Optimization and Parallelization of FIND Algorithm

Song Li Eric Darve

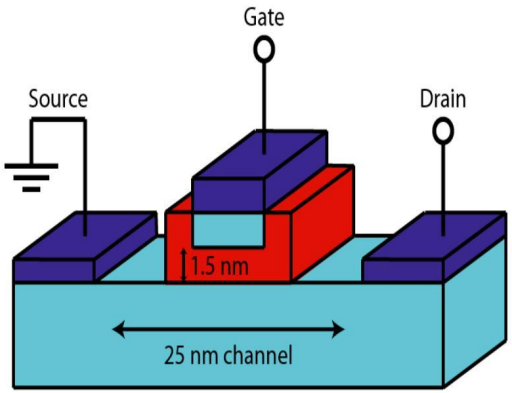
Institute for Computational and Mathematical Engineering, Stanford University
lisong@stanford.edu

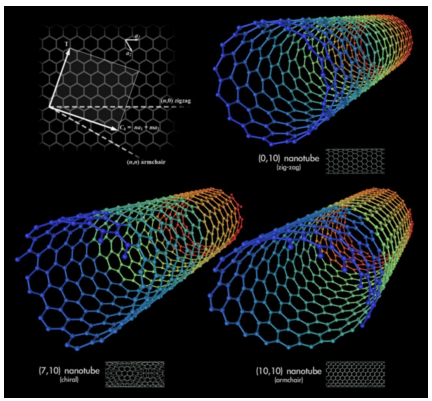
SIAM CSE09
March 4, 2009

Outline

Outline

Introduction





- Modeling the current through nano-devices by Non-Equilibrium Green's Function approach
- System of Schrödinger-Poisson equations
- Best known algorithm (RGF) has running time $O(n_x^3 n_y)$
- Our method (FIND): $O(n_x^2 n_y)$
- Other devices: nanotubes and nanowires

The Math Problem

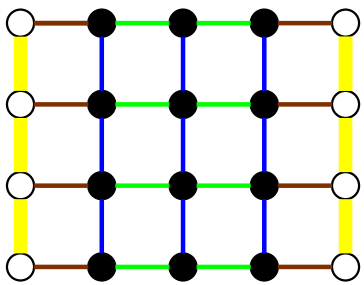
- What we want: the diagonal of $G^r = A^{-1}$
- What we have: a sparse matrix A from a discretized 2D mesh

The Math Problem

- What we want: the diagonal of $G^r = A^{-1}$
- What we have: a sparse matrix A from a discretized 2D mesh

$n_y = 5$

$n_x = 4$



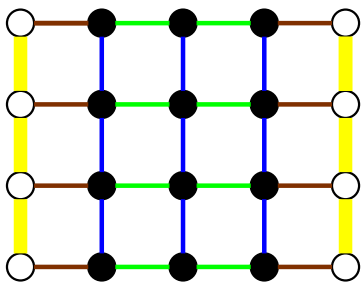
4×5 mesh

The Math Problem

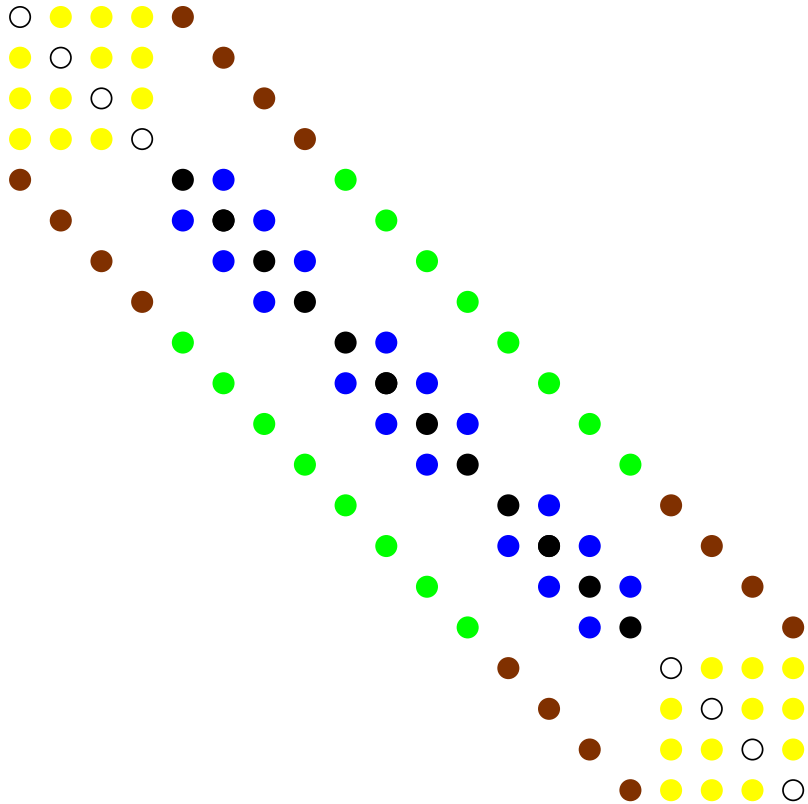
- What we want: the diagonal of $G^r = A^{-1}$
- What we have: a sparse matrix A from a discretized 2D mesh

$n_y = 5$

$n_x = 4$



4×5 mesh



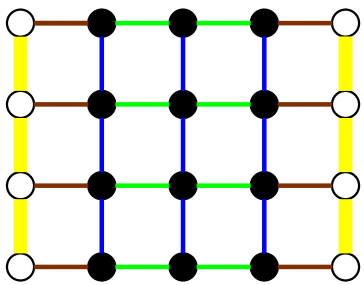
20×20 matrix A

The Math Problem

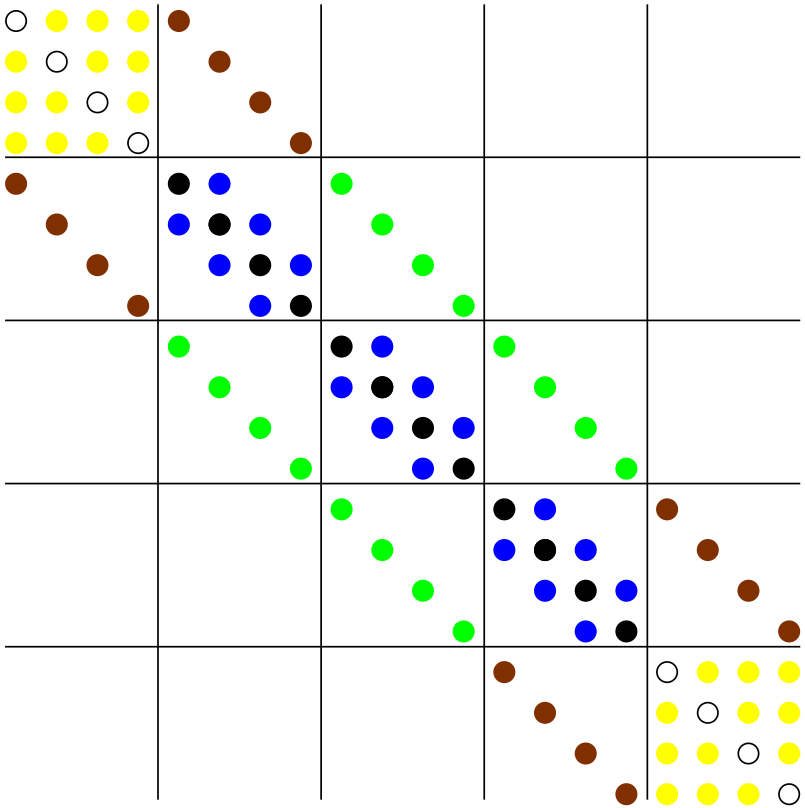
- What we want: the diagonal of $G^r = A^{-1}$
- What we have: a sparse matrix A from a discretized 2D mesh

$n_y = 5$

$n_x = 4$



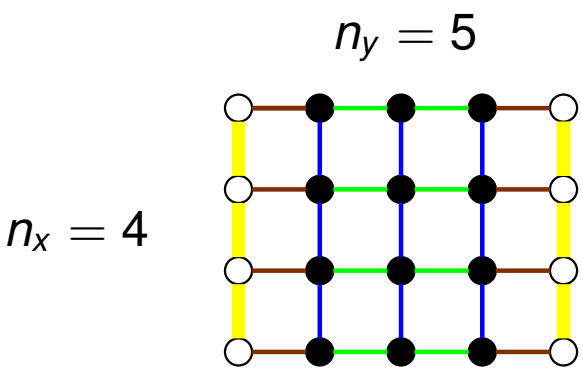
4×5 mesh



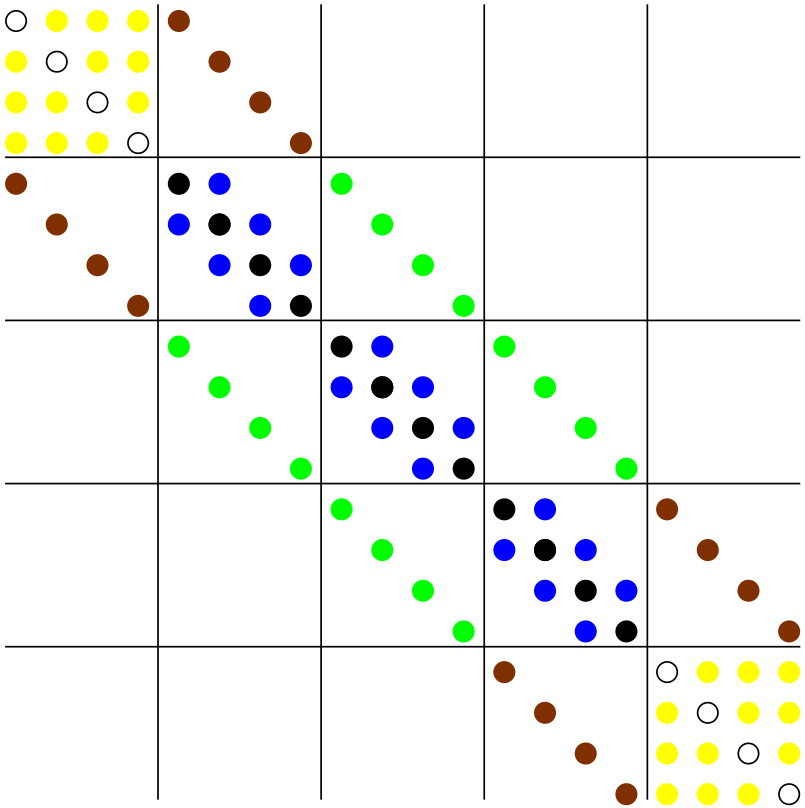
20×20 matrix A

The Math Problem

- What we want: the diagonal of $G^r = A^{-1}$
- What we have: a sparse matrix A from a discretized 2D mesh



4×5 mesh



20×20 matrix A

Outline

Key Observations

- Last entry in A^{-1} can be obtained through LU factorization: $(A^{-1})_{nn} = (U^{-1})_{nn} = (U_{nn})^{-1}$
- Obtain all the diagonals through multiple factorizations
- Local connectivity $\Rightarrow$ problem decomposition: partial factorizations feasible
- Proper ordering makes most of them identical: subproblems overlap $\Rightarrow$ dynamic programming
- Computational cost for all the diagonal entries of the inverse is of the same order as a single LU factorization!

Key Observations

- Last entry in A^{-1} can be obtained through LU factorization: $(A^{-1})_{nn} = (U^{-1})_{nn} = (U_{nn})^{-1}$
- Obtain all the diagonals through multiple factorizations
- Local connectivity $\Rightarrow$ problem decomposition: partial factorizations feasible
- Proper ordering makes most of them identical: subproblems overlap $\Rightarrow$ dynamic programming
- Computational cost for all the diagonal entries of the inverse is of the same order as a single LU factorization!

Key Observations

- Last entry in A^{-1} can be obtained through LU factorization: $(A^{-1})_{nn} = (U^{-1})_{nn} = (U_{nn})^{-1}$
- Obtain all the diagonals through multiple factorizations
- Local connectivity $\Rightarrow$ problem decomposition: partial factorizations feasible
- Proper ordering makes most of them identical: subproblems overlap $\Rightarrow$ dynamic programming
- Computational cost for all the diagonal entries of the inverse is of the same order as a single LU factorization!

Key Observations

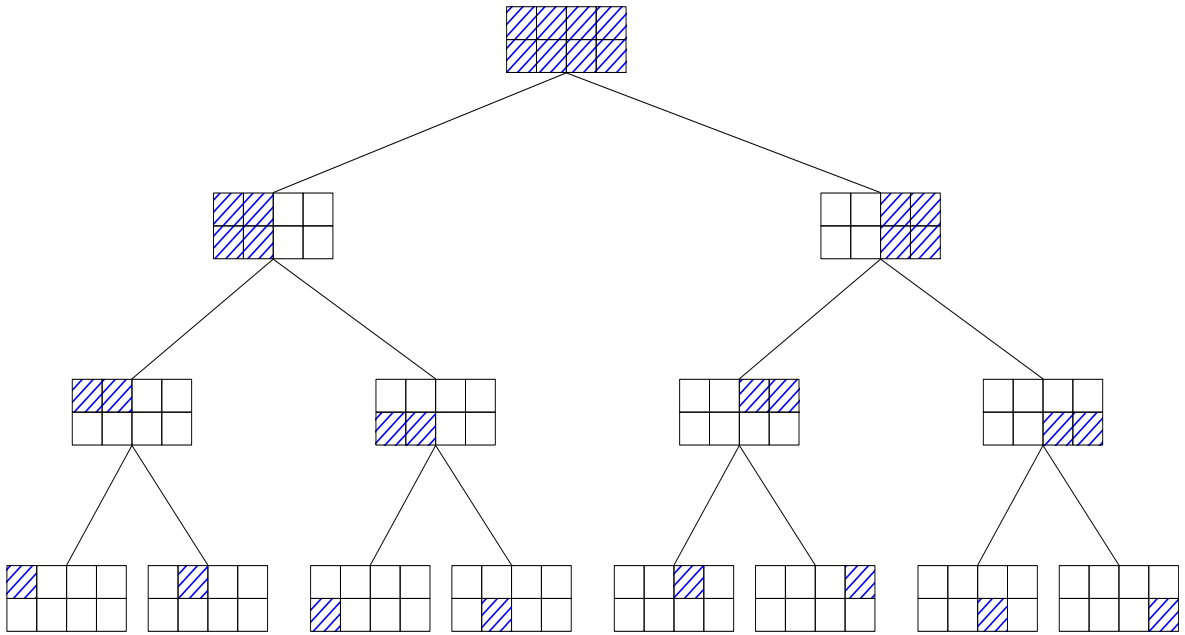
- Last entry in A^{-1} can be obtained through LU factorization: $(A^{-1})_{nn} = (U^{-1})_{nn} = (U_{nn})^{-1}$
- Obtain all the diagonals through multiple factorizations
- Local connectivity $\Rightarrow$ problem decomposition: partial factorizations feasible
- Proper ordering makes most of them identical: subproblems overlap $\Rightarrow$ dynamic programming
- Computational cost for all the diagonal entries of the inverse is of the same order as a single LU factorization!

Key Observations

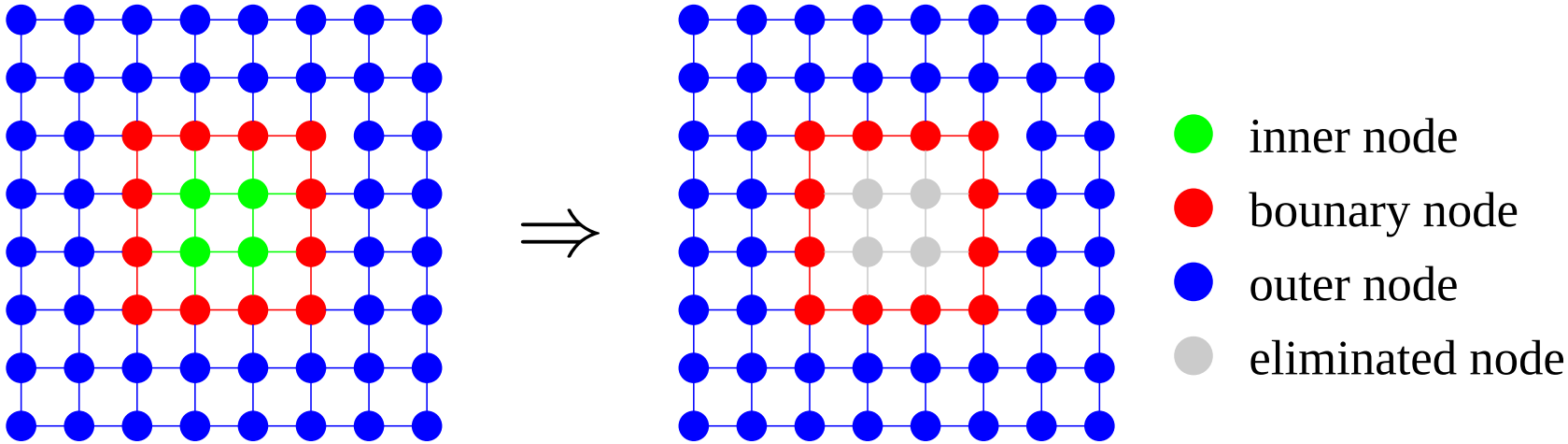
- Last entry in A^{-1} can be obtained through LU factorization: $(A^{-1})_{nn} = (U^{-1})_{nn} = (U_{nn})^{-1}$
- Obtain all the diagonals through multiple factorizations
- Local connectivity $\Rightarrow$ problem decomposition: partial factorizations feasible
- Proper ordering makes most of them identical: subproblems overlap $\Rightarrow$ dynamic programming
- Computational cost for all the diagonal entries of the inverse is of the same order as a single LU factorization!

Overall Structure: Partition Tree

- Order the mesh nodes in a way similar to nested dissection
- Partition the whole mesh and form a tree structure to exploit the subproblem overlap



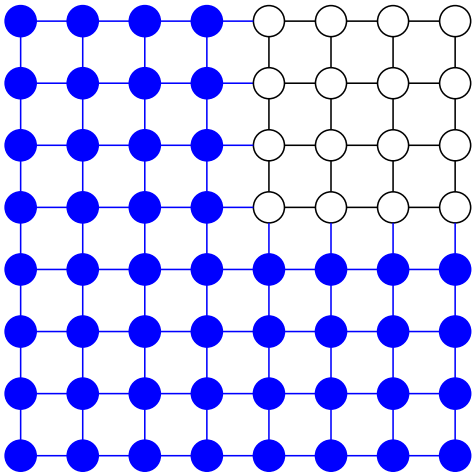
One Step of Elimination

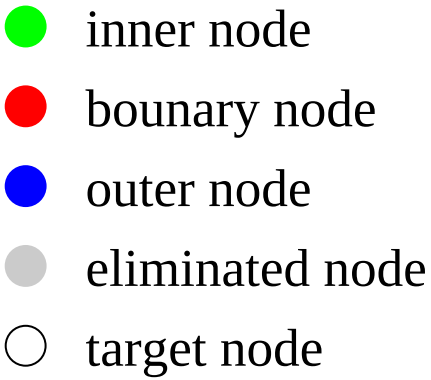


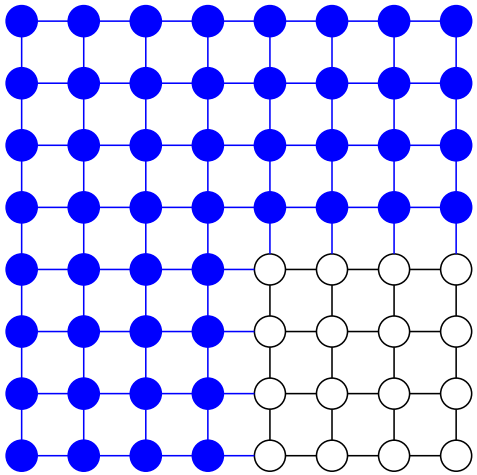
$$\begin{pmatrix} A(\bullet,\bullet) & A(\bullet,\bullet) & 0 \\ A(\bullet,\bullet) & A(\bullet,\bullet) & A(\bullet,\bullet) \\ 0 & A(\bullet,\bullet) & A(\bullet,\bullet) \end{pmatrix} \overset{\text{elimination}}{\Longrightarrow} \begin{pmatrix} A(\bullet,\bullet) & A(\bullet,\bullet) & 0 \\ 0 & A^*(\bullet,\bullet) & A(\bullet,\bullet) \\ 0 & A(\bullet,\bullet) & A(\bullet,\bullet) \end{pmatrix}$$

Gaussian elimination: $A^*(\bullet,\bullet) \overset{\text{def}}{=} A(\bullet,\bullet) - A(\bullet,\bullet)A(\bullet,\bullet)^{-1}A(\bullet,\bullet)$

Two Full Elimination Processes

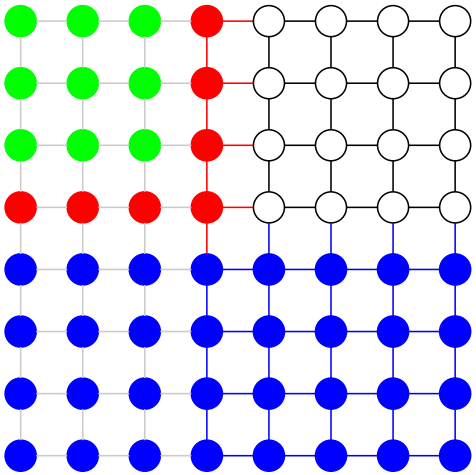


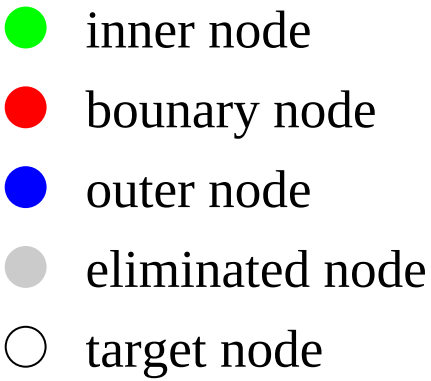


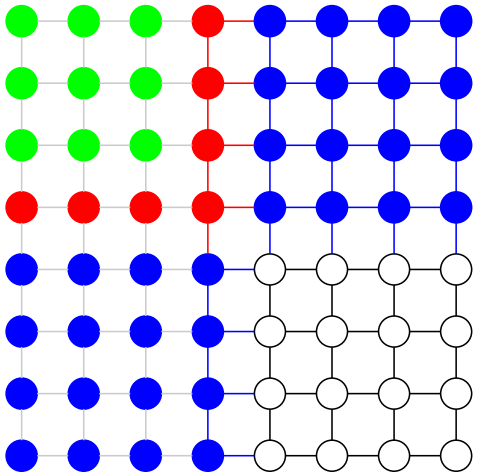


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes

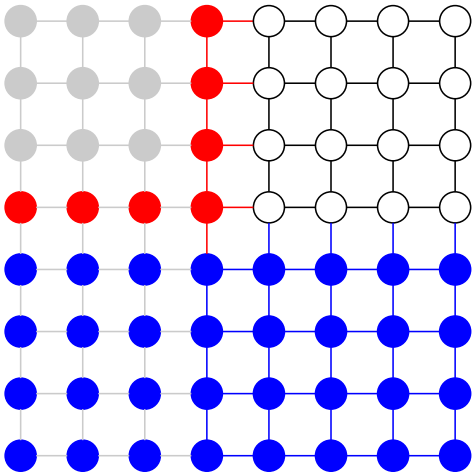


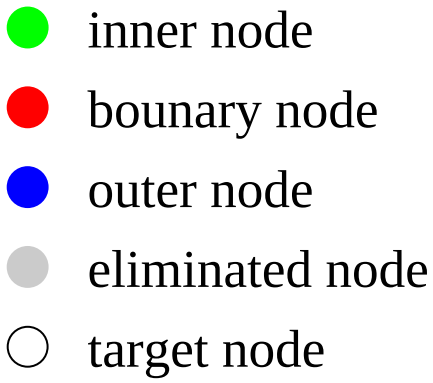


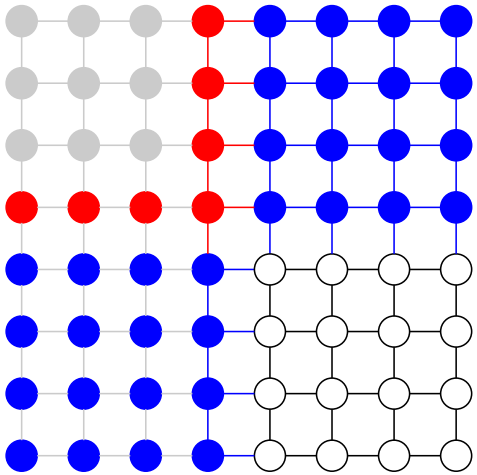


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes

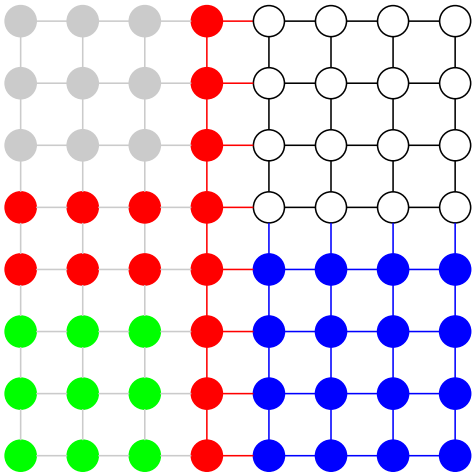


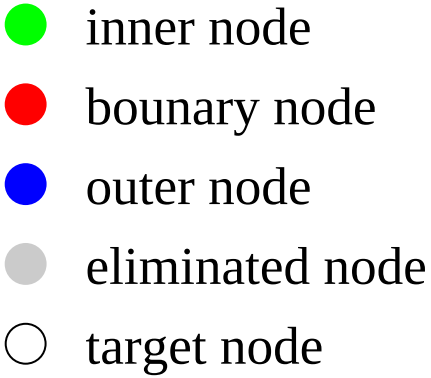


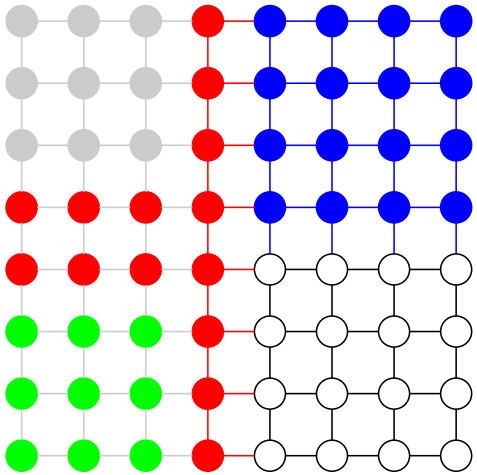


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes

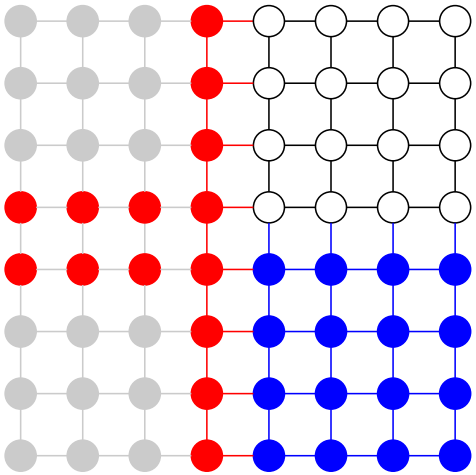


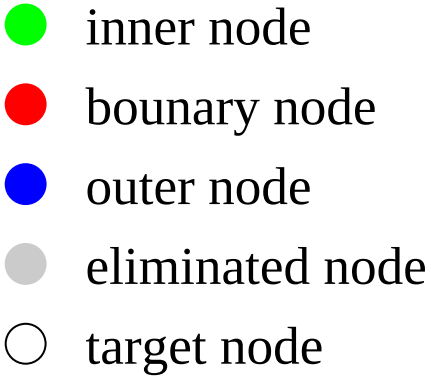


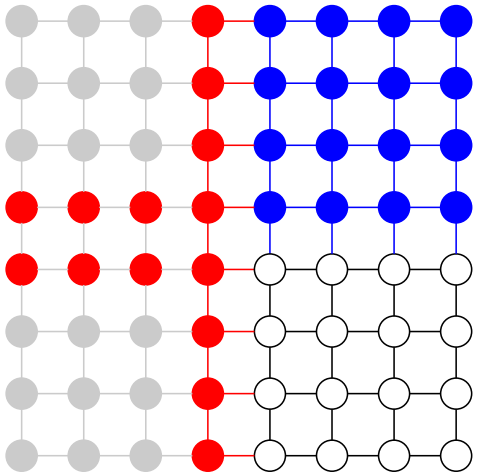


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes

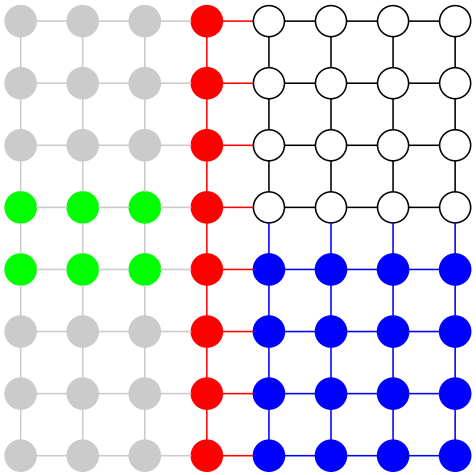


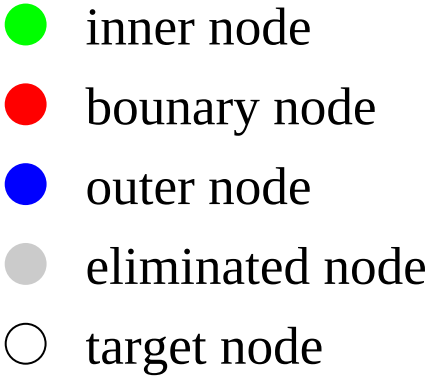


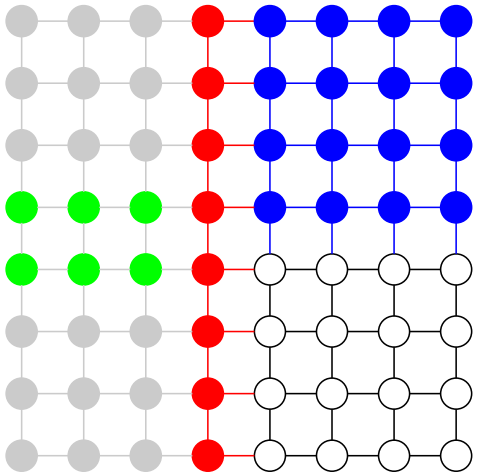


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes

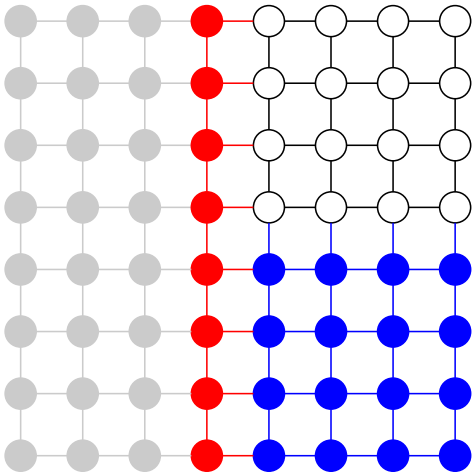


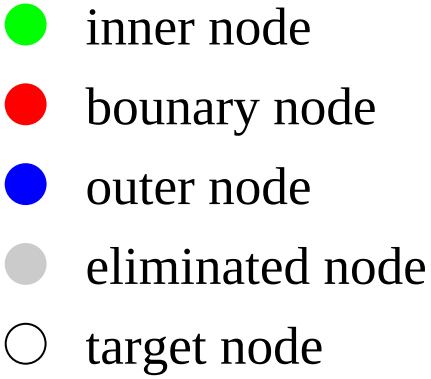


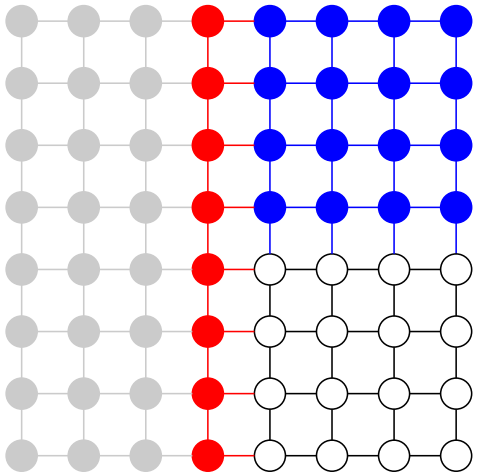


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes

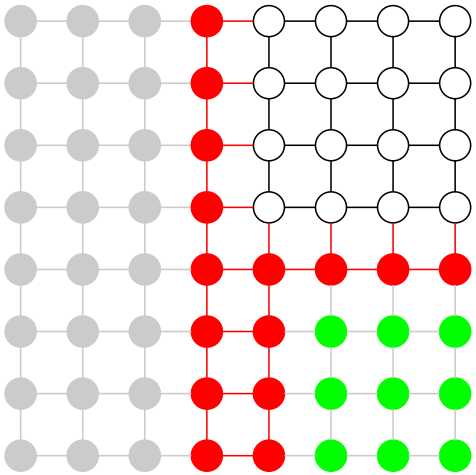


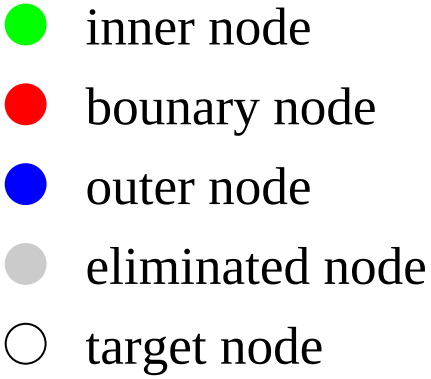


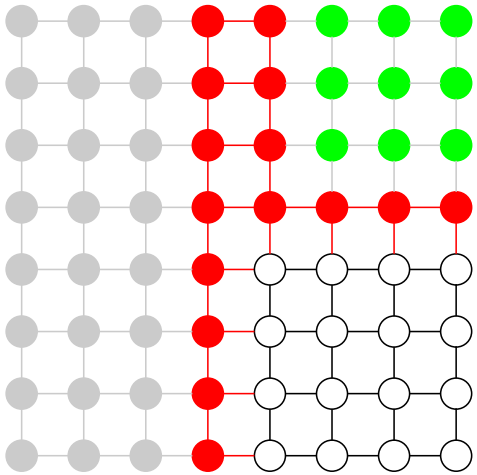


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes

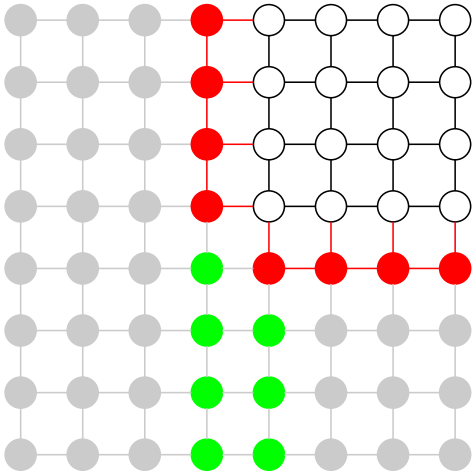




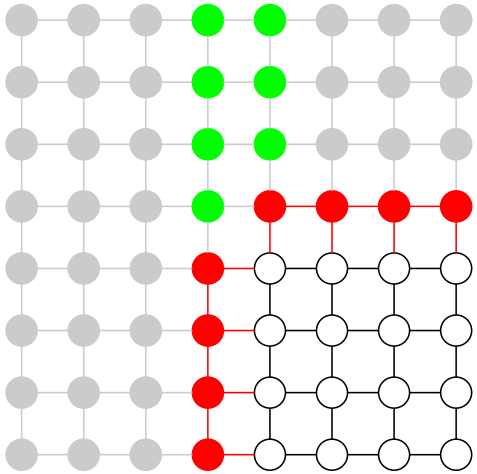


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes

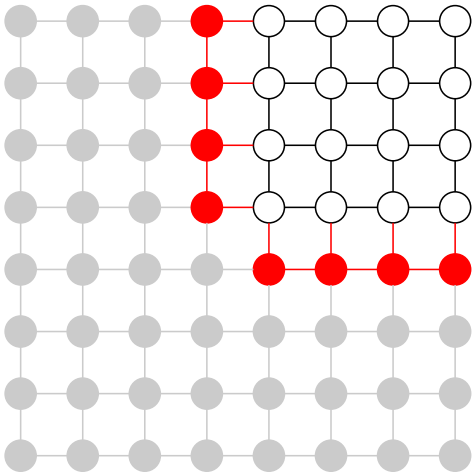


inner node
bounary node
outer node
eliminated node
target node

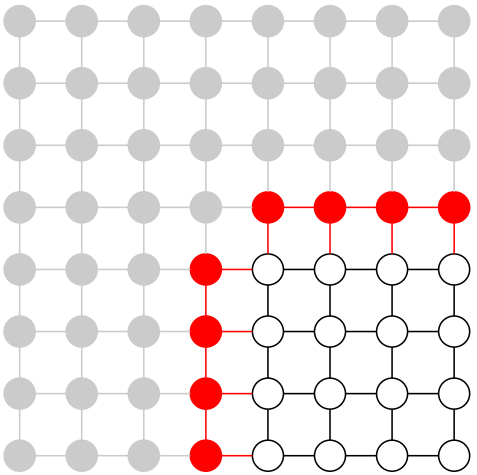


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes

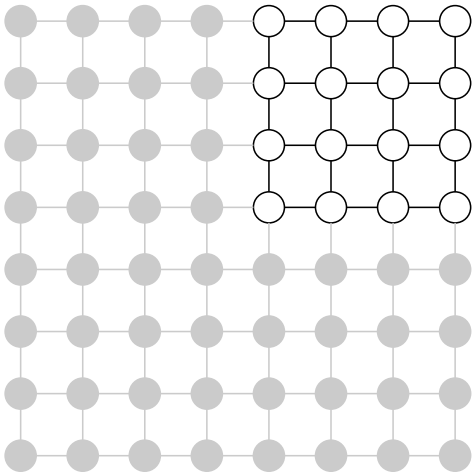


inner node
bounary node
outer node
eliminated node
target node

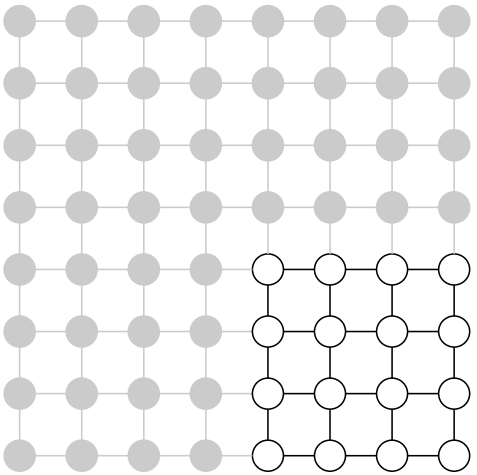


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes

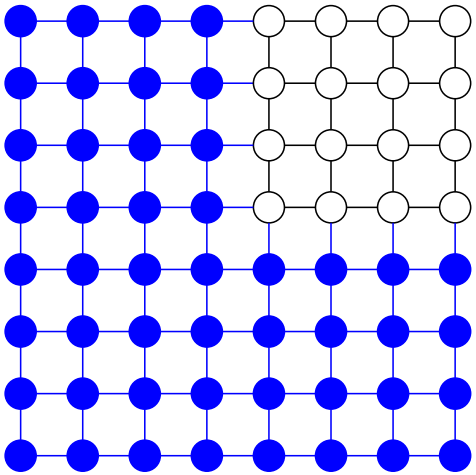


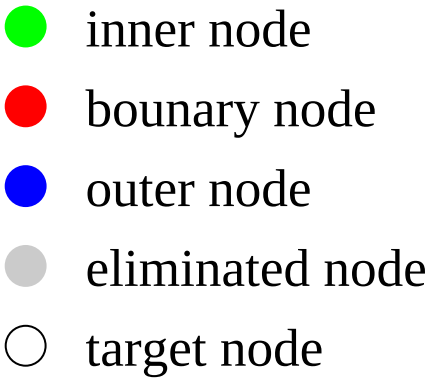
inner node
bounary node
outer node
eliminated node
target node

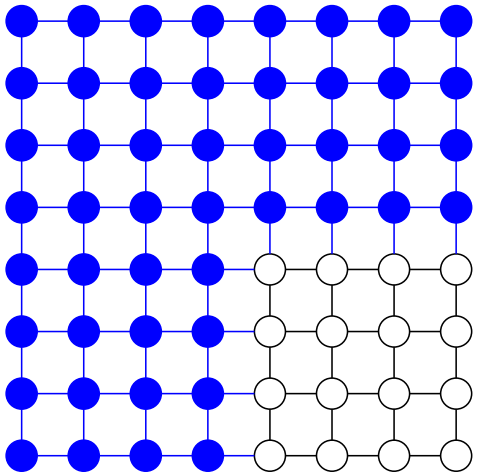


- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

Two Full Elimination Processes







- Keep partitioning the mesh to get small clusters
- Store results of each partial elimination
- The partial results could be reused

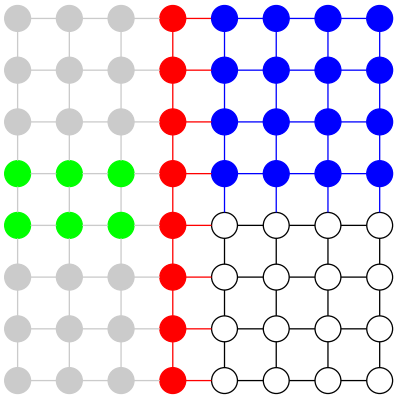
Extensions and Optimizations

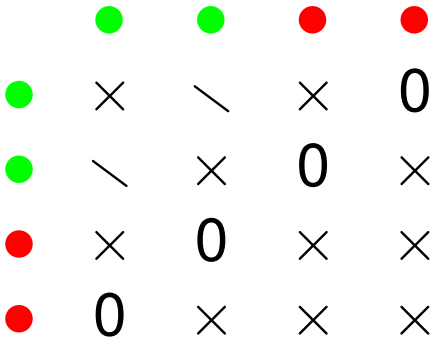
- $G^< = A^{-1}\Sigma A^{-\dagger}$ has similar sparsity pattern so our method is applicable as well
- Also for computing off-diagonal entries
- Extra sparsity
  - rewrite the one step elimination:
    $A^*(\bullet,\bullet) \stackrel{\text{def}}{=} A(\bullet,\bullet) - A(\bullet,\bullet)A(\bullet,\bullet)^{-1}A(\bullet,\bullet)$
  - these blocks are themselves sparse
- Exploit to optimize!
- The elimination preserves symmetry and this further reduces cost

Extensions and Optimizations

- $G^< = A^{-1}\Sigma A^{-\dagger}$ has similar sparsity pattern so our method is applicable as well
- Also for computing off-diagonal entries
- Extra sparsity
  - rewrite the one step elimination: $A^*(\bullet,\bullet) \stackrel{\text{def}}{=} A(\bullet,\bullet) - A(\bullet,\bullet)A(\bullet,\bullet)^{-1}A(\bullet,\bullet)$
  - these blocks are themselves sparse
- Exploit to optimize!
- The elimination preserves symmetry and this further reduces cost

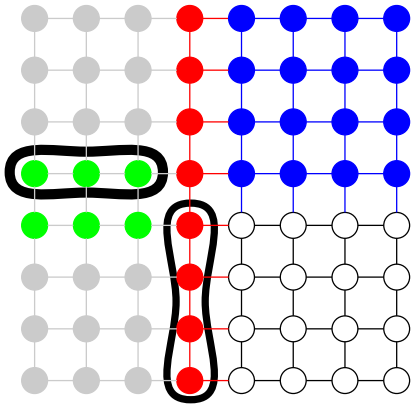
Extensions and Optimizations

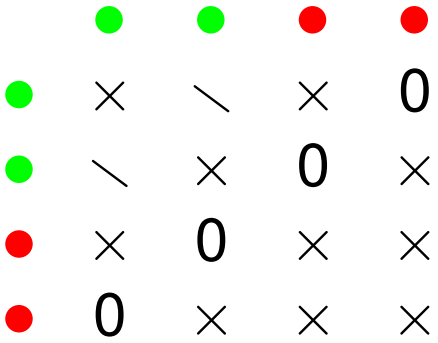




- $G^< = A^{-1}\Sigma A^{-\dagger}$ has similar sparsity pattern so our method is applicable as well
- Also for computing off-diagonal entries
- Extra sparsity
  - rewrite the one step elimination:
    $A^*(\bullet,\bullet) \overset{\text{def}}{=} A(\bullet,\bullet) - A(\bullet,\bullet)A(\bullet,\bullet)^{-1}A(\bullet,\bullet)$
  - these blocks are themselves sparse
- Exploit to optimize!
- The elimination preserves symmetry and this further reduces cost

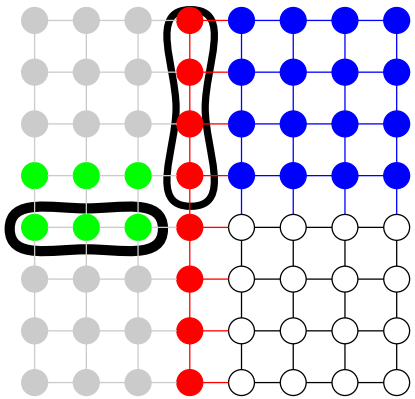
Extensions and Optimizations

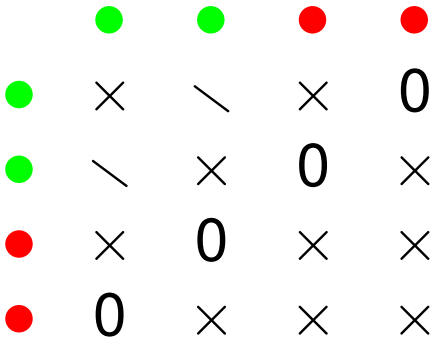




- $G^< = A^{-1}\Sigma A^{-\dagger}$ has similar sparsity pattern so our method is applicable as well
- Also for computing off-diagonal entries
- Extra sparsity
  - rewrite the one step elimination:
    $A^*(\bullet,\bullet) \stackrel{\text{def}}{=} A(\bullet,\bullet) - A(\bullet,\bullet)A(\bullet,\bullet)^{-1}A(\bullet,\bullet)$
  - these blocks are themselves sparse
- Exploit to optimize!
- The elimination preserves symmetry and this further reduces cost

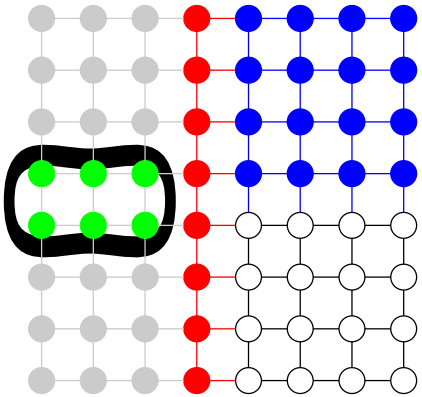
Extensions and Optimizations

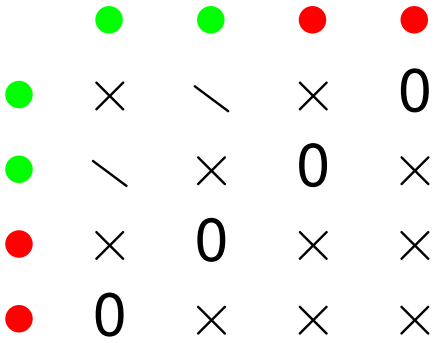




- $G^{<} = A^{-1}\Sigma A^{-\dagger}$ has similar sparsity pattern so our method is applicable as well
- Also for computing off-diagonal entries
- Extra sparsity
  - rewrite the one step elimination:
    $A^{*}(\bullet,\bullet) \stackrel{\text{def}}{=} A(\bullet,\bullet) - A(\bullet,\bullet)A(\bullet,\bullet)^{-1}A(\bullet,\bullet)$
  - these blocks are themselves sparse
- Exploit to optimize!
- The elimination preserves symmetry and this further reduces cost

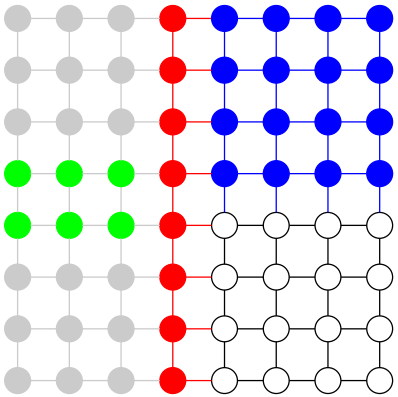
Extensions and Optimizations

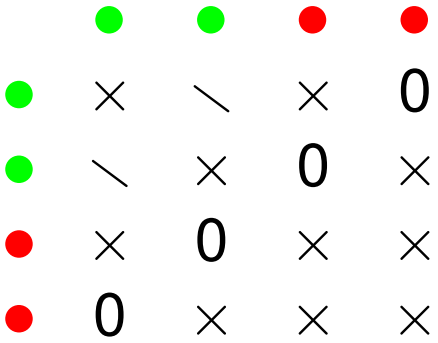




- $G^< = A^{-1}\Sigma A^{-\dagger}$ has similar sparsity pattern so our method is applicable as well
- Also for computing off-diagonal entries
- Extra sparsity
  - rewrite the one step elimination:
    $A^*(\bullet,\bullet) \overset{\text{def}}{=} A(\bullet,\bullet) - A(\bullet,\bullet)A(\bullet,\bullet)^{-1}A(\bullet,\bullet)$
  - these blocks are themselves sparse
- Exploit to optimize!
- The elimination preserves symmetry and this further reduces cost

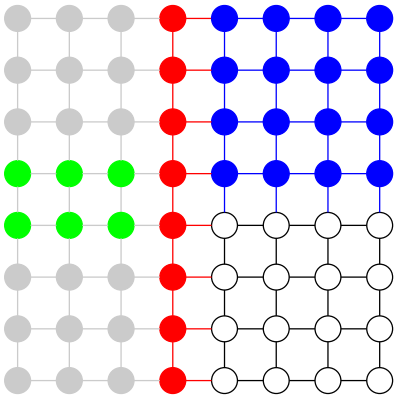
Extensions and Optimizations

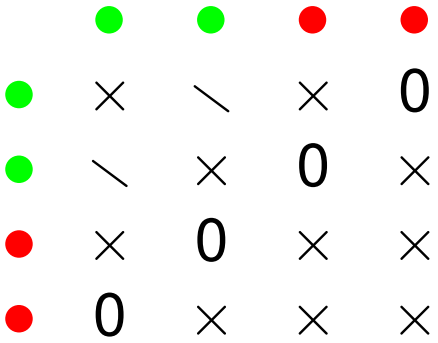




- $G^< = A^{-1}\Sigma A^{-\dagger}$ has similar sparsity pattern so our method is applicable as well
- Also for computing off-diagonal entries
- Extra sparsity
  - rewrite the one step elimination:
    $A^*(\bullet,\bullet) \stackrel{\text{def}}{=} A(\bullet,\bullet) - A(\bullet,\bullet)A(\bullet,\bullet)^{-1}A(\bullet,\bullet)$
  - these blocks are themselves sparse
- Exploit to optimize!
- The elimination preserves symmetry and this further reduces cost

Extensions and Optimizations

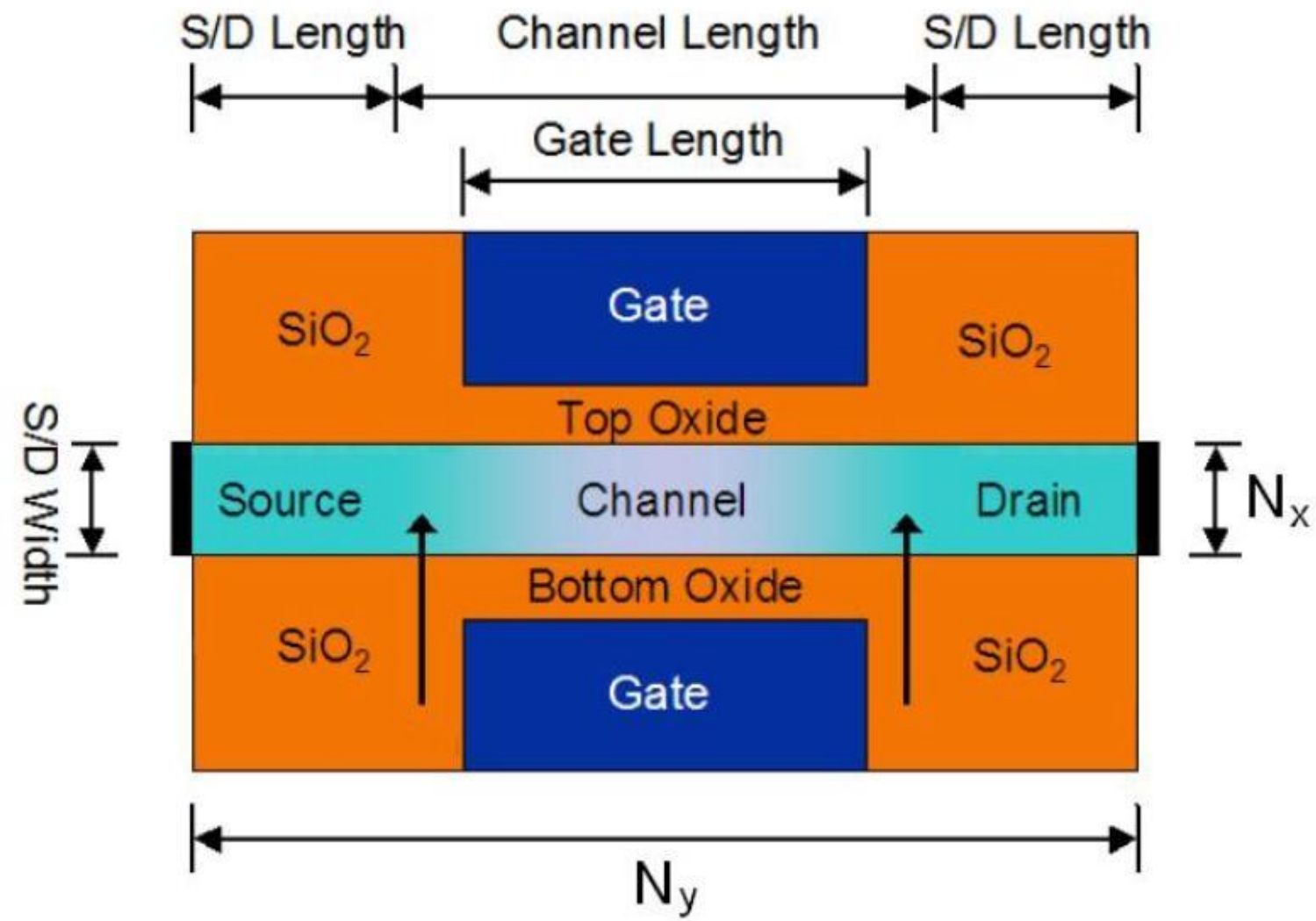




- $G^< = A^{-1}\Sigma A^{-\dagger}$ has similar sparsity pattern so our method is applicable as well
- Also for computing off-diagonal entries
- Extra sparsity
  - rewrite the one step elimination:
    $A^*(\bullet,\bullet) \stackrel{\text{def}}{=} A(\bullet,\bullet) - A(\bullet,\bullet)A(\bullet,\bullet)^{-1}A(\bullet,\bullet)$
  - these blocks are themselves sparse
- Exploit to optimize!
- The elimination preserves symmetry and this further reduces cost

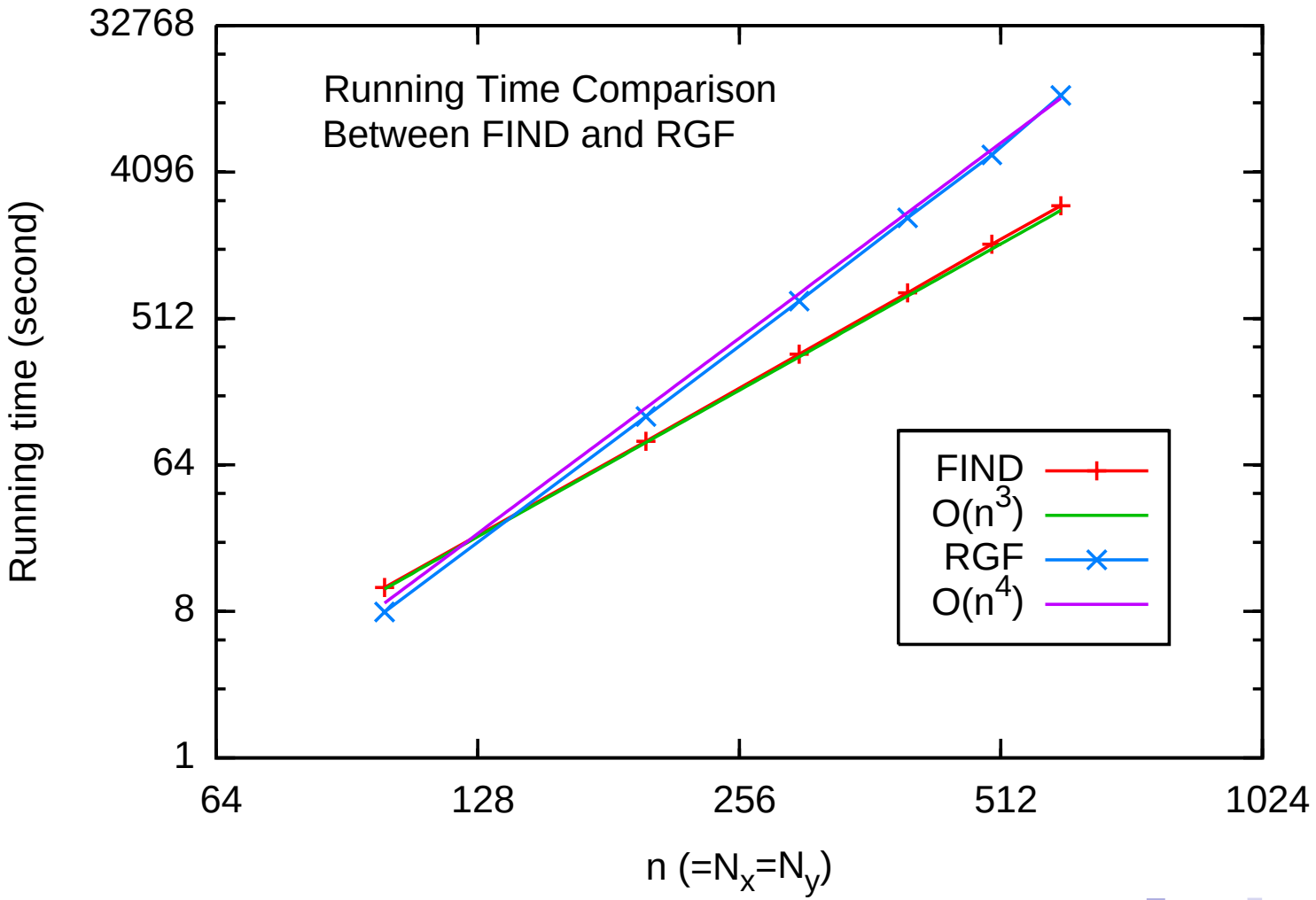
Outline

Simulation Device

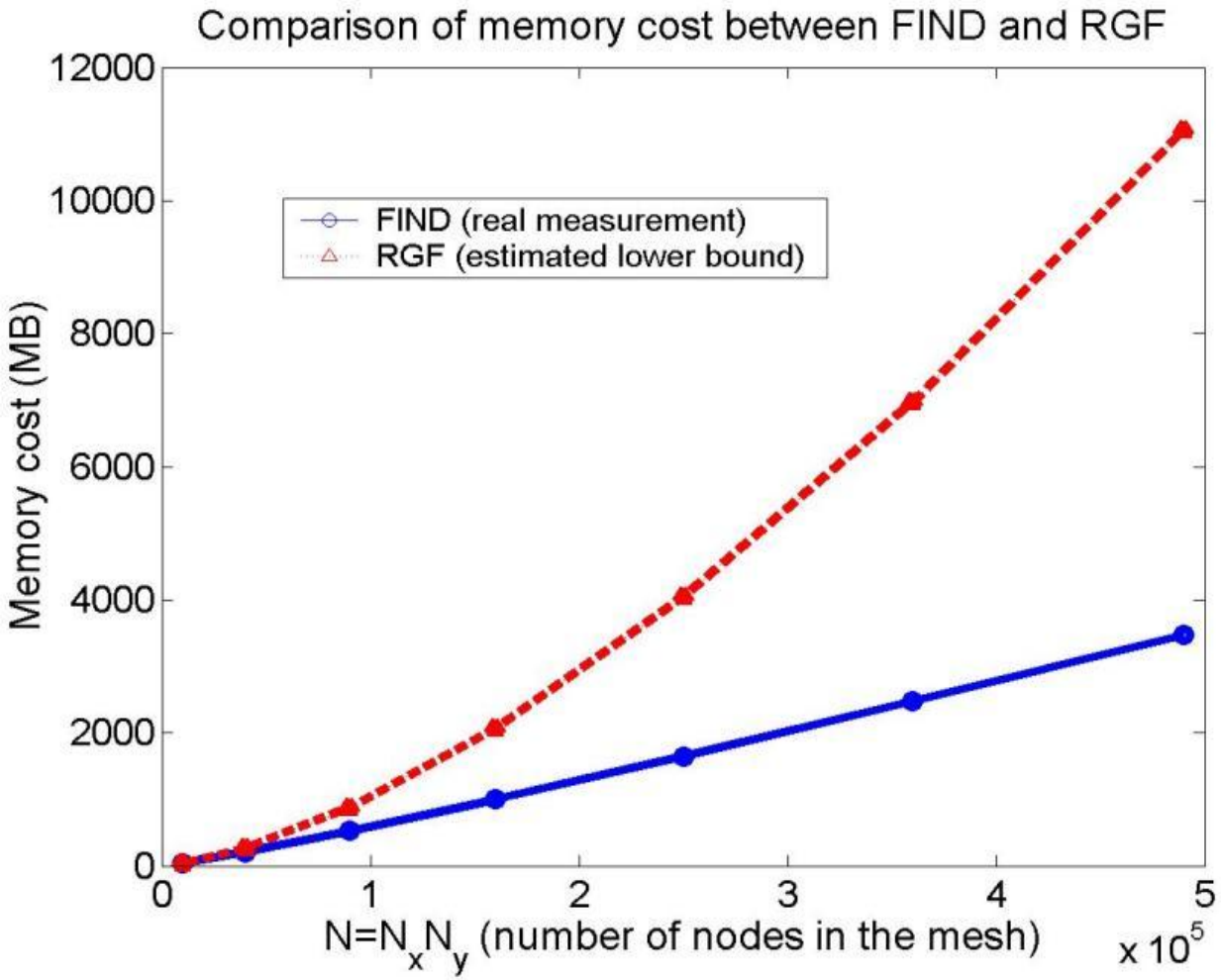


Running Time Comparison

Log-Log Scale with Reference Lines



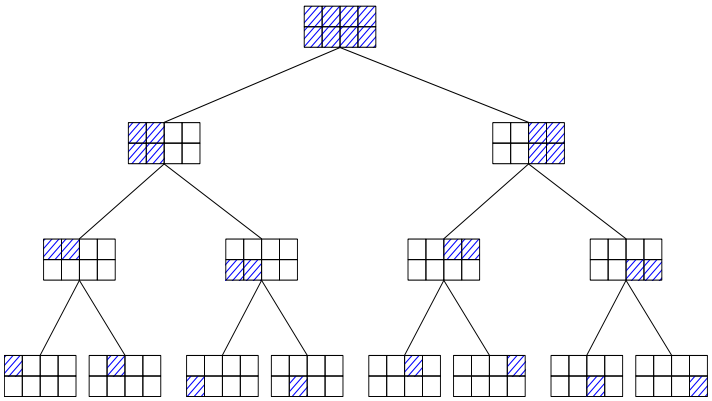
Memory Cost Comparison



- FIND: $O(N \log(N))$
- RGF: $O(N^{3/2})$

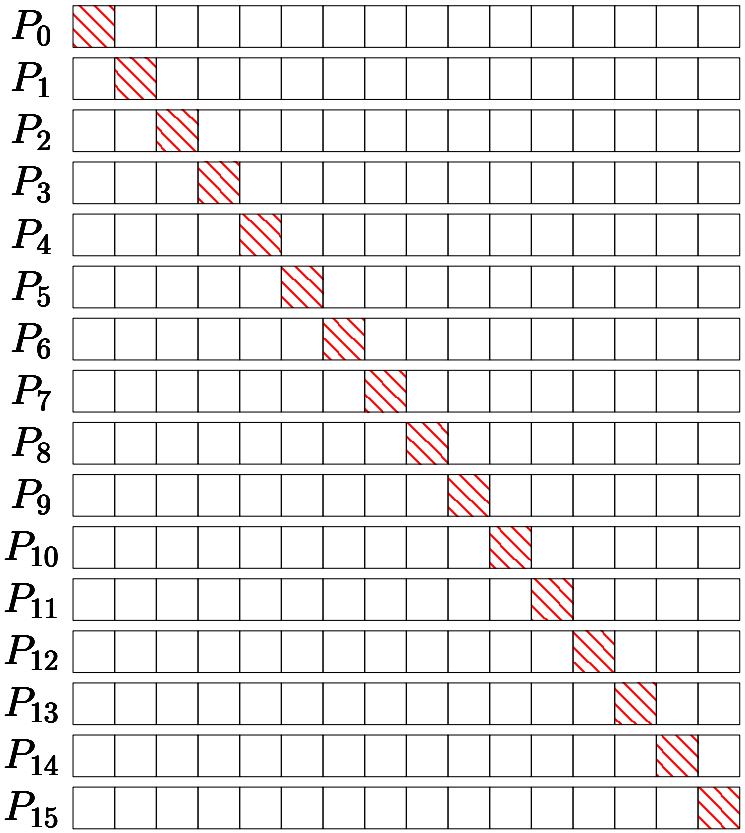
Outline

How to Parallelize?



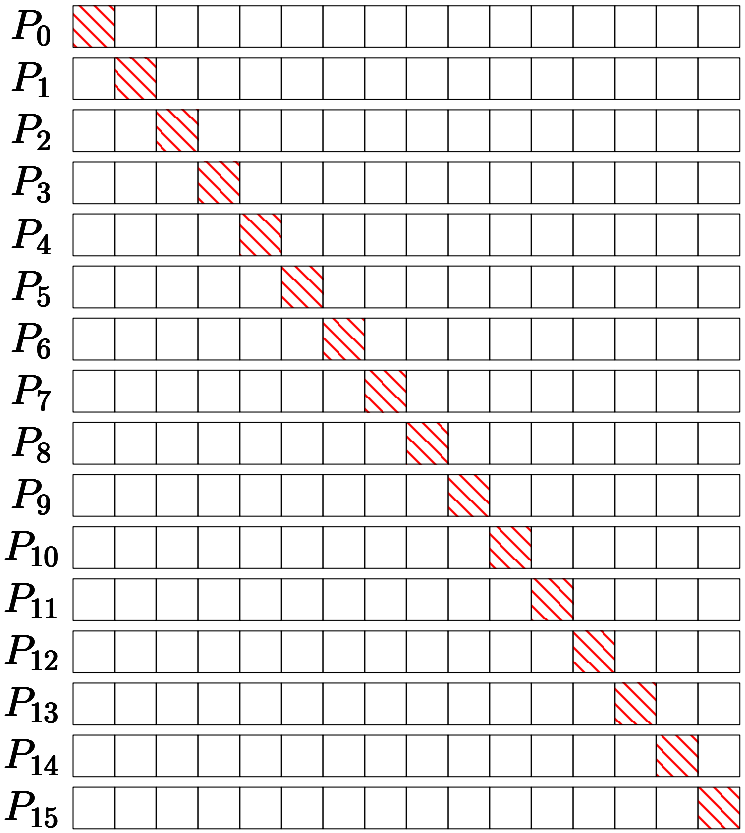
- Straightforward for leaf clusters
- Top level clusters dominate running time with less degree of parallelism
- Use the idle processors for redundant computations
- More floating point operations but shorter wall clock time
- Works for 1D, 2D, and 3D domains

Problem and Processor Settings



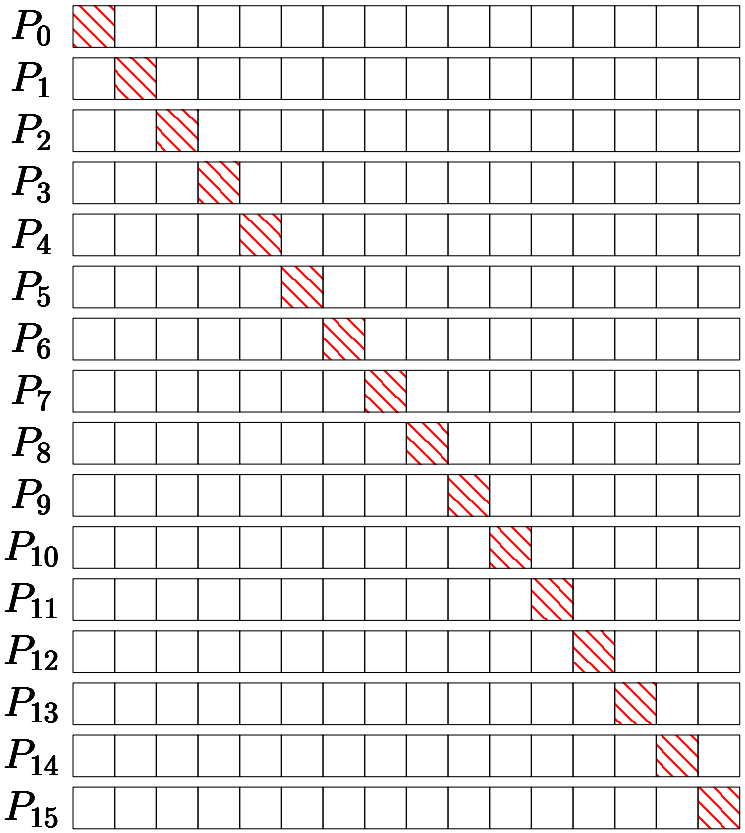
- 16 processors, 16 clusters in 1D
- One target cluster per processor
- Keep merging all the other clusters until we have them all merged as the complement of the target cluster
- Eliminate the merged complement clusters and compute the inverse

Problem and Processor Settings



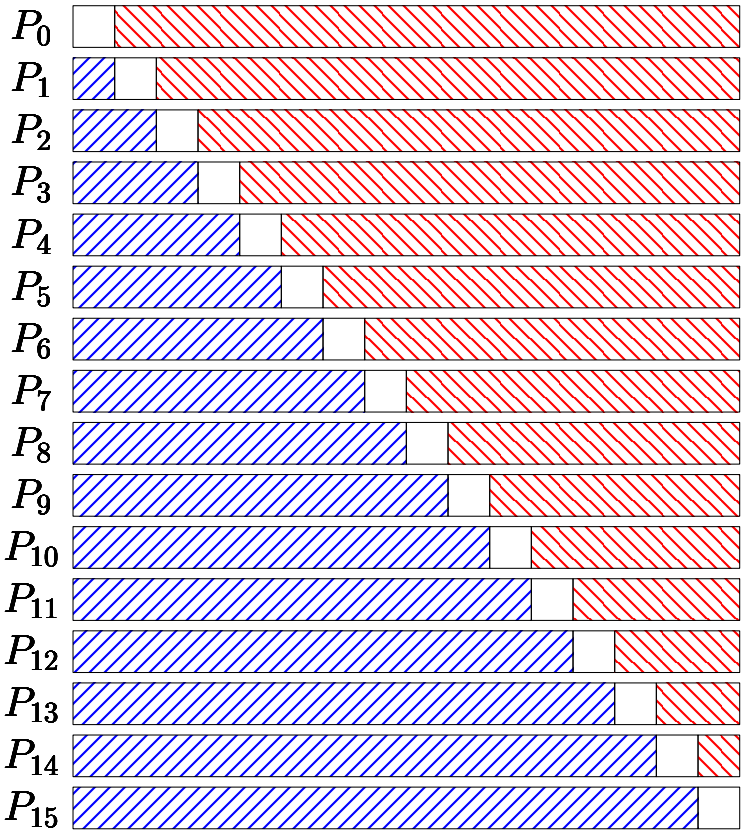
- 16 processors, 16 clusters in 1D
- One target cluster per processor
- Keep merging all the other clusters until we have them all merged as the complement of the target cluster
- Eliminate the merged complement clusters and compute the inverse

Problem and Processor Settings



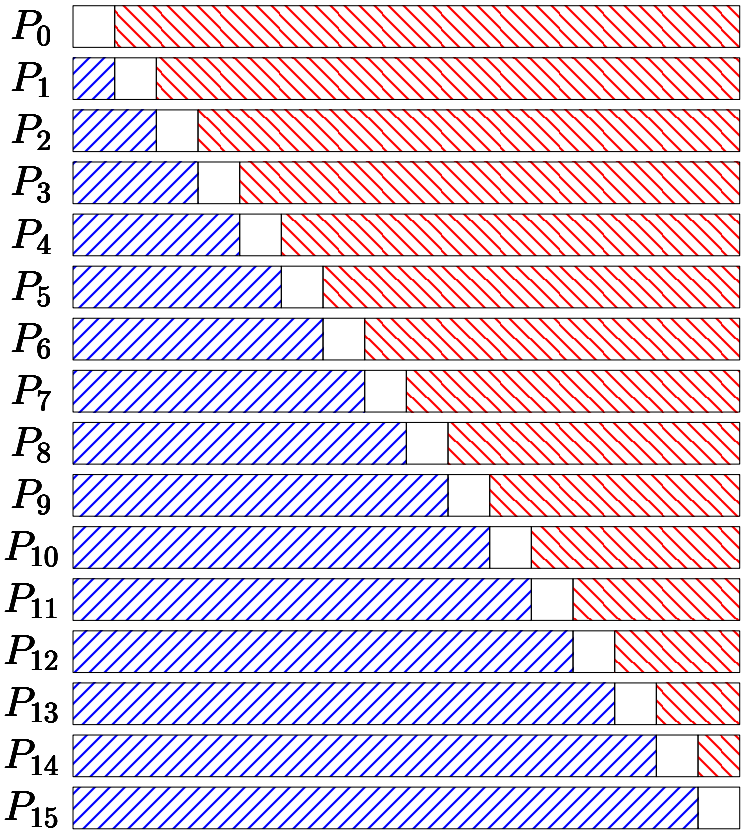
- 16 processors, 16 clusters in 1D
- One target cluster per processor
- Keep merging all the other clusters until we have them all merged as the complement of the target cluster
- Eliminate the merged complement clusters and compute the inverse

Problem and Processor Settings



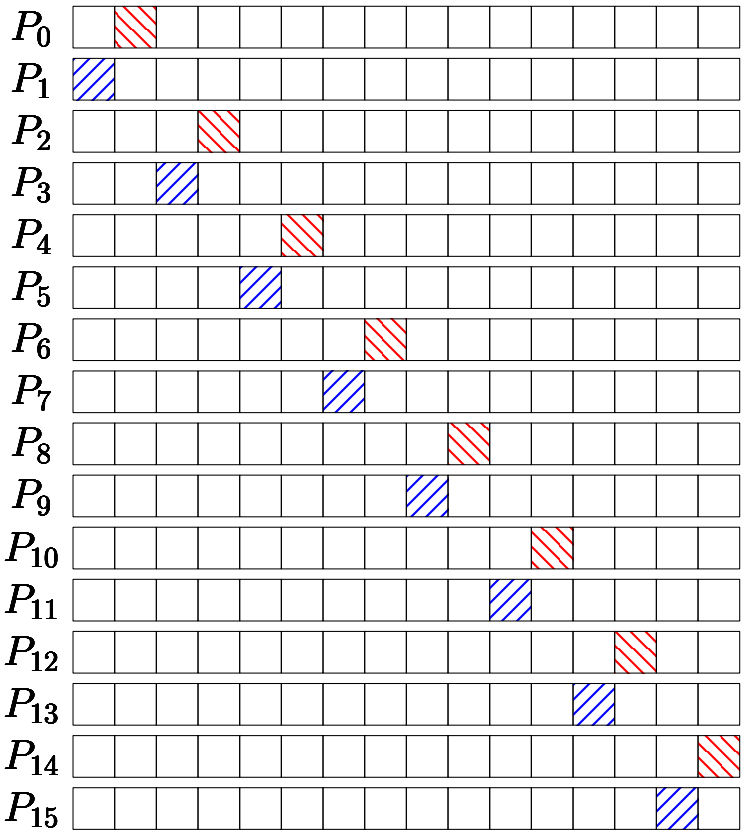
- 16 processors, 16 clusters in 1D
- One target cluster per processor
- Keep merging all the other clusters until we have them all merged as the complement of the target cluster
- Eliminate the merged complement clusters and compute the inverse

Problem and Processor Settings



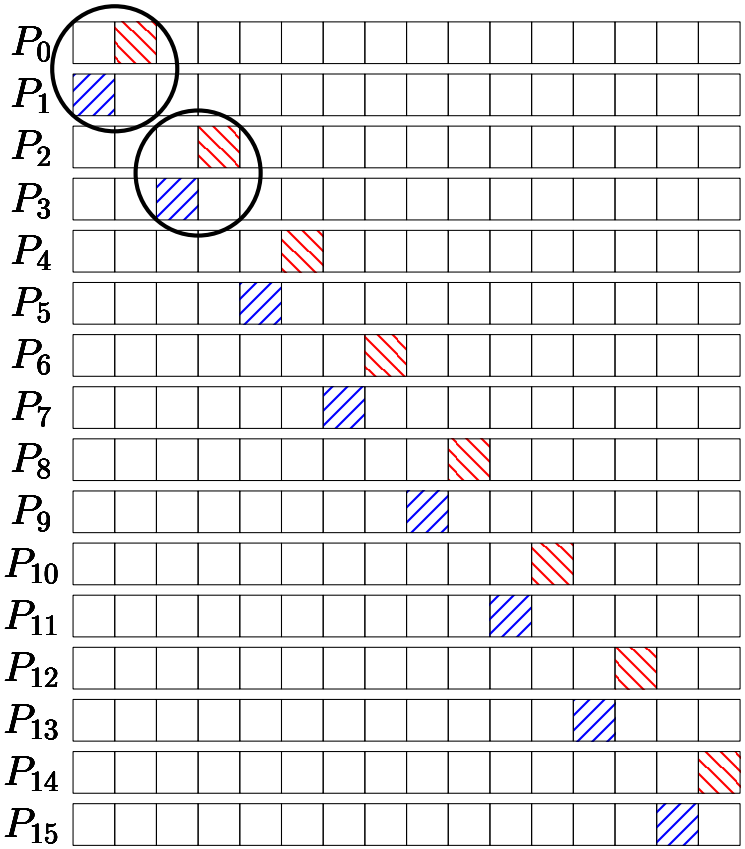
- 16 processors, 16 clusters in 1D
- One target cluster per processor
- Keep merging all the other clusters until we have them all merged as the complement of the target cluster
- Eliminate the merged complement clusters and compute the inverse

Detailed Merging Process



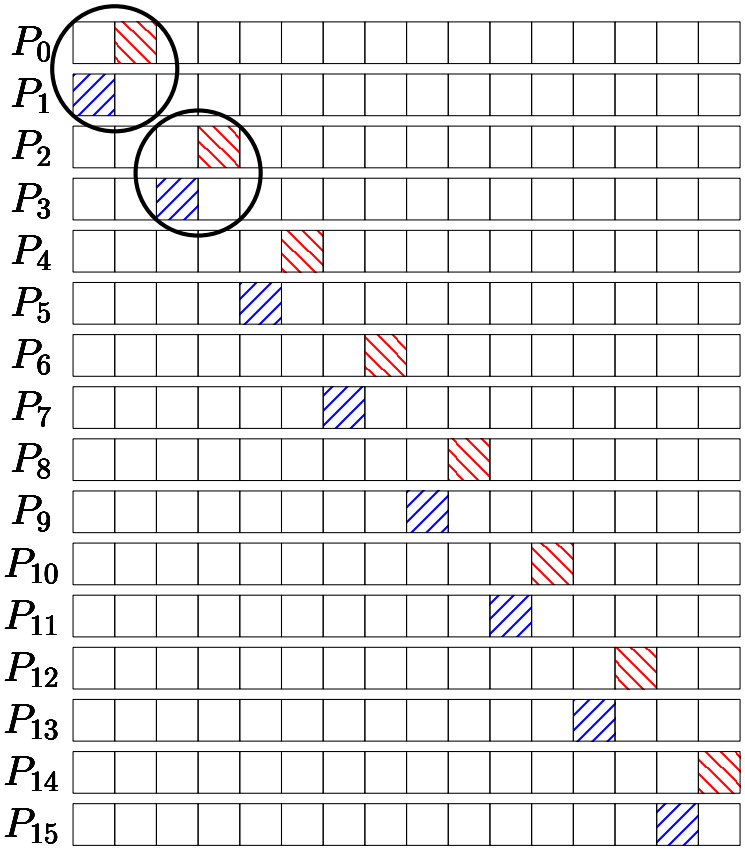
- Each processor keeps the complement of its target cluster with respect to the current subdomain
- Start with subdomain of size 2
- Expand to subdomains of size 4
- Some processors are idle
- Use them to prepare for the next subdomain expansion
- Until the subdomain is expanded to the whole domain
- Additional speedup of factor 2

Detailed Merging Process



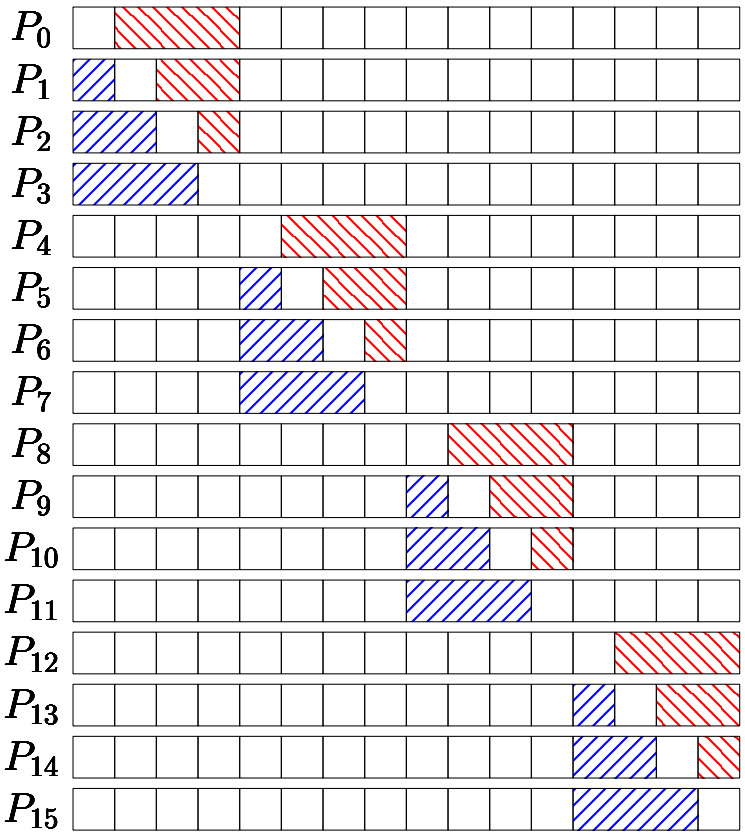
- Each processor keeps the complement of its target cluster with respect to the current subdomain
- Start with subdomain of size 2
- Expand to subdomains of size 4
- Some processors are idle
- Use them to prepare for the next subdomain expansion
- Until the subdomain is expanded to the whole domain
- Additional speedup of factor 2

Detailed Merging Process



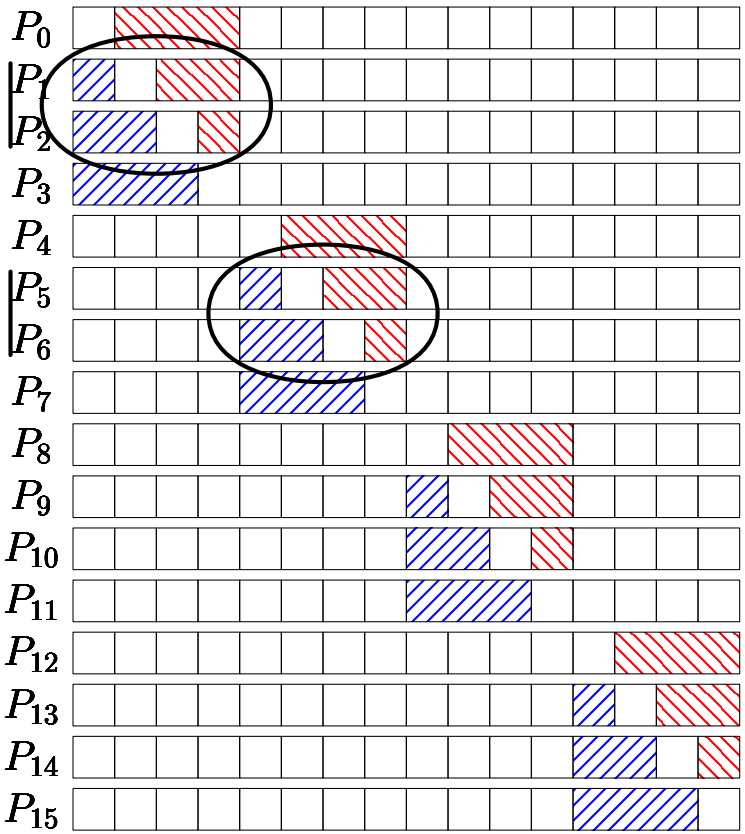
- Each processor keeps the complement of its target cluster with respect to the current subdomain
- Start with subdomain of size 2
- Expand to subdomains of size 4
- Some processors are idle
- Use them to prepare for the next subdomain expansion
- Until the subdomain is expanded to the whole domain
- Additional speedup of factor 2

Detailed Merging Process



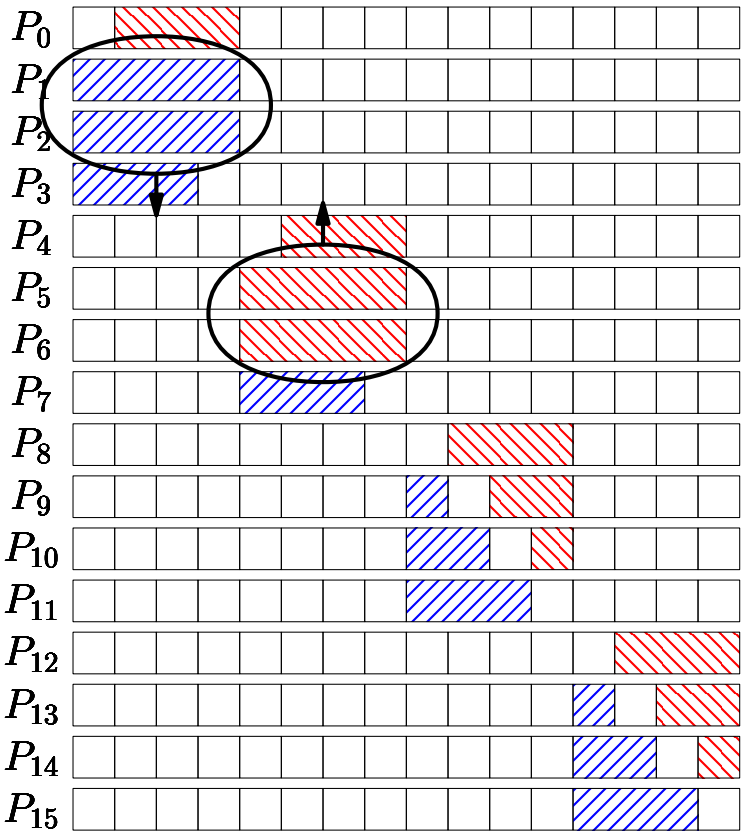
- Each processor keeps the complement of its target cluster with respect to the current subdomain
- Start with subdomain of size 2
- Expand to subdomains of size 4
- Some processors are idle
- Use them to prepare for the next subdomain expansion
- Until the subdomain is expanded to the whole domain
- Additional speedup of factor 2

Detailed Merging Process



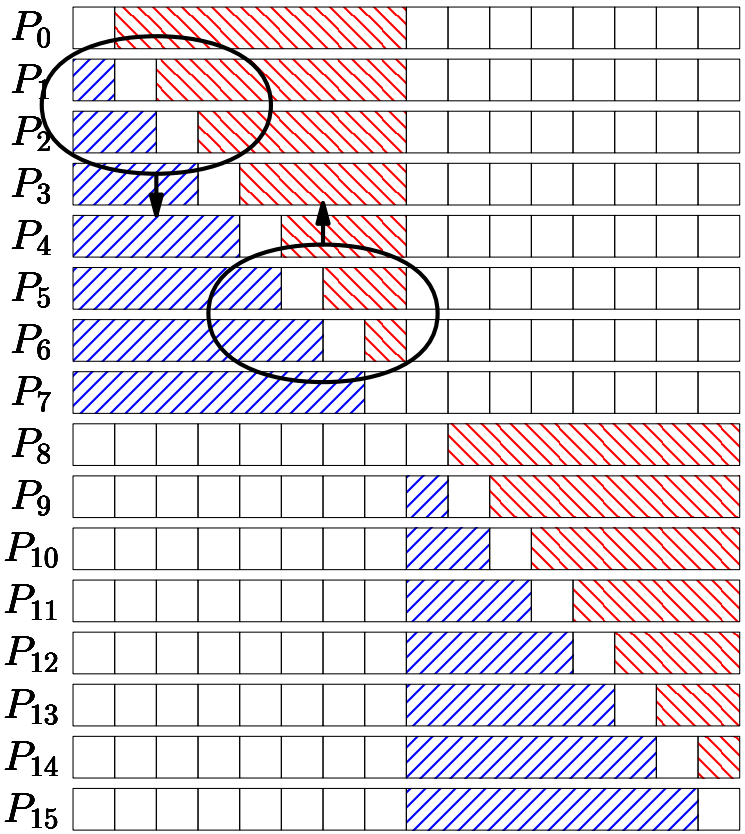
- Each processor keeps the complement of its target cluster with respect to the current subdomain
- Start with subdomain of size 2
- Expand to subdomains of size 4
- Some processors are idle
- Use them to prepare for the next subdomain expansion
- Until the subdomain is expanded to the whole domain
- Additional speedup of factor 2

Detailed Merging Process



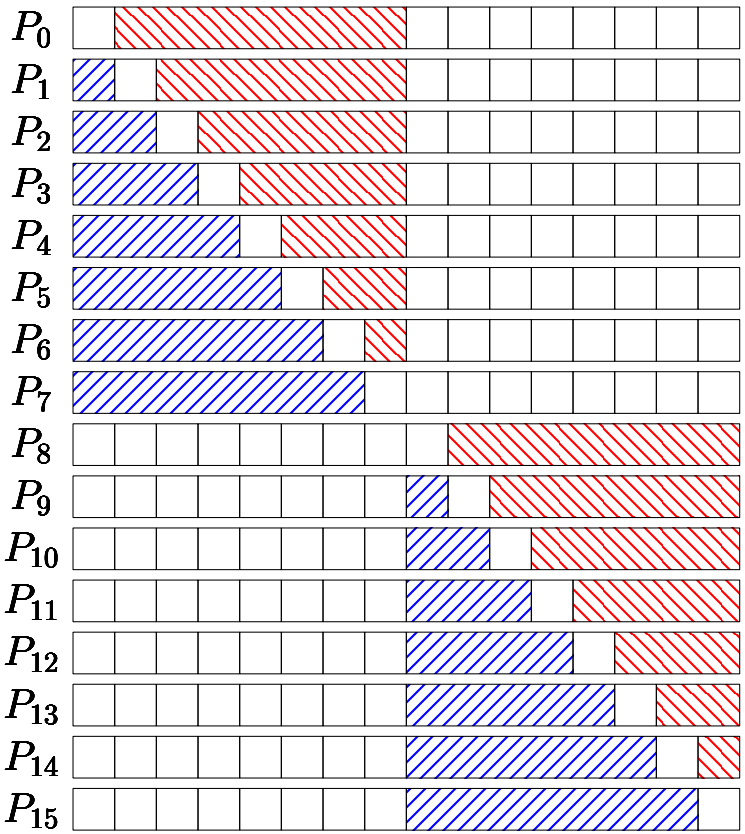
- Each processor keeps the complement of its target cluster with respect to the current subdomain
- Start with subdomain of size 2
- Expand to subdomains of size 4
- Some processors are idle
- Use them to prepare for the next subdomain expansion
- Until the subdomain is expanded to the whole domain
- Additional speedup of factor 2

Detailed Merging Process



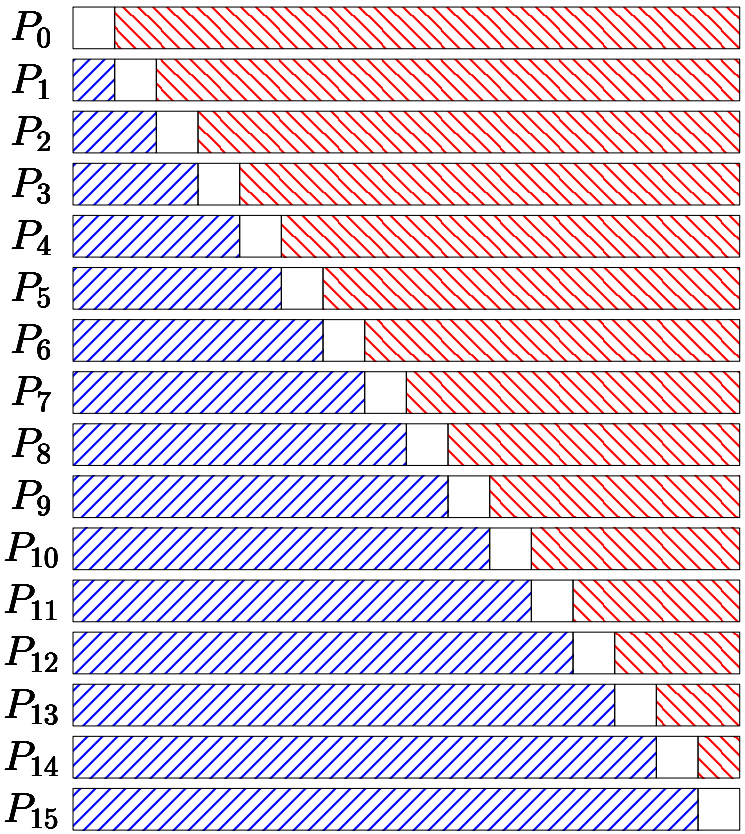
- Each processor keeps the complement of its target cluster with respect to the current subdomain
- Start with subdomain of size 2
- Expand to subdomains of size 4
- Some processors are idle
- Use them to prepare for the next subdomain expansion
- Until the subdomain is expanded to the whole domain
- Additional speedup of factor 2

Detailed Merging Process



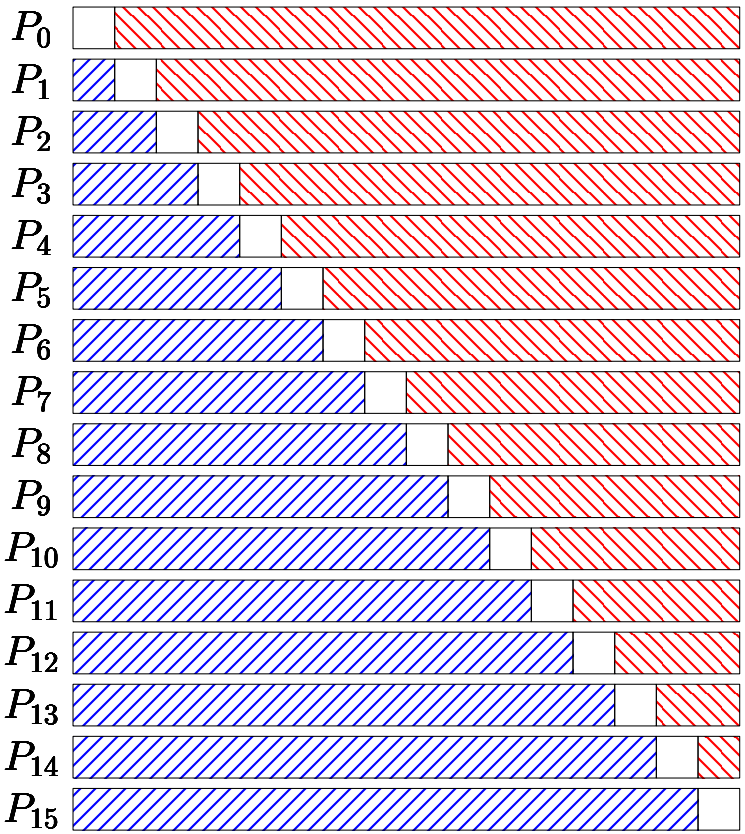
- Each processor keeps the complement of its target cluster with respect to the current subdomain
- Start with subdomain of size 2
- Expand to subdomains of size 4
- Some processors are idle
- Use them to prepare for the next subdomain expansion
- Until the subdomain is expanded to the whole domain
- Additional speedup of factor 2

Detailed Merging Process



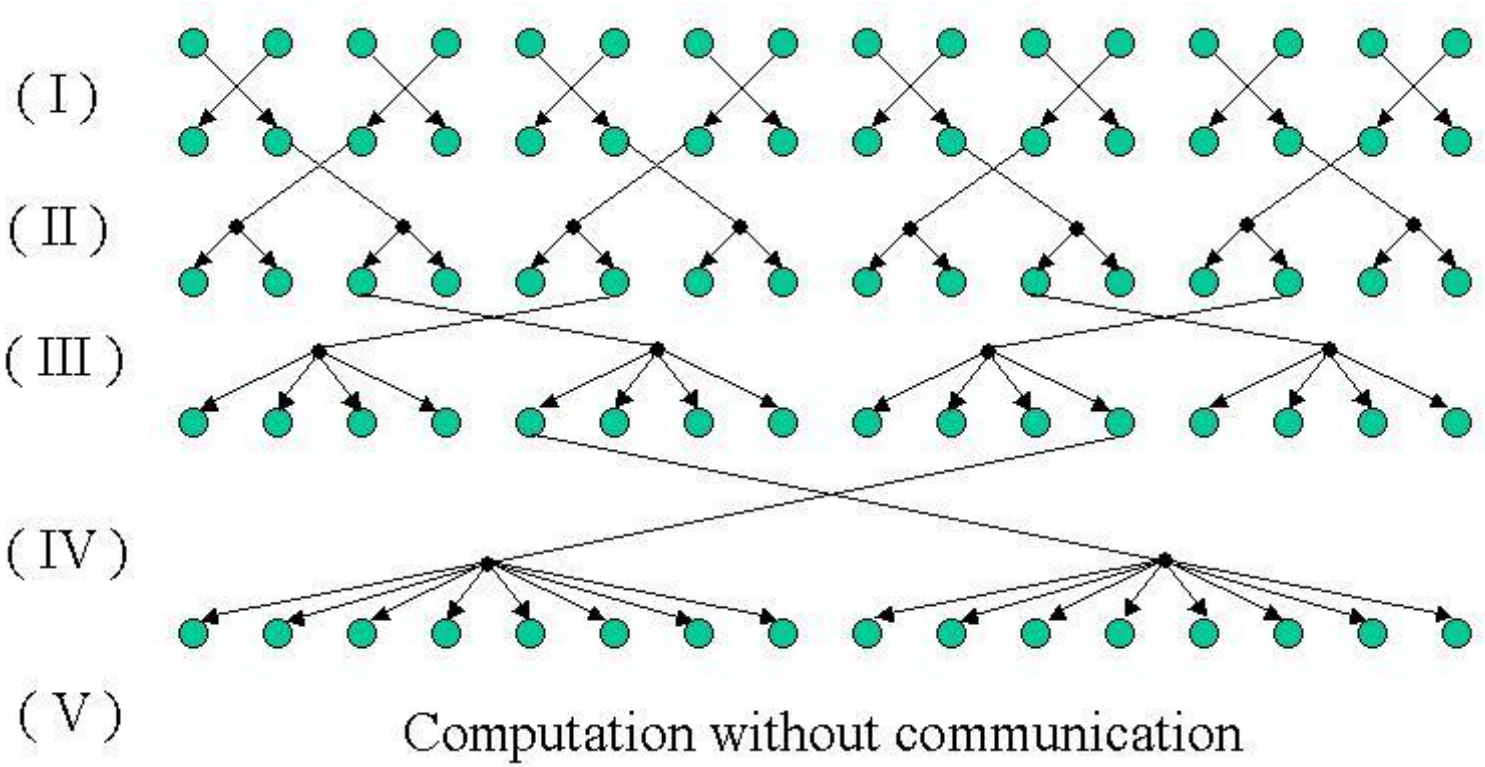
- Each processor keeps the complement of its target cluster with respect to the current subdomain
- Start with subdomain of size 2
- Expand to subdomains of size 4
- Some processors are idle
- Use them to prepare for the next subdomain expansion
- Until the subdomain is expanded to the whole domain
- Additional speedup of factor 2

Detailed Merging Process



- Each processor keeps the complement of its target cluster with respect to the current subdomain
- Start with subdomain of size 2
- Expand to subdomains of size 4
- Some processors are idle
- Use them to prepare for the next subdomain expansion
- Until the subdomain is expanded to the whole domain
- Additional speedup of factor 2

Communication Pattern



Summary

- Direct method for fast inverse
- Two extensions, two optimizations
- An optimal parallel scheme
- Collaboration with other groups for more applications