# Julia Multithreading

Multi processing, no real support for native threads means that in order for parallelisation to occur effectively, we’re looking at large grain task granularity.

Communication costs very high.  
Very large overhead

# Possible Algorithms

## Prims (Minimum spanning tree)

Parallelize the greedy step of finding the next best edge. (Every iteration can be done in parallel but the iterations themselves have to be done in a sequential manner). May be slower for some cases as the task creation overhead may outweigh the speed benefits?

## Dijkstra

Same deal with Prims, the path building has to be sequential however the consideration of all possible nodes can be done in parallel.

## All pairs shortest path

Naive solution: Share the starting nodes between processors, then each processor will run the dijkstra single path algorithm to completion>  
Pros of this approach are that it is simple.

Cons are this approach scales really badly.

Slightly more exploitation of parallelism is to split the processors into n partitions where n is the number of nodes in the graph. Then each partition of n/p processors will each conduct a single source dijkstra algorithm run.

<https://julialang.org/blog/2019/02/light-graphs> Very useful