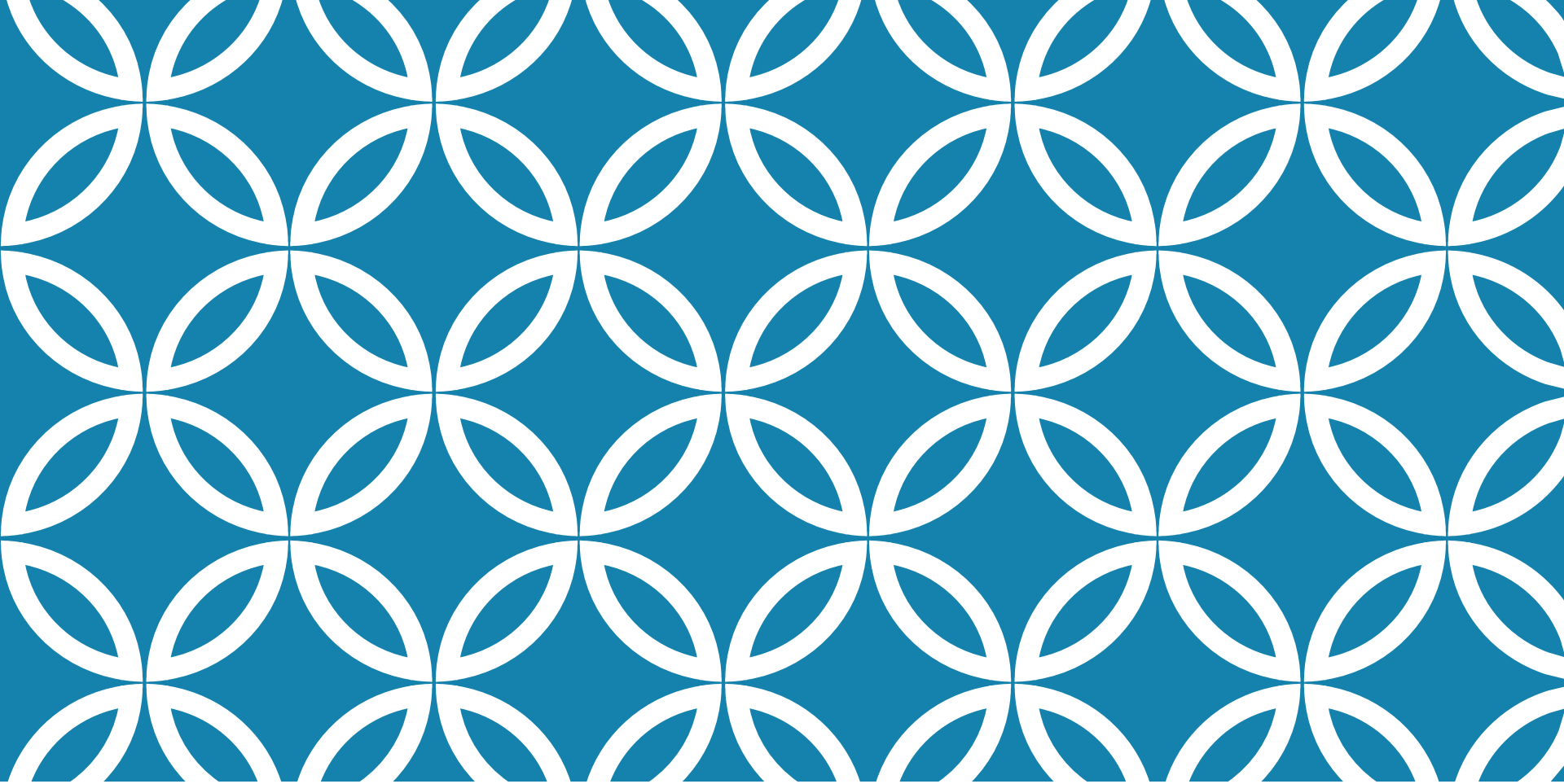


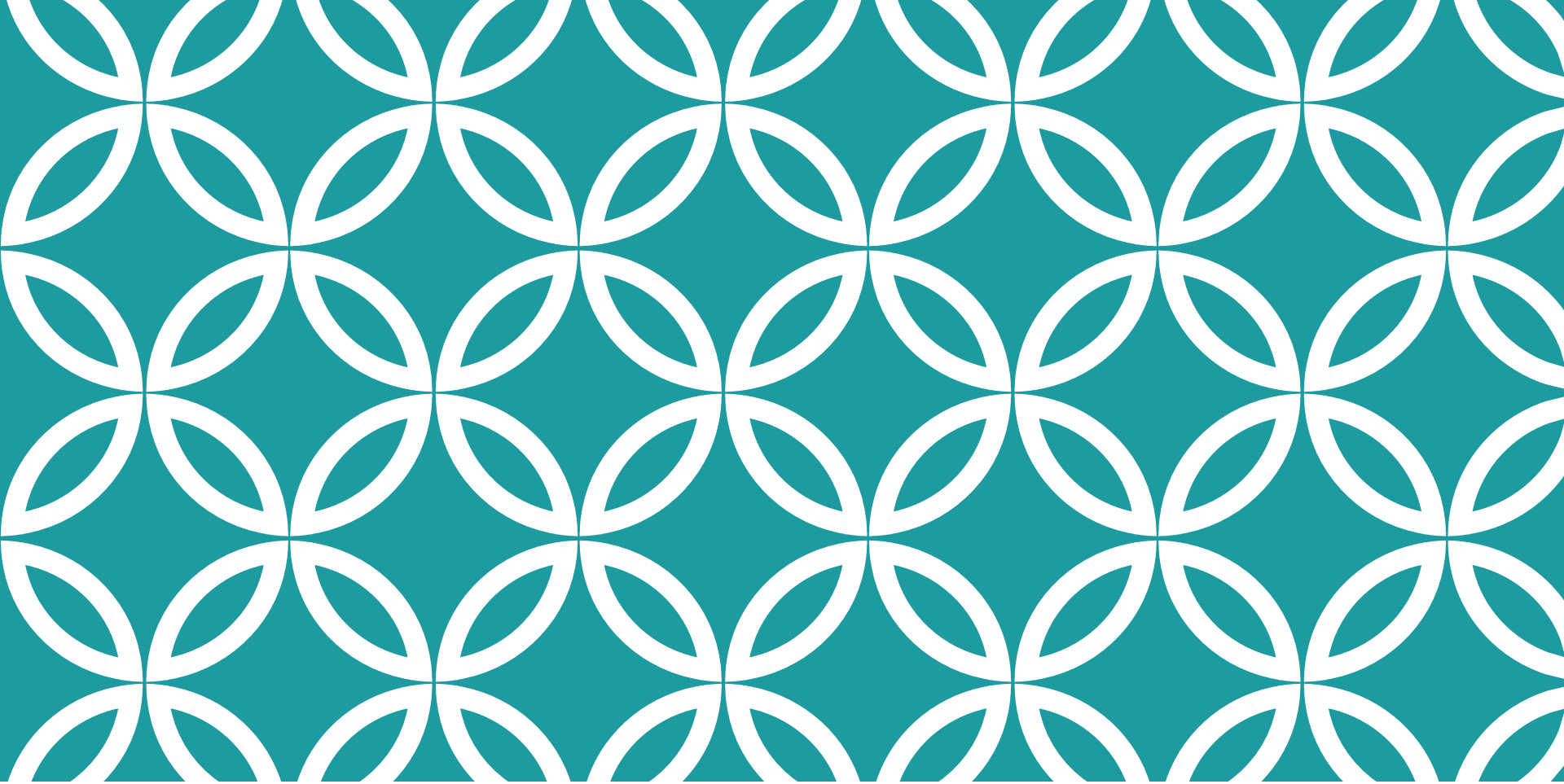
PARALLELIZING NETWORK MOTIFS

Matt Kipps

Fall 2014

OVERVIEW

1. Parallelize network motif finding.
2. Compare MASS and MPI.



UNDERSTANDING THE PROBLEM

QUESTION

Can we parallelize network motif finding?

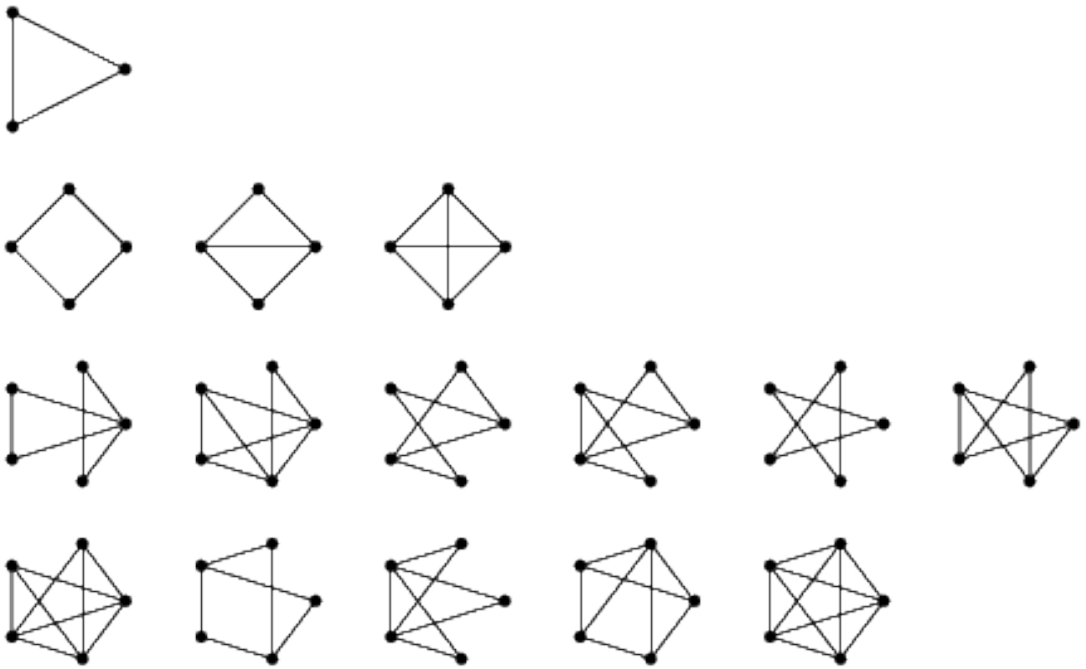
UNDERSTANDING THE PROBLEM

- What are network motifs?
- Why are we trying to find a parallel solution?

NETWORK MOTIFS

Recurring subgraph isomorphs within a larger network

- statistically significant compared to random networks



NETWORK MOTIFS

…why are we trying to find motifs?

One of the applications is biology.

Network motifs might provide insight to connections between the molecular level and system level of biological systems.

NETWORK MOTIFS

Network motifs can be determined using a 3-step process

1. Find all subgraphs within a target network
2. Aggregate subgraphs into subgraph groups (isomorphs)
3. Compare frequency of subgraph groups to random networks

NETWORK MOTIFS

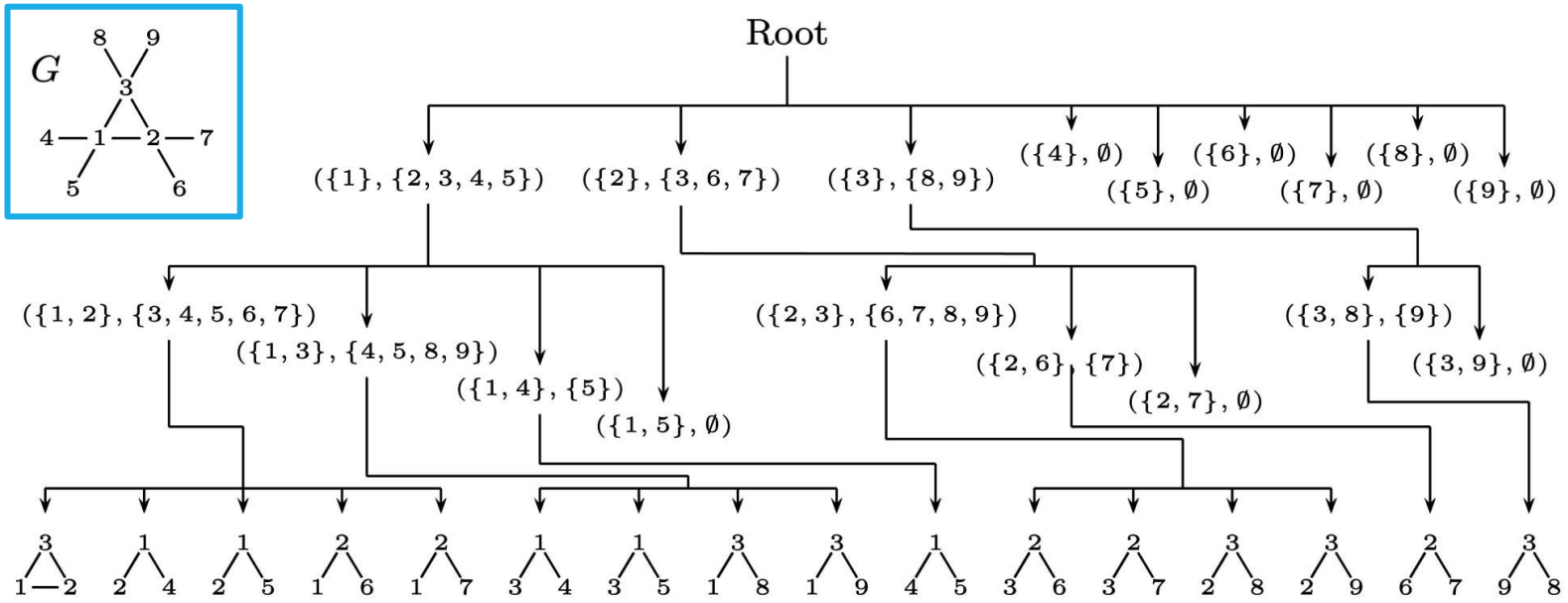
We are focused primarily on parallelizing Step 1.

- Step 2 is accomplished relatively quickly with McKay's *nauty* algorithm
- Step 3, random network evaluation, is beyond the scope of this project

NETWORK MOTIF STEP 1

Find all subgraphs within a target network.

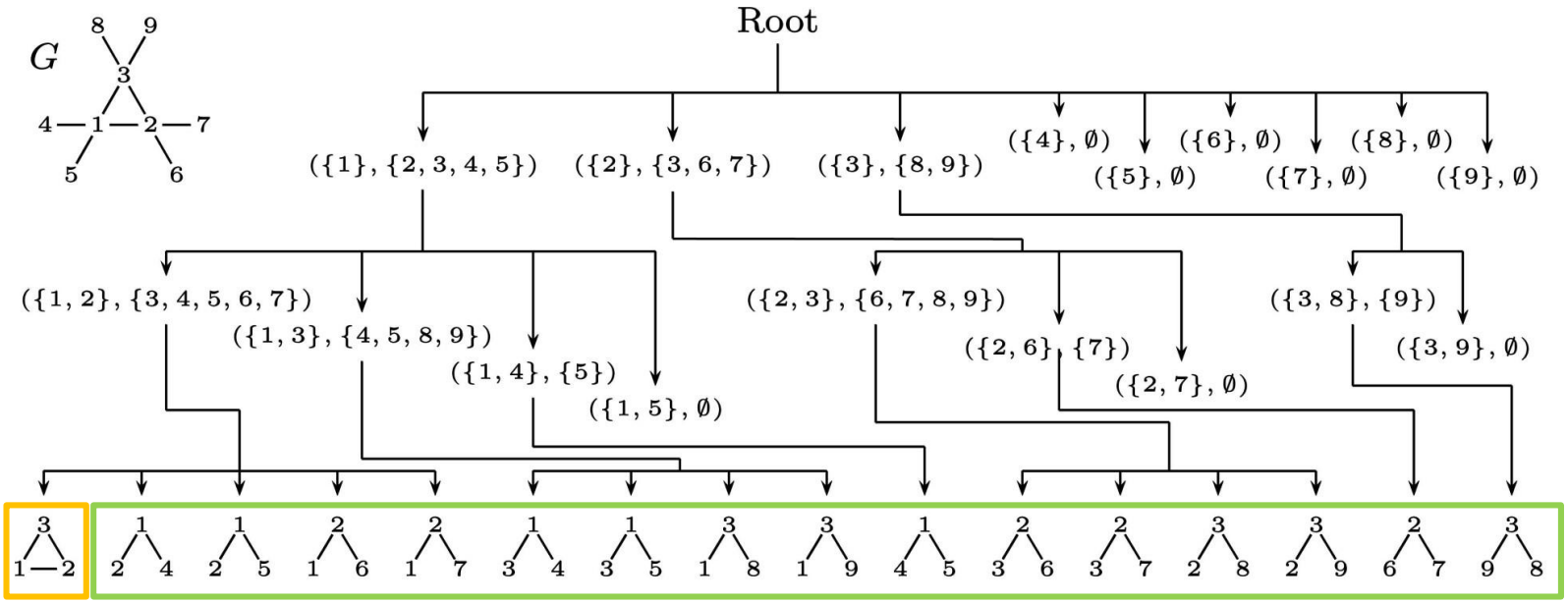
We will use the graph traversal algorithm **E**numerate **SU**bgraphs (**ESU**).



NETWORK MOTIF STEP 2

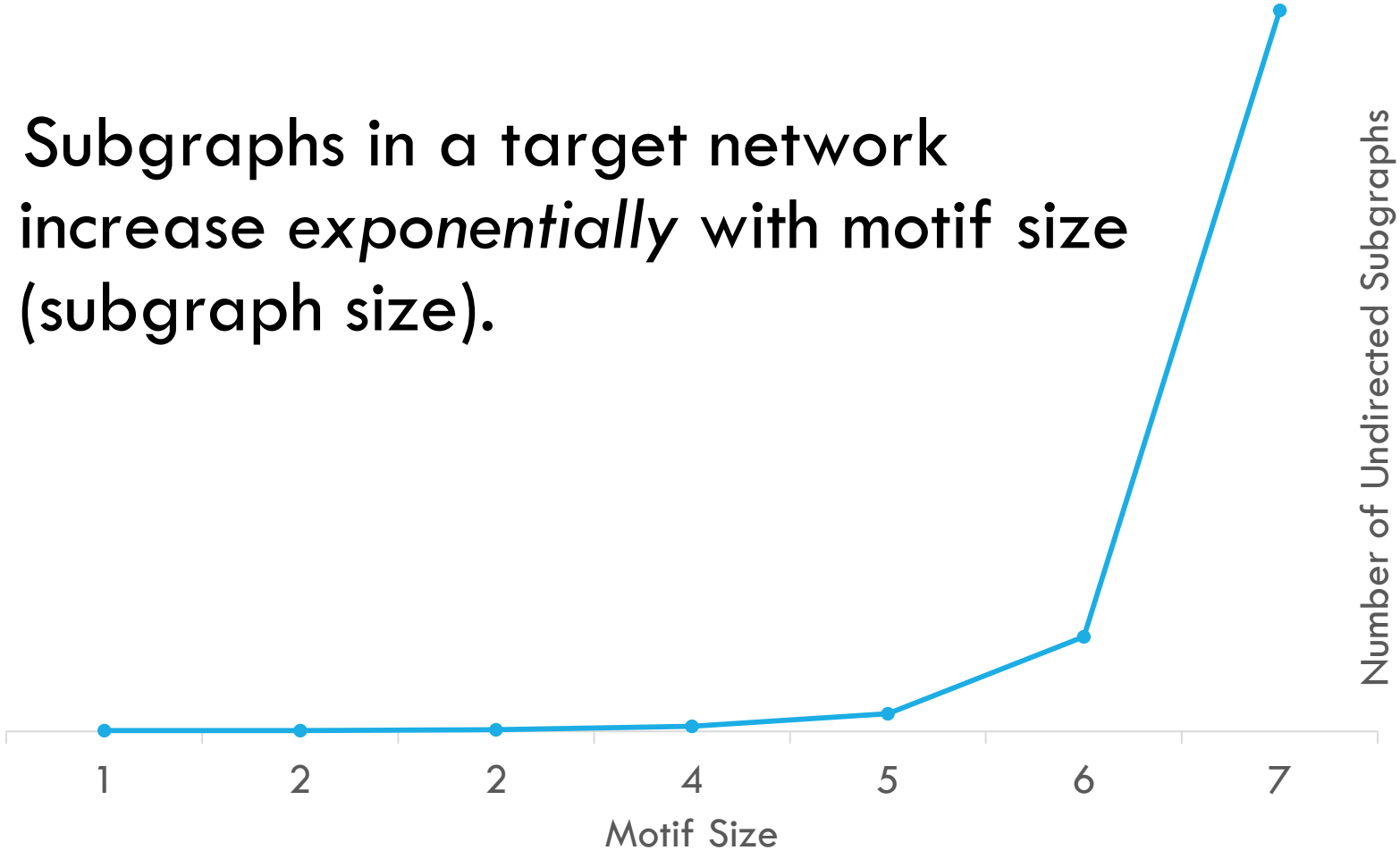
Aggregate subgraphs into subgraph groups (isomorphs).

This is accomplished using the `labelg` program.



PARALLELIZATION MOTIVATION

Subgraphs in a target network increase *exponentially* with motif size (subgraph size).



PARALLELIZATION MOTIVATION

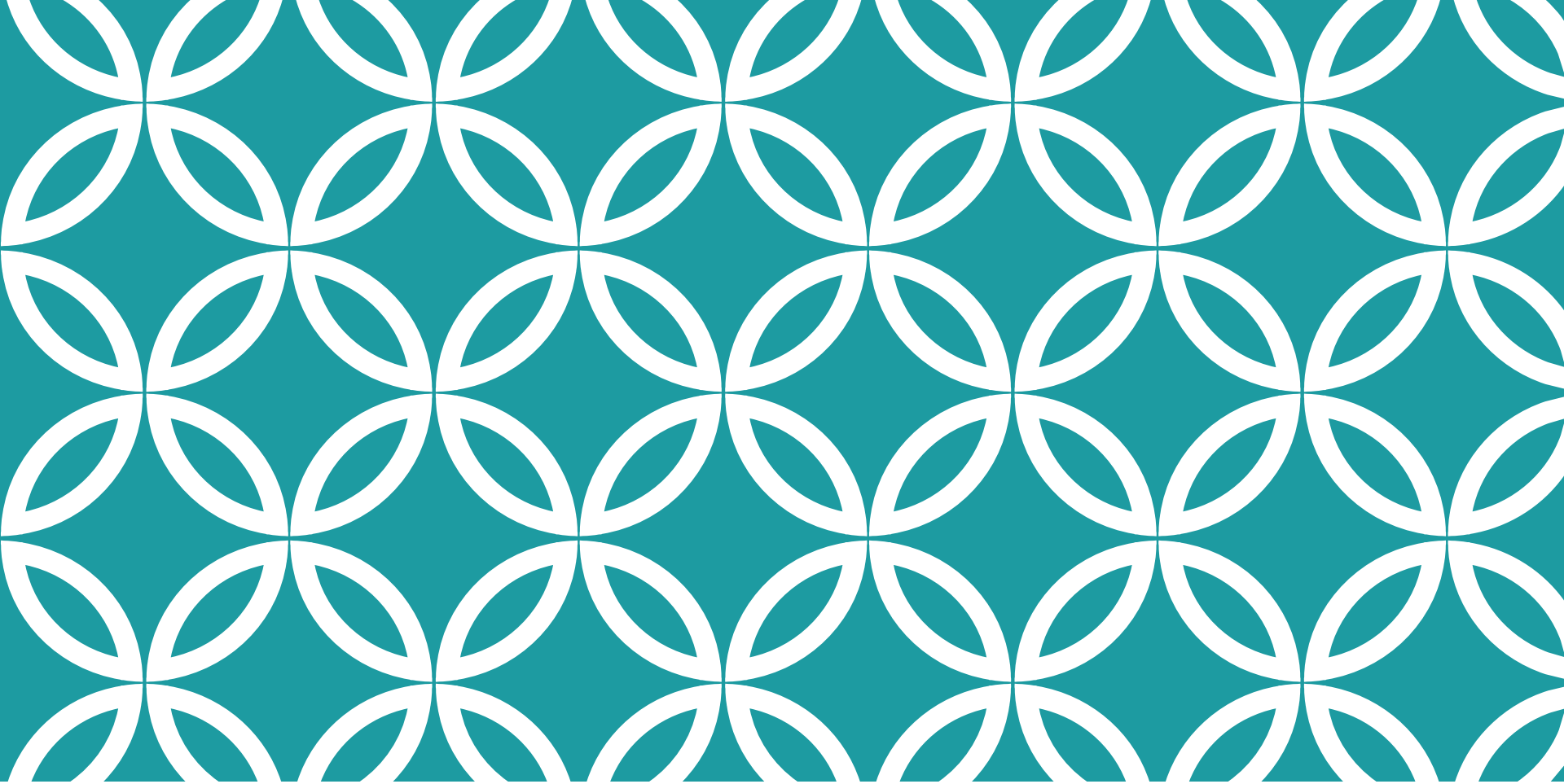
Does it really matter? Examine performance of a real biological data set (5,000 node network)…

Sequential implementation, modern CPU

Motif Size	Execution Time (sec)
2	0.629
3	1.214
4	21.619
5	1522.486
6	180000.000
7	?

PARALLELIZATION MOTIVATION

For the size of our data sets, finding network motifs using a complete enumeration algorithm like ESU is unfeasible sequentially.



SOLVING THE PROBLEM

SOLVING THE PROBLEM

1. Implement network motif finding
2. Design and implement parallel programs using various tools/approaches

SOLVING THE PROBLEM

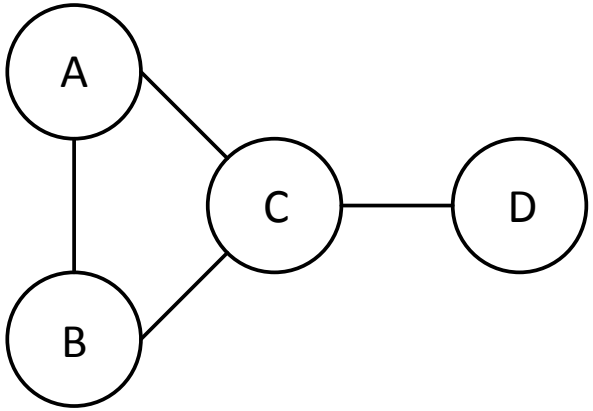
1. Implement network motif finding
   a. Parse input data
   b. Find all subgraphs
   c. Get corresponding isomorphs

PARSER

Graphs are represented with a simple text-based input file.

The parser must turn this into a graph object.

```
Node_A   Node_B
Node_B   Node_C
Node_A   Node_C
Node_C   Node_D
```



SUBGRAPH FINDING

Find all subgraphs. This can be done by implementing the ESU algorithm.

SUBGRAPH FINDING

Algorithm: ENUMERATESUBGRAPHS(G, k) (ESU)
Input: A graph $G = (V, E)$ and an integer $1 \leq k \leq |V|$.
Output: All size-k subgraphs in G.

01 **for** each vertex $v \in V$ **do**
02 $\quad V_{Extension} \leftarrow \{u \in N(\{v\}) : u > v\}$
03 $\quad$ **call** EXTENDSUBGRAPH$(\{v\}, V_{Extension}, v)$
04 **return**

EXTENDSUBGRAPH$(V_{Subgraph}, V_{Extension}, v)$
E1 **if** $|V_{Subgraph}| = k$ **then output** $G[V_{Subgraph}]$ and **return**
E2 **while** $V_{Extension} \neq \emptyset$ **do**
E3 $\quad$ Remove an arbitrarily chosen vertex w from $V_{Extension}$
E4 $\quad V'_{Extension} \leftarrow V_{Extension} \cup \{u \in N_{excl}(w, V_{Subgraph}) : u > v\}$
E5 $\quad$ **call** EXTENDSUBGRAPH$(V_{Subgraph} \cup \{w\}, V'_{Extension}, v)$
E6 **return**

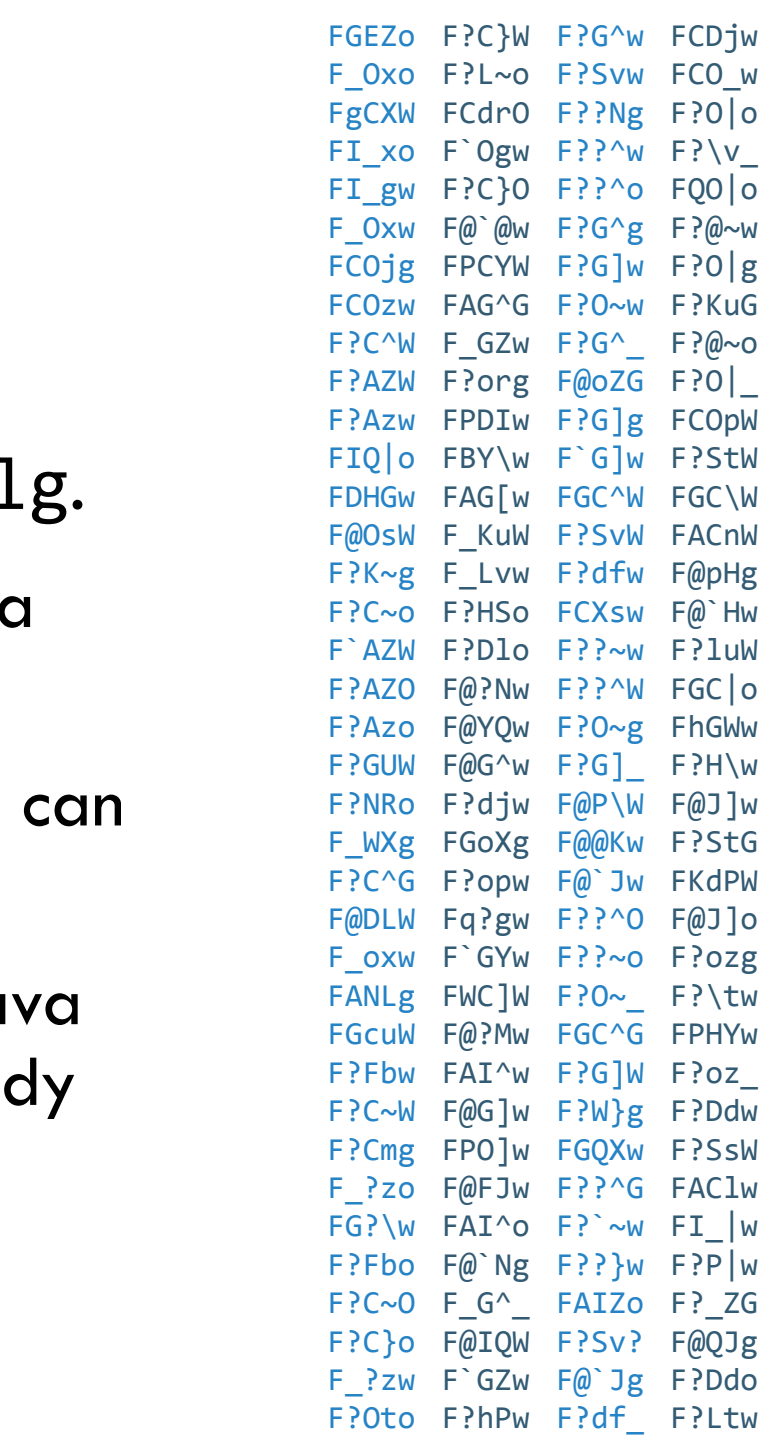
CONVERT TO ISOMORPHS

For this step, we leverage `labelg`.

The `labelg` program requires a special data format.

The good news is that the output can be read as strings.

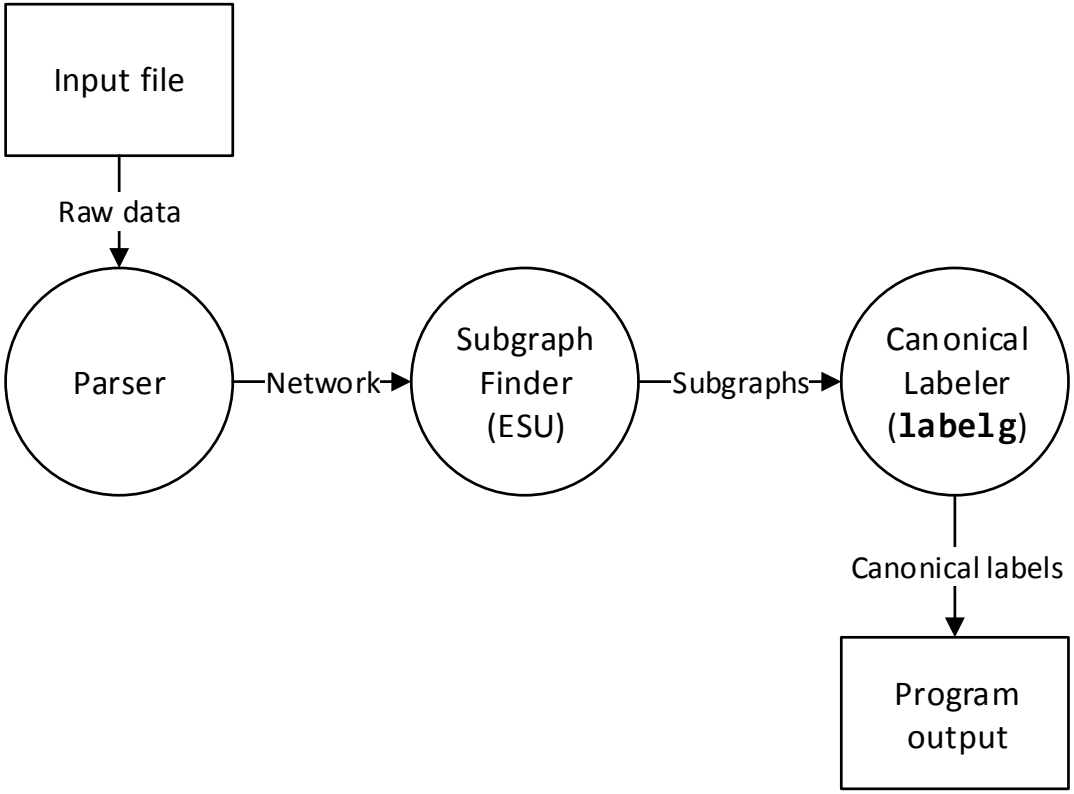
The other good news is that a Java converter for the input has already been written (thanks to Vartika Verma).

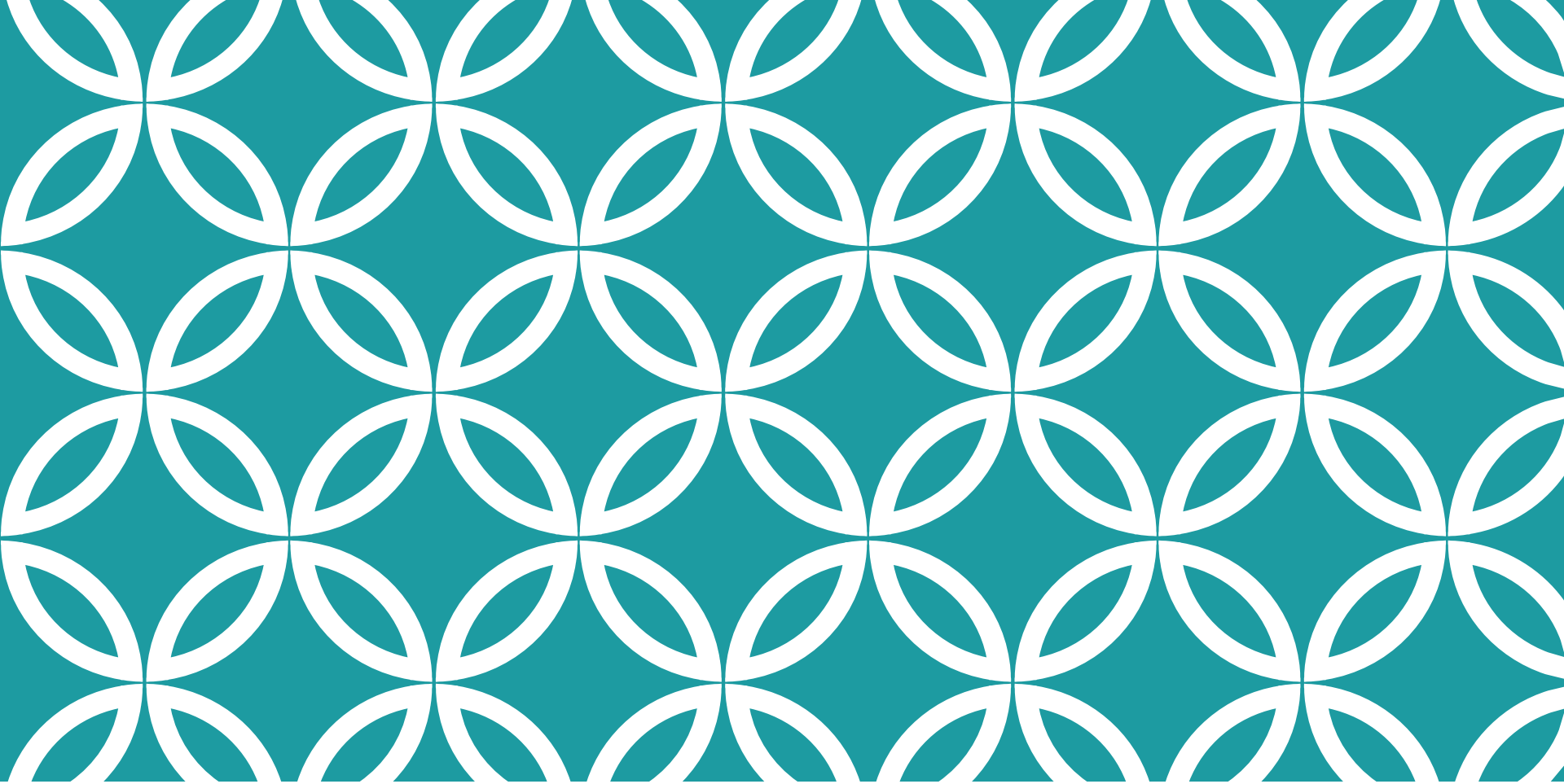


SEQUENTIAL IMPLEMENTATION

These 3 pieces comprise the bulk of the sequential network motif finder.

DATA FLOW


Input file
Raw data
Parser
Network
Subgraph Finder (ESU)
Subgraphs
Canonical Labeler (labelg)
Canonical labels
Program output



MASS AND MPI

SOLVING THE PROBLEM

2. Design and implement parallel programs using various tools/approaches
   a. MASS (simulation)
   b. MASS (hybrid)
   c. MPI

WORKING WITH MASS

MASS is an environment for parallelizing programs across a cluster.

It is being developed by Prof. Fukuda and the MASS research group here at UWB.

WORKING WITH MASS

A user defines component behavior.

Many components interact with each other over a virtual space (a simulation).

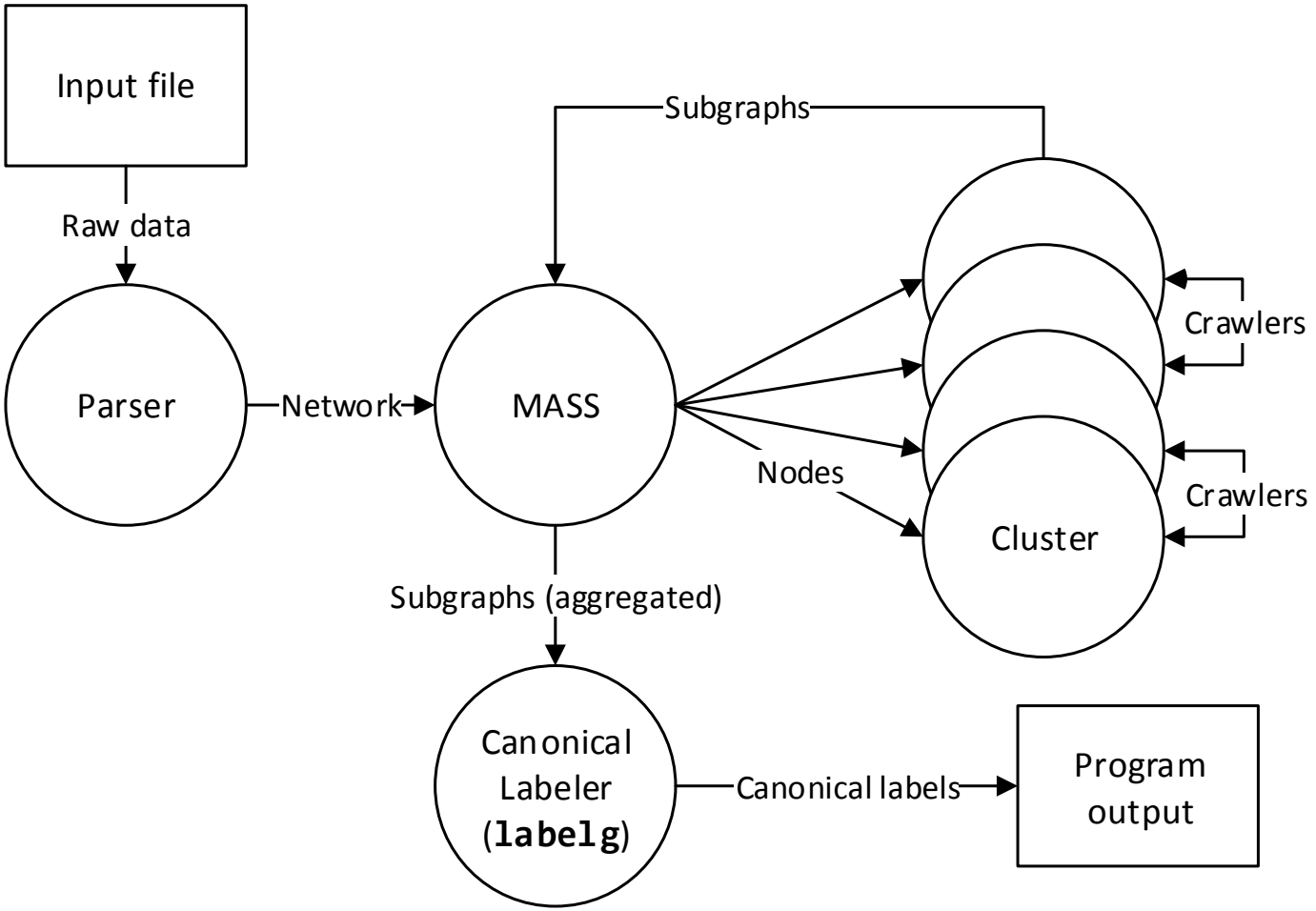
The space exists across a computing cluster.

SIMULATION DESIGN

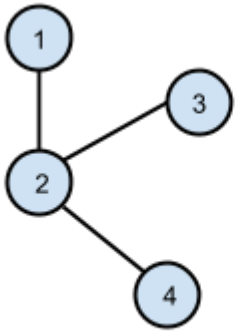
Simulation implementation – get all subgraphs by implementing "crawlers" that move according to the ESU algorithm.

- Create a graph as the virtual space
- Each crawler represents one subgraph
- When faced with a branching decision, a crawler follows one path, and clones itself to follow the other.

SIMULATION DESIGN: DATA FLOW



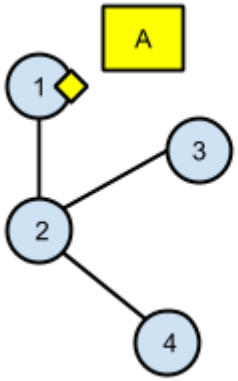
SIMULATION WALKTHROUGH



Step 0
Initial network

Agents will be searching for a subgraph motif size of **3** nodes.

(For simplicity, this sketch only shows the behavior of the first agent spawned. Another agent will also be spawned at Node 2 and will recover another subgraph not shown here)

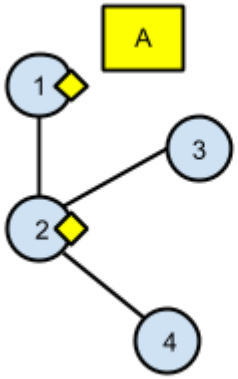


Step 1
Agent A is spawned at node 1

Agent A has a size of 1, so Agent A will keep searching.

Node 1 has only 1 branch so Agent A will not need to spawn any new Agents.

SIMULATION WALKTHROUGH

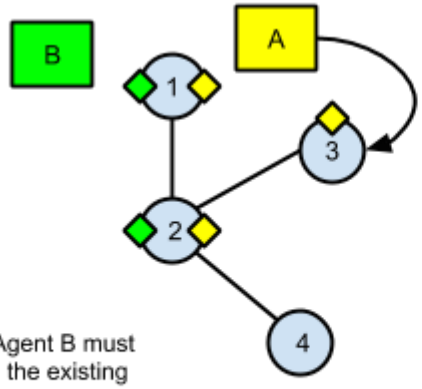


Step 2

Agent A is now at Node 2, and represents the subgraph of "1,2"

Agent A has a size of 2, so Agent A will keep searching.

Node 2 has 2 branches so Agent A will traverse one, and will create a child clone to traverse the other. This can be seen in the next step.



Note - Agent B must "inherit" the existing subgraph path "1,2" from Agent A.

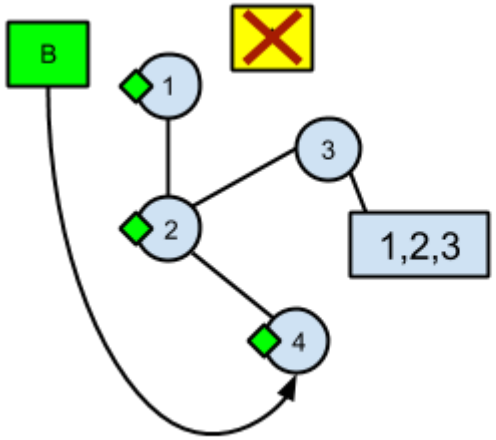
Step 3

Agent A is now at Node 3, with subgraph of "1,2,3" Agent B is spawned with subgraph of "1,2" and directions to move to Node 4

Agent A has a size of 3, so it will terminate itself and deposit the results Node 3.

Notice that Agent B is one step behind Agent A because Agent B must be spawned first and then migrate, before proceeding with the algorithm.

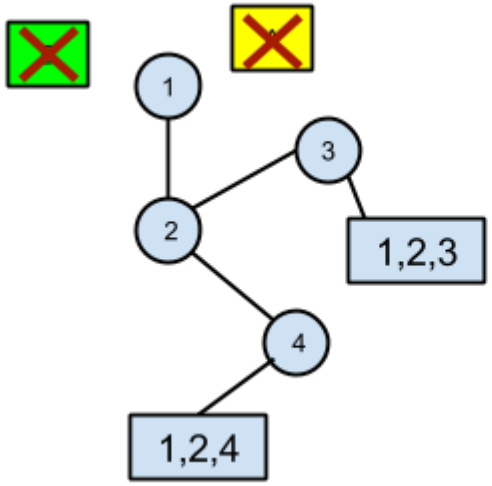
SIMULATION WALKTHROUGH



Step 4

Agent A is now terminated, and its subgraph of "1,2,3" is stored at Node 3.
Agent B migrated to Node 4, and now has a subgraph of "1,2,4"

Agent B has a size of 3, so now it will also terminate itself and deposit the subgraph results at it's final node, Node 4.



Step 5

Agent B is now terminated, and its subgraph of "1,2,4" is stored at Node 4.

When the entire network traversal is complete, the MASS-based program collects all data from all network nodes, using the return values through the places callAll() method.

MPI

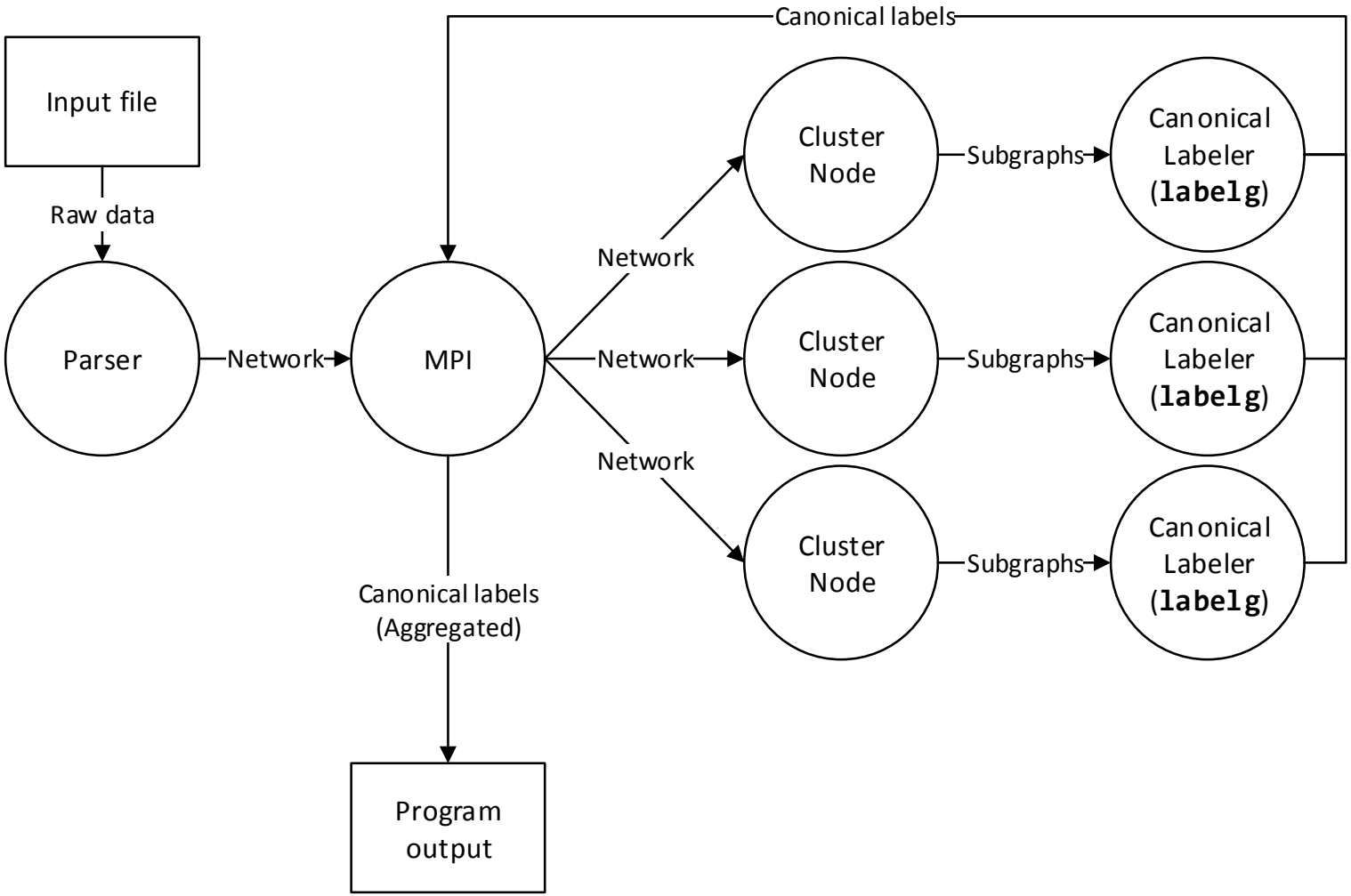
MPI (Message Passing Interface) is a library for message passing across a group of processes.

MPI DESIGN

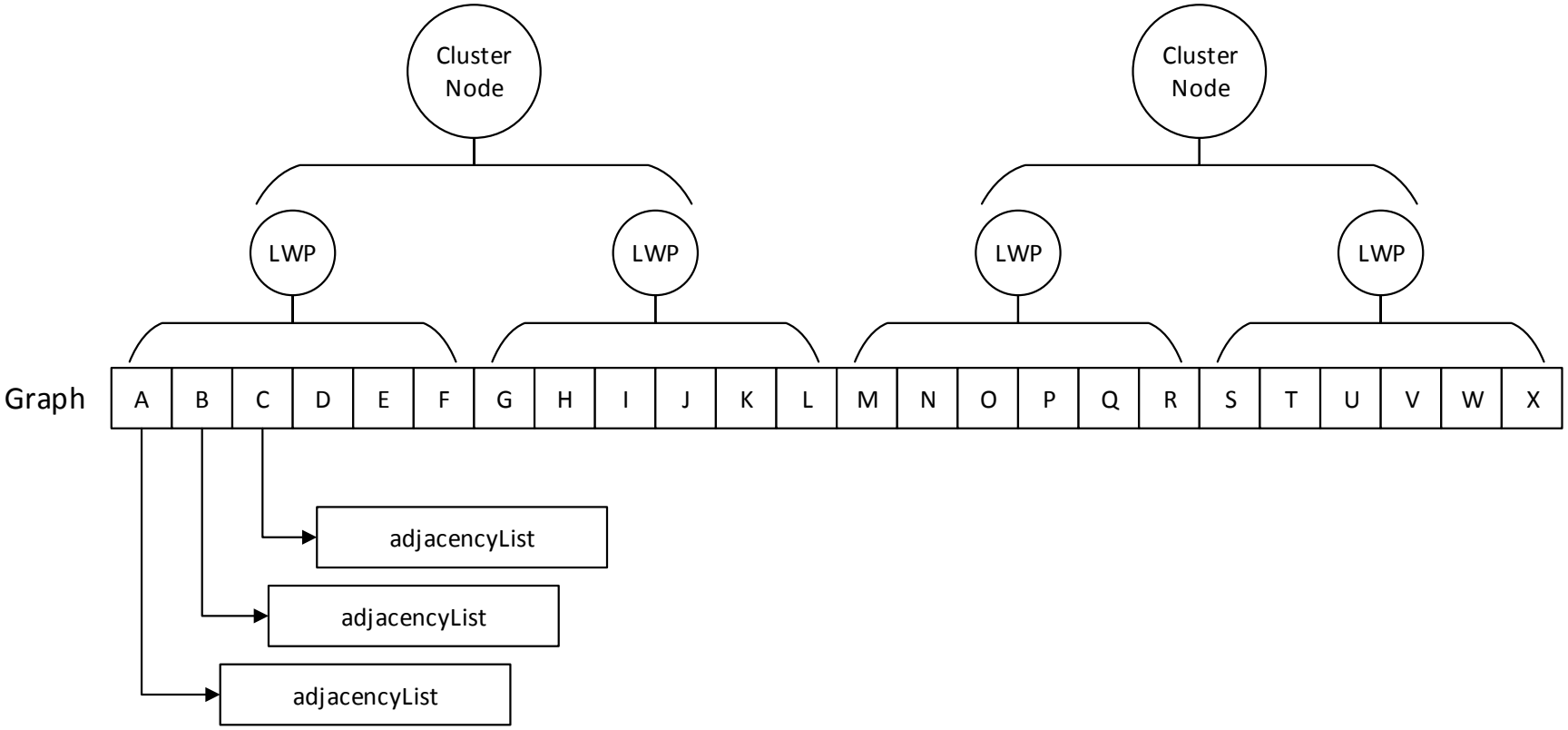
ESU is easily parallelizable at the subgraph root level.

Partition the ESU work for each node of the target graph across the MPI cluster and the local threads of cluster nodes.

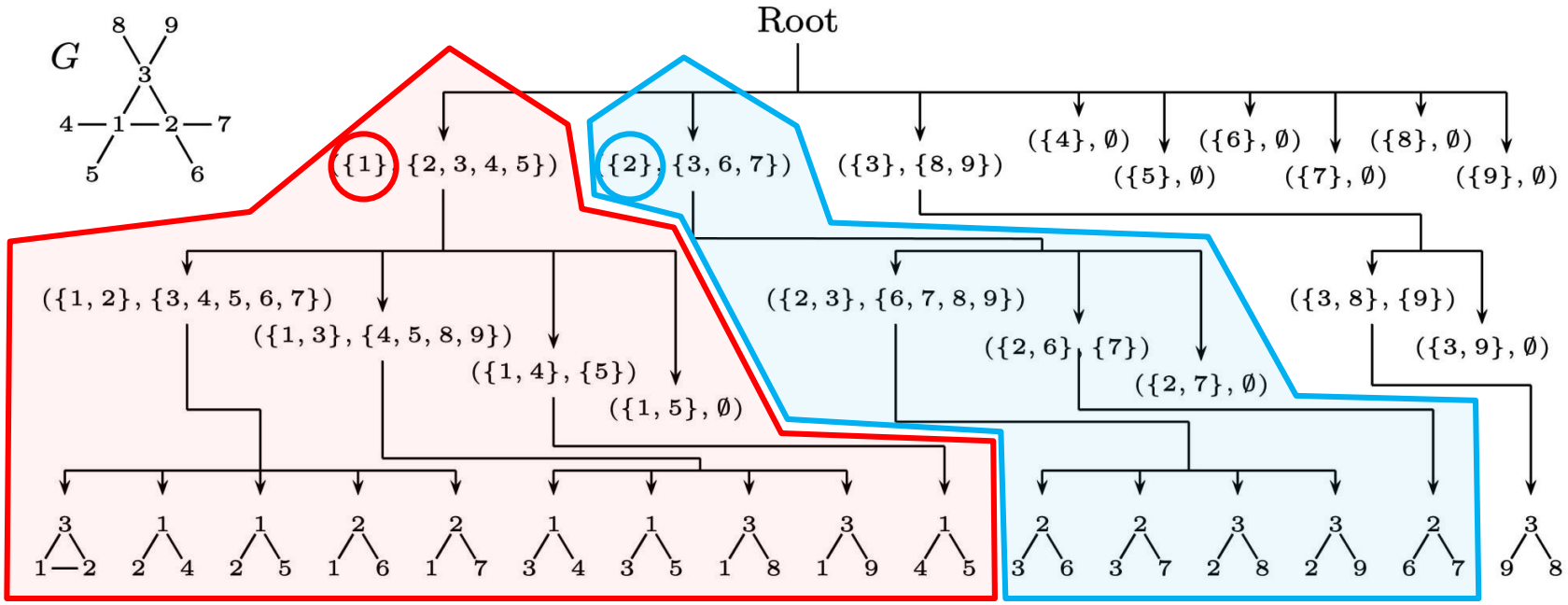
MPI DESIGN: DATA FLOW



PARTITIONING ESU



PARTITIONING ESU

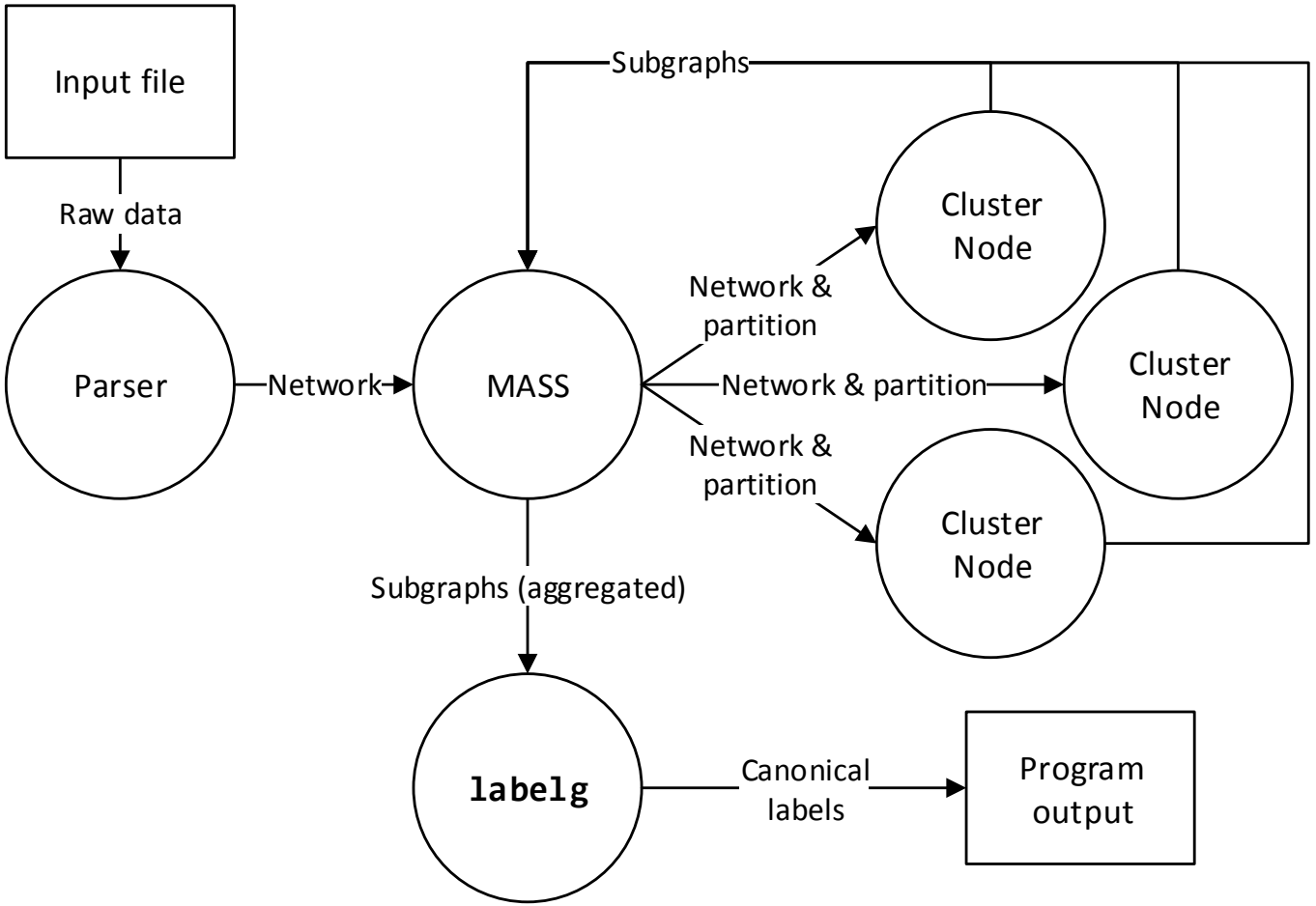


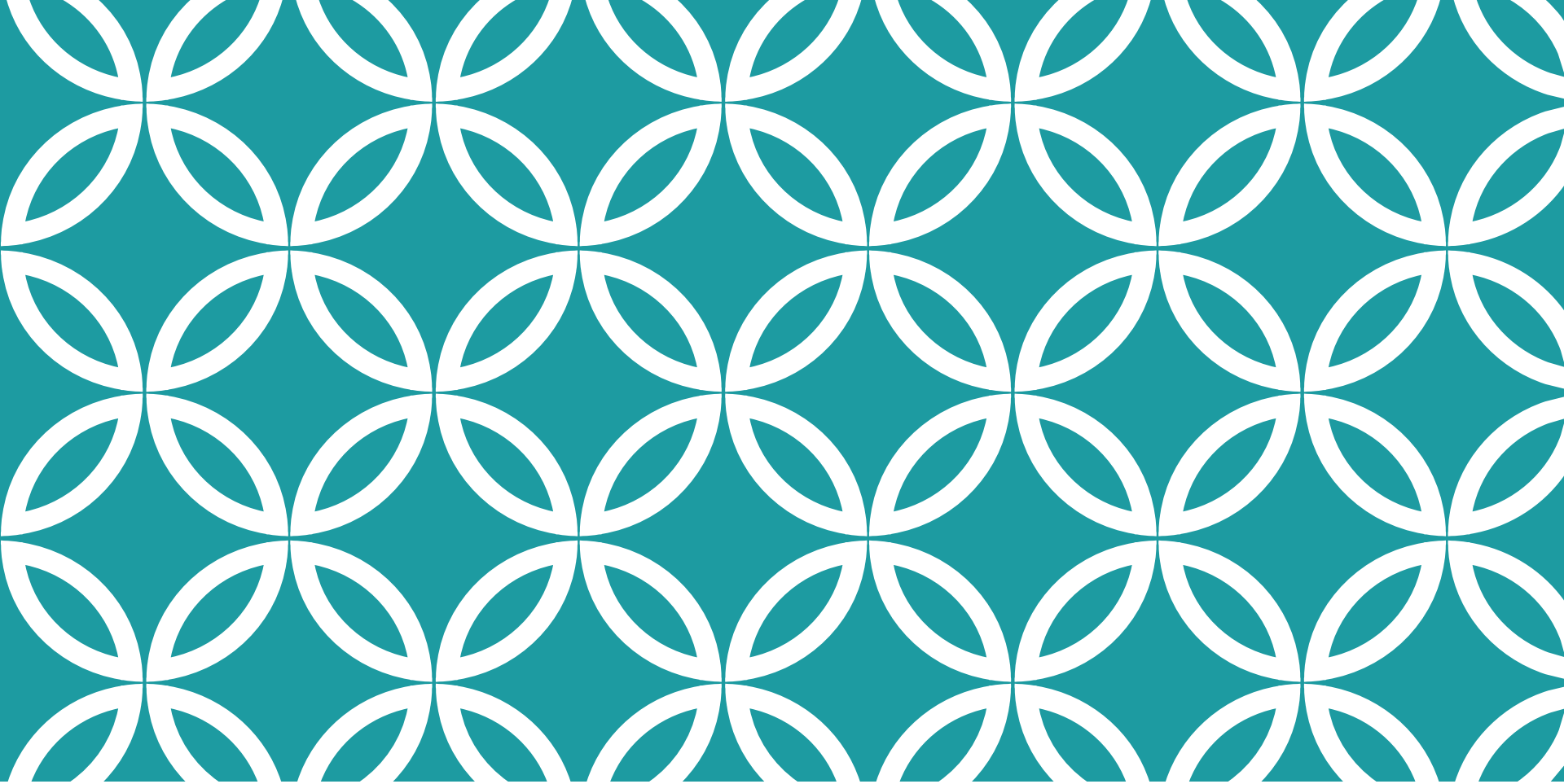
HYBRID DESIGN

Hybrid implementation – mimics MPI design

- Again, ESU is easily parallelizable at the subgraph root level – partition the network nodes (which are the subgraph roots) as "work" using the MASS virtual space.
- Not really the way MASS was designed…
- But, we know MASS partitions the virtual space across the cluster and across threads at each cluster node.

HYBRID DESIGN: DATA FLOW





RESULTS

LEGEND

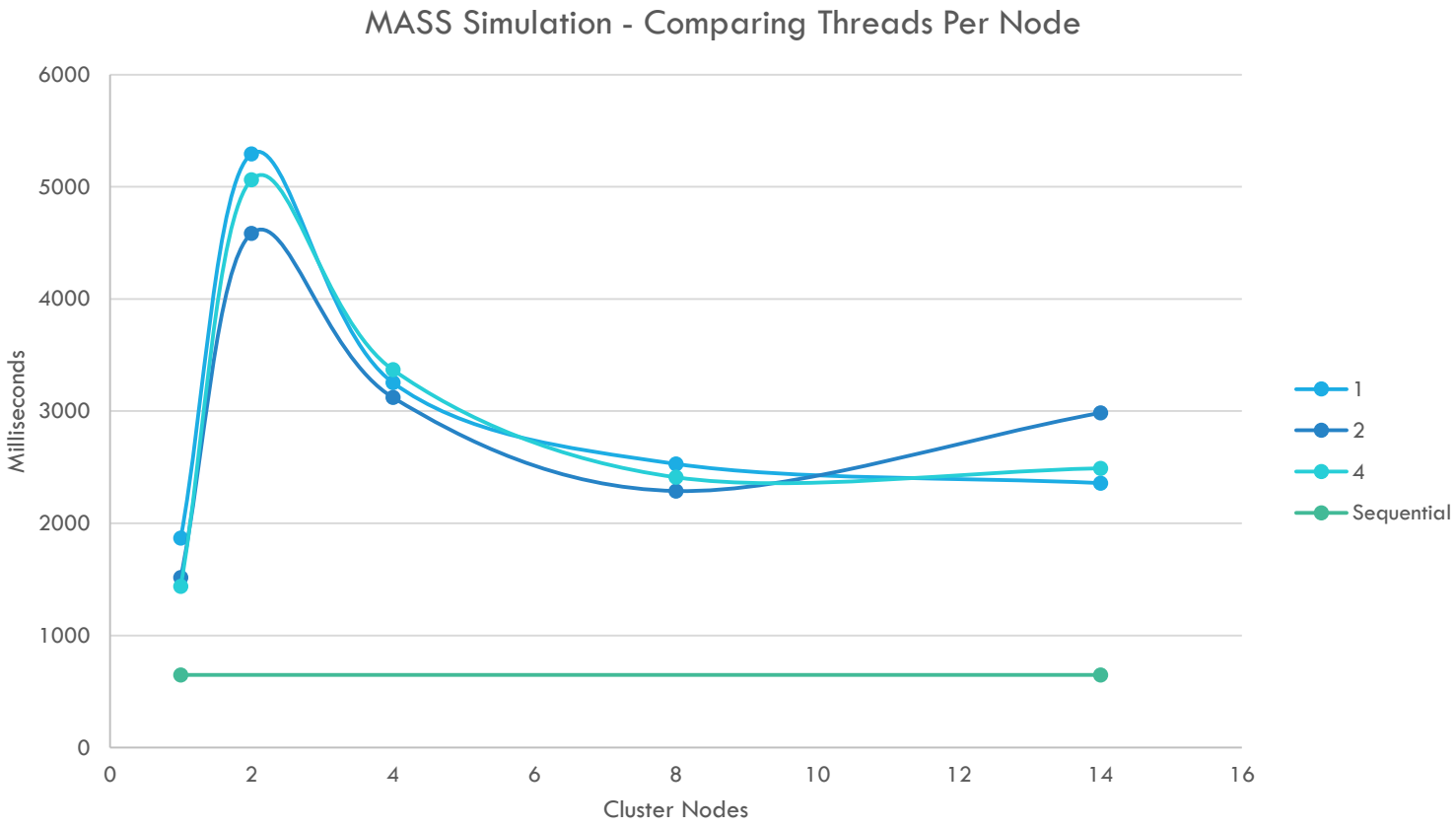
"Sequential" – sequential network motif program

"MASS Simulation" – simulation program that uses the MASS library

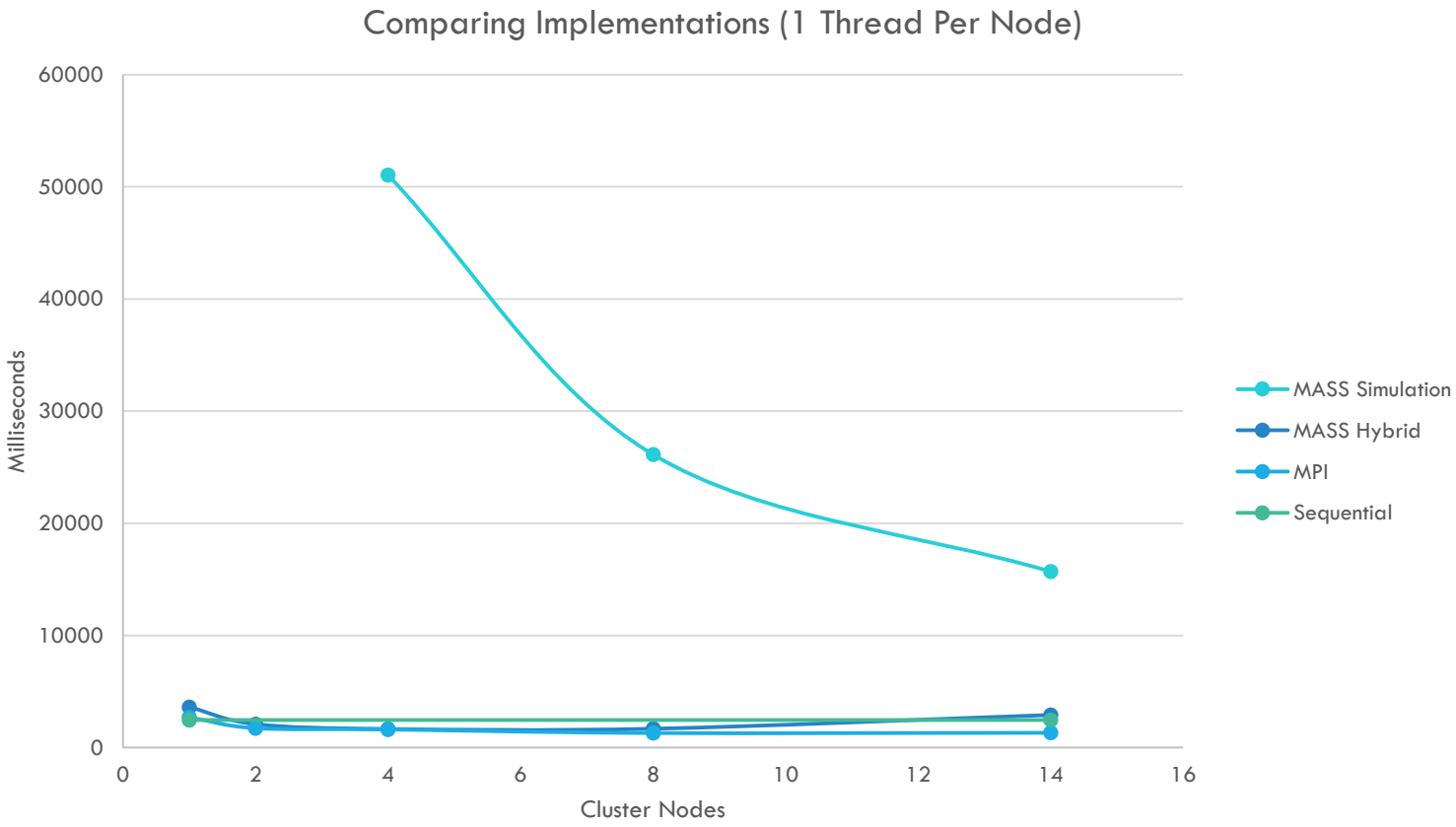
"MASS Hybrid" – hybrid program that uses the MASS library

"MPI" – program that uses the MPI library

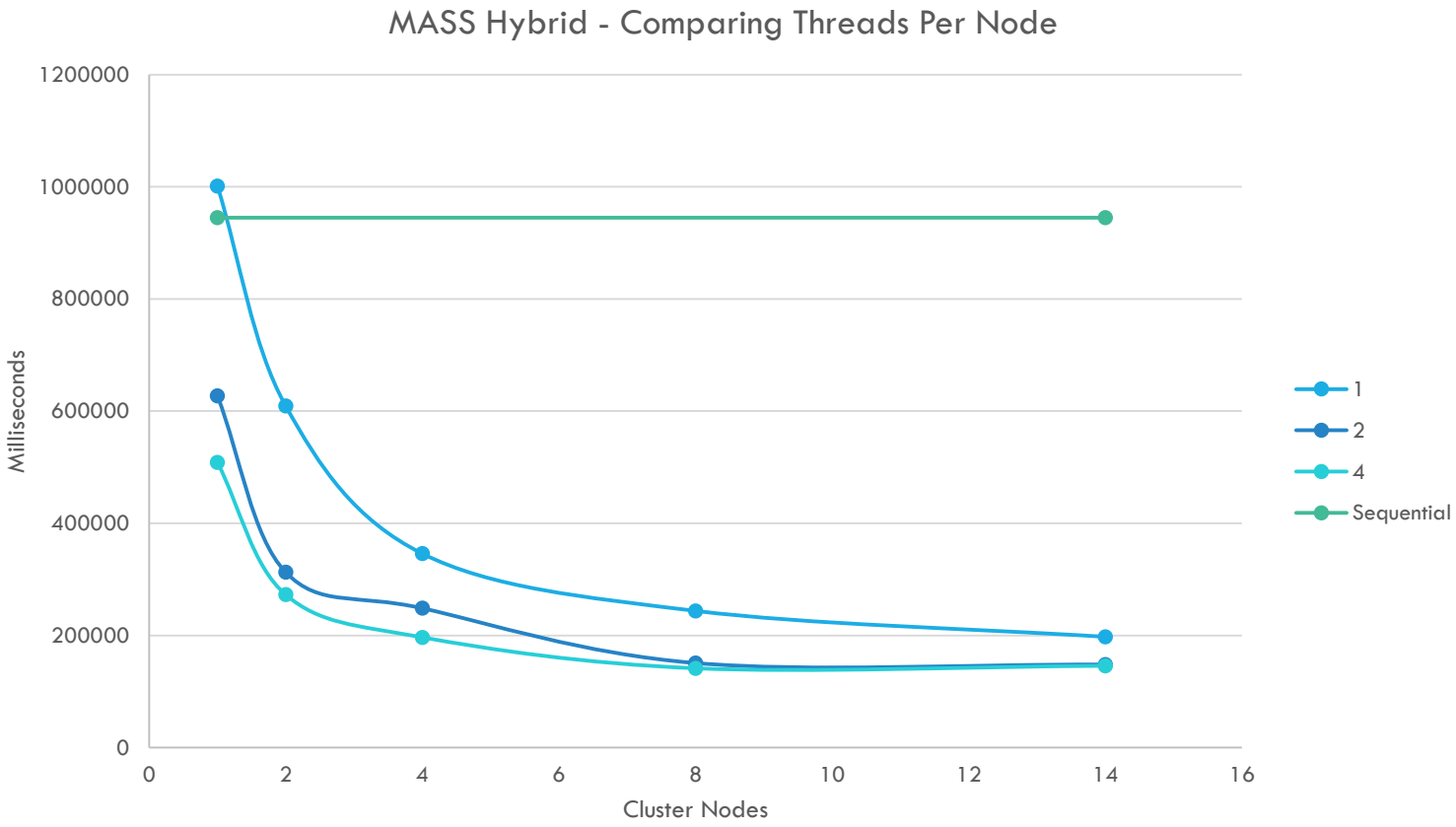
MASS SIMULATION
2365 nodes, motif 4



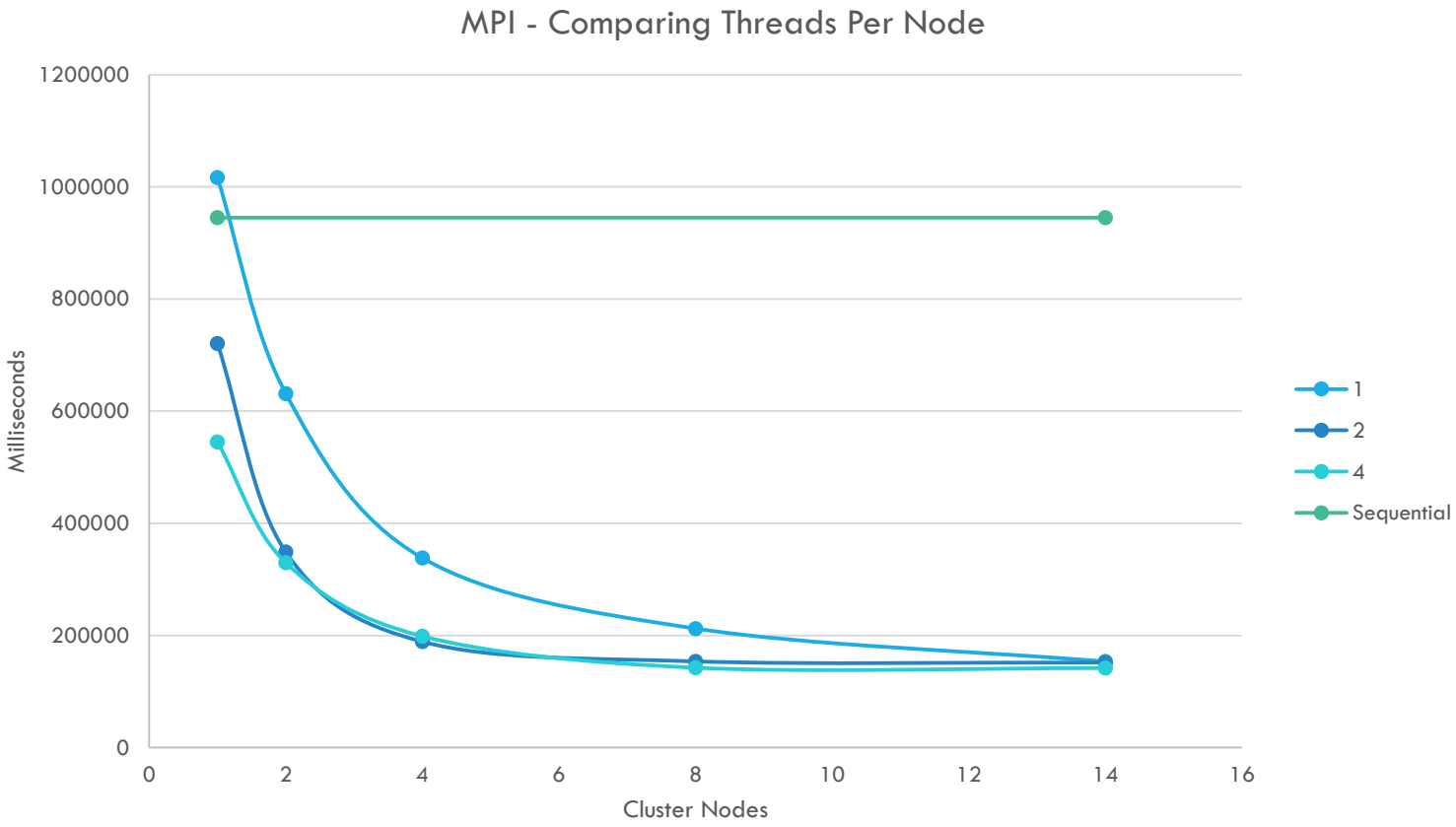
COMPARISON
2365 nodes, motif 5



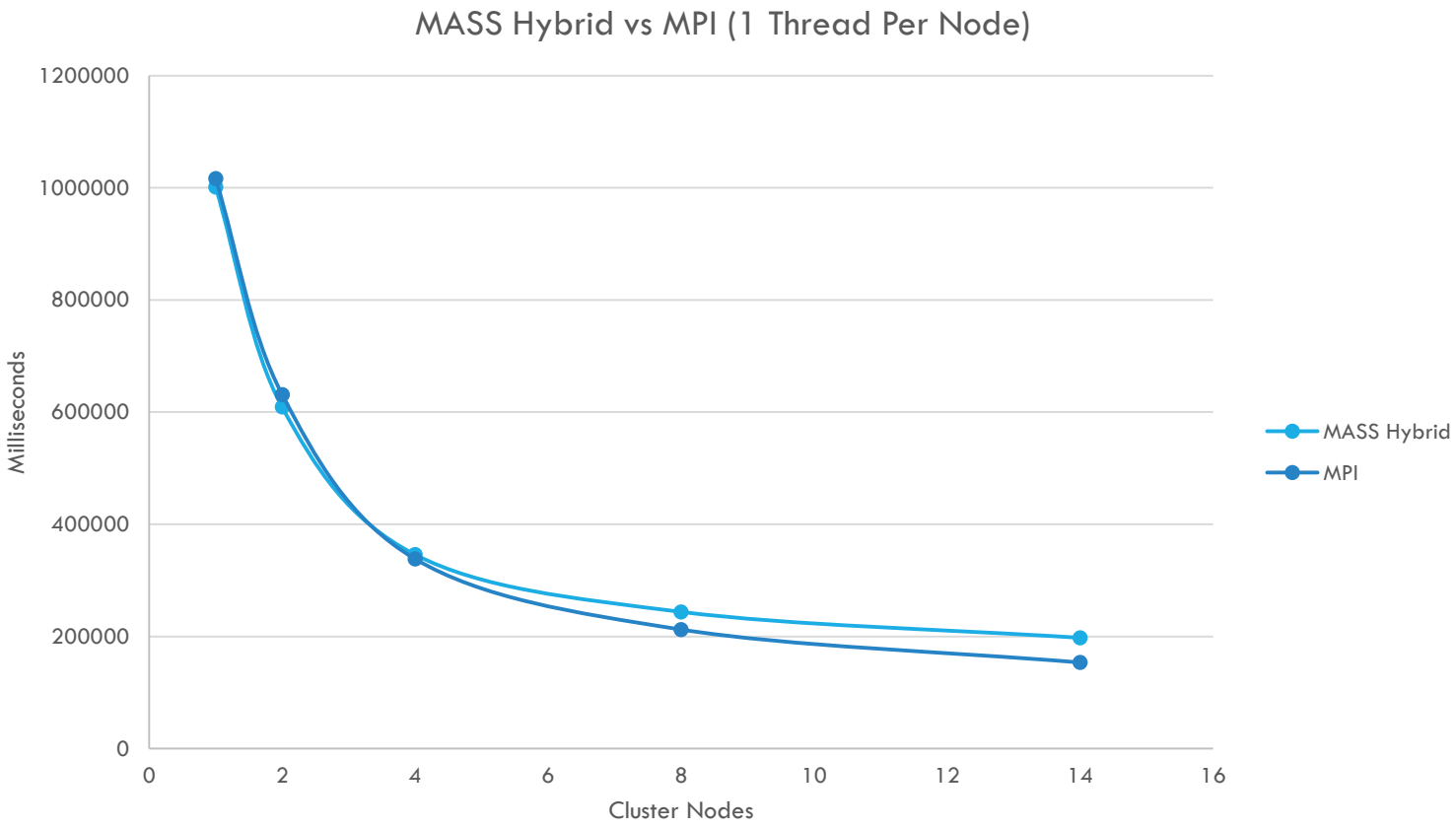
MASS HYBRID
5134 nodes, motif 5



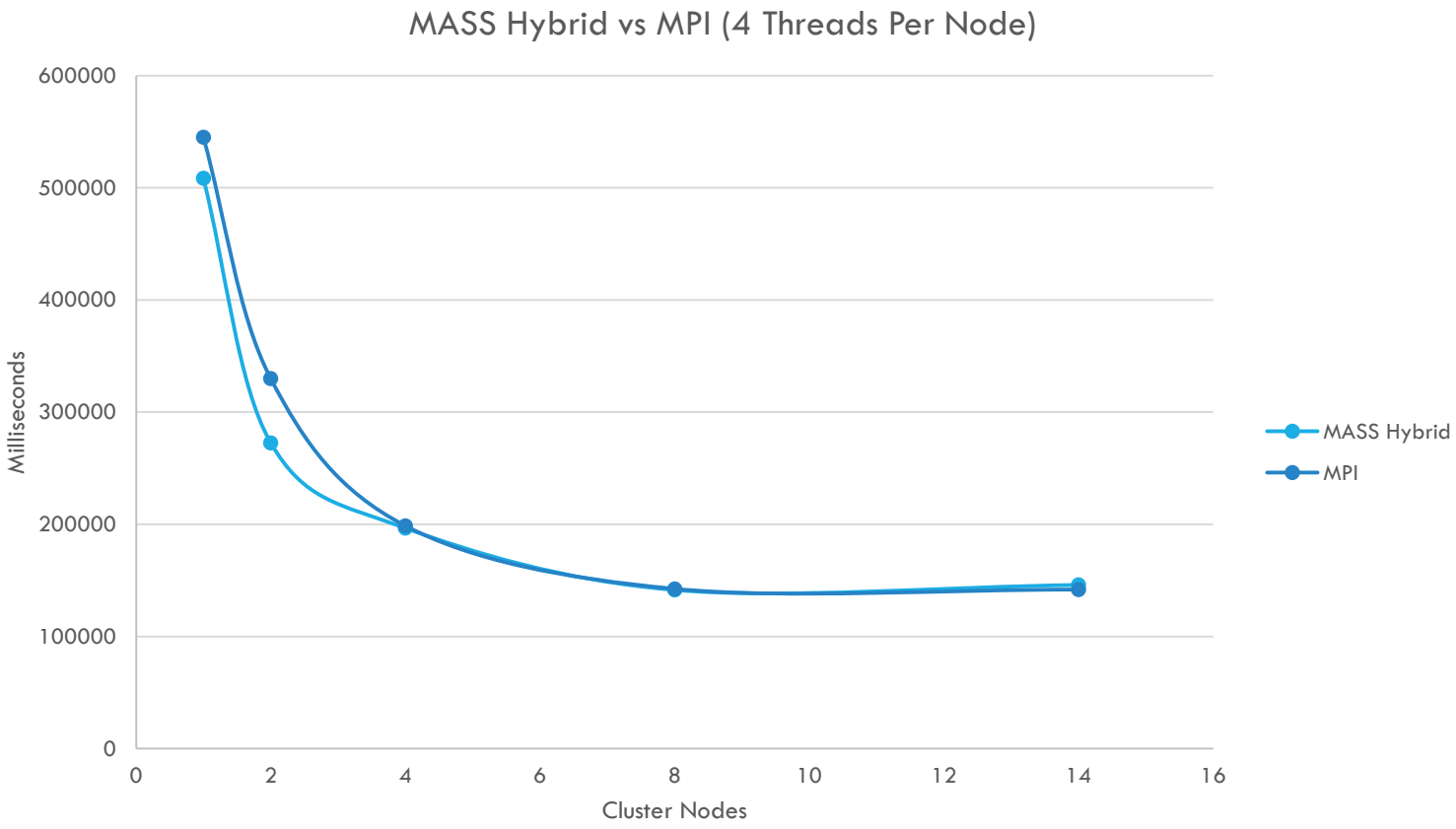
MPI
5134 nodes, motif 5

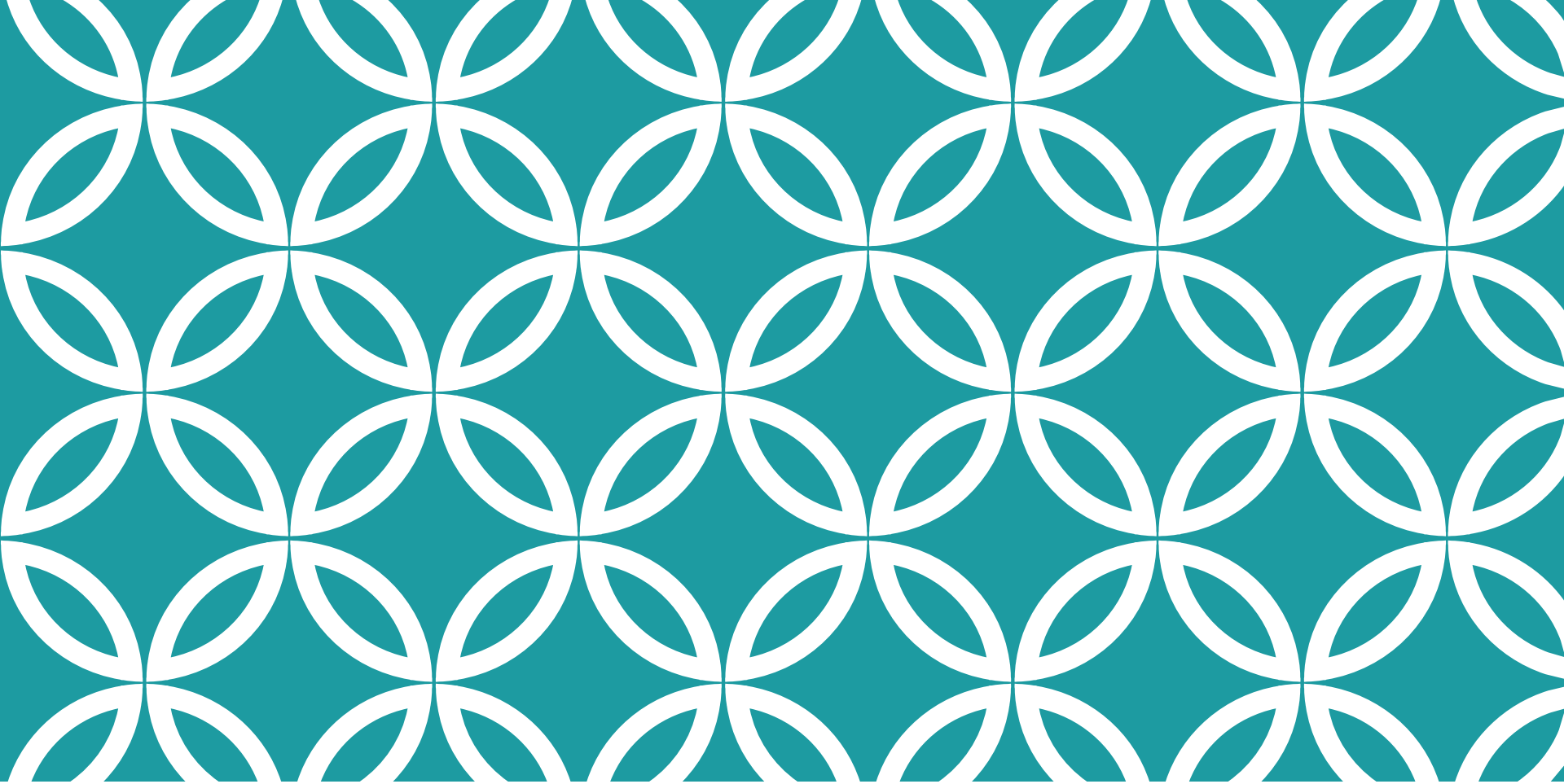


MASS HYBRID VS MPI



MASS HYBRID VS MPI





OBSERVATIONS & ANALYSIS

PERFORMANCE

MASS Simulation implementation shows promise in parallelizability at scale.

However, it also struggles with memory usage at scale.

For the tested scenarios, this program performed significantly slower compared to the other programs.

PERFORMANCE

MASS Hybrid implementation and MPI implementation are very comparable in terms of performance.

This is despite MPI having several (small) advantages:

- Dynamic partitioning between local threads within a cluster node
- distrubuted `labelg` execution minimizing data transfer

PERFORMANCE CONCERNS

All implementations are at risk of load imbalance, especially as the network motif size gets larger.

This occurs when the number of subgraphs generated from a single root is extremely large.

PROGRAMMABILITY

The MASS Simulation program presented a unique challenge due to the design principles (entity "behavior").

Very little user-level synchronization required.

Zero thread management.

Somewhat limited by the inherent design of MASS.

It proved difficult to control the rate of crawler spawning (limited library tools).

PROGRAMMABILITY

The MASS Hybrid program is unusual in that it does not necessarily mirror the intentions of MASS.

Clear separation of program logic from parallelization logic.

Required some knowledge of how the library works.

Less control over fine-tuning.

Zero user-level thread management and synchronization.

PROGRAMMABILITY

The MPI program is a fairly simple parallel implementation of the ESU algorithm.

Extremely customizable. Flexible communication protocol, but requires some experience.

No threading support, nor any synchronization support (requires user to implement).

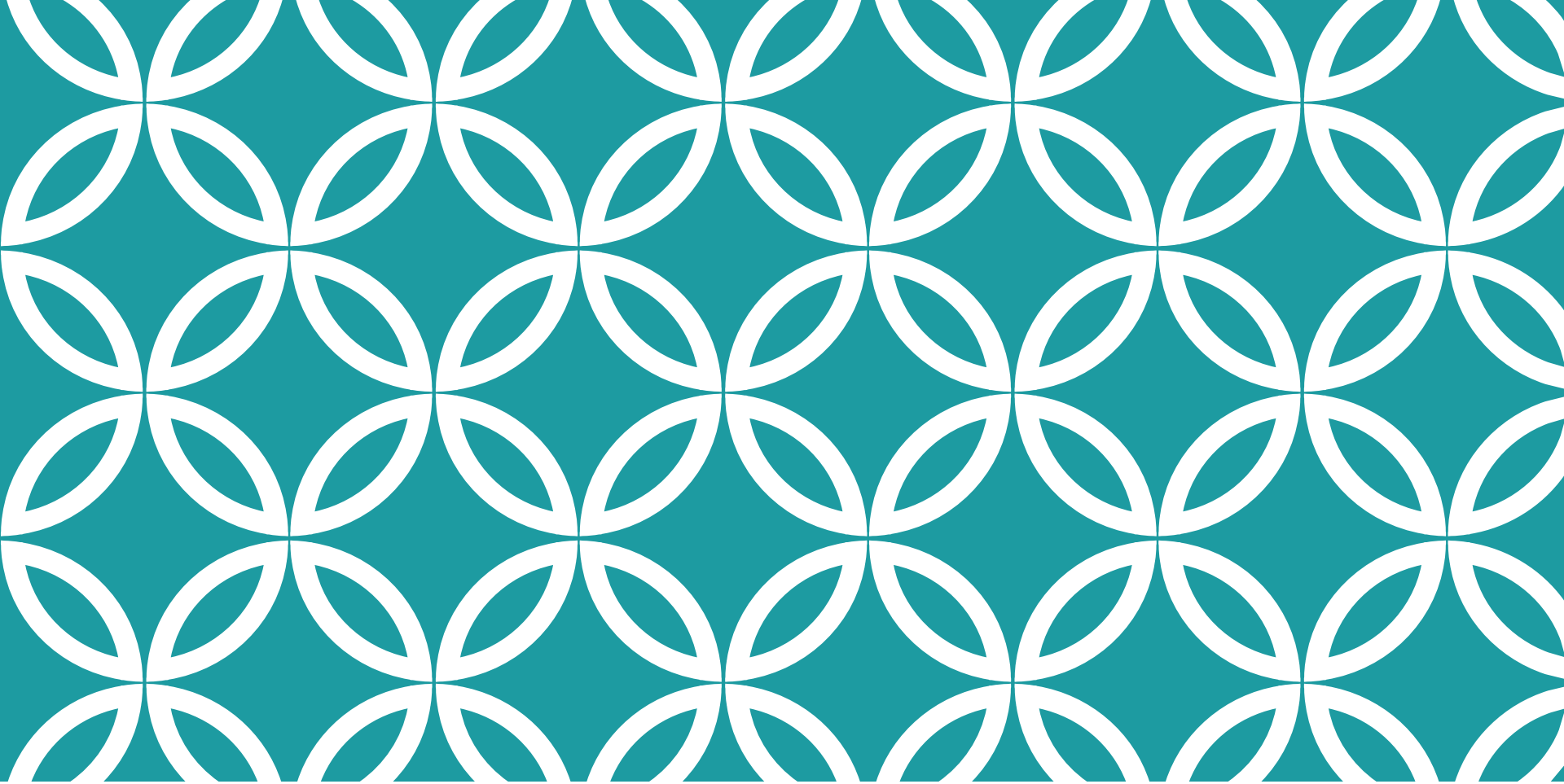
Risk of tight coupling between parallelization code and algorithm code makes things harder to change.

FUTURE CONSIDERATIONS

Extend all implementations to include RAND-ESU variation of ESU.

For MPI, exchange status periodically across the cluster for global dynamic partitioning.

Compare these programs to an implementation using Hadoop (currently under development at UWB).



QUESTIONS?

REFERENCES

B. H. Junker and F. Schreiber, *Analysis of Biological Networks*, Wiley, 2008.

S. Wernicke, *Efficient detection of network motifs*, IEEE/ACM Trans. Comp. Biol. Bioinformatics, vol. 3, no. 4, pp. 347-359, 2012.

Wooyoung Kim, et al., *Network Motif Detection: Algorithms, Parallel and Cloud Computing, and Related Tools*, Tsinghua Science and Technology, vol. 18, no. 5, pp. 469-489, 2013.

T. Chuang and M. Fukuda, *A Parallel Multi-Agent Spatial Simulation Environment for Cluster Systems*, IEEE CSE, pp. 143-150, 2013.

LINKS

MASS research project homepage:
http://depts.washington.edu/dslab/MASS/index.html

mpiJava homepage (with links to MPI):
http://www.hpjava.org/mpiJava.html

nauty and Traces homepage (labelg):
http://cs.anu.edu.au/~bdm/nauty/

Sequential program (ESU in Java):
https://github.com/mtkp/network-motif

Parallel program (ESU with MPI):
https://github.com/mtkp/mpi-network-motif