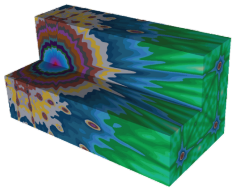
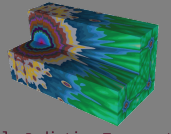


CERT
Center for Exascale Radiation Transport

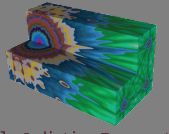
Partitioning Optimization for Massively Parallel Transport Sweeps on Unstructured Grids

Tarek Ghaddar
Chair: Jean Ragusa
Committee: Jim Morel, Marvin Adams, Nancy Amato

Introduction



- Massively parallel transport sweeps have been shown to scale up to 750,000 cores on logically cartesian grids.
- Structured meshes are somewhat limiting when attempting to simulate more complex problems and experiments.
- Unstructured meshes allow us to simulate realistic problems, but introduce unbalanced partitions.
- PDT (Texas A&M's massively parallel transport code) introduced two load balancing algorithms that repartition the mesh in order to obtain a roughly equivalent amount of cells per processor.
- However, this can sacrifice the optimal sweep partitioning (cut lines all the way through the domain) in favor of balance.
- The work proposed will balance perfect load balancing and optimal sweep partitioning in order to achieve the best possible time to solution.



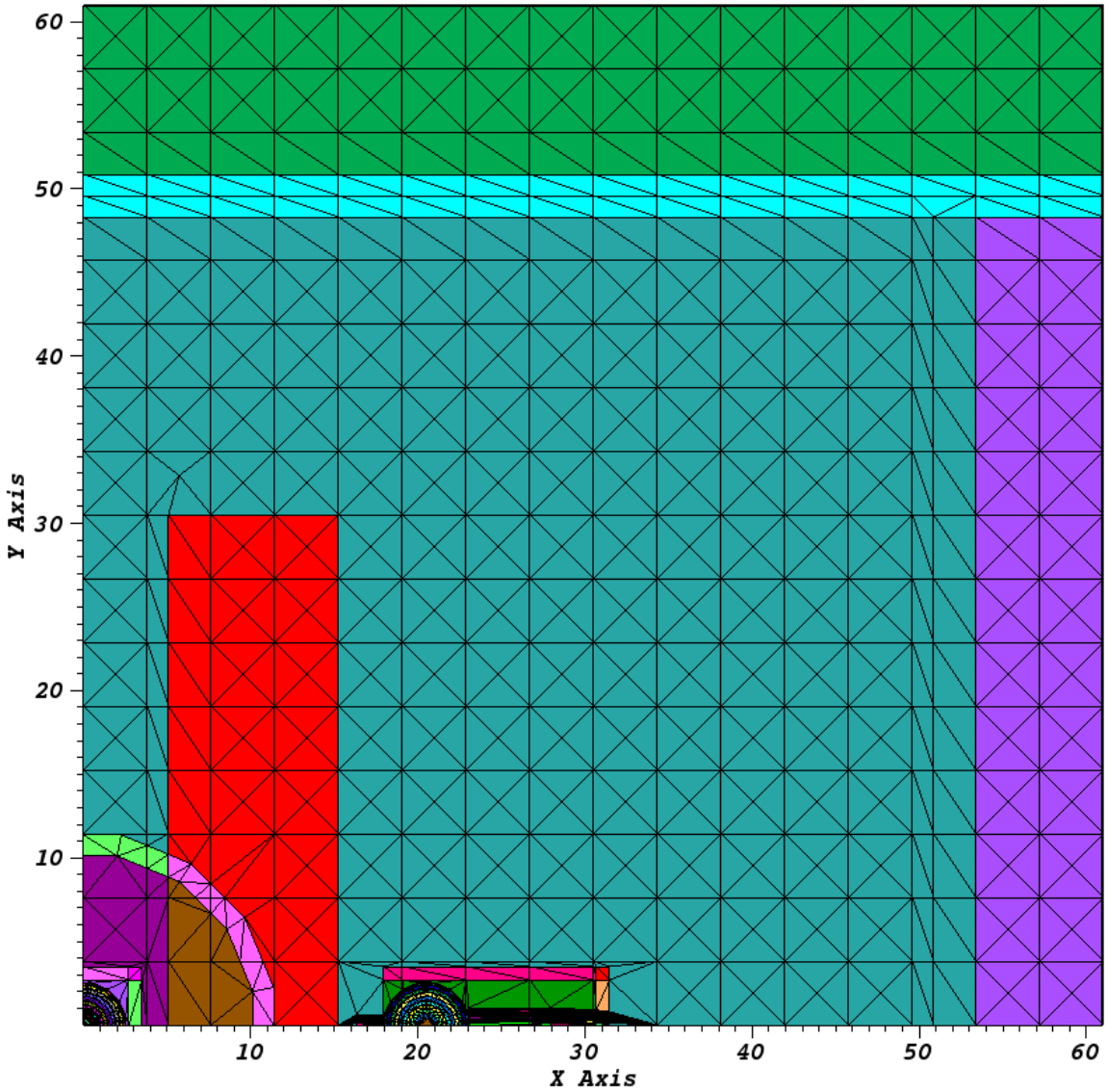
Load Balance Metric

- Max cells per subodmain divided by the average cells per subdomain:
  - $f = \frac{\max_{ij}(N_{ij})}{\frac{N_{tot}}{I \cdot J}}$
- Column-wise metric: $f_I = \max_i[\sum_j N_{ij}] / \frac{N_{tot}}{I}$
- Row-wise metric: $f_J = \max_j[\sum_i N_{ij}] / \frac{N_{tot}}{J}$
- Per Column row-wise metric: $f_{J_i} = \max[N_{ij}]_i / \frac{\sum_j N_{ij}}{J_i}$

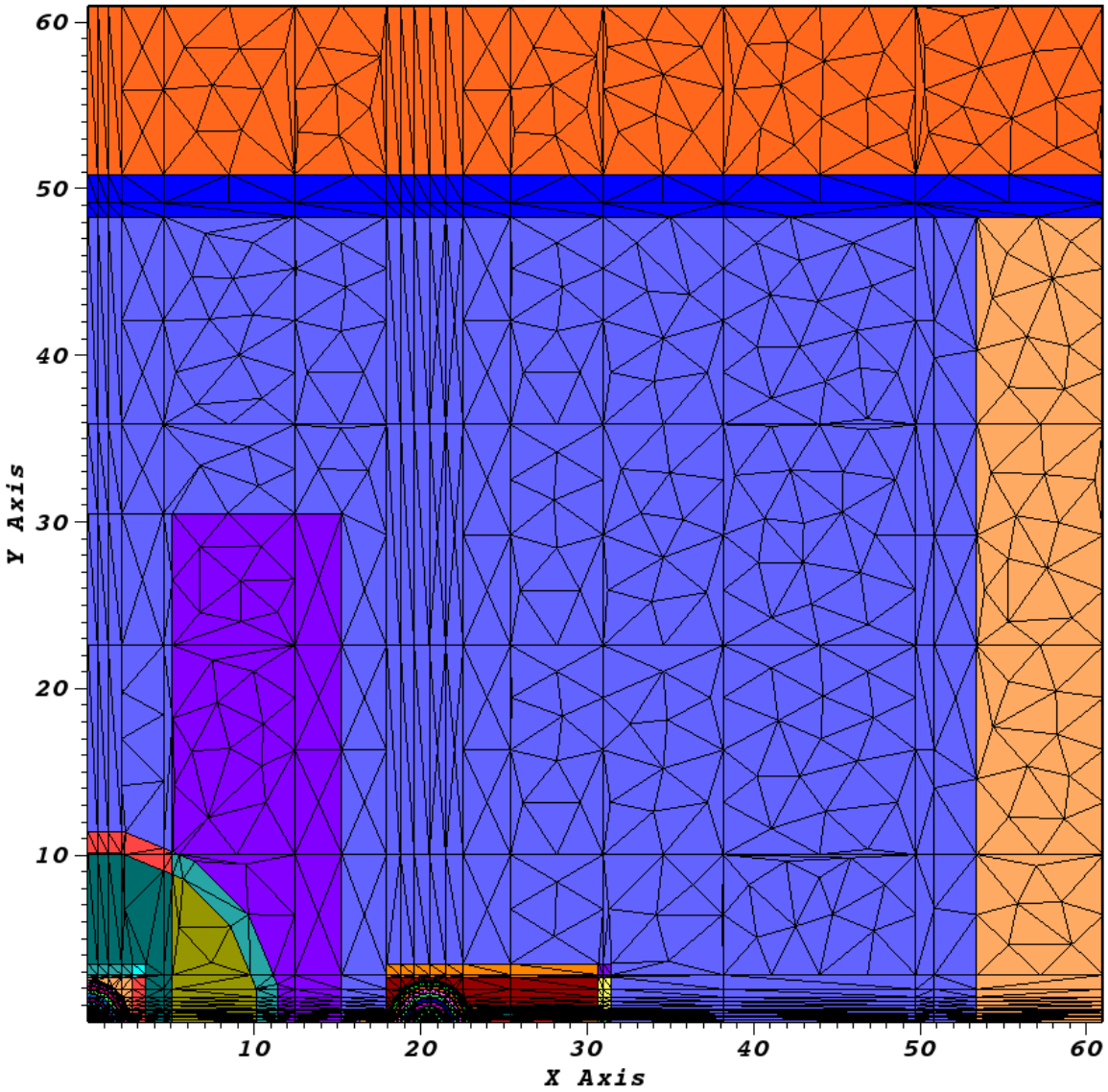
Goal: minimize f using locations of cut lines in X and Y

Subsequent improvement in algorithm: once dimension 1 has been balanced, balance dimension 2, then balance dimension 3 (load balancing by dimension)

No Load Balancing, f = 41.82

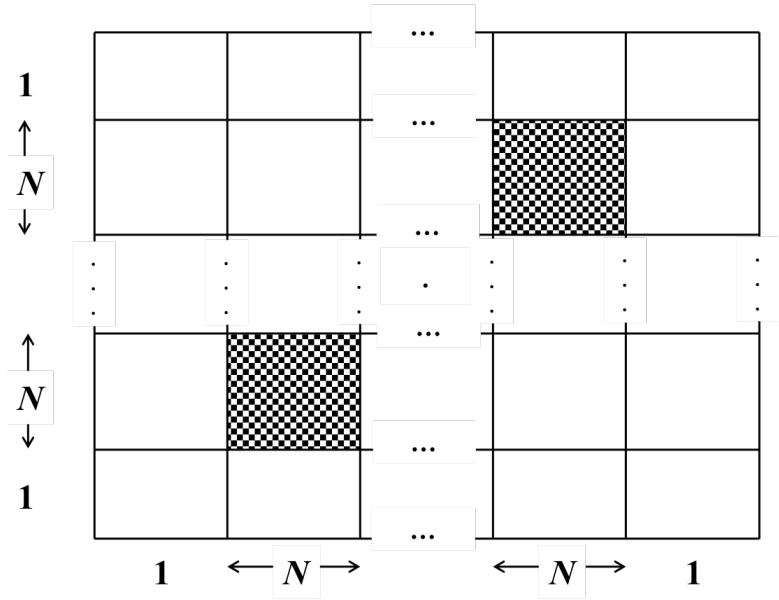


Load Balancing, f = 2.97

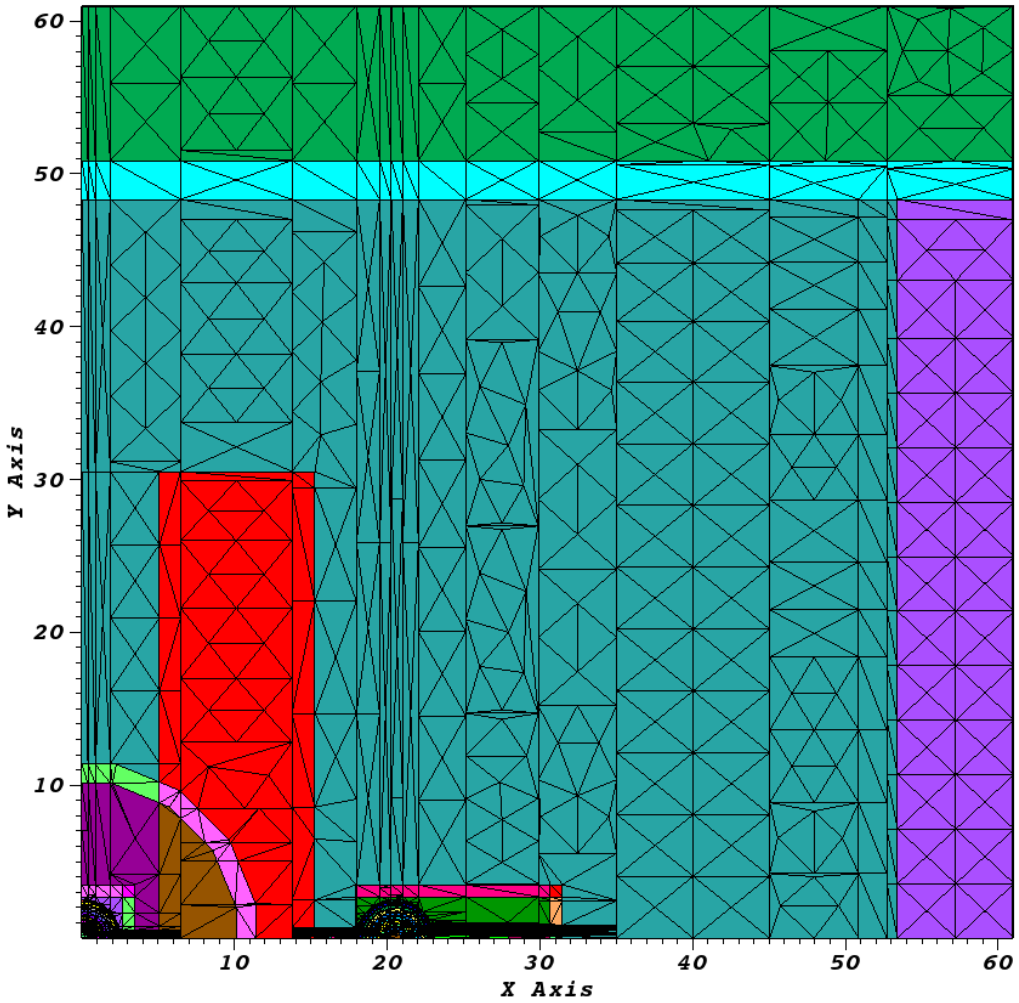


Theoretical Motivation for LBD (Courtesy of Dr. Adams)

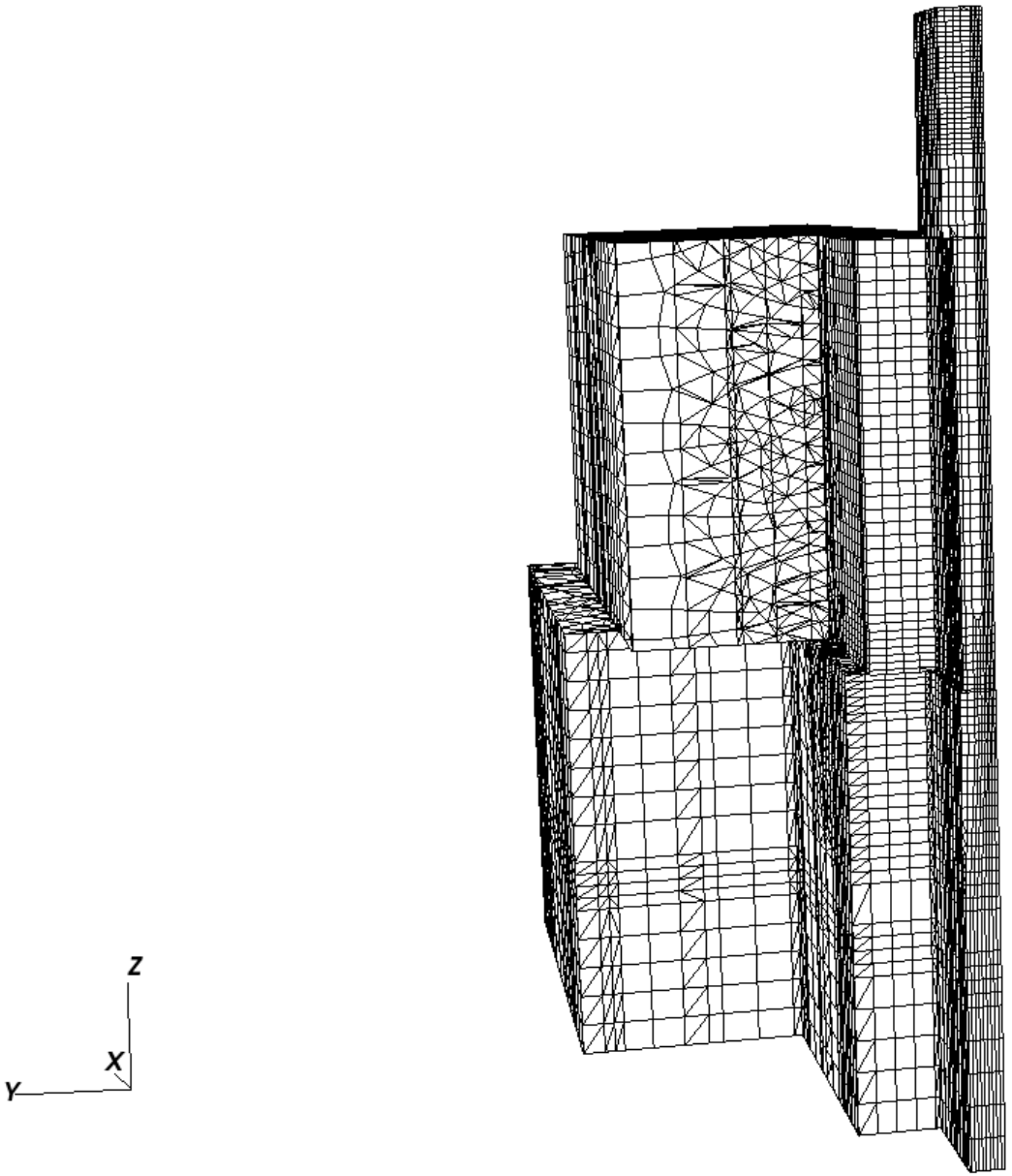
- Consider simple 2D layout with M unaligned patches of high mesh density
- The cellset layout is $[M(N+1)+1] \times [M(N+1)+1]$ but only MN^2 cellsets have much work.
- Load Imbalance Factor $= \frac{(M(N+1)+1)^2}{MN^2} \xrightarrow{N \to \infty} \frac{M^2N^2}{MN^2} = M$



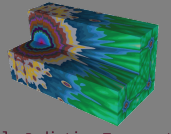
Load Balancing By Dimension, f = 2.02



3D Load Balancing By Dimension

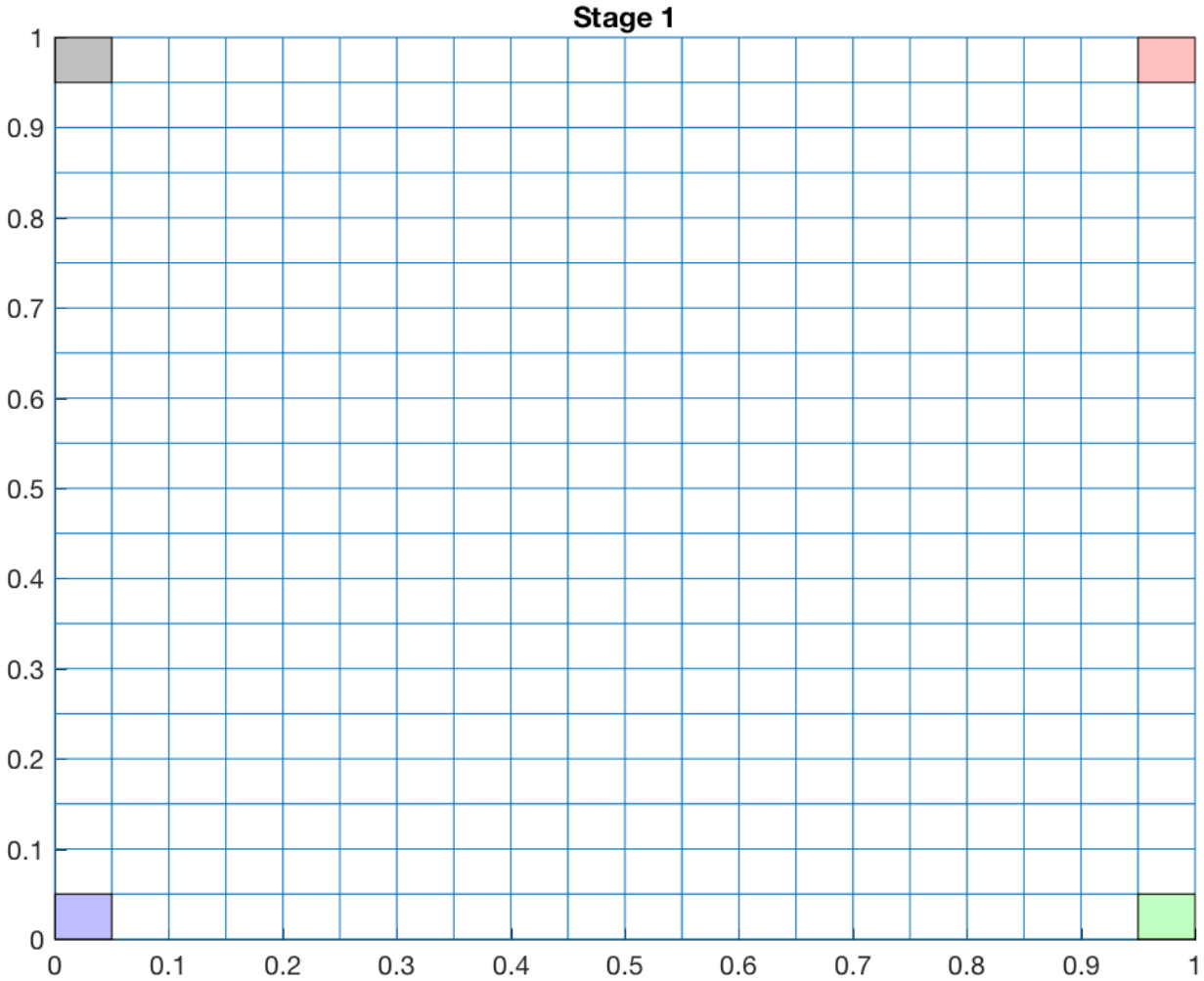


Consequences of Load Balancing By Dimension

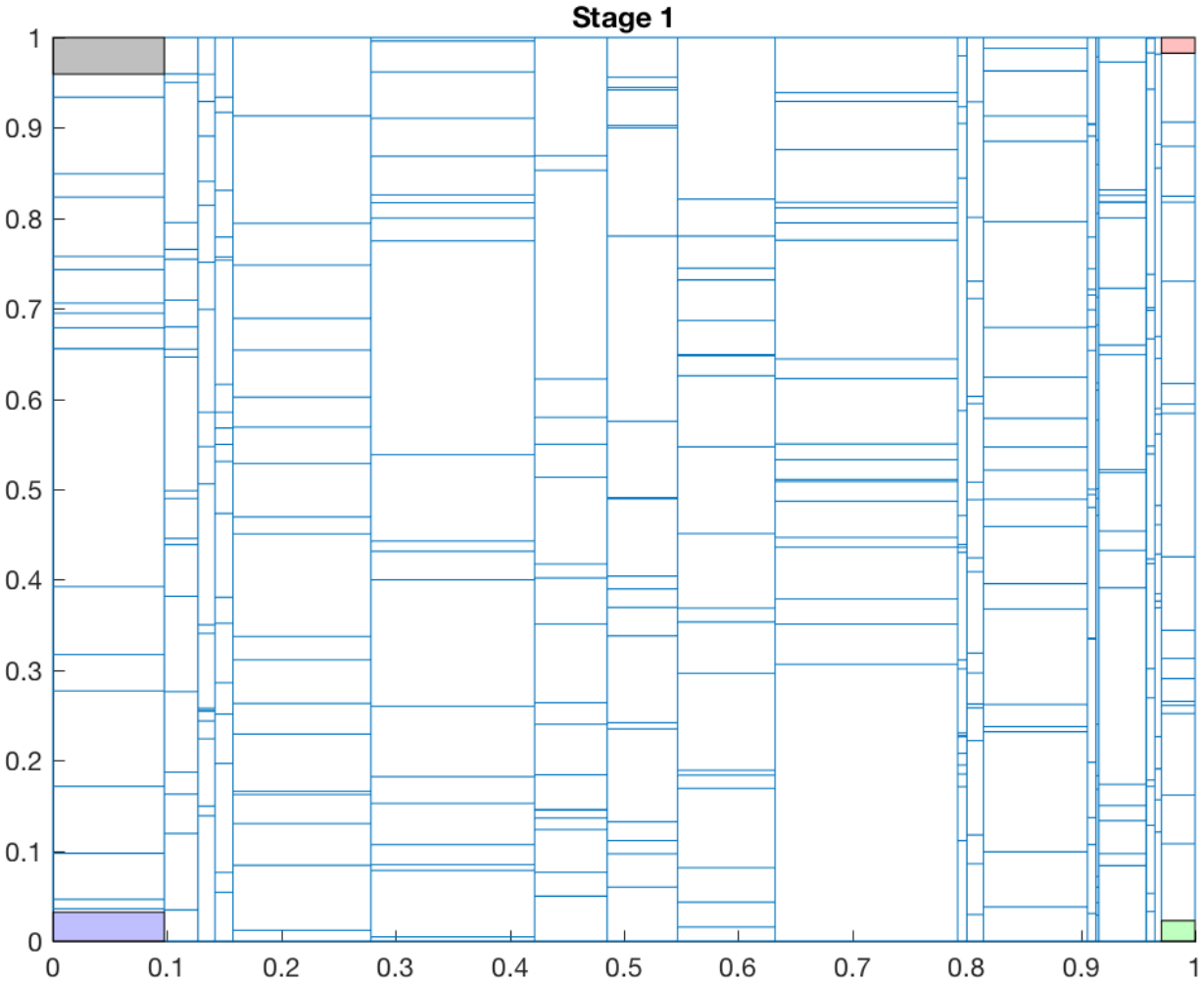


- Perfect load balance in some cases will come at the cost of optimal sweeping.
- Time to solution is the most important parameter, and if keeping a more optimal sweeping grid means a less balanced problem, then so be it.
- The concept of a stage may be misleading when dealing with imbalanced partitions, as we cannot easily characterize the idle time.
- A time-to-solution estimator must be built to more accurately predict sweep time.

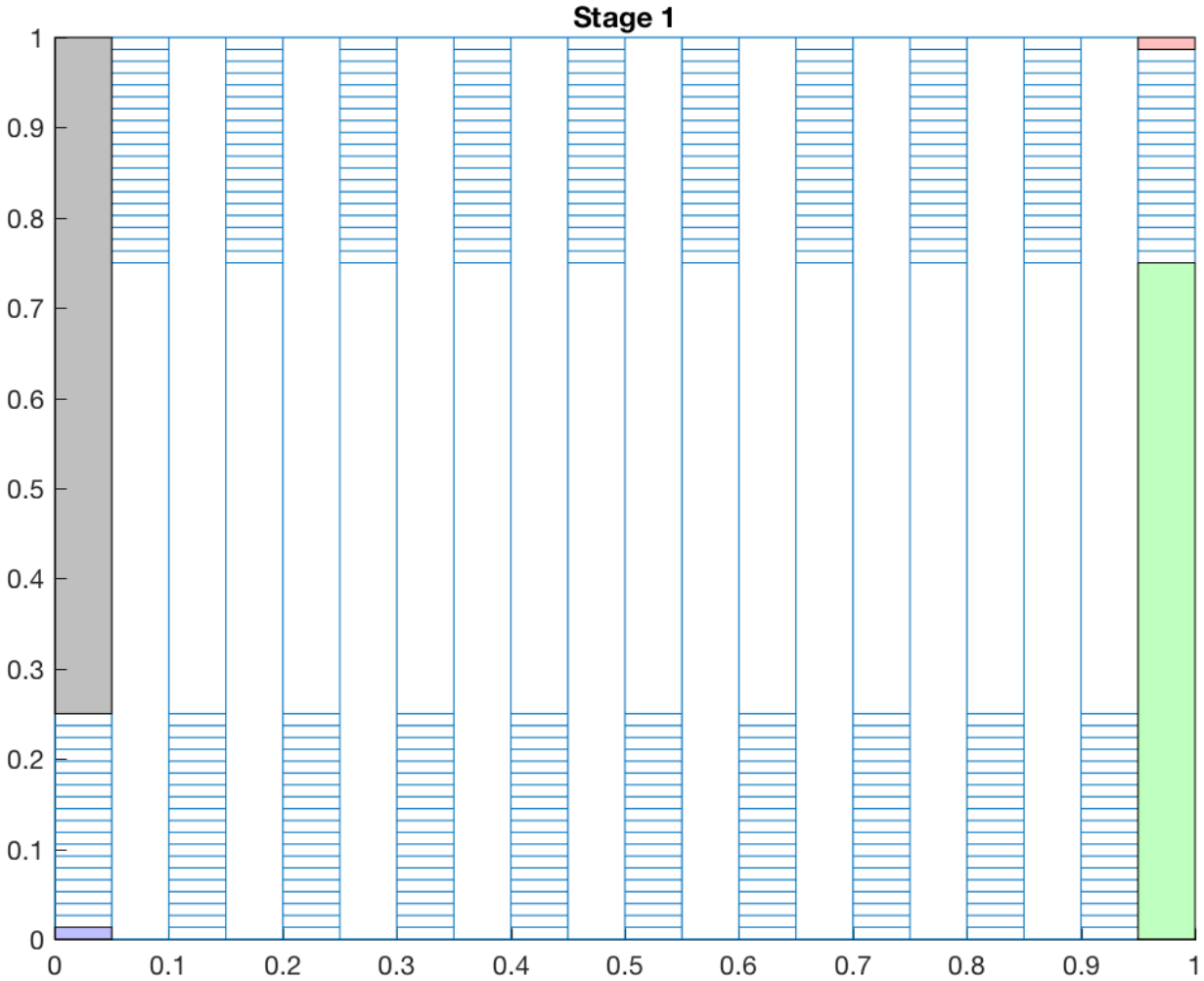
Sweep on Regular Grid with 3 Angle Sets



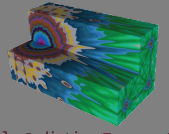
Sweep on LBD Grid with 3 Angle Sets



Sweep on Worst Grid with 1 Angle Set

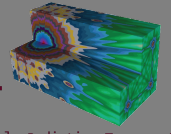


Overview



- We need to optimize the cut line location not for balance, but for the best possible sweep time.
- We must build a time-to-solution estimator that calculates the time to solution for a given cut line partitioning.
- The time to solution estimator will be fed into an optimizing function that minimizes the time to solution. The cut lines corresponding to the minimum time to solution are the optimal partitioning scheme.

Time to Solution Estimator



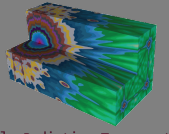
- For a given partition, this tool will build cellset-level (not cell level) directed task dependence graphs for each octant, and weight the edges of that graph based on a set of criteria.
- The estimator returns the maximum time to sweep across one of the graphs.
- The maximum sweep time corresponds to the graph that has the maximum longest weighted path.

$$\text{TOS} = f(P_x, P_y, P_z, \text{cut lines, threads, machine params}) \quad (1)$$

Important Assumption

This process assumes that there are NO cycles in the TDGs. All graphs are acyclic.

Time to Solution Estimator



1. Given a partitioning scheme (cut lines), build adjacency matrix.
2. Build Directed Acyclic Graphs (DAGs) from the adjacency matrix, one for each octant.
3. Weight the DAG's edges based on:
   - Solve and communication of each subset to its neighbors.
   - Conflicts that arise between DAGs due to the sweep.
   - This final edge weight reflects the amount of time it takes to solve the base node.
4. Compute solve time by getting the maximum edge-weighted sum of the longest paths for all DAGs.

Weighting the Graphs

- In order to weight the graph properly, we need to have a mesh density function that we can calculate the cells (and then unknowns) per subset from.
- The weight of the edge between two nodes (subsets) in the graph represents the solve and communication time of the base node.

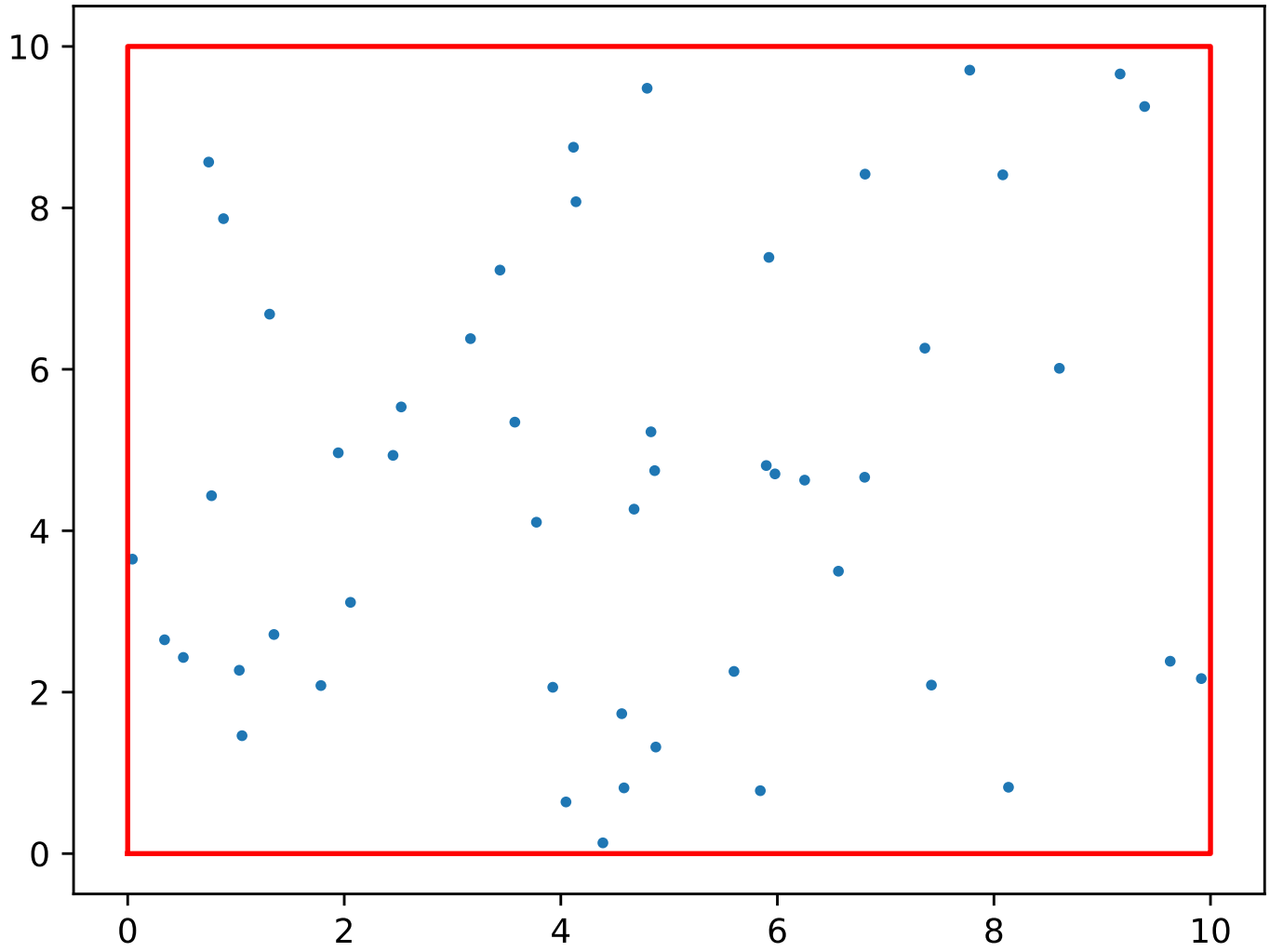
$$\text{cells per subset} = \int_{x_i}^{x_{i+1}} \int_{y_j}^{y_{j+1}} \int_{z_k}^{z_{k+1}} \text{mesh density } dxdydz \tag{2}$$

$$\text{weight} = N_u \cdot T_u(\text{threads}) + N_b \cdot T_{\text{comm}} + \text{latency} \cdot M_L \tag{3}$$

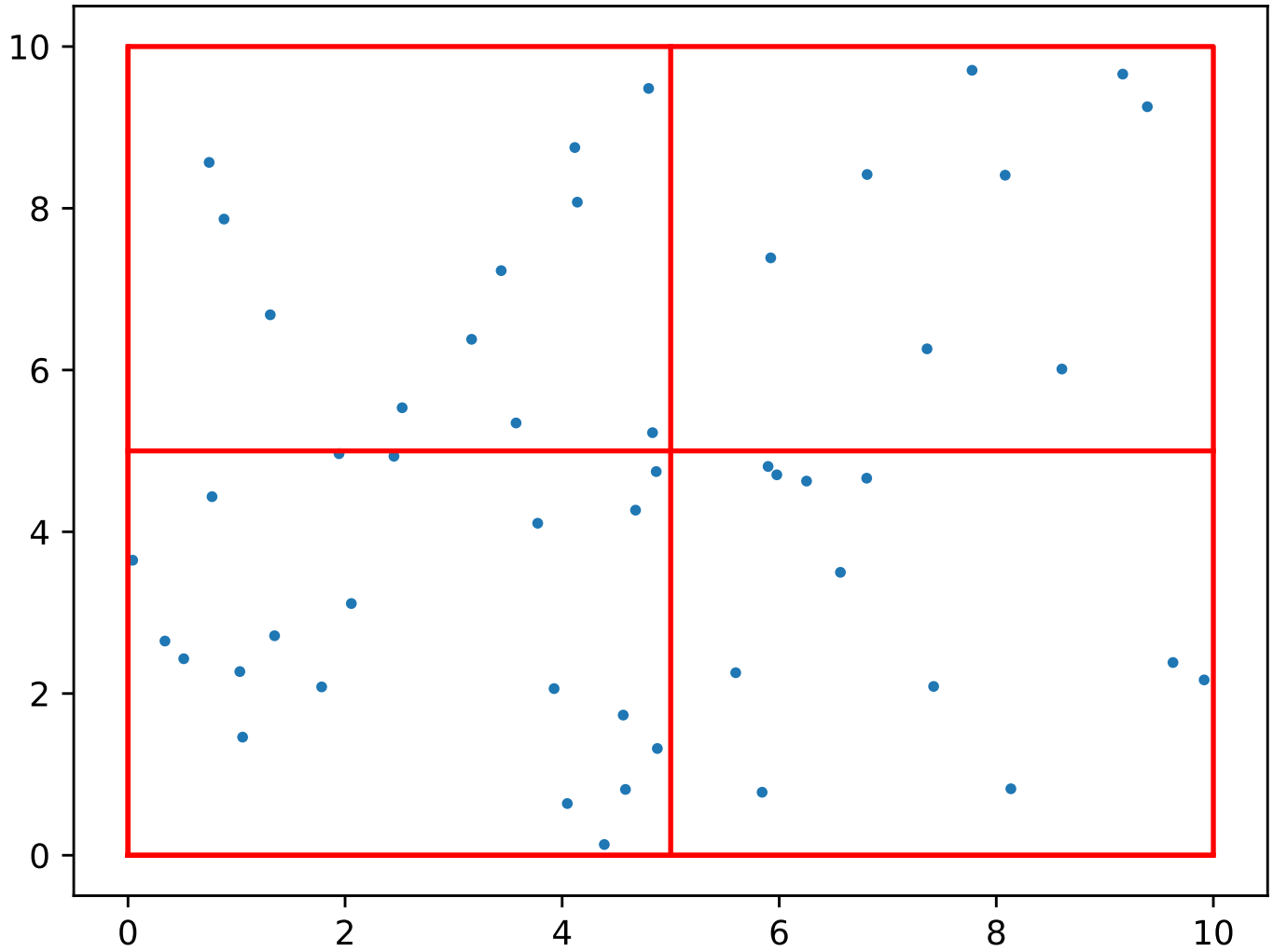
$$N_u = \text{num cells} \cdot \text{unknowns per cell} \tag{4}$$

$$N_b \approx (\text{num cells})^{\frac{2}{3}} \cdot \text{unknowns per boundary cell} \tag{5}$$

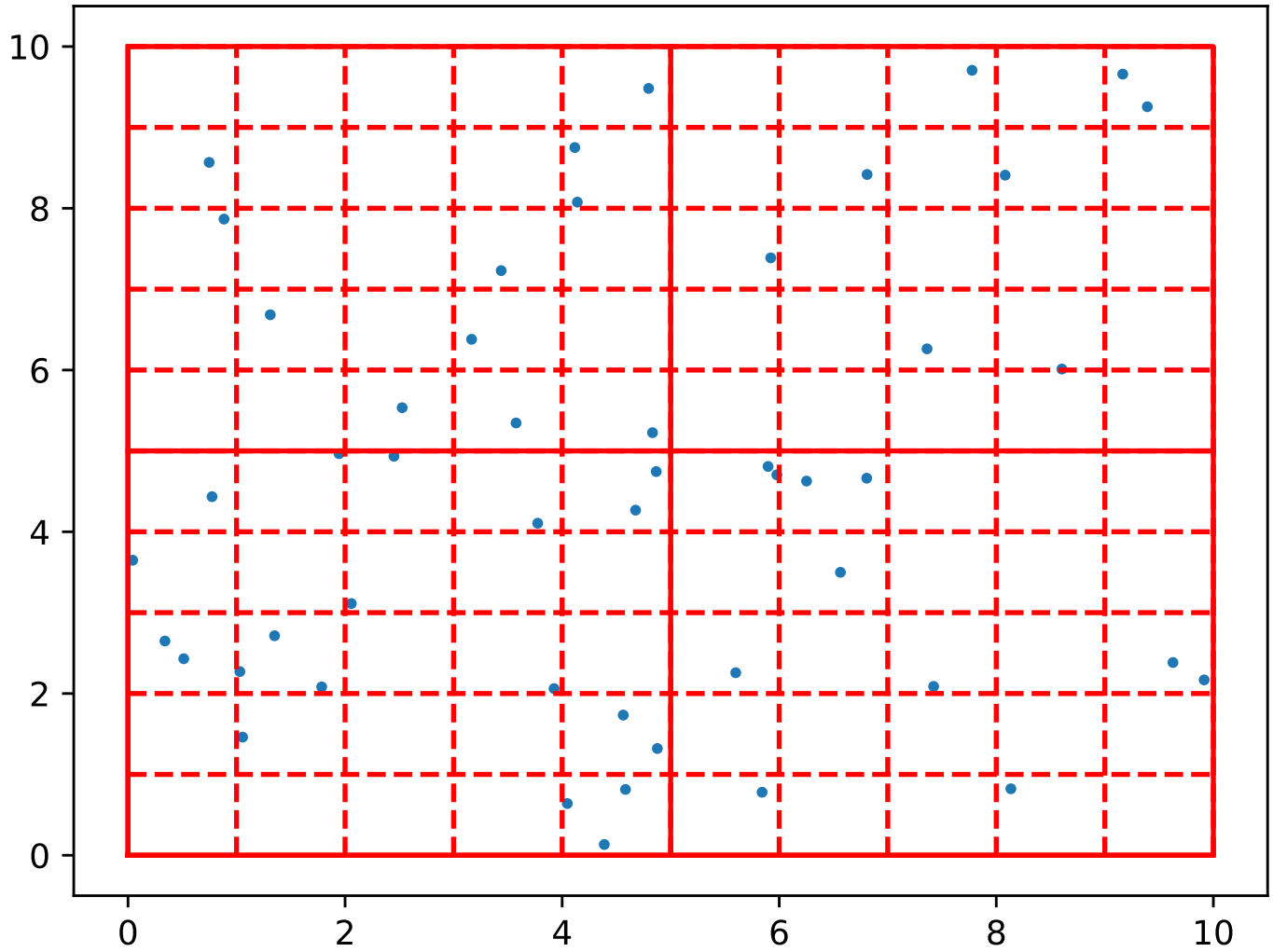
Determining Cells per Subset



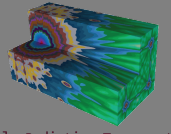
Determining Cells per Subset



Determining Cells per Subset

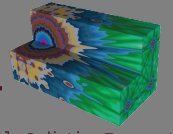


Conflict Detection and Resolution



- Once all edges in all TDGs are appropriately weighted according to Eq. 3, we need to detect and resolve sweep conflicts between the 8 octants.
- For perfectly balanced and optimally partitioned problems (structured grids) we will default to a depth-of-graph conflict resolution.
- For unbalanced partitions, we will default to a first come first serve conflict resolution.
- We check for conflicts only amongst the longest paths of each graph. This simplification takes into account the upstream dependencies of each node, without losing the accuracy of the time to sweep each graph.

First Come First Serve Conflict Resolution

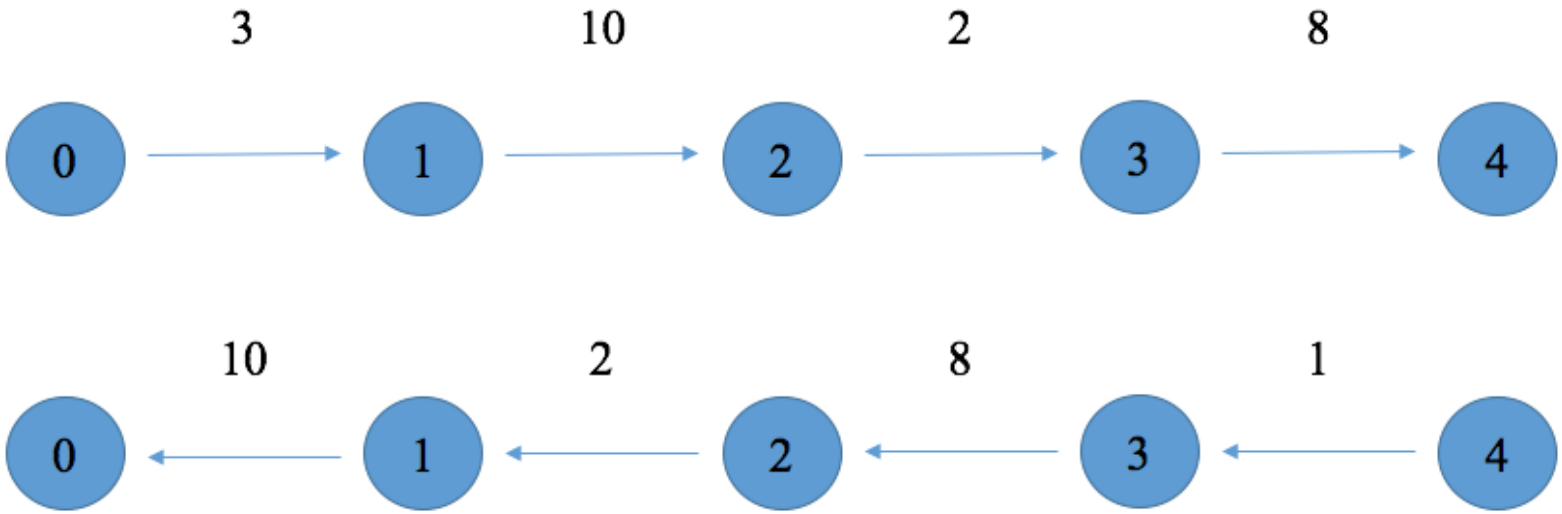


- The first octant to arrive to a node will begin solving it, and the remaining octants will incur a delay (if applicable).
- The delay is reflected in each remaining TDG by adding the corresponding delay as a weight to the applicable edge.
- If two octants arrive to a node at the same time, the octant with the greater remaining depth-of-graph and priority octant wins the tie.

Potential Future Work: hybrid depth-of-graph/first come first serve conflict resolution

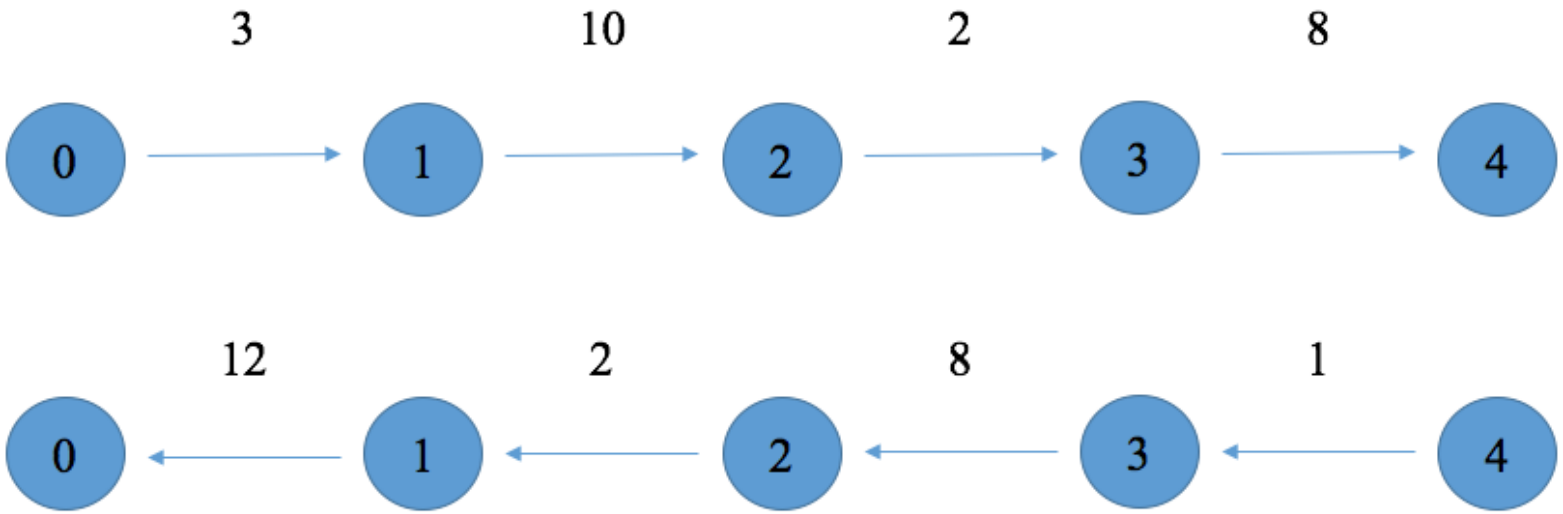
- Two (or more) paths arrive to a node at different time, and when that node is free to solve, the path with the greater depth-of-graph remaining solves first.

Conflict Detection



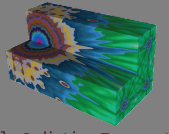
- A conflict arises at Node 1. The first path arrives at $t = 3$, but takes 10 seconds to solve.
- The second path arrives at $t = 11$, while the first path is solving Node 1.

Conflict Resolution



- The second path waits 2 seconds to solve Node 1.
- This delay is added to the edge weight between Node 1 and Node 0 to reflect the new total solve time for Node 1 in the second path.

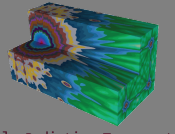
General Conflict Resolution Algorithm



1: **for** $g = 1$ to $G - 1$ **do**
2: **for** *node* $= 1$ to N **do**
3: Find fastest path to *node*
4: Add delays to slower paths
5: Remove fastest path to *node* for future iterations
6: **end for**
7: **end for**

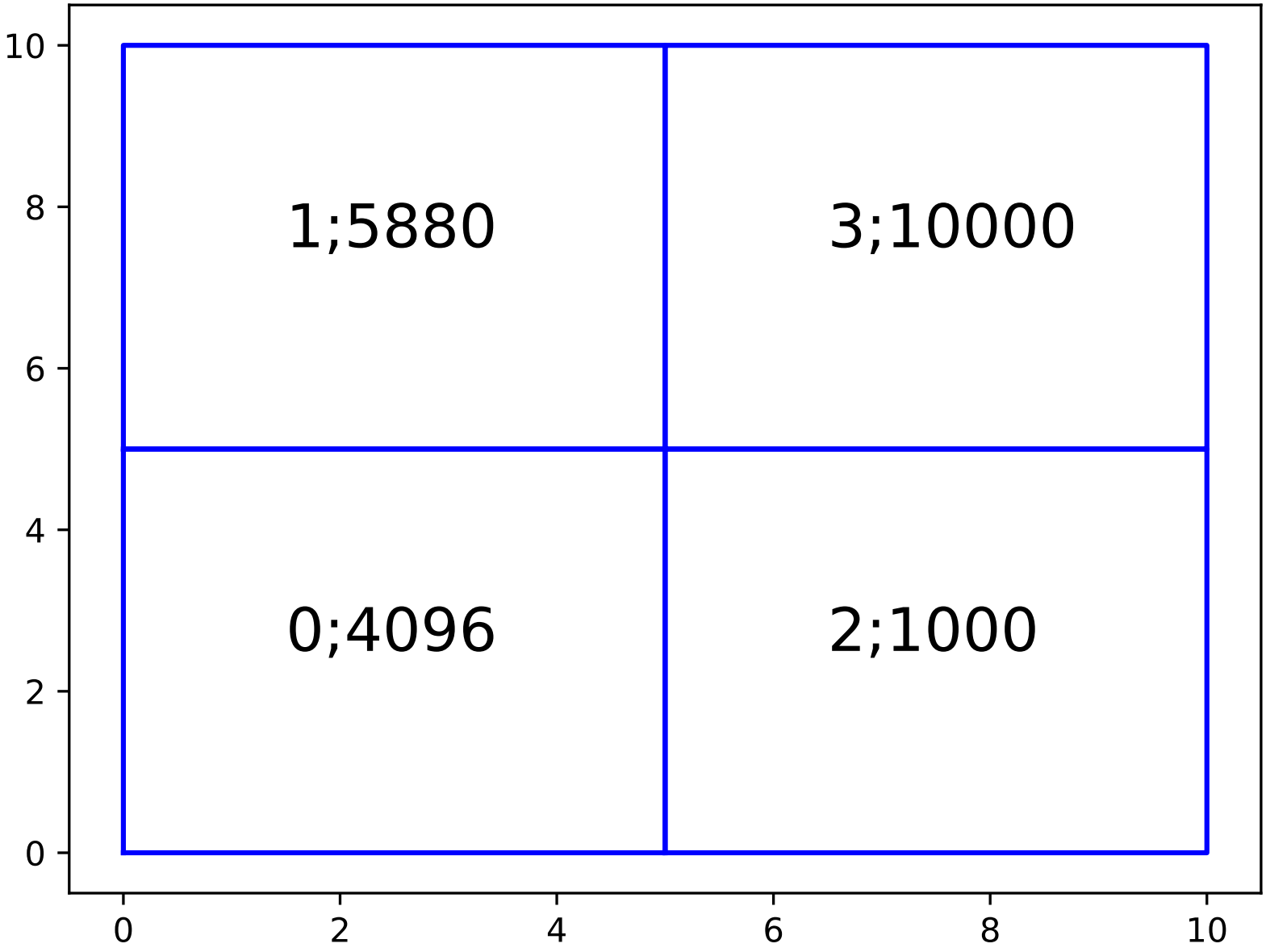
- G is the total number of DAGs.
- N is the total number of nodes per DAG.
- It is necessary to have $G - 1$ iterations so that slower paths to each node will address conflicts with each other. This will be illustrated in the test case shown.

Sweeping the Graph



- Now that all TDGs are weighted based on solve time, communication time, and delays, we can easily calculate the time it takes each octant to sweep across the graph.
- The sum of the longest path's edge weights in each graph represents the time each octant takes to complete its sweep.
- The maximum weighted edge sum of these longest paths is the estimated time to solution that we use.

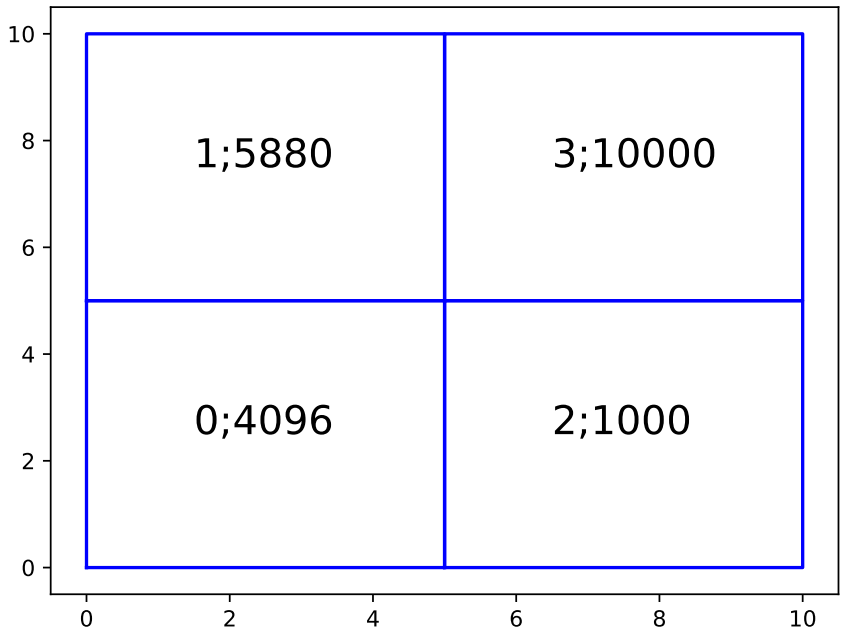
Uniform Unbalanced Test (2D)



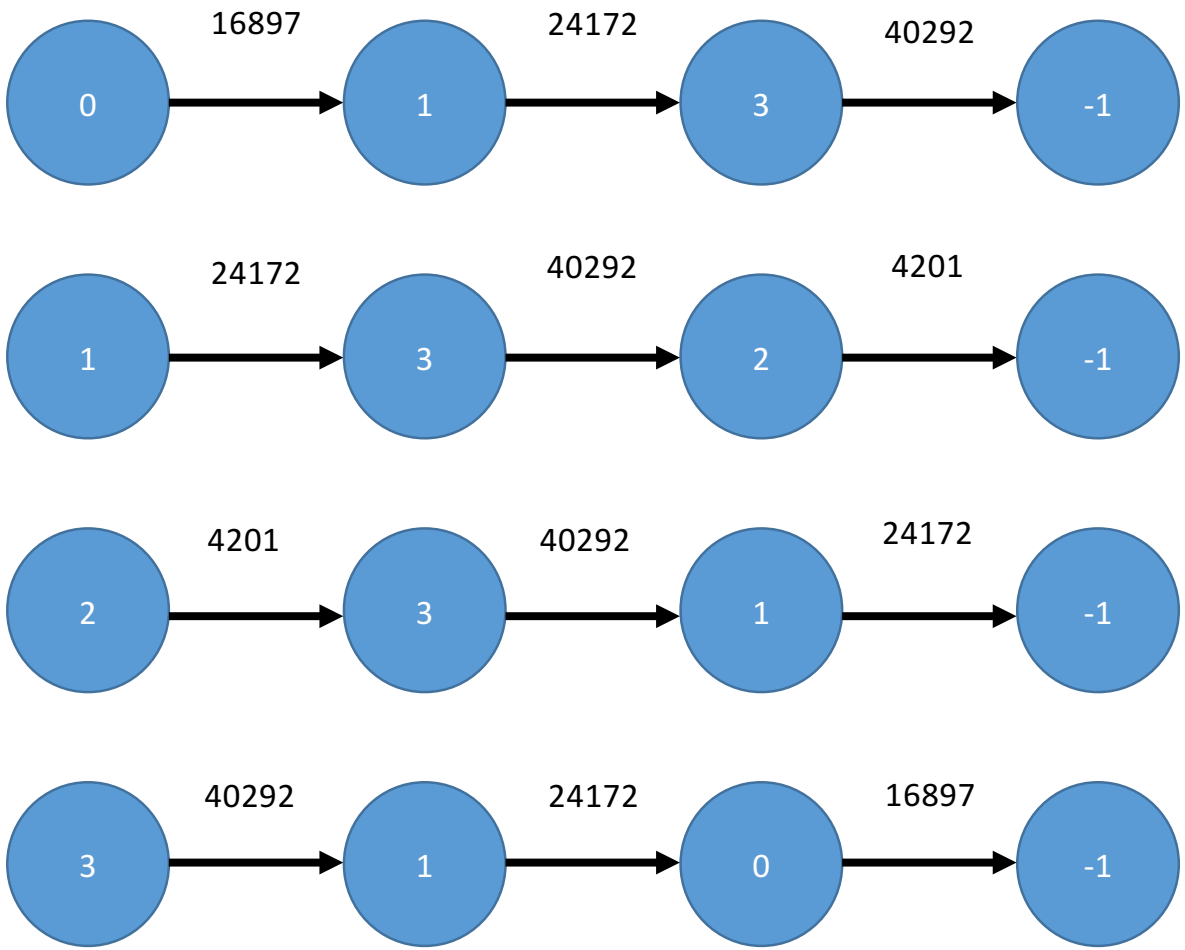
Longest Paths per Quadrant

1. 0,1,3
2. 1,3,2
3. 2,3,1
4. 3,1,0

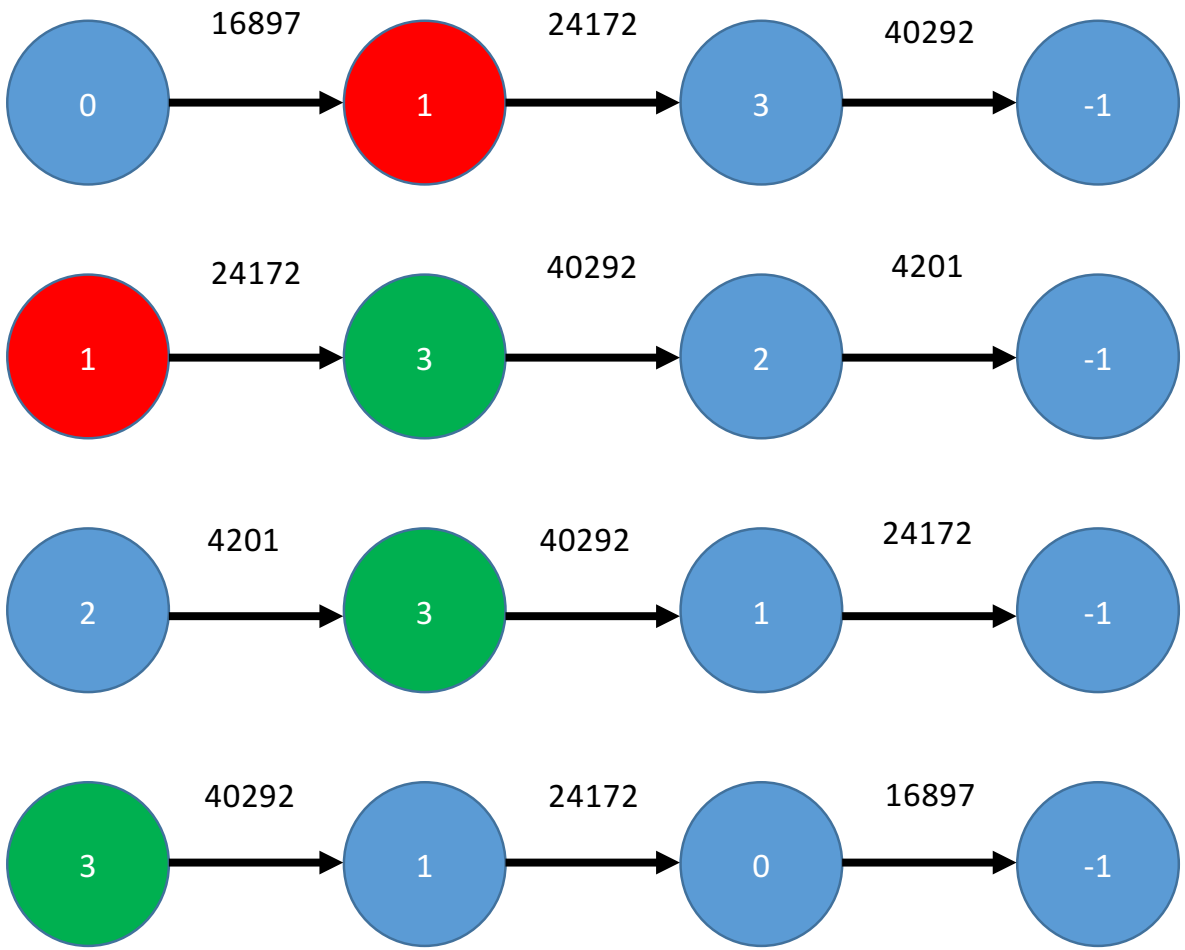
On future slides you will see a dummy node with the label "-1".



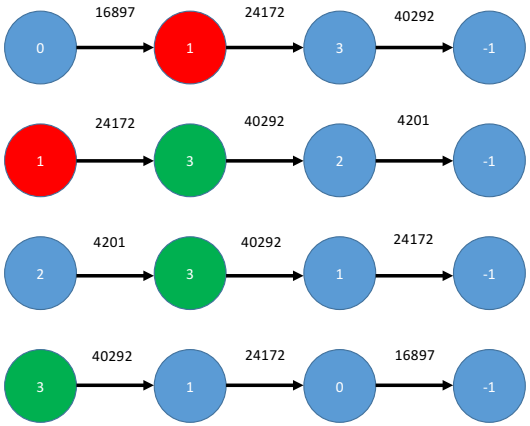
Graph Weights with Solve and Comm. Time Only



Conflicts at Iteration 0

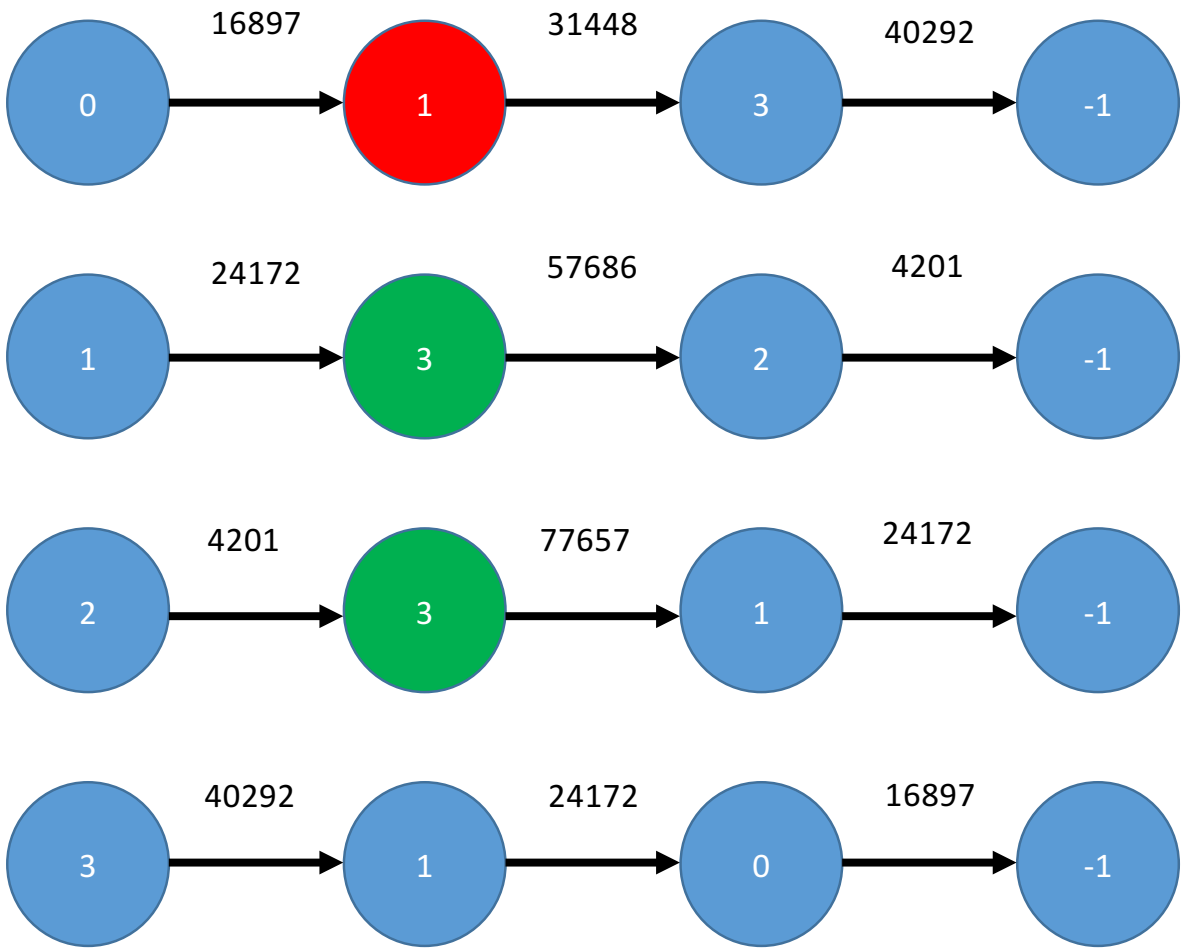


Resolving Iteration 0

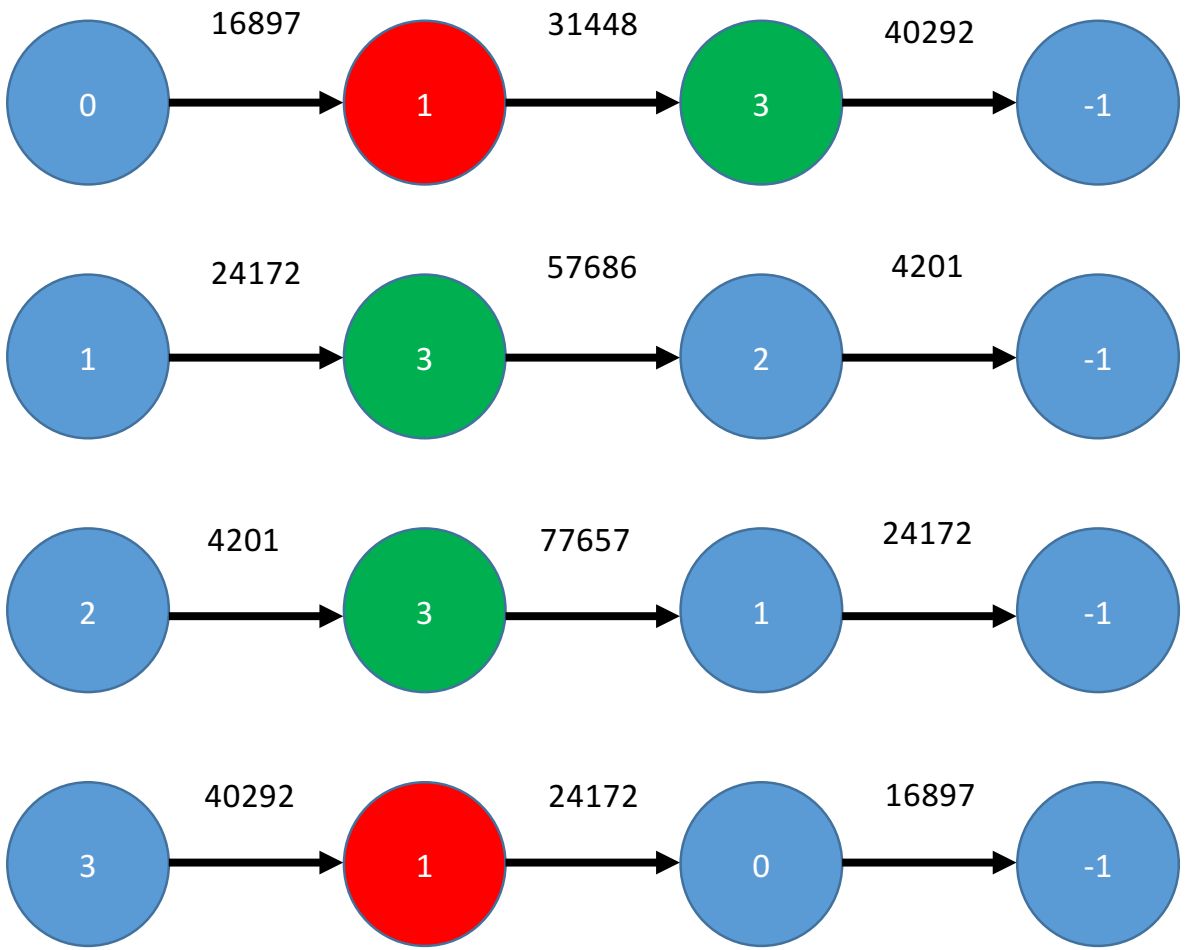


- There are conflicts at Node 0 or Node 2. The fastest paths finish by the time the slower paths arrive.
- Path 1 is still solving Node 1 when Path 0 arrives. Delay of $24172 - 16897 = 7275$ added to Path 0.
- Paths 1 and 2 arrive at Node 3 before Path 3 finishes solving. Delays of $40929 - 24172 = 16757$ and $40929 - 4201 = 36728$ are added.

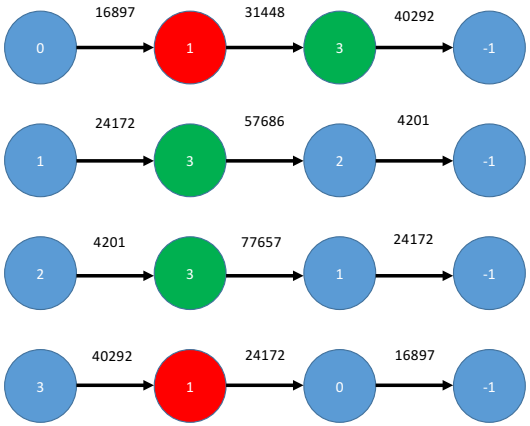
Iteration 1


16897
31448
40292
0
1
3
-1
24172
57686
4201
1
3
2
-1
4201
77657
24172
2
3
1
-1
40292
24172
16897
3
1
0
-1

Conflicts at Iteration 1

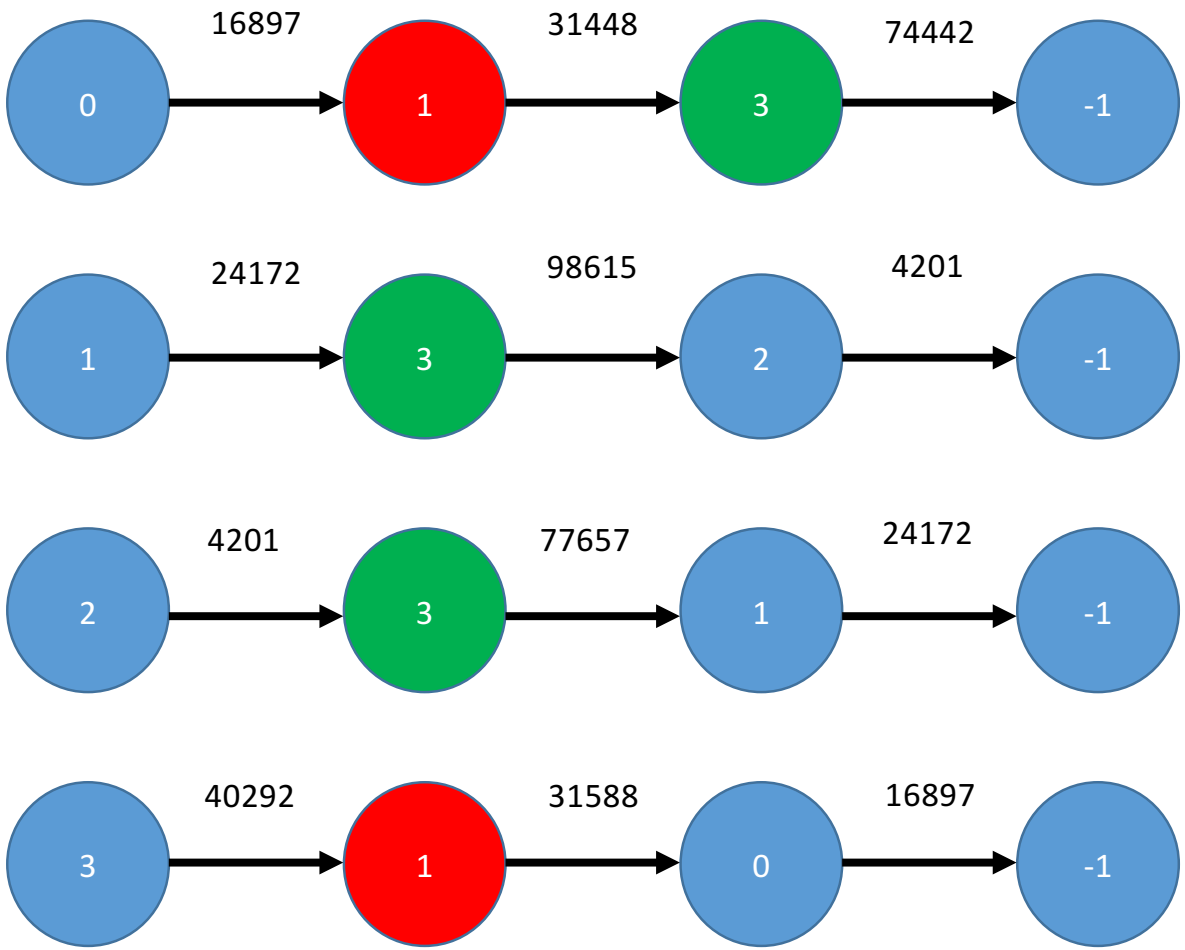


Resolving Iteration 1

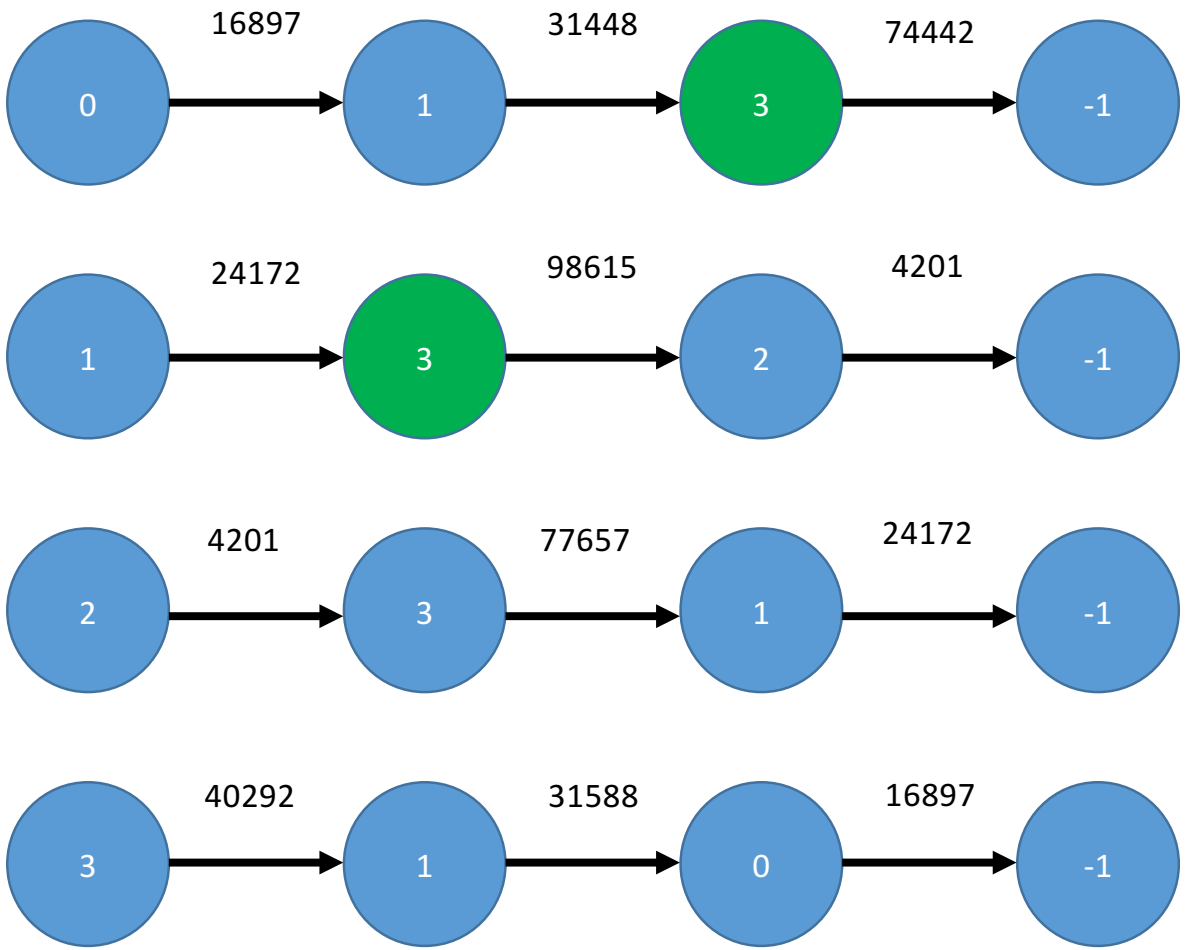


- Still no conflicts at Node 0 or Node 2.
- Path 1 is still solving Node 1 when Path 3 arrives. Delay of $48345 - 40292 = 7416$ is added.
- Path 2 starts solving Node 3 at t=40929. Path 1 is delayed a further 40929 (Node 3's solve time).
- Path 0 can solve Node 3 at t=48345. Path 2 is not finished solving until t=81858. Delay of $81858 - 48345 = 33513$ added.

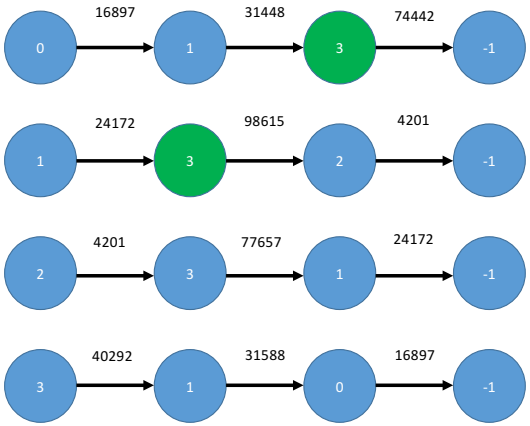
Iteration 2


16897
31448
74442
0
1
3
-1
24172
98615
4201
1
3
2
-1
4201
77657
24172
2
3
1
-1
40292
31588
16897
3
1
0
-1

Conflicts at Iteration 2

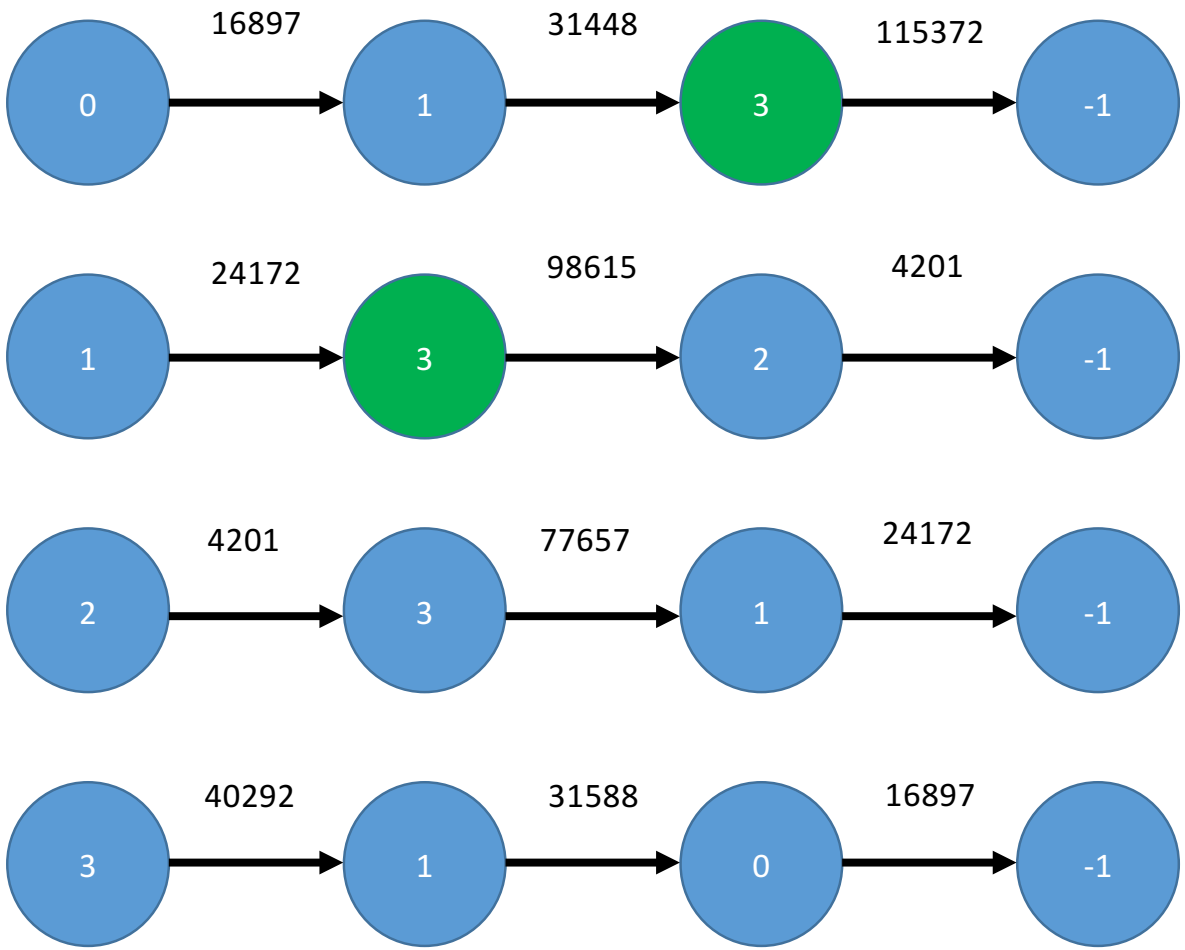


Resolving Iteration 2

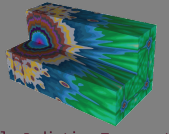


- No conflicts at Nodes 0,1, or 2.
- Path 1 is done solving Node 3 at t=122787. Path 2 is delayed a further 40929 (Node 3's solve time).

Final Weights

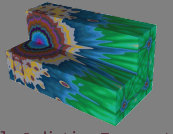


Current Setbacks



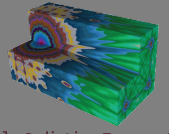
- For perfectly balanced problems, every simple path is the longest path.
- Each octant should finish sweeping in the same amount of time.
- Current conflict resolution attempts achieve the correct answer for small problems (4-9 subsets), but as the number of paths grows, the octants sweep times no longer match.
- Correctly estimating sweep time for perfectly balanced problems is an immediate priority.

Immediate Priorities



- Correctly estimate sweep time for perfectly balanced partitions.
- Add options for angular and spatial aggregation.
- Match PDT's results for scaling on perfectly balanced partitions.

Goals of the Dissertation



- Verify time-to-solution estimator matches PDT's sweep time with equivalent aggregation parameters and scheduling.
- Use time-to-solution estimator to optimize cut line locations by using it as a cost function in a Python optimization library (scipy.minimize, scipy.optimize, etc.).
- Build test problems, in 2D and 3D, that balance approximately perfectly via both load balancing algorithms.
- Using these test problems, show that optimizing partitions with the proposed approach yields a better runtime.
- Implement these tests in PDT and show that the optimized partitions yield a better runtime than perfectly load balanced problems.
- Explore adding the threads per processor as a variable to our parameter space during optimization in addition to cut lines.