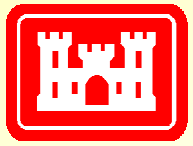


US Army Corps of Engineers®
Engineer Research and Development Center

Recomposing An Irregular Algorithm Using A Novel Low-Level PGAS Algorithm

Megan Cason (presenter)
U.S. Army Engineer Research and Development Center
Scientific Computing Research Center
email: megan.l.cason@us.army.mil

Peter Kogge
University of Notre Dame
email: kogge@nd.edu

Presented at the *Fourth International Workshop on Parallel Programming Models and Systems Software for High-End Computing*, Taipei, Taiwan, 13 September 2011

Outline

- **Motivation & Goals**
- PGAS Scalability Model
- Minimum Spanning Forest Search
- Analysis of Traditional Parallelization
- MoS Active Edge Toolkit
- MoS Active Edge Min Spanning Forest
- Conclusion

Motivation for PGAS Development

- Previous Generation HPC Architectures
  - One core per chip, one chip per node, many nodes per system
- Current Parallel Code Development Options
  - C/Fortran with OpenMP/MPI
  - Goals: portability, performance on distributed memory systems
  - Parallelism & communication supported in library calls & extensions

- Next Generation HPC Parallel Architectures
  - Many Cores per Chip: Shared DRAM
  - Many Chips per Node: Shared Address Space
  - Many Nodes per System: Distributed Memory
  - Heterogeneous System: Multiple execution models
- New Parallel Languages
  - UPC, HPF, Titanium, X10, Chapel
  - Goals: productivity, portability, performance on next gen systems
  - Parallelism & communication supported in language constructs/extensions

Motivation for Low Level PGAS Development & Analysis

- High Level PGAS Language Constructs
  - Goals: Productivity, Performance, Transparency
  - Implicit communication and parallelism
    - Predictable performance & load balance based on known data access patterns
  - Examples:
    - Distributed dense matrices (blocked, cyclic, etc)
    - Parallel for loops over arrays
    - Structured synchronization – global barriers, reductions
- Low Level PGAS Language Constructs
  - Goals: Performance, Opacity
  - Explicit communication and parallelism
    - Performance & load balance prediction depends on attributes of input data and runtime task creation, synchronization
    - How to analyze scalability as an attribute of the algorithm?
  - Examples:
    - Distributed graphs & sparse matrices
    - Recursive task creation
    - Unstructured synchronization – irregular data sharing

Related Work

- Low Level Constructs/Data Control
  - HPF-2: Indirect data distributions
  - Chapel: User defined domain maps
  - UPC: Global pointers
- Analysis of PGAS Algorithms
  - Other work uses system specific measurements: e.g. latency of get/put
- Graph Traversal Parallelization
  - Largest parallelizations: 1000s of threads running Breadth first search (Blue Gene)
  - Largest MSF parallelization: 100s of threads

Outline

- Motivation & Goals
- **PGAS Scalability Model**
- Minimum Spanning Forest Search
- Analysis of Traditional Parallelization
- MoS Active Edge Toolkit
- MoS Active Edge Min Spanning Forest
- Conclusion

Abstract PGAS System Model

- Define a set of P processes
  - Notated: p_0, p_1, …
- Define a set of P memory partitions
  - Notated: M_0, M_1, …
- Each process p_j owns a unique partition of memory, M_j
  - M_j is local to p_j
  - All M_k != M_j is remote to p_j
- Time to algorithm completion measured in steps
- Each process executes at most one action per step, a process may execute zero actions in a step if waiting for other processes.
- 4 Action types:
  - **Operate on local memory**
  - **Synchronize with all processes**
  - **Request another process to get/put data to its memory, i.e. request remote data**
  - **Respond to a request to get/put data from local memory, i.e. respond to request for remote data**

Scalability Definition

- Speed Efficiency (Es)
  - T = execution time
  - W = work completed
  - P = # of processors
  - $E_s = W / TP$
  - *Algorithm-System Scalability of Heterogeneous Computing*. Chen et al, J. Parallel Distrib. Comput, 2008
- Algorithm-System Scalability Condition
  - E_s remains constant for increased algorithm-system size
  - **$W / TP = W' / T'P'$**
  - Execution time drops: T > T'
  - Processor count increases: P < P'
  - Workload must increase to the degree that time does not scale perfectly: W <= W
    - Serialization
    - Communication overhead

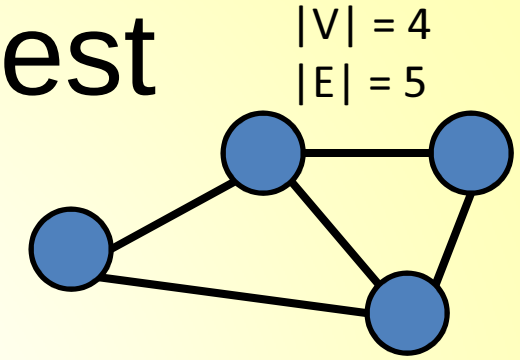
Real System Features Captured

- *Asynchronous execution*: an overloaded process slows down the whole system
- *Shared memory contention*: if one process gets more requests than others, it takes longer to complete
- *Weak Scaling pattern*: allow problem size to grow with system size
- *Load Balancing requirement*: scalability definition fails if work/communication does not scale evenly on all processes

Outline

- Motivation & Goals
- PGAS Scalability Model
- **Minimum Spanning Forest Search**
- Analysis of Traditional Parallelization
- MoS Active Edge Toolkit
- MoS Active Edge Min Spanning Forest
- Conclusion

Minimum Spanning Forest



- Given a graph, G=(V,E)
  - |V| = # vertices in the graph
  - |E| = # edges in the graph
  - Semantic graph: no physical position of vertices
- Produce a subset of edges from E that
  - Induces a set of trees (forest) over G
  - The forest has the same connected components as the original graph
  - The forest has minimal edge weight
- Useful applications of MSF
  - VLSI layout
  - Wireless communication optimization
  - Biology and medical data analysis
  - Social network analysis

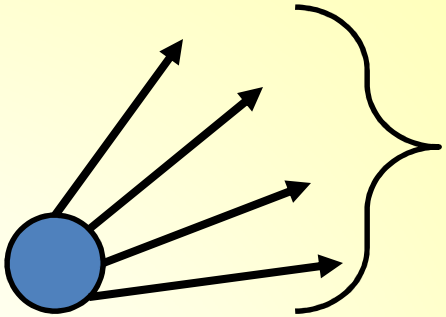
Minimum Spanning Forest

- Parallel algorithms derived from Boruvka's algorithm
  - While there is more than one vertex
    - 1. Choose the lightest edge from each vertex
    - 2. Find connected components defined by these edges
    - 3. Compact graph by collapsing all connected components defined by these edges
    - 4. Cleanup by removing loop and duplicate edges
- Vertex set drops by at least half during each iteration -> log(N) iterations

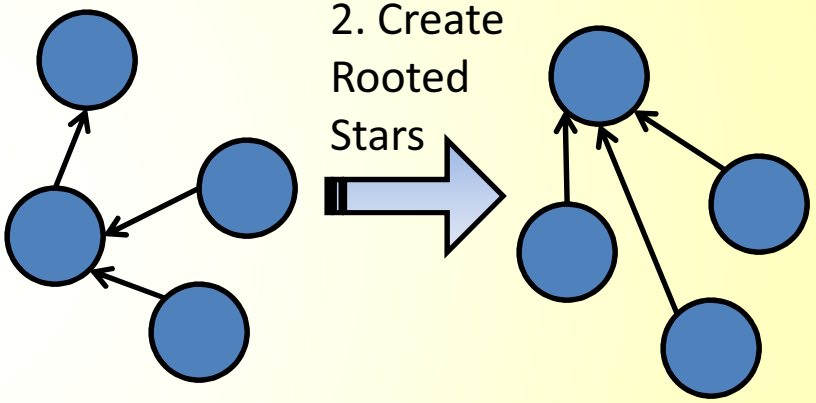
Outline

- Motivation & Goals
- PGAS Scalability Model
- Minimum Spanning Forest Search
- **Analysis of Traditional Parallelization**
- MoS Active Edge Toolkit
- MoS Active Edge Min Spanning Forest
- Conclusion

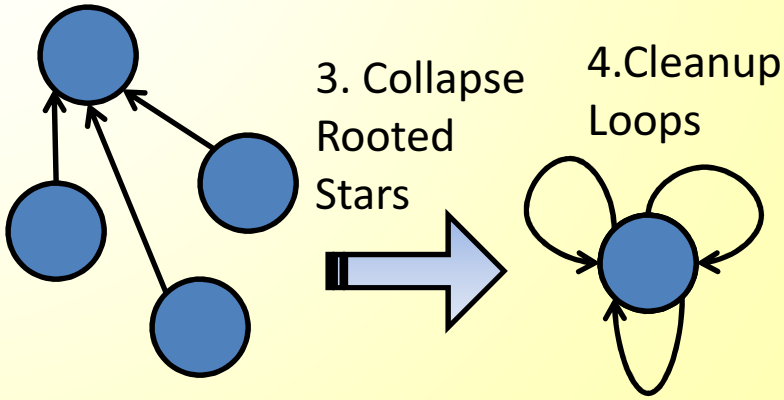
```
1:  procedure BORUVKA-PARALLEL(G = (V, E))
2:    V' ← V
3:    E' ← nil
4:    while |V'| > 0 do
5:      for all v_i ∈ V' in parallel do          ▷ 1. find lightest
6:        search edges of v for lightest edge, e_i
7:        add e_i to E'
8:      end for                                  ▷ synchronize processes
9:      V'' ← V'
10:     while |V''| > 0 do     ▷ 2. find connected components
11:       for all v_i ∈ V'' in parallel do
12:         if v_i is a tree root or points to a tree root then
13:           remove v_i from V''
14:         else
15:           perform pointer jump on v_i through e_i
16:         end if
17:       end for                                ▷ synchronize processes
18:     end while
19:     for all v_i ∈ V' in parallel do          ▷ 3. compact graph
20:       if v_i points to a tree root, v_j then
21:         append all edges from v_i to v_j
22:         remove v_i from V'
23:       end if
24:     end for                                  ▷ synchronize processes
25:     for all v_i ∈ V' in parallel do                    ▷ 4. cleanup
26:       remove loop and duplicate edges from v_i
27:     end for                                  ▷ synchronize processes
28:   end while
29: end procedure
```



1.Choose lightest edge per vertex



2. Create Rooted Stars

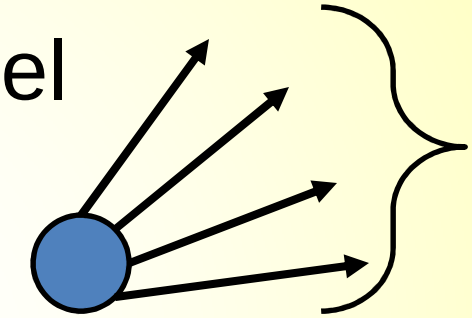


3. Collapse Rooted Stars

4.Cleanup Loops

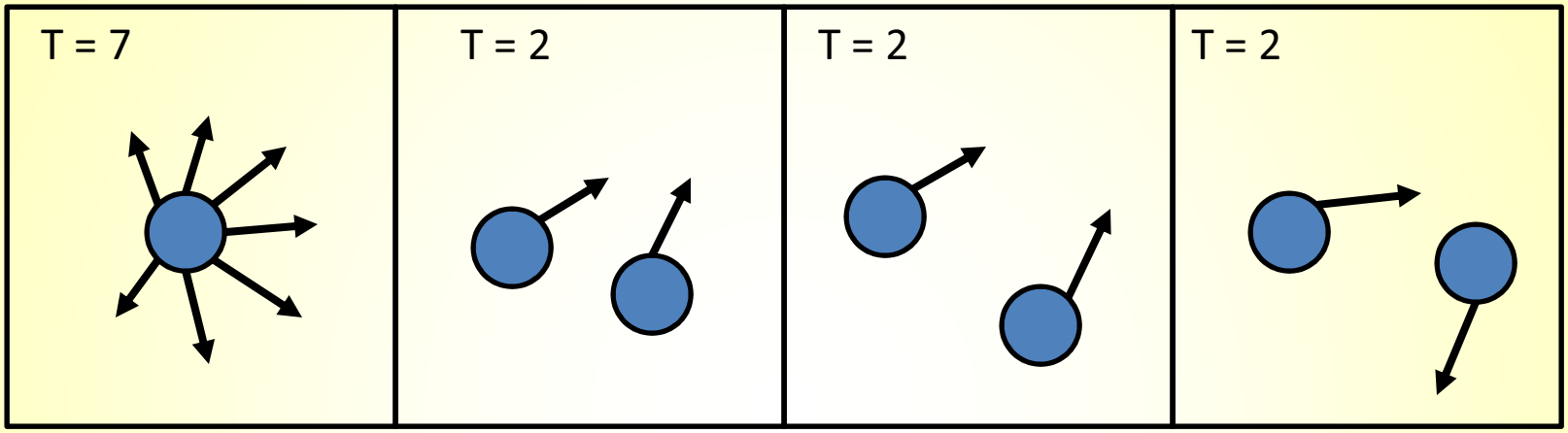
Searching Edge Lists in Parallel

- Vertices processing in parallel
  - Distribution?
  - Steps per Processor



- Is there a guaranteed scalable distribution?
  - Depends on input graph
  - Consider graph G, with |V| vertices, 2|V| edges
    - One vertex has |V| edges. All others have 1
    - Each vertex is stored on a processor with all its edges
    - One processor will end up with half the edges in the entire graph….

Searching Edge Lists In Parallel



- Time Taken On Abstract System
  - Case 1: All vertex lists searched in parallel by the process that owns them
    - Slowest process takes **T = |V|**
  - Case 2: Edge lists are copied out to be searched in parallel
    - One process must send |V| edges out, **T = |V|**
- Scalability Condition: W = 2|V|, T = |V|
  - (2|V| / |V| P) = (2|V'| / |V'| P') -> **2 / P = 2 / P'**

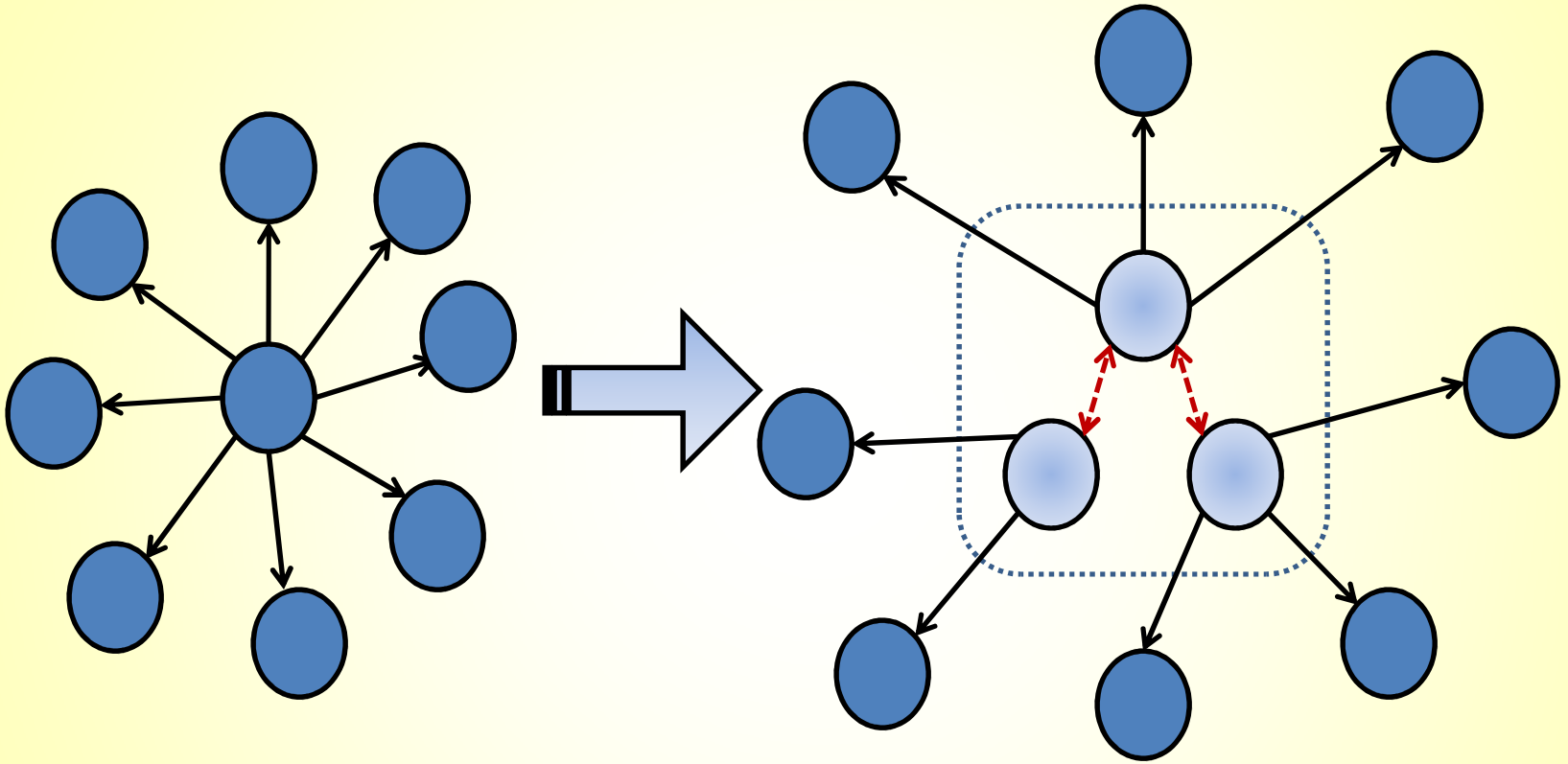
Outline

- Motivation & Goals
- PGAS Scalability Model
- Minimum Spanning Forest Search
- Analysis of Traditional Parallelization
- **MoS Active Edge Toolkit**
- MoS Active Edge Min Spanning Forest
- Conclusion

Toolkit Goals & Approach

- Provide low level reformulation of graph data while maintaining a view of high level “actual graph” algorithm
  - Actual graph: input data set, i.e. actual set of objects and relations to be analyzed
  - Reformulated graph: augmented data set, add objects and relations to represent overhead of parallelization
- Expose parallelization costs with Mobile Subjective (MoS) execution mode
  - Subjects: all state required to initiate parallel actions
  - Subject types: animals and plants
    - Plants incur cost of state storage
    - Animals incur cost of state storage and movement
  - Synchronizing data: stateful variables with blocking operations

Transform Shared Data



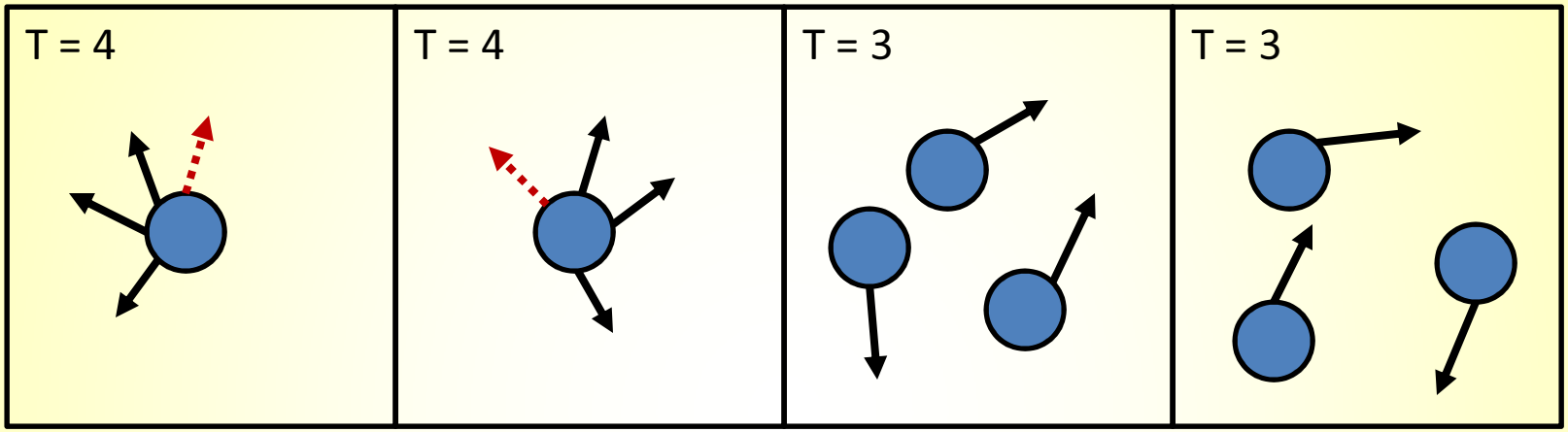
Actual Graph

Reformulated Graph
At most k(=3) edges per pvertex
Each vertex is a m-way of pvertices

Toolkit Operation by Edges on Vertices

- Barrier
  - An edge (v1, v2) joins a barrier with all other edges connected to v1 (or v2). No edge leaves the barrier until all edges enter the barrier
- Leave Vertex
  - An edge disconnects from a given vertex, effective after the next barrier
- Merge Into
  - An edge (v1, v2) causes v1 and v2 to merge and join all edges from both vertices into a single super vertex
- Reduce Operation
  - An edge (v1, v2) contributes to a given reduction operation (add, multiply, max, min, etc) on a given vertex. Results available after next barrier

Showing Scalability



- Assume each process holds an equal portion of shared data
  - Possible because vertices have become a distributed interface
- Each piece of shared data is accessed by no more than a fixed number of parallel tasks
  - Programmer specifies task per edge
  - Each edge shares data with few other edges

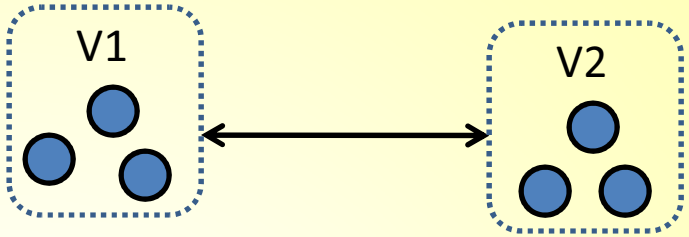
Showing Scalability

- After transforming shared data
  - W = # edges = |E|
  - Gets/puts per process
    - Depends on how many shared data
    - At most **m|E| / kP** = O(|E| / P)
    - Where *m* = order of pvertex trees, *k* = max edges per pvertex
  - Computation time per process
    - Includes structured synchronization time
    - At most **c (|E| log |E| / P**)
    - Where *c* is constant
  - Scalability Condition is satisfiable for all toolkit operations:
    - W / TP = W' / T'P'
    - |E| / P (|E| log |E| / P) = |E'| / P' (|E'| log |E'| / P')
    - 1 / (log |E|) = 1 / (log |E'|)

Outline

- Motivation & Goals
- PGAS Scalability Model
- Minimum Spanning Forest Search
- Analysis of Traditional Parallelization
- MoS Active Edge Toolkit
- **MoS Active Edge Min Spanning Forest**
- Conclusion

Scalable MSF

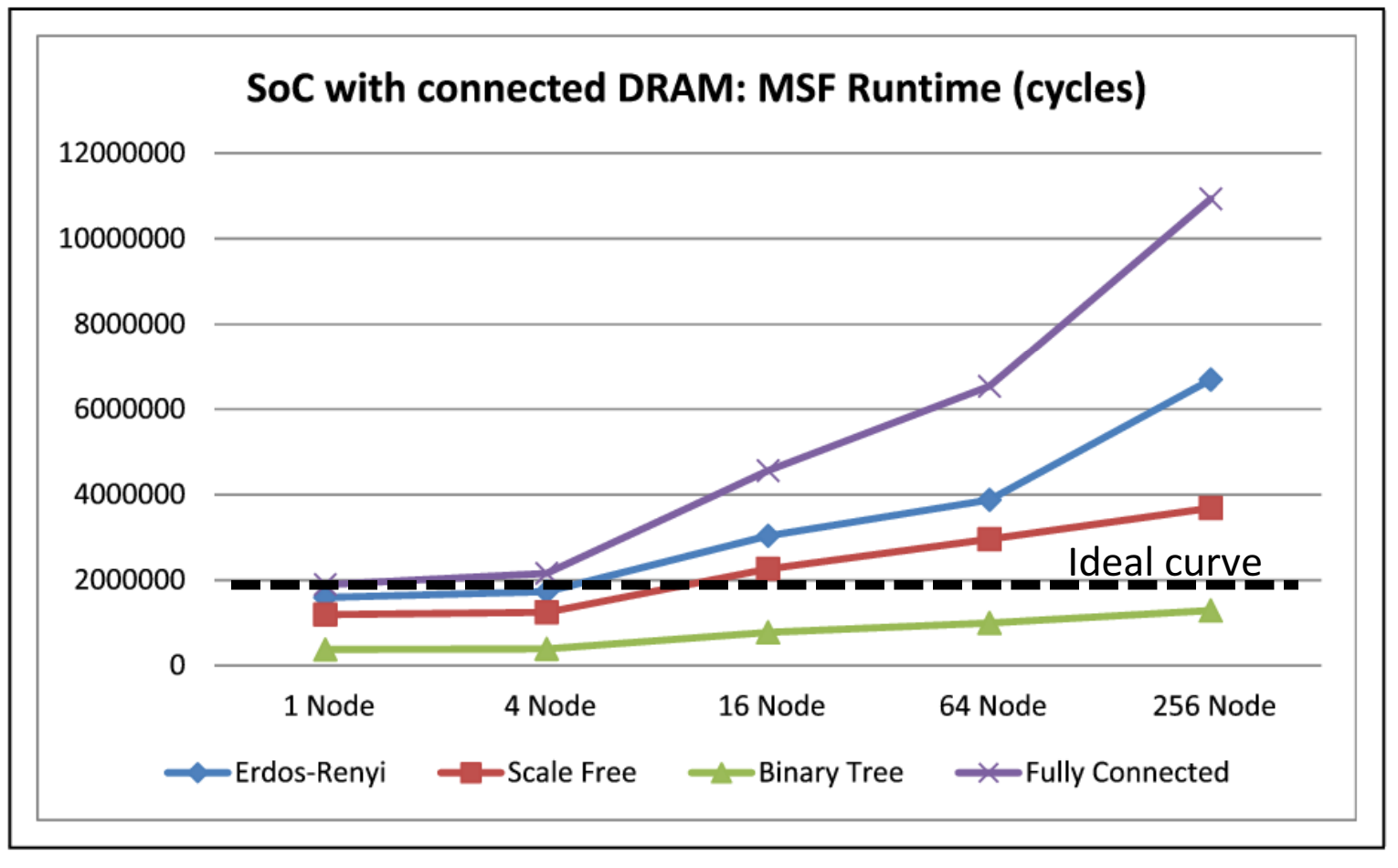


- For all edges (V1, V2) in parallel
  - Repeat
    - In parallel enter a *reduction* at V1 & V2
    - In parallel *barrier* with V1 & V2
    - Check if this is the lightest edge connected to V1 or V2 (result of reduction available after barrier)
    - If this is a lightest edge, *merge* V1 with V2
    - In parallel *barrier* with V1 & V2
    - If V1 merged with V2, remove this edge

Target System: Massively Multithreaded

- MoS was designed alongside novel LWP-mNUMA architecture
  - Subjects are first class architectural entities
  - No cache data duplication
  - Hardware supported subject creation/destruction/movement
  - Architectural PGAS support
- Simulation Results
  - Goal: to create millions of subjects performing useful computation in simulation

Simulated Runtimes



Outline

- Motivation & Goals
- PGAS Scalability Model
- Minimum Spanning Forest Search
- Analysis of Traditional Parallelization
- MoS Active Edge Toolkit
- MoS Active Edge Min Spanning Forest
- **Conclusion**

Conclusion

- Using generalized PGAS scalability metric
  - Exposes data “hotspots” by penalizing data sharing in a partition
  - Allows parallel algorithm design to include analysis of low level reduction-type communication
  - Avoids requiring system specific metrics
- Reformulation of graph traversal
  - Exposes low level costs while maintaining higher level algorithm structure
  - Is applicable to other load imbalanced problems
    - Particle interaction neighbor lists
    - Sparse matrices with some dense rows
    - Unstructured meshes

Questions?