Group 14: Parallelisation of graph algorithms in Julia

Speaking Order:

1. Blain Cribb
2. Aorthi Afroza
3. Kelsey Murray

Overview

Blain Cribb:

- Julia

Aorthi Afroza:

- Considered algorithms
- Prim's algorithm

Kelsey Murray:

- A* Algorithm
- Julia benchmarking tools

Julia

- High-level language
- General-purpose
  - Data science
  - Machine learning
  - Fast numerical and scientific computing
  - Parallel computing
- Dynamically typed but with great performance
- Open-sourced with fantastic documentation
- “Looks like Python, feels like lisp, runs like Fortran”

Julia - Looks like Python

- Code is easy to read and write
- Designed to be easy to pick up for other language users
- Plenty of modern features that make coding easier:
    - Large base library
    - Garbage collection
    - Optional data typing
    - Interoperability

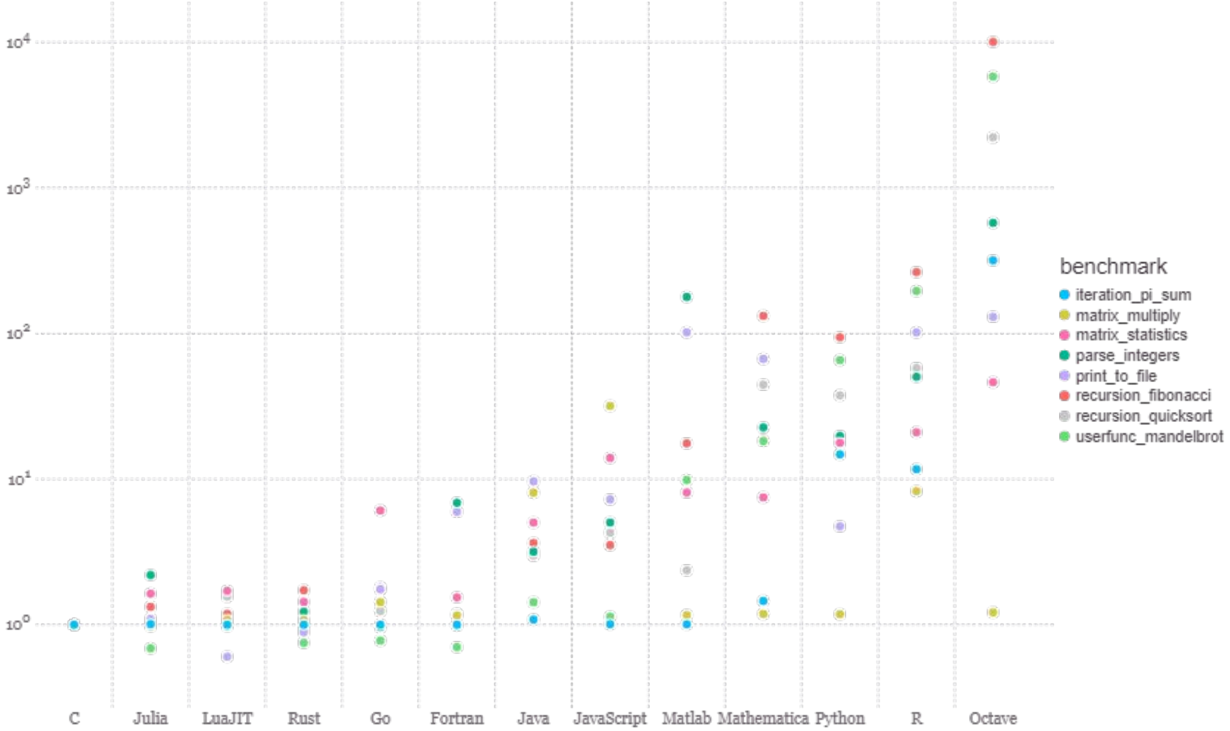
Julia - Feels like Lisp

- Support for metaprogramming
  - Code is stored as a data structure within the language
  - This code can be manipulated during runtime
  - Very useful for applications such as machine learning
- True Lisp-style macros unlike static macros
- Multiple dispatch for OOP like behaviour

Julia - Runs like Fortran

- Created to be fast from its inception
- Static analysis and just in time compilation allows for running of some code in a static manner
- Optional static typing means with smart programming we can achieve static-like speeds
- Multiple dispatch can help with the static analysis

Julia - Runs like Fortran



https://julialang.org/benchmarks/

Julia - Parallelism

- Natively supported
- Multithreading (experimental)
- Loops are easily parallelisable through use of macros
- Supports atomic access for variables

Considered Algorithms

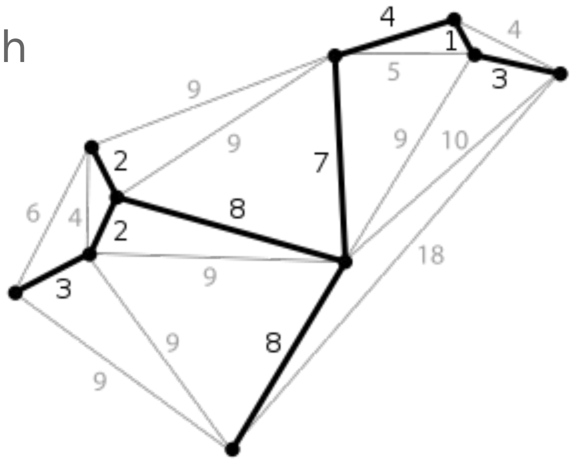
- Criteria for algorithms:
  - Different algorithm types; e.g. shortest path, traversal, minimum spanning tree
  - Algorithms with a considerable amount of literature out there
  - Successful existing implementations of their parallelised algorithms

Julia - LightGraphs.jl

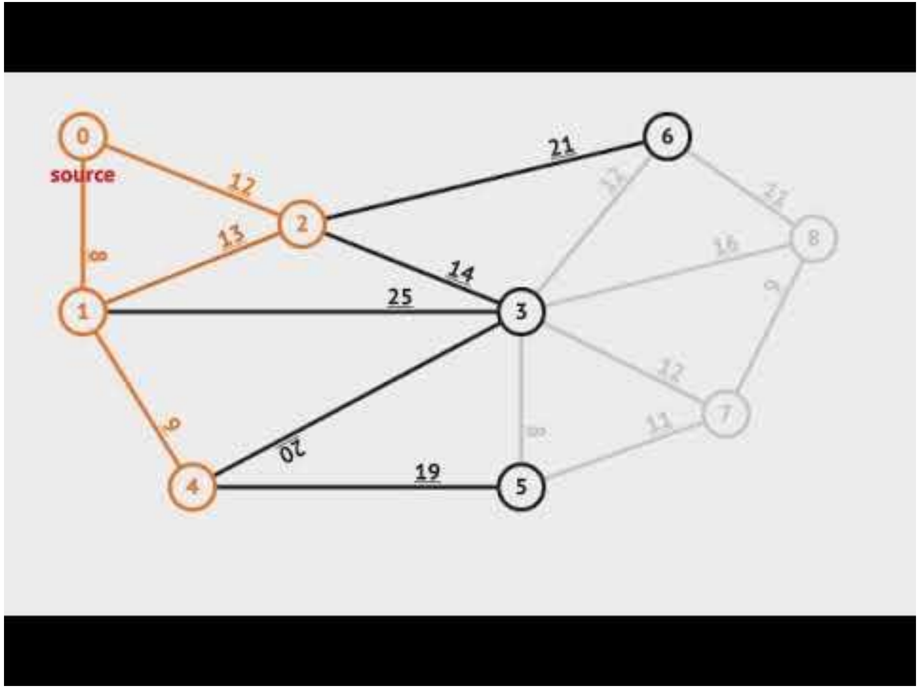
- A Julia Package that provides the framework for:
    - Building graphs
    - Traversing them
    - Building your own graphing algorithms
- Contains the module Parallel, which contains parallel implementations of:
    - Bellman Ford Shortest Paths
    - Dijkstra Shortest Paths
    - Floyd Warshall Shortest Paths
    - Johnson Shortest Paths
    - Bfs
    - Greedy Color
    - 5 Centrality measures
    - 4 Distance measures

Prim's Algorithm

- Minimum spanning tree (MST)
  - A tree that connects all nodes in the graph
  - The **least total cost** among all trees that connect all the nodes
  - This may not be the shortest path

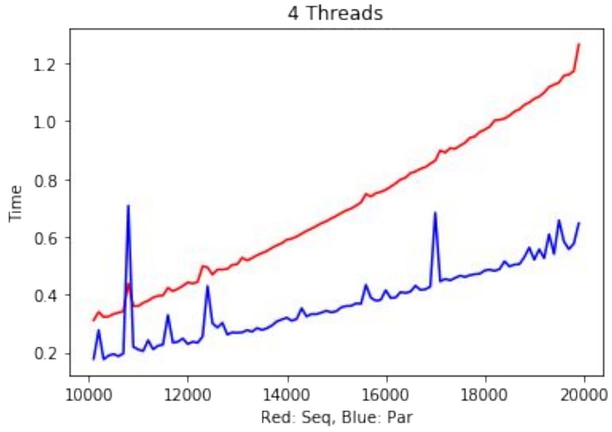


Prim’s Algorithm: Example



Previous Attempts at Parallelising Prim's

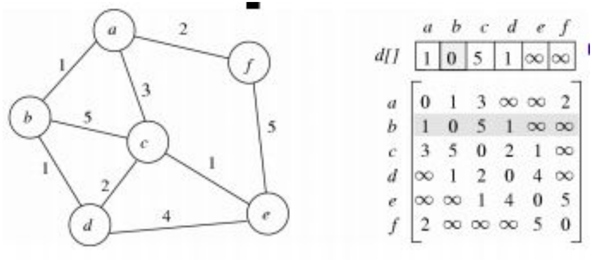
- PES University in India
- University of California, Irvine
  - Implemented in OpenMP

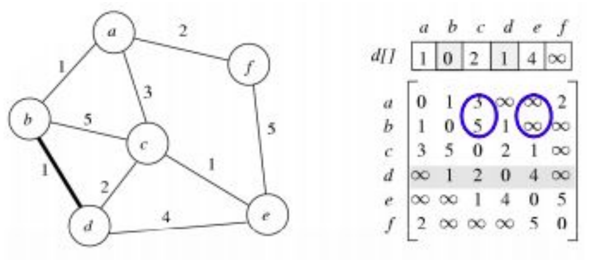


https://github.com/parthvshah/parallel-prims

- University of Singapore
  - Implemented in C++
  - This paper proposes an adaptation of Prim's Algorithm

Implementation Plan: Prim's





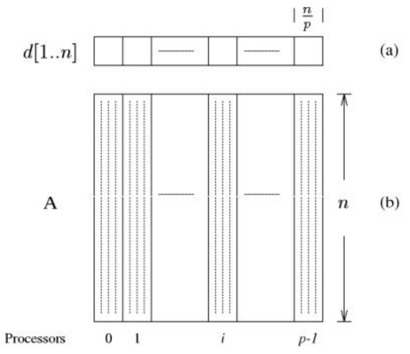
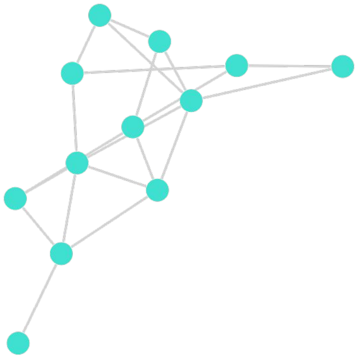
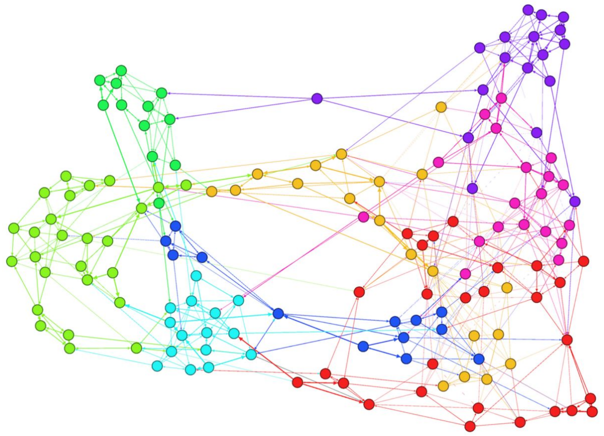


Figure 7.6 The partitioning of the distance array d and the adjacency matrix A among p processors. Copyright (r) 1994 Benjamin/Cummings Publishing Co.

Testing and Benchmarking: Prim's

- Datasets:
  - Densely connected, large, undirected graphs
- Metrics
  - Speed-up
  - Correctness

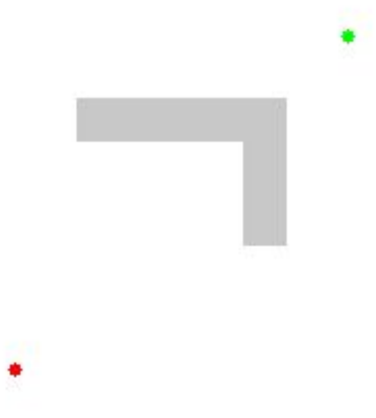




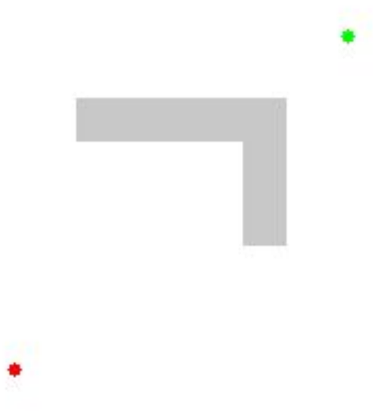
A* Algorithm

- **Pathfinding algorithm**
  - Popular in gaming where there's one source, one destination, and possibly obstacles
  - "Informed" Dijkstra's algorithm
  - Cost of a considered path takes into account heuristic as well as edge weight
- **Best-first search, but not greedy**
  - Doesn't solely rely on what seems best at that time (the heuristic) but also includes the exact cost of the potential path (the edge weights)

A* Algorithm: Example



Dijkstra's Algorithm



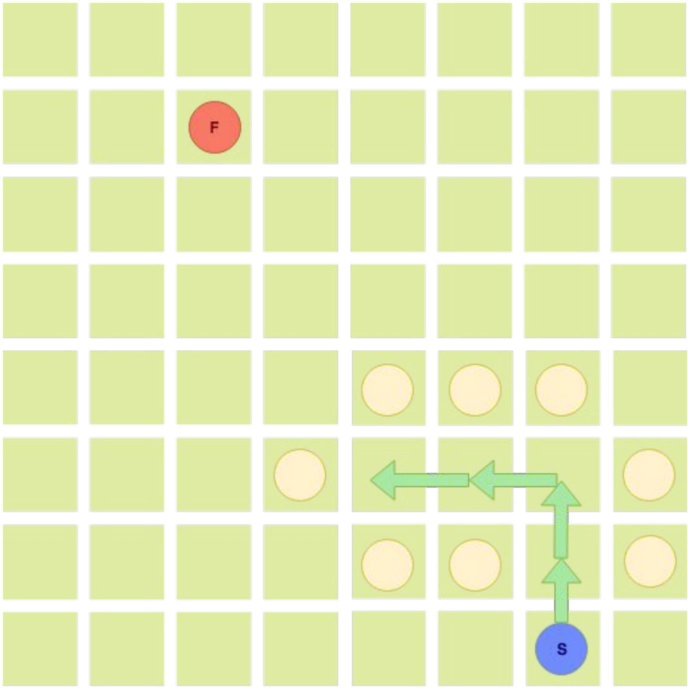
A* Algorithm

https://en.wikipedia.org/wiki/A*_search_algorithm#/media/File:Astar_progress_animation.gif
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstras_progress_animation.gif

Parallel A* in Literature

First Approach:

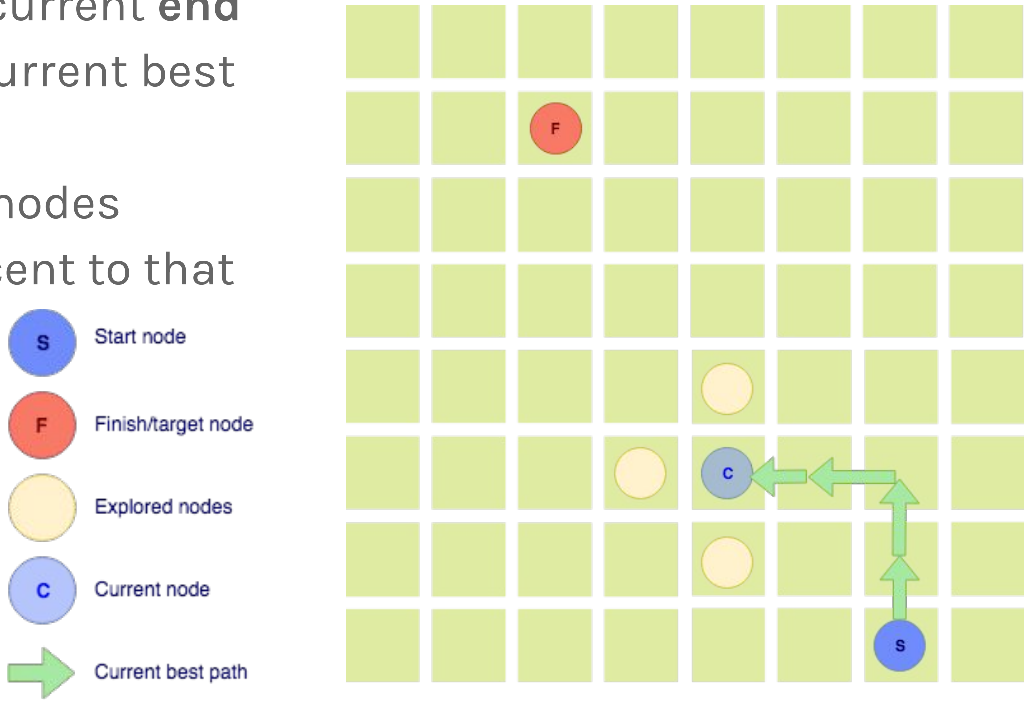
- Looks at entire current best **path**
- Investigates all the nodes surrounding the **path**



Parallel A* in Literature

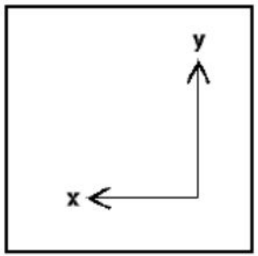
Second Approach:

- Looks at the current **end node** of the current best path
- Investigates nodes directly adjacent to that **end node**

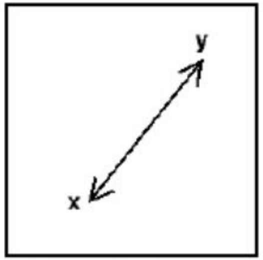


Implementation Plan: A*

- Can we move **diagonally**? Will this make a difference to the speedup?
    - If we can move diagonally: **Euclidean** distance as heuristic
    - If we can't: **Manhattan** distance as heuristic



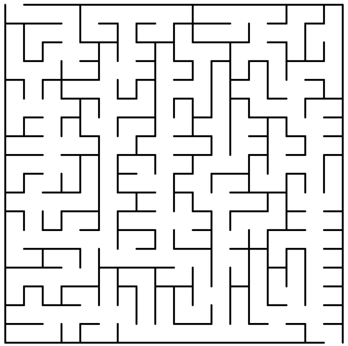
Manhattan

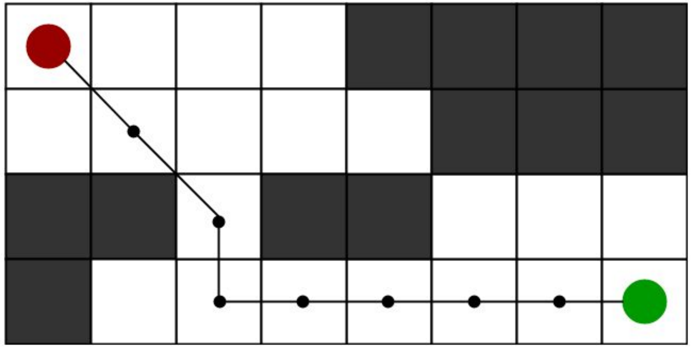


Euclidean

Testing and Benchmarking: A*

- Metrics:
  - Speedup
  - Correctness: Do we care if the **exact path** is the same?
    - Ideally parallel is **at least as good** as serial but we don't care if the path isn't exactly the same
- Datasets:
  - Large grid maps with obstacles





Julia **Benchmarking** Tools

- @time macro

Output:

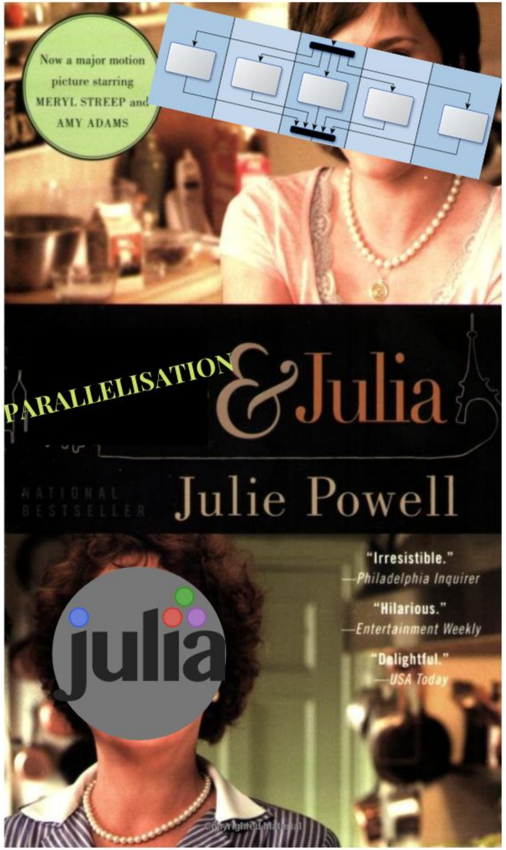
```
0.244729 seconds (294.16 k allocations: 14.614 MiB, 5.74% gc time)
```

- @benchmark macro

Output:

```
BenchmarkTools.Trial:
  memory estimate:  2.13 KiB
  allocs estimate:  19
  --------------
  minimum time:     1.770 μs (0.00% GC)
  median time:      2.170 μs (0.00% GC)
  mean time:        3.924 μs (37.21% GC)
  maximum time:     9.772 ms (99.92% GC)
  --------------
  samples:          10000
  evals/sample:     10
```

Thank you!


Now a major motion picture starring MERYL STREEP and AMY ADAMS
PARALLELISATION & Julia
NATIONAL BESTSELLER
Julie Powell
"Irresistible."
—Philadelphia Inquirer
"Hilarious."
—Entertainment Weekly
"Delightful."
—USA Today
julia