# Parallelizing Irregular Gauss-Seidel Using Full Sparse Tiling


Michelle Mills Strout

in collaboration with Dave Rostron, Jeanne Ferrante, and Larry Carter

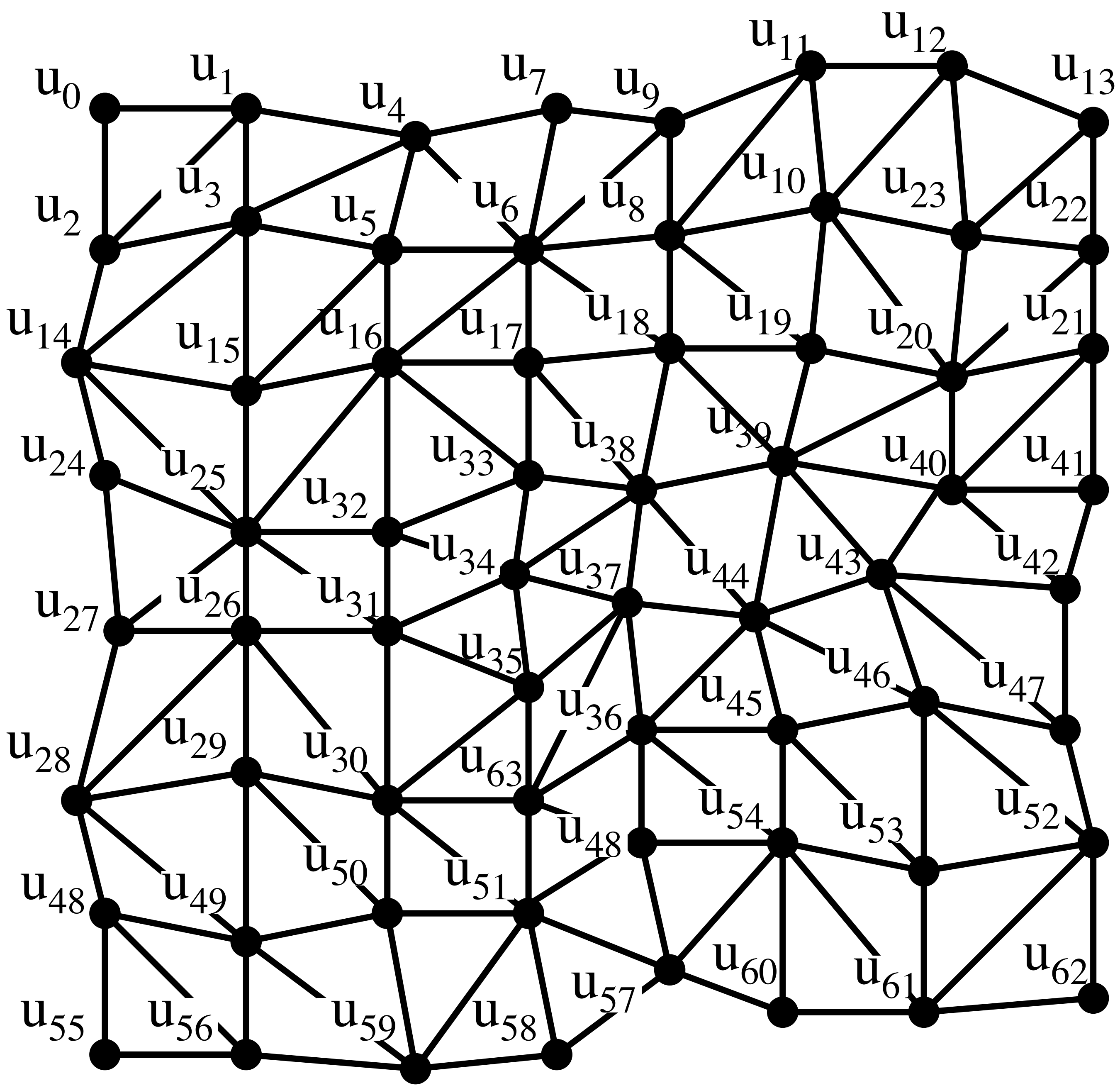


partially supported by CSCAPES (www.cscapes.org), an institute funded by DOE's Office of Science


FRACTAL

April 26, 2008

# Goal: Make computations over meshes fast



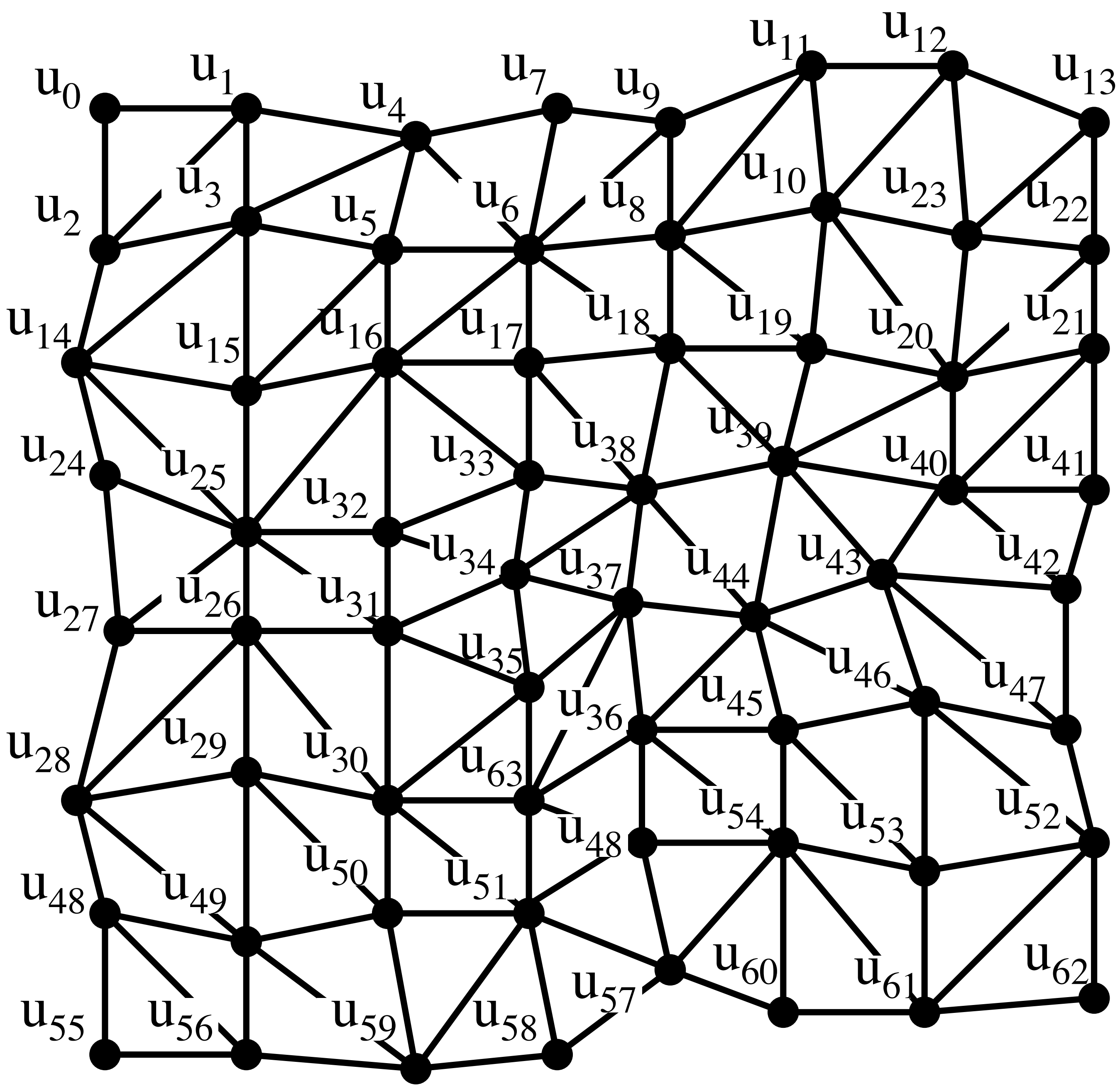

- Iterative smoothers such as Gauss-Seidel dominate the execution time of finite element applications
- Need to effectively utilize the memory hierarchy
- Performance should scale to multiple processors

# Full Sparse Tiling to the Rescue!

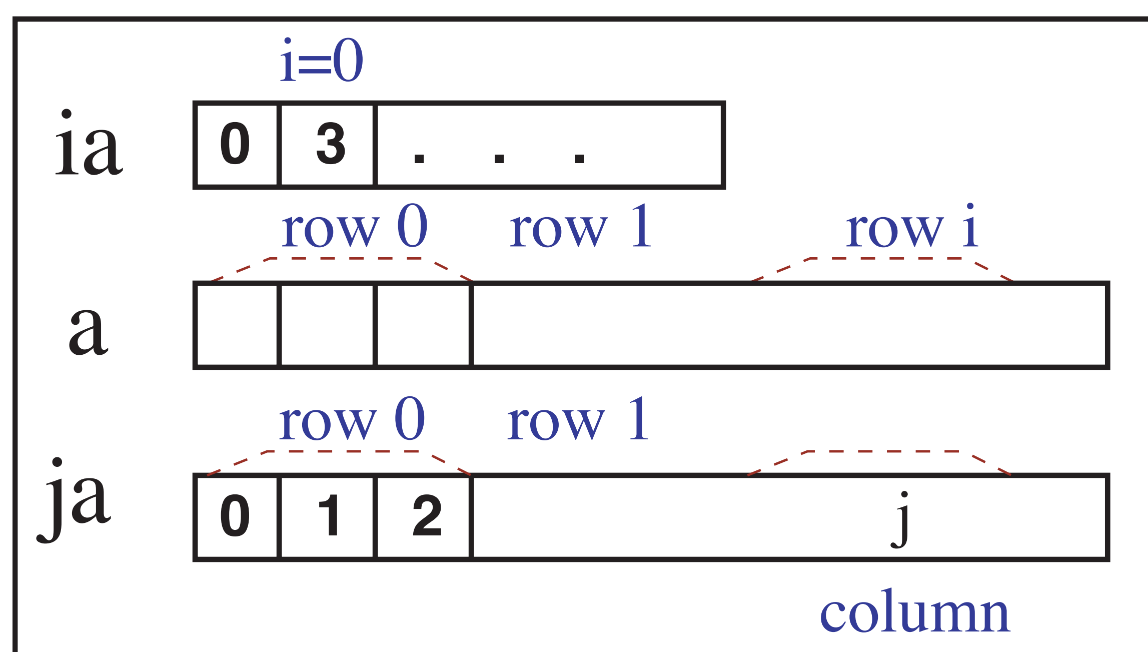
- **Problem**
  - Compile-time data reordering and computation scheduling to improve data locality and parallelism is not possible due to irregular memory references A[B[i]]
- **Solution**
  - Inspector/executor strategies perform data and computation reordering at runtime
  - Full sparse tiling is one such strategy that improves performance by exploiting parallelism, intra-iteration data reuse, and inter-iteration data reuse

# Gauss-Seidel Iteratively Solves Au = f

- u is a vector of unknowns
- A is a sparse matrix stored in the compressed sparse row format (CSR)

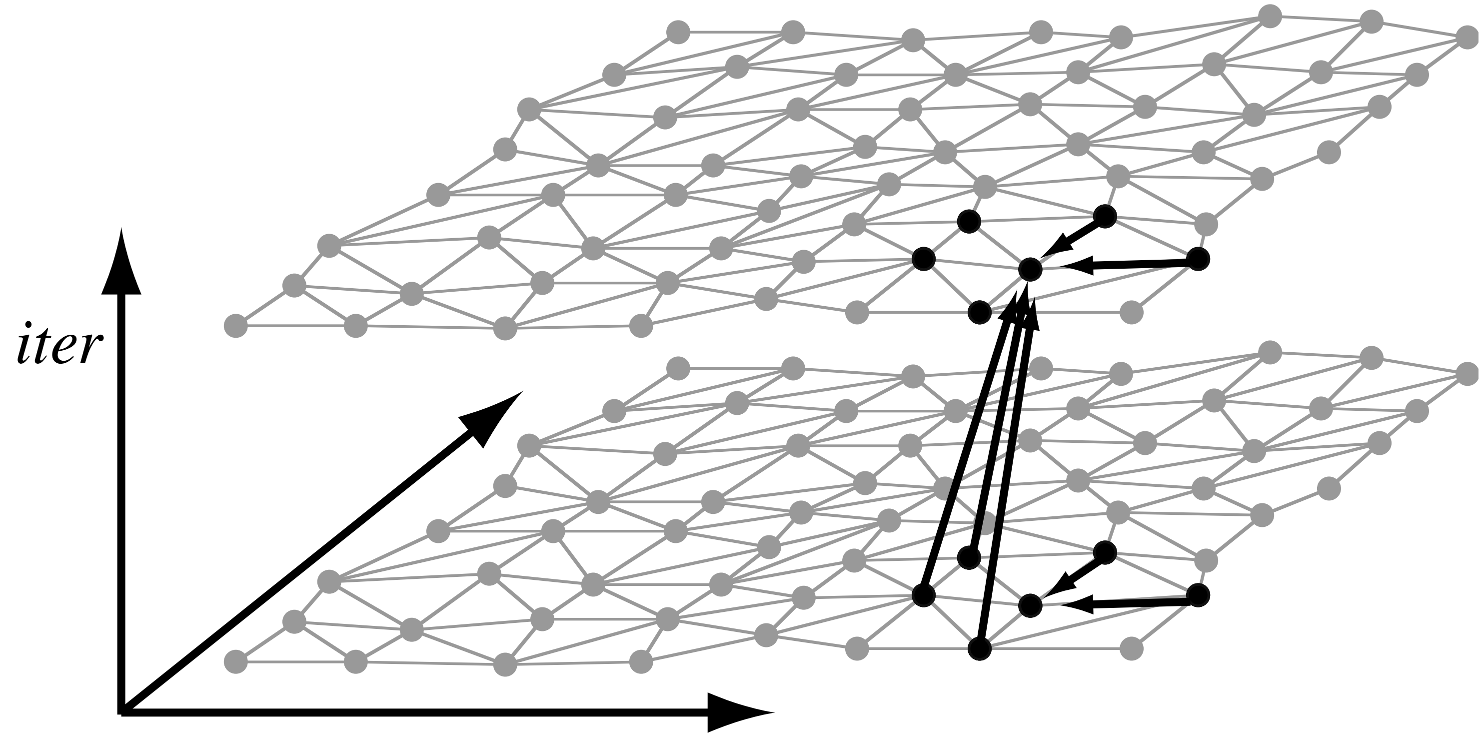



Matrix Graph

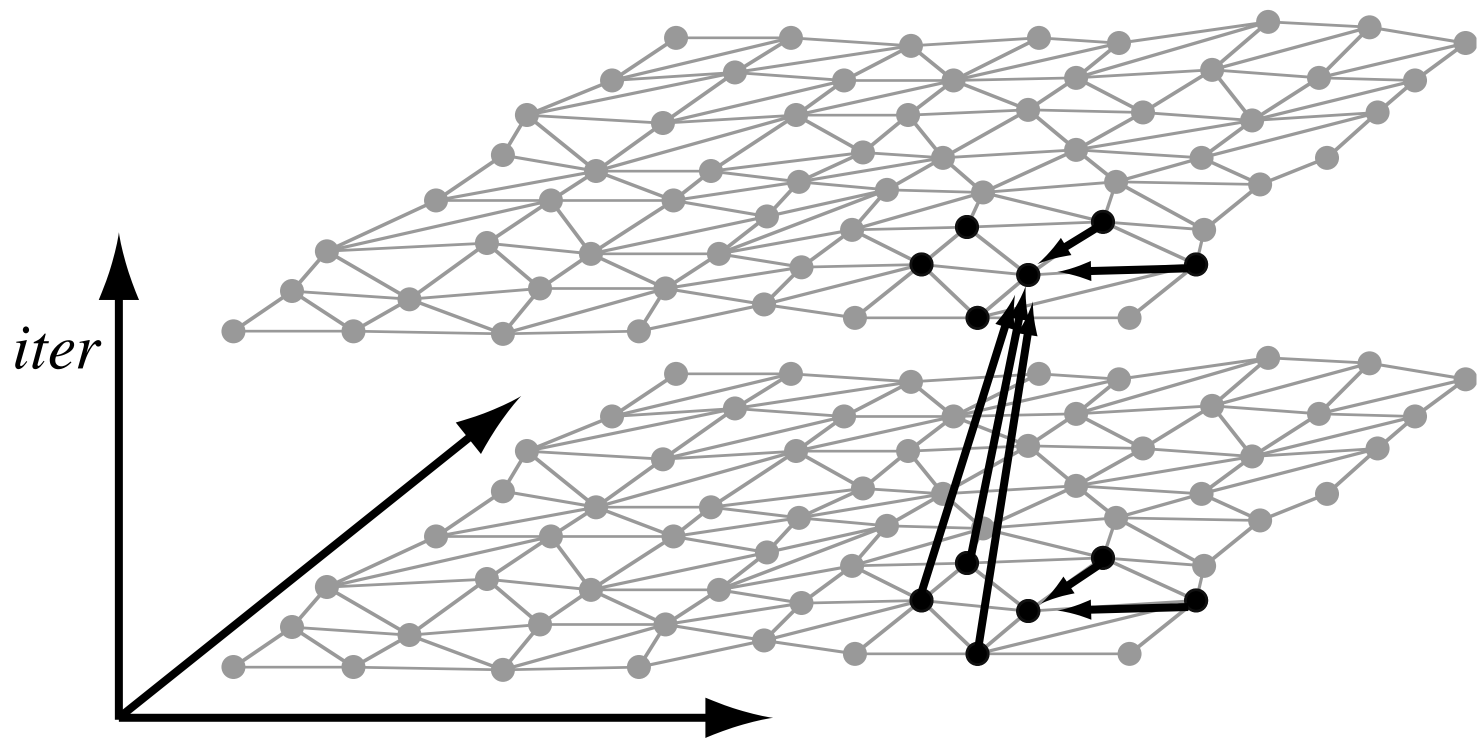


# Gauss-Seidel Iteration Space

```
do iter = 1, T
  do i = 1, R
    u(i) = f(i)
    do p = ia(i), ia(i+1) - 1
      j = ja(p)
      if (j != i)
        u(i) -= a(p) * u (ja(p))
      else
        diag = a(p)
      endif
    enddo
    u(i) = u(i) / diag
  enddo
enddo
```



# Performance Improvement Opportunities

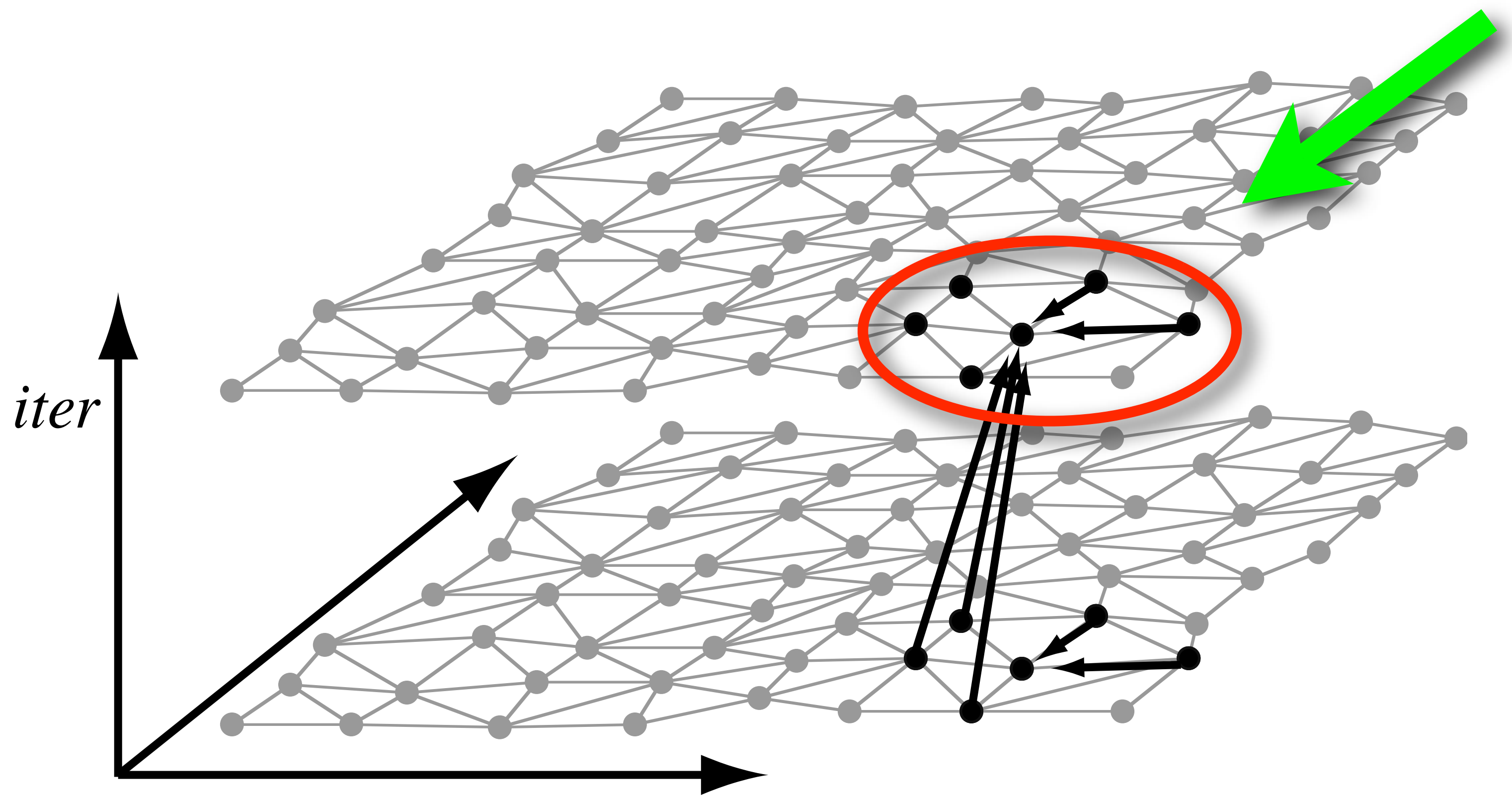



*Intra-iteration reuse*

*Inter-iteration reuse*

*Parallelism*

# Performance Improvement Opportunities

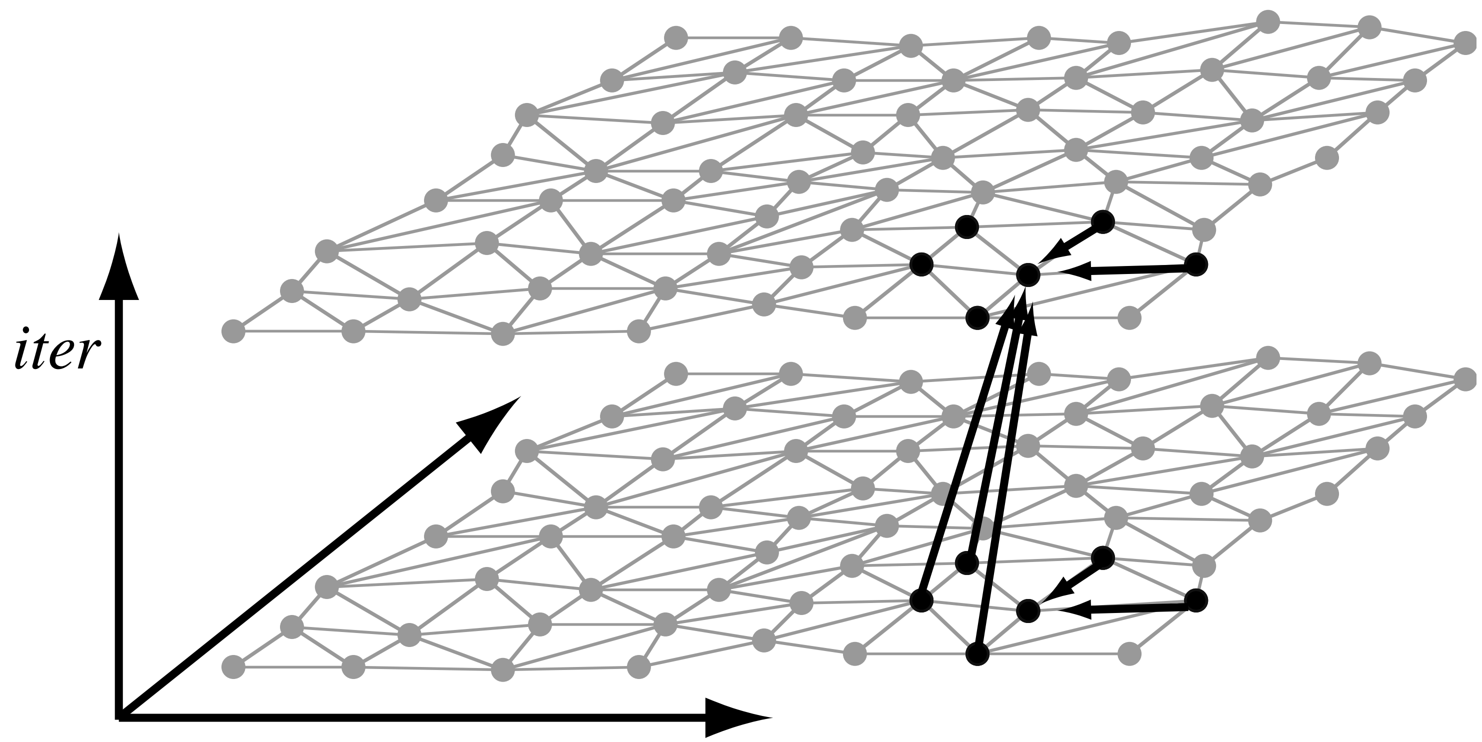



*Intra-iteration reuse*

*Inter-iteration reuse*

*Parallelism*

# Performance Improvement Opportunities

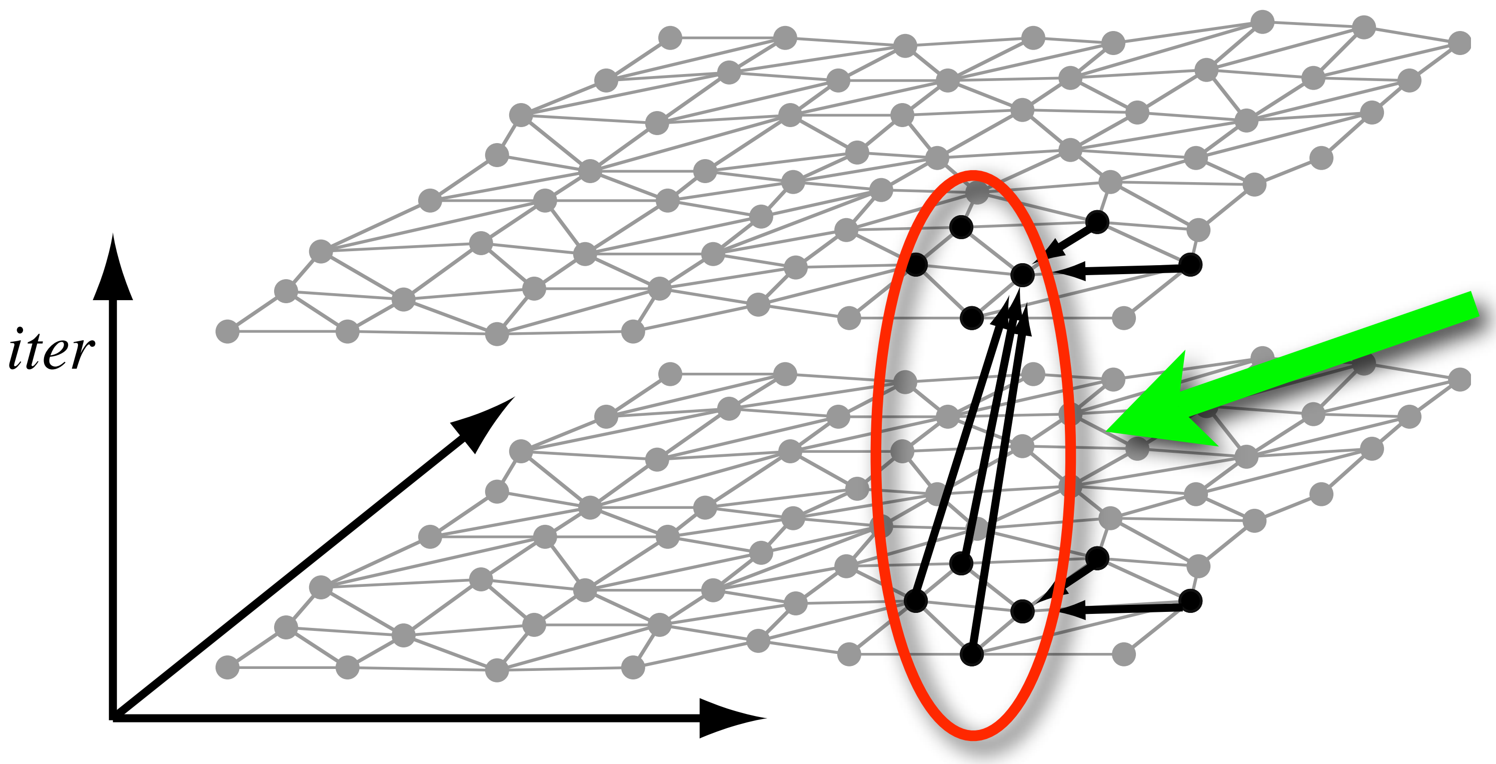



*Intra-iteration reuse*

*Inter-iteration reuse*

*Parallelism*

# Performance Improvement Opportunities

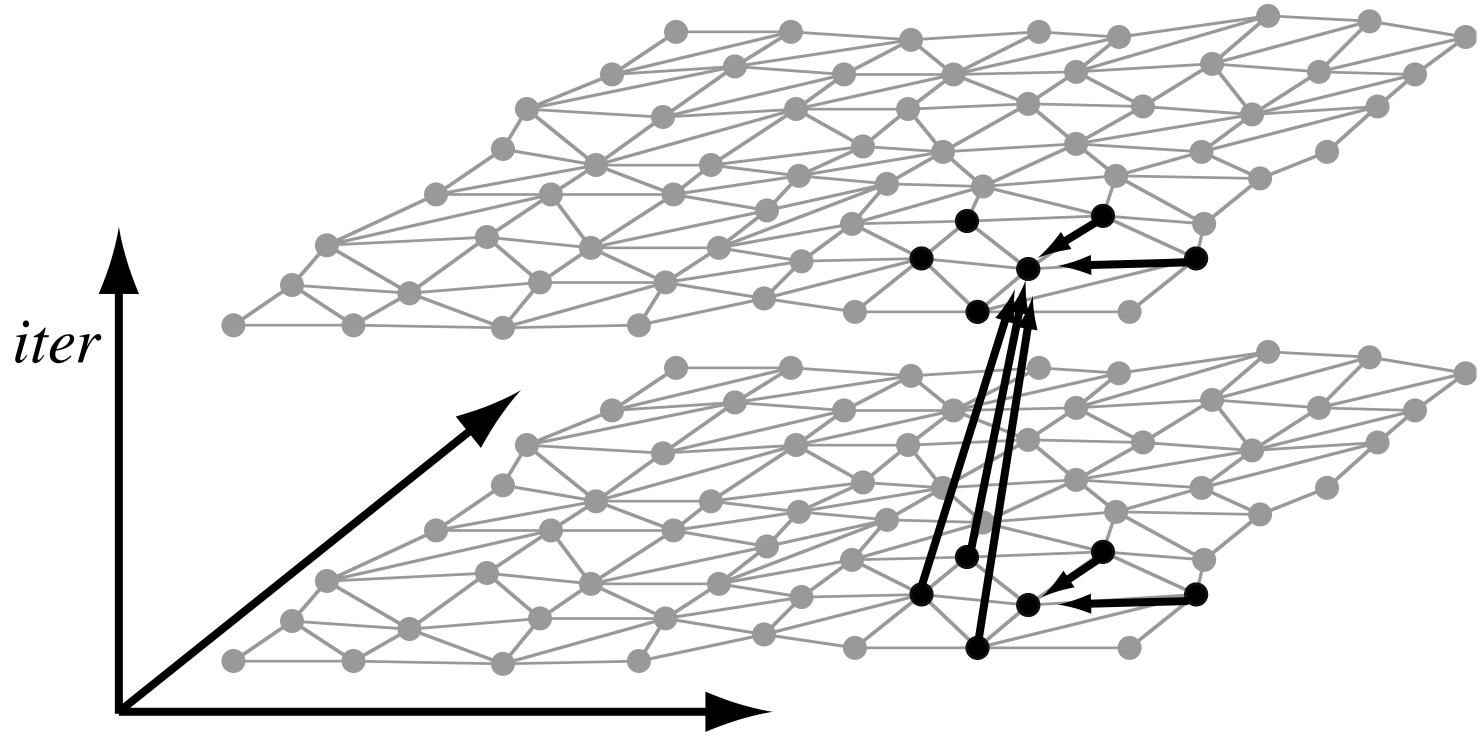



*Intra-iteration reuse*

*Inter-iteration reuse*

*Parallelism*

# Performance Improvement Opportunities

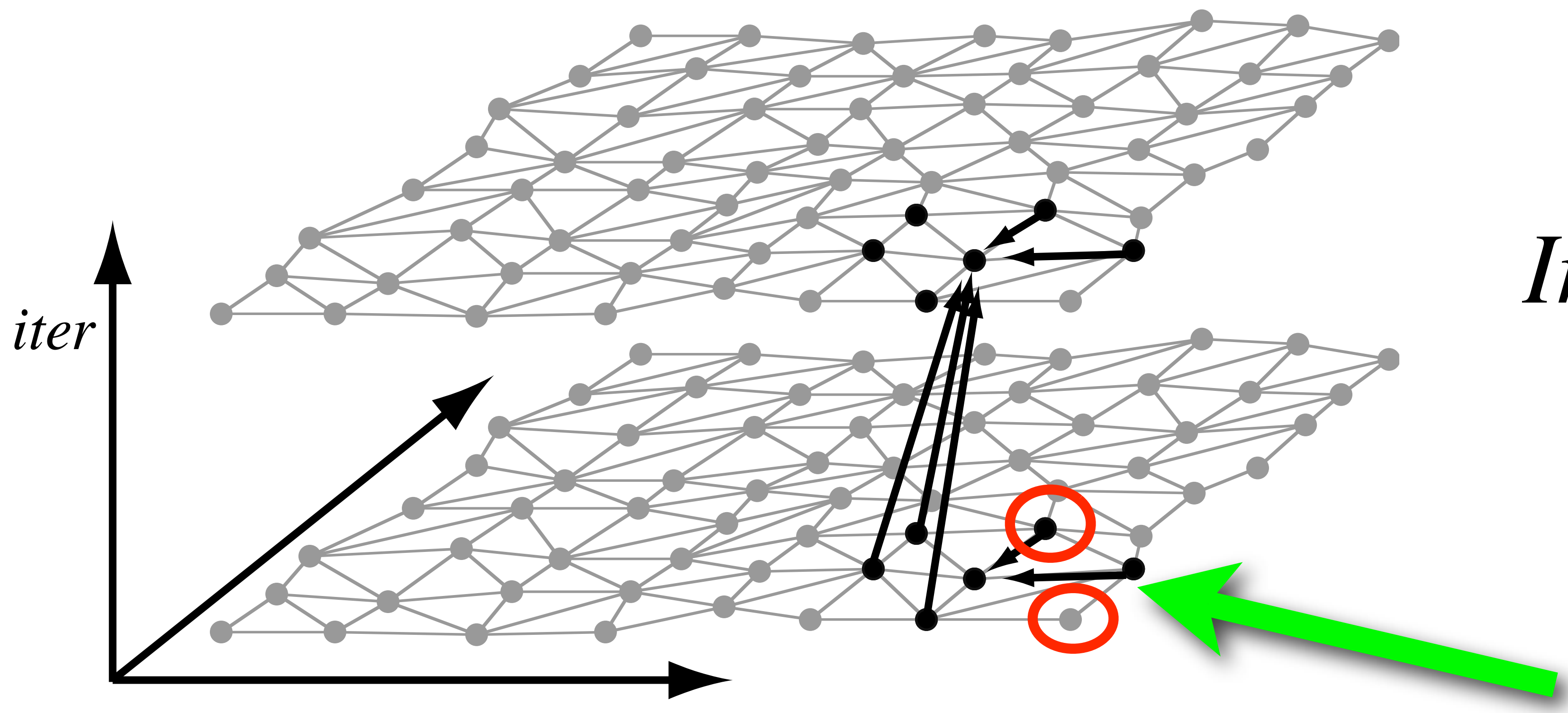



*Intra-iteration reuse*

*Inter-iteration reuse*

*Parallelism*

# Performance Improvement Opportunities



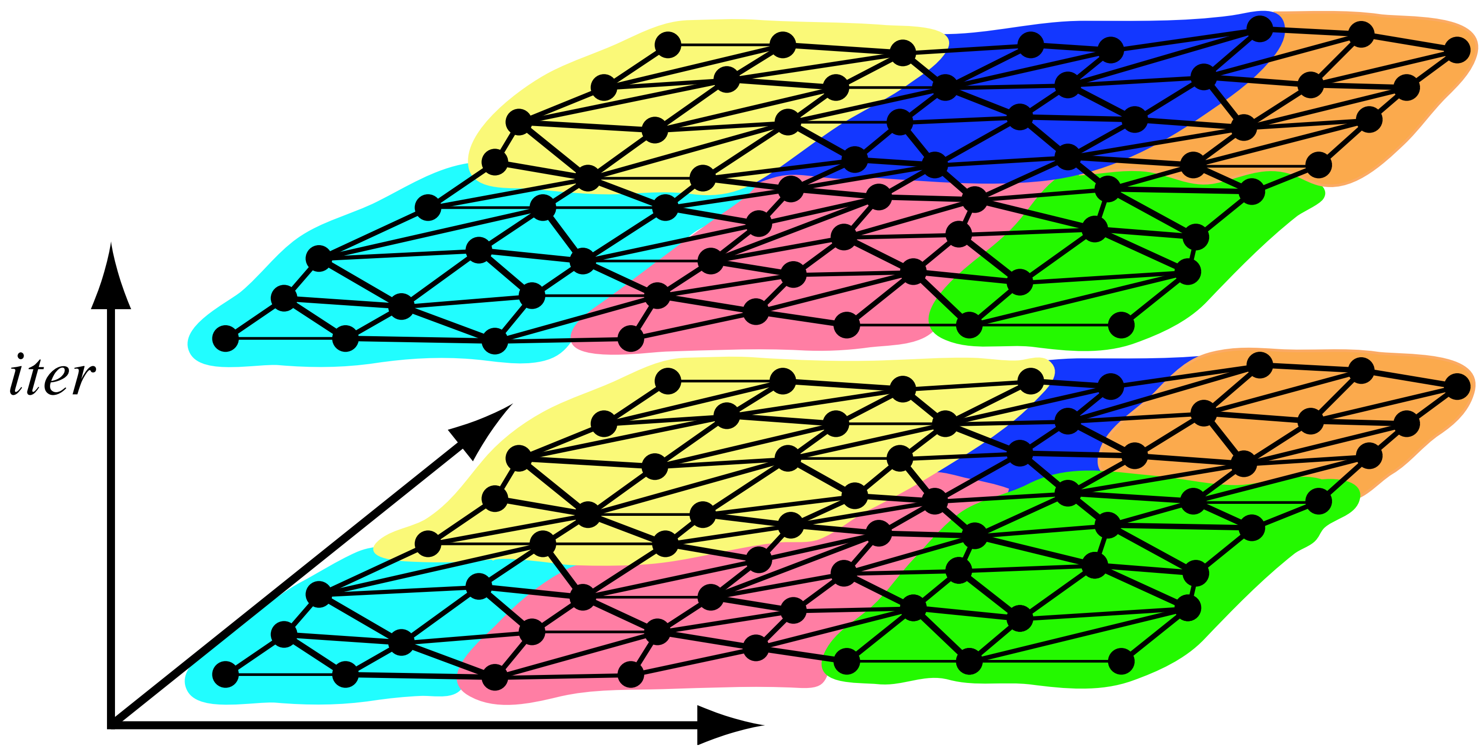

*Intra-iteration reuse*

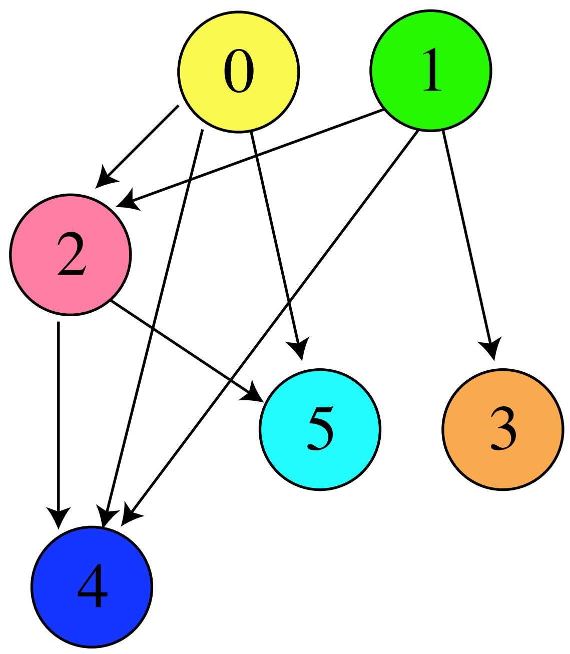
*Inter-iteration reuse*

*Parallelism*

Colorado State University

# Punchline: Full Sparse Tiling




- Breaks up computation into pieces
- Each piece has intra and inter-iteration data locality
- Some pieces can be executed in parallel

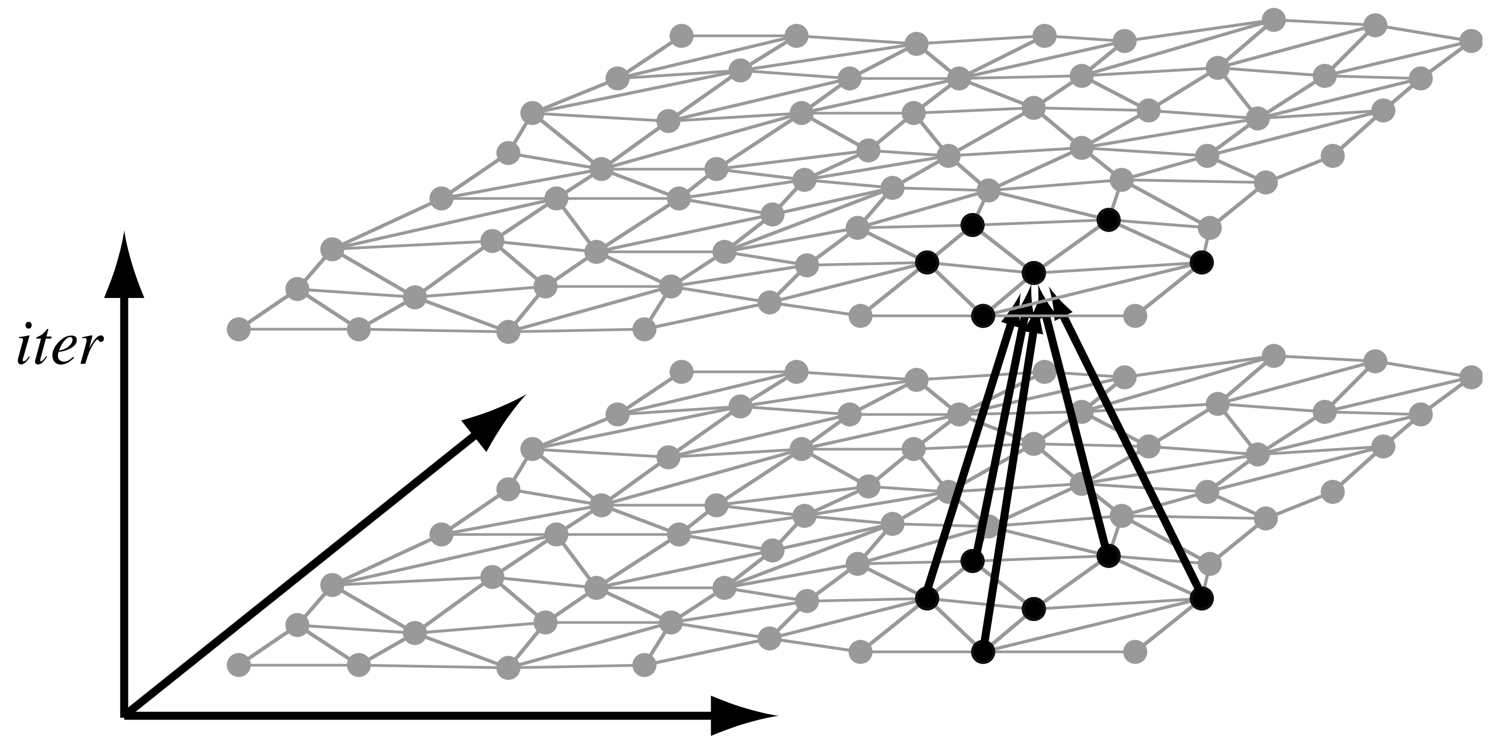
# Talk Outline

- Overview
- Parallelizing Jacobi with the owner computes method xor full sparse tiling
- Increasing the parallelism in full sparse tile schedules
- Extra work needed to handle Gauss-Seidel
- Experimental Results

Colorado State University

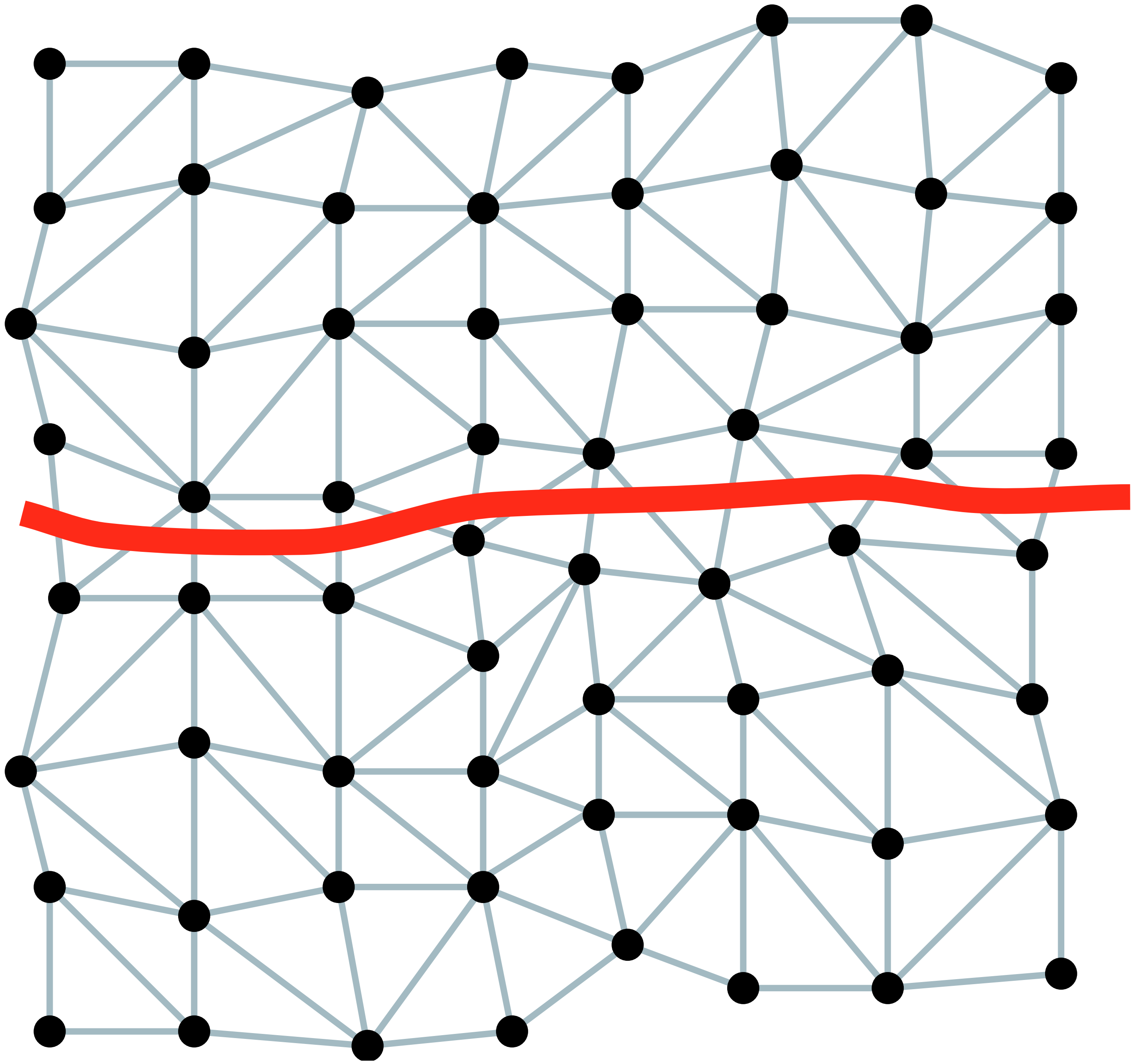
# Jacobi: similar yet simpler

```
do iter = 1, T
   do i = 1, R
      u(i) = f(i)
      do p = ia(i), ia(i+ 1) - 1
         j = ja(p)
         if (j != i) then
            u(i) -= a(p)*tmp (j)
         else
            diag = a(p)
         endif
      enddo
      u(i) = u(i)/diag
   enddo
   do i = 1, R
      tmp (i) = u(i)
   enddo
enddo
```




- data dependences not known until runtime
- no intra-iteration dependences
- i loop is a reduction, therefore parallelizable

# Parallelize with Owner Computes Method

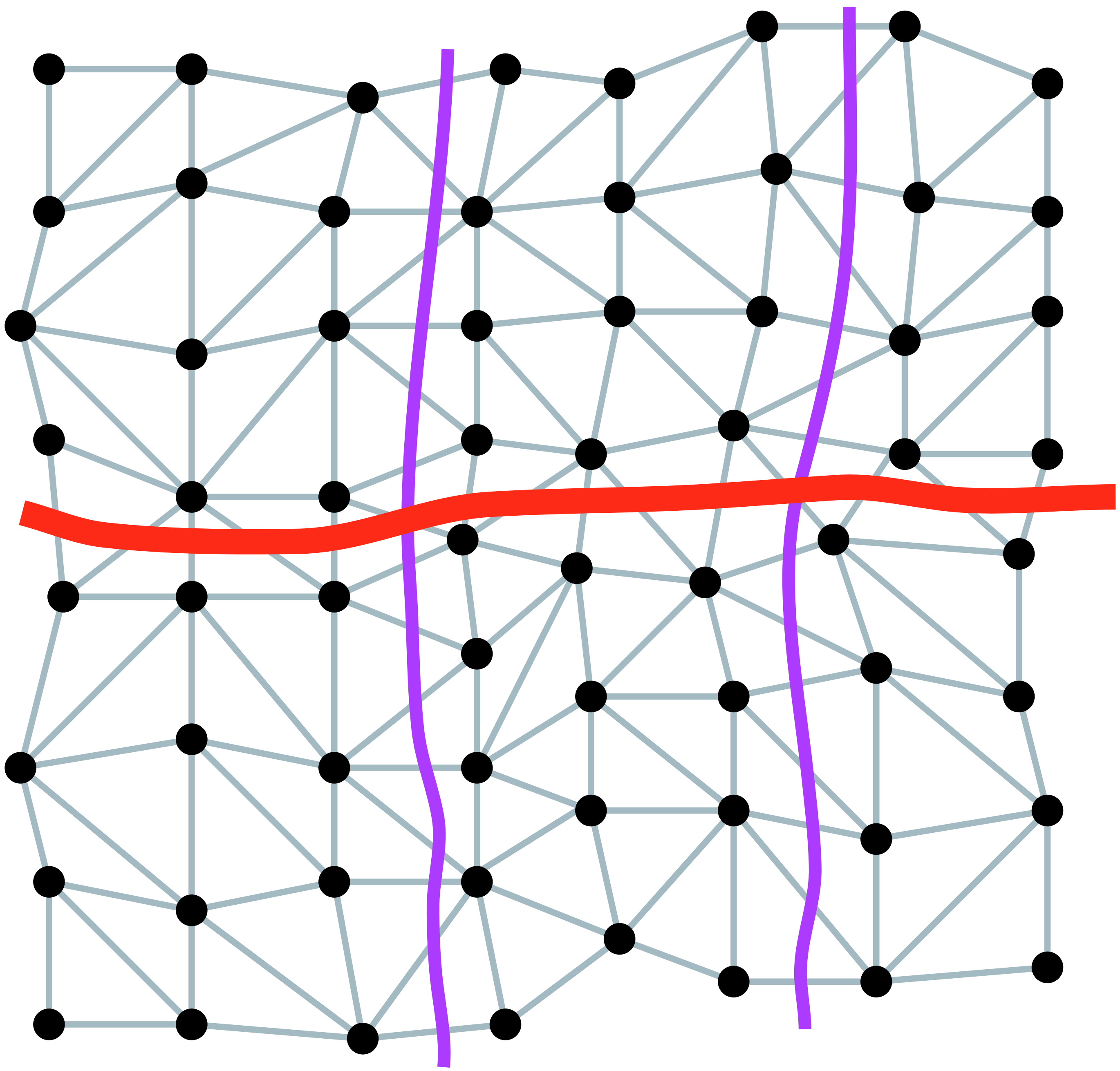


At each convergence iteration ...

- each partition receives data from previous convergence iteration
- each partition executes in parallel
- each partition sends data

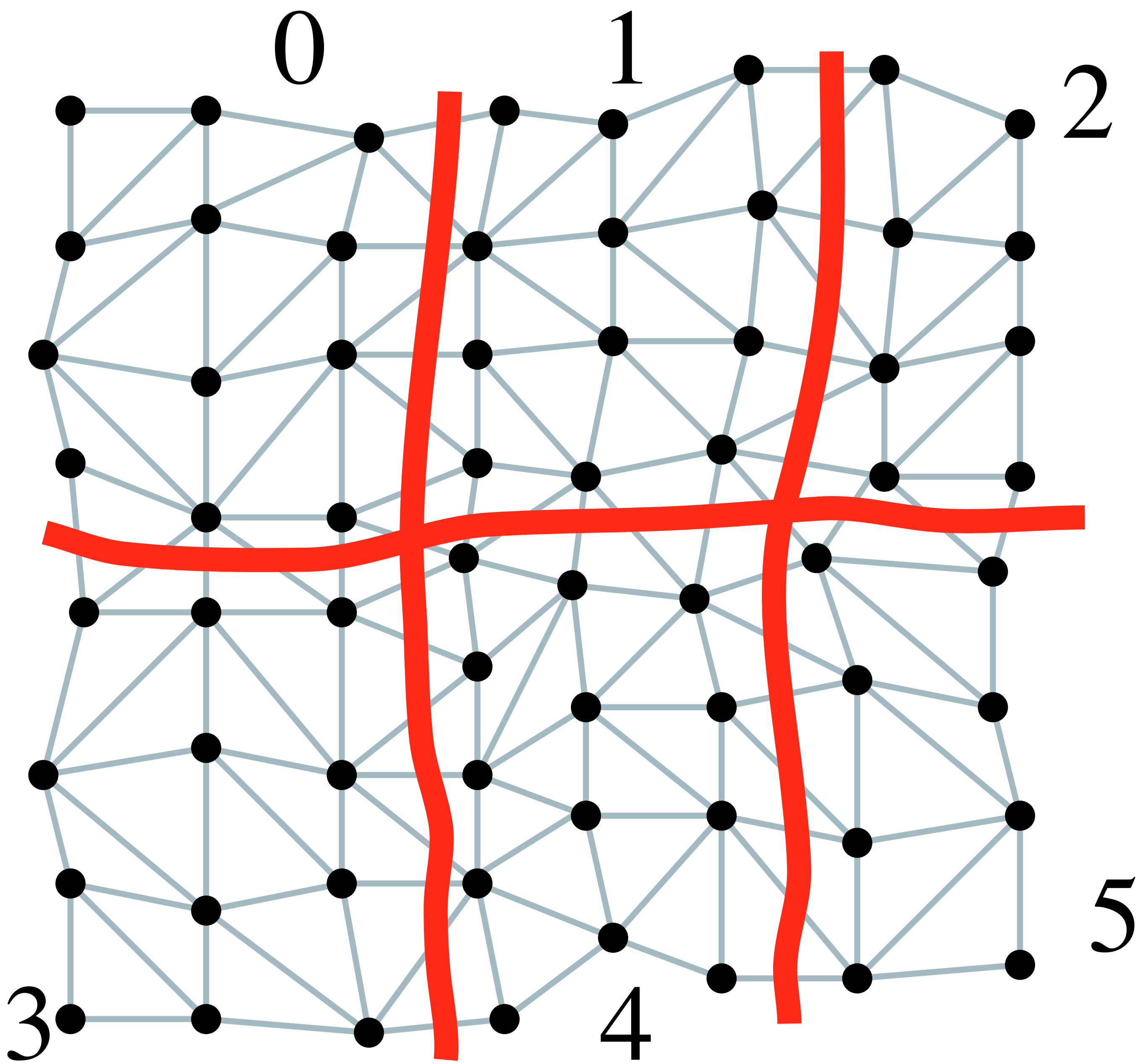
Colorado State University

# Locality in Owner Computes Method



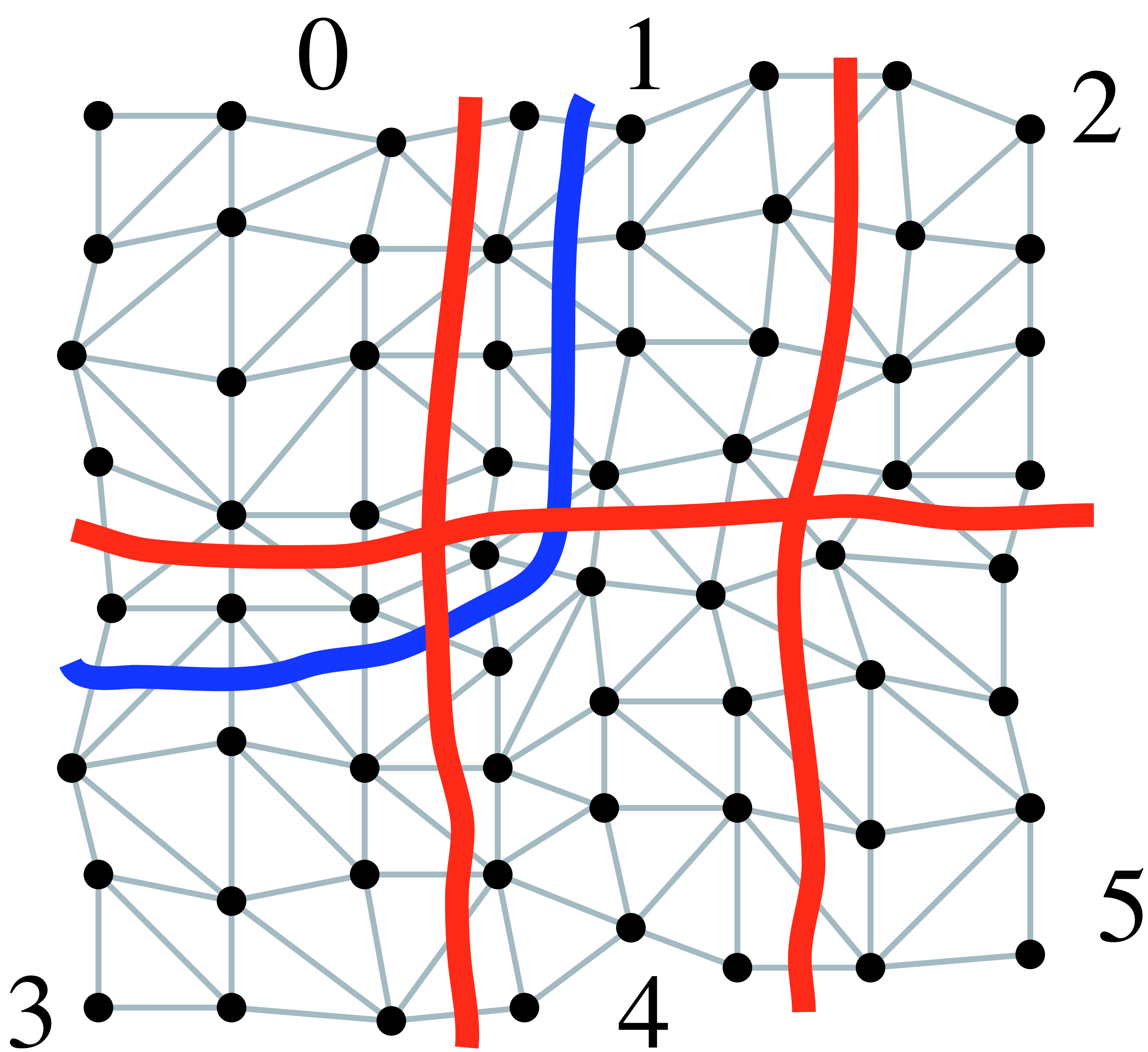
- Intra-iteration locality
  - schedule by sub-part
  - reorder data for consecutive order in sub-parts
- NO Inter-iteration locality, main partitions don't fit in cache

# Parallelize with Full Sparse Tiling



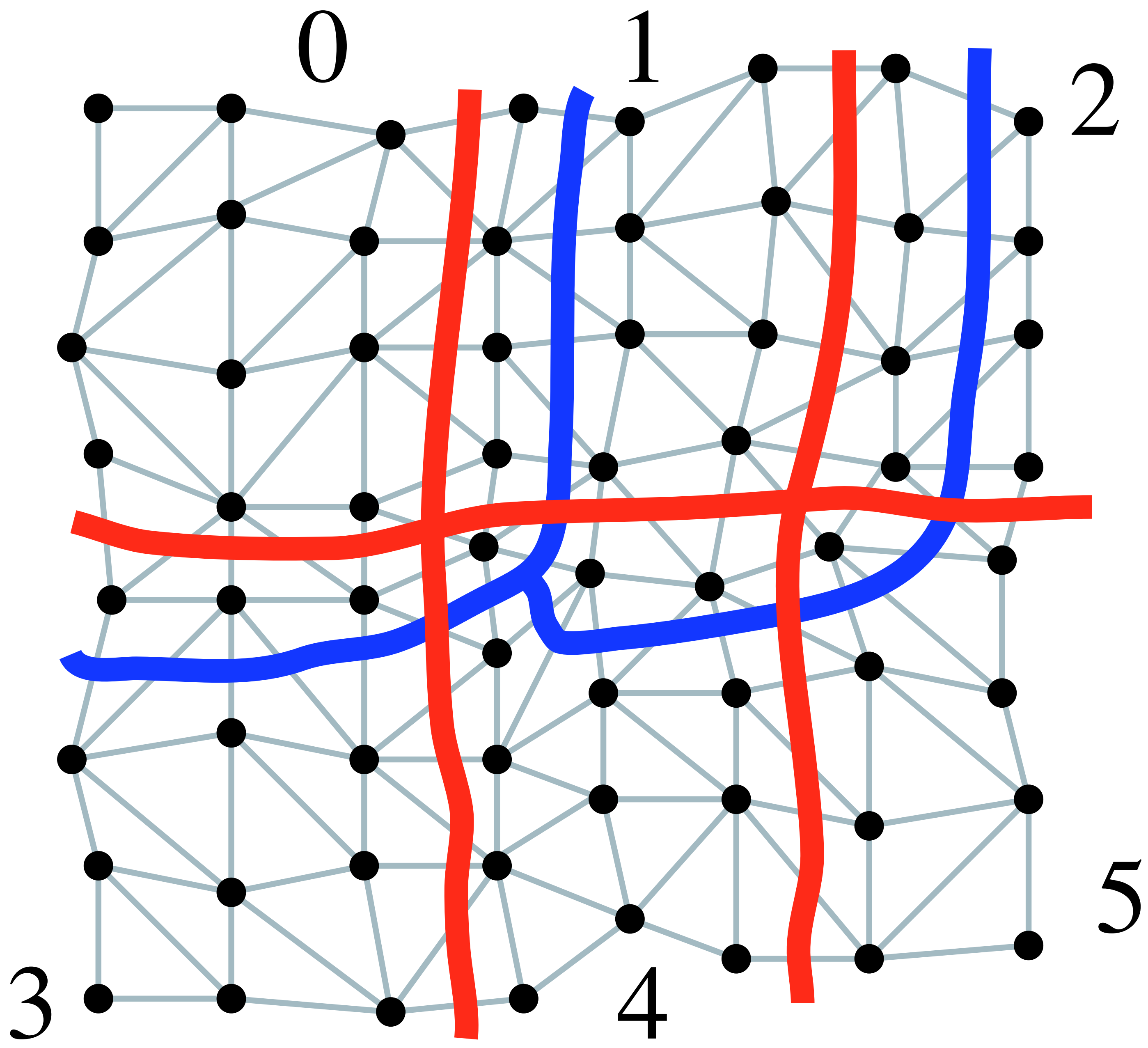

- Create seed partitioning at *iter* = 2

# Parallelize with Full Sparse Tiling



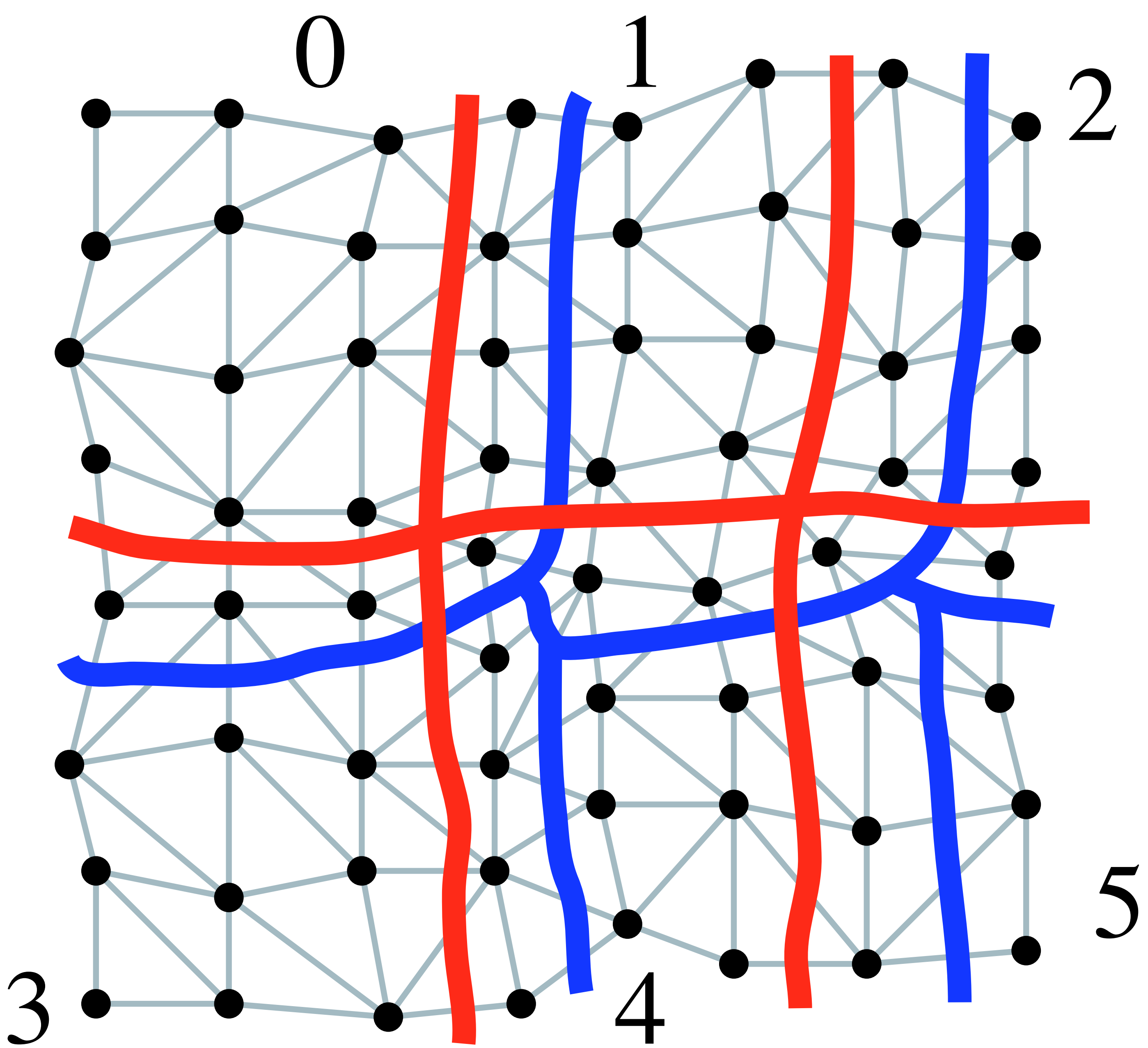

- Create seed partitioning at *iter* = 2
- Grow tiles to *iter* = 1 based on ordering of partitions

# Parallelize with Full Sparse Tiling



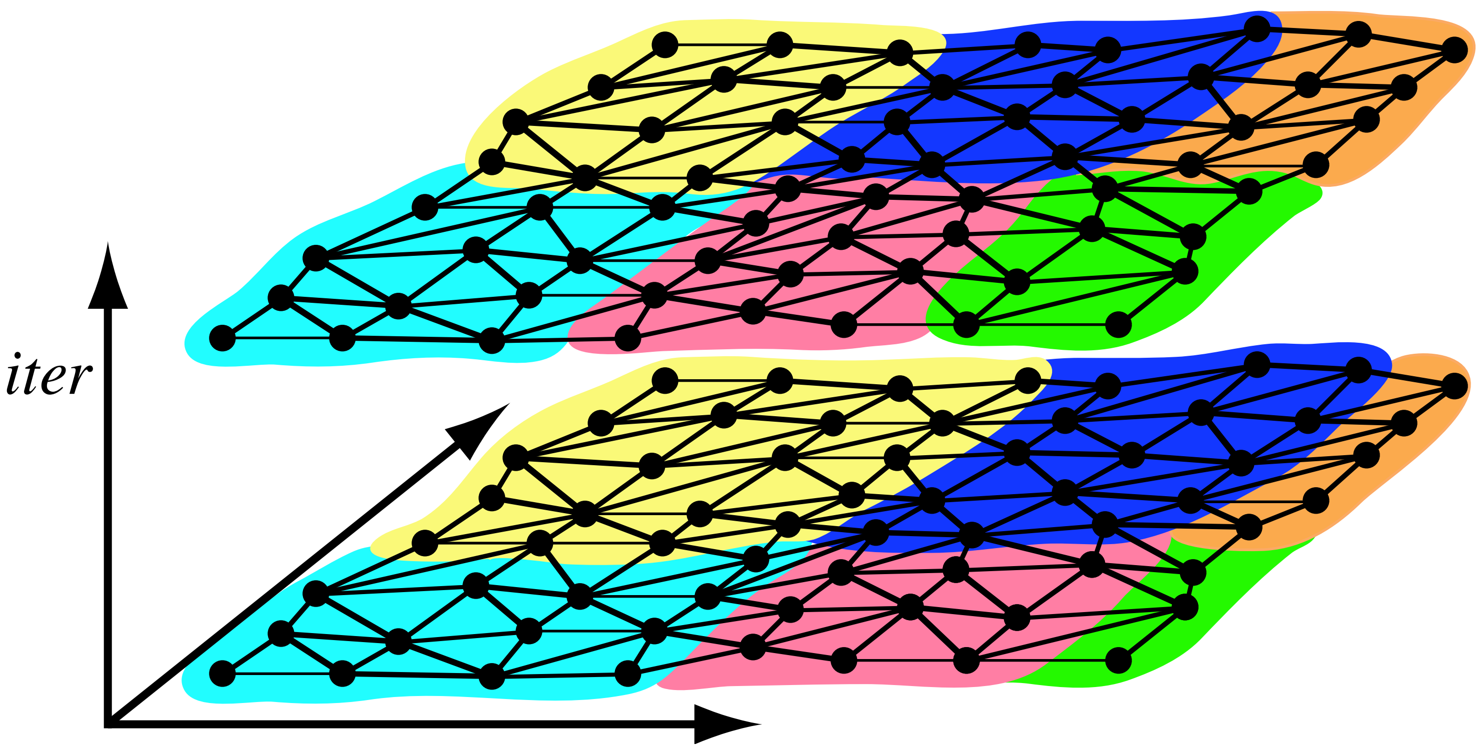

- Create seed partitioning at *iter* = 2
- Grow tiles to *iter* = 1 based on ordering of partitions

# Parallelize with Full Sparse Tiling




- Create seed partitioning at *iter* = 2
- Grow tiles to *iter* = 1 based on ordering of partitions

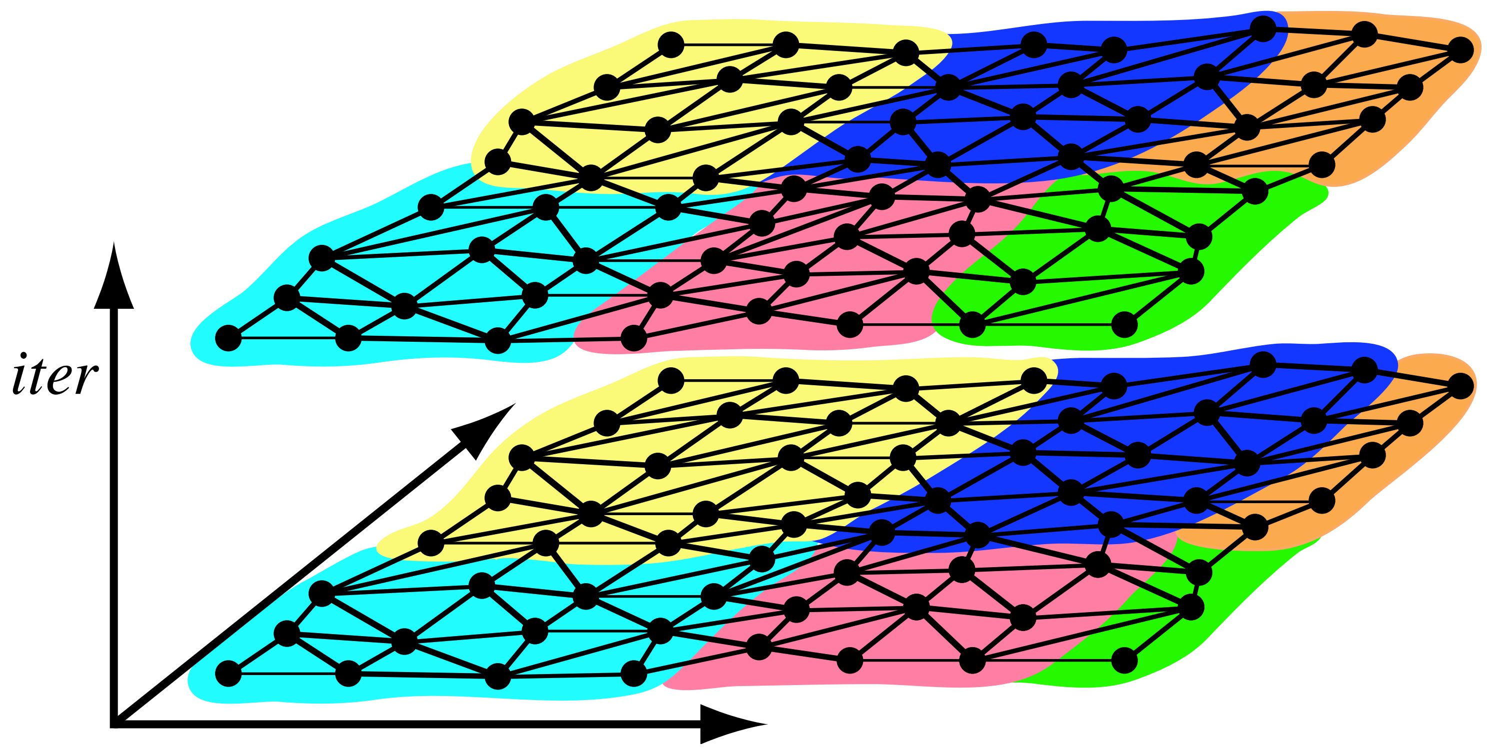
# Locality in Full Sparse Tiled Jacobi

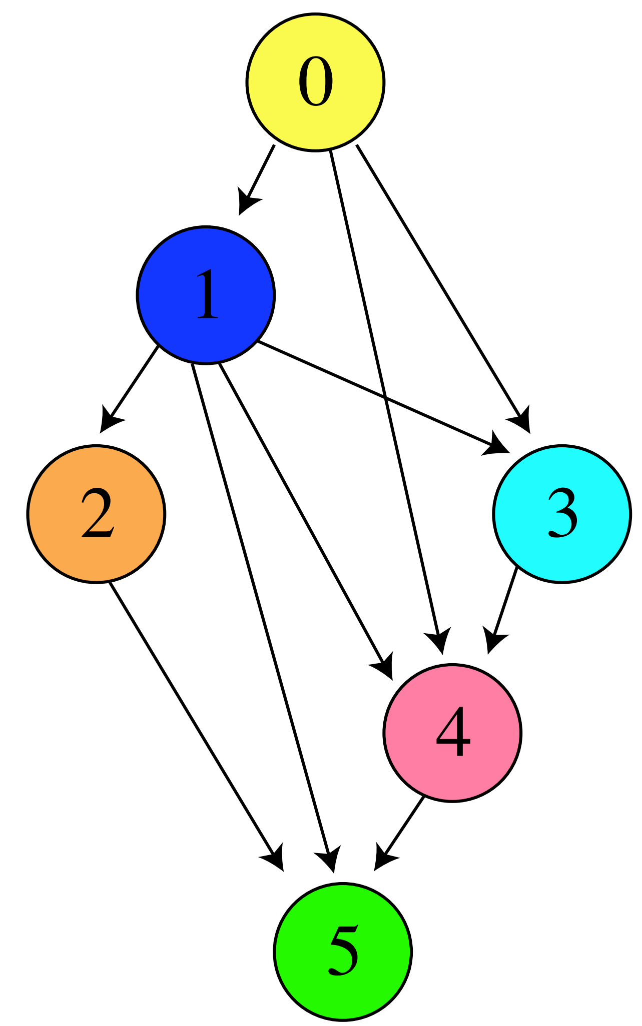



- **Intra-iteration** locality achieved by consecutively ordering matrix graph nodes within a seed partition
- **Inter-iteration** locality achieved with tile-by-tile execution

Colorado State University

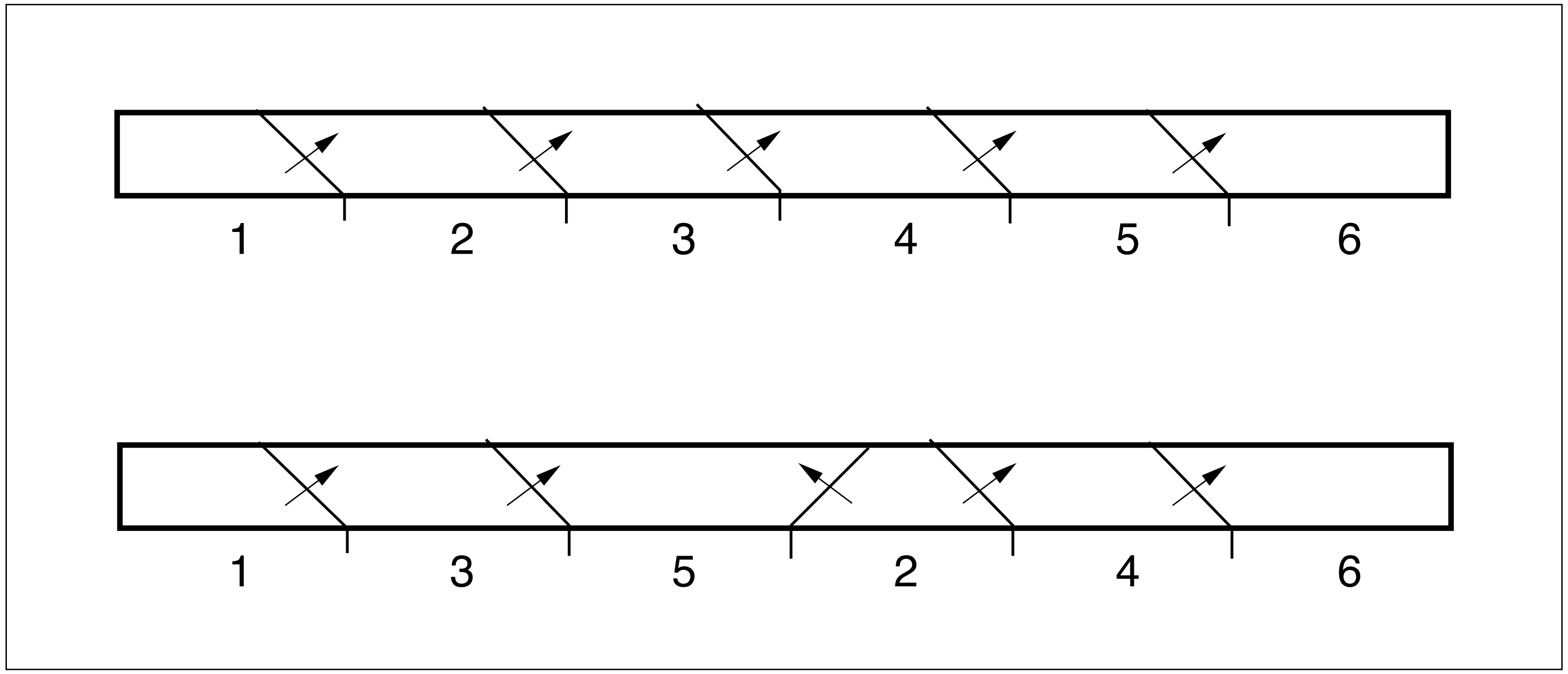
# Parallelism in Full Sparse Tiled Jacobi



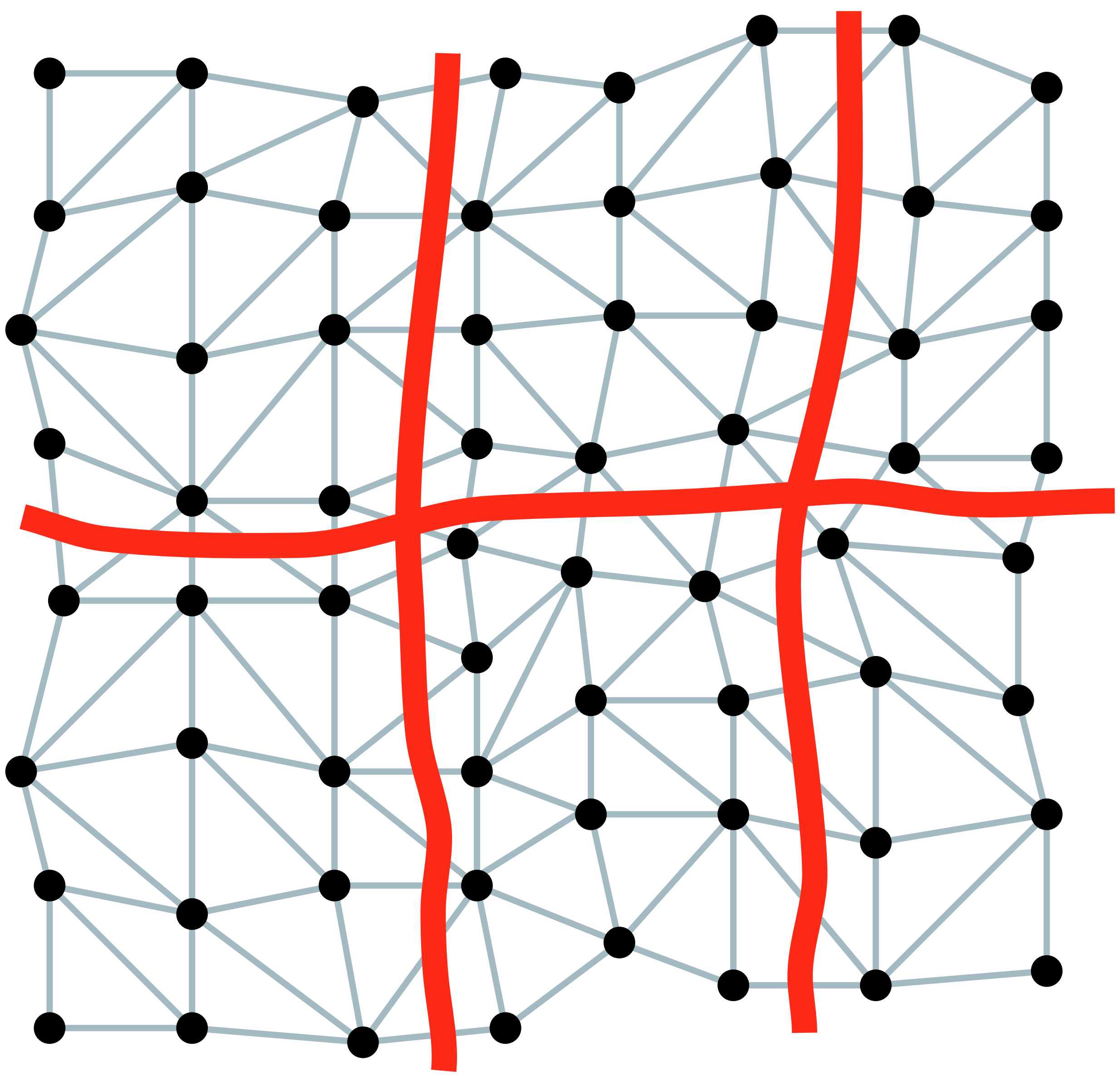

Average parallelism = (# tiles) / (# tiles in critical path)
= 6 / 5 = 1.2

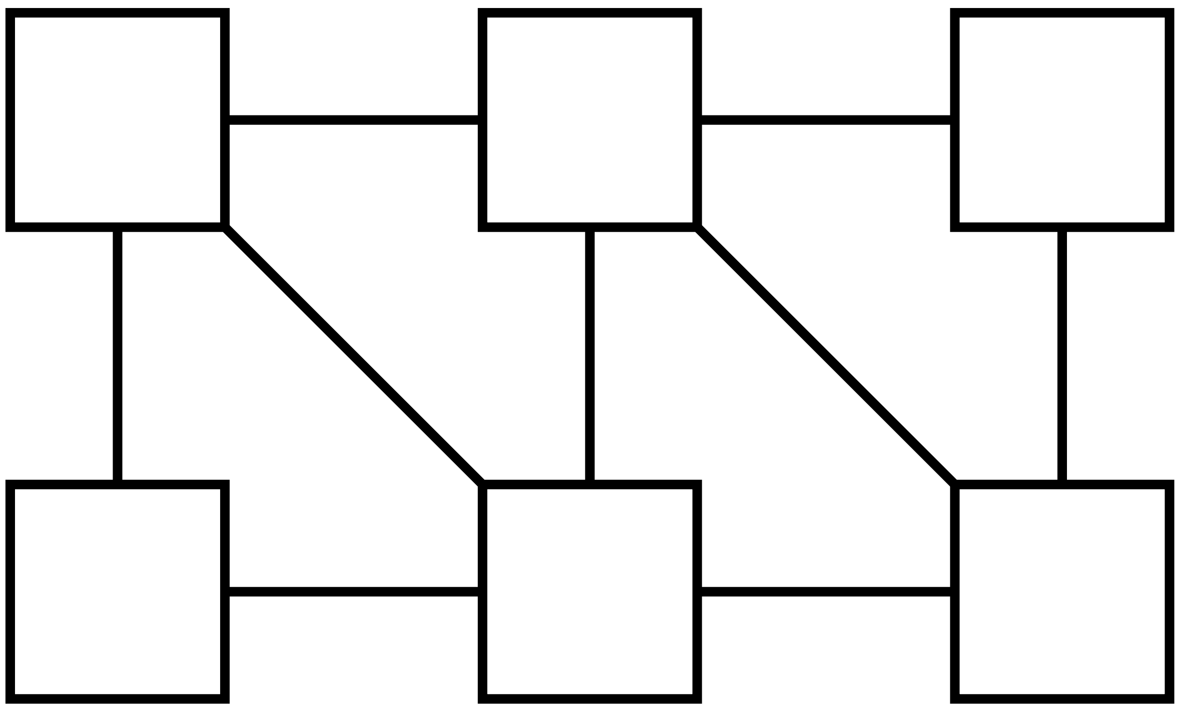
# How can we increase parallelism between tiles?




- Order that tile growth is performed matters
- Best is to first grow tiles whose seed partitions are not adjacent

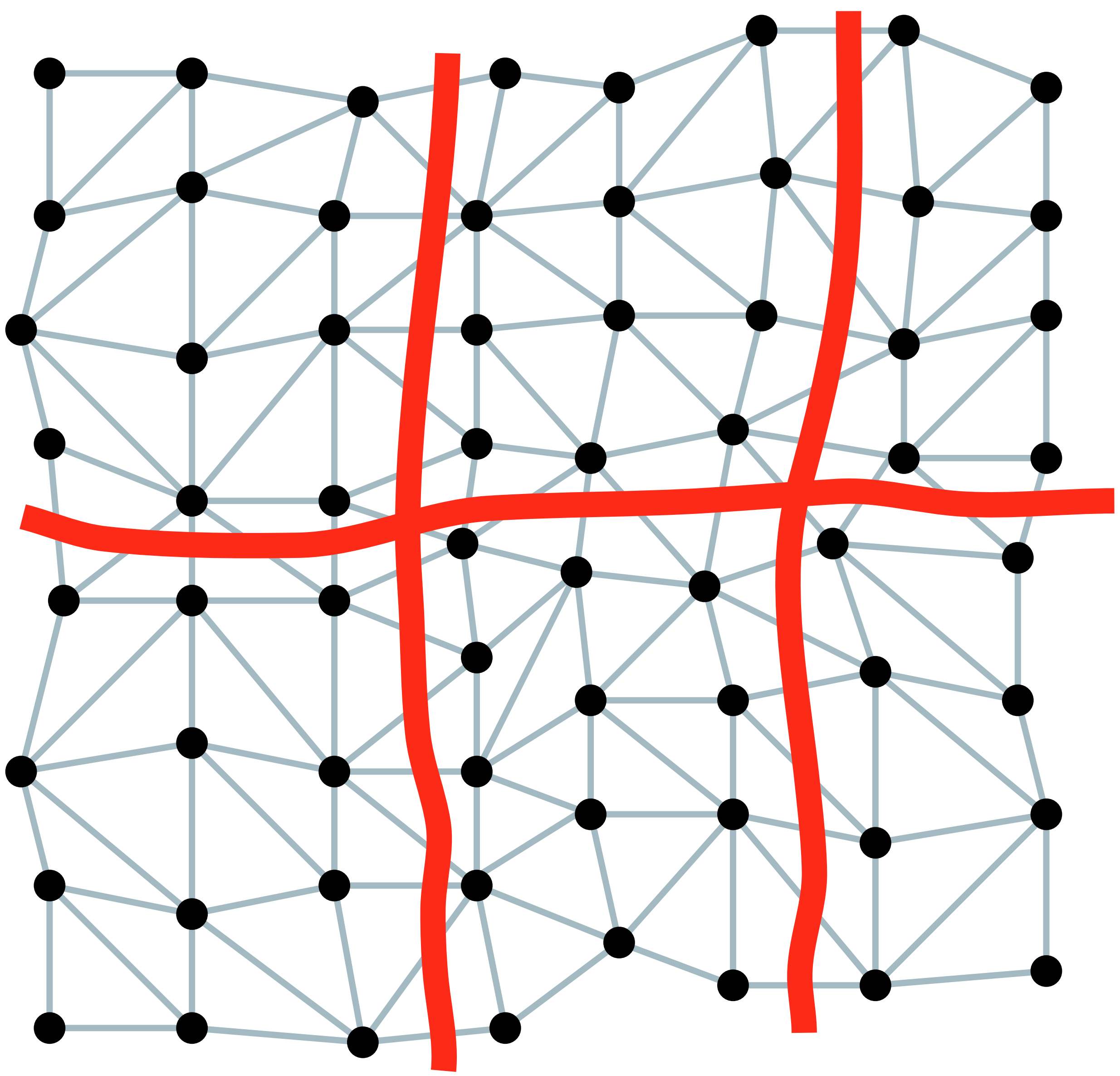
# Improving Average Parallelism Using Coloring

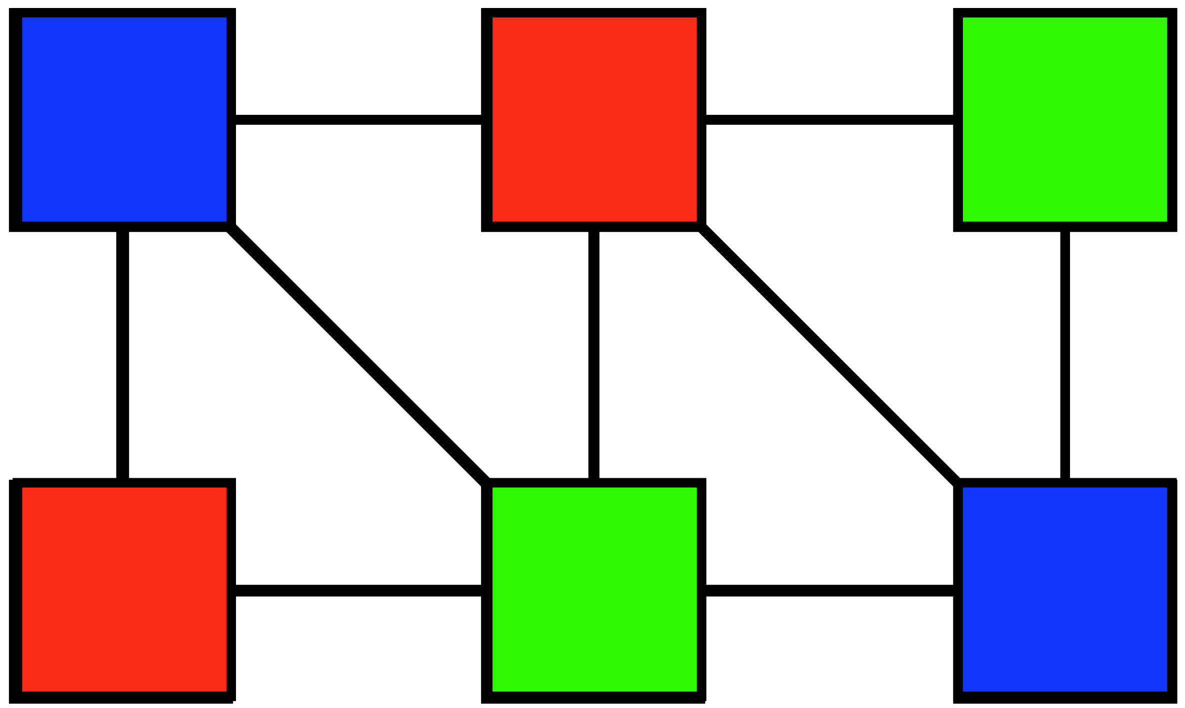




- Create a partition graph

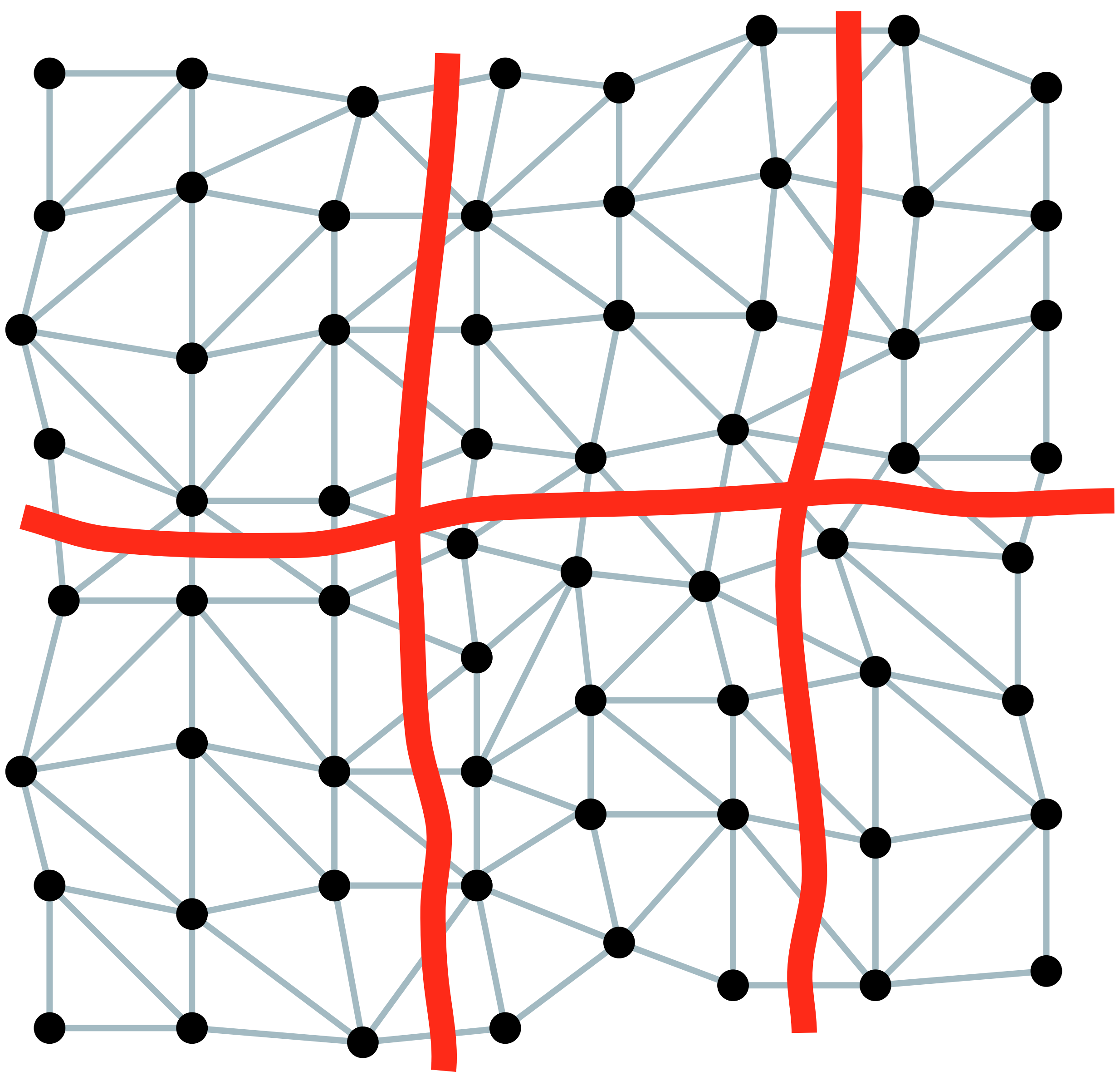
# Improving Average Parallelism Using Coloring

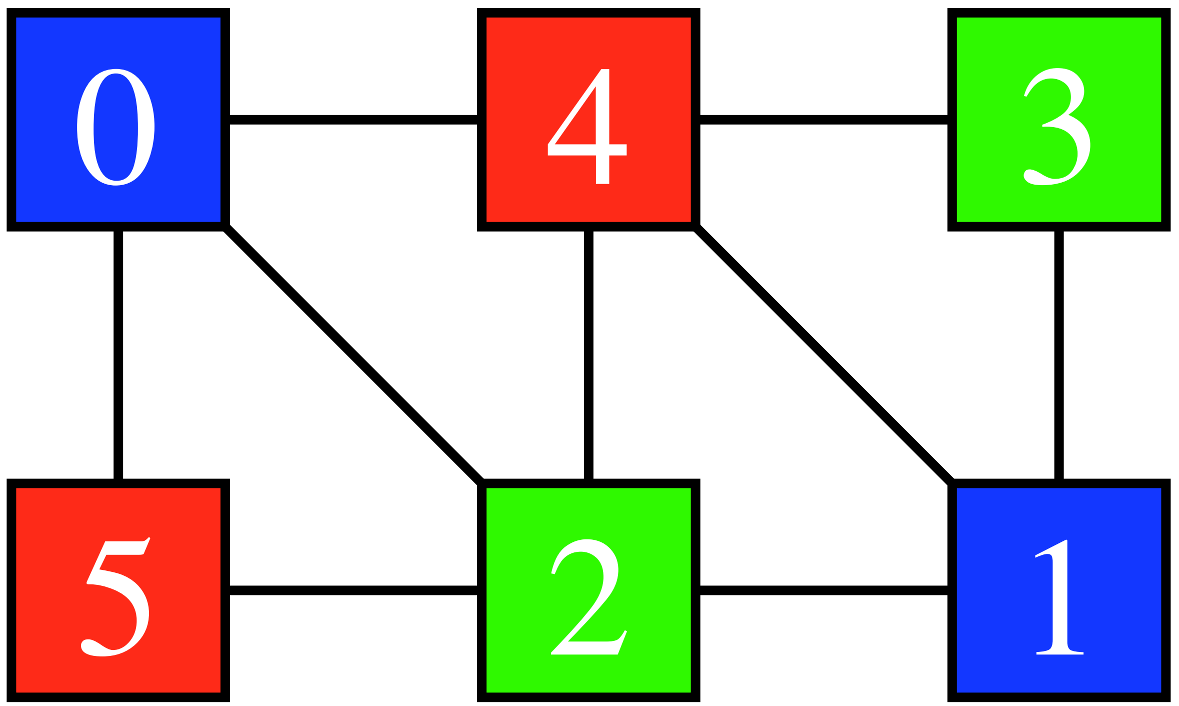




- Create a partition graph
- Color the partition graph

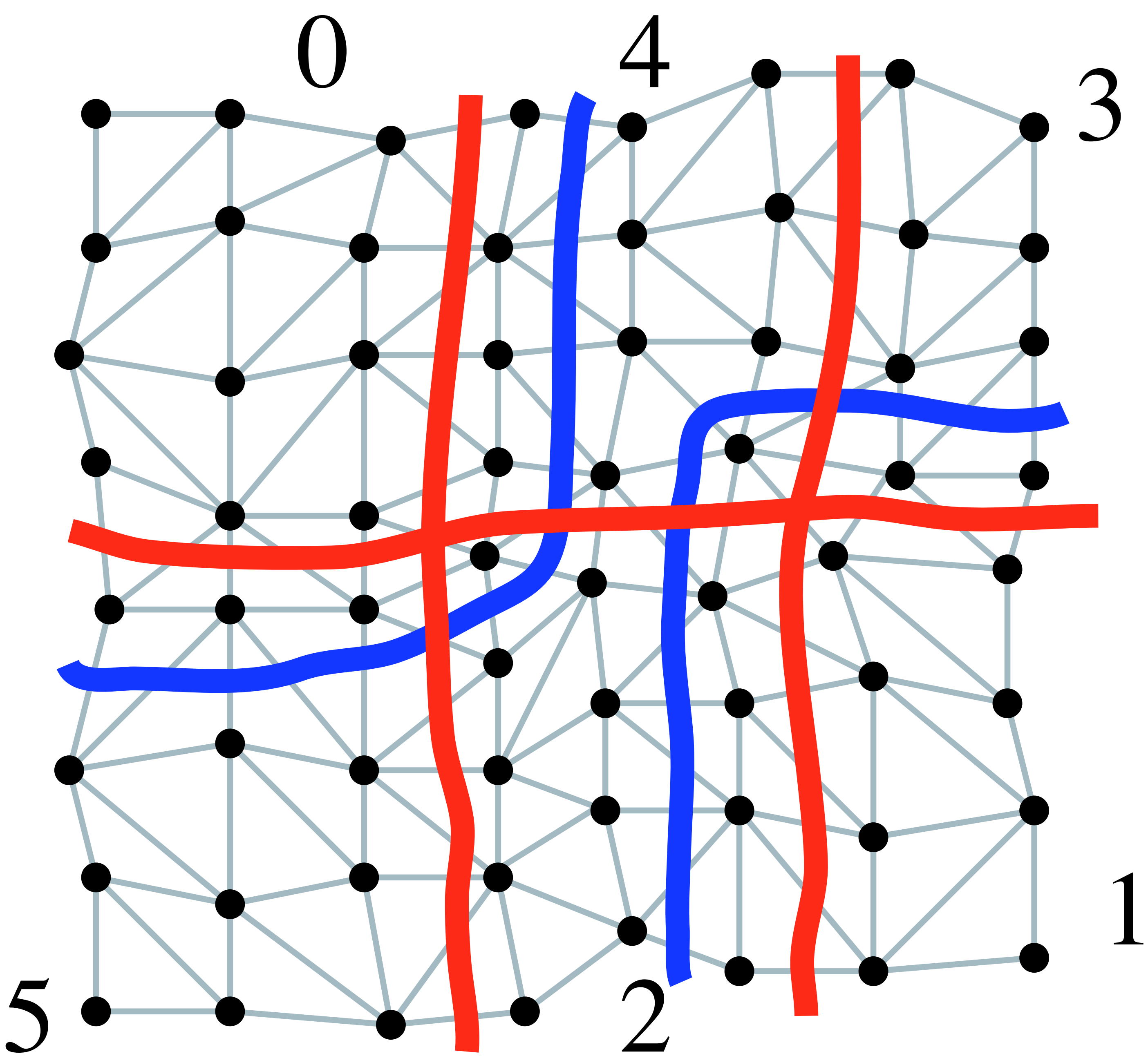
# Improving Average Parallelism Using Coloring






- Create a partition graph
- Color the partition graph
- Renumber partitions consecutively by color

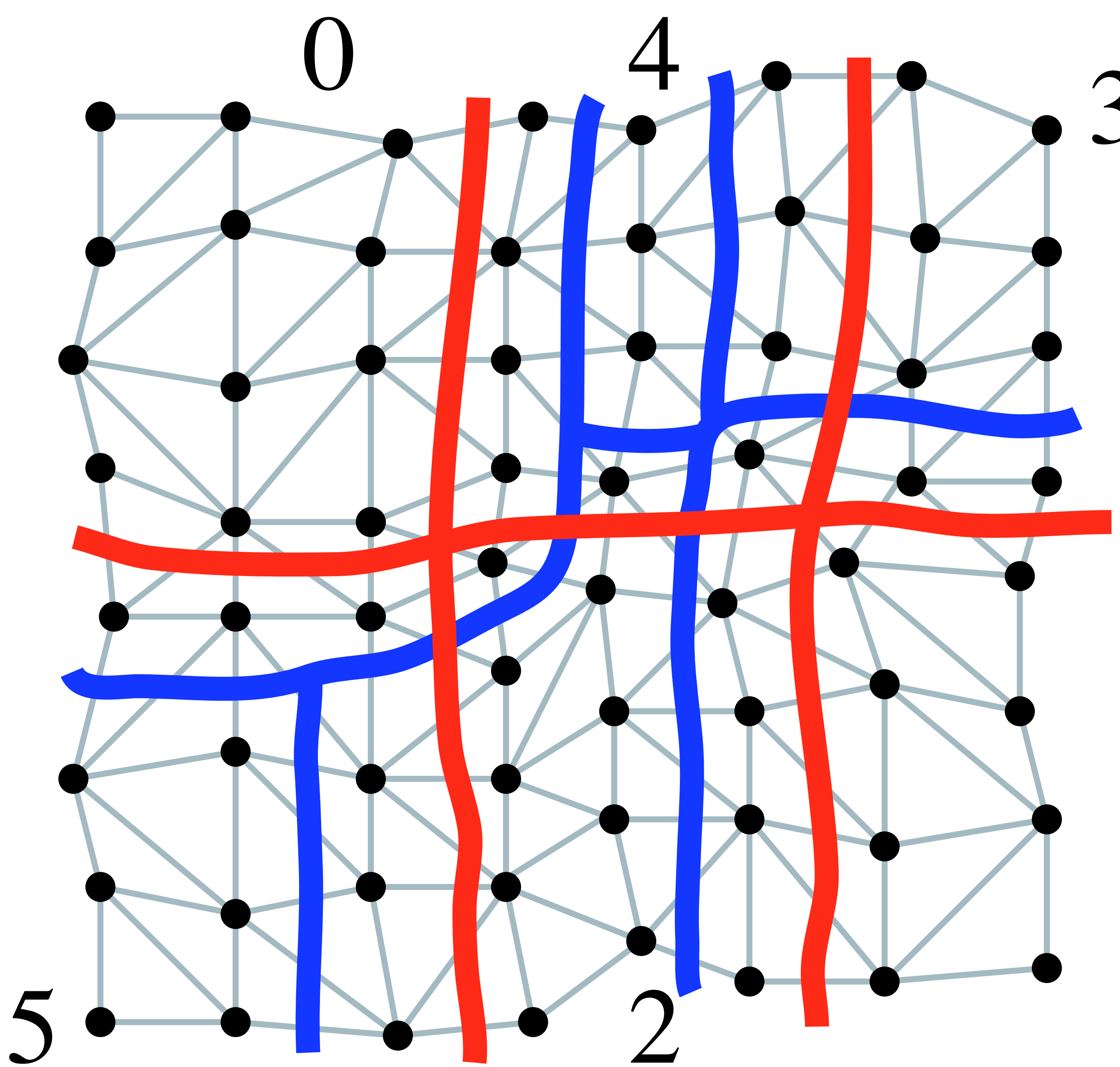
# Grow Using New Partition Order




- Renumber the seed partition cells based on coloring
- Grow tiles using new ordering
- Notice that tiles 0 and 1 may be executed in parallel

Colorado State University

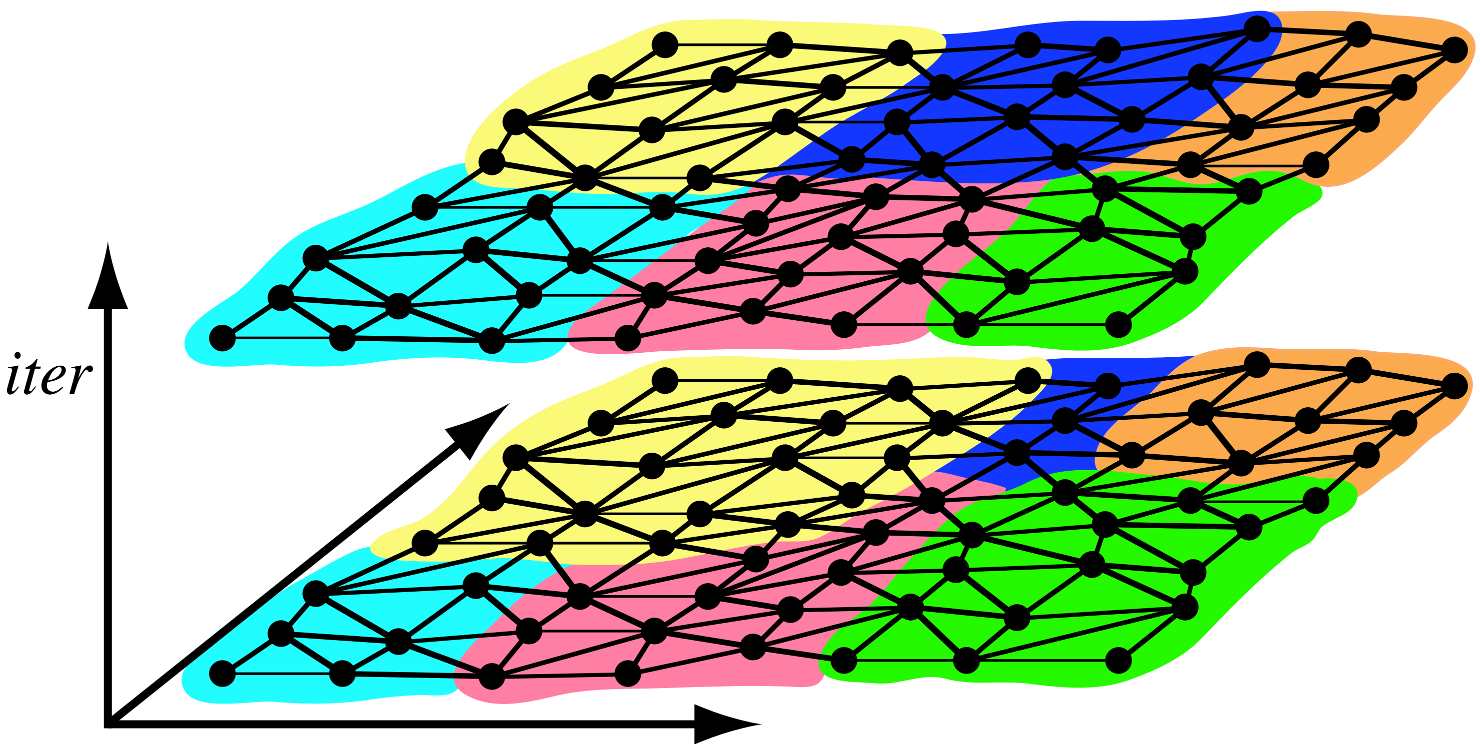
# Re-grow Using New Partition Order

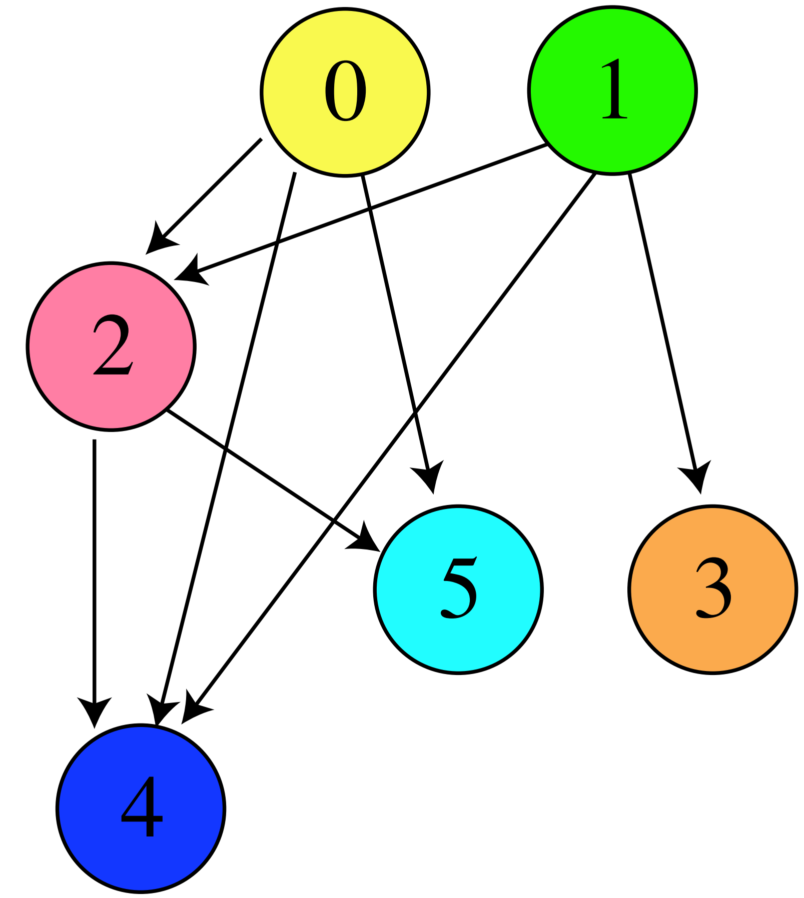



- Renumber the seed partition cells based on coloring
- Grow tiles using new ordering
- Notice that tiles 0 and 1 may be executed in parallel
- Tiles 4 and 5 may also be executed in parallel

Colorado State University

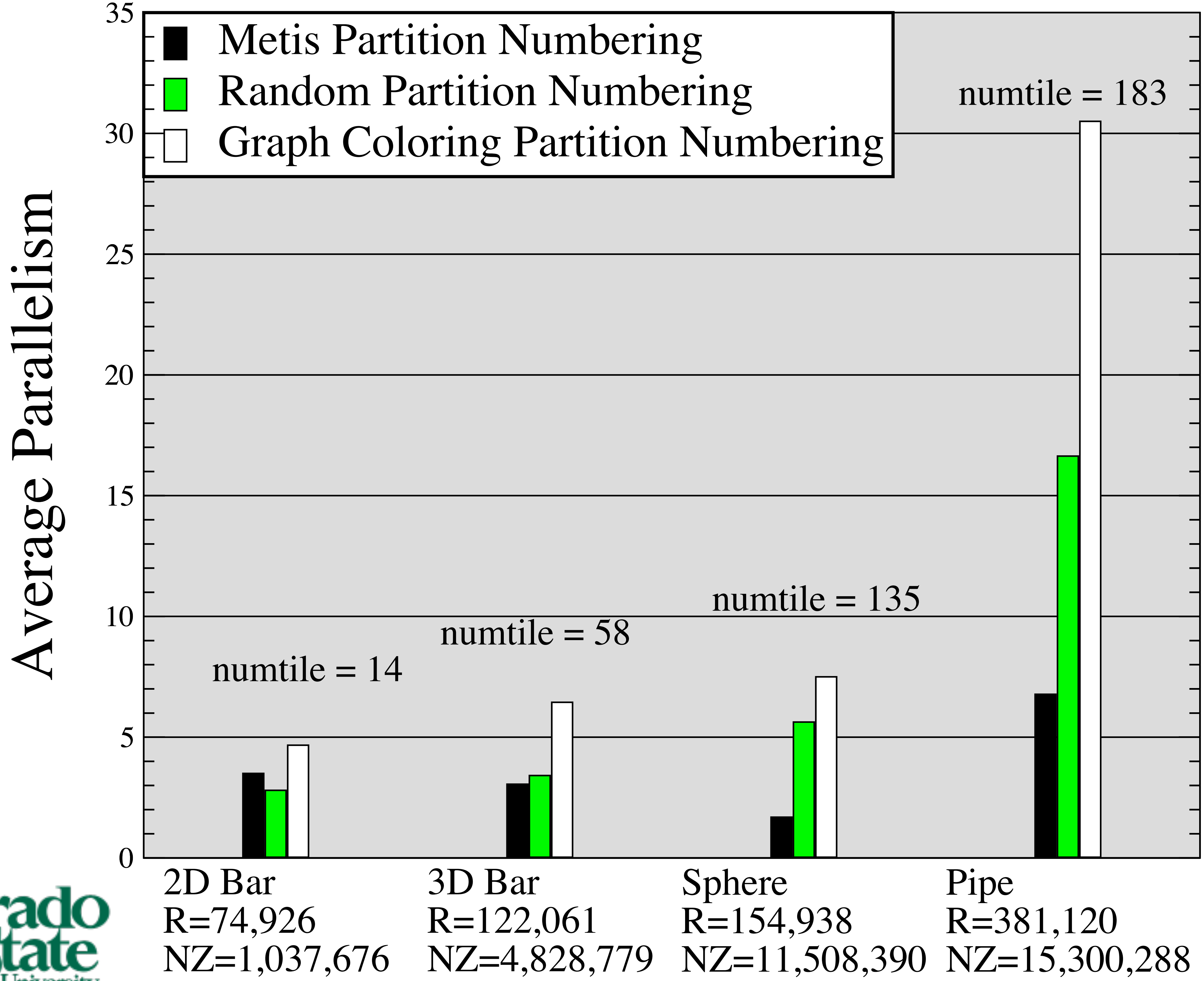
# Average Parallelism is Improved



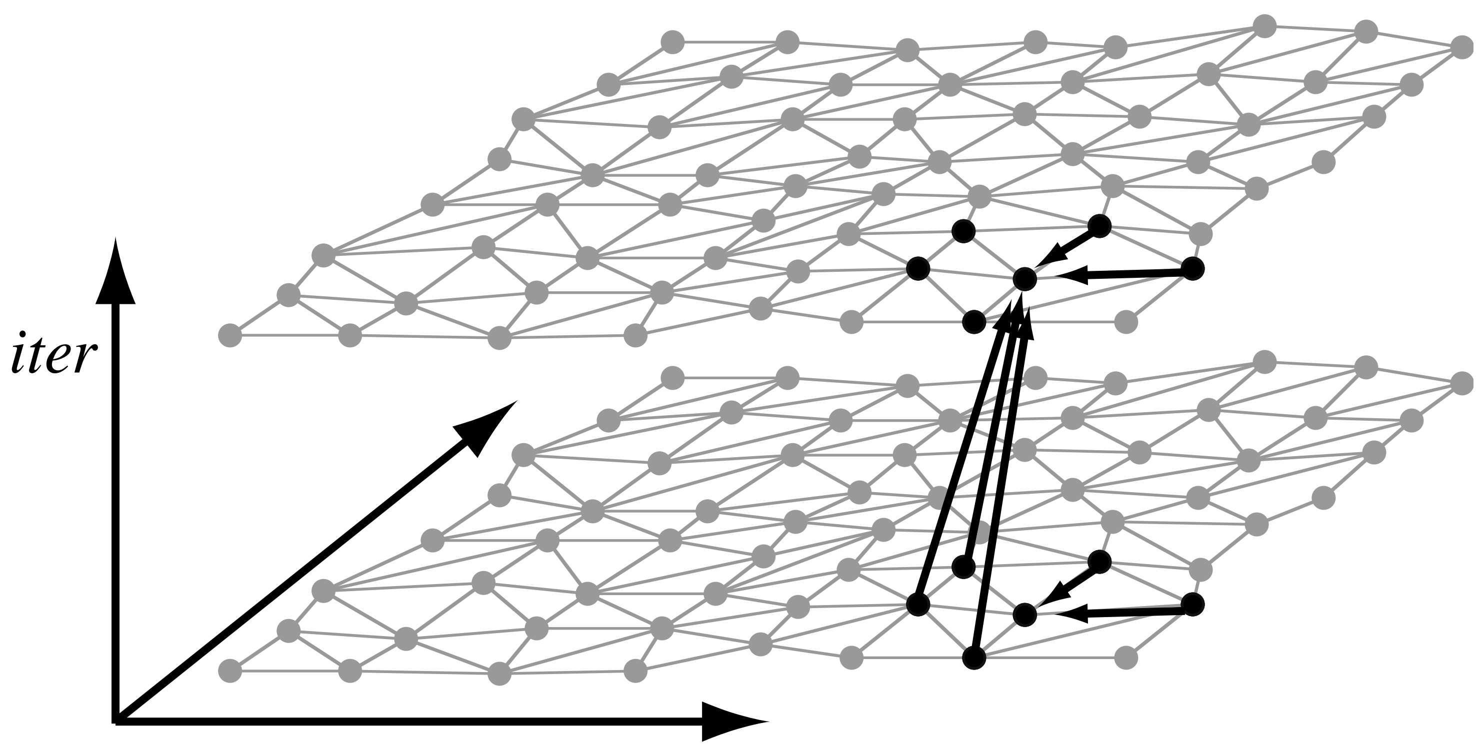

Average parallelism = (# tiles) / (# tiles in critical path)
= 6 / 3 = 2

# Improvement with Real Matrix Graphs



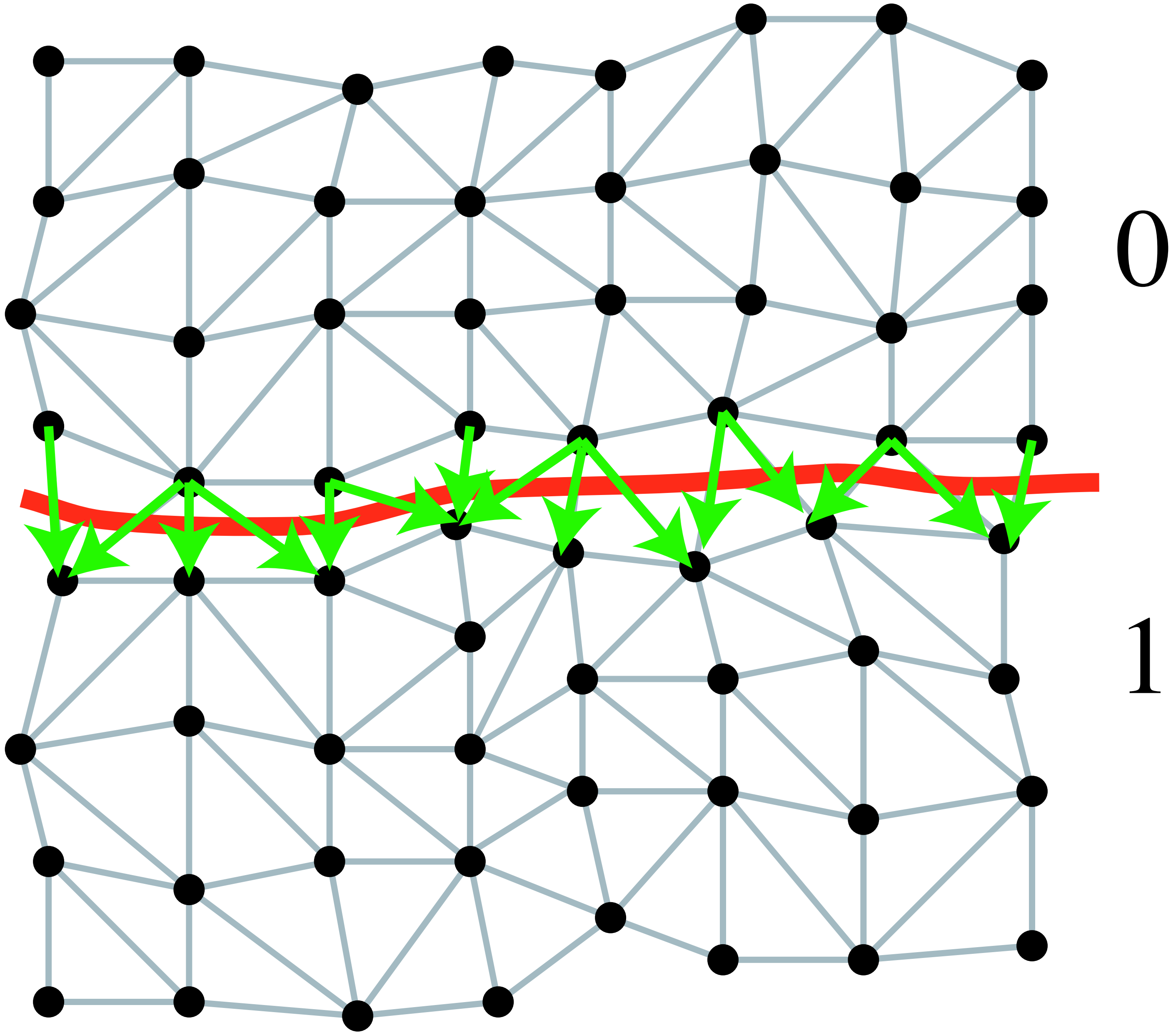

Colorado State University

# Gauss-Seidel



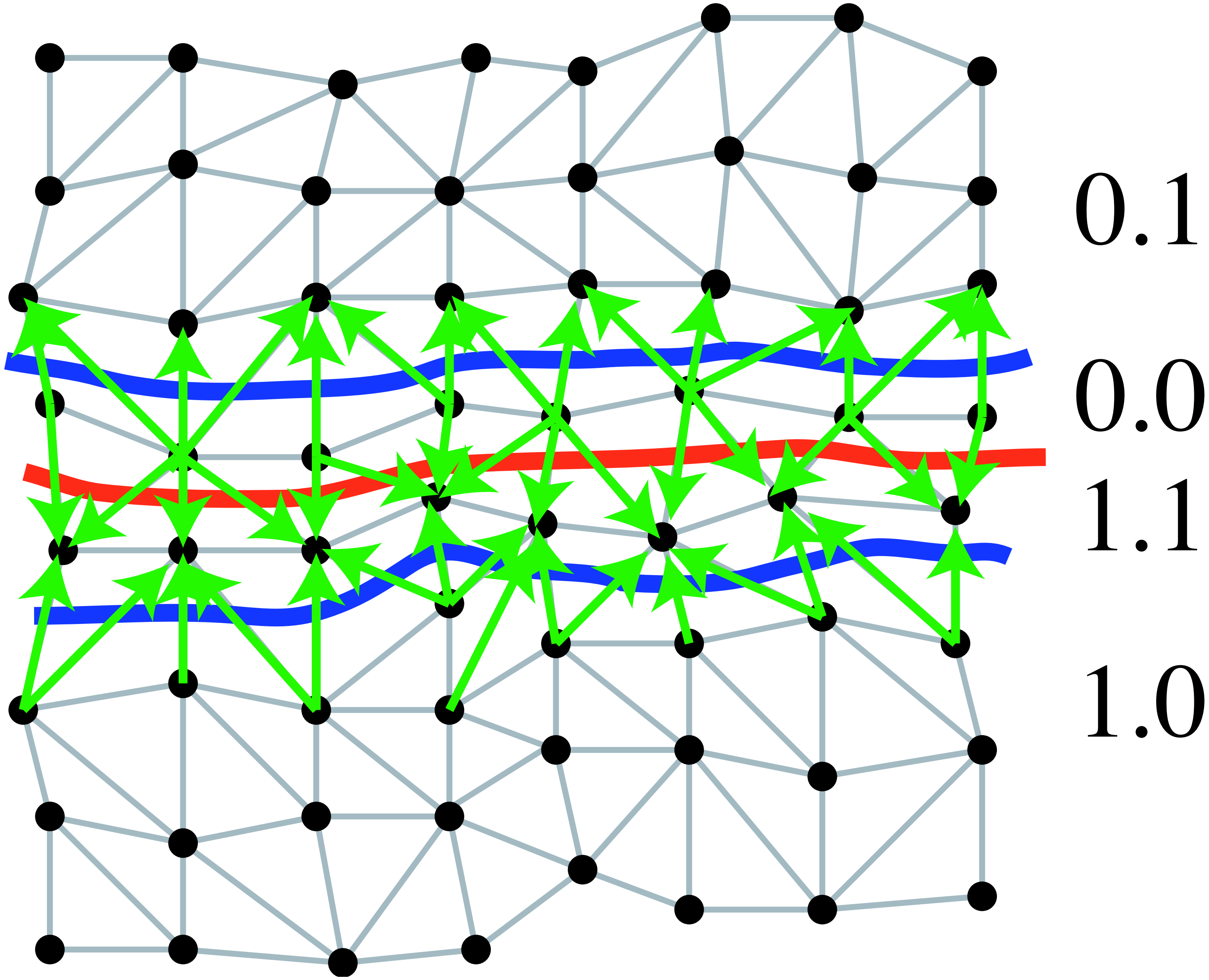

- Loop carried dependences within convergence iteration as well as between them
- Dependences depend on the ordering of the nodes
- Nodes can be reordered apriori

# Owner Computes Method Nodal Gauss-Seidel [Adams 2001]




- Renumbers nodes (rows/columns in sparse matrix)
- Make data dependences between main partitions consistent

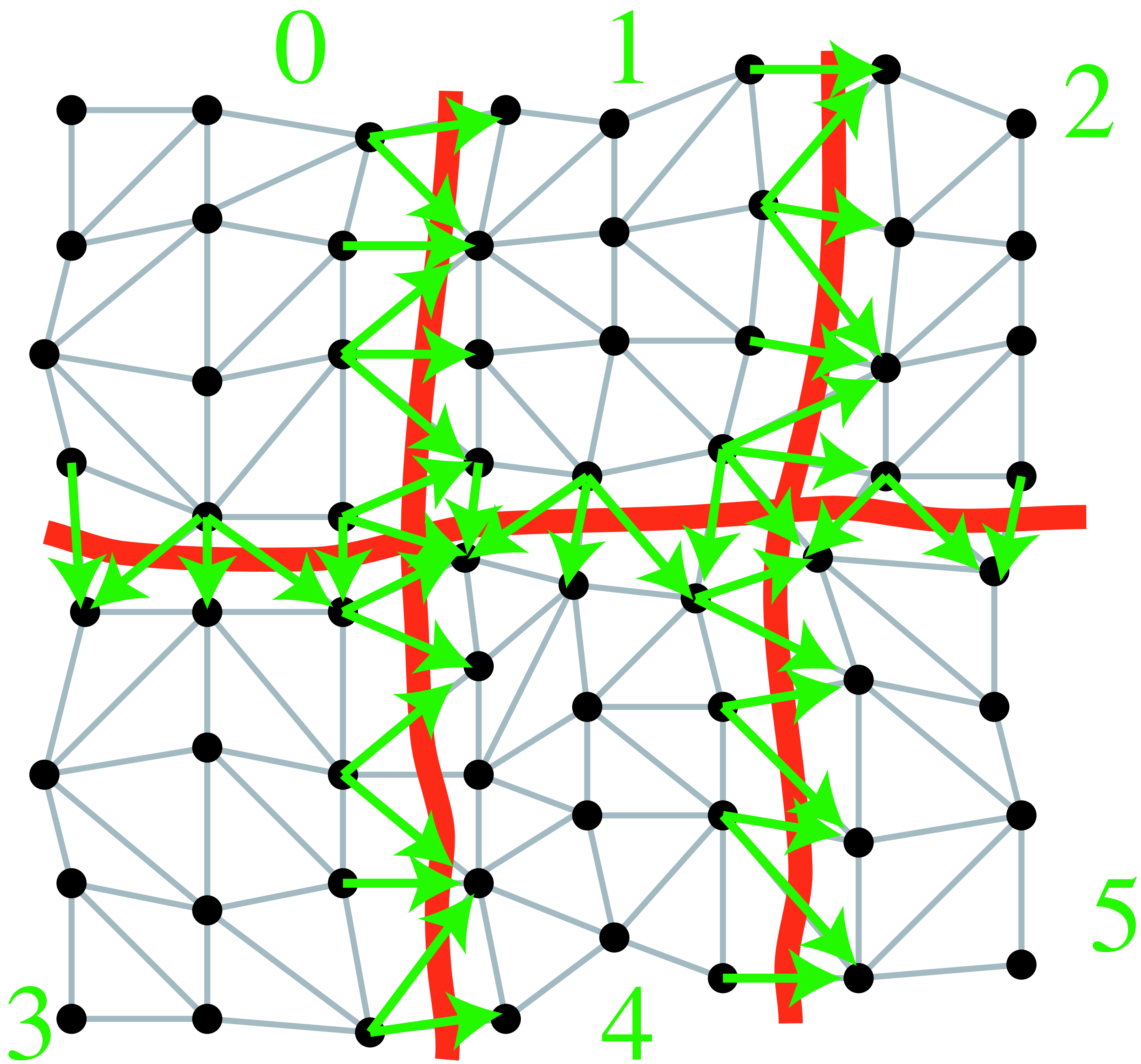
# Owner Computes Method
# Nodal Gauss-Seidel [Adams 2001]




- Renumbers nodes (rows/columns in sparse matrix)
- Make data dependences between main partitions consistent
- Subpartition for better parallelism

Colorado State University

# Full Sparse Tiled Gauss-Seidel




- Tile growth creates and maintains a partial ordering between nodes in matrix graph
- Reorder nodes so that partial ordering is maintained

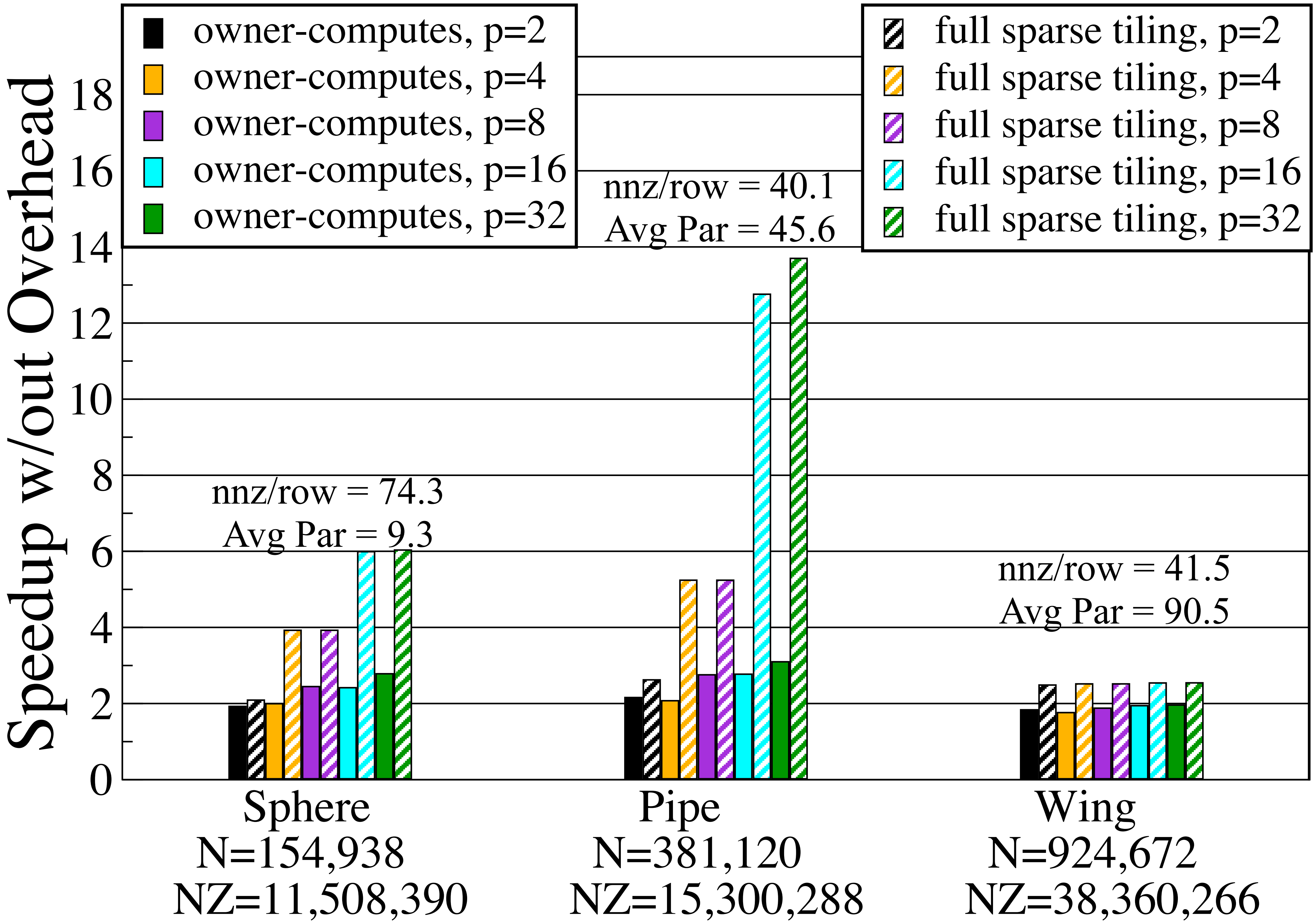
# Experimental Methodology Part 1

- **Baseline is Gauss-Seidel implemented CSR and uses provided ordering**
- **Owner computes implementation**
  - Partition matrix graph into equal-sized cells for each processor
  - Sub-partition and reorder on each processor for intra-iteration locality
  - Violate intra-iteration dependences for an idealistic parallel efficiency
- **Full sparse tiling implementation**

Colorado State University

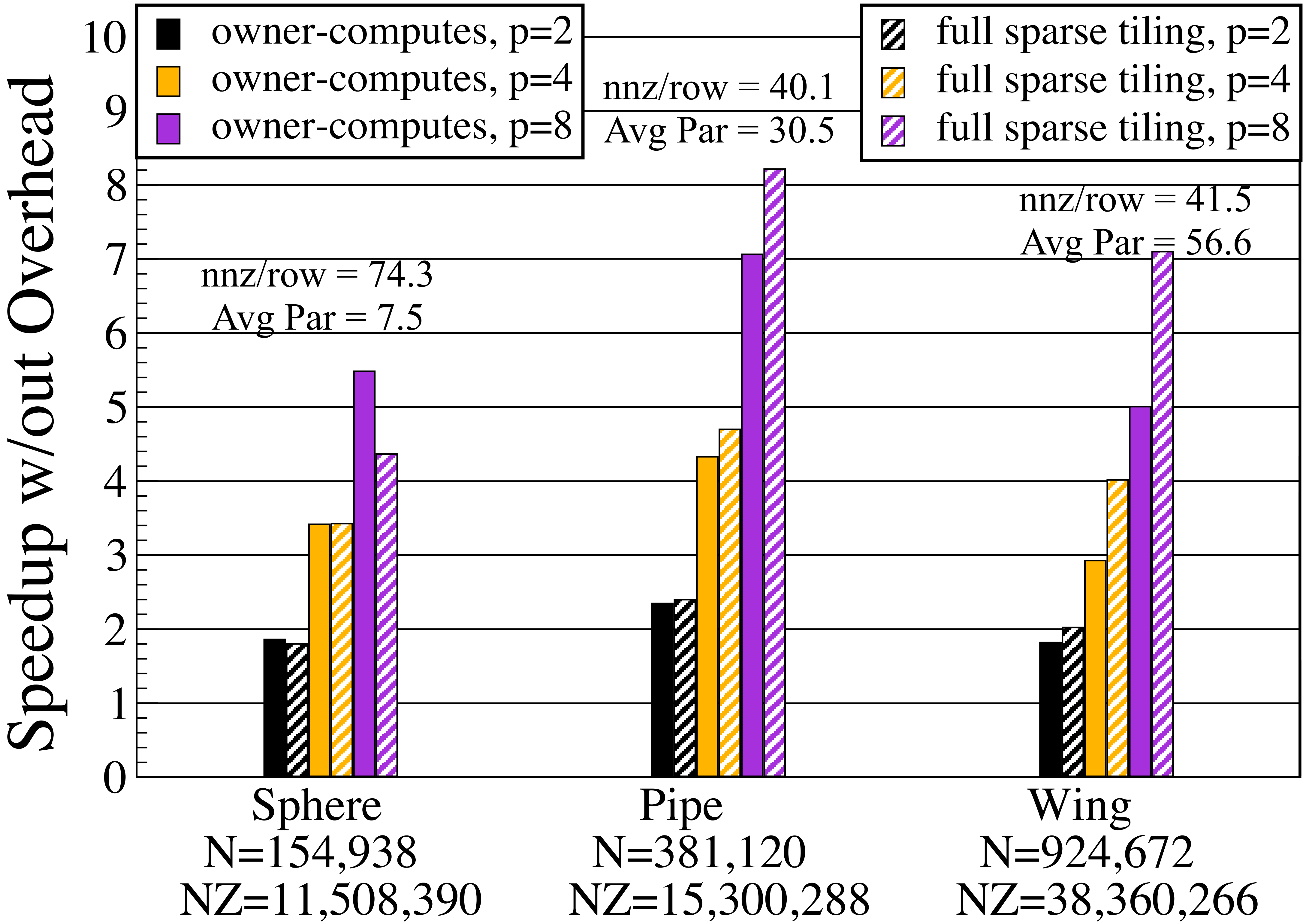
# Experimental Results

SUN HPC10000, 36 UltraSPARC II, 400MHz, 4MB L2



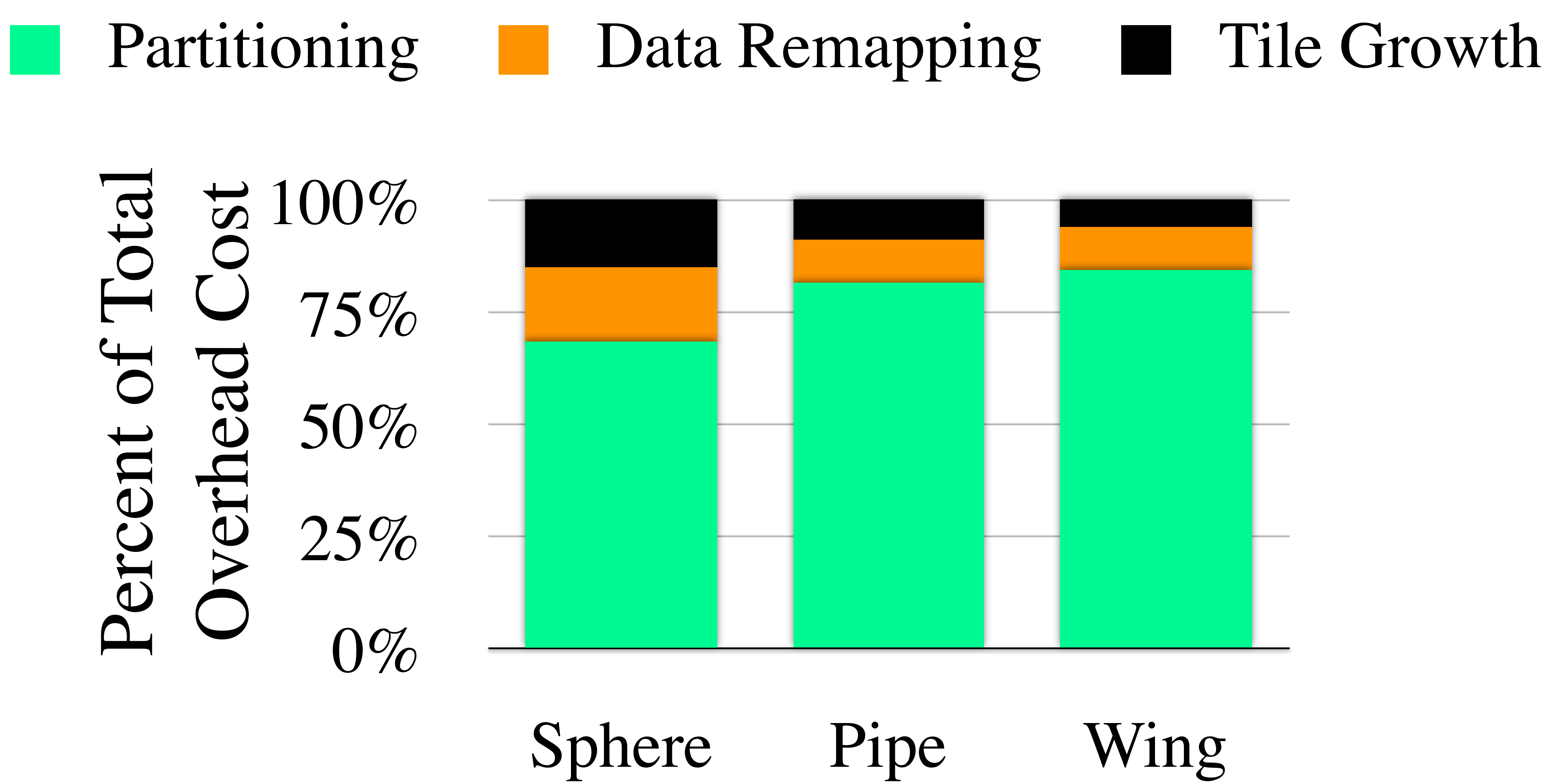
# Experimental Results

## IBM Power 3, 375MHz, 8MB L2 cache



# Overhead of Full Sparse Tiling on IBM Power 3

- Break even at 114 thru 400 convergence iterations
- Partitions created with Metis package [Karypis98]



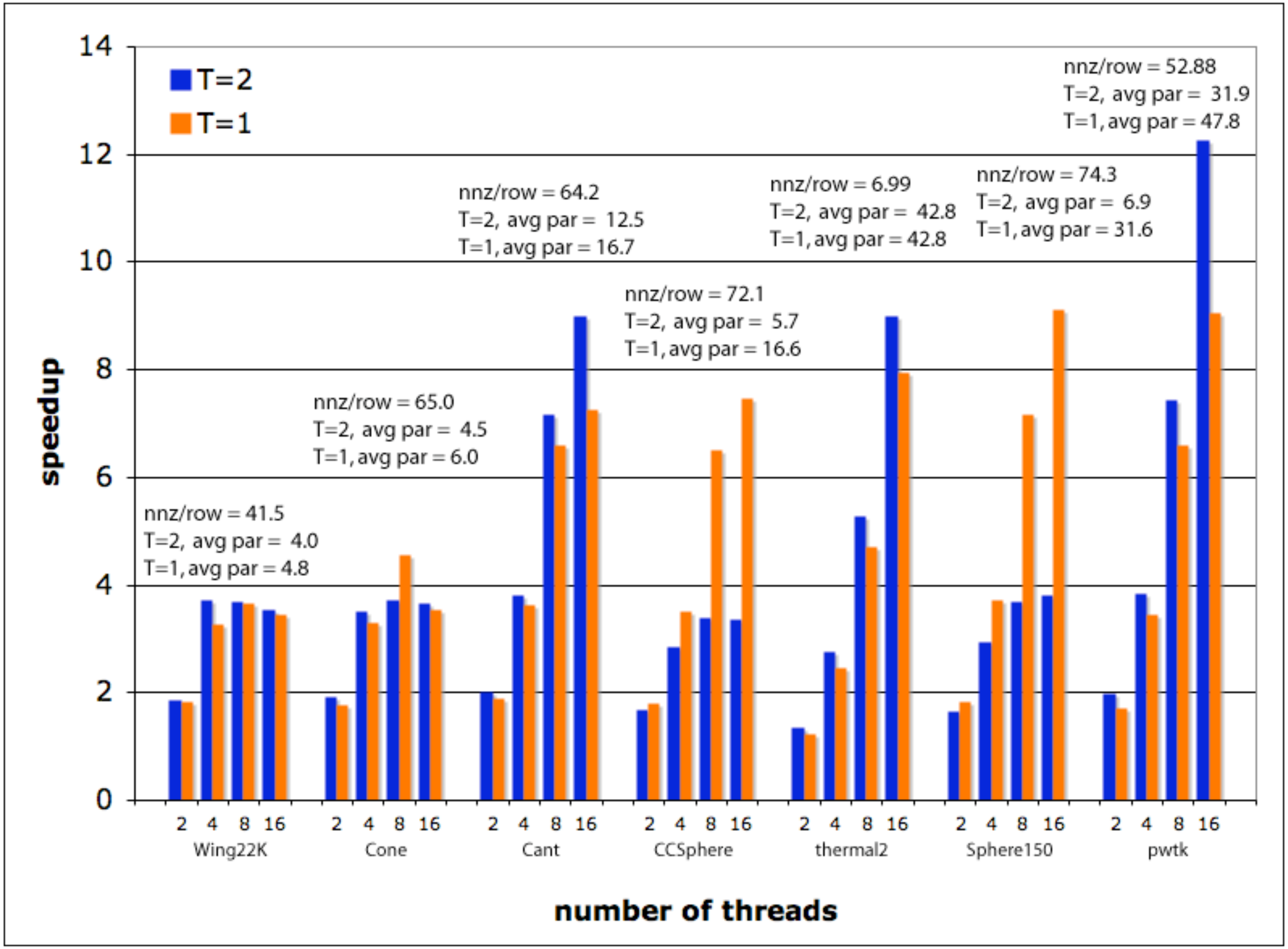
# Experimental Methodology Part 2

- **Baseline is one processor version of full sparse tiled Gauss-Seidel**
- **Owner computes implementation**
  - one convergence iteration is partitioned and a task graph is constructed between partitions, T = 1
- **Full sparse tiling implementation**
  - one convergence iteration is partitioned and a task graph is constructed between partitions, T = 2

Colorado State University

# Experimental Results for Blue Ice

IBM Power 5+, 1.9GHz, 1.9MB L2 cache, 36MB L3 cache



# Parameters Identified for Possible Auto-tuning

- Given mesh characteristics such as the average non-zeros per row, size, etc., model and tune for the following:
  - seed partition size
  - number of iterations to tile across
  - static or dynamic tile scheduling for parallelism
  - parameters for either scheduling strategy
- Would overlapping tiles remove the parallelism versus data locality tradeoff?

Colorado State University

# Conclusions

- Full sparse tiling exploits parallelism, intra-iteration data reuse, and inter-iteration data reuse
- Coloring the partition graph to number seed partitions improves the average parallelism in the tile dependence graph
- On shared memory processors, full sparse tiling can outperform owner-computes methods when enough parallelism is present
- More study needed of temporal data locality versus parallelism tradeoff

Colorado State University