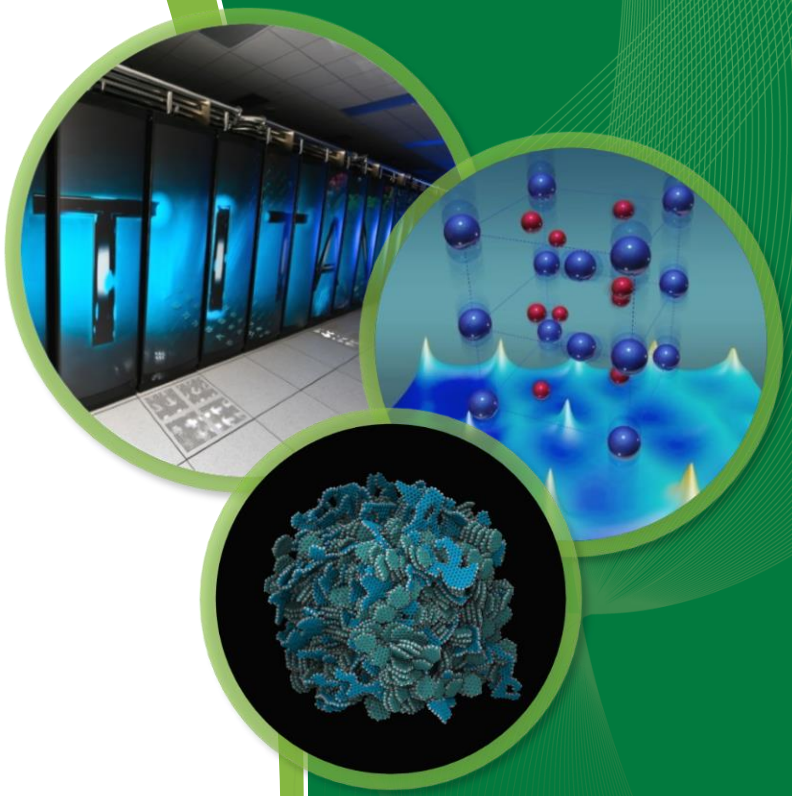
Finite difference DFT solver: Direct functional minimization with eigensolver in projected subspace



Jean-Luc Fattebert

Computational Sciences & Engineering Division

Outline

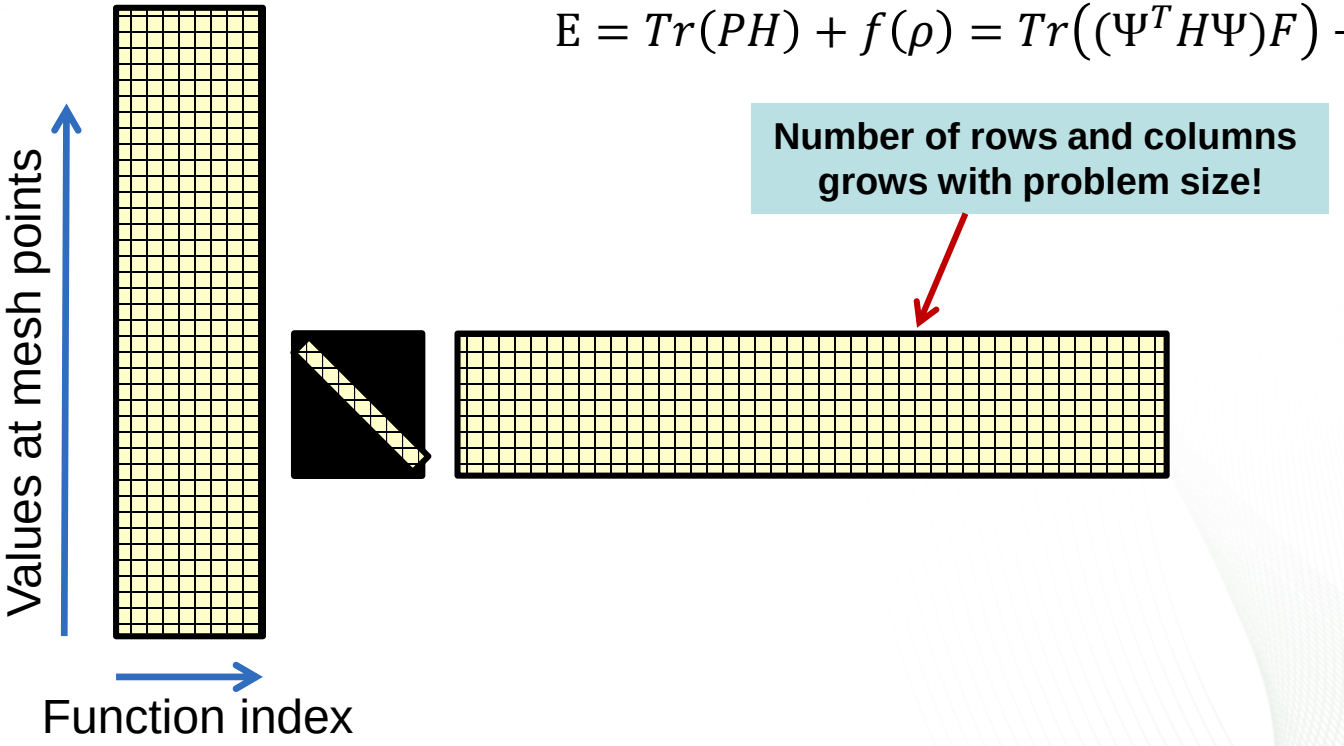
- Direct solver for Kohn-Sham equations discretized by finite differences
- Scalable O(N) solver
- Towards GPU
  - BML library

Find invariant subspace solution of DFT equations for large numerical basis set

- DFT solution as a set of N eigenvectors of length M

$$\Psi = \left(\psi_1 \ldots \psi_N\right) \qquad P = \Psi F \Psi^T \qquad 0 \le f_i \le 1$$

$$\mathrm{E} = Tr(PH) + f(\rho) = Tr\left((\Psi^T H \Psi)F\right) + \mathrm{f}(\rho)$$



Direct solver (vs. SCF)

- Minimize energy functional of fundamental variables
  - $\Psi, f_1, \ldots, f_N$ (wave functions + occupations)
  - Other quantities are derived quantities (elec. density, potential)
- Compute gradient of nonlinear Kohn-Sham energy functional
  - $\nabla E_{KS} = 2[H\Psi - \Psi(\Psi^T H\Psi)]$
- Repeat until convergence
  - Update wave functions
    - Block Preconditioned steepest-descent with Anderson acceleration
    - Block Nonlinear CG (Polak-Ribiere)
  - Update occupation numbers
    - Diagonalize H in projected subspace spanned by wave functions
    - Possibly mix newly computed DM with previous DM
      - Inner optimization loop [Marzari & Vanderbilt, PRL, 1997] for metals

Our Finite difference implementation

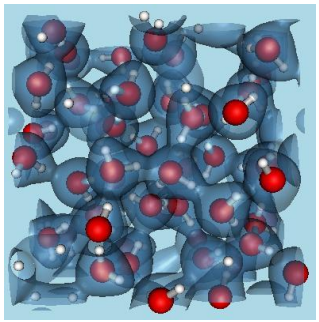
- Multigrid preconditioner applied to steepest descent directions
  - Similar to diagonal “frequency” preconditioner used in Plane-Waves codes
  - [JLF, Bernholc, PRB 2000]
- Mixed precision
  - Electronic wave functions are represented in single precision
  - Accumulation done in double precision for all dot products
  - [JLF et al., SC16 Proceedings]
- Nonorthogonal formulation

$$E_{KS} = \sum_{i,j=1}^{N} \left(S^{-1}\right)_{ij} \int_{\Omega} \phi_i(r) \Delta \phi_j(r) + F[\rho] + \sum_{i,j=1}^{N} \left(S^{-1}\right)_{ij} \int_{\Omega} \phi_i(r) \left(V_{ext} \phi_j\right)(r)$$

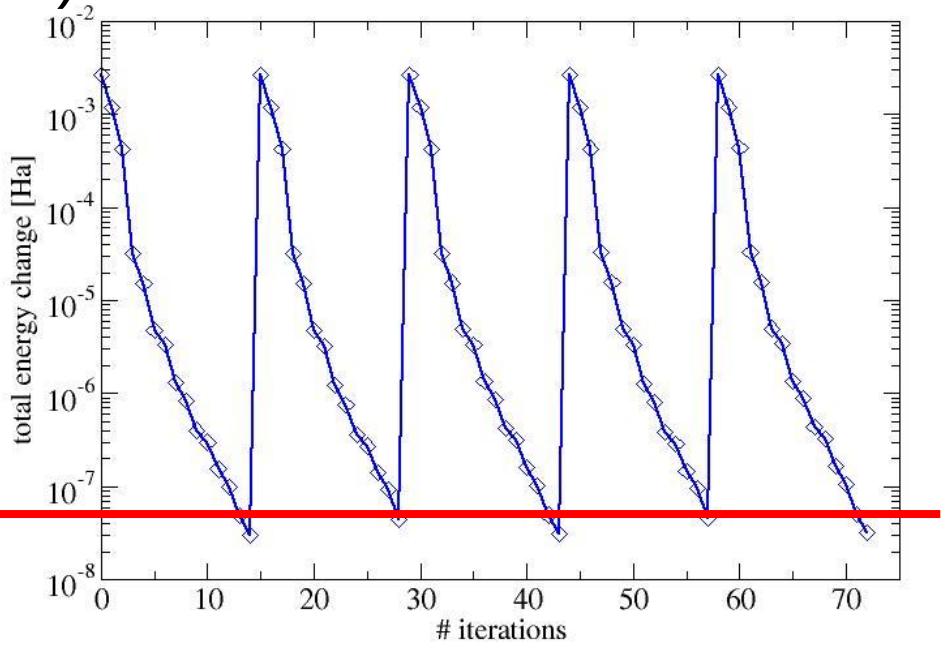
$$\rho(r) = \sum_{i,j=1}^{N} \left(S^{-1}\right)_{ij} \phi_i(r) \phi_j(r)$$

Repetitive solve at consecutive MD steps

- Molecular dynamics (MD) of liquid water
- 64 molecules with periodic boundary conditions
- Convergence for 5 MD steps
- N=256 (no unoccupied states)

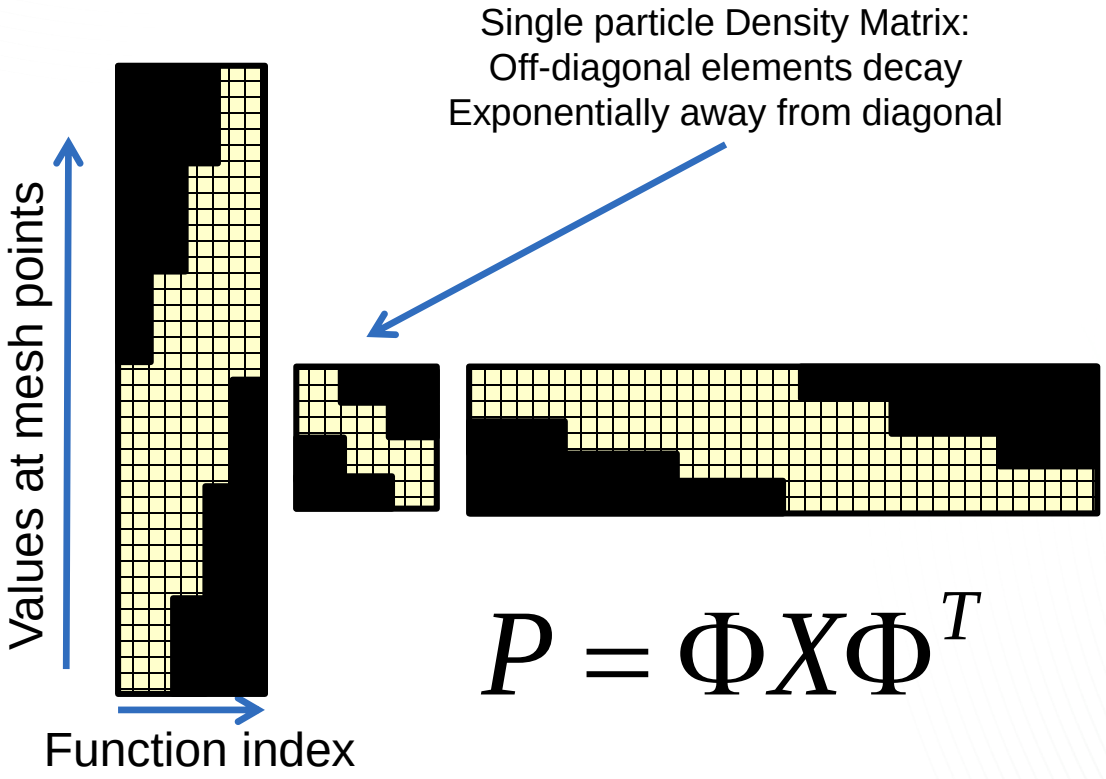


tolerance



Find sparsity in solution to reduce computational complexity

- DFT solution as a set of nonorthogonal localized functions (auxiliary basis set) spanning same subspace as exact solution



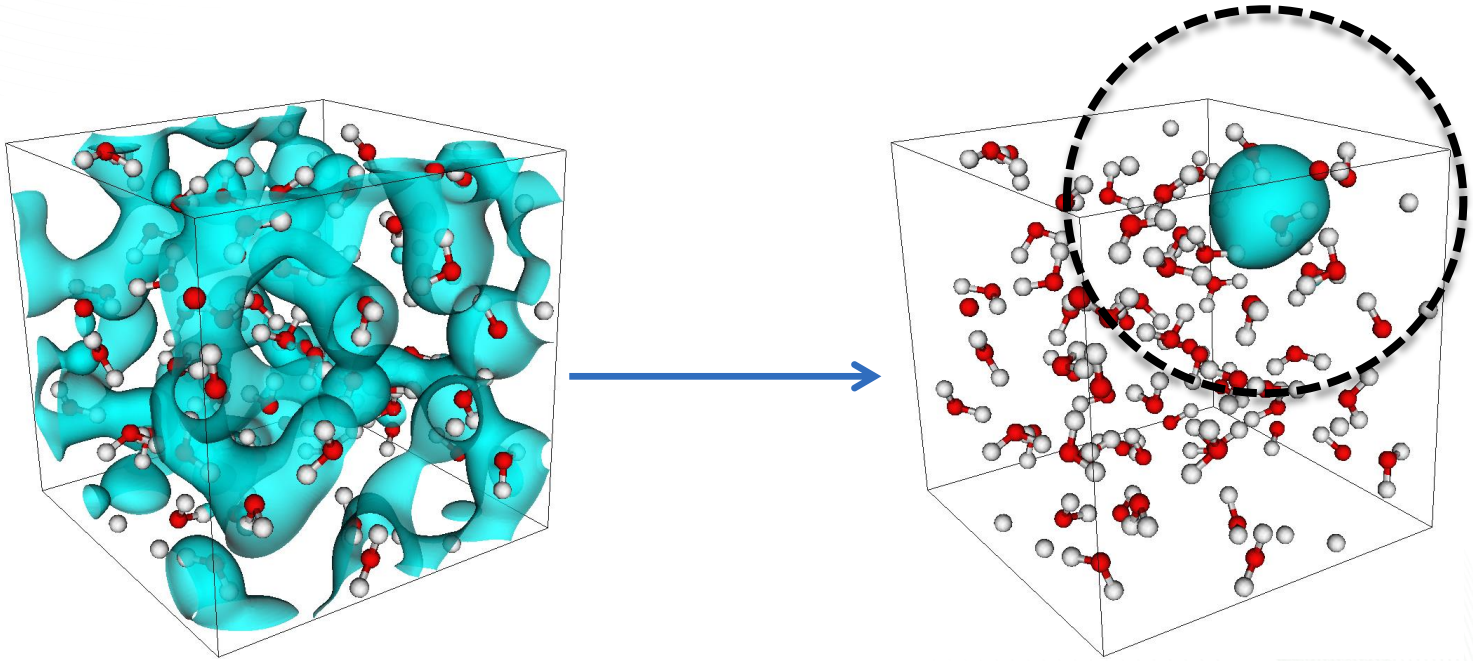
$$X = C^{-1} F C^{-T}$$

$$\Phi = \Psi \cdot C$$

$$P = \Phi X \Phi^{T}$$

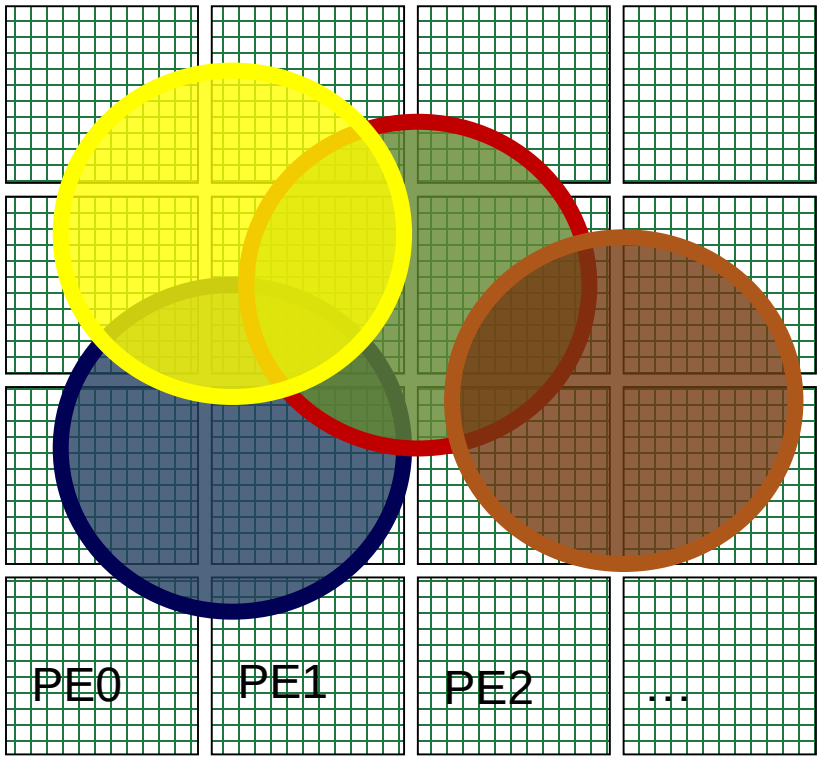
Sparsity in solution corresponds to physical locality

- We prescribe sparsity based on physical distances



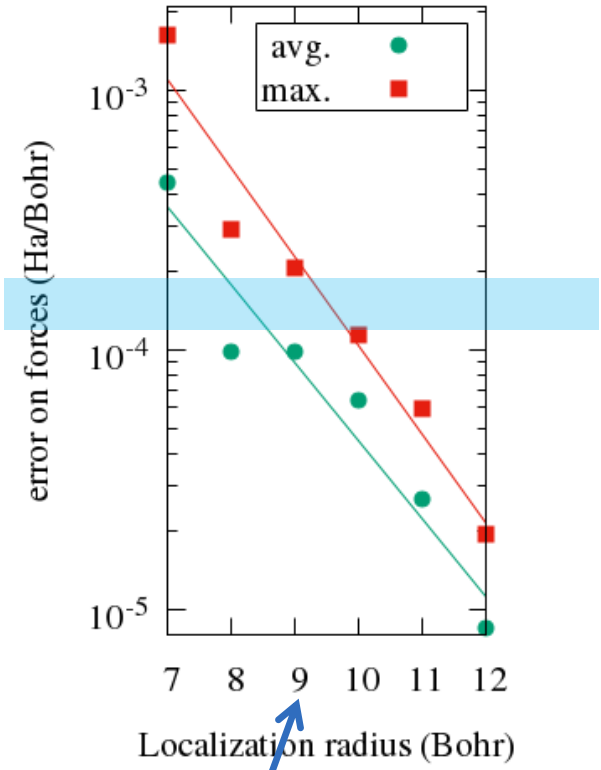
We make use of physical locality in parallel strategy

- Parallel domain decomposition
- Subdomains
  - 16×16×32 (close to strong scaling limit)
- Prescribe sparsity (spatial localization of solution) *a priori*
- Direct minimization of DFT energy functional with localization constraints



Controllable accuracy

- Error in relevant physical quantities (forces acting on atoms) decays exponentially with localization radius



Localization regions of size ~9 Bohr (contains ~50 atoms)

How about the occupation/single particle density matrix?

- For N~2000, ScaLAPACK PDSYEV
  - $O(N^3)$, but small compared to everything else
  - [JLF, Bernholc, PRB 2000]
- For larger N, and large number of MPI tasks, becomes bottleneck
  - Setting up matrices is actually bottleneck!
- Use sparsity of DM
  - Sparse linear algebra in parallel is hard!

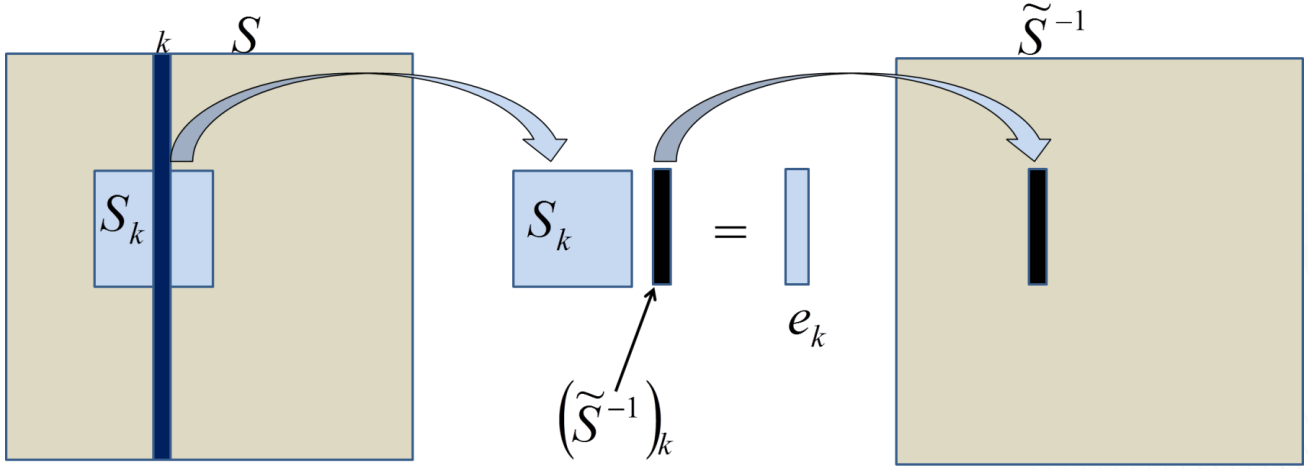
Matrix divide & conquer algorithm:
"Global" matrix made of blocks computed by "local" solves

Strategy for insulators, with only fully occupied states, case X=S^{-1}

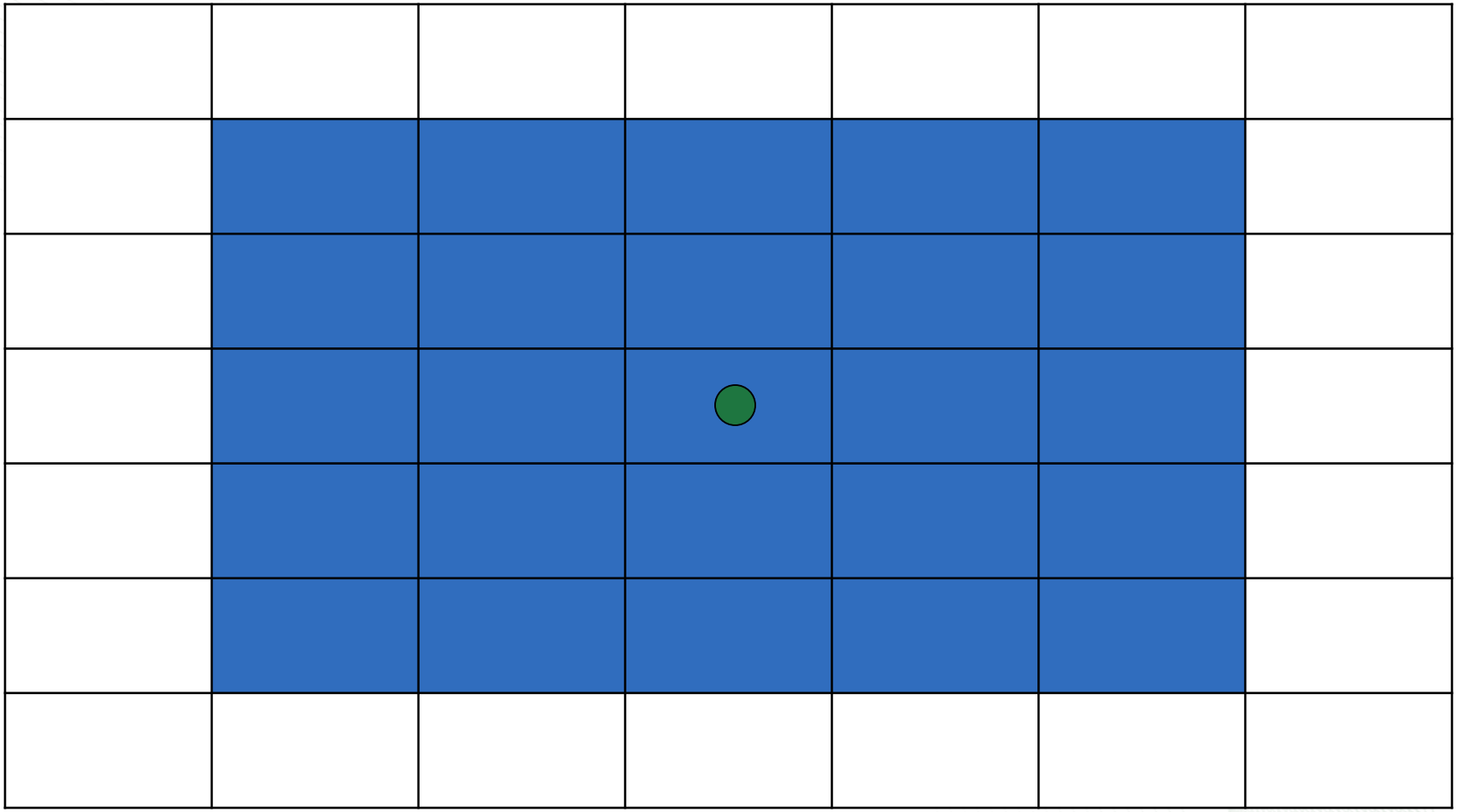
$$P = \Phi S^{-1} \Phi^T$$

$$S_{ij} = \vec{\phi}_i(r)^T \vec{\phi}_j(r)$$

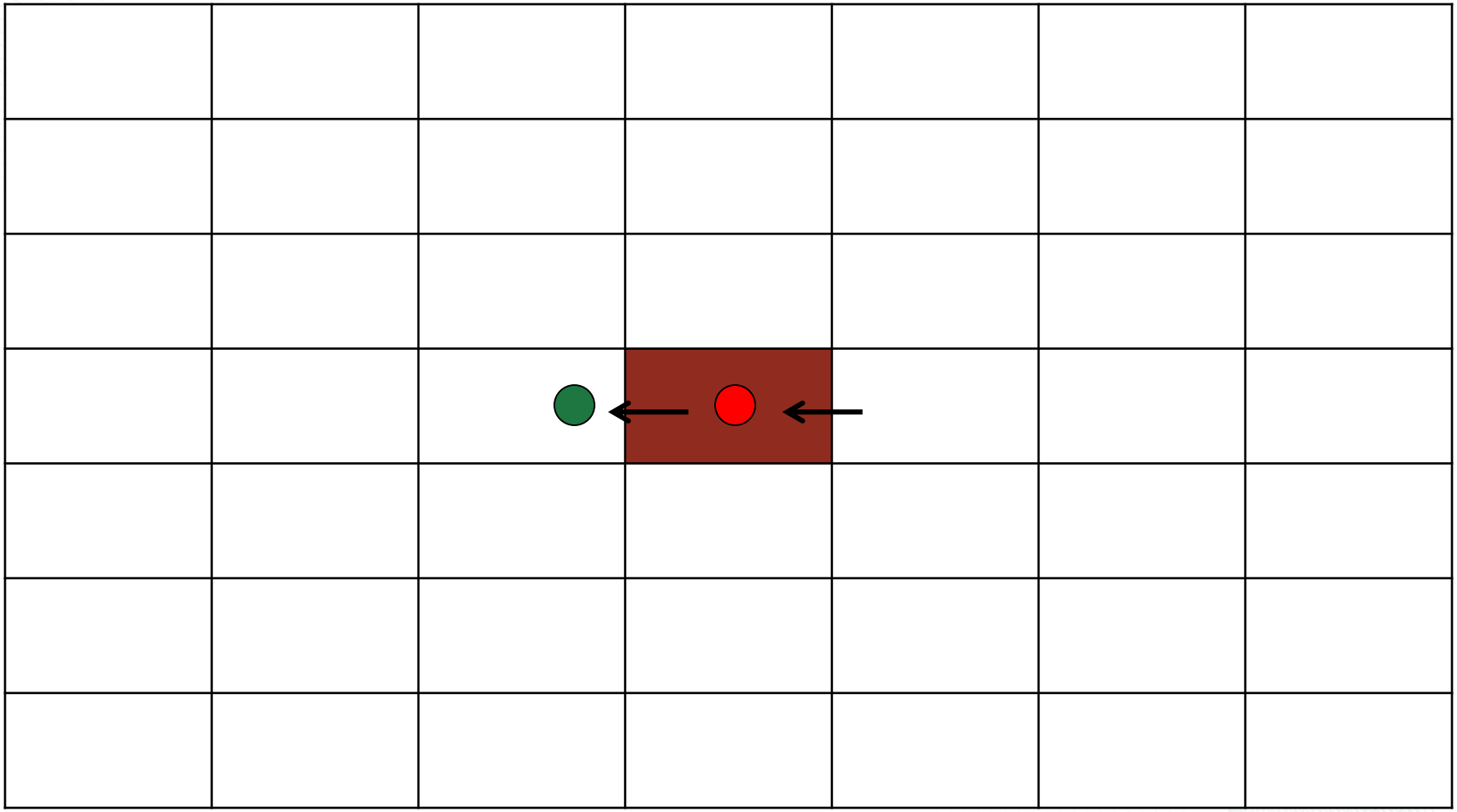
- Only need elements “close” to diagonal
- Off-diagonal elements decay exponentially [Benzi et al.]
- Accumulate on each MPI task principal submatrices of S corresponding to “closest” elements
  - Solve for S_k with ILU0-preconditioned GMRES
  - Compute subset of columns of S^{-1} on each processor



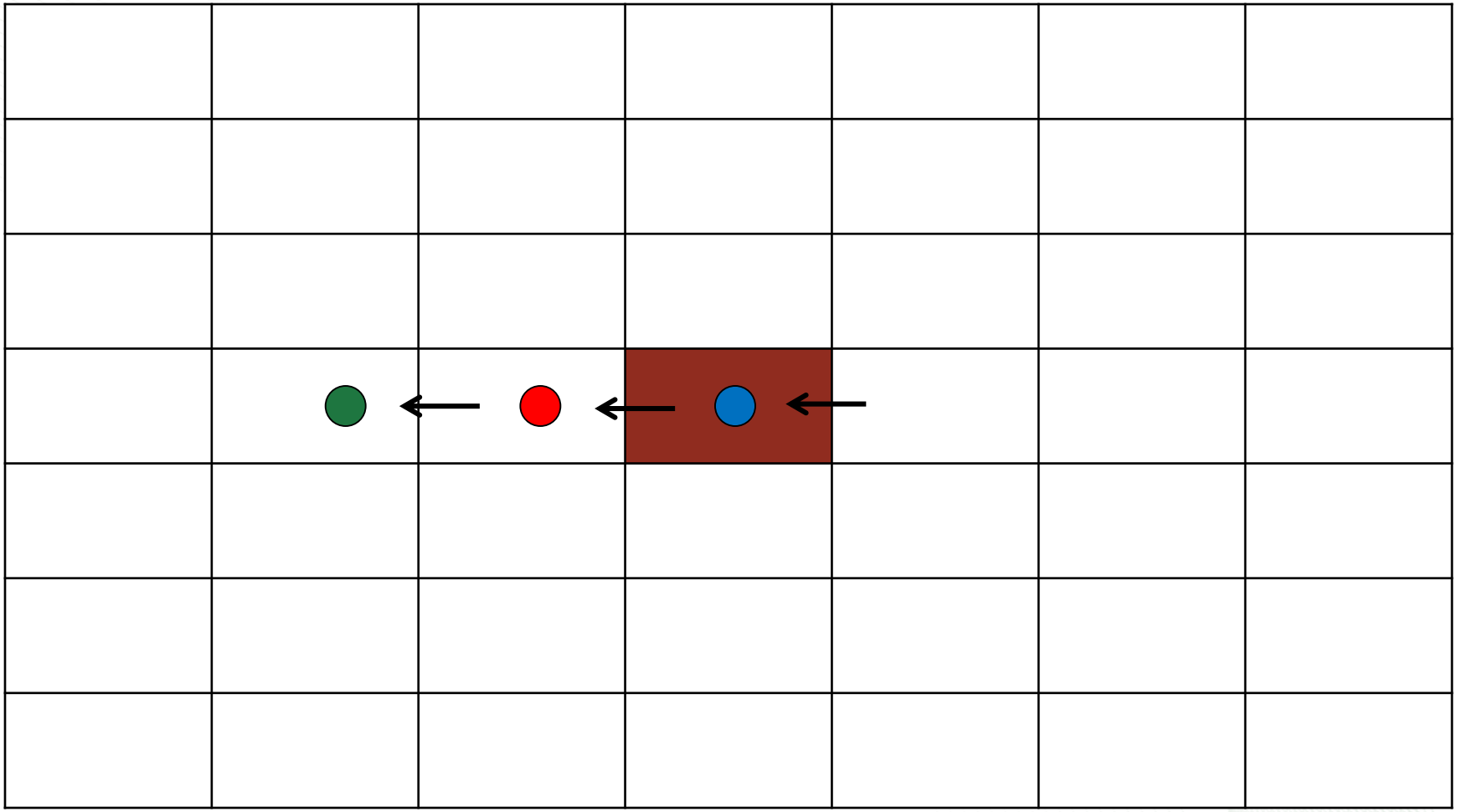
Data communication algorithm for matrix elements (applied before and after solve)



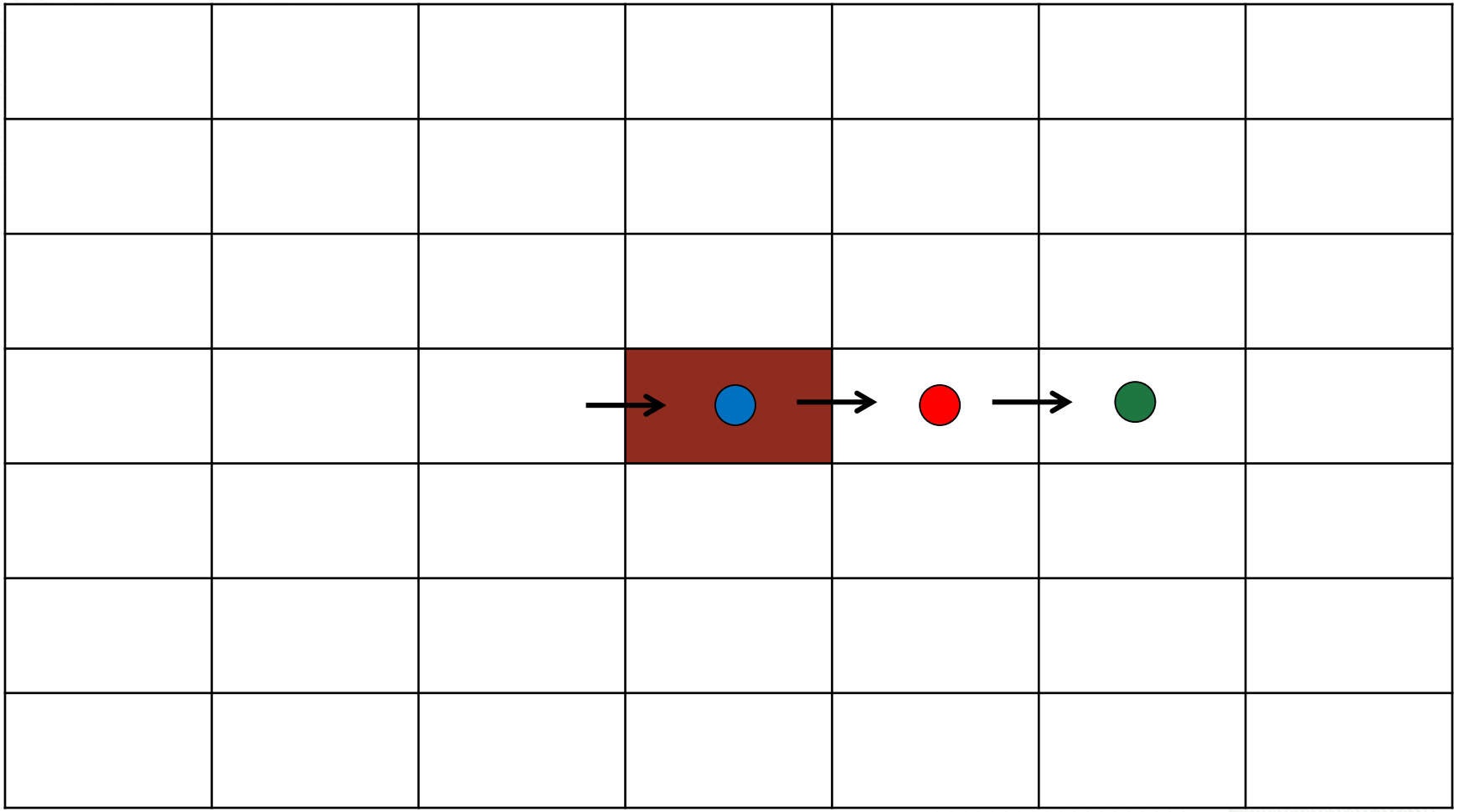
Send data to left neighbor, recv. from right neighbor and merge



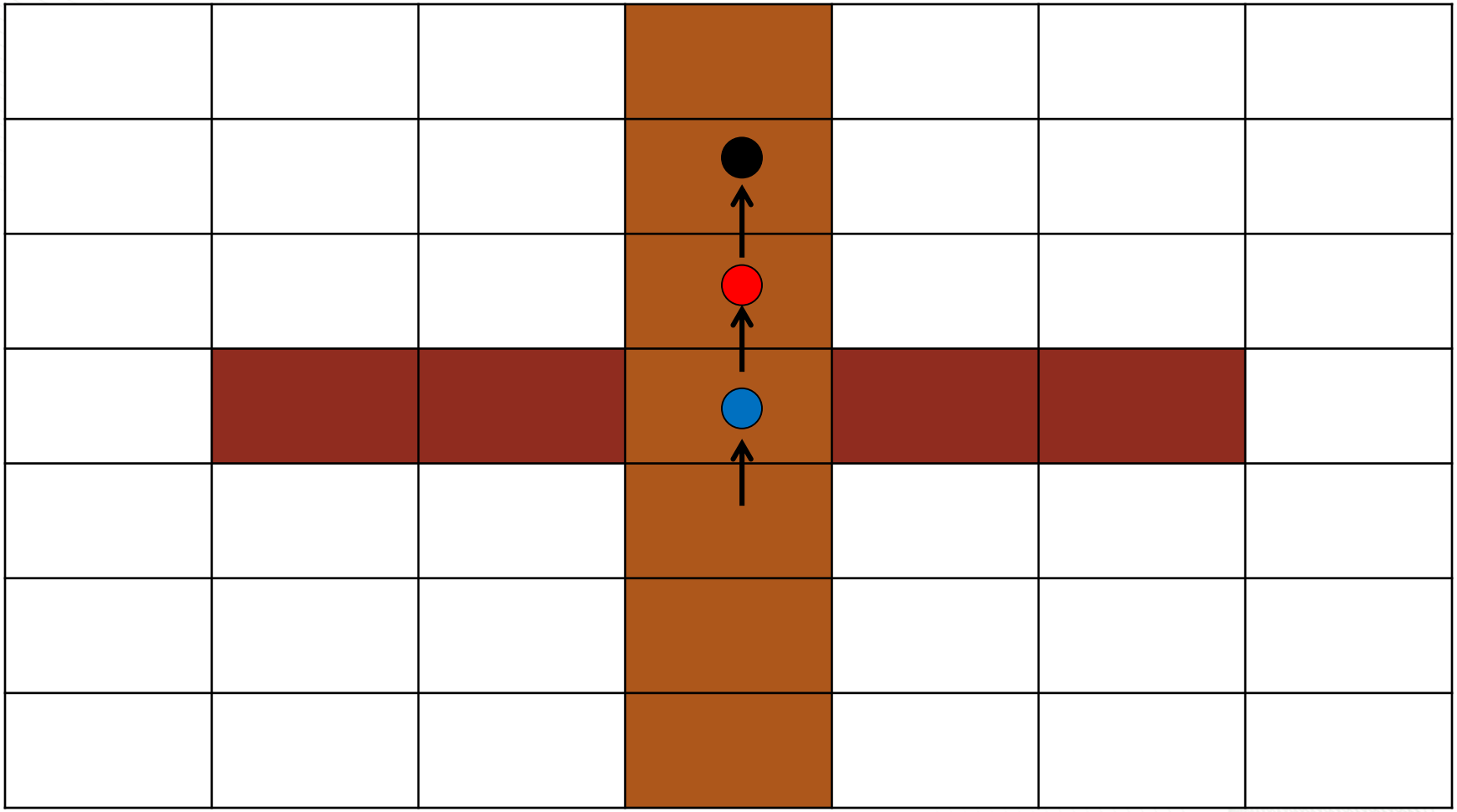
Data communication algorithm: repeat with received data



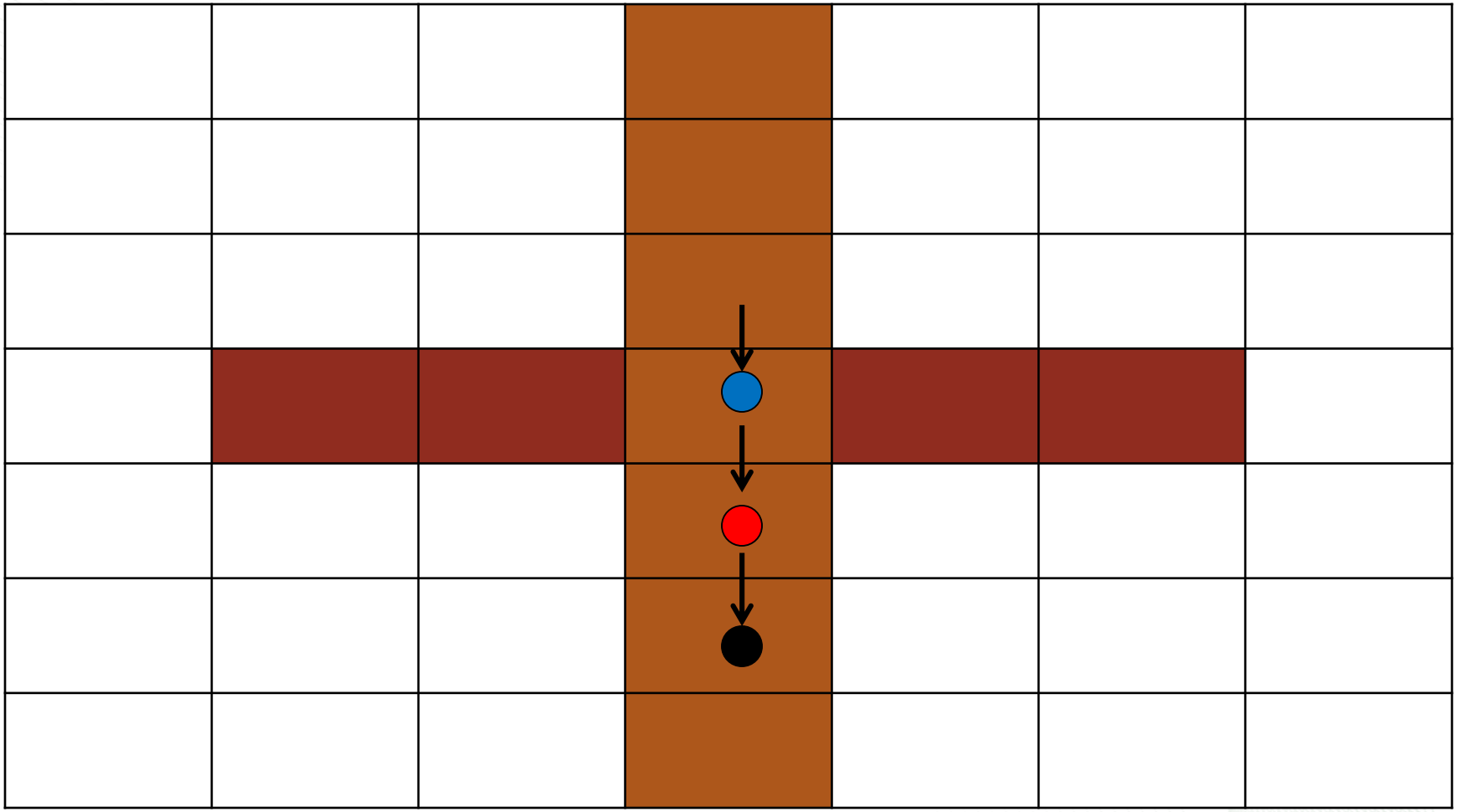
Data communication algorithm: Repeat in left-right direction



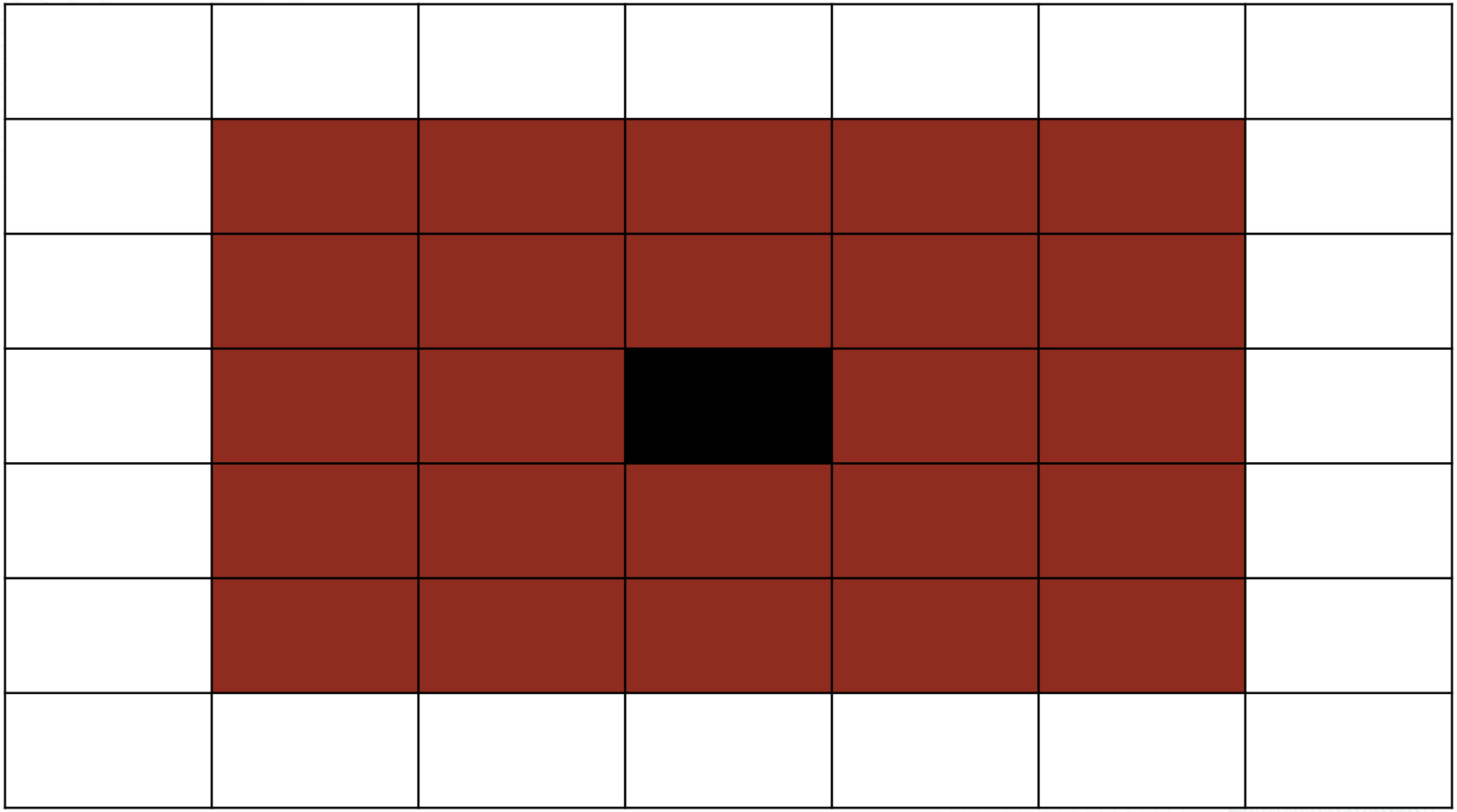
Data communication algorithm:
Repeat in Y (and Z-directions) using accumulated data



Data communication algorithm: Repeat in Y (and Z-directions)

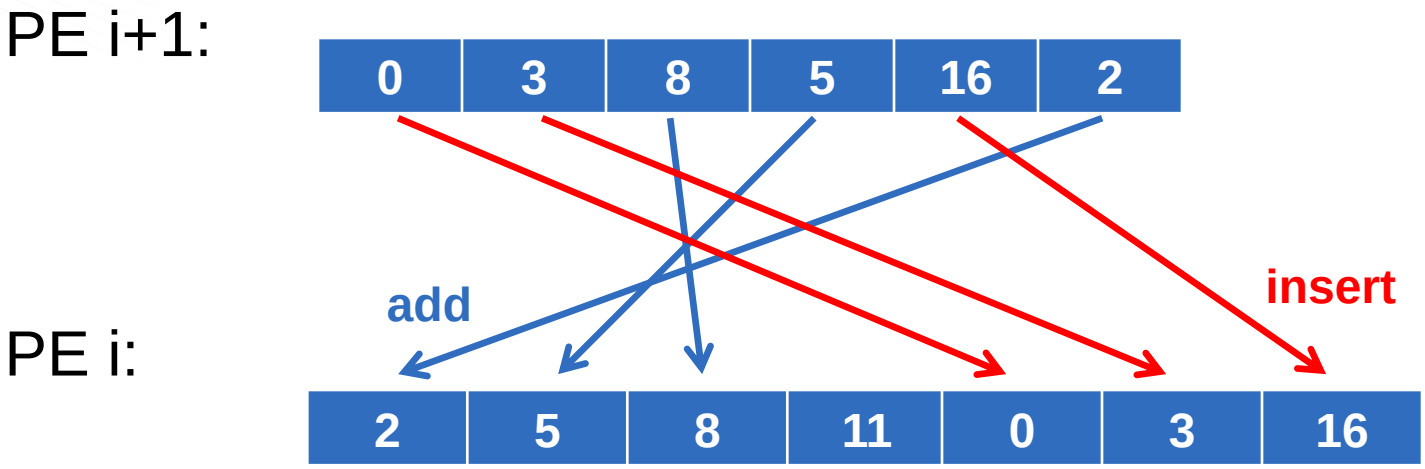


Data communication algorithm: Accumulated data



Merge received data with local CSR data

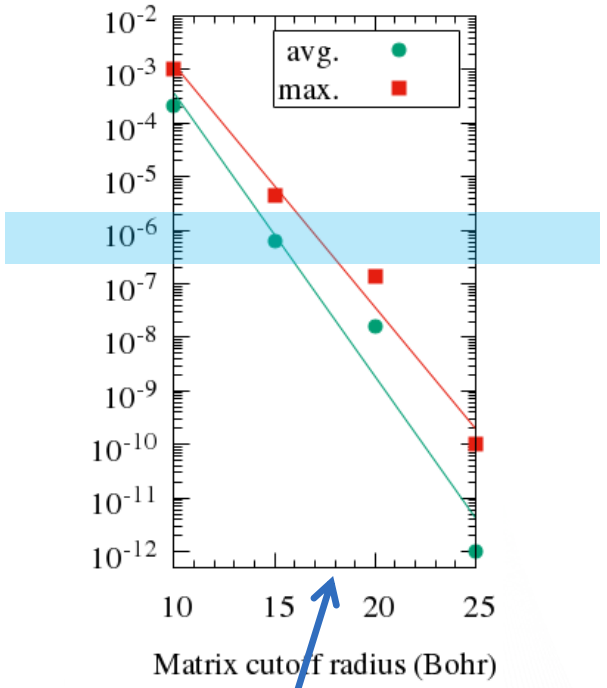
- Consider row j of local data (with global column indices):



Overlap with communication

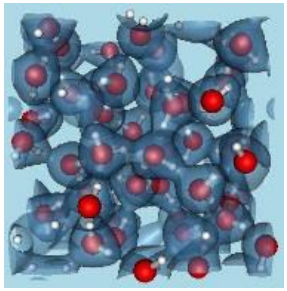
Controllable accuracy

- Error in relevant physical quantities (forces acting on atoms) decays exponentially with matrix cutoff radius
- [Osei-Kuffor, JLF, PRL 2014]



Principal submatrix of size ~4000×4000

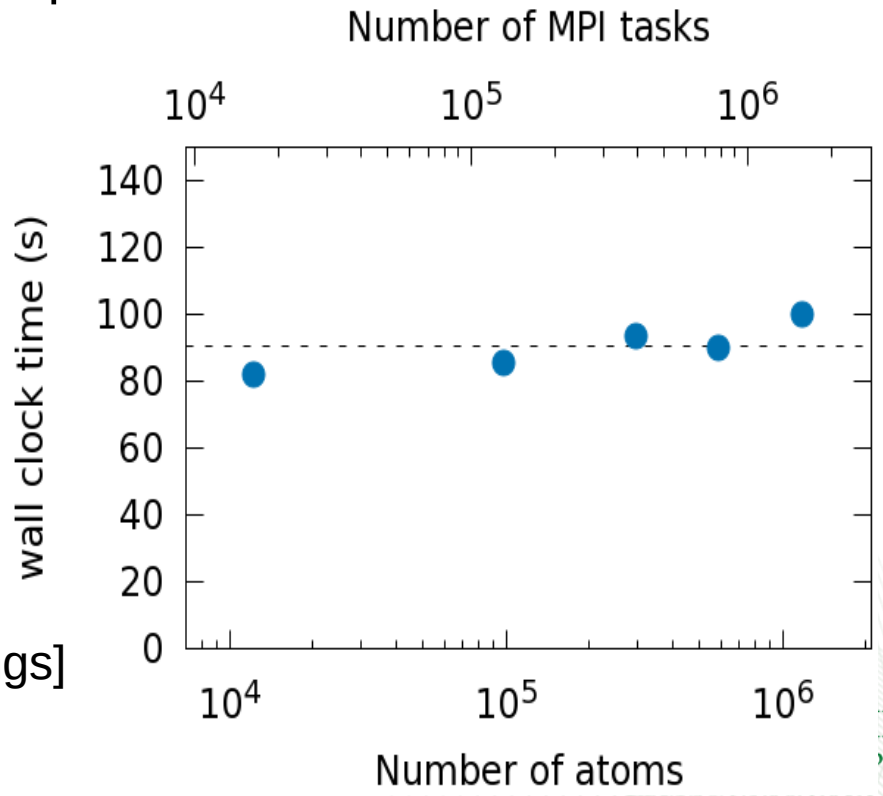
An O(N) scalable implementation: MGmol code



- O(N) operations for N electrons
- Parallel domain decomposition — Each processor needs to communicate only with processors within a limited radius
  - Localized electronic orbitals
  - Local solver to compute selected elements of S^{-1}
- The only global coupling is through a Coulomb interaction term
  - Poisson problem solved with Multigrid-preconditioned CG
- Open source
  - https://github.com/llnl/mgmol

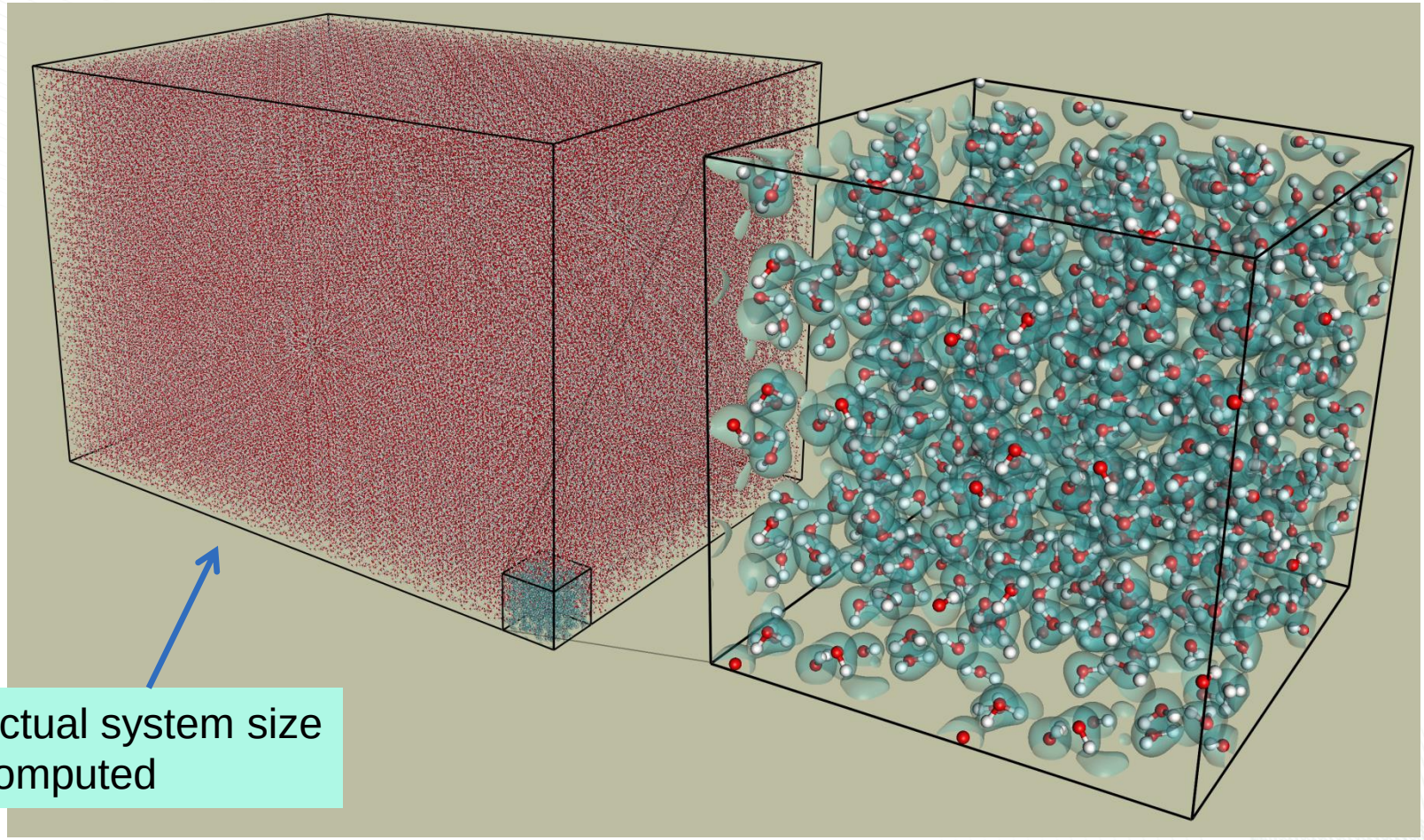
Scalability, time-to-solution O(N/p)

- Weak scaling on the full Sequoia machine
  - IBM/BGQ
  - 1 MPI task/core, 4 threads/MPI task
  - No. Processors proportional to problem size → Constant time-to-solution
- Liquid water
- 1 MD step in 1.5 minutes



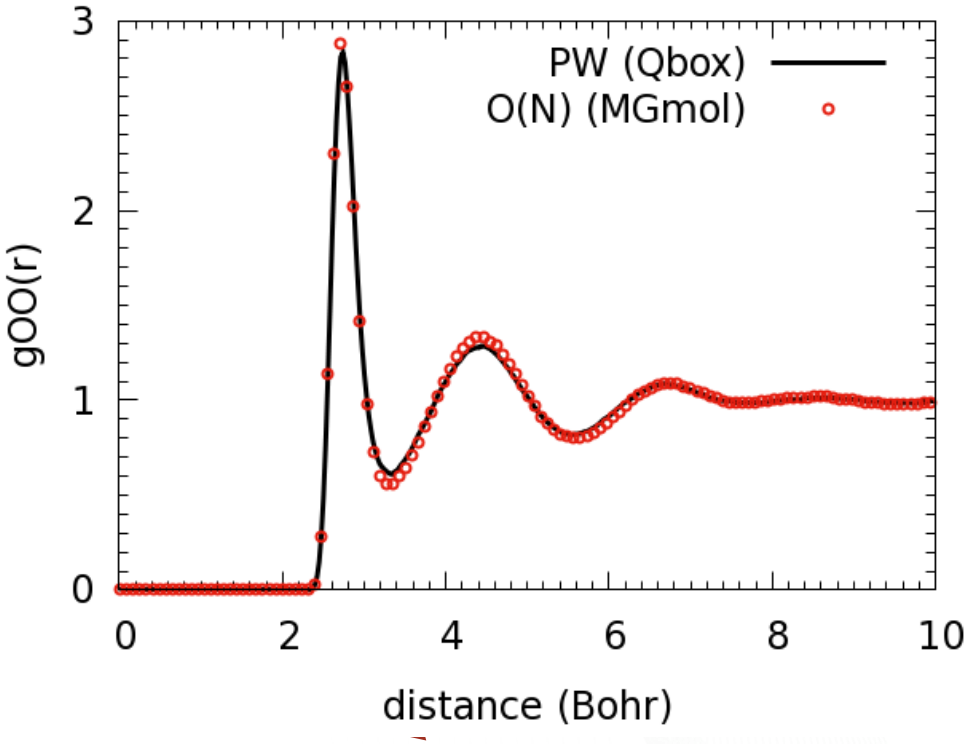
[JLF et al., SC16 Proceedings]

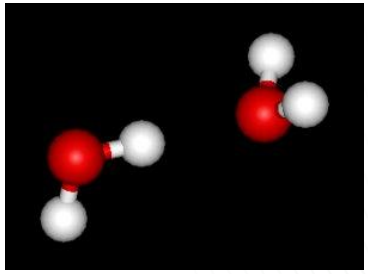
Full Sequoia run: Liquid water with 1,179,648 atoms and 1,572,864 MPI tasks



Excellent agreement with standard Plane Waves benchmark

- Validation for dynamic properties
  - Pair-correlation function
- Comparison with O(N^3) (Plane-Waves) result for relatively “small” problem
  - 1536 atoms



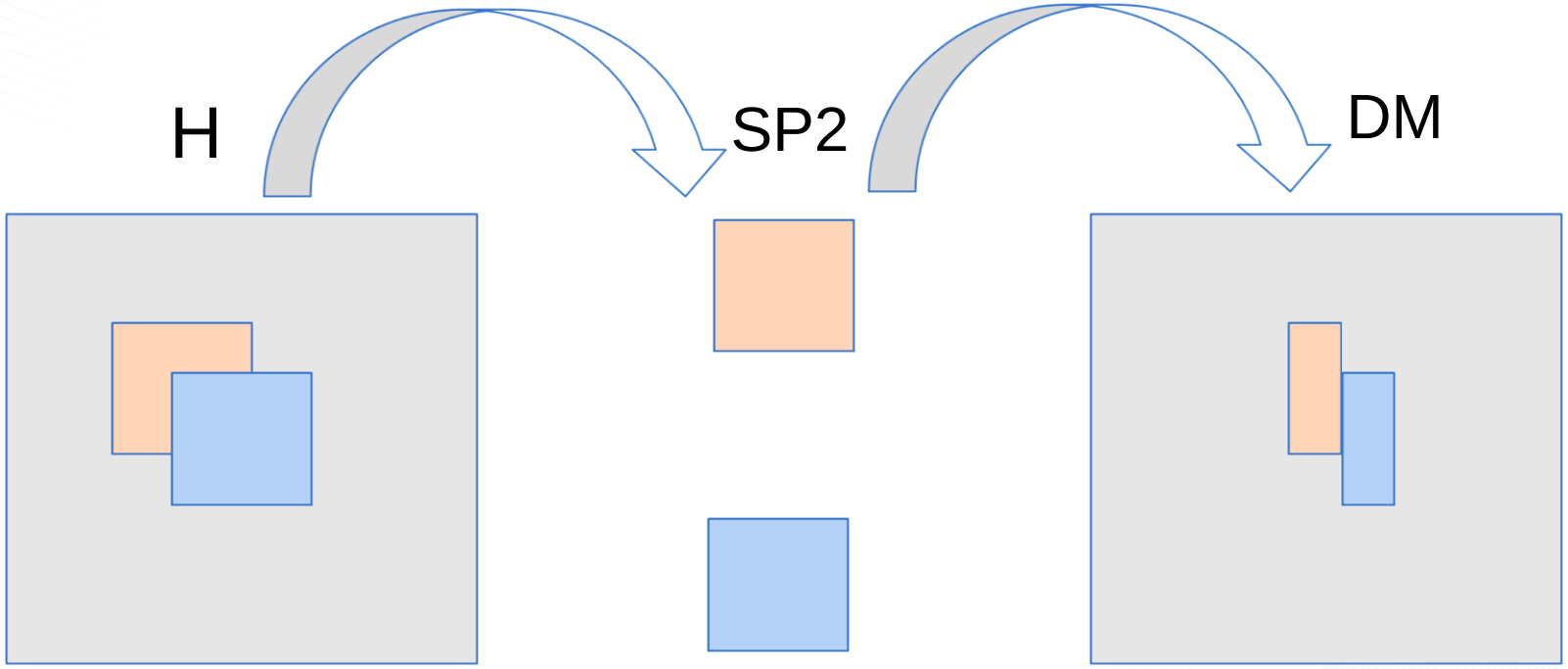


Divide & Conquer for matrices

- Solving principal submatrix problems in parallel
  - Use values “close to center” combined with others computed by other parallel tasks
- Above
  - computing inverse of Gram matrix
- Generalization: compute single particle Density Matrix in basis of localized orbitals
  - Nonorthogonal purification based on SP2 [Niklasson, Weber, Challacombe, J. Chem. Phys. 2005]

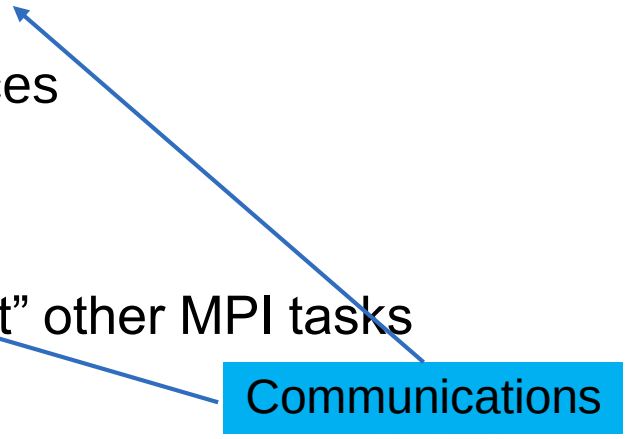
Density matrix computation

- SP2 algorithm applied to principal submatrix



DM solver in practice

- For each MPI task
  - Build sparse principal submatrices H and S matrices from elements computed by “nearest” other MPI tasks
  - Convert sparse matrices to dense matrices
  - Solve for DM
    - using SP2 (~15 iterations)/LAPACK dsyev
  - Distribute “local” DM columns to “nearest” other MPI tasks



Accuracy Results

- $H2O_{512}$
- 1 unoccupied state/molecule
- Localized orbitals with R=10 Bohr

Radius (Bohr)	Principal sub-matrix size	error
Inf.	2560	0.
20.	2400	$2.9x10^{-4}$
15	1566	$3.8x10^{-4}$

Using a third party library for DM solver?

- DFT codes typically do not rely on many third party library beside BLAS/LAPACK/ScaLAPACK
- Is it going to change with new architectures, in particular nodes with GPU accelerators?
  - Large effort needed to port codes
  - Harder to get performance
- Library of DM solvers on the node (SP2,…)?
  - The Basic Matrix Library (BML) for Quantum Chemistry is an attempt in that direction

BML (https://github.com/lanl/bml)

- The basic matrix library (BML) is a collection of various matrix data formats (in dense and sparse) and their associated algorithms for basic matrix operations
- Application programming interfaces (API) available for both C and FORTRAN
- Current status of this library allows us to use two different formats for representing matrix data: dense, sparse (ELLPACK, ELLSORT)
- In development
  - Sparse CSR format
  - Dense matrix operations using MAGMA (available soon)
    - A Matrix-matrix multiplication takes 3 ms for N=4000 on NVIDIA GPU P100

Acknowledgements

- Collaborators
  - Daniel Osei-Kuffuor, Lawrence Livermore National Laboratory
  - Christian Negre, Jamaludin Mohd-Yusof, and Susan Mniszewski, Los Alamos National Laboratory
- This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.