Scheduling for Realistic Global Memory Machines: New Scheduling Problems

Andrew A Chien (University of Chicago)
March 17, 2026

19th Scheduling for Large-Scale Systems Workshop
Frejus, France


THE UNIVERSITY OF CHICAGO

Argonne NATIONAL LABORATORY

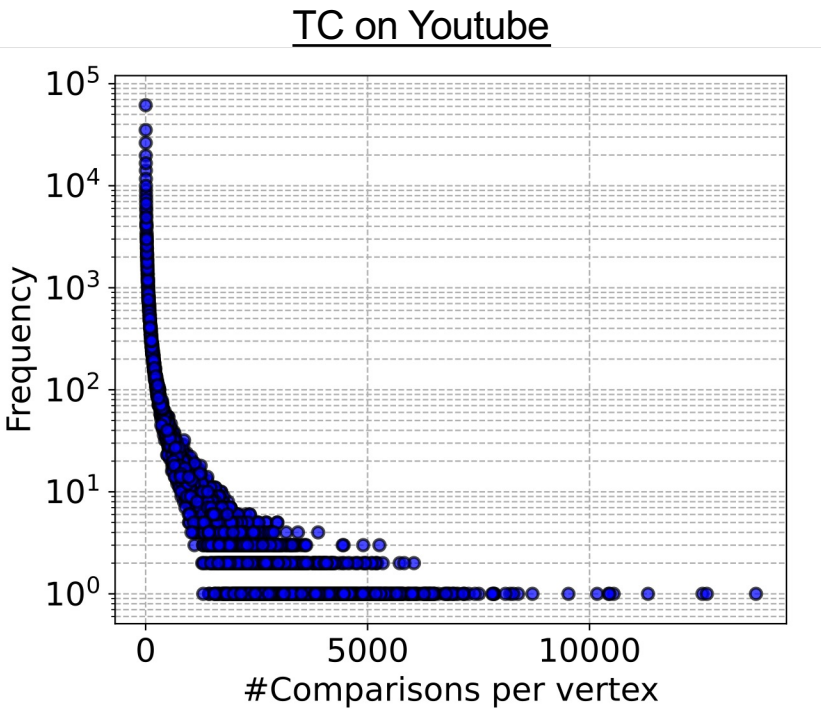

CERES
Center for Unstoppable Computing

Outline

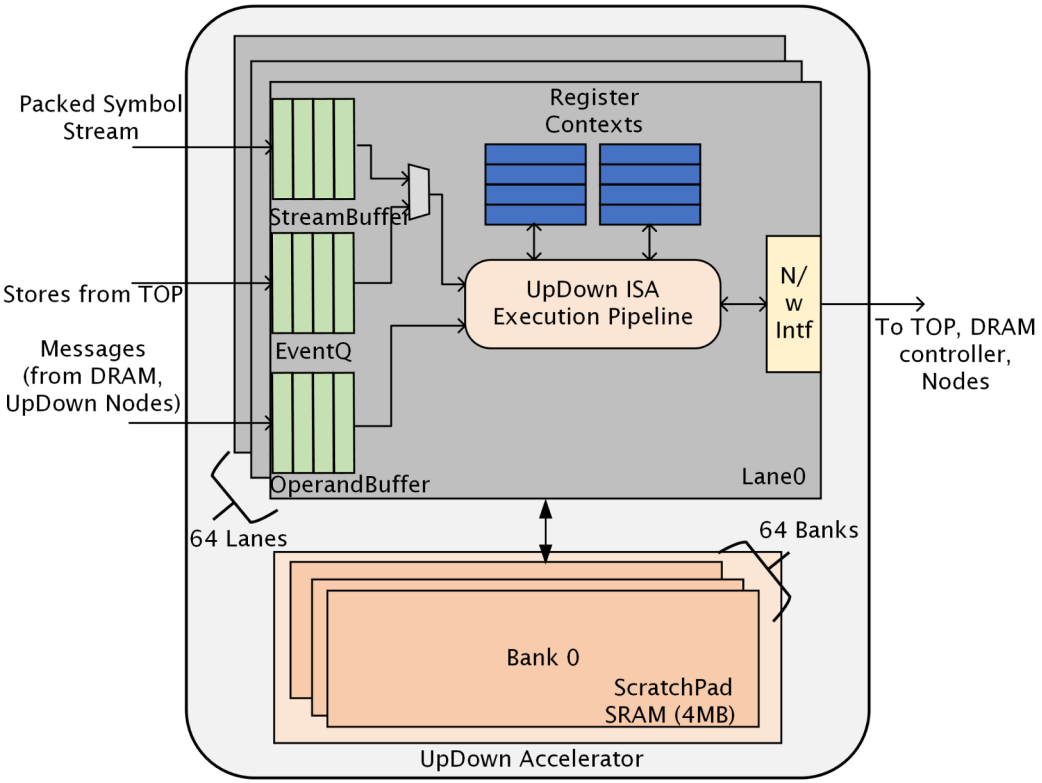
- Realistic Global Memory Machines
- Scheduling Problems
- Simple Models
- More Realistic Models
- Discussion

UpDown Motivation

- World's largest graphs
  - Meta/Facebook 2B Daily active users
  - Whatsapp 3B DAU, 140B messages/day
  - Google knowledge graph 5B entities, 500B facts (2020)
  - 9B mobile phones, 6 trillion text messages in USA in 2021
- Detect patterns, activities, emerging structure
  - Rich computations on graphs (properties, matching, etc.)
  - Real-time streaming and prediction
- Graphs and computations are large, irregular, highly skewed!
- UpDown designed is part of the IARPA AGILE Program



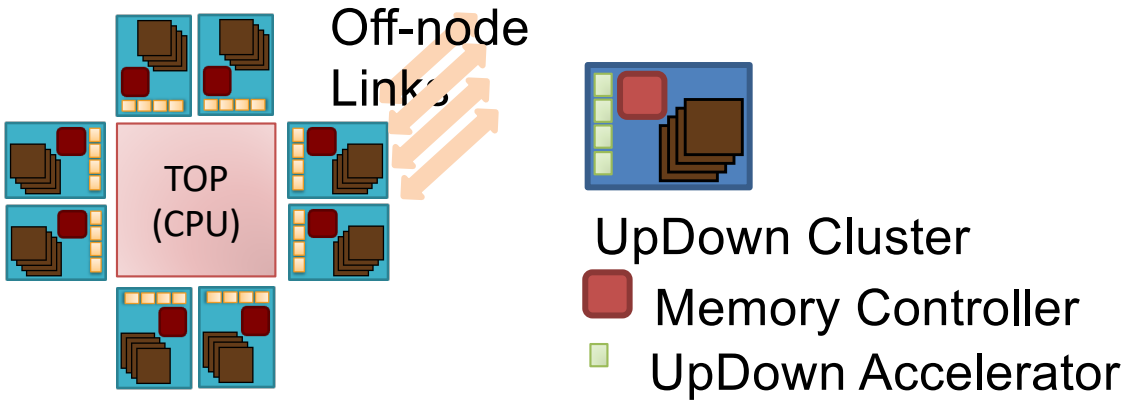
UpDown Lane Architecture (64 per accelerator)



- 1-cycle event/thread scheduling
  - Event scheduling,Thread create/schedule/destroy, 128 threads/lane, simple in-order pipeline
- Efficient short threads
  - (4 instructions, effective work)
- Scalable High memory Parallelism
  - Split-transactions to DRAM, system,
  - 22GB/s/lane, 1.4 TB/s accelerator = HBM3e
- Fast, software-named synchronization
  - Unlimited namespace
- Integrated Global Addressing and Sharing
  - Process isolation, Flexible sharing and locality
- Efficient Ingestion and Transcoding
  - PB/s streaming computation

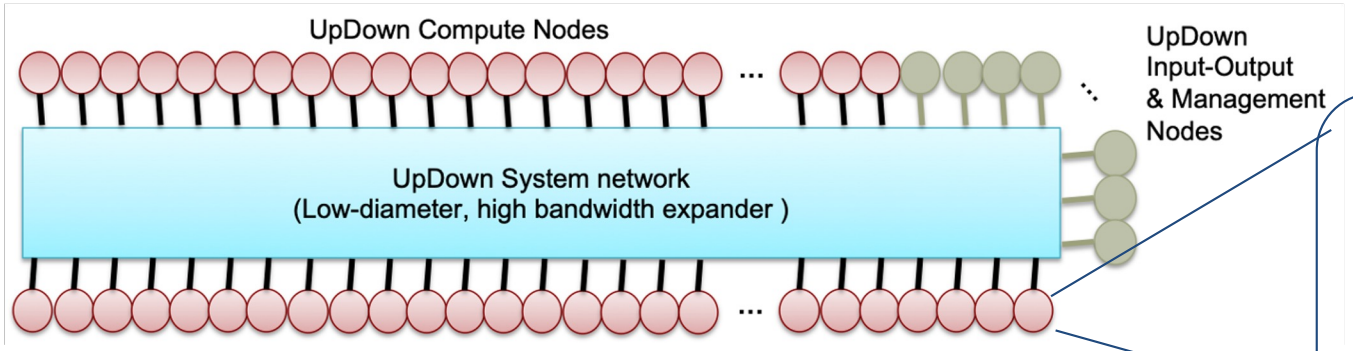
Each lane is a 2 Billion Insts/second computer

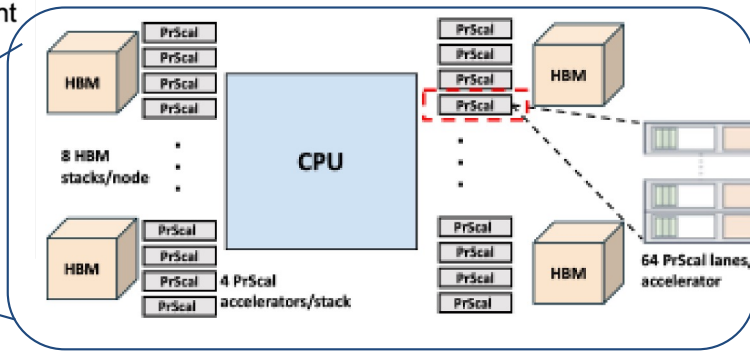
UpDown Node Design



- UpDown Node: 1 TOP, 8 UpDown Clusters
- UpDown Cluster: 4 UpDown Accelerators, memory controller, 1 HBMx Stack
- In Physical Design, Stacks/tiles are arrayed around the TOP CPU
- Switch is partitioned across TOP, package substrate, and additional tiles
  - AMD Epyc 9004 is an example, with fabric in substrate and DDR memory controllers on separate tile.

UpDown System Design





UpDown Node

- UpDown System: 16K Nodes, 524K accelerators, 33.5M Lanes
  - 67 PetaIPS, 3 insts/8B DRAM Load
  - DRAM: 8.4PB capacity, 153 PB/s Bandwidth (8 stacks HBM/node)
    - Sweep memory in 0.1 seconds
  - System Network (Polarstar): 4.4TB/s/node, 32PB/s network Bisection
    - Transpose system memory in 0.25 seconds
- Remarkable Input/output capability (can stream and ingest at 10x or even 100x > AGILE targets)

Network Design

Low-latency direct network

Diameter 3 topology called Polarstar (from Polarfly topology from Dragonfly)

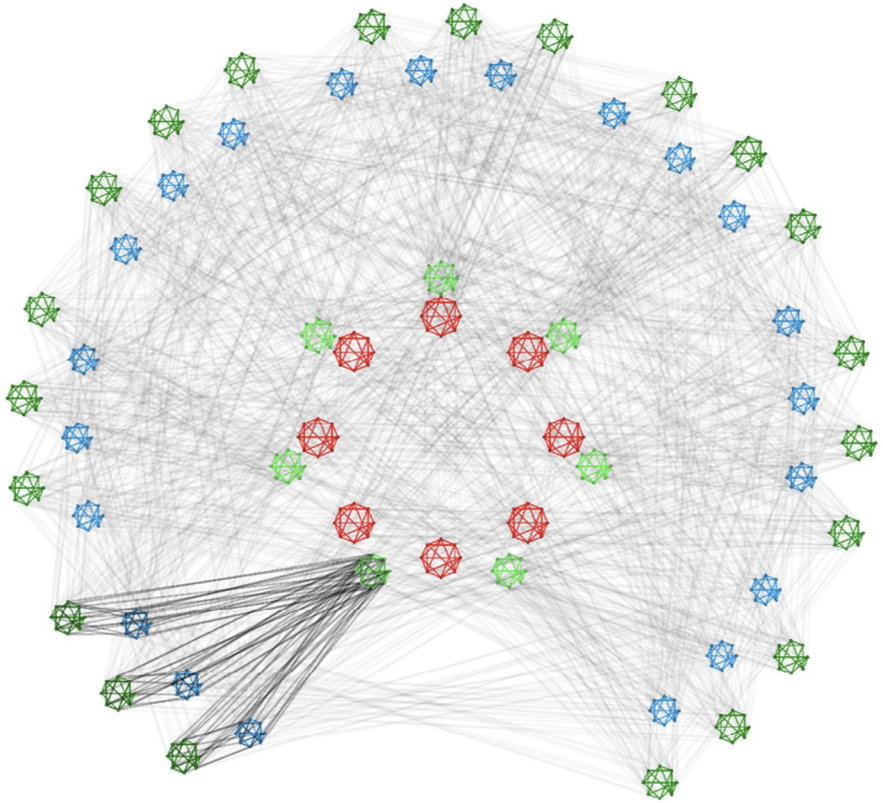
2TBps links, 2 links per node

64 PB/s all to all, 32 PB/s bisection

A mix of short (cheap) and longer optical links

High memory BW/capacity –sweep memory in 0.1 seconds

High network capability – transpose memory in 0.25 seconds



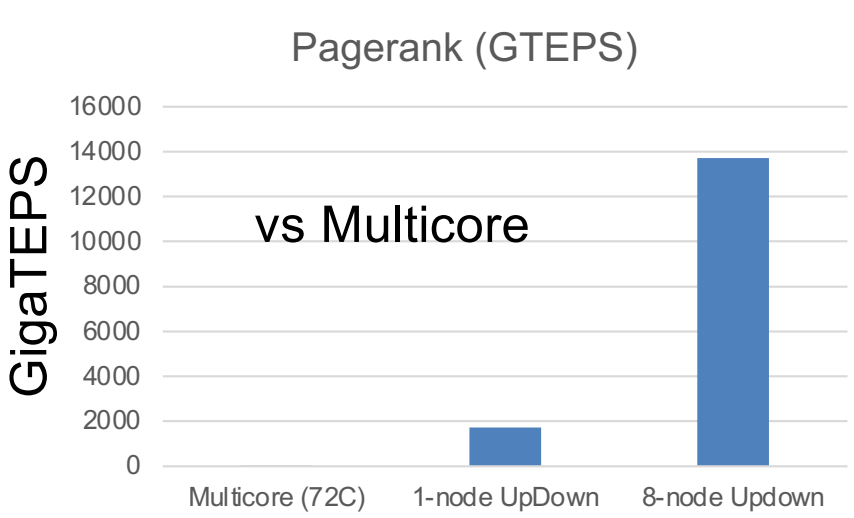
System Capabilites in Context (vs Aurora)

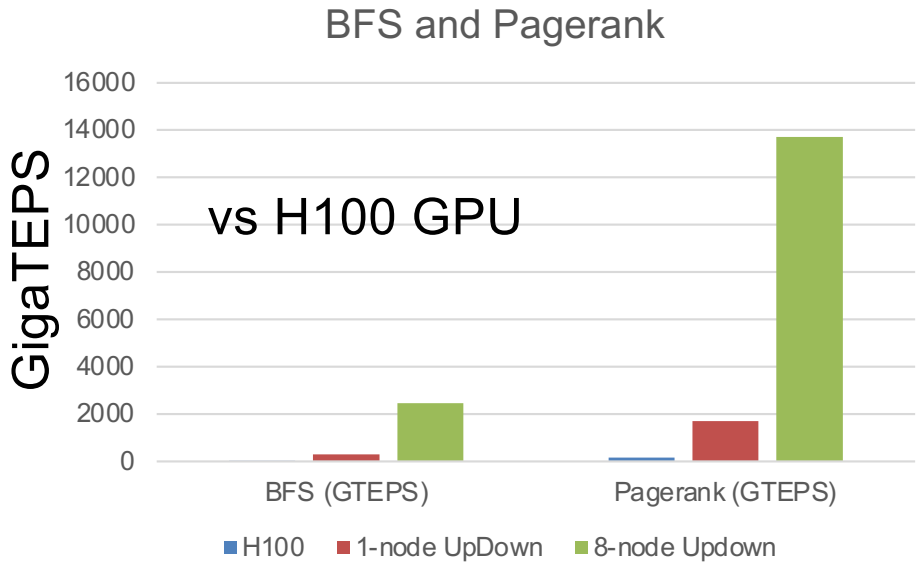
	Nodes	Sockets	System Power	Node Injection	Peak Network Injection	Per-Node Bisection	System Bisection
UpDown System	16,384	16,384	9.4 MW	4 TBps	64 PBps	2 TBps	32 PBps
Aurora System	10,624	84,992	55+ MW	0.2 TBps	2.12 PBps	0.069 TBps	0.69 PBps
Ratio (UD/Aur)	1.6	0.19	0.17	20	30.2	29	46.4
Ratio (Aur/UD)	0.64	5.2	5.85	0.05	0.033	0.035	0.022

Table 3: Comparing UpDown and Aurora

- UpDown is a communication bandwidth and global memory bandwidth supercomputer
- Network BW: 100x higher per socket, 250x higher network bisection (iso-power)
- Global Memory BW: All-to-All 64 PB/s, 4.4TBps/node (vs 9.3TBps local)

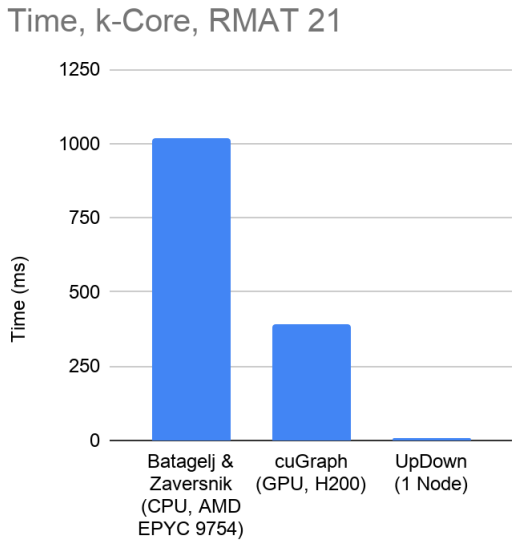
UpDown vs Multicore and GPU

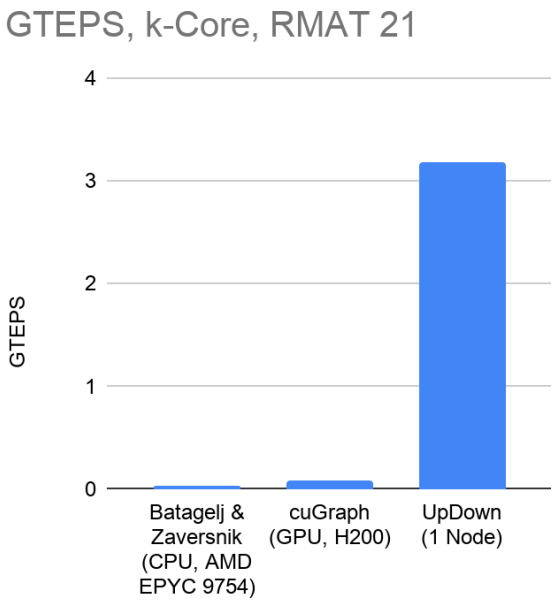




- 1N: 100x Single 72C CPU; 8N: 800x
- 1N: 8-11x Single GPU; 8N: 65-91x
- => Scalable, superior performance at 1, 8, 64… and 1000-node scale
- … and high-level programming and scaleup

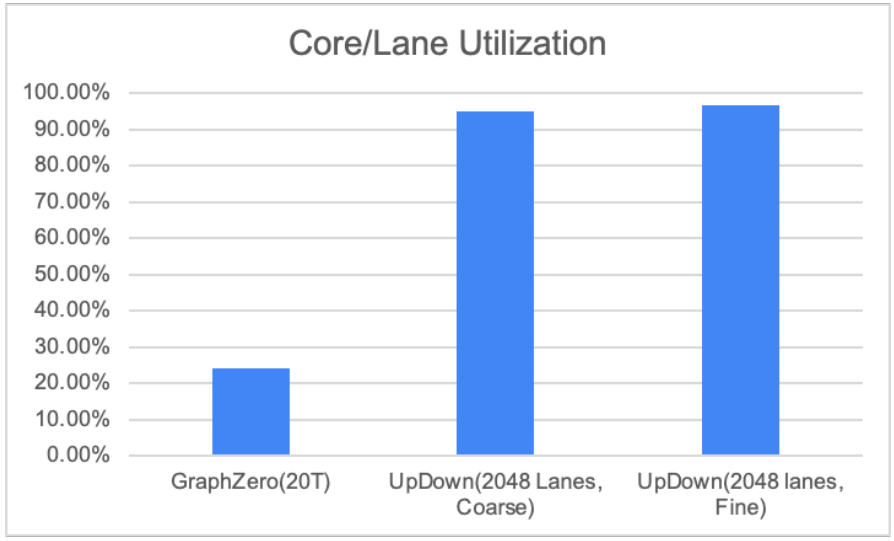
UpDown Performance: K-cores

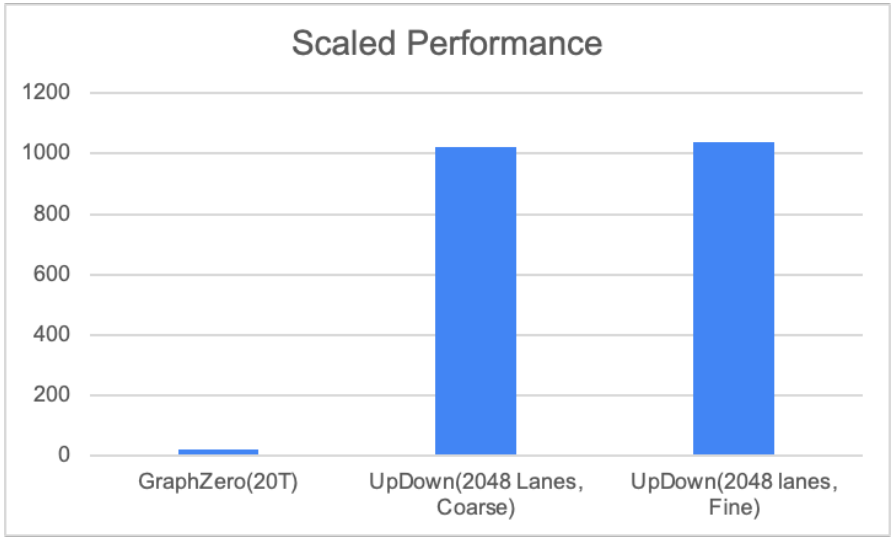




- RMAT 21: highly skewed graph of ~2M vertices, STRONG scaling
  - >100x CPU, 20x GPU
- Performance scales up with additional nodes, Throughput also!

Comparing Traditional Cores and UpDown



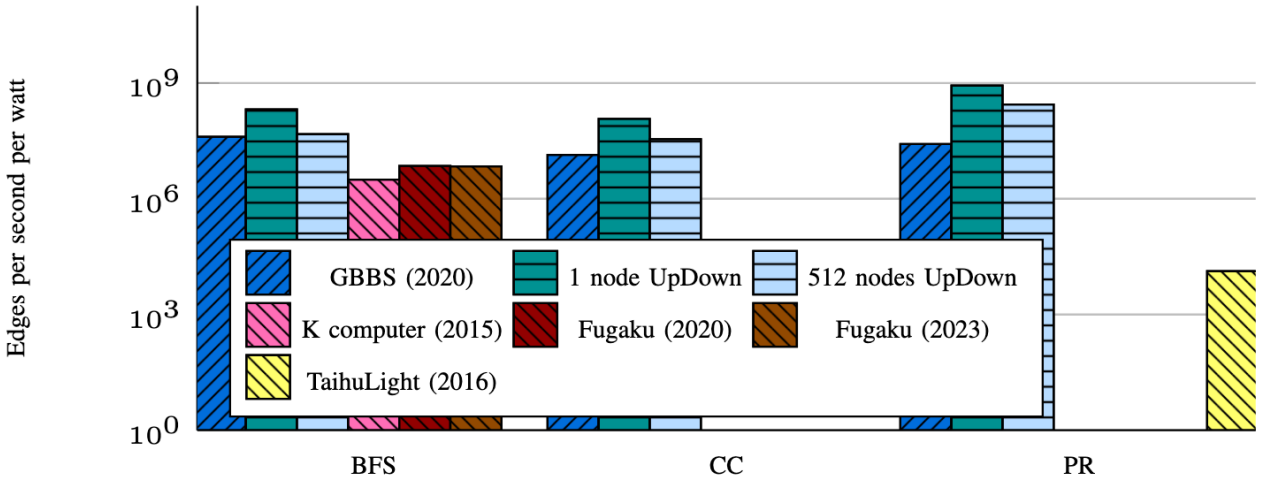


- 20T Multicore, best available software; TC on RMAT-25
- Core execution is memory-latency, branch prediction limited, producing poor issue slot utilization
- UpDown fine-grained threads produce much higher utilization; ISA 20% more efficient
- ~50x higher performance with ~1/3rd the area

Outperformance on Many Other Benchmarks

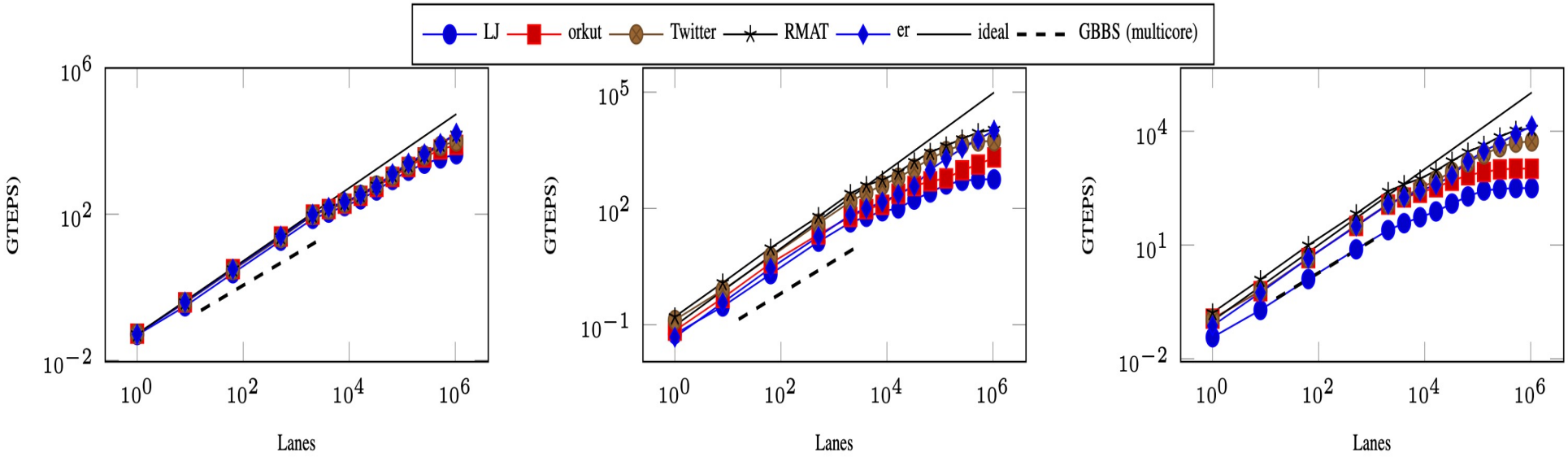
- Breadth-first Search
- Pagerank
- Connected Components
- Strongly Connected Components
- K-cores
- K-truss
- Triangle Count
- Louvain Modularity (community detection)
- Also
  - Bioinformatics Assembly
  - Streaming Graph Queries (low latency)
  - Graph Inference and Prediction
- …

UpDown has superior Power Efficiency



- Beats small-scale multicore systems (and scales up!)
- Beats large-scale parallel supercomputer systems

Scalable, High, Absolute Performance (512N, 1M lanes)



(a) PR (b) CC (c) BFS

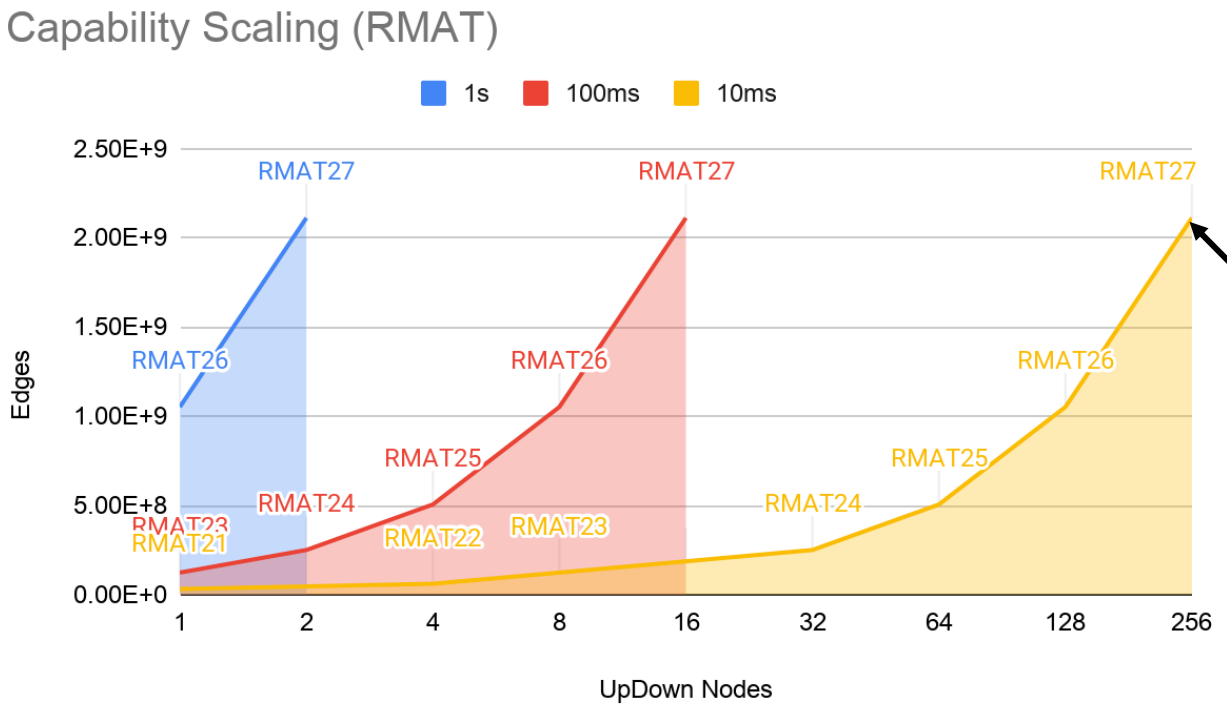
up to 186K speedup
up to 13,960 GTEPS

up to 314K speedup
up to 16,433 GTEPS

up to 216K speedup
up to 9,821 GTEPS

RMAT 28 graph is about 250M vertices (small)

Real-time Performance (online): K-cores

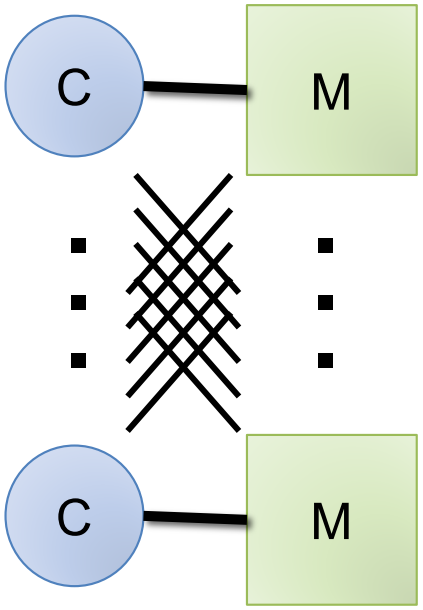


Meets 10ms deadlir
on RMAT27, w/ 256

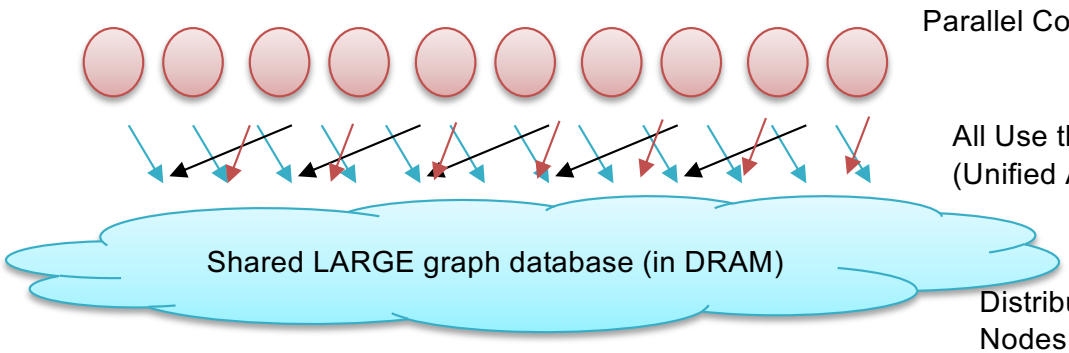
- UpDown Performance and Scaling under deadline

UpDown is a Global Memory Machine

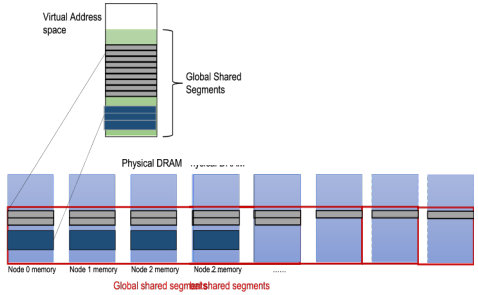
- Global Addressing
- Memory latency
  - Local: 150ns
  - Global: 1,100ns



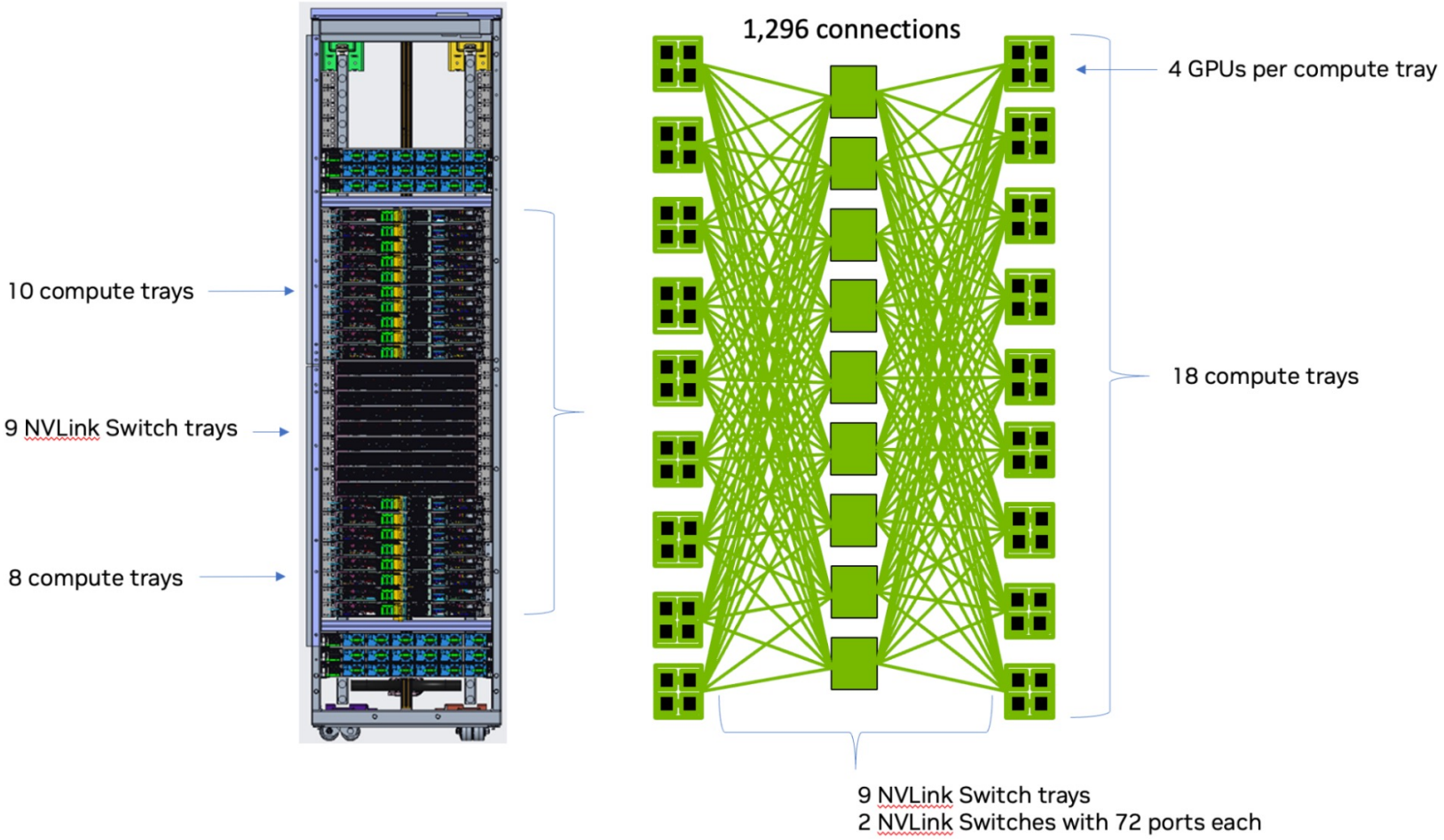
Global Addressing; Flexible Application Control



- Global address space – uniform naming of data independent of location
- Flexible, HW-supported control for mapping address space to distributed physical memory structures (access, locality)

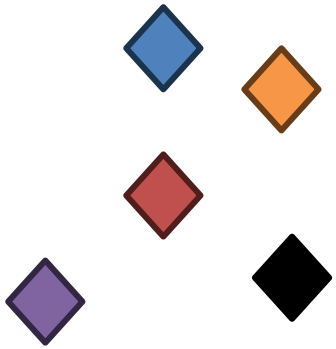


There are other systems like this….

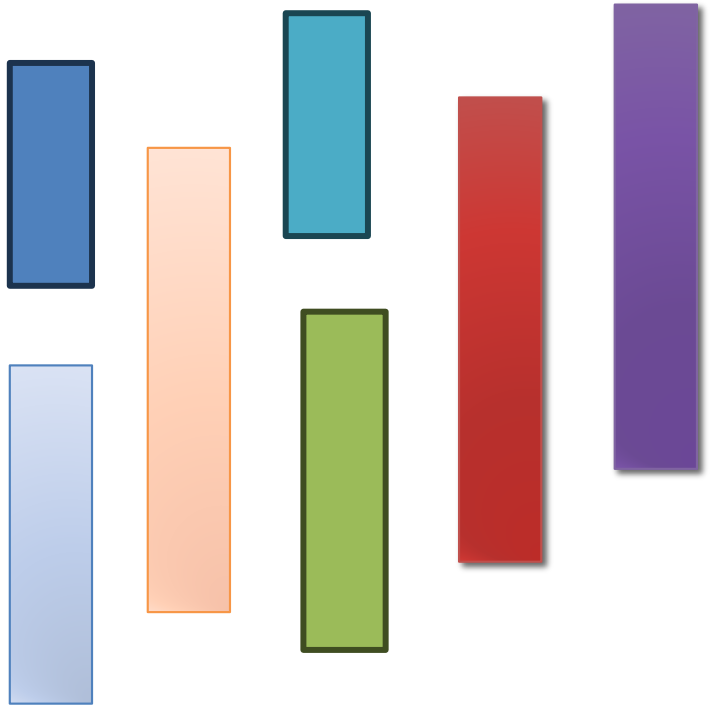


- Nvidia's NVL72
  - Global address space, remote accesses at 64B granularity

Shared Datasets are used by jobs



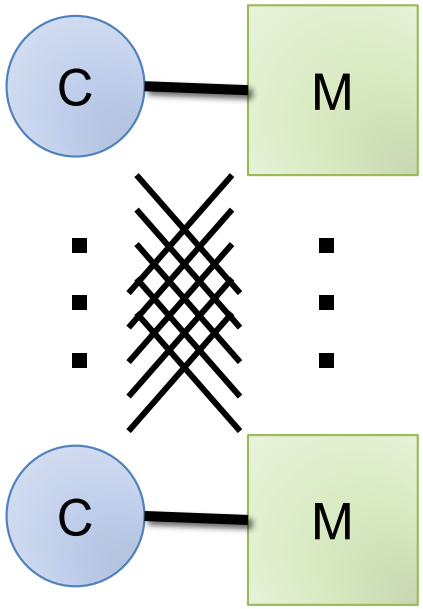
Job: <Ni, Dsi, Dsj, …>



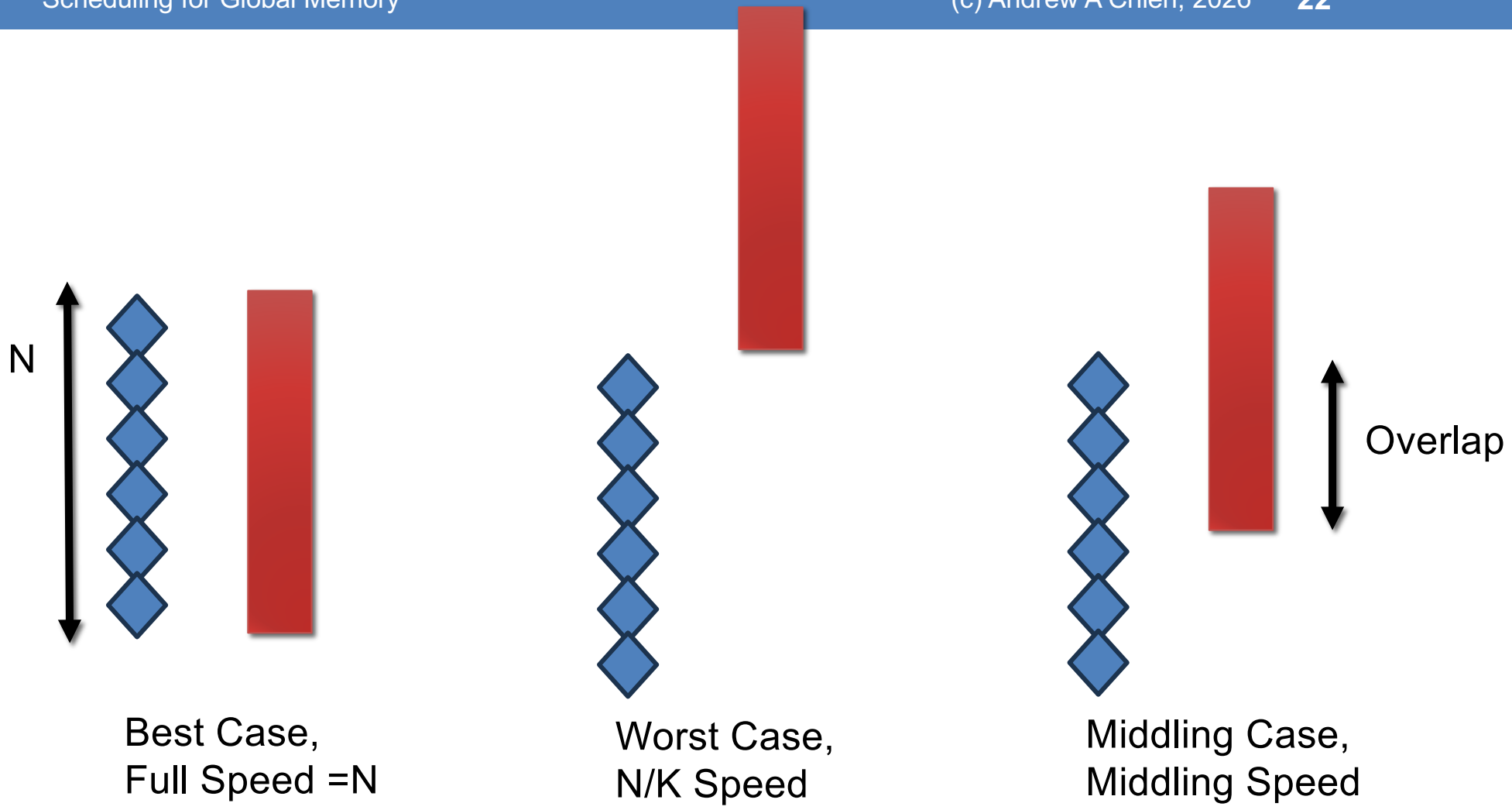
Dataset: <Sni, Eni>

- Job: number of compute nodes, datasets used
- Dataset: start node, end node

Thinking about the network (NOT)



- Simplest model is a flat network (no topology); ignore topology
  - Goal of hardware design is to neutralize topology in a flat memory machine
  - Updown: flat network to 16K nodes; NVL72: fat tree with flat bandwidth



- Overlap of Job compute nodes and datasets determines the execution speed (memory bandwidth is the key)
- Middling: perf = N*Overlap/N
- K=2 in UpDown, 8 in a typical GPU cluster

Scheduling Problems

- How to place the shared datasets? (given a set)
  - Fundamentals (no knowledge)
  - Based on statistics of the Job's use
  - Each dataset – how many nodes to spread over (there's a minimum), and how to align?
- Given dataset placements, how to schedule?
  - When to wait?
  - Where to schedule the jobs?
- Jobs that use

Simple Scheduling (Job use 1 dataset)

- Pack each of the collections Di across a subset Pi of the Total nodes, job size = dataset partition size
  - Try to schedule with affinity, if cannot, then wait
  - When job runs it achieves Ni performance
  - => system utilization, job wait, short runtimes

- Spread all the data, uniformly; Schedule the jobs
  - Traditional scheduling view, no locality consideration
  - Each job using
    - Ni nodes out of Total
      - Achieves Ni / K compute rate, or maybe more precisely
      - Ni / (K* (Total-Ni)/Total)
  - => higher system utilization, not job wait, completion time

Jobs that uses multiple datasets

- Datasets are packed, but overlapping
  - Try to schedule jobs at the overlap of datasets (if it exists)
  - If not, schedule aligned with one of the datasets; runs mix /1 and /K
  - => mix of wait, and job runtime

- Spread all datasets
  - Normal job scheduling
  - Jobs run at 2/K, 3/K speed?
  - => mix of wait and job runtime

Related Scheduling work

- CCoherent Shared Memory/Multicore
  - Not bandwidth sensitive?
- NUMA scheduling?
  - Sort of, but typically not persistent datasets (done for threads within a job)
- Supercomputer Parallel Job scheduling
  - No persistent datasets, no sharing
- Others?

More complicated Questions

- For a given “response time”, say no wait time – interactive
  - What system utilization can be achieved?
- More complicated models of application performance as a function of “overlap”?
- More complicated models of application performance as a function of “multiple dataset use” and perhaps “overlap”?
- Topology. (but I don’t like this)

BACKUP

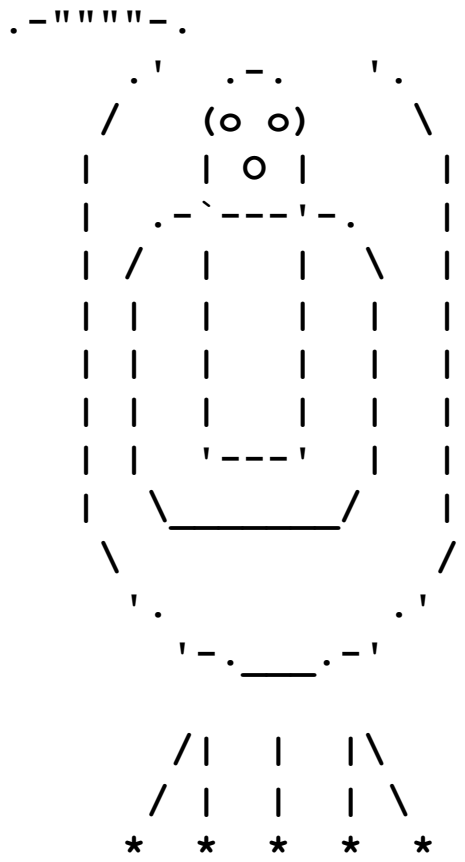
High Bandwidth Memory and Optical Networks

- Memory BW
  - HBM2e (480GB/s), HBM3e (1200GB/s), HBM4e (4100GB/s)
  - GPU's (and UpDown's) have 10-30TB/s per node

- Network BW (latency falls… but)
  - 4.0 H100: 18x50GB/s (900/2)
  - 5.0 B100: 18x100GB/s (1800/2)
  - 6.0 Rubin: 18x200GB/s (3600/2)

  - Off node bandwidth trails behind

An AI Scheduler



- According to ChatGPT

```
      /|  |  |\
     / |  |  | \      <-- floating holographic arms
    *  *  *  *  *

[09:00] Meeting
[11:30] Lunch
[14:00] Deadline
[✓] Tasks optimized

<< PROCESSING SCHEDULE >>
```

AI vs Algorithmic Schedulers

- Claude:
  - “Algorithmic schedulers excel when you need guarantees over statistics, speed over sophistication, or transparency over adaptability, classical algorithms remain the right tool.”
  - “AI schedulers are powerful optimizers, but optimization and correctness are different things.”
- Disruptions: “Out of distribution” scenarios
  - Dynamic Resource scenarios, Job distribution/burst changes
  - AI can adapt to new statistics – after some period of time
  - Algorithmic schedulers are often the default because their behavior is robust and characterizable