A Synthetic Task Model for HPC-Grade Optical Network Performance Evaluation

IA^3 Workshop, Denver, November 17th, 2013

Sébastien Rumley, Lisa Pinals, Keren Bergman

Lightwave Research Laboratory, Columbia University

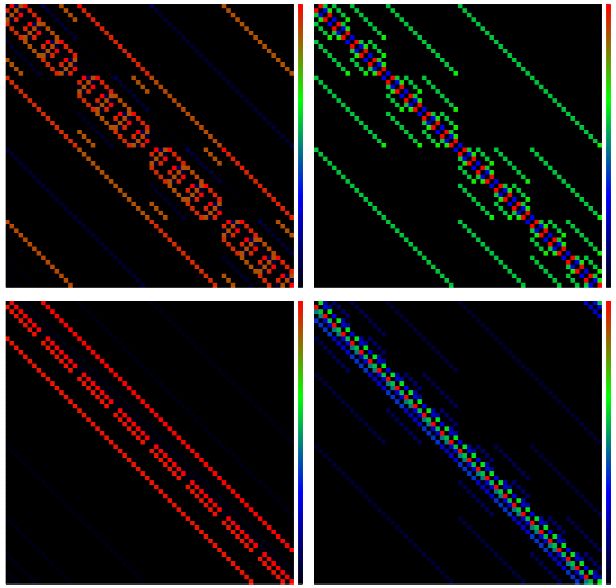
Gilbert Hendry

Sandia National Laboratories

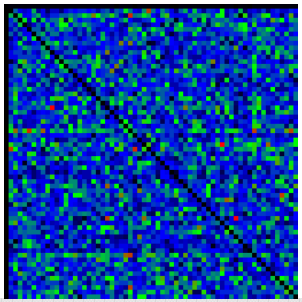
Contact: sr3061@columbia.edu

Motivation

- Increased parallelism translates to more data-movement
- For a given application, traffic is predictable[1][2]



- Other applications are less predictable
  - Graph exploration, operations on sparse matrices, etc.
- HPC also targets these applications.



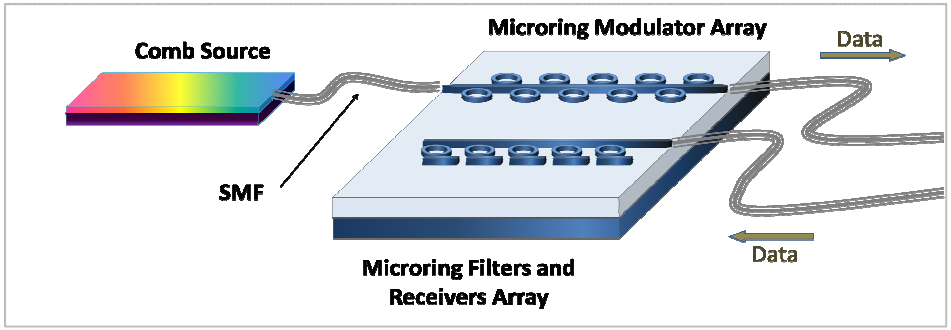
Can we devise a network that could be good for both regular and irregular apps?

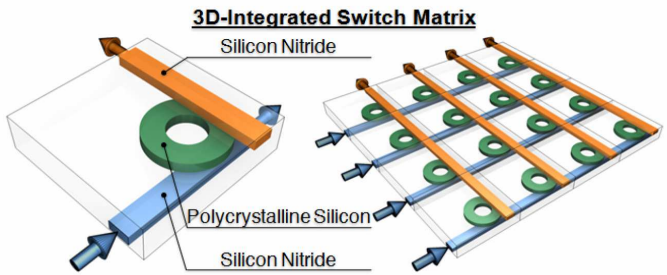
[1] Shalf, J., et al. Analyzing Ultra-Scale Application Communication Requirements for a Reconfigurable Hybrid Interconnect. in *SuperComputing* (2005)
[2] Barker, K.J., et al., On the Feasibility of Optical Circuit Switching for High Performance Computing Systems, in *SuperComputing* (2005)

Optics emerging as a hot topic for SuperComputing

- Potential for ultra **high bandwidth** density
  - Also over large distances
  - Low power
- Potential for **low latency** communication
  - Cut-through transmission
  - Optical switching
- Tight integration with **computing resources** using Silicon Photonics

High bandwidth-density, low-power WDM Transceivers





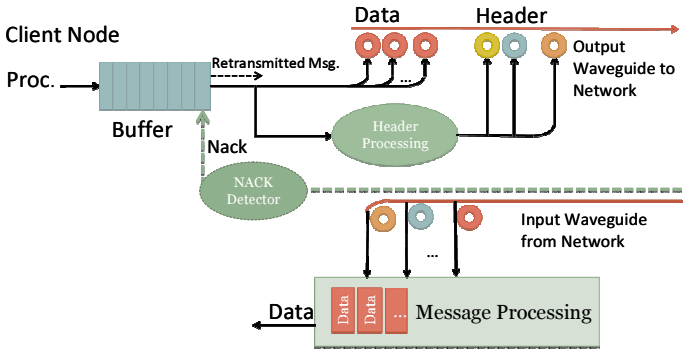
Fast-reconfigurable photonic core switches

What type of optics-enabled network is suited for both regular and irregular apps?

Traditional approach

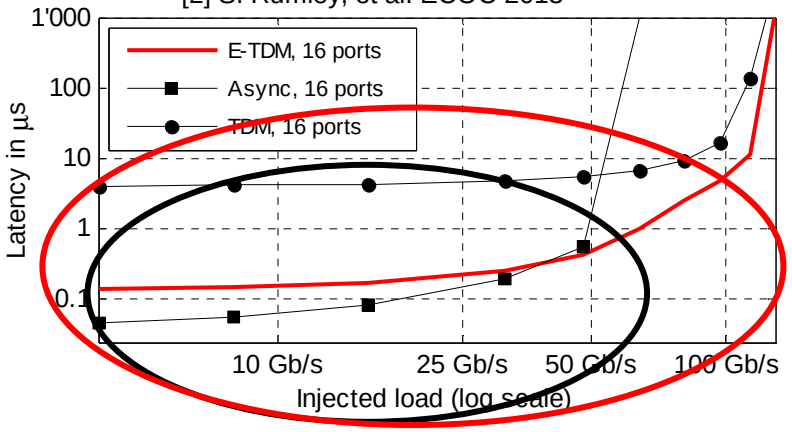
1. Devise a networking concept
   - Hardware links, interfaces, switches
   - Protocols for orchestration and allocation

[1] G. Dongaonkar, et al.
IEEE Optical Interconnects 2013



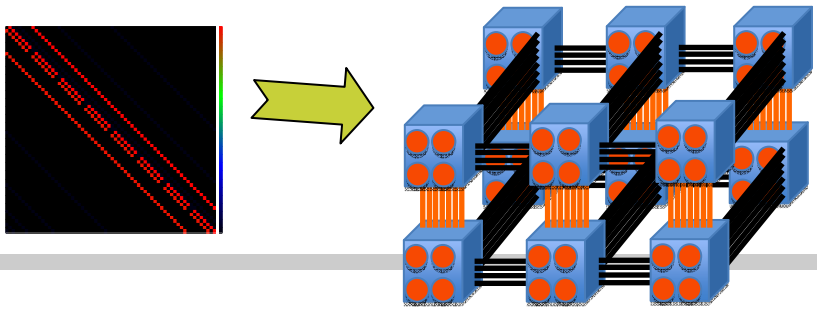
2. Model, analyze and optimize the network capabilities
   - Effective bandwidth
   - Cost (equipment, energy) for bandwidth
     - Modeling through random traffic

[2] S. Rumley, et al. ECOC 2013



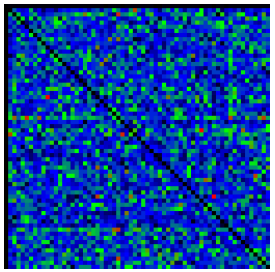
Typical operation range

3. Put the bandwidth where the traffic is
   - Dimension the network to meet traffic demands



Traditional approach not applicable

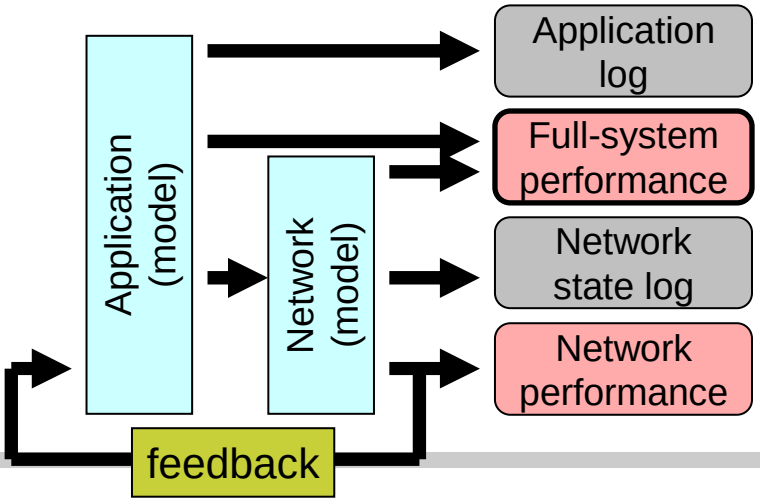
- Put bandwidth where the traffic is – but where is the traffic?
  - Can we at least quantify the requirements?



- What is the impact of latency?
  - Especially in an optical networking context
    - Lack of buffers might produce “networking hiccups”

→ Need to incorporate the **application** in the **network** modeling process

- Full-system simulation?
  - Execution of real code
- Coarse-grained simulation?
  - Execution of (simplified or not) code
- Can we further simplify the application while keeping the feedback?



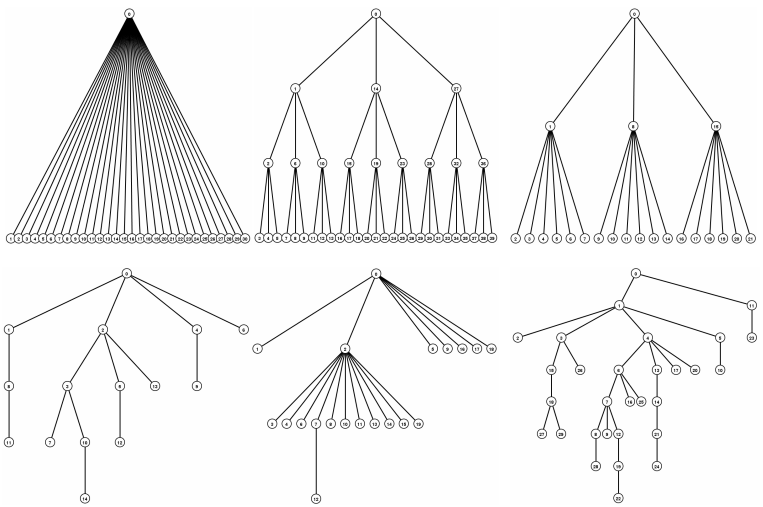
Proposal: Random **tasks** instead of **random traffic**

- Application reduced to its simplest form
  - **Computation phases** characterized by an execution time
  - **Communication phases** described by a byte amount

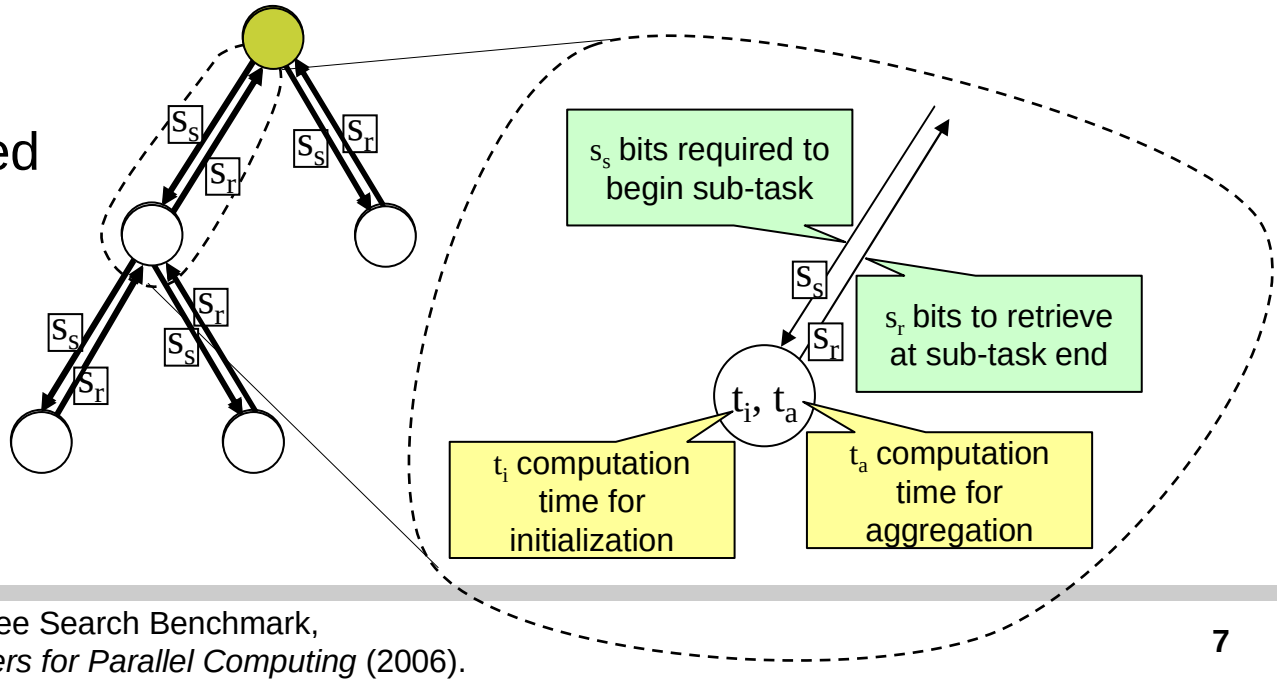
  - Interdependent computations and communication
    - A **computation** depends on one or more message arrivals
    - Messages are issued to the network simulator only upon a **computation** phase completion
    - Model tasks as directed acyclic graphs (DAGs)

- Mimic the workload as a **random sequence** of (**random or not) DAGs**
  - Exponentially distributed inter-arrival times
    - Small tasks occurring frequently → Data-Center
    - Large tasks occurring sporadically → SuperComputer

Tree based task generation



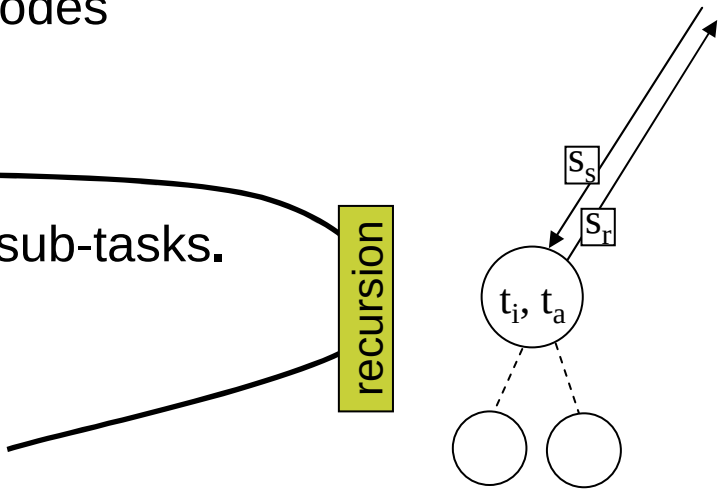
1. Start from a tree structure
   - Can be obtained through deterministic or random generation
   - Random trees for benchmarking proposed in the past[1]
2. Assign four (random or predetermined) values to each vertex
3. "Execute" the associated job described by the DAG



[1] Olivier, S., et al. UTS: An Unbalanced Tree Search Benchmark, in *Workshop on Languages and Compilers for Parallel Computing* (2006).

Scheduler and tasking model

- Central scheduling: one of the nodes hosts the scheduler
  - **This node is not available to execute tasks**
  - **Only** entry point for arising tasks
  - Keeps a list of activity flags for computing nodes
- When a task is assigned to a node
  - Receives descriptor (of size s_s)
  - Initialization phase (for t_i). Discovery of the sub-tasks**.**
  - Asks the scheduler for helpers
    - Which may or may not be allocated
    - Task descriptors (of size s_s) sent to helpers
    - Awaits results (of size s_r)
  - *Some sub-tasks kept locally (avoid stalling while waiting)*
  - Upon sub-task results collection (locally computed or received)
    - Aggregation phase starts (for t_a)
    - Task results are sent back
    - Inactivity is announced to the scheduler

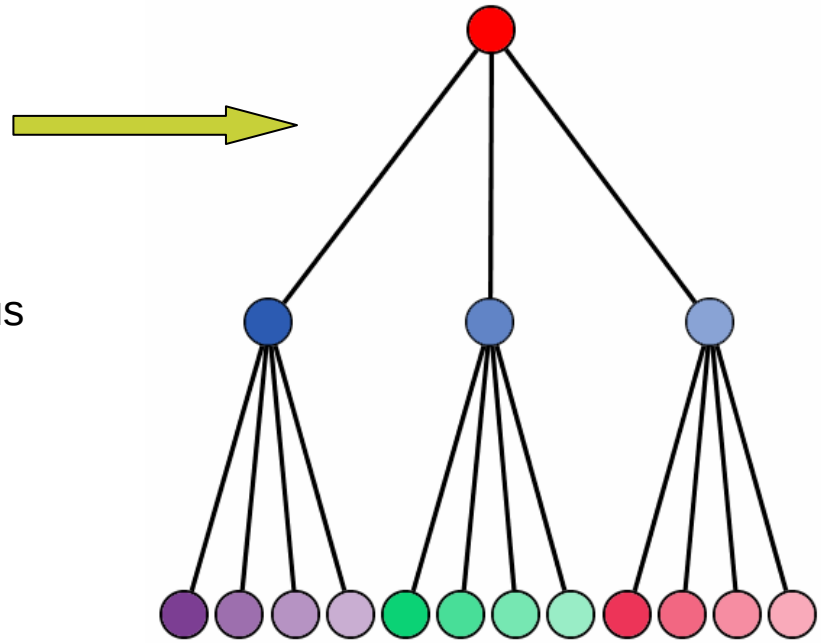


Implementation - verification

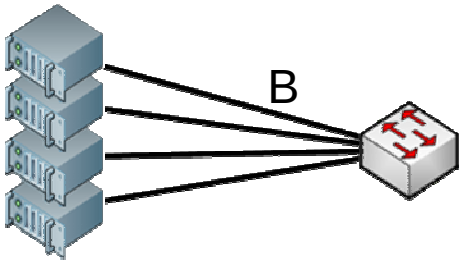
- Job generators, task runtimes (scheduler and clients) implemented on top of the LWSim simulator[1]

- Implementation verification:
  - Completion time for a single job
    - Execution time for each vertex: 20µs
    - Sequential execution time: 340µs
    - Lower bound for parallel execution: 60µs

    → **Maximum speed-up: 5.66**

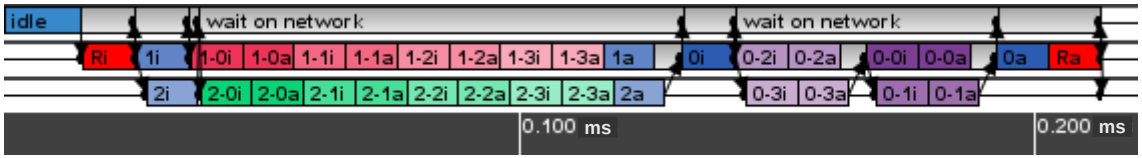


  - System with N computing nodes connected with simplistic star network with access bandwidth B

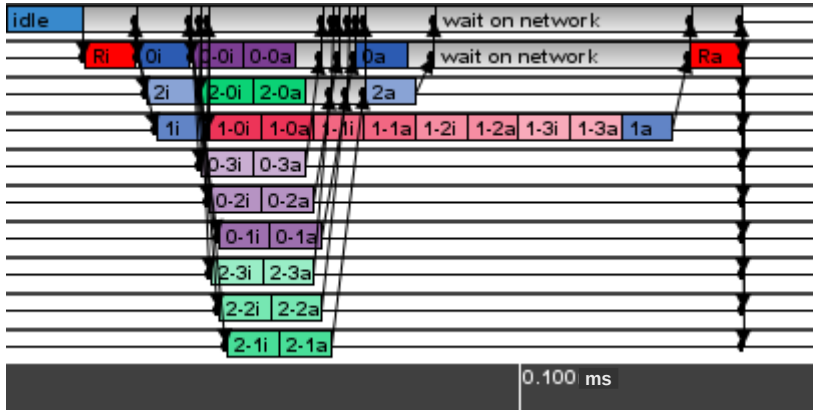


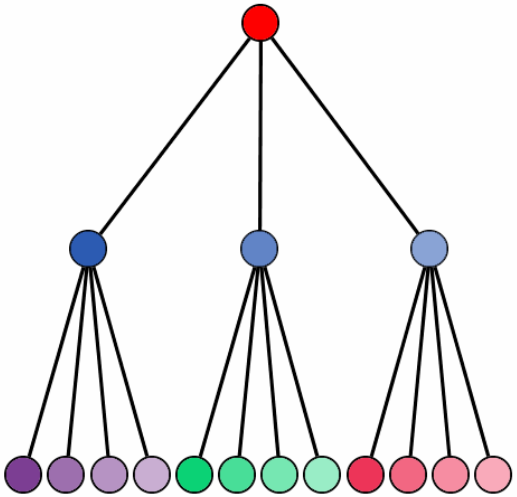
[1] S. Rumley, R. Hendry, K. Bergman, "Fast Exploration of Silicon Photonic Network Designs for Exascale Systems [position paper]," ASCR Workshop on Modeling and Simulation (Sep 2013).

N=3

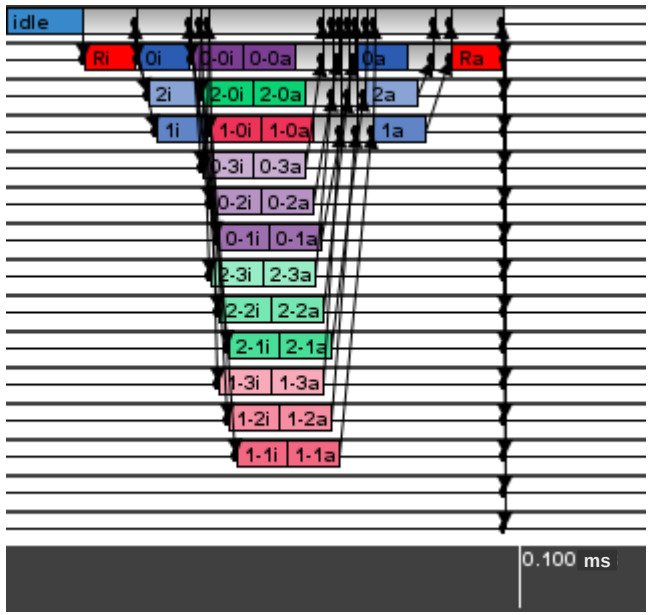

idle
wait on network
wait on network
Ri
1i
1-0i
1-0a
1-1i
1-1a
1-2i
1-2a
1-3i
1-3a
1a
0i
0-2i
0-2a
0-0i
0-0a
0a
Ra
2i
2-0i
2-0a
2-1i
2-1a
2-2i
2-2a
2-3i
2-3a
2a
0-3i
0-3a
0-1i
0-1a
0.100 ms
0.200 ms

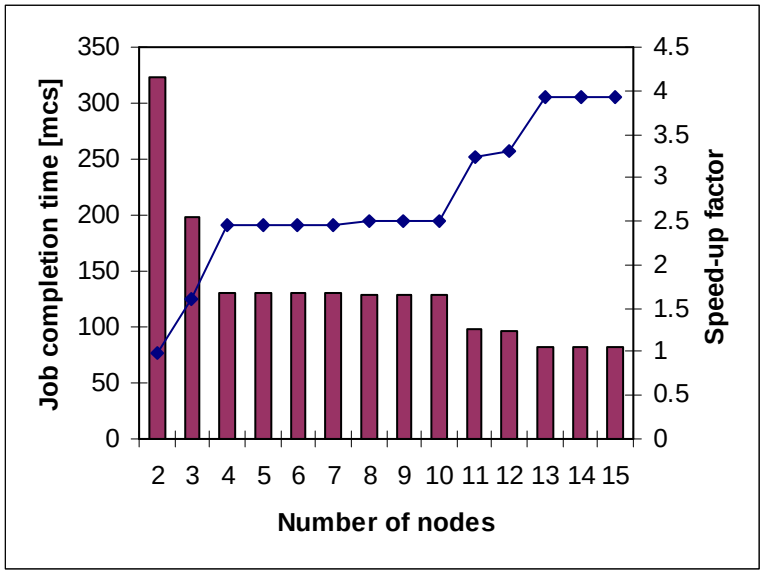
N=10


idle
wait on network
wait on network
Ri
0i
0-0i
0-0a
0a
Ra
2i
2-0i
2-0a
2a
1i
1-0i
1-0a
1-1i
1-1a
1-2i
1-2a
1-3i
1-3a
1a
0-3i
0-3a
0-2i
0-2a
0-1i
0-1a
2-3i
2-3a
2-2i
2-2a
2-1i
2-1a
0.100 ms



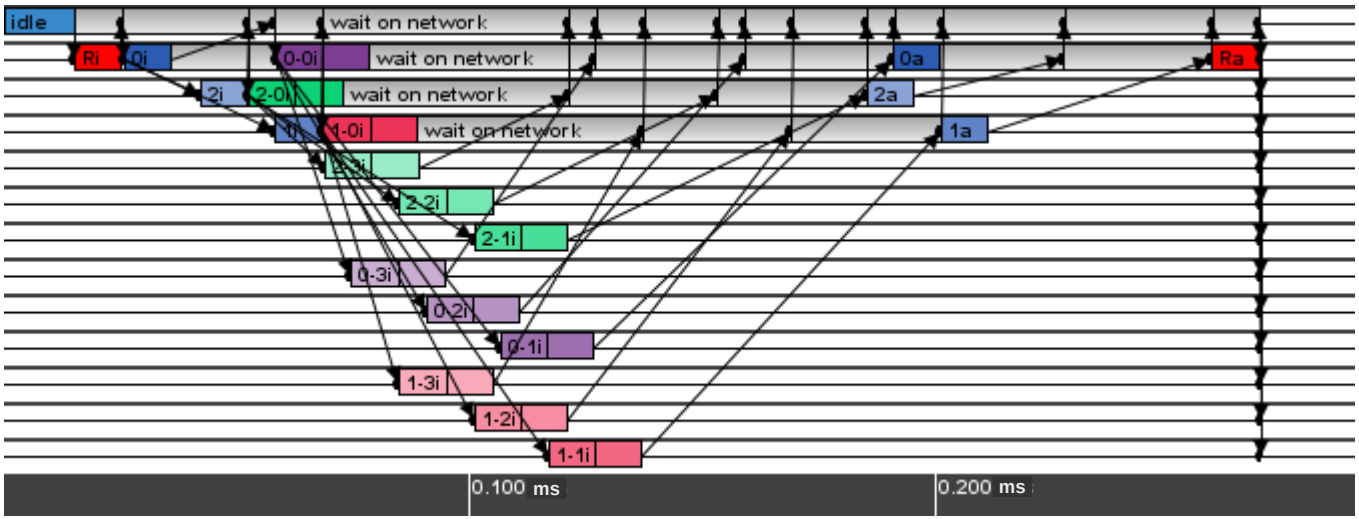
N=15


idle
Ri
0i
0-0i
0-0a
0a
Ra
2i
2-0i
2-0a
2a
1i
1-0i
1-0a
1a
0-3i
0-3a
0-2i
0-2a
0-1i
0-1a
2-3i
2-3a
2-2i
2-2a
2-1i
2-1a
1-3i
1-3a
1-2i
1-2a
1-1i
1-1a
0.100 ms


Job completion time [mcs]
350
300
250
200
150
100
50
0
Speed-up factor
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
2 3 4 5 6 7 8 9 10 11 12 13 14 15
Number of nodes

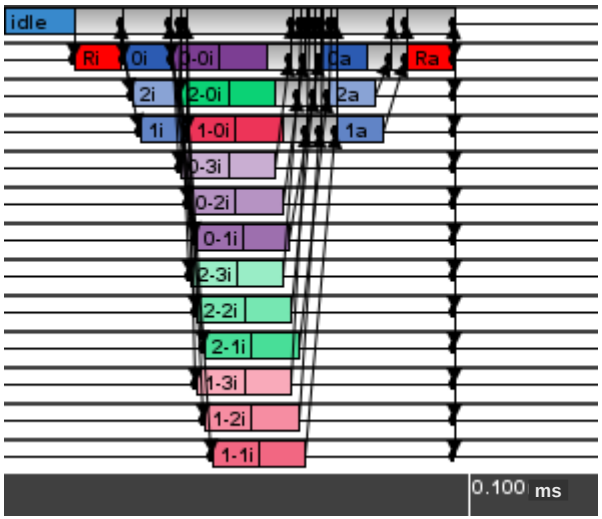
Impact of bandwidth

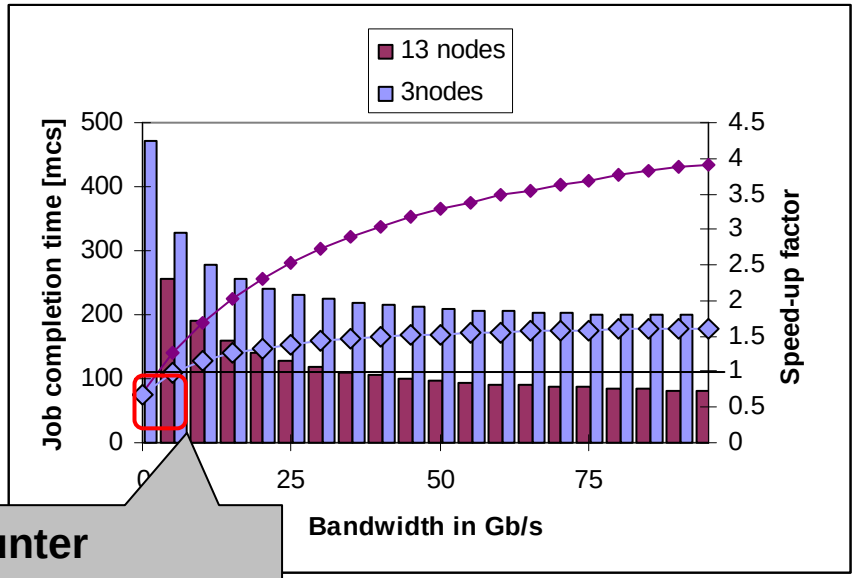
10 Gb/s



13 nodes

100 Gb/s

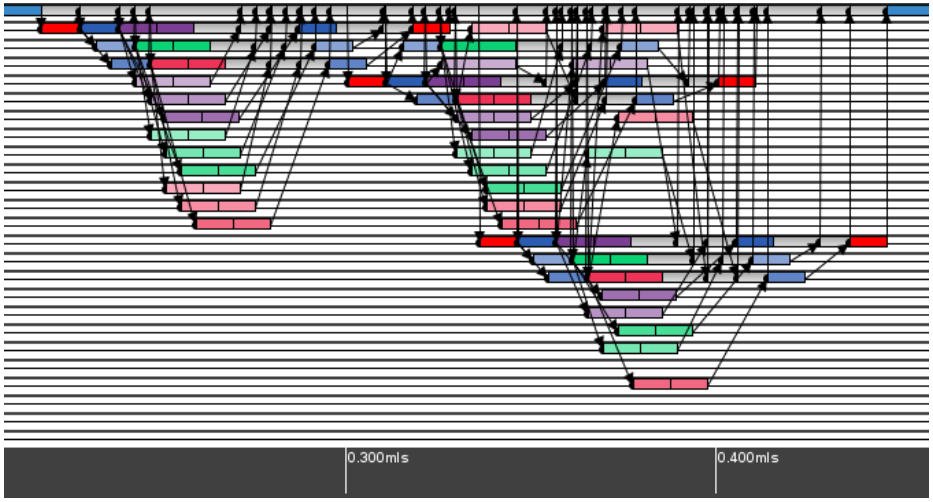


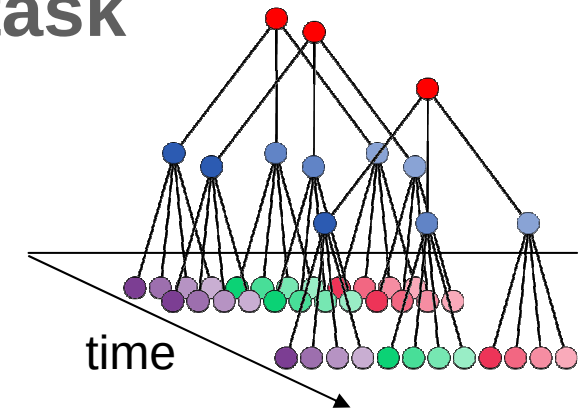


Counter productive case

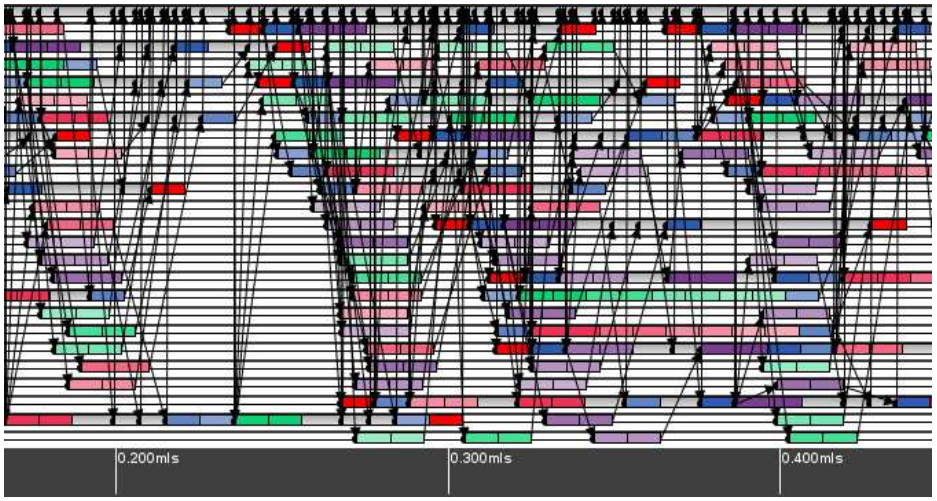
Let’s consider more than one task

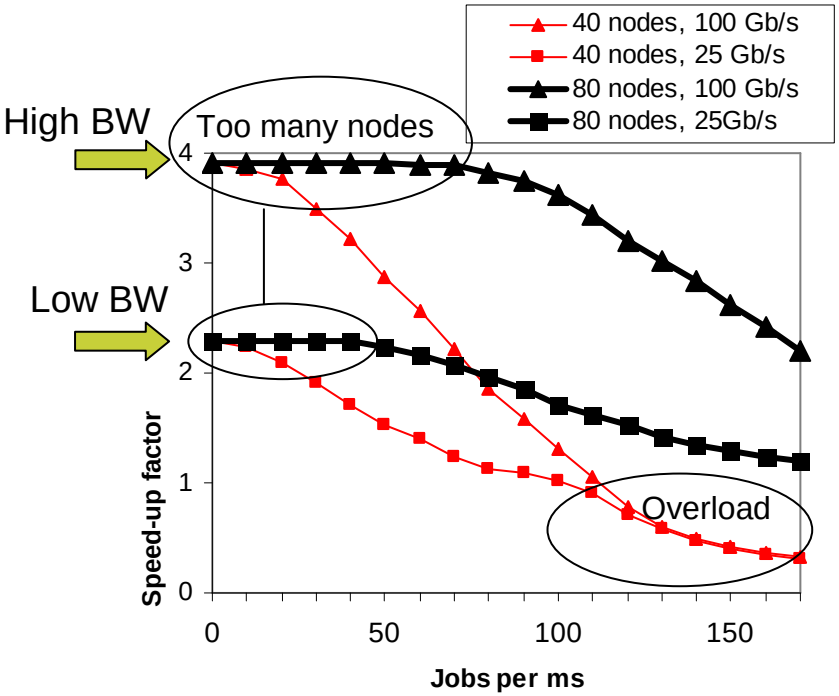
5 job/ms



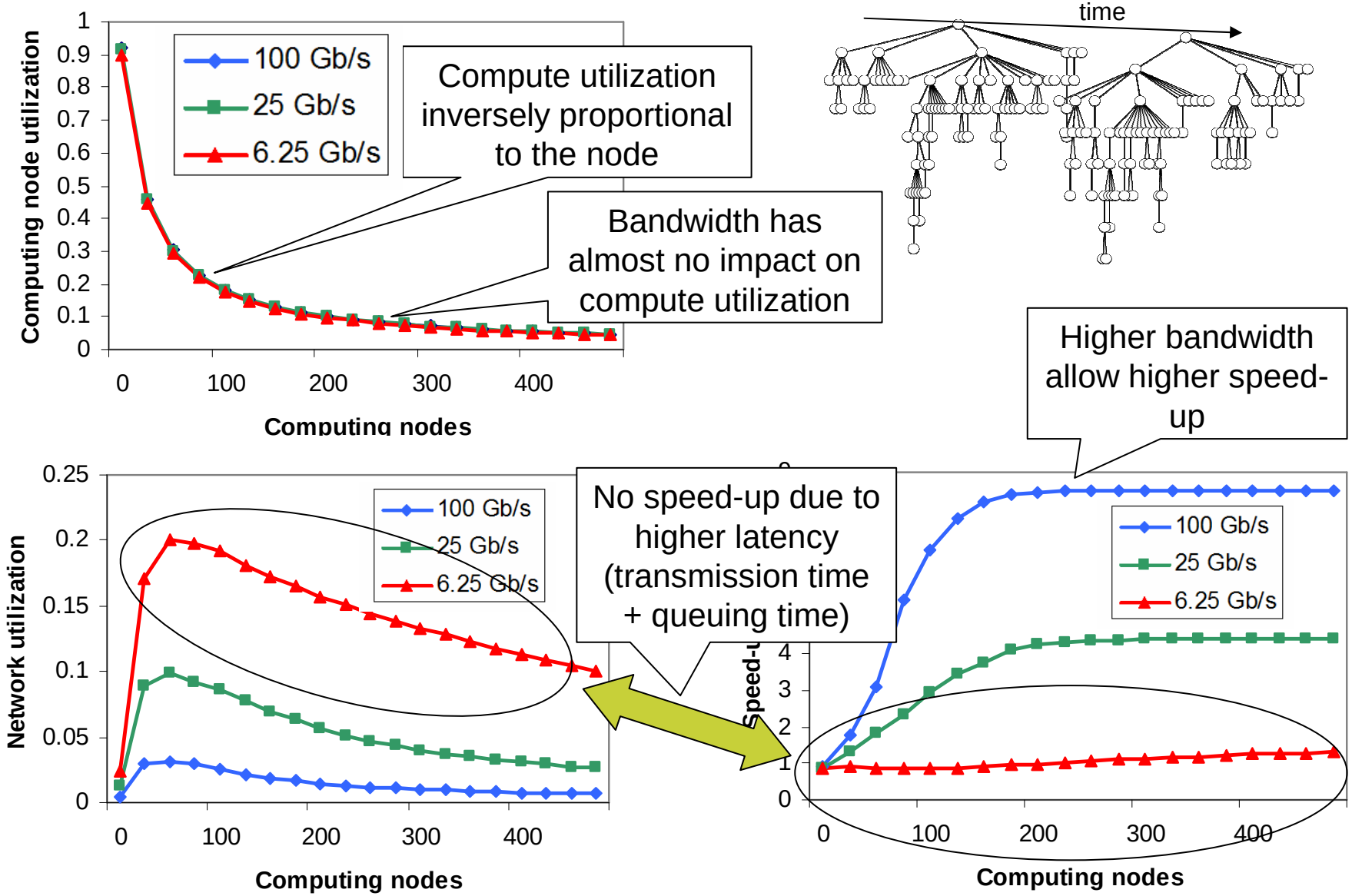


30 job/ms

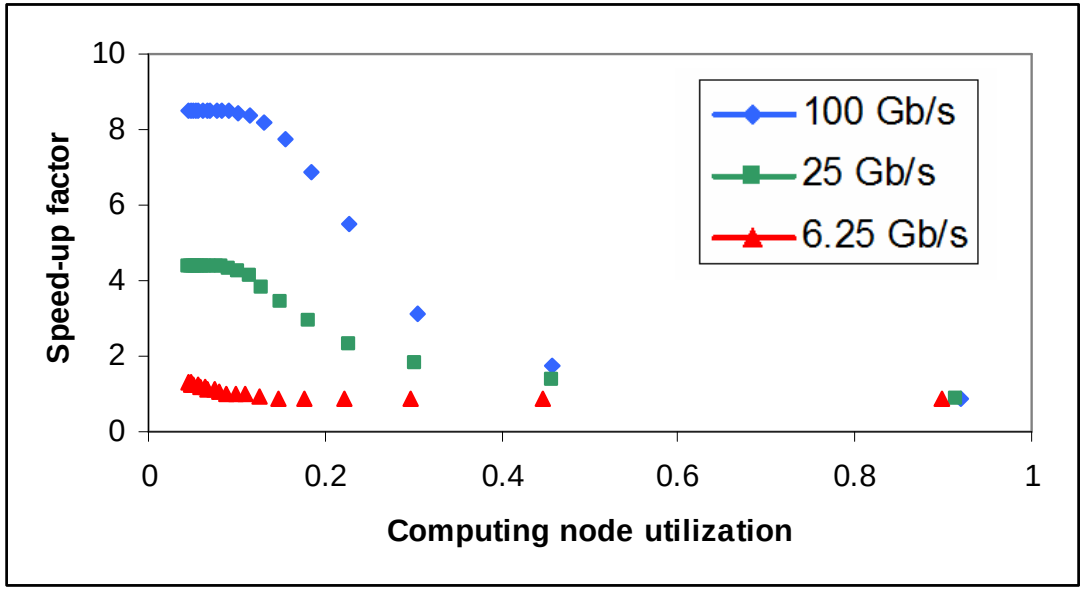




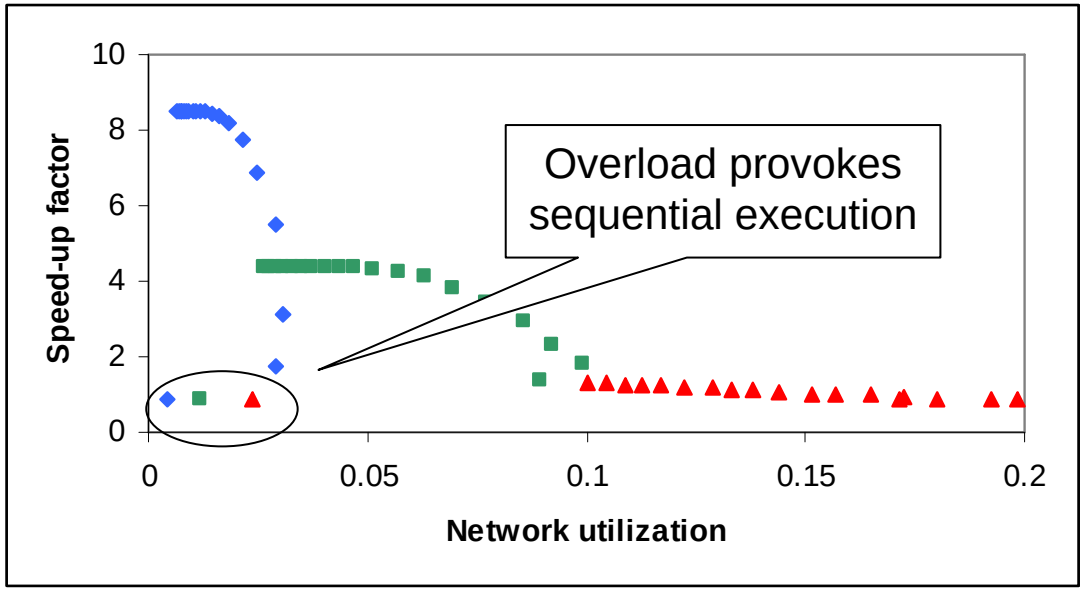
Simulation of the same sequence of tasks on different architectures



Speed-up relation with utilization



Trade-off between **speed-up** and **computing node** utilization



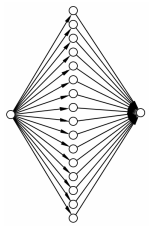
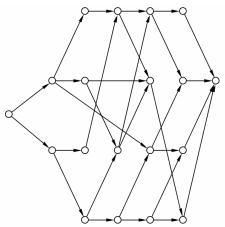
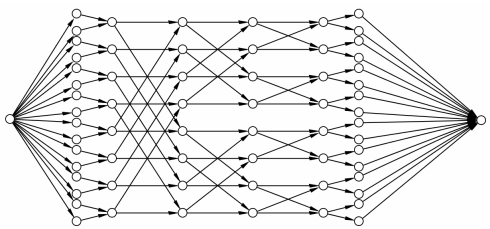
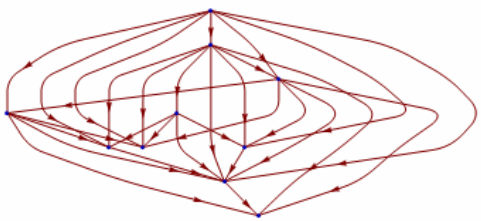
Trade-off between **speed-up** and **network** utilization

Conclusions

- Synthetic workload model generates both traffic and computing tasks
  - Easy to model, scalable
- Simulation of DAG described applications over a network
  - Random trees can be replaced by real-life DAGs
- Simulation results show that
  - Centralized scheduling is not adequate
  - Can we achieve speed-up and high network utilization
    - Not possible in the conditions tested here: high utilization means queuing, queuing means delays, delays means inefficiency.
- Room to investigate various topologies, even abstractly described
  - Optimize cluster sizes, over-subscription ratios, etc.
  - Test our **optical aware architectures** in more application-like environments

Next steps

- Main limiting factor: centralized dynamic scheduling
  - Static allocation?
  - Work stealing approach?
- Allocate bandwidth where traffic is?
  - SDN-connected scheduler?
    - Network topology configured (in advance) by the scheduler or
    - Network configuration aware scheduling
- Task generation:
  - Non-tree random pattern generation

  - **Comparison with real-life workload**

Thank you for your attention

- Any question?

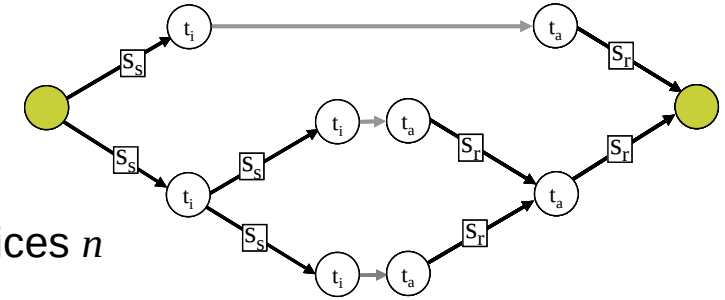
  Contact: sr3061@colubia.edu

  – Thanks to my co-authors!

Extra slides

Workload modeling

- A given tree has a communication and computation footprints F_{comm} and F_{comp}
  - F_{comp} is directly proportional to the number of vertices n and to E[$t_i + t_a$]
  - F_{comm} is also proportional to n and to E[$s_s + s_r$].
    - However, no communication arises if two sub-tasks (vertices) are executed on the same node.
    - $max(F_{comm}) = n(s_s + s_r)$ can be taken as upper bound



$F_{comp} = 4(t_i + t_a)$
$max(F_{comm}) = 4(s_s + s_r)$

- System loads can be estimated with these footprints
  - If β is the task arrival rate in [s^{-1}]:
    - βF_{comp} is the computation load in [-]
    - βF_{comm} is the communication load in [bit/s]

The computing load is unitless. It expresses the number of reference CPUs required to avoid task accumulation at 100% CPU utilization

The network system must support at least βF_{comm} [bits/s] of injection bandwidth.

Modeling – continued

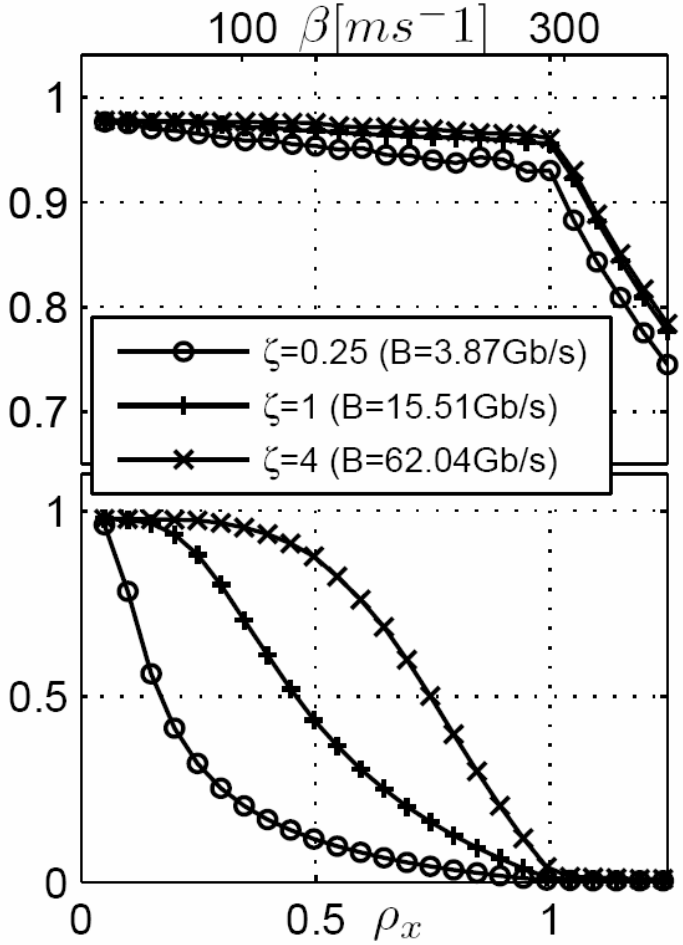
- The target system executing the workload can also be modeled
  - N **computing nodes**, each equipped with the reference CPU
  - Each node connected to the **network** with bandwidth B [bits/s]

- By relating the loads to the system size, one obtains normalized loads
  - $\rho_{comp} = \beta F_{comp} / N$ is the theoretical **computing node** utilization
    - Predicts **computing resource** utilization … except if tasks accumulate.
  - $\rho_{comm} = \beta F_{comm} / BN$ is the theoretical **network** utilization
    - Predicts **network utilization** … except if tasks accumulate or are executed sequentially

- To have $\rho_{comp} = \rho_{comm}$ one needs a bandwidth $B_{eq} = F_{comm} / F_{comp}$
  - B_{eq} is the bandwidth required if equal utilizations are targeted
  - We can relate any bandwidth to this "reference bandwidth"
    - $B/B_{eq} = B/(F_{comm}/F_{comp}) = \zeta$
    - **ζ expresses the asymmetry of a system** (large ζ → over-dimensioned network)

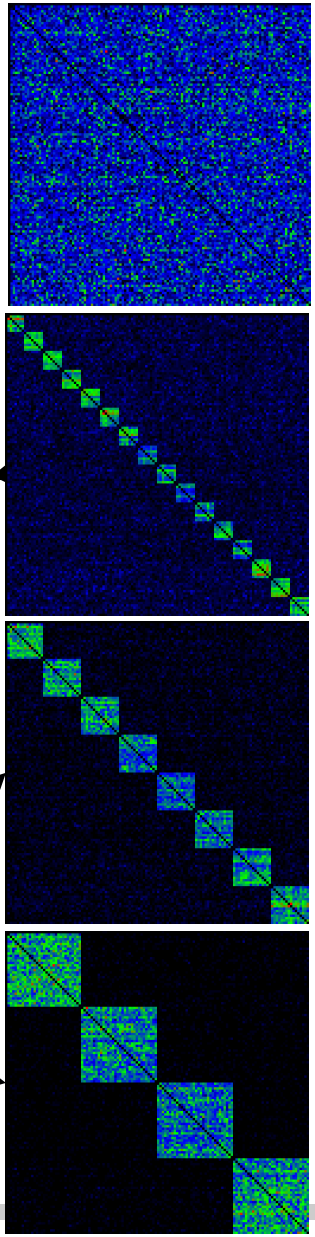
Measurement vs. predictions

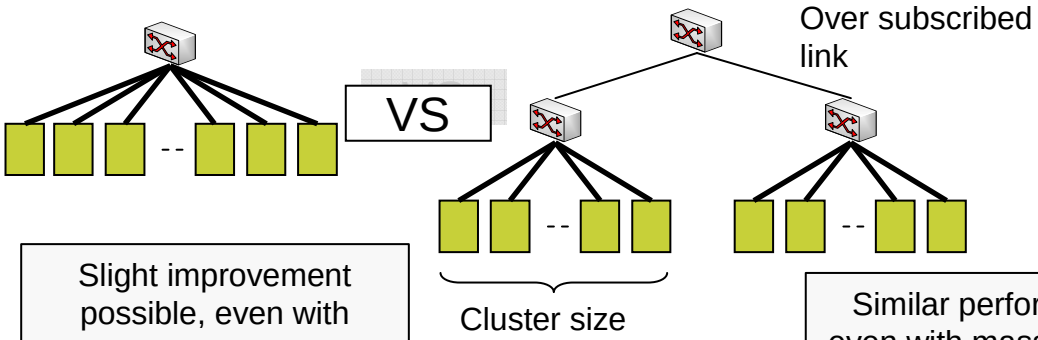
- Computing utilization accurately predicted as the load is kept below 1
  - Ratio prediction/measurement close to 1
  - Difference due to time spent by task in the system
- Communication utilization quickly diverges from the bound
  - Resulting from sequential execution
  - Higher bandwidth makes the system more reactive and prevents sequential execution

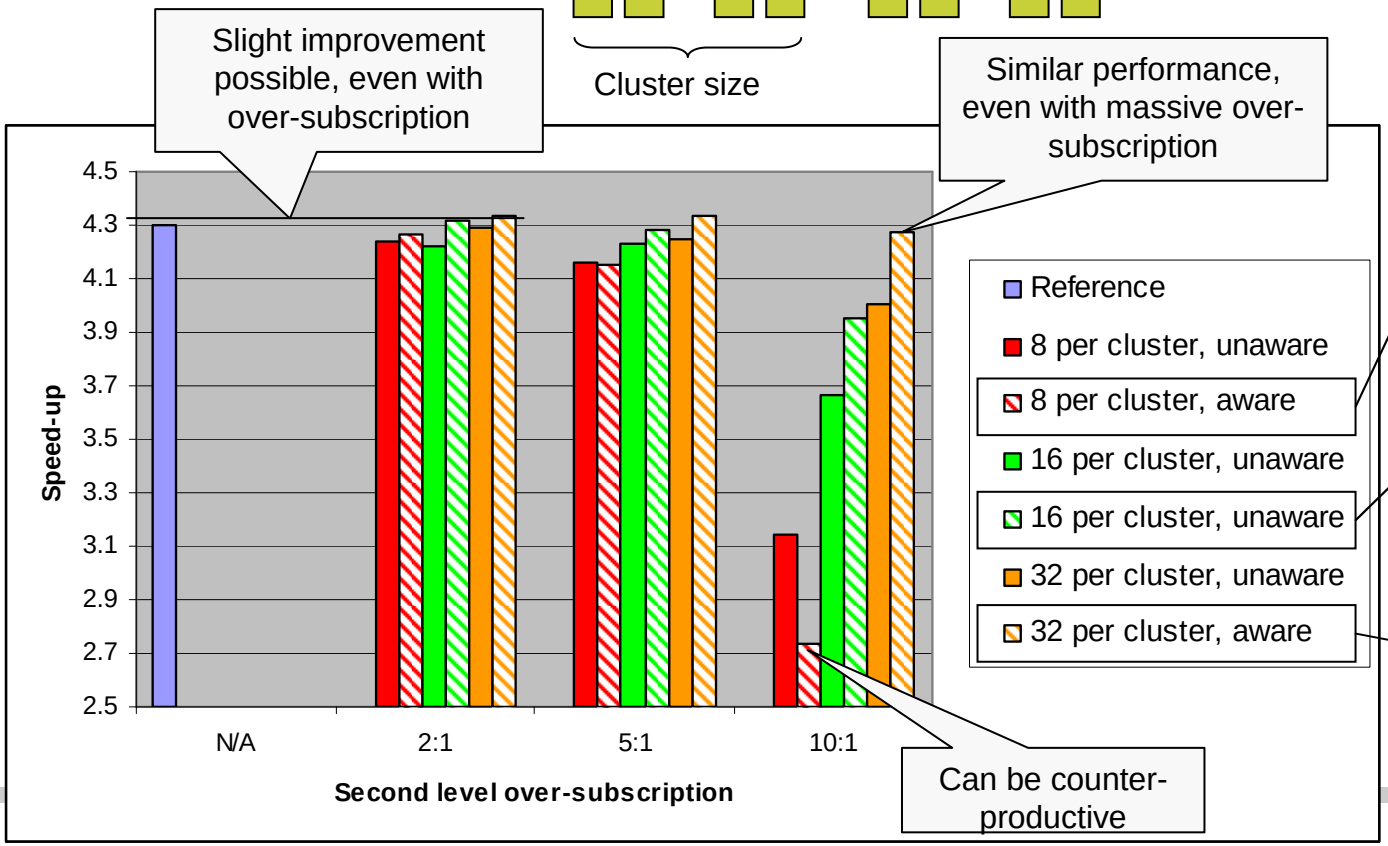


Topology aware scheduling

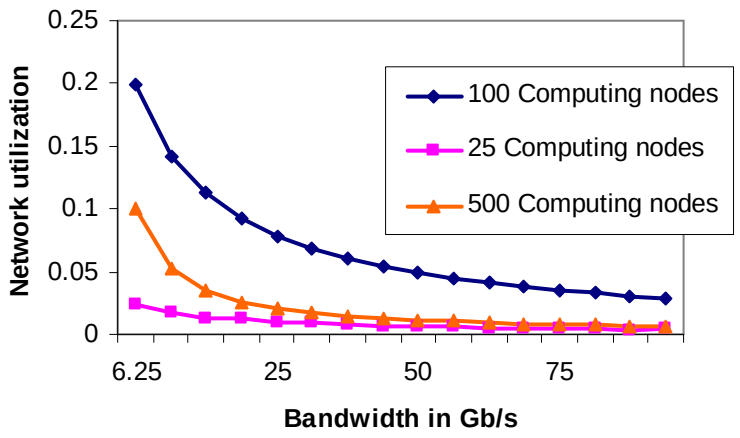
Traffic matrix for unaware scheduling

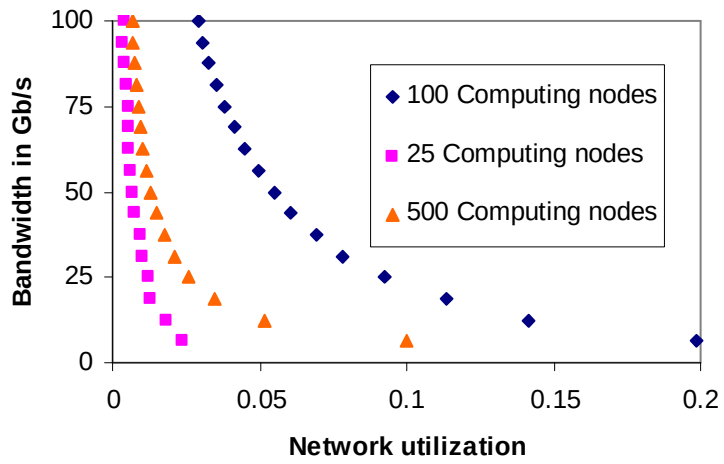
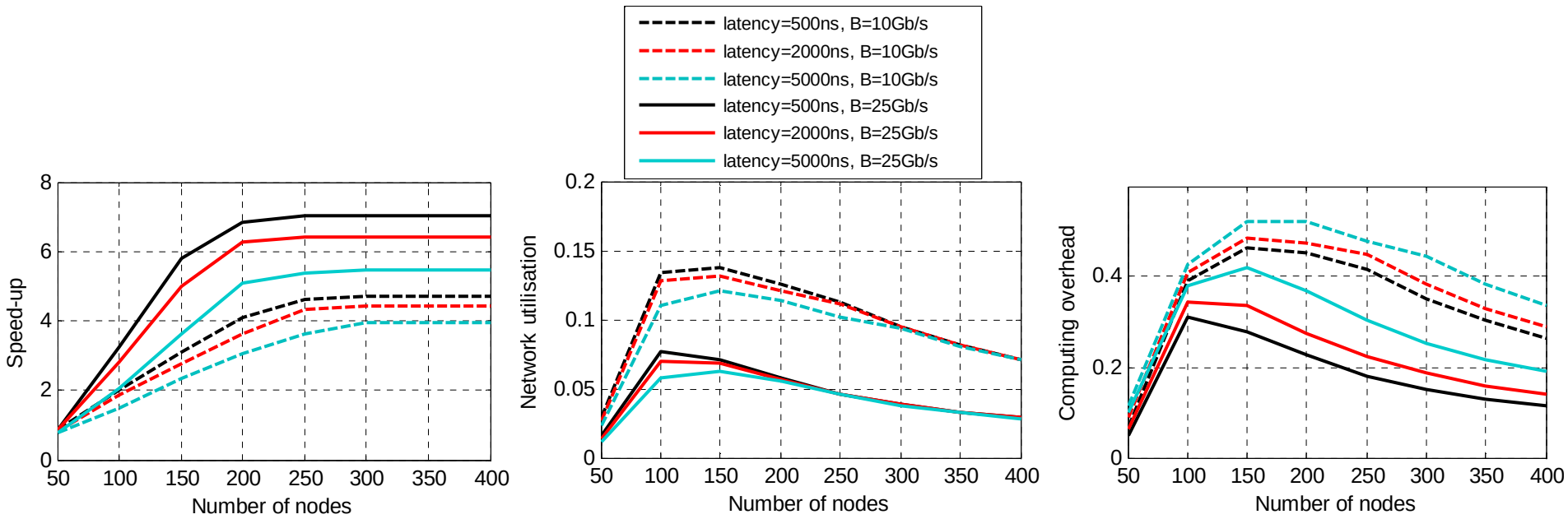






Network utilization vs. bandwidth





Impact of network latency