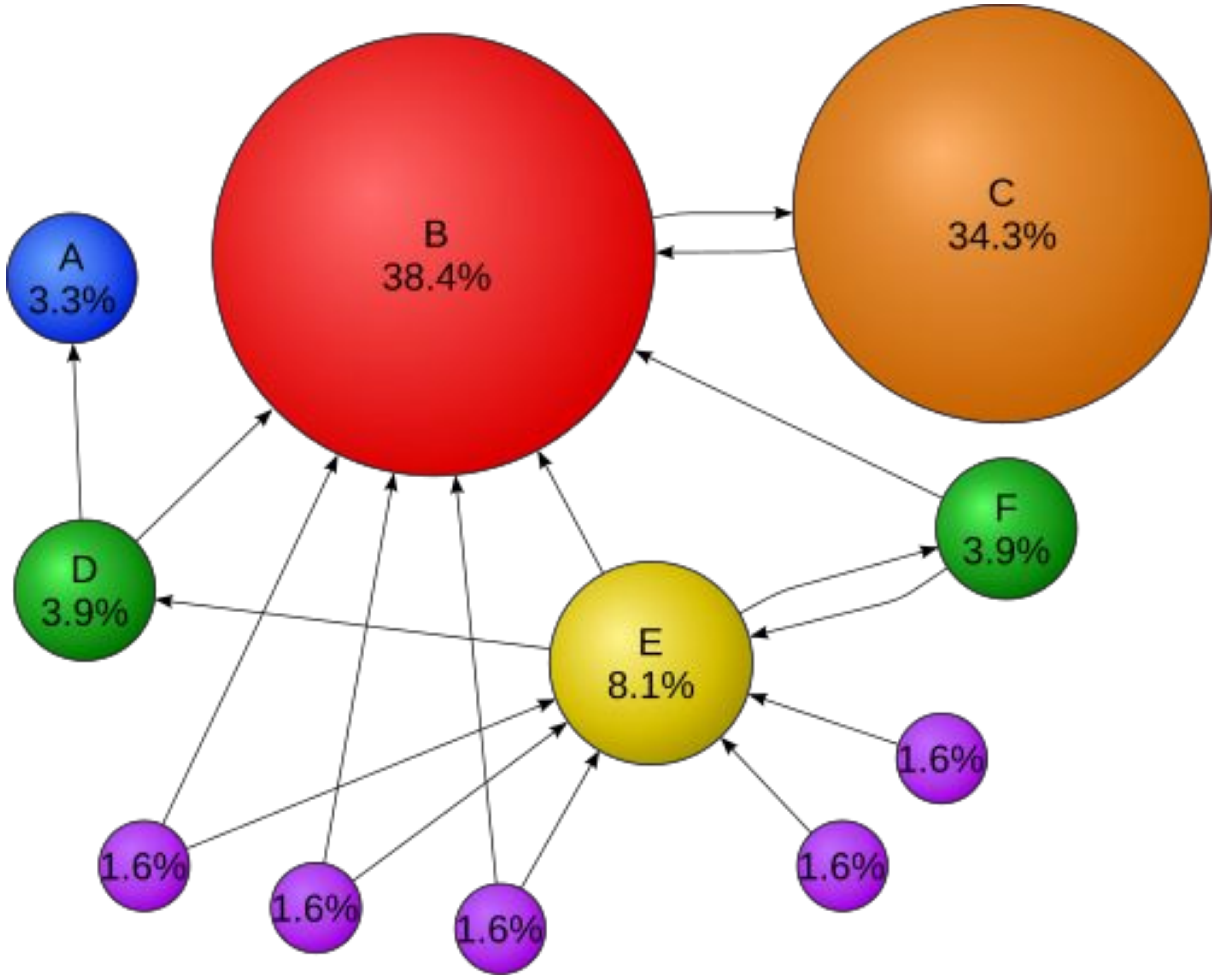
Parallelizing PageRank with MPI, OpenMP, CUDA

Ananya Mantravadi
amantra

NC STATE

Why parallel PageRank?

- Foundational algorithm used in web search (scale: billions of pages), social network analysis, and recommendation systems.
- Not just about search engines. It’s a universal ranking system.
- Highly compute-intensive and massively parallelizable.



https://en.wikipedia.org/wiki/PageRank#/media/File:PageRanks-Example.svg

NC STATE

Problem Description

- PageRank = Probability(landing on a given page)

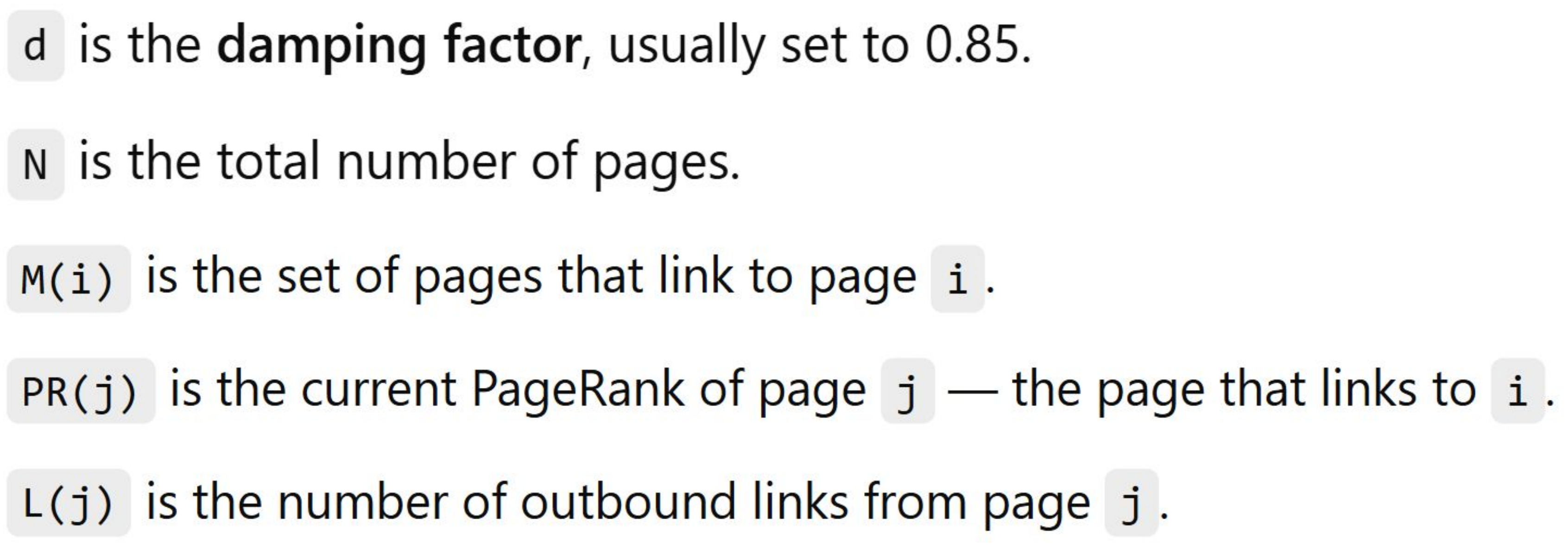
- Each page’s score of importance depends on:
  - number of pages linking to it
  - as well as their importance

- Random surfer model:
  User randomly clicking links - either follow current page’s links or jump to a random page.

- Iterative power method:
  Repeatedly update page scores till they converge. Each iteration every page’s new score is computed from scores of pages linking to it

PageRank Formula:

$$PR(i) = \frac{1-d}{N} + d \sum_{j \in M(i)} \frac{PR(j)}{L(j)}$$



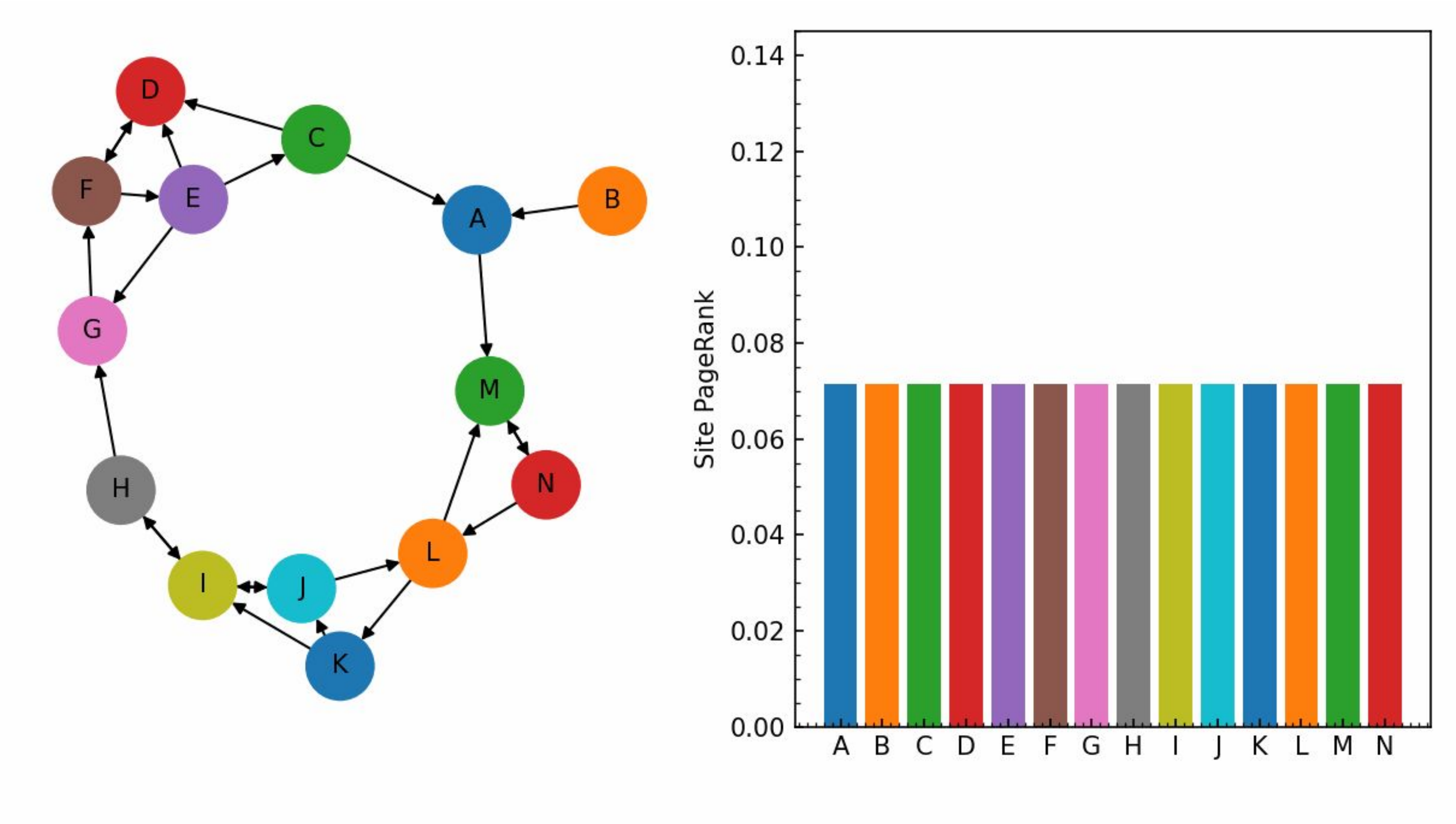
`d` is the **damping factor**, usually set to 0.85.

`N` is the total number of pages.

`M(i)` is the set of pages that link to page `i`.

`PR(j)` is the current PageRank of page `j` — the page that links to `i`.

`L(j)` is the number of outbound links from page `j`.



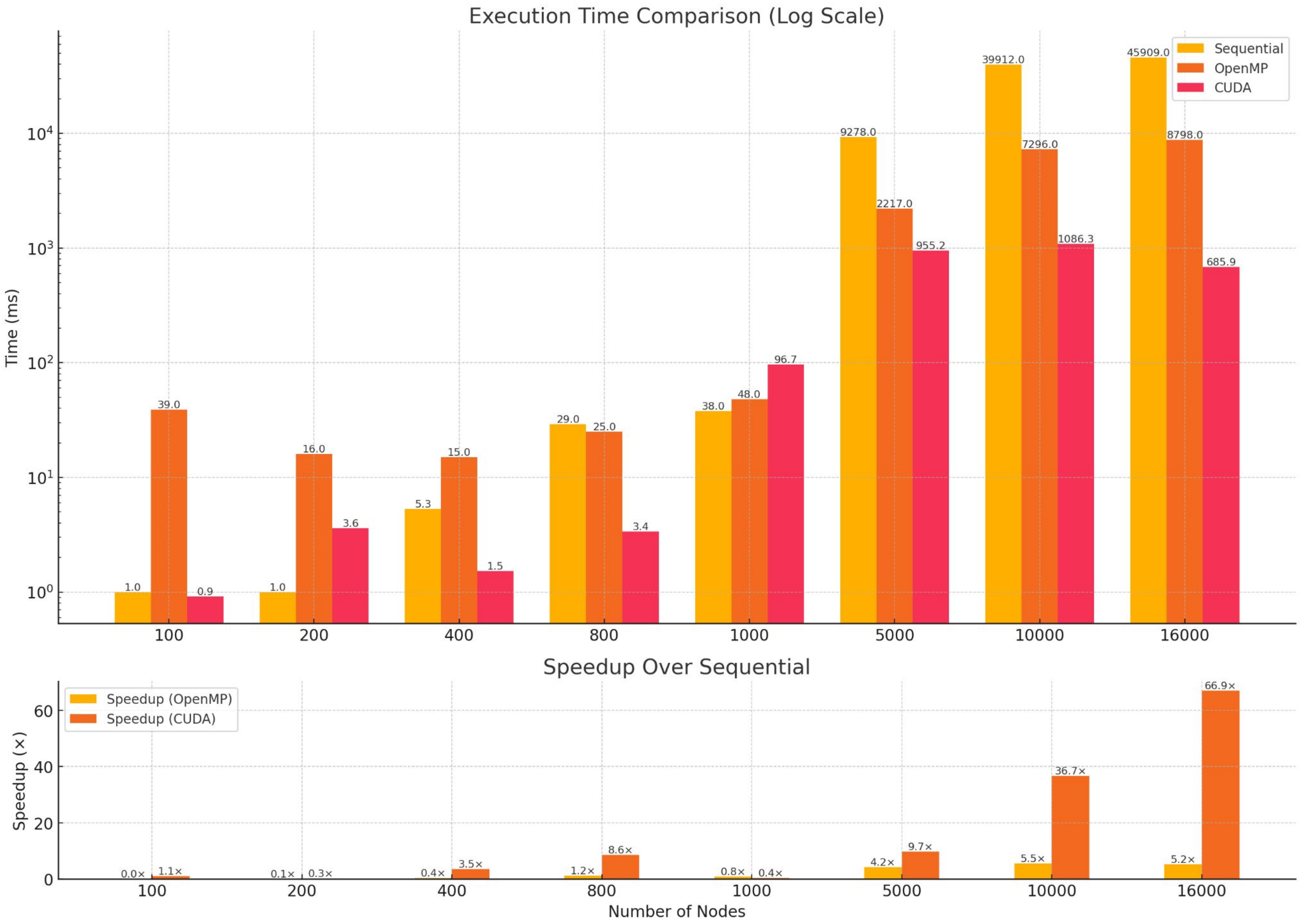
https://en.wikipedia.org/wiki/PageRank#/media/File:Page_rank_animation.gif

Related Work

- **Google's evolving approach:** From the start, Google computed PageRank on server clusters. They later described many algorithms including RankBrain (2015) and Hummingbird (2013)
- **Big data frameworks**: PageRank is implemented in frameworks like Apache Hadoop and Spark as a benchmark graph algorithm (Benchmarking Big Data Systems: Performance and Decision-Making Implications in Emerging Technologies)
- **Challenges:** A known bottleneck is the communication and synchronization required each iteration. Even with significant speedups, **inter-node synchronization overhead can limit efficiency** in MPI or distributed setups. Need to ensure load balance and efficient memory access.

NC STATE

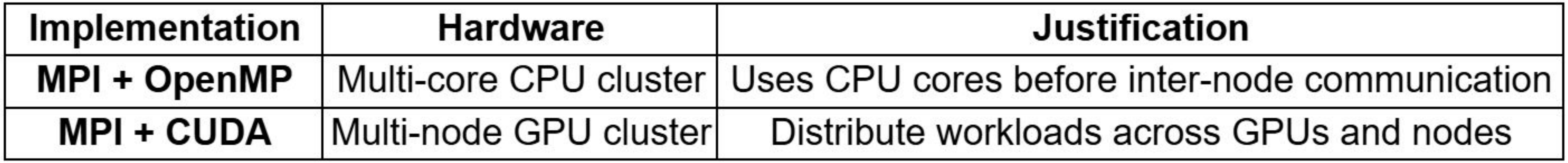
Current Progress



Execution Time Comparison (Log Scale)

Speedup Over Sequential

Milestones



Implementation	Hardware	Justification
MPI + OpenMP	Multi-core CPU cluster	Uses CPU cores before inter-node communication
MPI + CUDA	Multi-node GPU cluster	Distribute workloads across GPUs and nodes