw_ekin(ig) * psi(ig, j)
  end do
end do
```

Hamiltonian operator application

Application of non-local part of potential: $\sum_{\alpha\xi\xi'} |\beta^{\alpha}_{\xi}\rangle D^{\alpha}_{\xi\xi'} \langle\beta^{\alpha}_{\xi'}|$

$$\underbrace{\sum_{\alpha\xi} \beta^{\alpha}_{\xi}(\mathbf{G}) \underbrace{\sum_{\xi'} D^{\alpha}_{\xi\xi'} \underbrace{\sum_{\mathbf{G}'} \beta^{\alpha *}_{\xi'}(\mathbf{G}')\tilde{\psi}_j(\mathbf{G}')}_{\text{zgemm\#1}}}_{\text{zgemm\#2}}}_{\text{zgemm\#3}} \rightarrow \tilde{h}_{\psi_j}(\mathbf{G})$$

Hamiltonian operator application

Application of non-local part of potential: $\sum_{\alpha\xi\xi'} |\beta_\xi^\alpha\rangle D_{\xi\xi'}^\alpha \langle\beta_{\xi'}^\alpha|$

$$\underbrace{\sum_{\alpha\xi} \beta_\xi^\alpha(\mathbf{G}) \underbrace{\sum_{\xi'} D_{\xi\xi'}^\alpha \underbrace{\sum_{\mathbf{G}'} \beta_{\xi'}^{\alpha *}(\mathbf{G}')\tilde{\psi}_j(\mathbf{G}')}_{\text{zgemm\#1}}}_{\text{zgemm\#2}}}_{\text{zgemm\#3}} \rightarrow \tilde{h}_{\psi_j}(\mathbf{G})$$

```
! compute <beta|psi>
call zgemm('C', 'N', num_beta_tot, num_psi, num_pw, beta_pw, psi, beta_psi)
! apply block-diagonal D matrix
do ia = 1, num_atoms
  call zgemm('N', 'N', num_beta(ia), num_psi, num_beta(ia),
             D(:, :, ia), beta_psi(offset(ia), 1), d_beta_psi(offset(ia), 1))
end do
! compute beta*D*<beta|psi>
call zgemm('N', 'N', num_pw, num_psi, num_beta_tot, beta_pw, d_beta_psi, hpsi)
```

Hamiltonian operator application

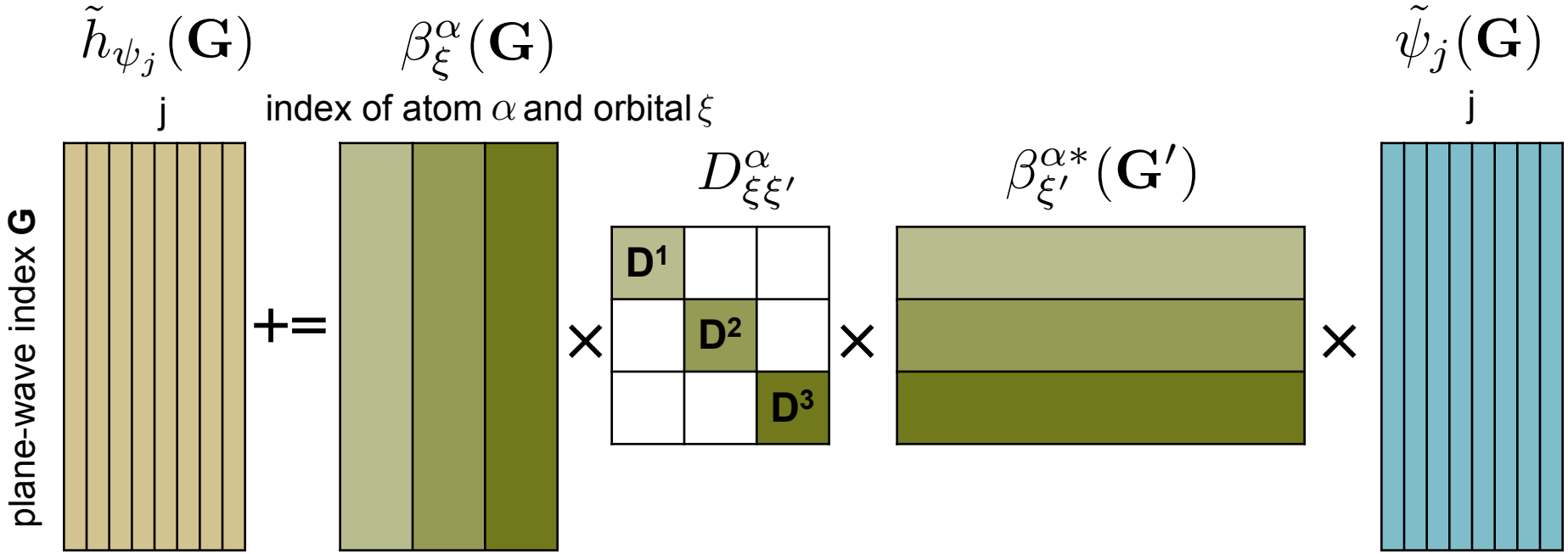
Application of non-local part of potential: $\sum_{\alpha\xi\xi'} |\beta^\alpha_\xi\rangle D^\alpha_{\xi\xi'} \langle\beta^\alpha_{\xi'}|$

$$\sum_{\alpha\xi} \beta^\alpha_\xi(\mathbf{G}) \sum_{\xi'} D^\alpha_{\xi\xi'} \sum_{\mathbf{G}'} \beta^{\alpha*}_{\xi'}(\mathbf{G}')\tilde{\psi}_j(\mathbf{G}') \rightarrow \tilde{h}_{\psi_j}(\mathbf{G})$$

zgemm#1

zgemm#2

zgemm#3



Davidson iterative solver

- We want to compute $$\mathbf{H}\tilde{\psi}_j = \varepsilon_j \tilde{\psi}_j$$
- We know how to compute $$\tilde{h}_{\psi_j} = \mathbf{H}\tilde{\psi}_j$$

Davidson iterative solver

- We want to compute $$\mathbf{H}\tilde{\psi}_j = \varepsilon_j \tilde{\psi}_j$$
- We know how to compute $$\tilde{h}_{\psi_j} = \mathbf{H}\tilde{\psi}_j$$

Key idea of Davidson iterative solver: start with a subspace spanned by a guess to ψ_j and expand it with preconditioned residuals

Davidson iterative solver

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$

Davidson iterative solver

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$

Davidson iterative solver

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$

Davidson iterative solver

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$
- Diagonalize reduced Hamiltonian matrix and get N lowest eigen pairs: $\mathbf{h}^m\mathbf{Z}^m = \epsilon_j\mathbf{Z}^m$

Davidson iterative solver

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$
- Diagonalize reduced Hamiltonian matrix and get N lowest eigen pairs: $\mathbf{h}^m\mathbf{Z}^m = \epsilon_j\mathbf{Z}^m$
- Compute residuals $(R_j = \hat{H}\psi_j - \epsilon_j\psi_j)$: $\tilde{R}_j^m = \tilde{h}_{\phi_j^m}\mathbf{Z}^m - \epsilon_j\tilde{\phi}_j^m\mathbf{Z}^m$

Davidson iterative solver

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$
- Diagonalize reduced Hamiltonian matrix and get N lowest eigen pairs: $\mathbf{h}^m\mathbf{Z}^m = \epsilon_j\mathbf{Z}^m$
- Compute residuals $(R_j = \hat{H}\psi_j - \epsilon_j\psi_j)$: $\tilde{R}_j^m = \tilde{h}_{\phi_j^m}\mathbf{Z}^m - \epsilon_j\tilde{\phi}_j^m\mathbf{Z}^m$
- Apply preconditioner to the unconverged residuals, orthogonalize and add the resulting functions to the basis: $\{\tilde{\phi}_j^{m+1}\} = \{\tilde{\phi}_j^m\} \bigoplus \{P\tilde{R}_j^m\}$

Davidson iterative solver

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$
- Diagonalize reduced Hamiltonian matrix and get N lowest eigen pairs: $\mathbf{h}^m\mathbf{Z}^m = \epsilon_j\mathbf{Z}^m$
- Compute residuals $(R_j = \hat{H}\psi_j - \epsilon_j\psi_j)$: $\tilde{R}_j^m = \tilde{h}_{\phi_j^m}\mathbf{Z}^m - \epsilon_j\tilde{\phi}_j^m\mathbf{Z}^m$
- Apply preconditioner to the unconverged residuals, orthogonalize and add the resulting functions to the basis: $\{\tilde{\phi}_j^{m+1}\} = \{\tilde{\phi}_j^m\} \bigoplus \{P\tilde{R}_j^m\}$
- Recompute the wave-functions: $\tilde{\psi}_j = \tilde{\phi}_j^m\mathbf{Z}^m$

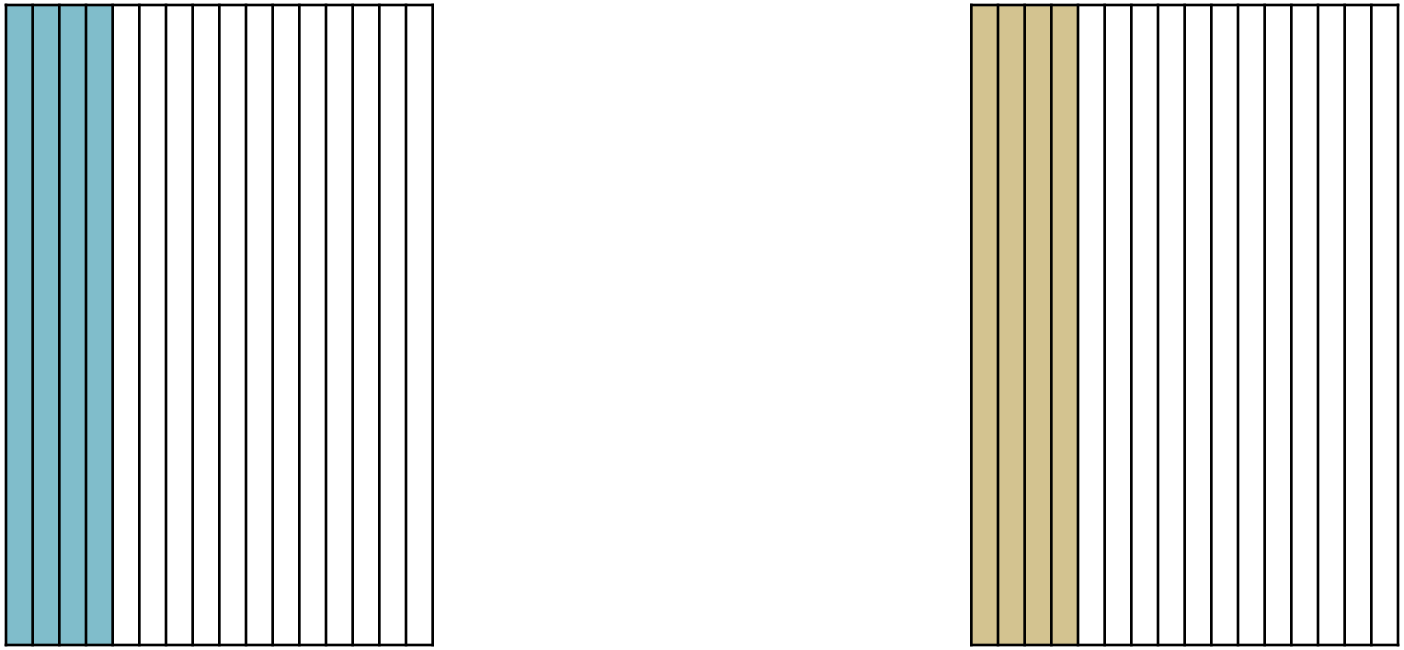
Davidson iterative solver

- Initialize subspace basis functions and apply Hamiltonian

$\tilde{\phi}_j^0$

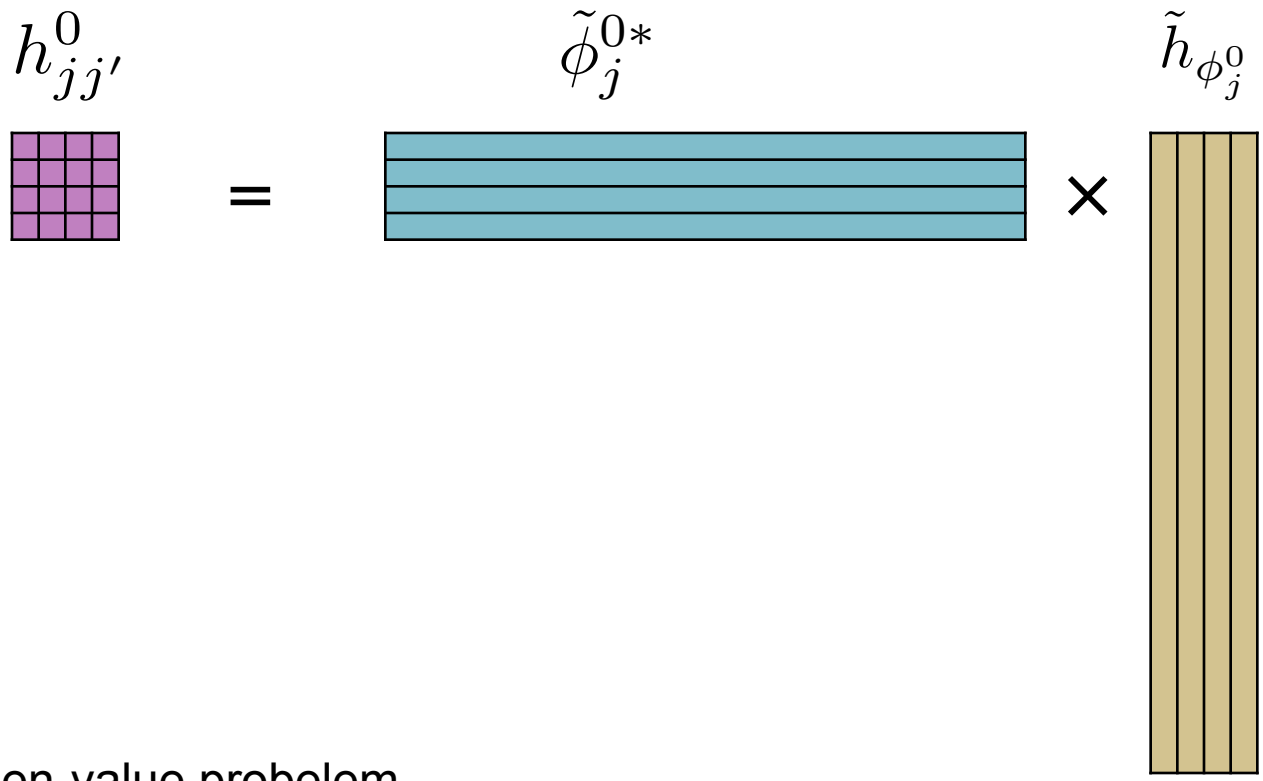
$\tilde{h}_{\phi_j^0} = \mathbf{H}\tilde{\phi}_j^0$

plane-wave index **G**

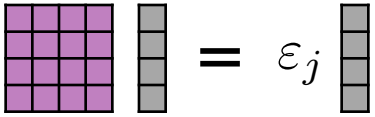


Davidson iterative solver

- Compute reduced Hamiltonian matrix

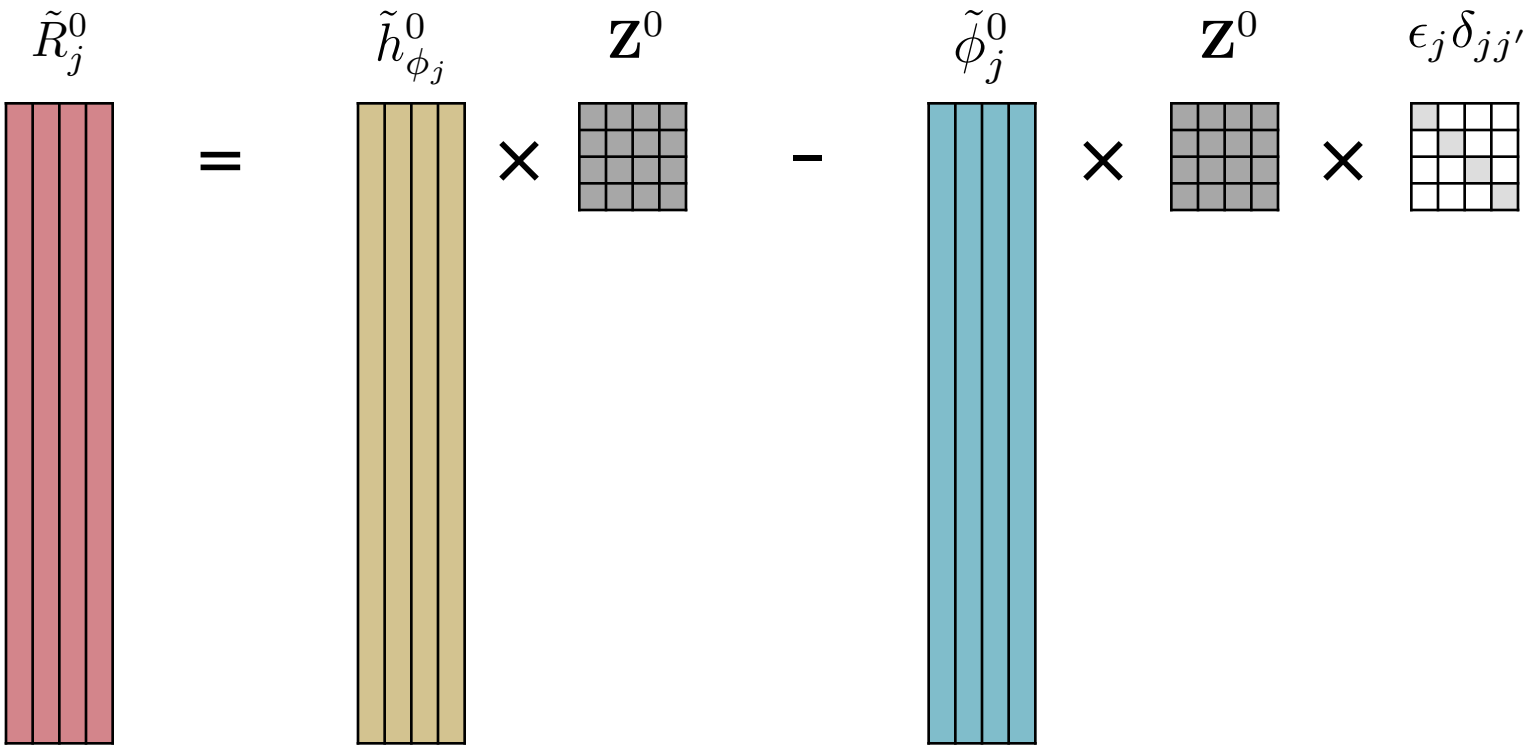


- Compute eigen-value probelem



Davidson iterative solver

- Compute residuals



- Apply preconditioner: $P\tilde{R}^0_j(\mathbf{G}) = (H_{\mathbf{GG}} - \varepsilon_j)^{-1}\tilde{R}^0_j(\mathbf{G})$

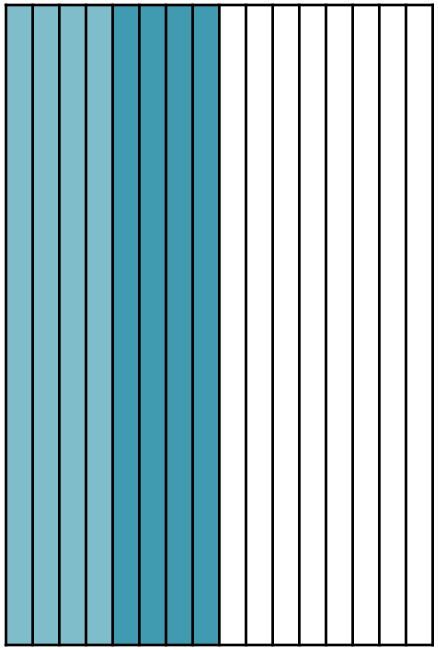
Davidson iterative solver

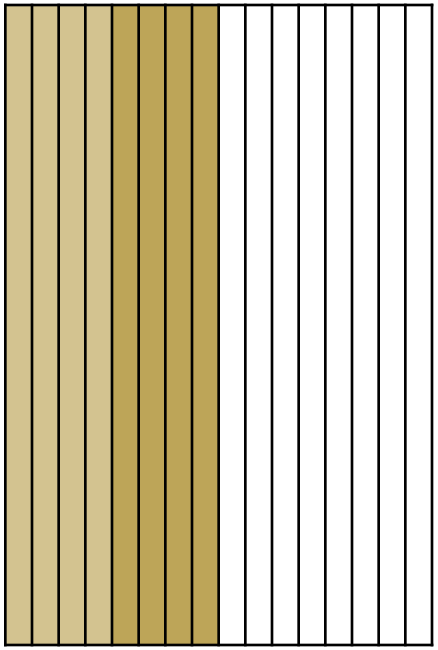
- Expand variational space and apply Hamiltonian to new basis functions

$$\tilde{\phi}_j^1 = \tilde{\phi}_j^0 \bigoplus P\tilde{R}_j^0$$

$$\tilde{h}_{\phi_j^1} = \mathbf{H}\tilde{\phi}_j^1$$

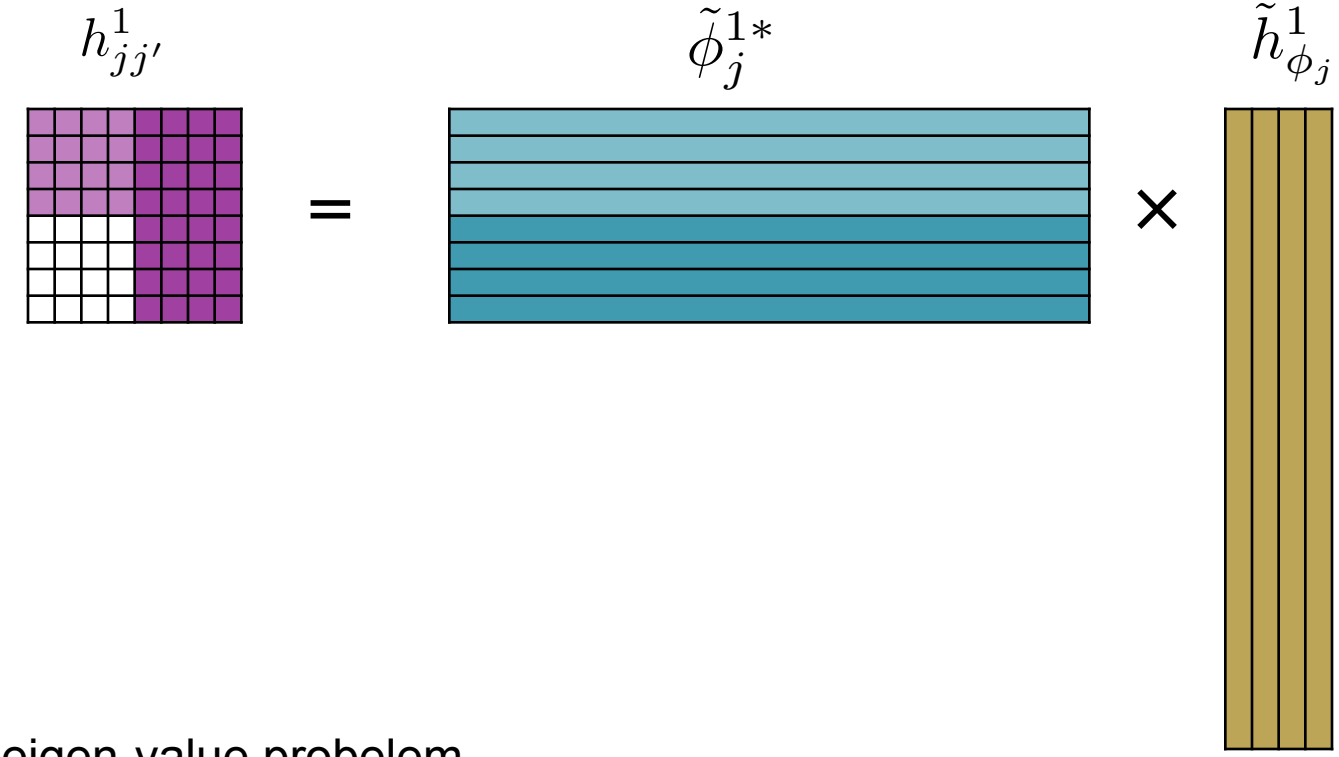
plane-wave index **G**



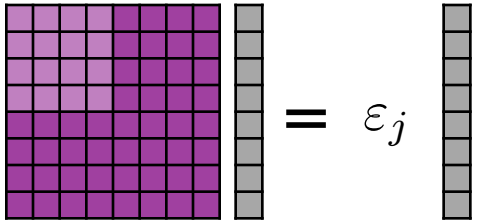


Davidson iterative solver

- Compute reduced Hamiltonian matrix

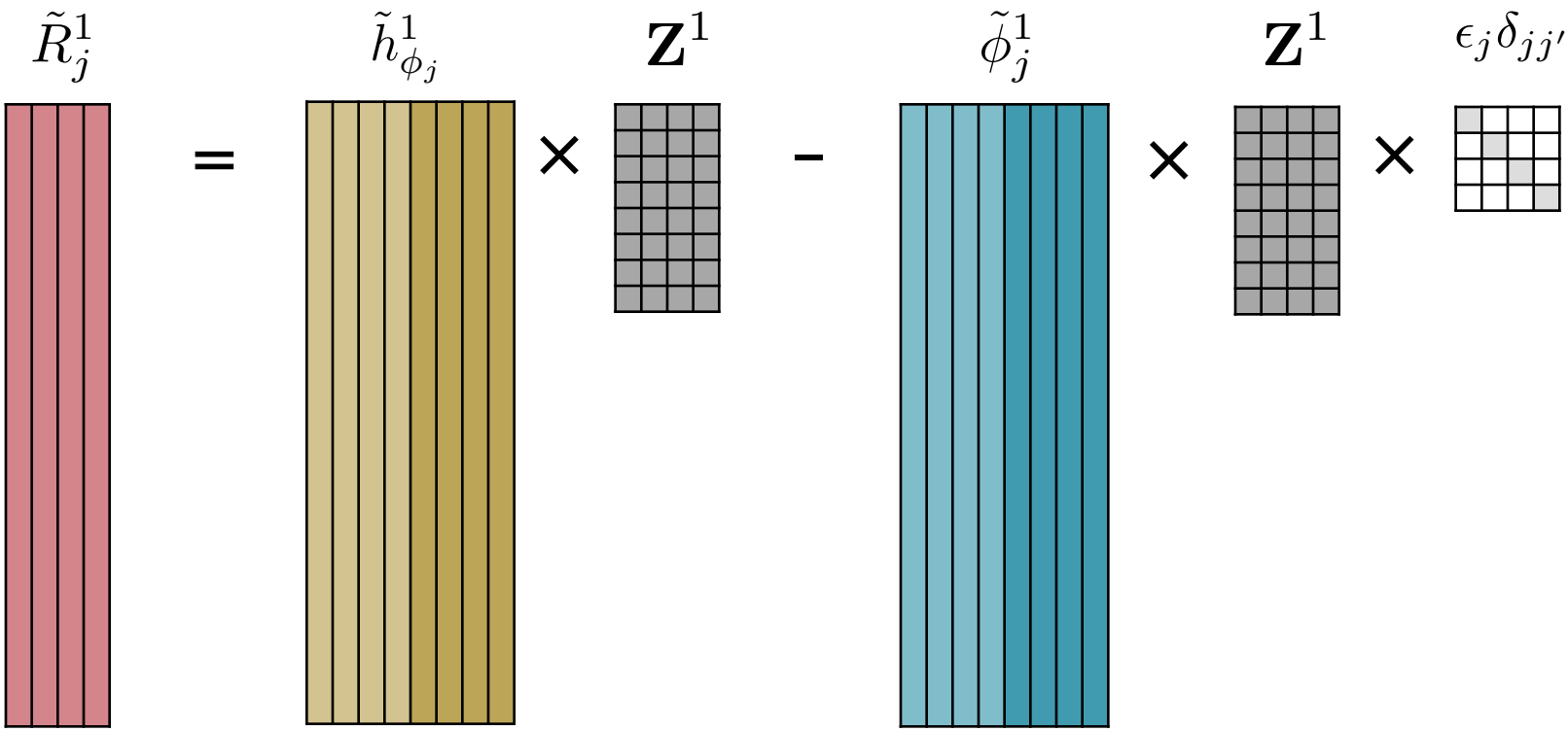


- Compute eigen-value probelem



Davidson iterative solver

- Compute residuals



- Apply preconditioner: $P\tilde{R}^1_j(\mathbf{G}) = (H_{\mathbf{GG}} - \varepsilon_j)^{-1}\tilde{R}^1_j(\mathbf{G})$

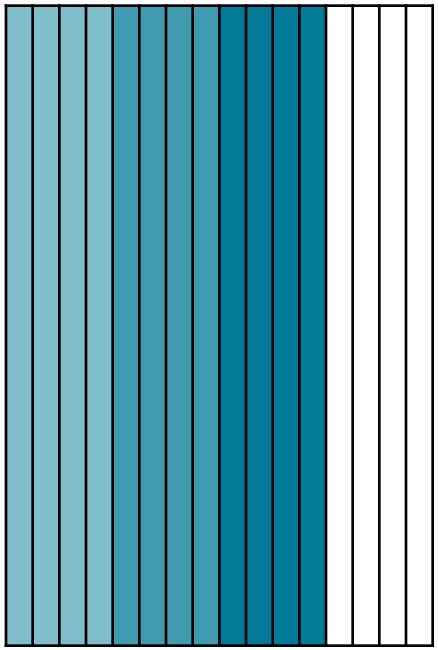
Davidson iterative solver

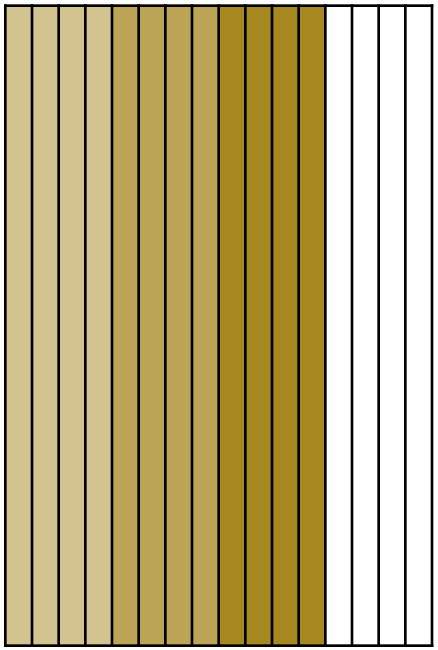
- Continue to expand variational space

$$\tilde{\phi}_j^2 = \tilde{\phi}_j^1 \bigoplus P\tilde{R}_j^1$$

$$\tilde{h}_{\phi_j^2} = \mathbf{H}\tilde{\phi}_j^2$$

plane-wave index **G**





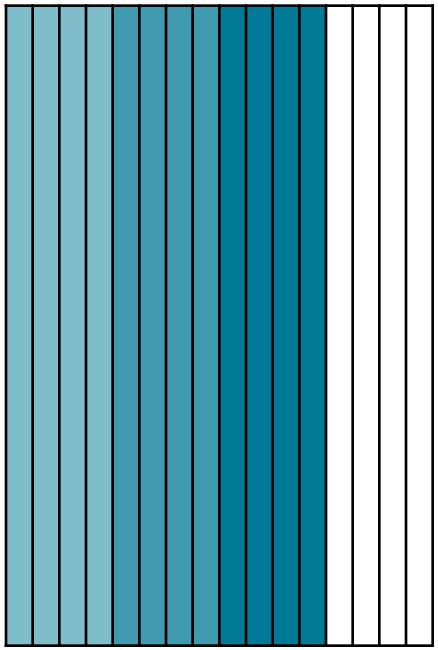
Davidson iterative solver

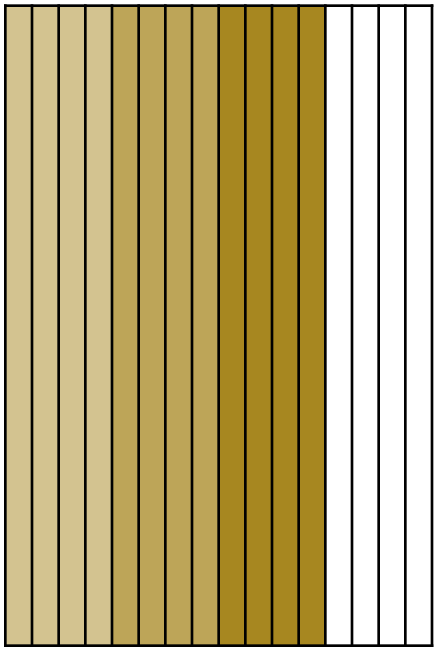
- Continue to expand variational space

$$\tilde{\phi}_j^2 = \tilde{\phi}_j^1 \bigoplus P\tilde{R}_j^1$$

$$\tilde{h}_{\phi_j^2} = \mathbf{H}\tilde{\phi}_j^2$$

plane-wave index **G**

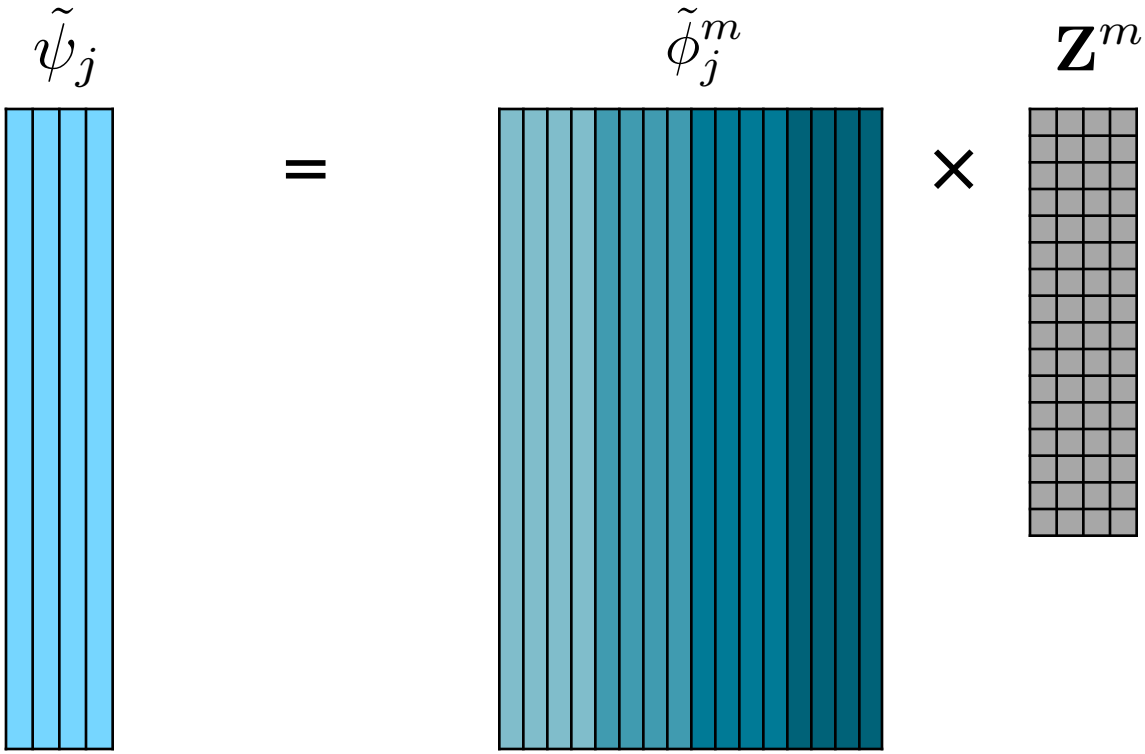




Iterate until the convergence (all residuals are zero) is reached

Davidson iterative solver

- Recompute the wave-functions

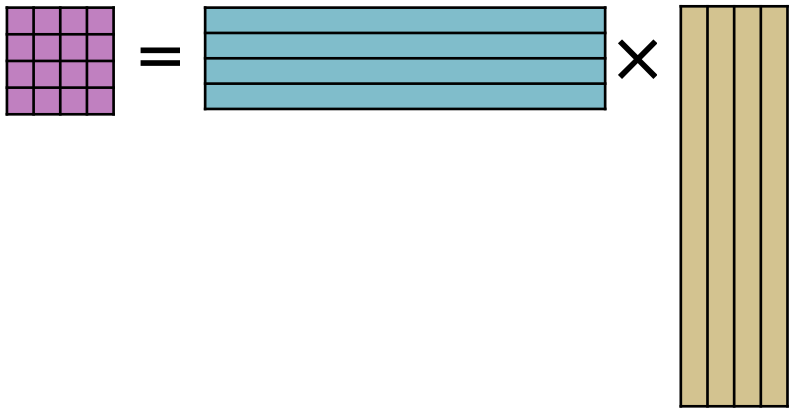


- Take ε_j from the last subspace diagonalization

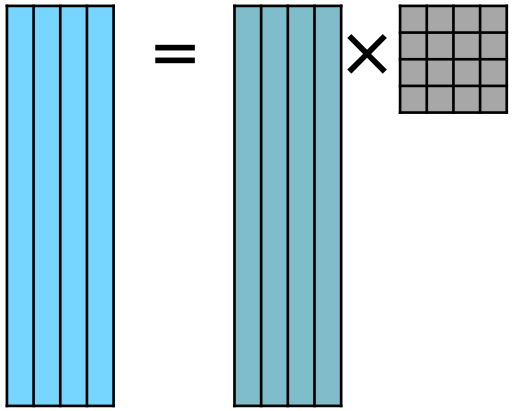
Part 2: Implementation details

Common operations

- FFTs (V_{loc} kernel, density summation)
- Subspace Hamiltonian construction



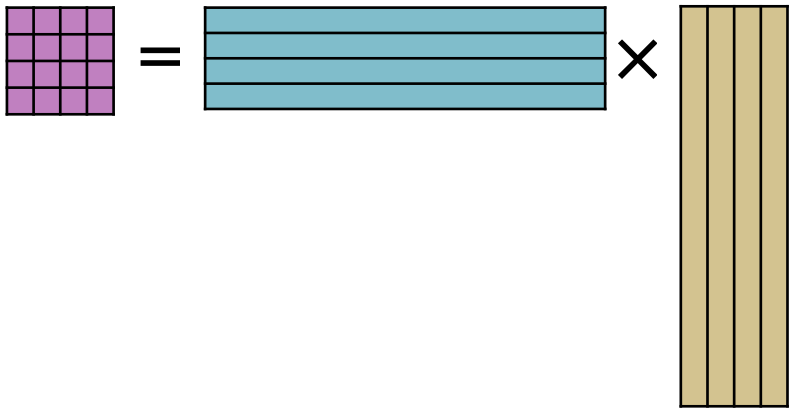
- Wave-functions and residuals update



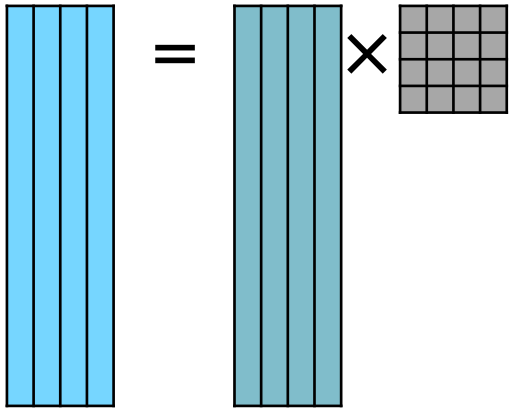
- Wave-function orthogonalization

Common operations

- FFTs (V_{loc} kernel, density summation)
- Subspace Hamiltonian construction



- Wave-functions and residuals update

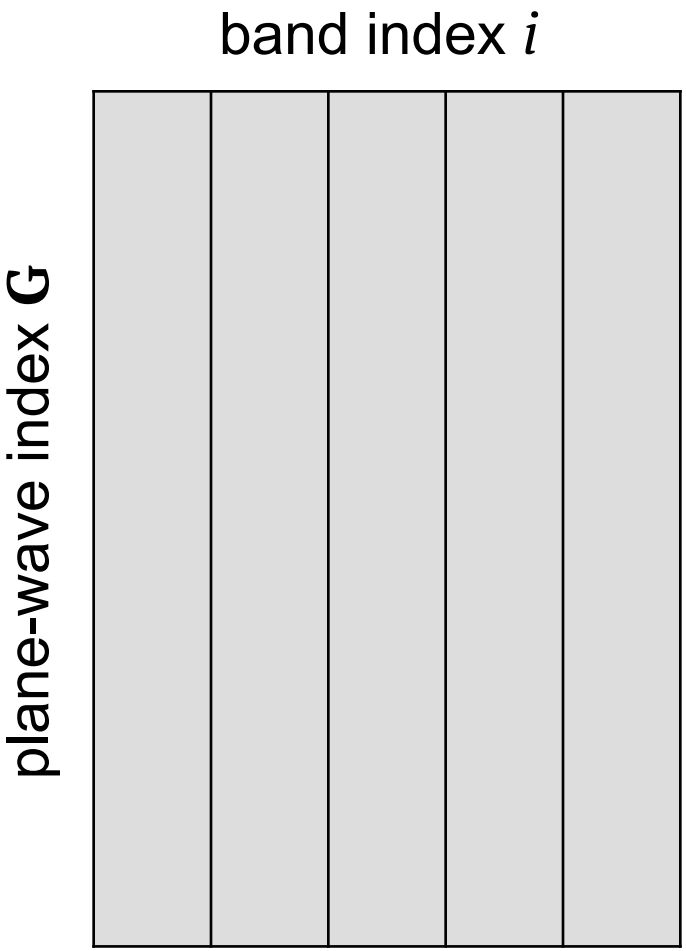


- Wave-function orthogonalization

Need scalable parallel implementation!

Wave-function storage and distribution

Wave-functions $\Psi_i(\mathbf{G})$ are stored as a matrix: each column of the matrix represents a single wave-function.



Number of bands:
~100–10K

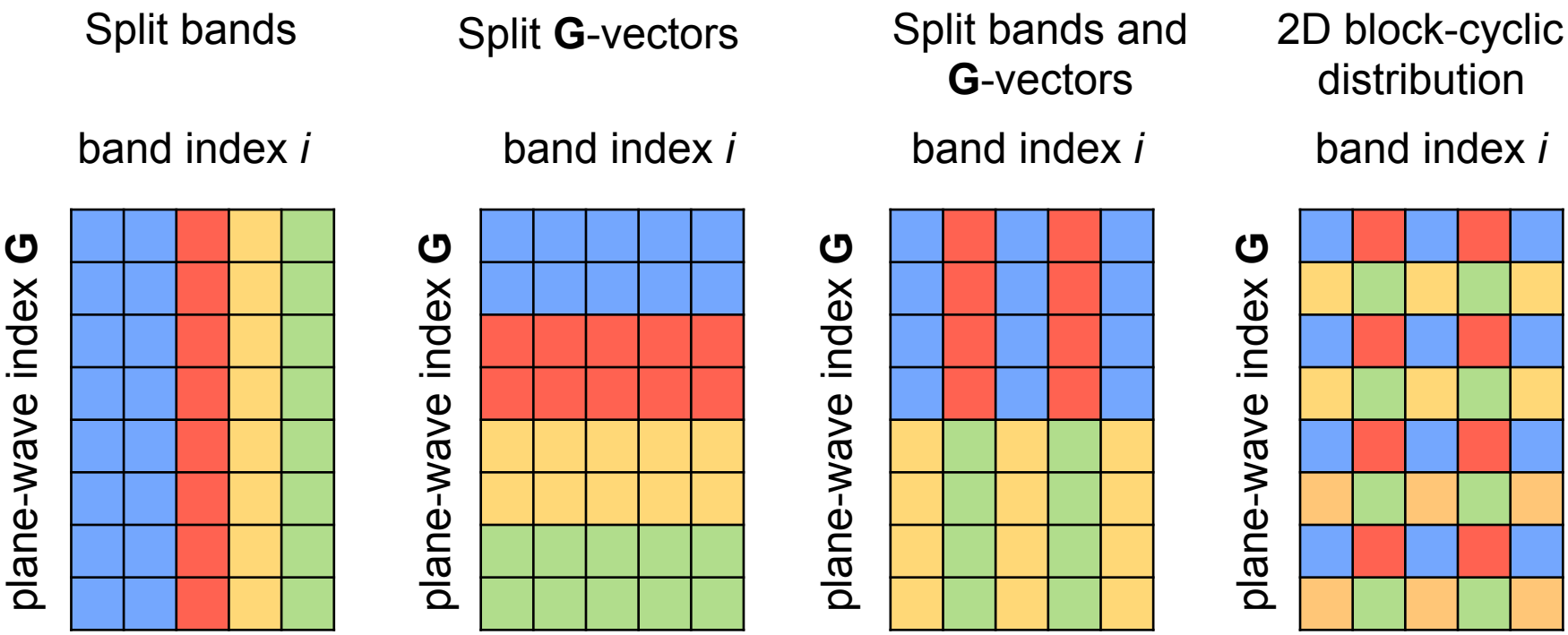
Number of plane-waves:
~100K–10M

Wave-function storage and distribution

How to distribute the matrix of plane-wave coefficients?

Wave-function storage and distribution

How to distribute the matrix of plane-wave coefficients?



Wave-function storage and distribution

How to distribute the matrix of plane-wave coefficients?

Split **G**-vectors

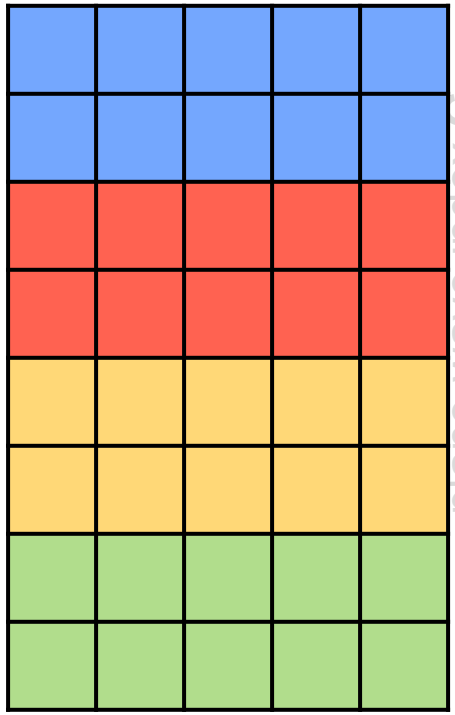
Split bands

band index *i*

plane-wave index **G**

band index *i*

plane-wave index **G**



Split bands and **G**-vectors

band index *i*

plane-wave index **G**

2D block-cyclic distribution

band index *i*

plane-wave index **G**

FFT kernel

```
do i=1, Nb
```

$$\Psi_i(\mathbf{G}) \xrightarrow{FFT^{-1}} \Psi_i(\mathbf{r}) \rightarrow \Psi_i(\mathbf{r}) \cdot V_{loc}(\mathbf{r}) \xrightarrow{FFT} [\Psi_i V](\mathbf{G})$$

```
enddo
```

FFT kernel

```
do i=1, Nb
```

$$\Psi_i(\mathbf{G}) \xrightarrow{FFT^{-1}} \Psi_i(\mathbf{r}) \rightarrow \Psi_i(\mathbf{r}) \cdot V_{loc}(\mathbf{r}) \xrightarrow{FFT} [\Psi_i V](\mathbf{G})$$

```
enddo
```

Wave-functions in the plane-wave domain are defined inside a sphere with a given energy cutoff:

$$\Psi_i(\mathbf{G}) \neq 0 \text{ for } |\mathbf{G}| \leq G_{max}; \ \frac{G_{max}^2}{2} = E_{cut}$$

FFT kernel

```
do i=1, Nb
```

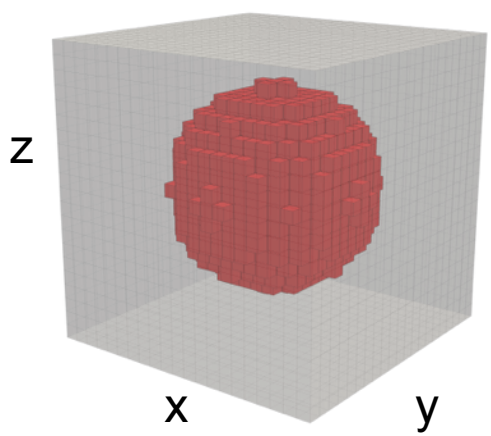
$$\Psi_i(\mathbf{G}) \xrightarrow{FFT^{-1}} \Psi_i(\mathbf{r}) \rightarrow \Psi_i(\mathbf{r}) \cdot V_{loc}(\mathbf{r}) \xrightarrow{FFT} [\Psi_i V](\mathbf{G})$$

```
enddo
```

Wave-functions in the plane-wave domain are defined inside a sphere with a given energy cutoff:

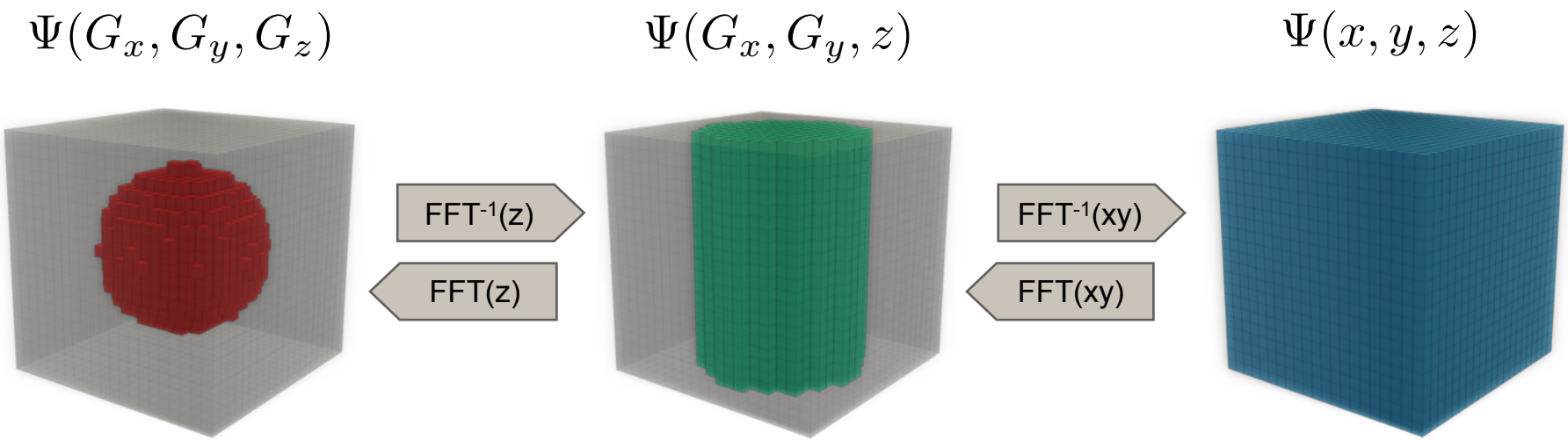
$$\Psi_i(\mathbf{G}) \neq 0 \text{ for } |\mathbf{G}| \leq G_{max}; \ \frac{G_{max}^2}{2} = E_{cut}$$

FFT box for the V_{loc} kernel typically circumscribes a sphere of radius $2G_{max}$.



FFT decomposition to 1D and 2D

The full 3D transformation is decomposed into a 1D transformation along the z-direction and a 2D transformation in the xy-plane:

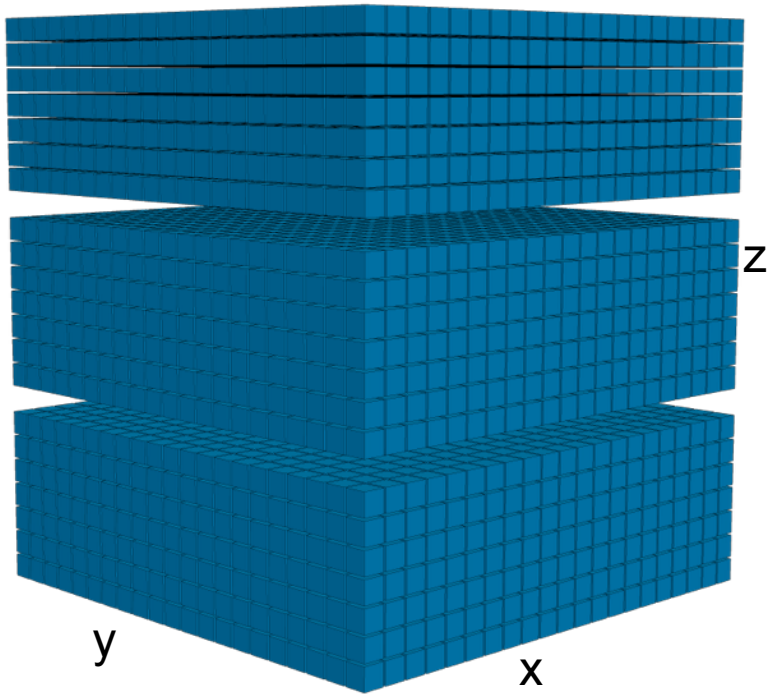


FFT parallelization

FFT parallelization

- In real space

z-dimension of the FFT buffer
is split among MPI ranks



MPI rank#0

MPI rank#1

MPI rank#2

FFT parallelization

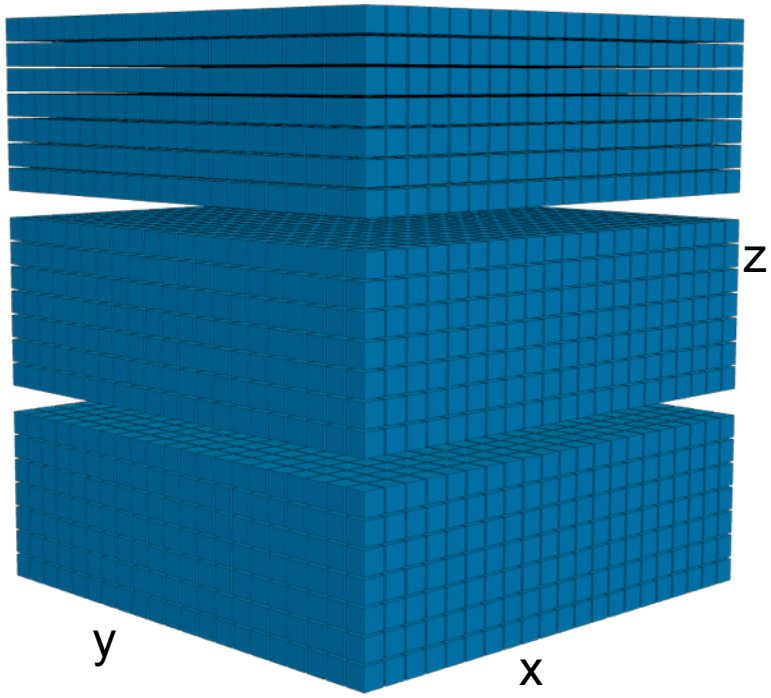
- In real space

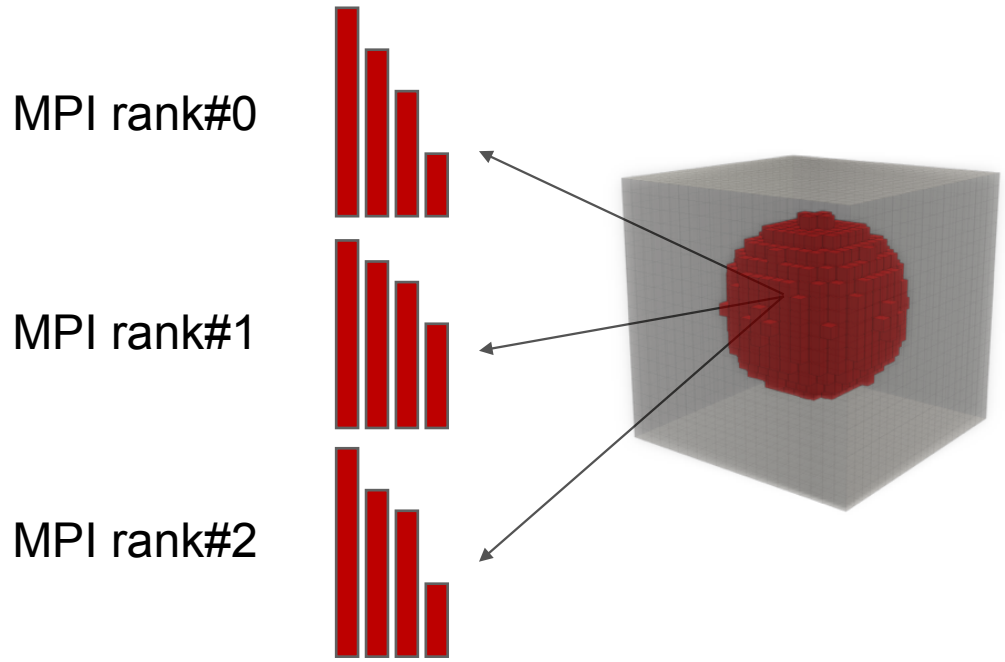
z-dimension of the FFT buffer is split among MPI ranks

- In reciprocal space

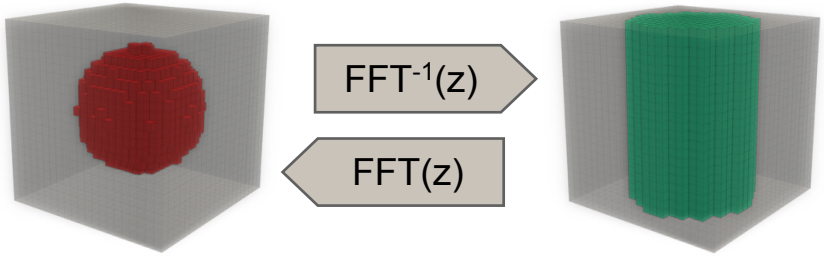
z-sticks of the G-vector sphere are distributed among MPI ranks such that:

- local number of z-sticks is roughly equivalent
- sum of lengths of z-sticks (local number of G-vectors) is roughly equivalent

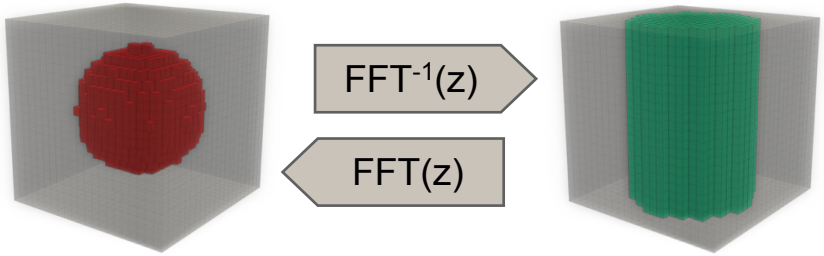




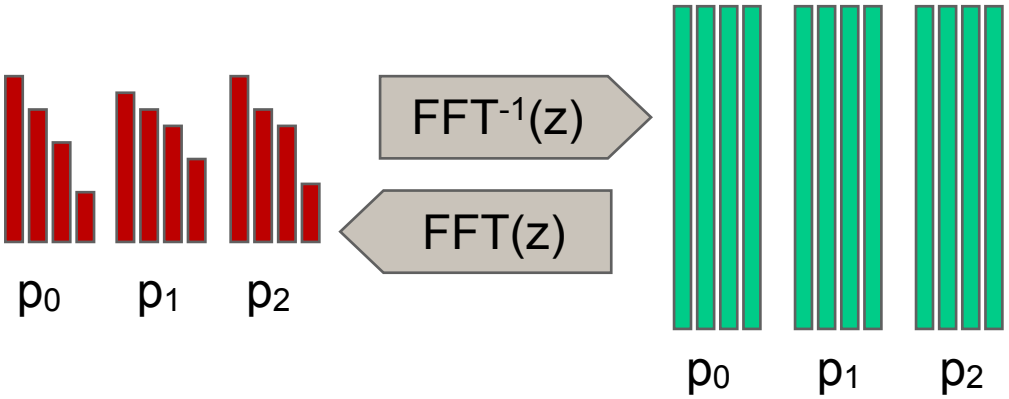
Parallel transformation of z-sticks



Parallel transformation of z-sticks

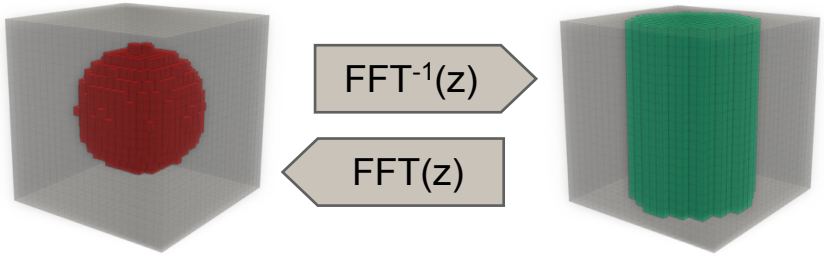


- each rank executes 1D transformations of the local fraction of z-sticks

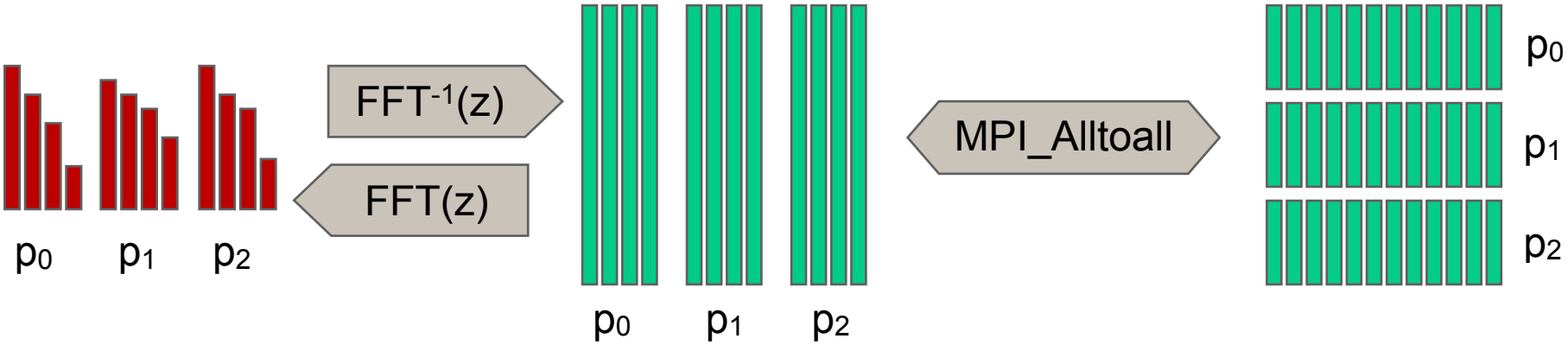


{full-z, partial-xy} distribution of a cylinder

Parallel transformation of z-sticks



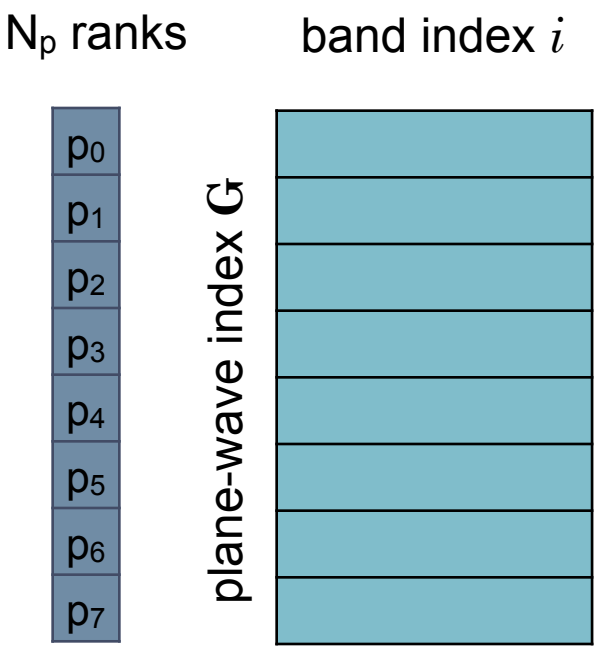
- each rank executes 1D transformations of the local fraction of z-sticks
- z-sticks are swapped between MPI ranks using MPI_Alltoall



{full-z, partial-xy} distribution of a cylinder

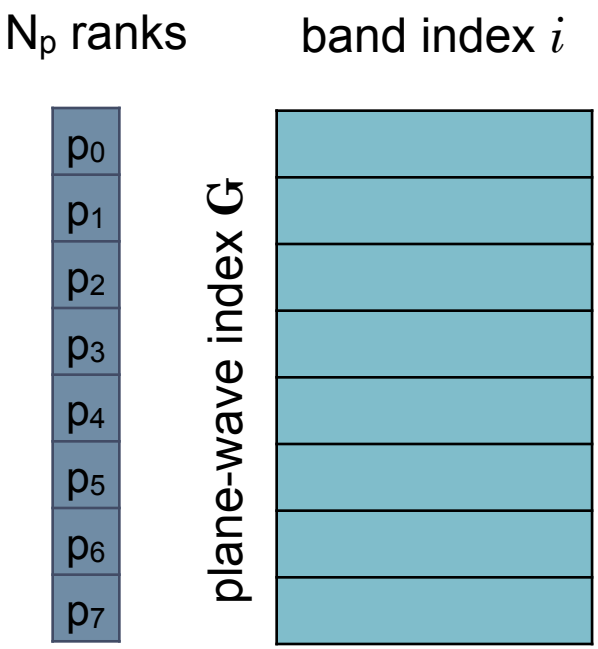
{partial-z, full-xy} distribution of a cylinder

Data redistribution



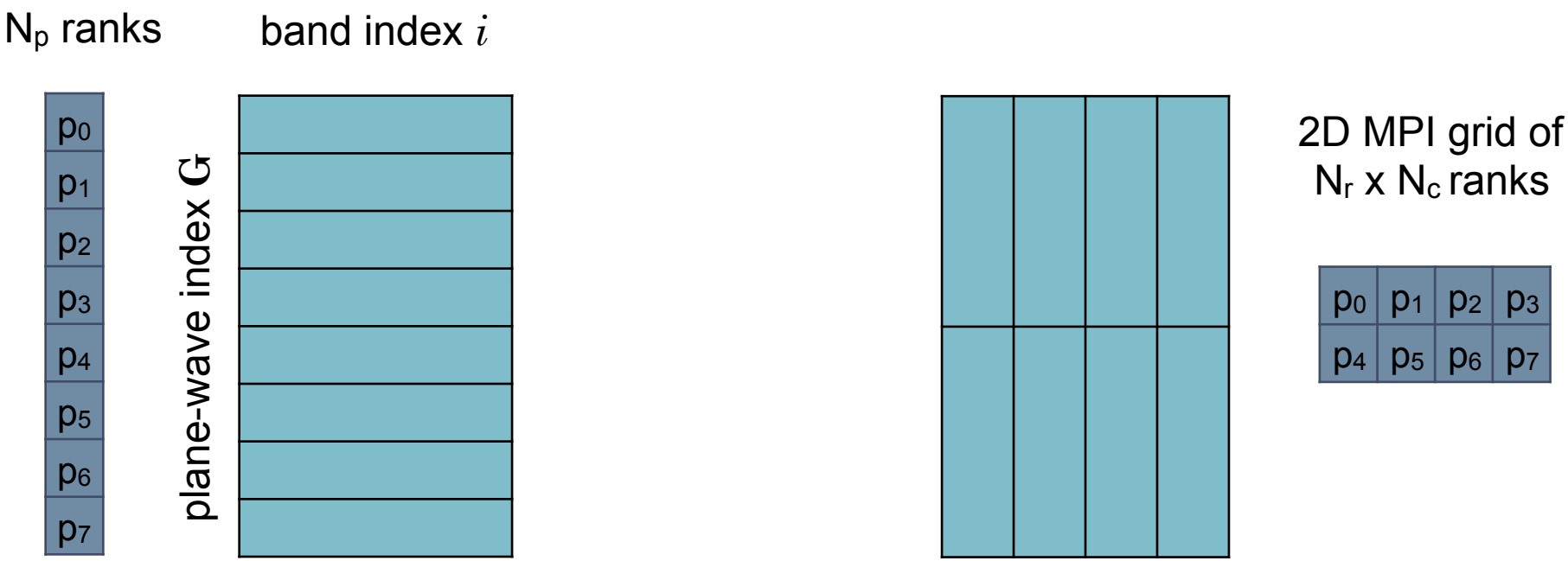
Data redistribution

- Using all available MPI ranks for a single FFT is not efficient!



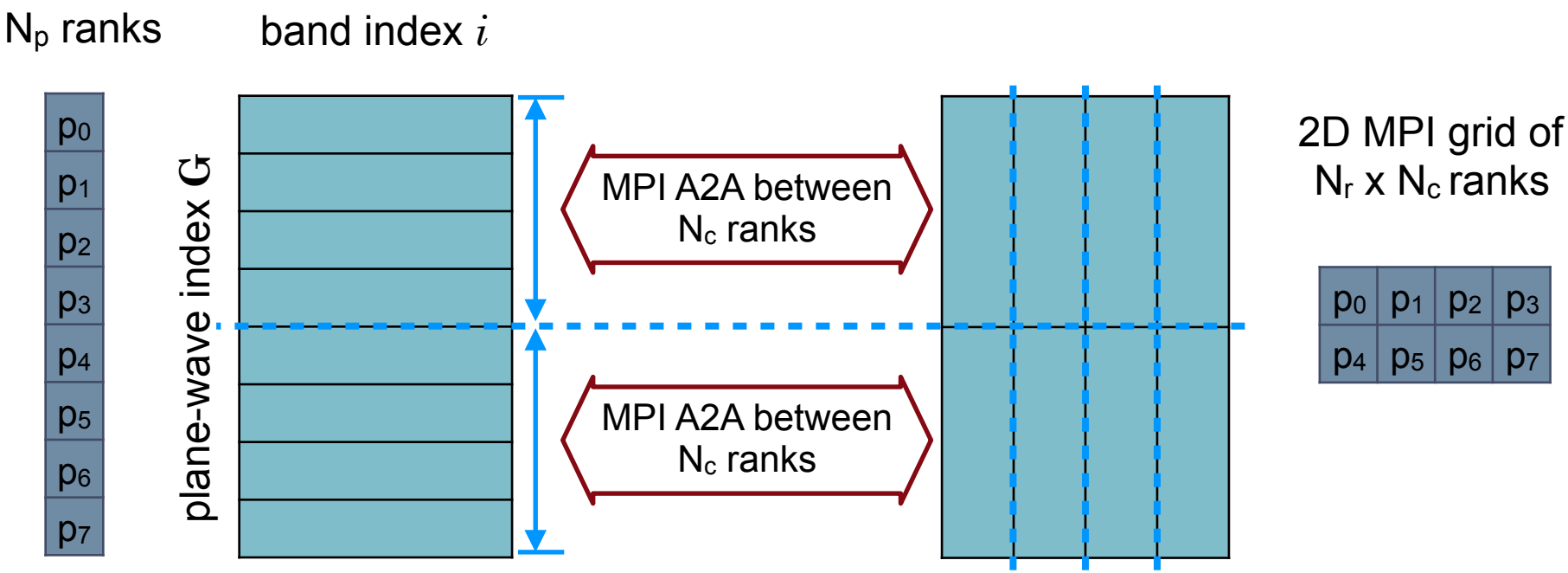
Data redistribution

- Using all available MPI ranks for a single FFT is not efficient!
- Remap wave-functions to a 2D MPI grid where one dimension is dedicated to a parallel FFT and second dimension is used to distribute wave-function band index



Data redistribution

- Using all available MPI ranks for a single FFT is not efficient!
- Remap wave-functions to a 2D MPI grid where one dimension is dedicated to a parallel FFT and second dimension is used to distribute wave-function band index



Groups of N_c ranks contain the “fat” slab of plane-wave coefficients for all bands

FFT Vloc kernel

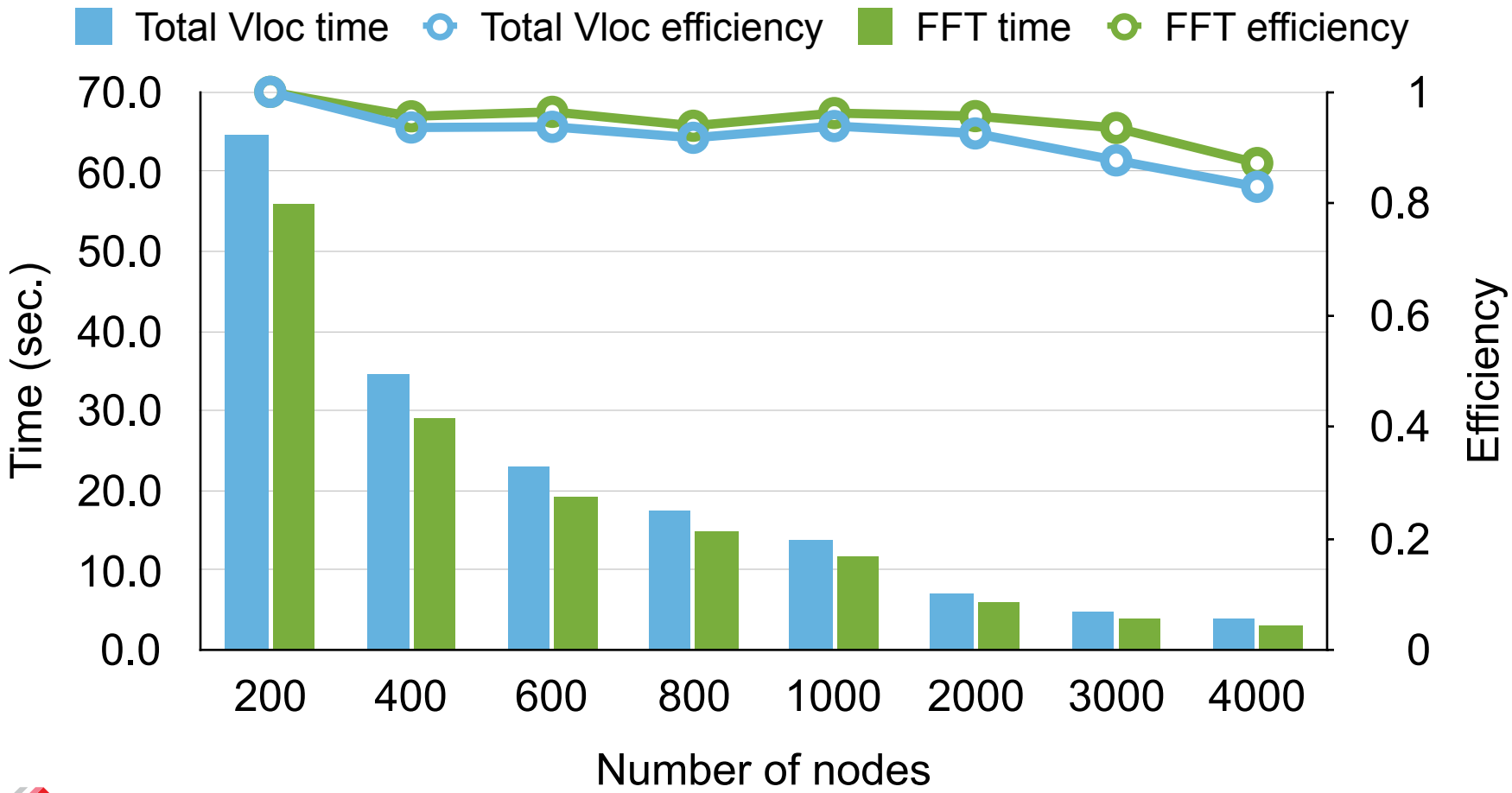
- Change wave-function distribution from “slab” to “2D grid” (MPI_A2A between N_c ranks)
- Execute 1D backward FFTs along z for the local set of G-vector sticks
- swap z-columns (MPI_A2A between N_r ranks)

- execute 2D backward FFTs in the xy-plane
- multiply by effective potential
- execute 2D forward FFTs in the xy-plane

- swap z-columns (MPI_A2A between N_r ranks)
- Execute 1D forward FFTs along z for the local set of z-sticks
- Change wave-function distribution from “2D grid” to “slab” (MPI_A2A between N_c ranks)

FFT kernel benchmark on Piz Daint

Number of bands is 3000, number of times V_{loc} is applied is 4, number of G-vectors is ~9.2M, FFT grid size is 540×540×540



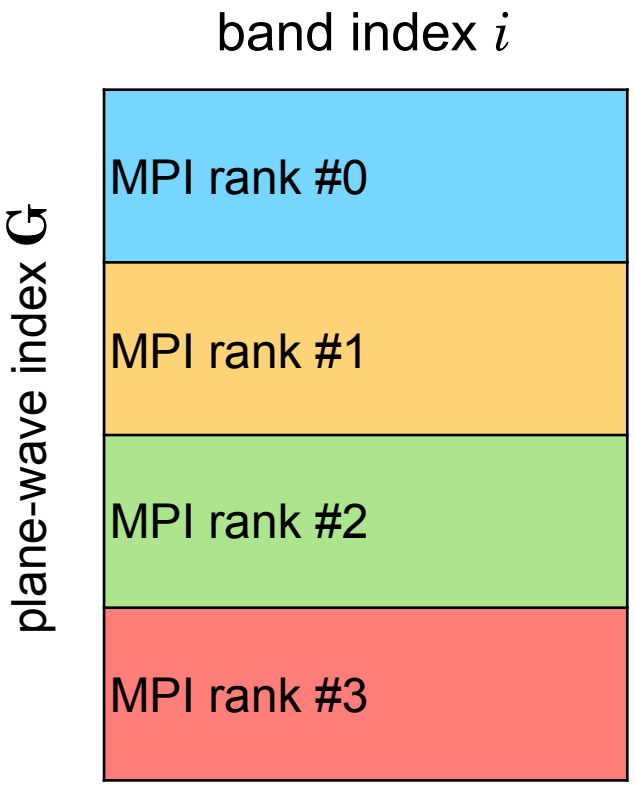
Inner product

$$O_{ij} = \sum_{\mathbf{G}} \Psi_i^*(\mathbf{G})\Psi_j(\mathbf{G})$$

Inner product

$$O_{ij} = \sum_{\mathbf{G}} \Psi_i^*(\mathbf{G})\Psi_j(\mathbf{G})$$

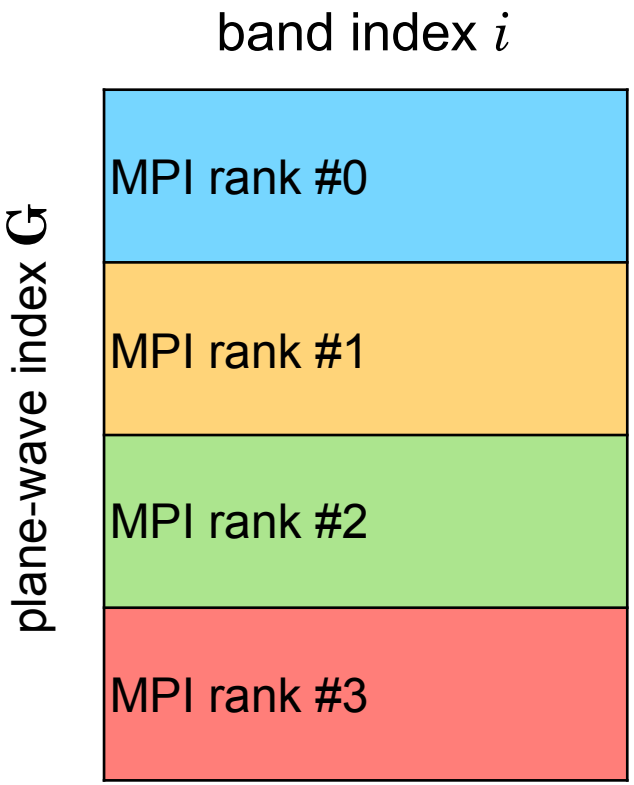
Slab data distribution of $\Psi_i(\mathbf{G})$



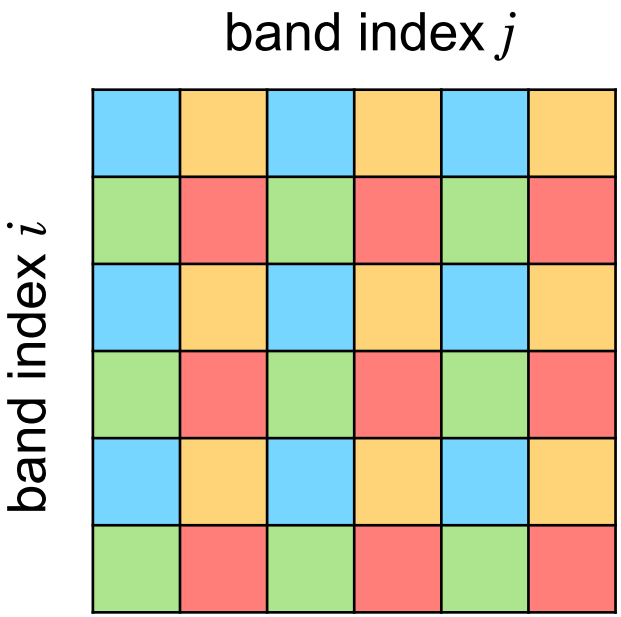
Inner product

$$O_{ij} = \sum_{\mathbf{G}} \Psi_i^*(\mathbf{G})\Psi_j(\mathbf{G})$$

Slab data distribution of $\Psi_i(\mathbf{G})$



2D block-cyclic distribution of O_{ij}

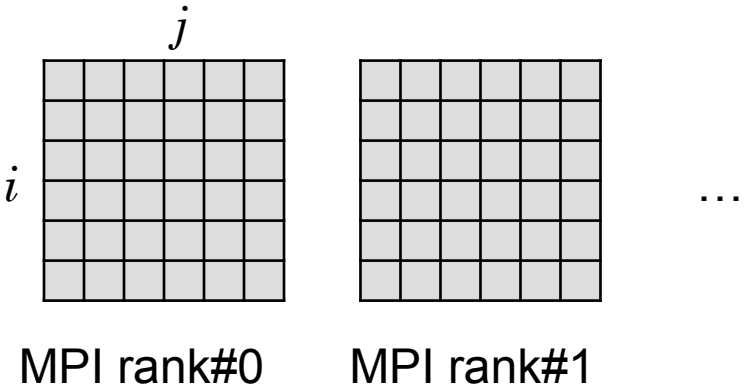


Inner product: simple implementation

Inner product: simple implementation

1. Each rank computes a contribution to the global matrix

$$O_{ij}^{p} = \sum_{\mathbf{G}}^{N_{\mathbf{G}}^{p}} \Psi_i^*(\mathbf{G})\Psi_j(\mathbf{G})$$



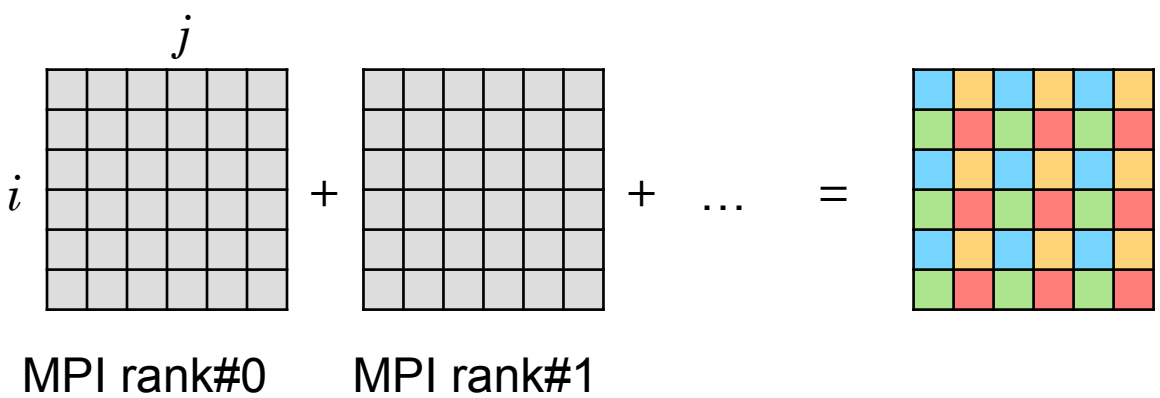
Inner product: simple implementation

1. Each rank computes a contribution to the global matrix

$$O_{ij}^{p} = \sum_{\mathbf{G}}^{N_{\mathbf{G}}^{p}} \Psi_i^*(\mathbf{G})\Psi_j(\mathbf{G})$$

2. MPI_Allreduce of global matrix is performed

$$O_{ij} = \sum_{p=1}^{N_p} O_{ij}^{p}$$



Inner product: simple implementation

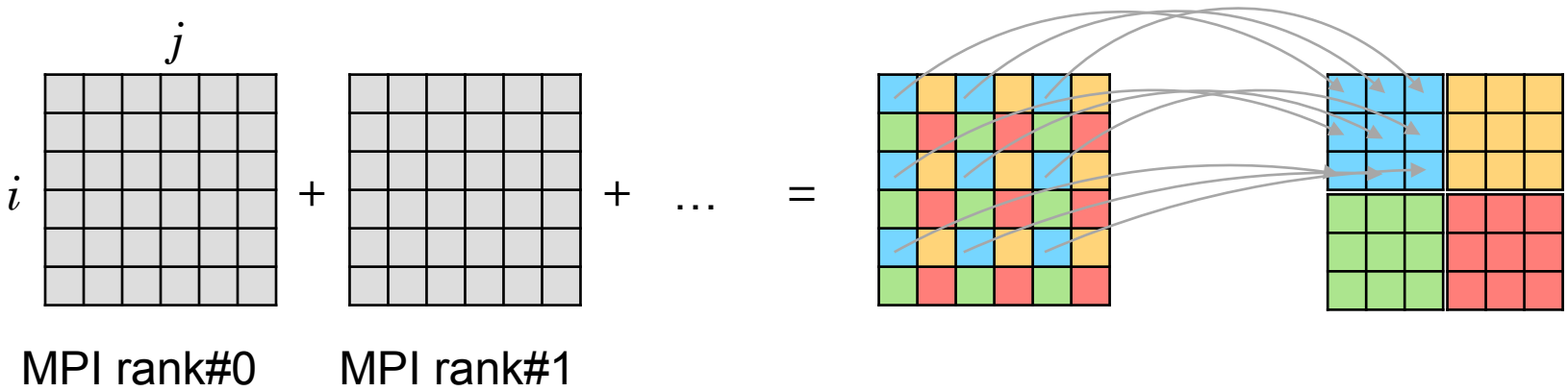
1. Each rank computes a contribution to the global matrix

$$O_{ij}^{p} = \sum_{\mathbf{G}}^{N_{\mathbf{G}}^{p}} \Psi_i^*(\mathbf{G})\Psi_j(\mathbf{G})$$

2. MPI_Allreduce of global matrix is performed

$$O_{ij} = \sum_{p=1}^{N_p} O_{ij}^{p}$$

3. Each rank picks the local fraction of matrix elements from O_{ij}



Inner product: simple implementation

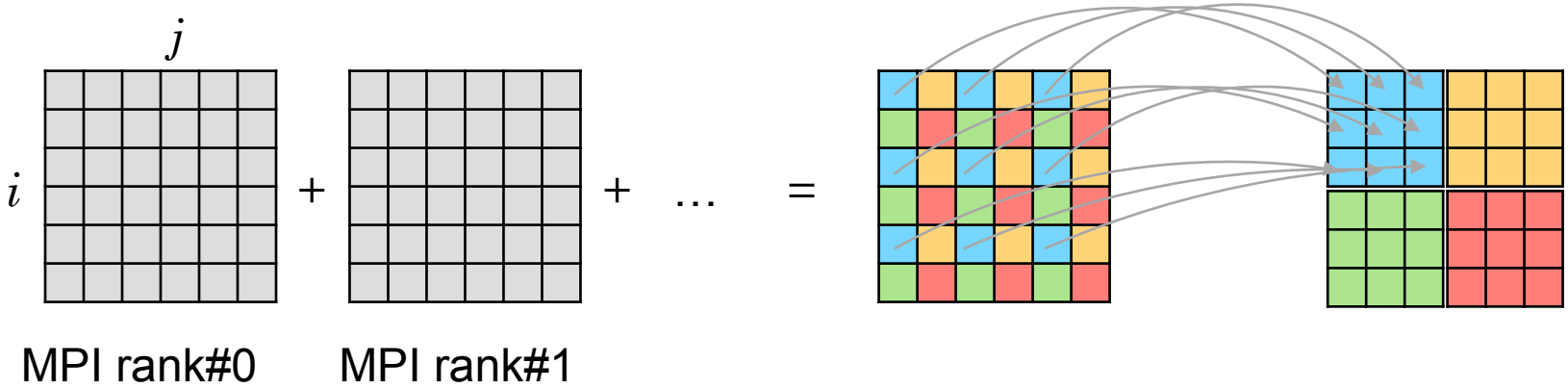
1. Each rank computes a contribution to the global matrix

$$O_{ij}^{p} = \sum_{\mathbf{G}}^{N_{\mathbf{G}}^{p}} \Psi_i^*(\mathbf{G})\Psi_j(\mathbf{G})$$

2. MPI_Allreduce of global matrix is performed

$$O_{ij} = \sum_{p=1}^{N_p} O_{ij}^{p}$$

3. Each rank picks the local fraction of matrix elements from O_{ij}



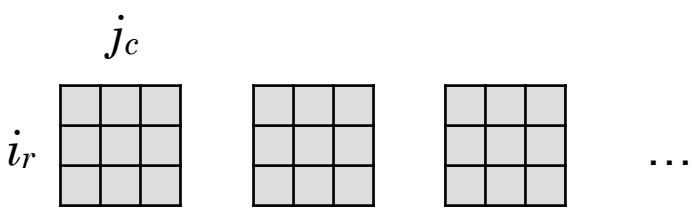
Requires MPI_Allreduce of global matrix!
Each MPI rank receives a lot of unnecessary data!

Inner product: “reduce-to-one” implementation

Inner product: “reduce-to-one” implementation

1. Each rank computes a contribution to the local panel of (p_r, p_c) Cartesian rank

$$O^p_{i_r j_c} = \sum_{\mathbf{G}}^{N^p_{\mathbf{G}}} \Psi^*_{i_r}(\mathbf{G})\Psi_{j_c}(\mathbf{G})$$



MPI rank#0 MPI rank#1 MPI rank#2

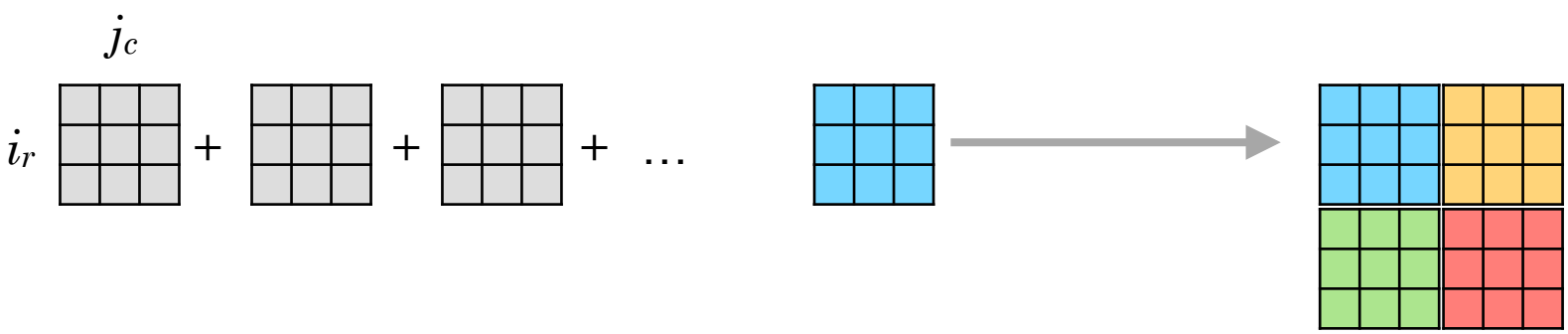
Inner product: “reduce-to-one” implementation

1. Each rank computes a contribution to the local panel of (p_r, p_c) Cartesian rank

$$O^p_{i_r j_c} = \sum_{\mathbf{G}}^{N^p_{\mathbf{G}}} \Psi^*_{i_r}(\mathbf{G})\Psi_{j_c}(\mathbf{G})$$

2. MPI_Reduce of local panel is performed to (p_r, p_c) Cartesian rank

$$O_{i_r j_c} = \sum_{p=1}^{N_p} O^p_{i_r j_c}$$



MPI rank#0 MPI rank#1 MPI rank#2

Inner product: “reduce-to-one” implementation

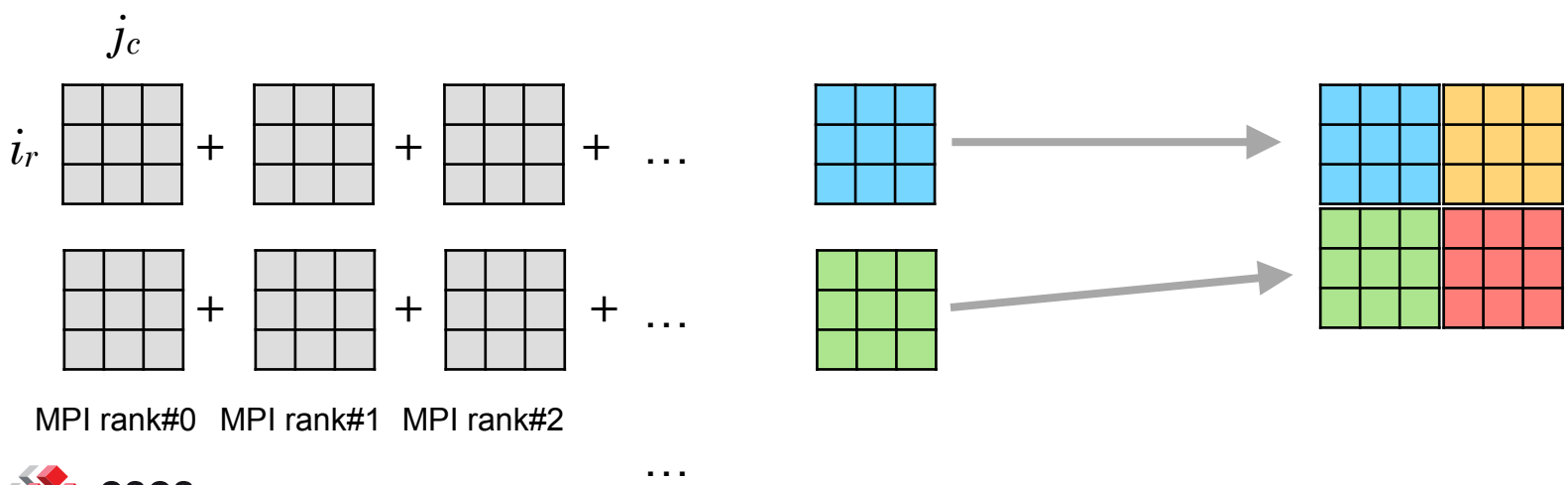
1. Each rank computes a contribution to the local panel of (p_r, p_c) Cartesian rank

$$O^p_{i_r j_c} = \sum_{\mathbf{G}}^{N^p_{\mathbf{G}}} \Psi^*_{i_r}(\mathbf{G})\Psi_{j_c}(\mathbf{G})$$

2. MPI_Reduce of local panel is performed to (p_r, p_c) Cartesian rank

$$O_{i_r j_c} = \sum_{p=1}^{N_p} O^p_{i_r j_c}$$

3. Steps 1-2 are repeated for all Cartesians rank of a 2D BLACS grid



Inner product: “reduce-to-one” implementation

- synchronous
  - zgemm
  - MPI_Reduce

- asynchronous
  - MPI_Wait
  - zgemm
  - MPI_Ireduce

Inner product: “reduce-to-one” implementation

- synchronous
  - zgemm
  - MPI_Reduce

- asynchronous
  - MPI_Wait
  - zgemm
  - MPI_Ireduce

A lot of MPI_Reduce!
The resulting sub-matrices are small!

Inner product: “block-allreduce” implementation

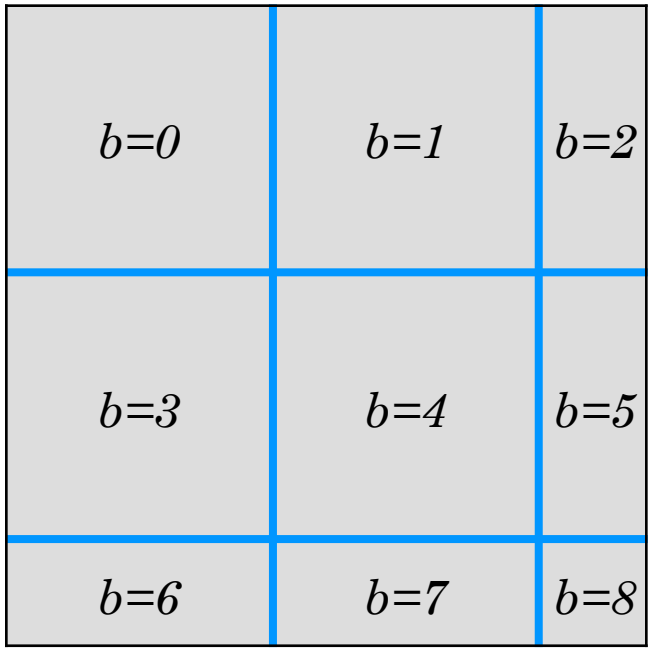
1. Each rank computes a contribution to the block of global matrix

$$O^p_{i_b j_b} = \sum_{\mathbf{G}}^{N^p_{\mathbf{G}}} \Psi^*_{i_b}(\mathbf{G})\Psi_{j_b}(\mathbf{G})$$

2. MPI_Allreduce of block is performed

$$O_{i_b j_b} = \sum_{p=1}^{N_p} O^p_{i_b j_b}$$

3. Each rank picks the local fraction of matrix elements from $O_{i_b j_b}$ block



Inner product: “block-allreduce” implementation

1. Each rank computes a contribution to the block of global matrix

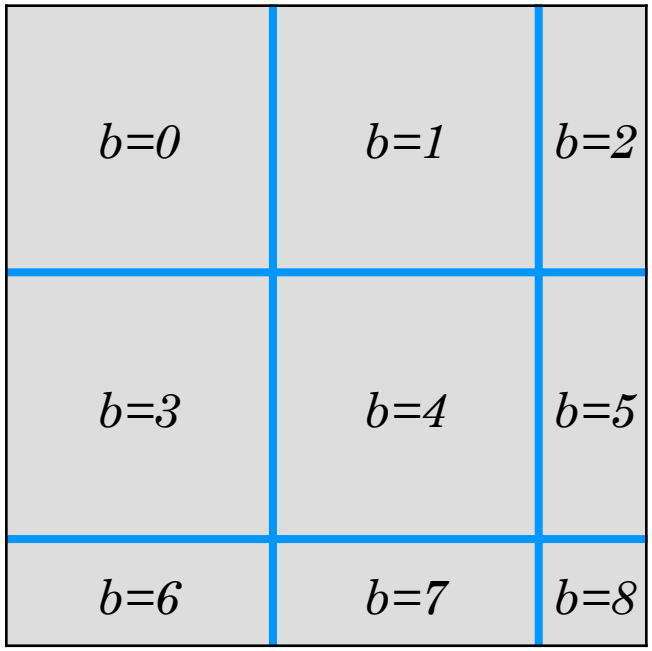
$$O^p_{i_b j_b} = \sum_{\mathbf{G}}^{N^p_{\mathbf{G}}} \Psi^*_{i_b}(\mathbf{G})\Psi_{j_b}(\mathbf{G})$$

2. MPI_Allreduce of block is performed

$$O_{i_b j_b} = \sum_{p=1}^{N_p} O^p_{i_b j_b}$$

3. Each rank picks the local fraction of matrix elements from $O_{i_b j_b}$ block

4. Steps 1-3 are repeated for all blocks



Inner product: “block-allreduce” implementation

- synchronous
  - zgemm
  - MPI_Allreduce
  - update local panels

- asynchronous
  - MPI_Wait
  - update local panels
  - zgemm
  - MPI_Ireduce

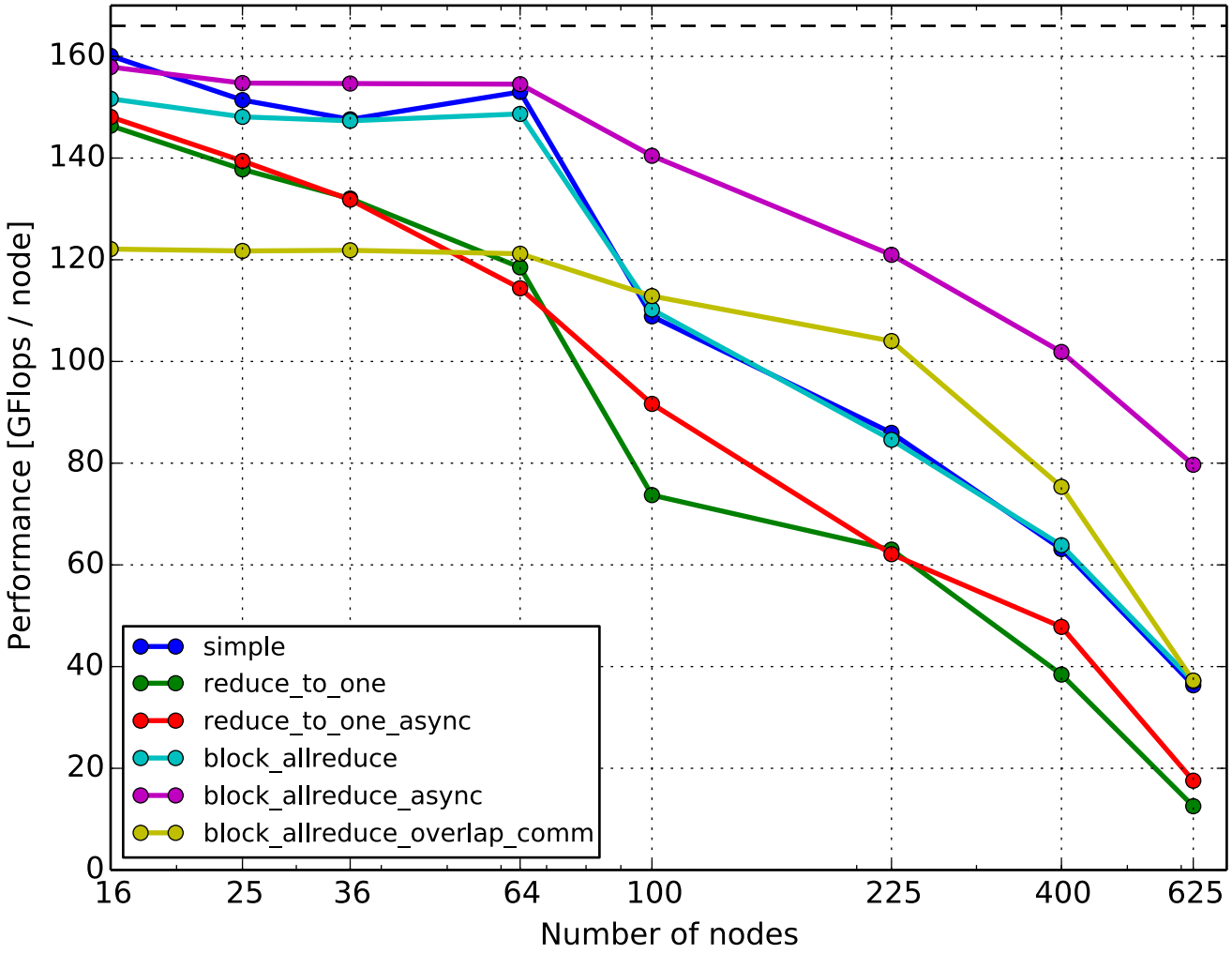
- overlap computation and communication
  - single OMP thread is executing MPI_Allreduce and update of local panels
  - remaining OMP threads execute zgemm

Example: communication and computation overlap

```
omp_set_nested(1); /* allow nested OMP */
int nt = omp_get_max_threads(); /* get total number of threds */
#pragma omp parallel num_threads(2) shared(buf_state) /* spawn two threds */
{
    if (omp_get_thread_num() == 1) { /* thread #1 executes zgemms */
        int s{0};
        omp_set_num_threads(nt - 1); /* set number of nested threads to nt - 1 */
        for (...) { /* loop over blocks */
            int state = 1;
            while (state == 1) { /* wait for the release of the buffer */
                #pragma omp atomic read
                state = buf_state[s % 2]; /* read from shared varray */
            }
            zgemm(...,buf[s % 2]); /* execute local zgemm */
            #pragma omp atomic write
            buf_state[s % 2] = 1; /* lock the buffer */
            s++;
        }
    } else { /* thread #0 is doing communication */
        int s{0};
        for (...) { /* loop over blocks */
            int state = 0;
            while (state == 0) { /* wait for the lock of the buffer */
                #pragma omp atomic read
                state = buf_state[s % 2]; /* read from shared array */
            }
            MPI_Allreduce(...,buf[s % 2]); /* execute global reduction */
            /* do something with the result, for example, store it */
            #pragma omp atomic write
            buf_state[s % 2] = 0; /* release the buffer */
            s++;
        }
    }
}
omp_set_nested(0); /* forbid nested */
```

Inner product kernel benchmark

Piz Daint, Intel SB (8 CPU cores) M=1’000 N=1’000 K=300’000

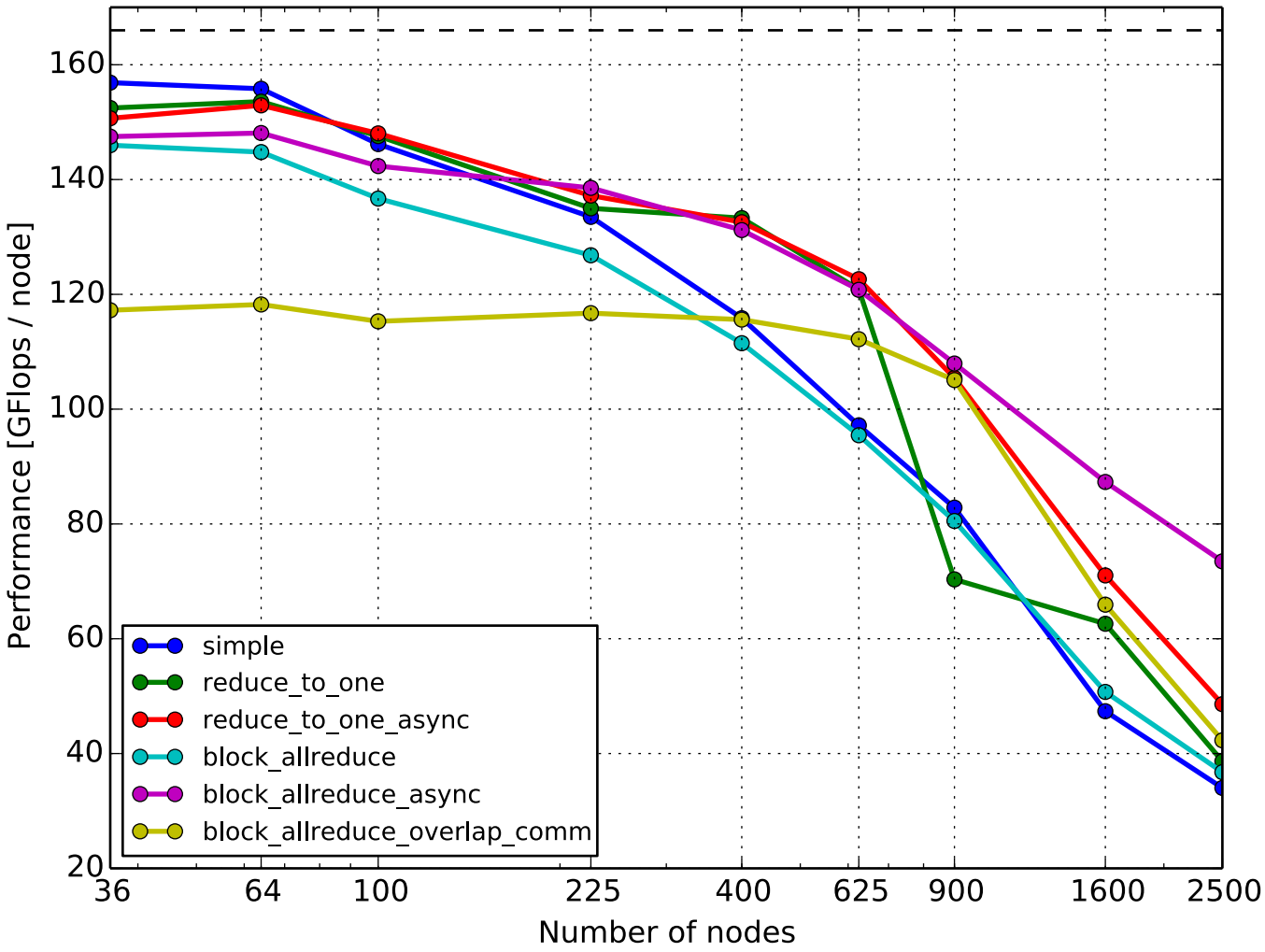


CSCS

ETH zürich

Inner product kernel benchmark

Piz Daint, Intel SB (8 CPU cores) M=10’000 N=10’000 K=1’000’000



CSCS

ETH zürich

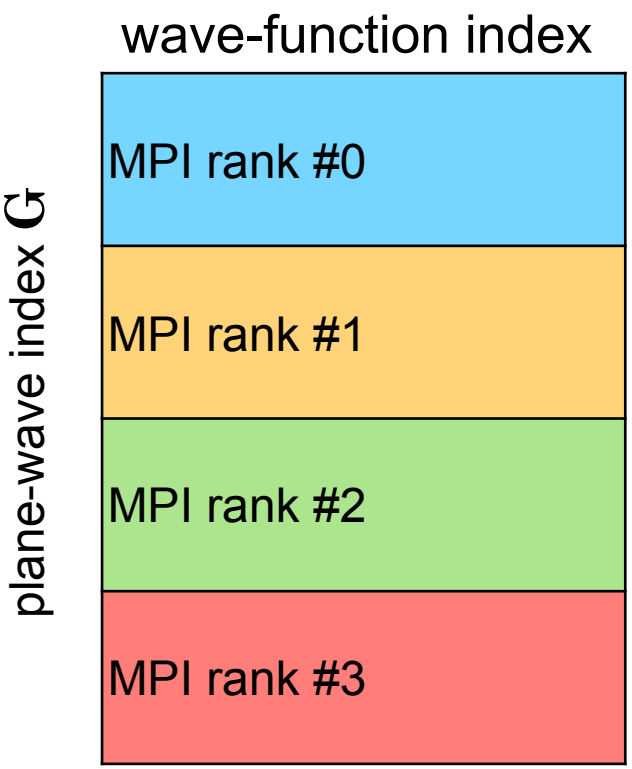
Wave-function transformation

$$\Psi_i(\mathbf{G}) = \sum_j \phi_j(\mathbf{G}) Z_{ji}$$

Wave-function transformation

$$\Psi_i(\mathbf{G}) = \sum_j \phi_j(\mathbf{G}) Z_{ji}$$

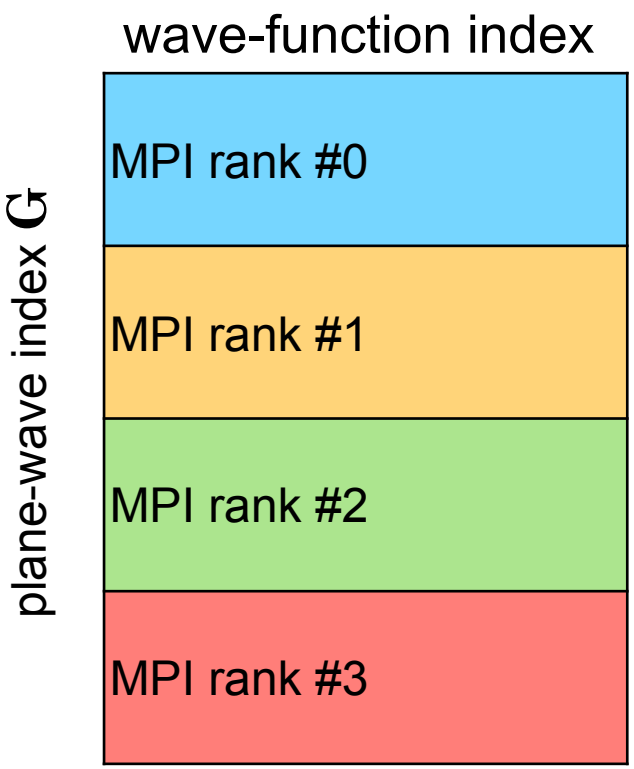
Slab data distribution of $\Psi_i(\mathbf{G})$ and $\phi_j(\mathbf{G})$



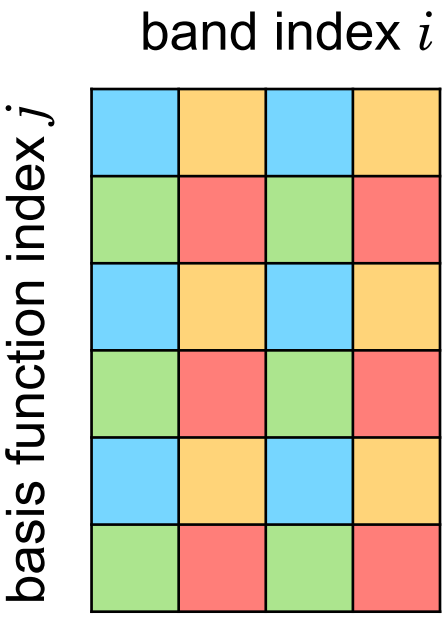
Wave-function transformation

$$\Psi_i(\mathbf{G}) = \sum_j \phi_j(\mathbf{G}) Z_{ji}$$

Slab data distribution of $\Psi_i(\mathbf{G})$ and $\phi_j(\mathbf{G})$



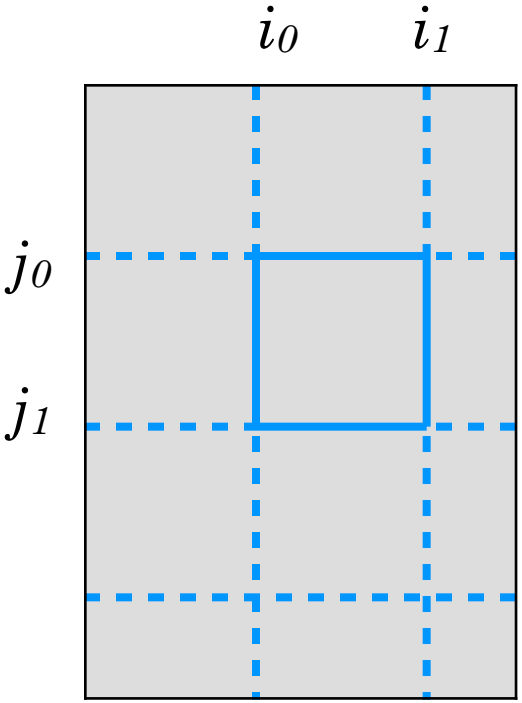
2D block-cyclic distribution of Z_{ji}



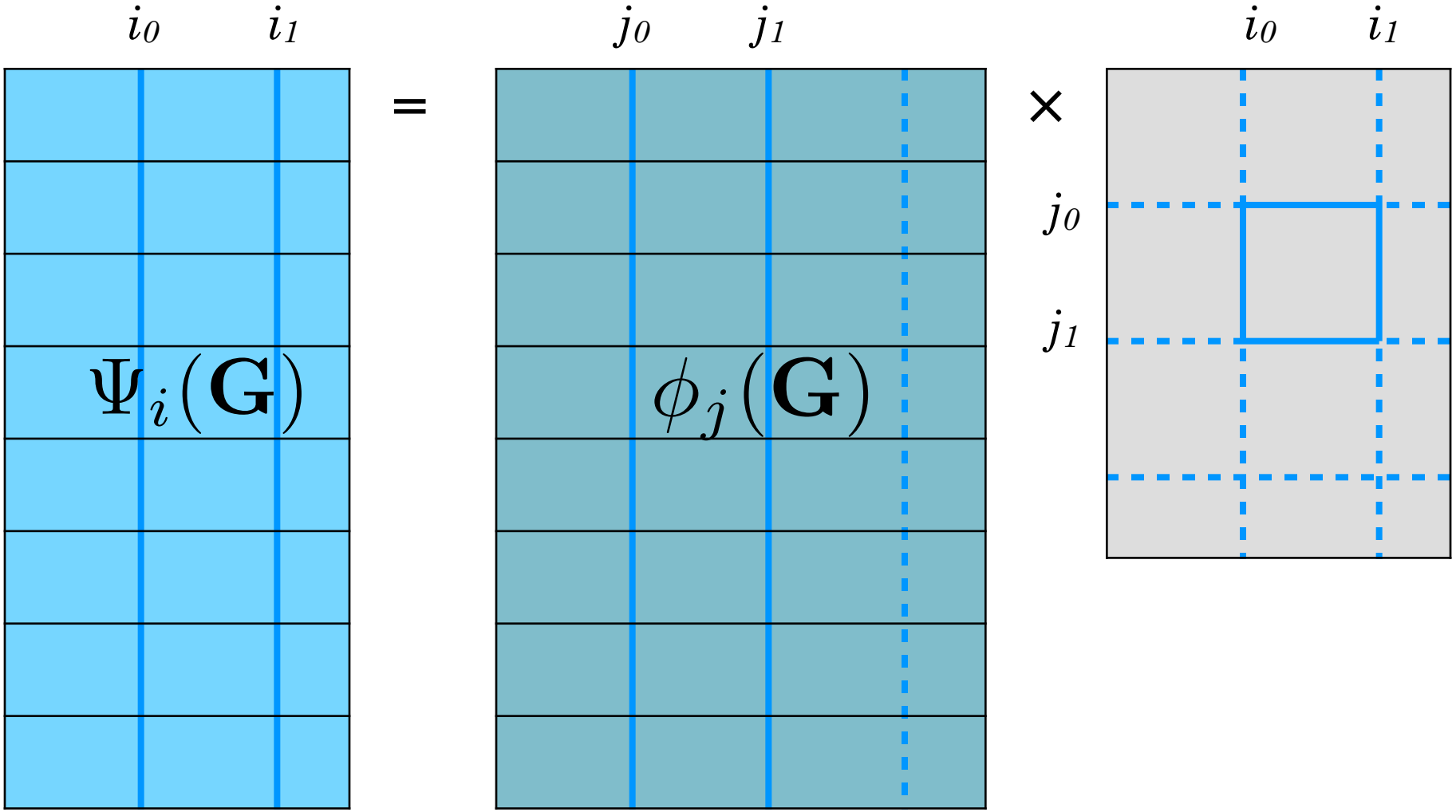
Wave-function transformation

For each block of the transformation matrix:

- each MPI rank packs its contribution to the block
- elements of block are collected with MPI_Allgatherv
- elements are unpacked and the whole block is formed
- the updated of the wave-functions $\Psi_{i_0:i_1}(\mathbf{G})$ from the wave-functions $\phi_{j_0:j_1}(\mathbf{G})$ is done by the local **zgemm**



Wave-function transformation



Wave-function orthogonalization

For the orthonormal basis $\{\phi_j^0\}$, $\langle\phi_j^0|\phi_{j'}^0\rangle = \delta_{jj'}$ the task is to expand it with the new functions $\{\phi_j^{new}\}$, $\{\phi_j^0\}\bigoplus\{\phi_j^{new}\} = \{\phi_j^1\}$ such that $\langle\phi_j^1|\phi_{j'}^1\rangle = \delta_{jj'}$

Wave-function orthogonalization

For the orthonormal basis $\{\phi_j^0\}$, $\langle\phi_j^0|\phi_{j'}^0\rangle = \delta_{jj'}$ the task is to expand it with the new functions $\{\phi_j^{new}\}$, $\{\phi_j^0\}\bigoplus\{\phi_j^{new}\} = \{\phi_j^1\}$ such that $\langle\phi_j^1|\phi_{j'}^1\rangle = \delta_{jj'}$

- Orthogonalize to the old basis functions

$$|\tilde{\phi}_i\rangle = \left(1 - \sum_j |\phi_j^0\rangle\langle\phi_j^0|\right)|\tilde{\phi}_i^{new}\rangle = |\tilde{\phi}_i^{new}\rangle - \underbrace{\sum_j |\phi_j^0\rangle\underbrace{\langle\phi_j^0|\tilde{\phi}_i^{new}\rangle}_{\text{inner product}}}_{\text{wave-function transformation}}$$

Wave-function orthogonalization

For the orthonormal basis $\{\phi_j^0\}$, $\langle\phi_j^0|\phi_{j'}^0\rangle = \delta_{jj'}$ the task is to expand it with the new functions $\{\phi_j^{new}\}$, $\{\phi_j^0\}\bigoplus\{\phi_j^{new}\} = \{\phi_j^1\}$ such that $\langle\phi_j^1|\phi_{j'}^1\rangle = \delta_{jj'}$

- Orthogonalize to the old basis functions

$$|\tilde{\phi}_i\rangle = \left(1 - \sum_j |\phi_j^0\rangle\langle\phi_j^0|\right)|\tilde{\phi}_i^{new}\rangle = |\tilde{\phi}_i^{new}\rangle - \underbrace{\sum_j |\phi_j^0\rangle\underbrace{\langle\phi_j^0|\tilde{\phi}_i^{new}\rangle}_{\text{inner product}}}_{\text{wave-function transformation}}$$

- Orthonormalize new functions

$$O_{ii'} = \langle\tilde{\phi}_i|\tilde{\phi}_{i'}\rangle$$ inner product

$$\mathbf{O} = \mathbf{L}\mathbf{L}^H$$ Cholesky decomposition

$$|\phi_i^{new}\rangle = \sum_{i'} |\tilde{\phi}_{i'}\rangle\mathbf{L}_{i'i}^{-1}$$ wave-function transformation

Wave-function orthogonalization

Test case:

- orthogonalization and transformation of $|\psi_i\rangle,\ \hat{H}|\psi_i\rangle,\ \hat{S}|\psi_i\rangle$ for $i \in [1, N]$
- orthogonalization and transformation of $|\psi_i\rangle,\ \hat{H}|\psi_i\rangle,\ \hat{S}|\psi_i\rangle$ for $i \in [N+1, 2N]$ w.r.t to the first N wave-functions

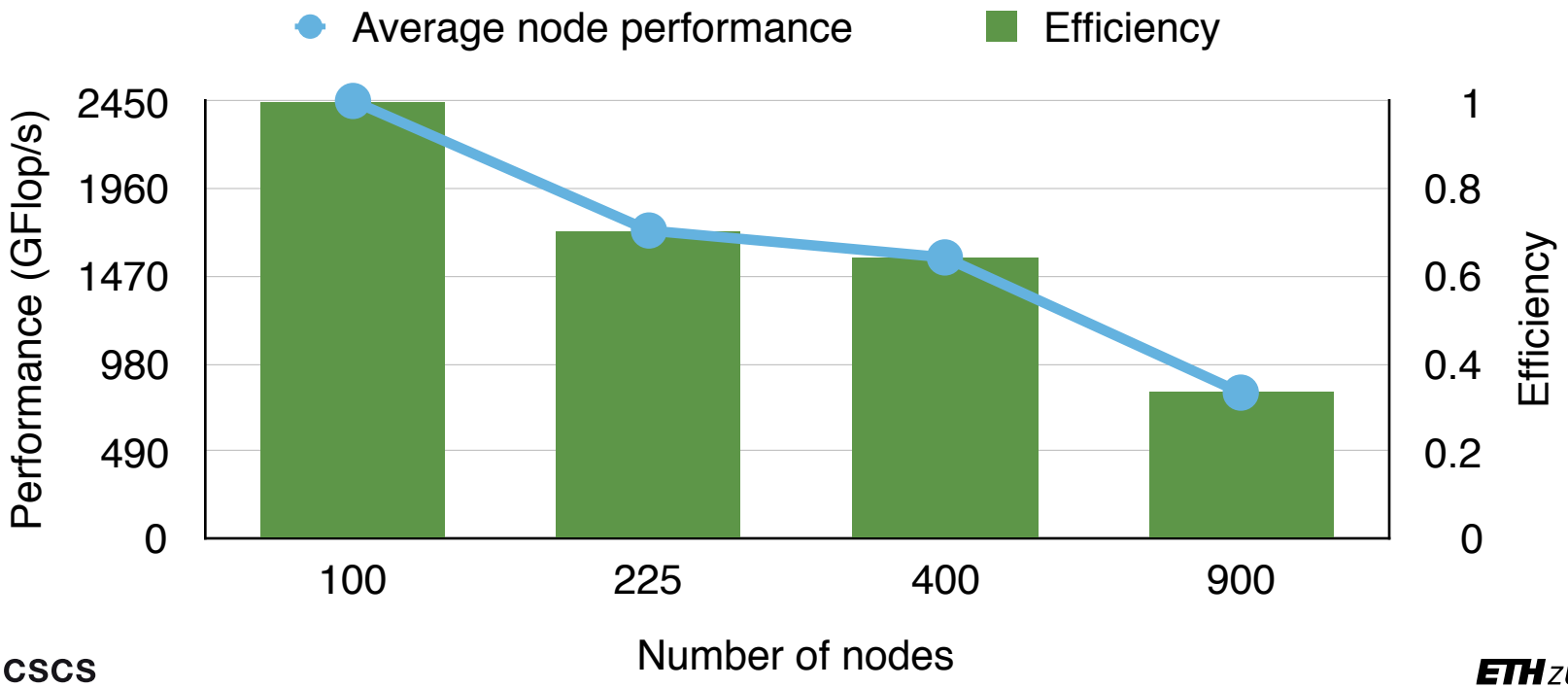
Upgraded Piz Daint, Intel HW (12 CPU cores) + P100 GPU, N=4096 K=3’000’000

Wave-function orthogonalization

Test case:

- orthogonalization and transformation of $|\psi_i\rangle,\ \hat{H}|\psi_i\rangle,\ \hat{S}|\psi_i\rangle$ for $i \in [1, N]$
- orthogonalization and transformation of $|\psi_i\rangle,\ \hat{H}|\psi_i\rangle,\ \hat{S}|\psi_i\rangle$ for $i \in [N+1, 2N]$ w.r.t to the first N wave-functions

Upgraded Piz Daint, Intel HW (12 CPU cores) + P100 GPU, N=4096 K=3'000'000

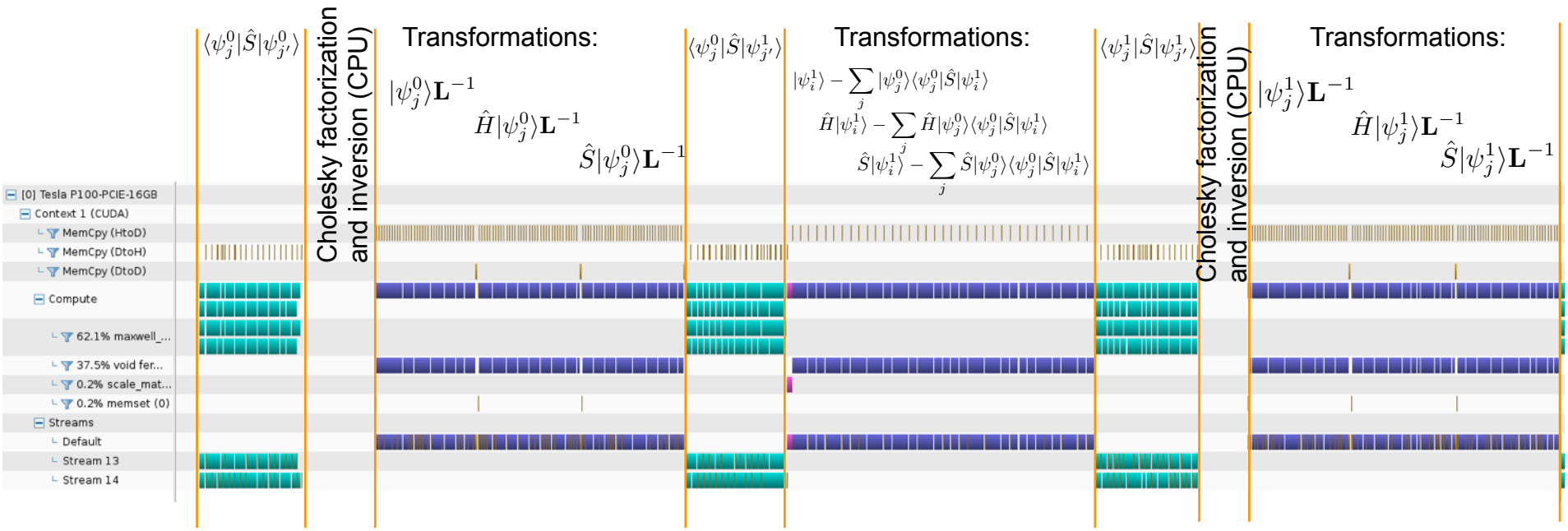


Wave-function orthogonalization

Test case:

- orthogonalization and transformation of $|\psi_i\rangle,\ \hat{H}|\psi_i\rangle,\ \hat{S}|\psi_i\rangle$ for $i \in [1, N]$
- orthogonalization and transformation of $|\psi_i\rangle,\ \hat{H}|\psi_i\rangle,\ \hat{S}|\psi_i\rangle$ for $i \in [N+1, 2N]$ w.r.t to the first N wave-functions

Upgraded Piz Daint, Intel HW (12 CPU cores) + P100 GPU, N=4096 K=3'000'000

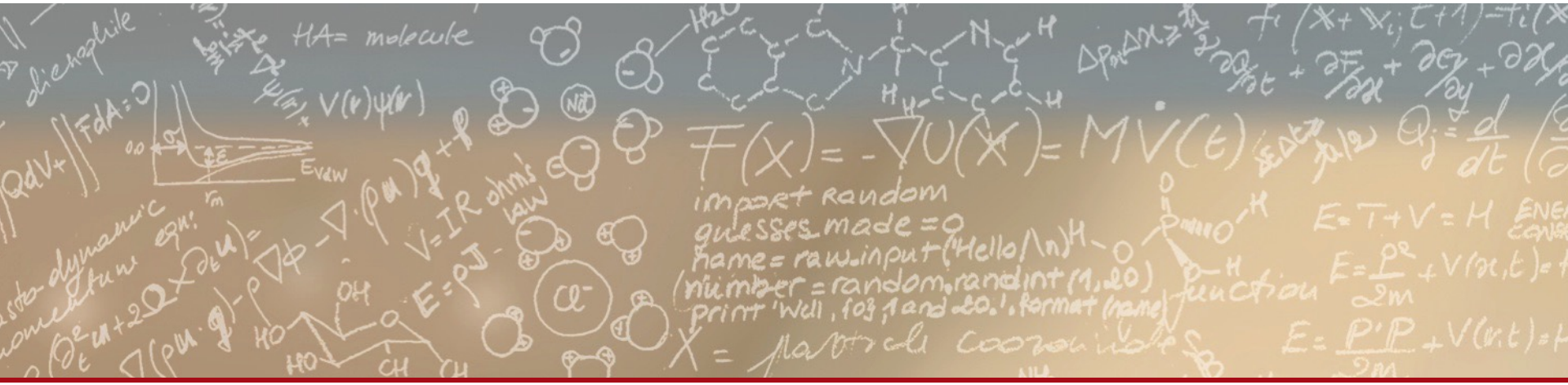


Conclusion

- Domain specific libraries allow a separation of concerns between HPC and scientific communities
- Several compute-intensive kernels that are one level above the standard BLAS/LAPCK/ScaLAPACK/FFTW were identified
- MaX centre of excellence is working on the initial DSL implementation and API (WP4)

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Thank you for your attention.