Introduction

Topological ordering of DAGs is essential in various domains such as scheduling, deep learning computation order, and dependency resolution in rendering. Traditional topological sort algorithms, like Kahn's algorithm or depth-first search (DFS), are inherently sequential and can become bottlenecks in systems requiring high performance.
Our project aims to accelerate the topological sorting process by exploiting parallelism through:

- **Serial Implementation** (Baseline): We will implement a standard sequential topological sort algorithm to serve as a baseline for measuring performance improvements.
- **CUDA for Matrix Manipulation**: By representing the DAG as an adjacency matrix, we can leverage CUDA's parallel processing capabilities to perform simultaneous operations on multiple nodes and edges.
- **Domain-Specific Language** (DSL): Utilizing or developing a DSL tailored for expressing dependencies, we aim to optimize parallel processing by abstracting the complexity of parallelism and allowing for more efficient execution of topological sorting algorithms.

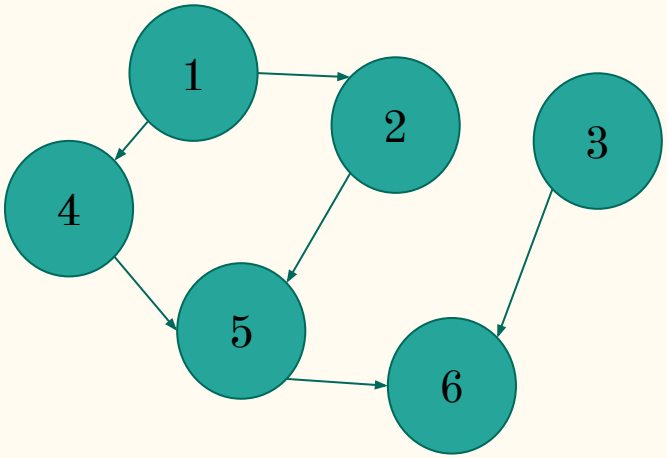
Related Work

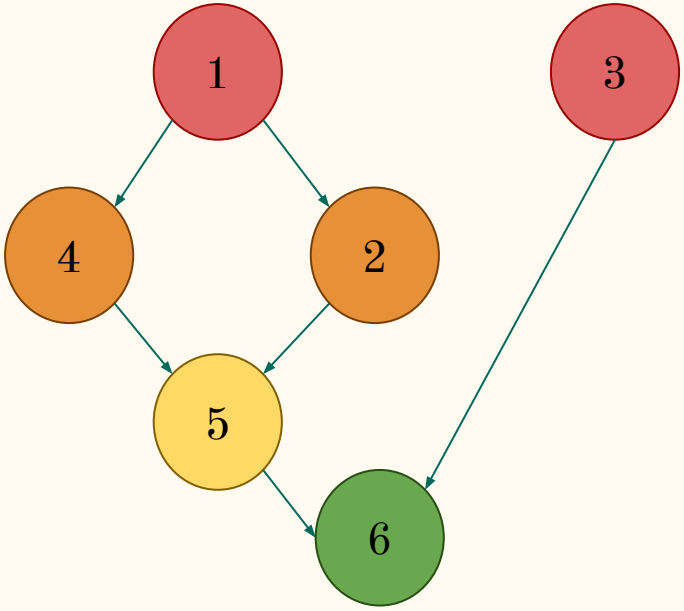
[1] Kahn, A. B. (1962). Topological sorting of large networks. Communications of the ACM, 5(11), 558-562.

[2] Pearce, D. J., & Kelly, P. H. (2007). A dynamic topological sort algorithm for directed acyclic graphs. Journal of Experimental Algorithmics (JEA), 11, 1-7.

[3] Svantesson, D., & Eklund, M. (2016). A naive implementation of Topological Sort on GPU: A comparative study between CPU and GPU performance.

Data Inputs/Outputs and Problem Definition





data.in

```
6
0
1 1
0
1 1
2 2 4
2 3 5
```

data.out

```
4
2 1 3
2 2 4
1 5
1 6
```

Method - Naive Serial

- Keep count of incoming degrees of codes

1	2	3	4	5	6
0	1	0	1	2	2

1	2	3	4	5	6
0	0	0	0	2	1

1	2	3	4	5	6
0	0	0	0	0	1

1	2	3	4	5	6
0	0	0	0	0	1

Method - Cuda

- In a serial approach, you'd typically find zero in-degree nodes, process them, and then update the in-degrees of neighbors one-by-one using a CPU loop. This inherently limits speed due to sequential operations and reduced utilization of CPU resources.
- The CUDA implementation parallelizes these steps: finding zero in-degree nodes, updating their processed status, and decrementing the in-degrees of their neighbors all occur in parallel across thousands of GPU threads, dramatically improving throughput.
- Instead of iterating over nodes and edges sequentially, the CUDA code launches kernel functions that exploit data-level parallelism. Every suitable node and edge operation can be handled simultaneously, reducing the time complexity of each iteration.

Method - DSL (NetworkX)

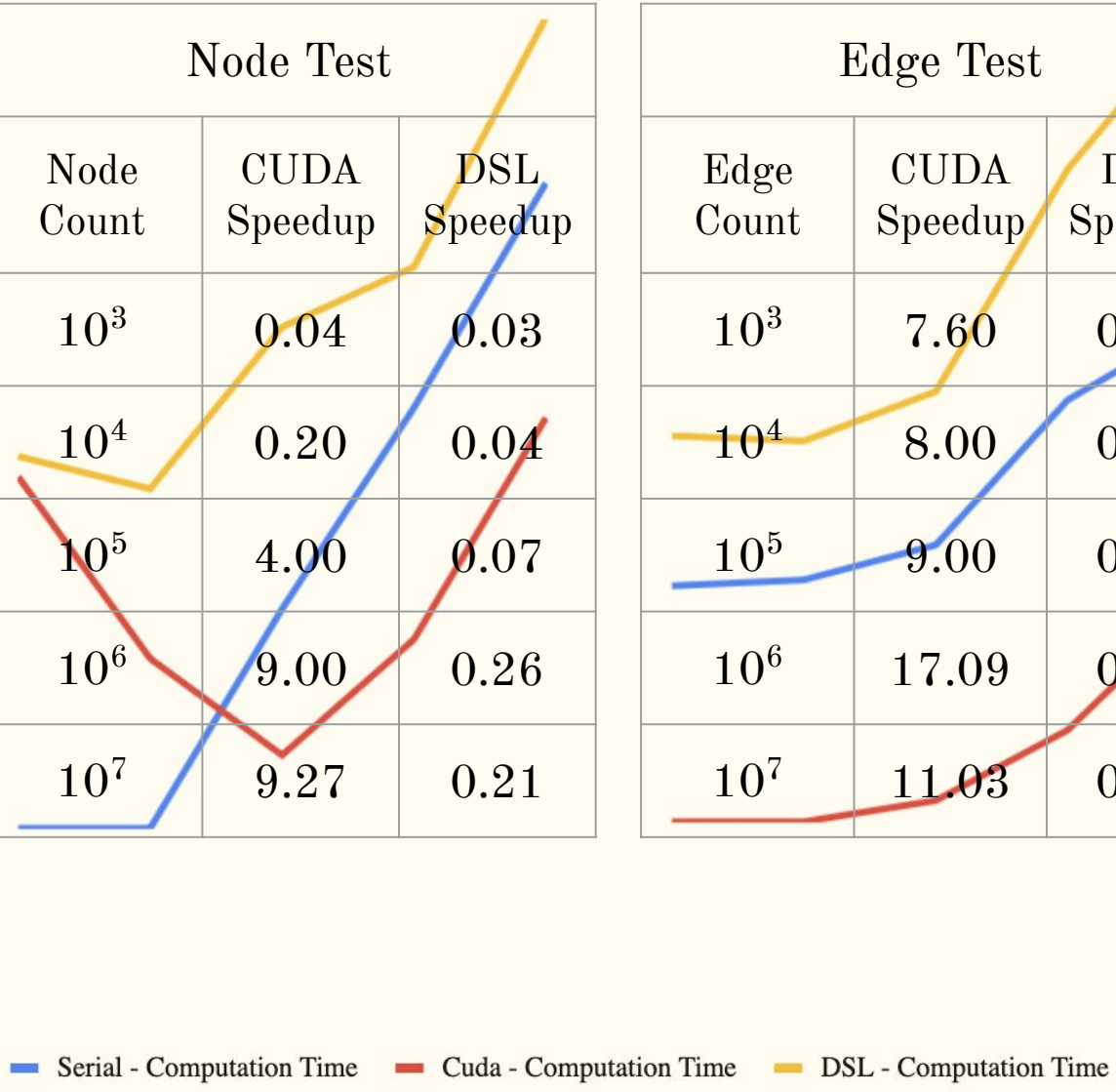
- NetworkX provides a Python interface that greatly simplifies working with graphs and networks.
- A Directed Graph (DiGraph) was created and built in methods were used to calculated dependency layers.
- Many different options exist in order to maximize performance, but speedup depends greatly on the methods being used and the logic of the program. Our team found the default backend with PyPy runtime to be most performant for our work, which generally provides a 2-5x speedup
- The total times are especially impacted by poor init times caused by the use of Python3/PyPy

Tests and Results

Node Test		
Node Count	CUDA Speedup	DSL Speedup
10^3	0.04	0.03
10^4	0.20	0.04
10^5	4.00	0.07
10^6	9.00	0.26
10^7	9.27	0.21

Edge Test		
Edge Count	CUDA Speedup	DSL Speedup
10^3	7.60	0.28
10^4	8.00	0.30
10^5	9.00	0.27
10^6	17.09	0.14
10^7	11.03	0.07

Depth Test		
Depth Count	CUDA Speedup	DSL Speedup
1	7.60	0.33
10^1	31.22	0.16
10^2	6.19	0.17
10^3	0.71	0.18
10^4	0.07	0.18
10^5	0.01	0.22
10^6	0.00	0.37



Conclusions

- CUDA & SIMD may be used to accelerate topological sorting of DAGs, especially with many nodes and edges and when depth is limited. However, it comes at the cost of added complexity over a simple serial algorithm.
- Topological sorting of DAGs using SIMD may provide significant computational speedup over serial approaches especially when considering large datasets.
- DSLs offer a powerful, quick and convenient way to implement algorithms, but developers will be limited to the bounds of the framework. Developers should thoroughly review capabilities before deciding if a DSL will fit their needs.