



# Parallel Network Analysis

Cynthia A. Phillips

Sandia National Laboratories

Topical presentation 2009 SIAM Annual Meeting, July 10



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.





# My Topic

---

- What's new with me (Sandia National Laboratories) in  
Discrete math/algorithms  $\cap$   
High-performance computing  $\cap$   
Applications?

Why would we want to use a parallel algorithm for an application?

- When we **have** to:
  - Too Slow
  - Too Big
  - Too Inaccurate
- Application evolution
  - More constraints
  - Finer discretization
  - Larger instances



# Overview

---

- Applications
  - Sensor network design/management
  - Analysis of large-scale (e.g. social) networks
- Methods
  - Coarse-grained (“embarrassing” parallelism)
  - Unusual (for us) hardware/architecture
    - Old primitives
  - Memory reduction → parallel algorithms



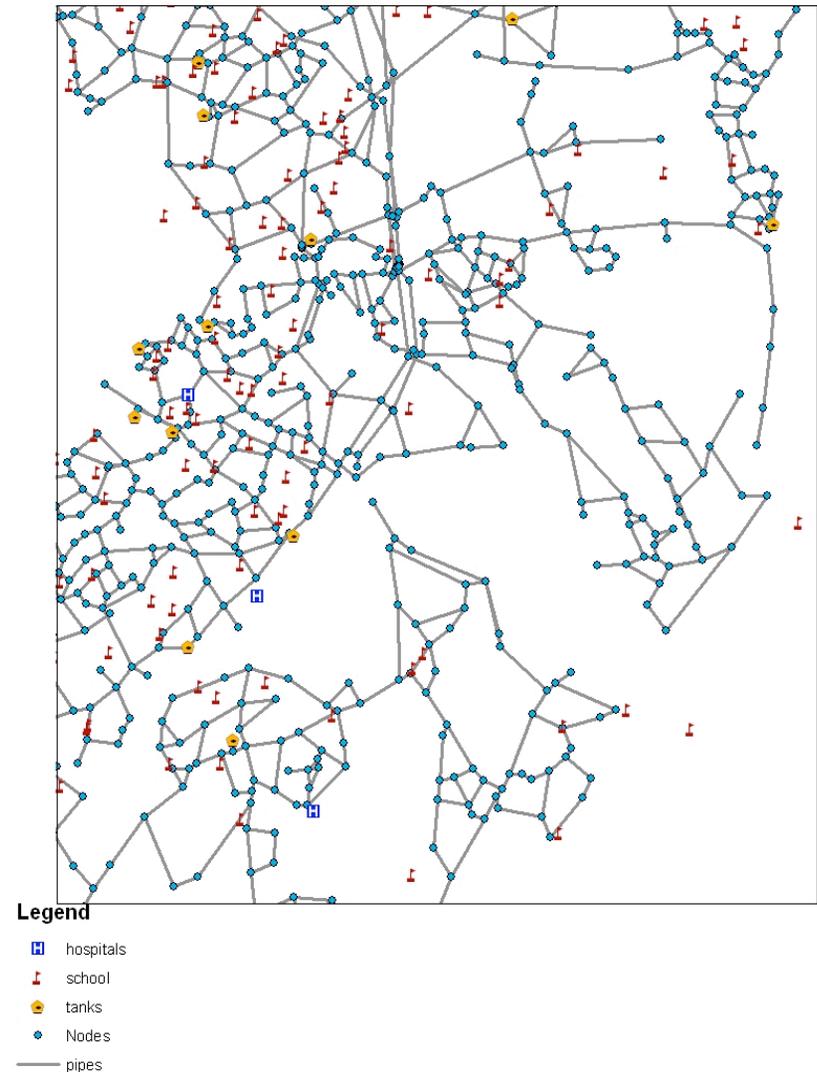
# A Sensor Placement Problem

Issue: Contamination released in a municipal water network

Goal: Place  $k$  sensors on network nodes as an early warning system

- Protect human populations
- Limit network remediation costs

Sponsored by the US Environmental Protection Agency (EPA) National Homeland Security Research Center (NHSRC)





# One Water Sensor Placement Formulation

---

Given an enumerable set of events: (location, time) pairs

- Simulate the evolution of a contaminant plume
- For each event determine
  - Where event can be observed
  - Impact prior to that observation
- Assume first sensor witness of contamination signals general alarm
- Minimize average impact

There can be 100,000s to millions of scenarios.

- Parallelize the simulations and impact calculations

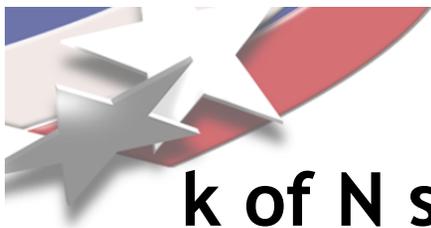
Obvious, but important: from weeks to hours using cluster



# Finding an Approximate Solution

---

- For more complex versions, can express as an **integer program**
  - Linear objective, linear and integrality constraints
- Benchmarks heuristics
- A good approximate solution
  - Speeds search (enables pruning)
  - Allows early stop
  
- Use **linear programming relaxation** (drop integrality)
- LP optimum gives lower bound, fractional solution
- Goal: “round” to a real solution
  
- Trials completely independent



## k of N selection

---

Given N variables with  $0 < x_i^* < 1$  for  $i = 1..N$  and  $\sum_i x_i^* = k$

- Select k of the  $x_i^*$  such that probability of selecting i is reasonably related to  $x_i^*$

In multiple applications, this selection is the main (only) decision

- Sensor placement
- Mobile sink scheduling for wireless sensor networks
- Picking a tail in robust optimization formulations
- Enforcing node degree in graph generation



# Randomized Rounding

---

- Simplest form: treat  $0 \leq x_i \leq 1$  as probability
- Set  $y_i = 1$  with probability  $x_i$  and  $y_i = 0$  otherwise
- If don't select exactly  $k$ , try again (and again...)
- But can use **conditional Poisson sampling** to efficiently sample from this “lucky” distribution (Chen, Dempster, Lui, 2004)
- Use dynamic programming to precompute conditional probabilities
- Decode a random toss to a feasible solution
- Selects uniformly over “lucky” tosses.



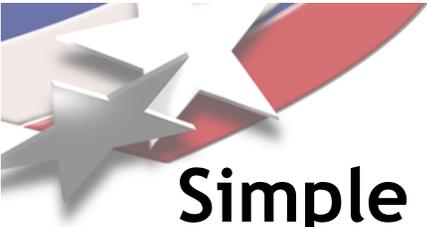
# Rounding with One Cardinality Constraint

---

Doerr (2004), motivated by Srinivasen (2001)

Finds a randomized rounding  $y$  such that:

- $\Pr(y_i = 1) = x_i^*$
- $\sum_i y_i = k$  (respects cardinality constraint)



## Simple (base) case

---

All  $x_i^*$  are  $1/2$ .

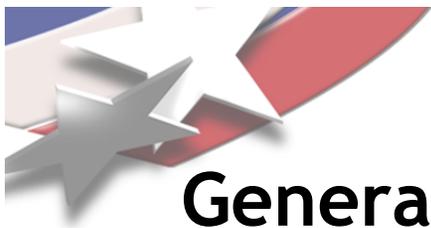
Let  $X$  be the set of  $x_i^*$  with value  $1/2$ .

$|X|$  is even because  $\sum_i x_i^* = k$  and  $k$  is integer

Pair elements of  $X$ :  $(x_i^*, x_j^*)$

Set  $(y_i, y_j) = (1, 0)$  or  $(0, 1)$  each with probability  $1/2$ .

$$(0,1) \xleftarrow{\frac{1}{2}} \left(\frac{1}{2}, \frac{1}{2}\right) \xrightarrow{\frac{1}{2}} (1,0)$$



## General Case

---

- Do the base case for lowest order bit  $\ell$  (most to right of binary point)

$$(x_i - 2^{-\ell}, x_j + 2^{-\ell}) \xleftarrow{\frac{1}{2}} (x_i, x_j) \xrightarrow{\frac{1}{2}} (x_i + 2^{-\ell}, x_j - 2^{-\ell})$$

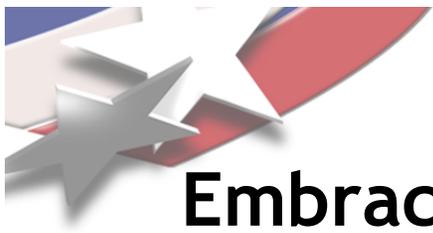
- After this operation, the rightmost bit is in place  $\ell - 1$ .
- Iterate to compute  $y$  in  $O(n \ell)$  time.
  - $n$  = number of variables  $0 < x_i^* < 1$
  - $\ell$  = lowest order bit of any of the  $x_i^*$ , maybe 1000
- Numerical issue: In (floating point) practice,  $\sum_i x_i^* = k$  not an integer



# Cardinality-Constrained Rounding Summary

---

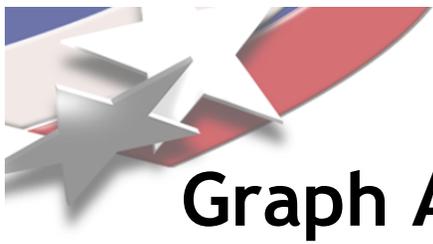
- Doerr
  - $O(nL)$  time  $L \approx 1000$  (multiprecision)
  - Paired total correlation, otherwise independent
  - $\Pr(y_i = 1) = x_i^*$
- Conditional Poisson sampling
  - $O(k(n-k))$  preprocessing ( $k < 100$ ), then  $O(n)$  sampling
  - Pairwise independence
  - $p(y_i = 1) = p(y_i = 1 \mid k \text{ selected})$
  - 3 orders of magnitude faster
  - Any subset is possible



# Embrace “Embarrassing” Parallelism

---

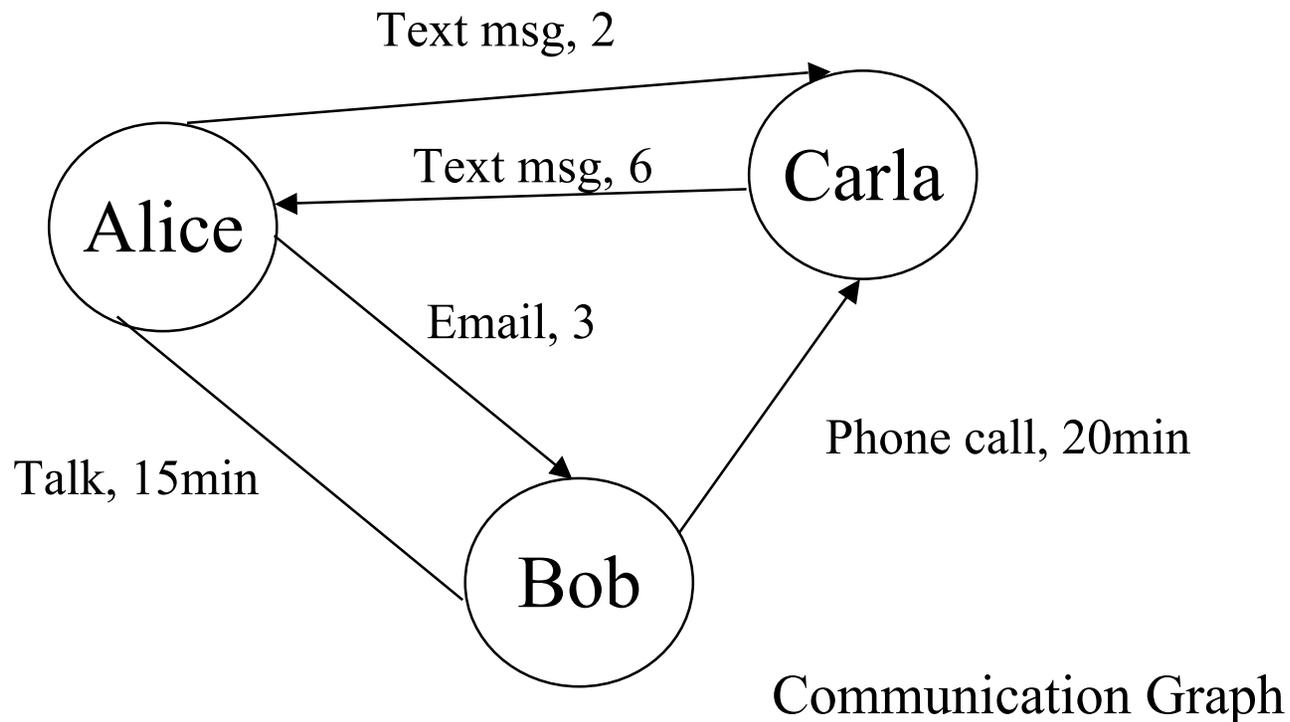
- Other recent uses
  - Integer programming pseudocost initialization
  - Feasibility pump integer programming heuristic
  - Progressive Hedging for stochastic programs
  - Constraint generation for scheduling mobile sinks in a wireless network
- Embarrassing parallelism increases the maximum feasible problem size
- Buys time to do the harder parallelization if necessary
- Using it can present other interesting algorithmic questions



# Graph Analysis

---

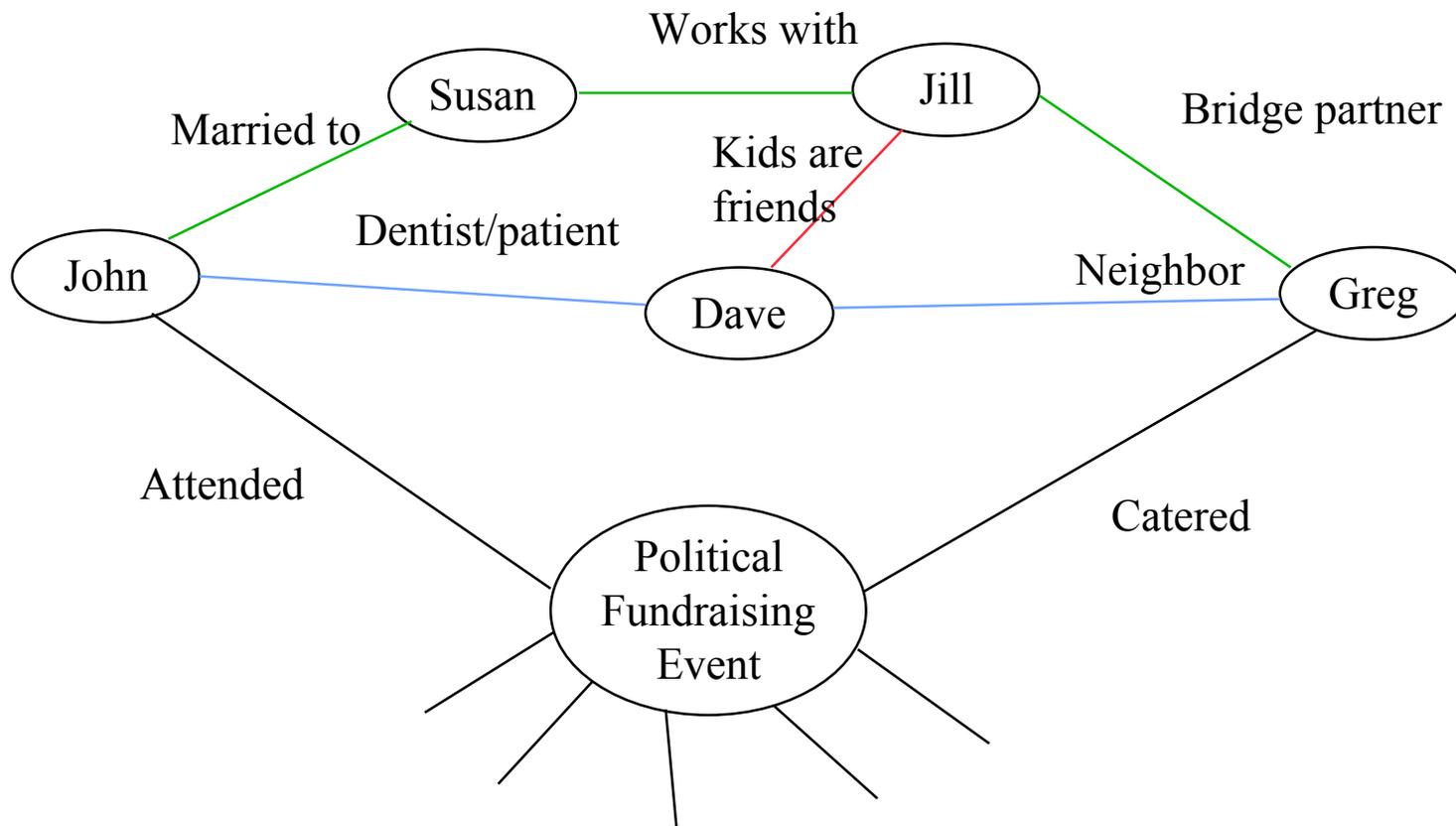
- Nodes (circles) represent entities
- Edges (lines) represent a relationship between a pair of entities
- Nodes and/or edges can have labels (names) and weights (values)





# A Semantic Relationship Graph

Every path between two points represents a potential relationship

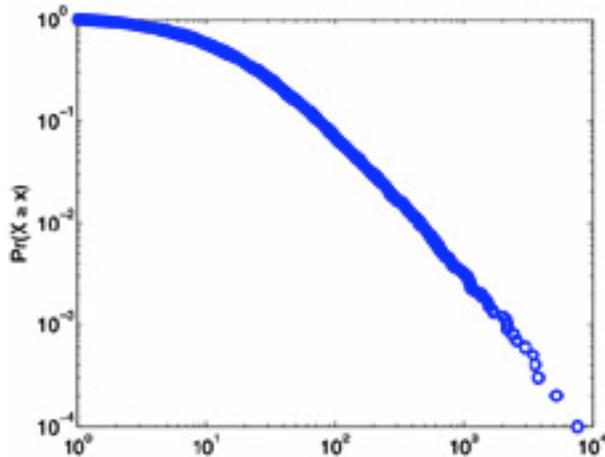




# Analysis of Massive Graphs

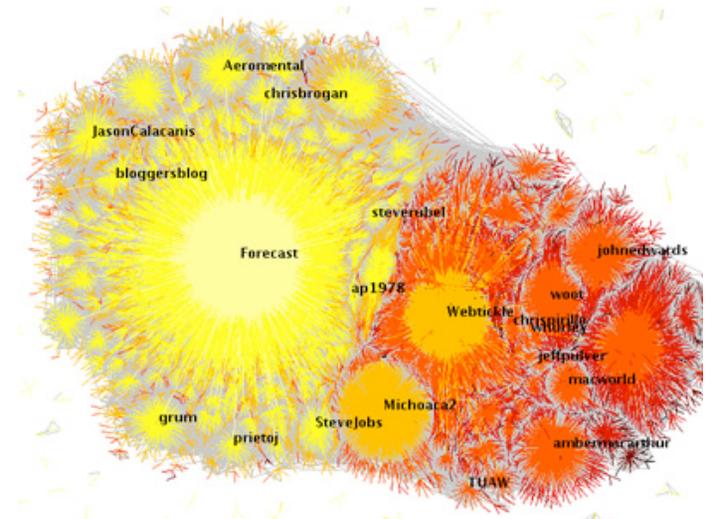
- Finding communities
  - Subgraphs where nodes are more connected to each other than to the rest of the graph
- Exploring relationships between individuals
- Finding patterns (normal/abnormal)
  
- Power law degree distribution common

Frequency  
(log scale)



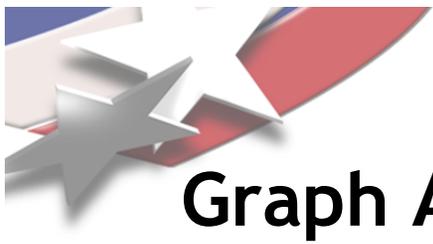
[Clauset, 2007]

Degree (log scale)



Twitter social network ( $|V| \approx 200M$ )

[Akshay Java, 2007]



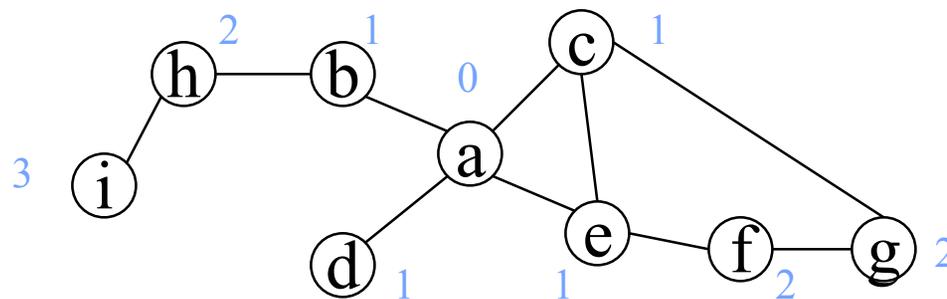
# Graph Algorithms

---

- There are many good serial algorithms (powerful modeling tool)
  - Generally nodes gather information from neighbors, traversals
  - Large amount of communication relative to computation
  - Limited locality, unpredictable

This is hard for

- Distributed memory: how to partition the graph?
- SMP (shared): cache management



Breadth-First Search Levels



# Caveat

---

There are lots of parallel architectures/systems, many new

- Distributed memory - tightly coupled or cluster
- Symmetric Memory Processors (SMP)
- Grid
- Cloud
- Multicore
- Graphical Processing Units (GPUs)
- Massive multithreading (XMT)

For any given application, one of these may be faster and/or give better performance/unit cost.

Example (Devine, Plimpton): matrix-vector multiplication on a distributed-memory machine (pagerank, some graph traversal)



# Massive Multithreading: The Cray MTA-2

---

- Slow clock rate (220Mhz)
- 128 “streams” per processor
- Global address space
- Word-level synchronization
- Atomic increments
- Simple, serial-like programming model
- Advanced parallelizing compilers

No Processor Cache

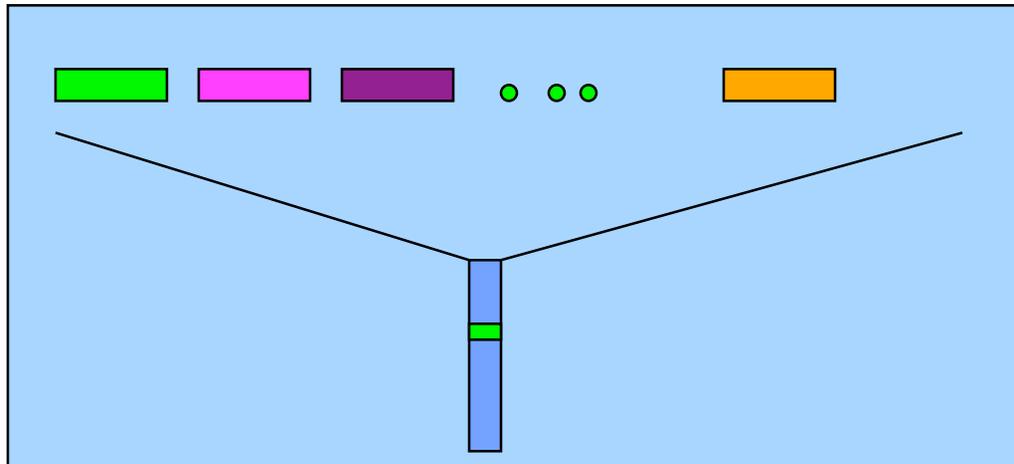
*Latency Tolerant:*  
important for Graph Algorithms

Hashed Memory



# Cray MTA Processor

---



- Each thread can have up to 8 memory refs in flight
- Round trip to memory ~150 cycles (MTA-2)
- New Cray XMT combines up to 8192 MTA proc. with Red Storm network
  - *Faster clock, but less network bandwidth*

Slide 20 *More memory (up to 128TB), but slower memory*



# Additional Challenges

---

- Deep pipeline (21)
- One instruction per thread in the pipe

The good news: FAST context switches (one clock cycle)

The bad news: Context switch is mandatory every clock cycle

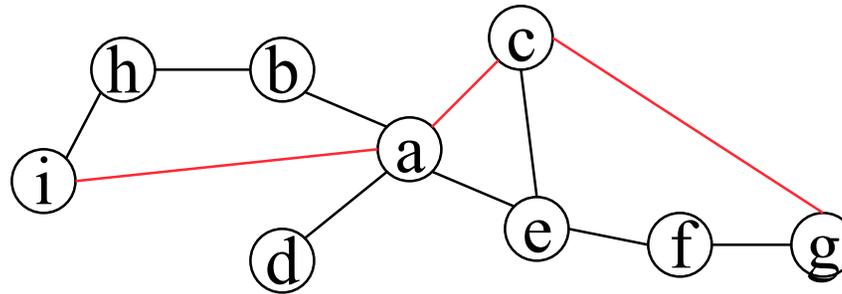
- Example: 40 streams, each with 4 memory references in flight will tolerate latency
- One processor is approximately equal to a linux box if using perfectly



# Unweighted S-T Connectivity

---

- Compute the minimum number of edges between two specific nodes

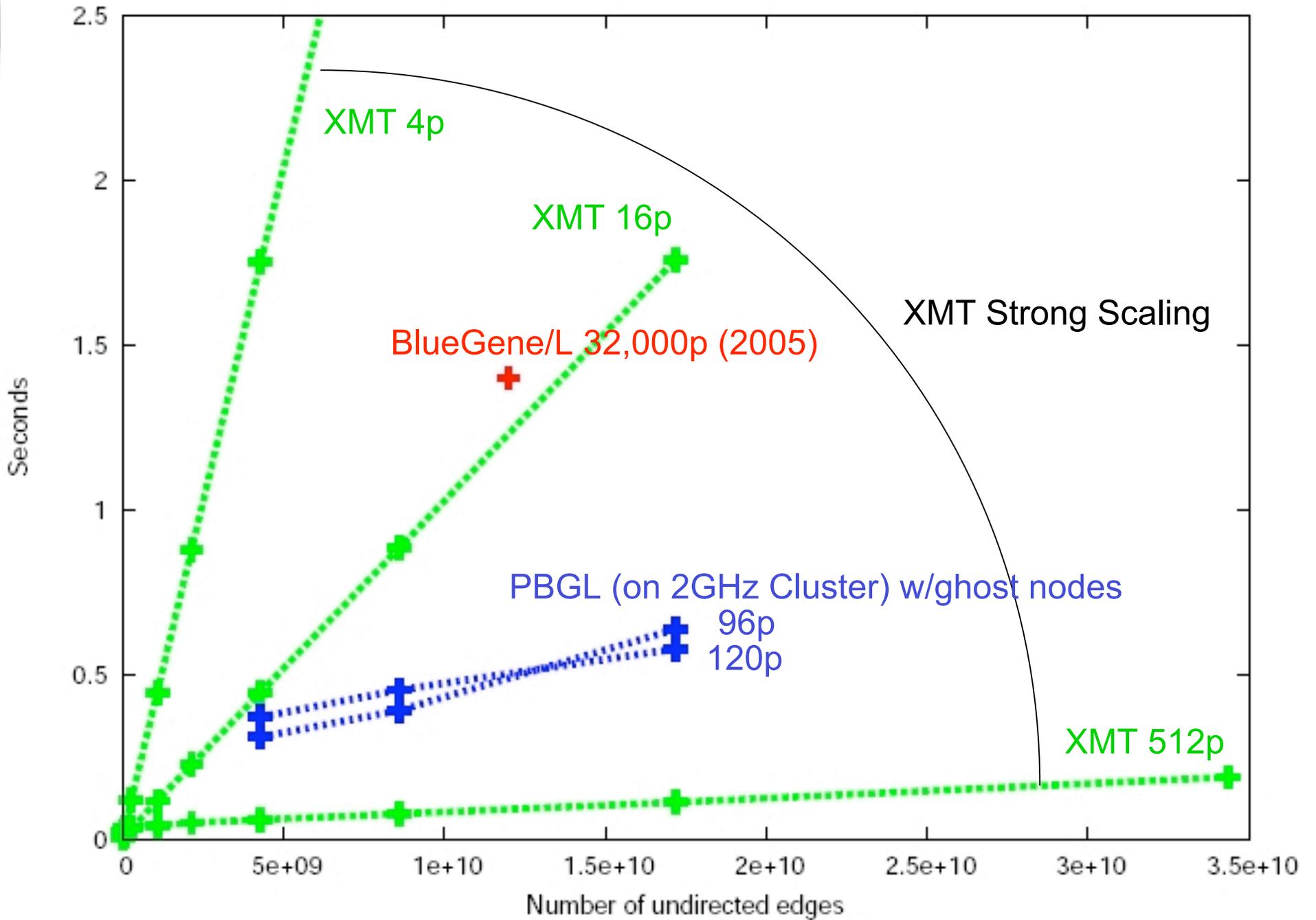


Shortest i-to-g path is length **3**

BFS from both s and t till they meet

Computational example:

- Erdos-Renyi graphs
- Expected shortest path is constant sized
- 5 trials for each of 10 random s-t pairs

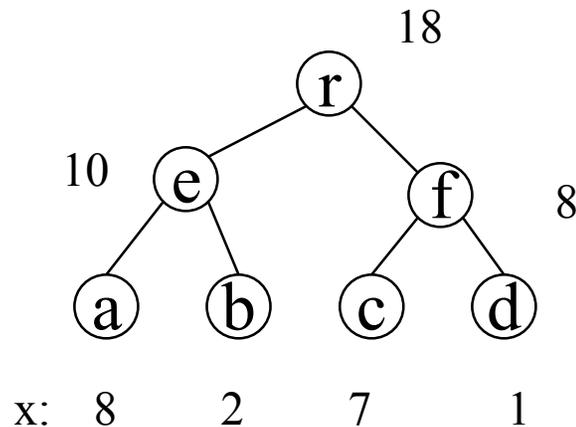




# Programming the XMT

---

- Compiler directives
  - As with Cilk++, permission, not commands
- Negotiate with the compiler
- Multithreaded graph library (MTGL) encapsulates some primitives (e.g. BFS)
- Compiler recognizes a reduction:



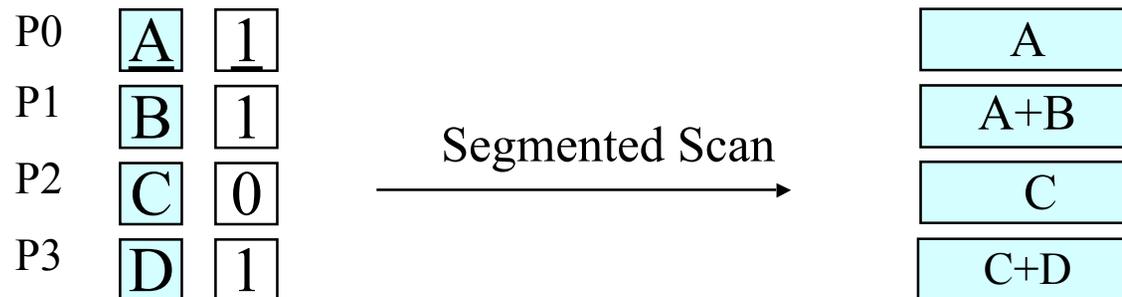
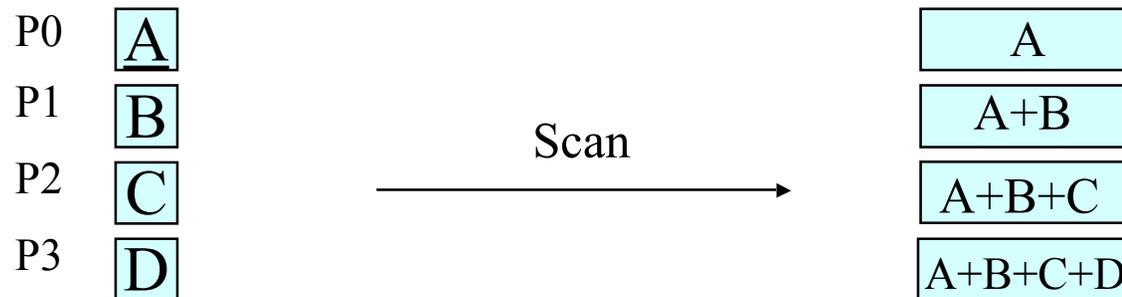
```
total := 0  
For i := 1 to n  
  total += x[i]
```



# Parallel Prefix (Prefix-sum, Scan)

---

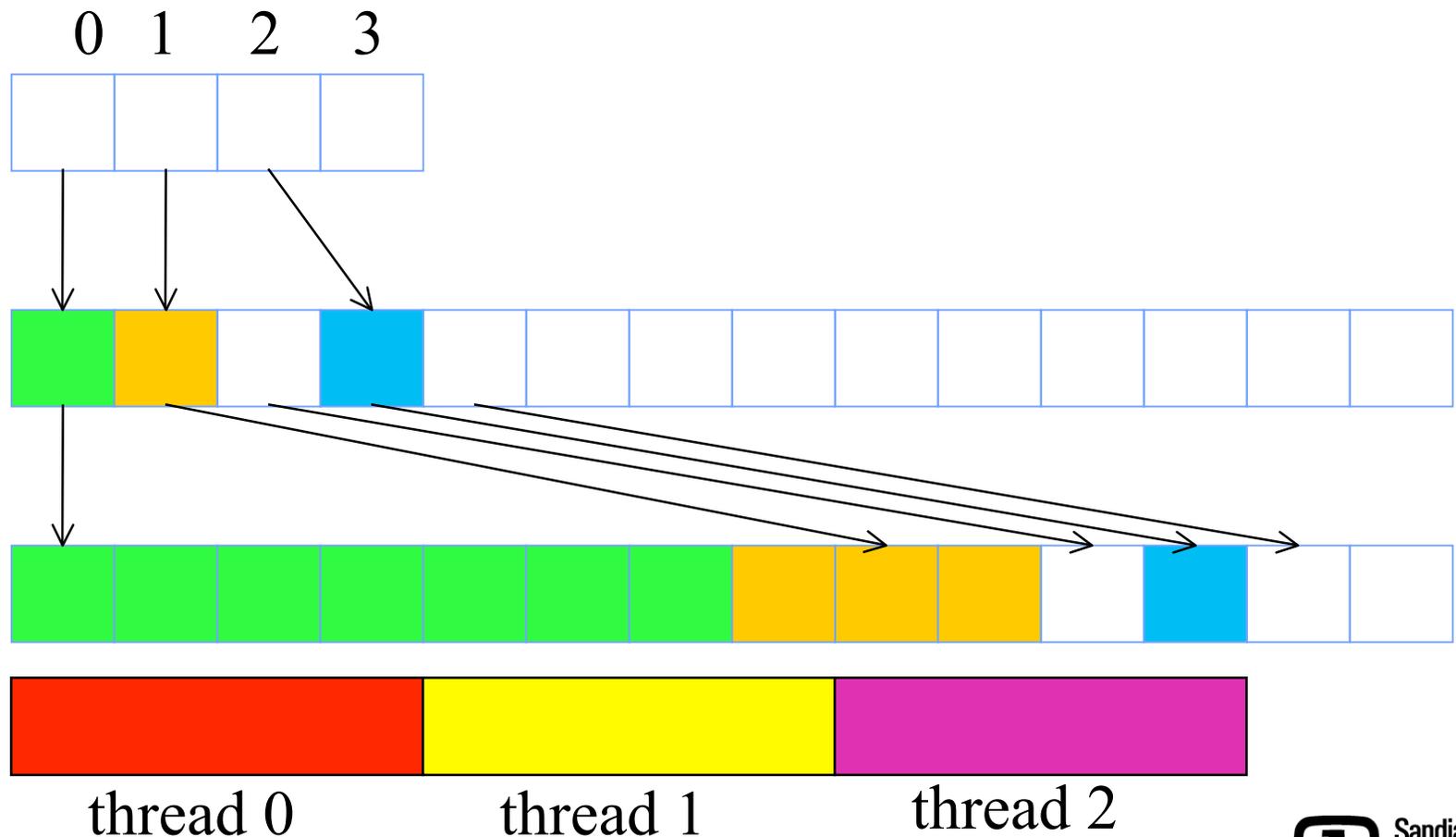
- Introduced by Blelloch [1990].
- “Sum” is binary associative (+, \*, min, max, left-copy)
- Applications: lexically comparing strings of characters, adding multiprecision numbers, evaluating polynomials, sorting and searching.





# Parallel prefix example in BFS

- Parallelize each level of a breadth-first search
- Create C chunks by equally dividing the neighborhood of the nodes currently in the queue





# Parallel prefix example in BFS

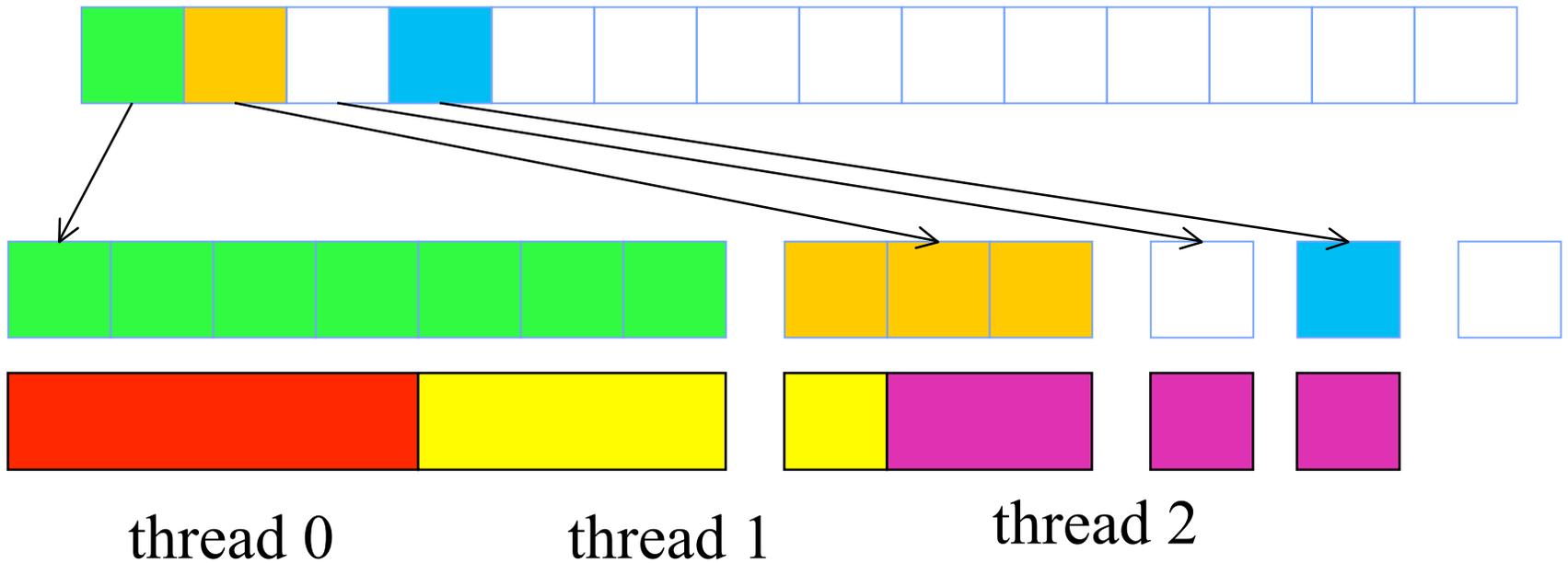
- Total work 12. Each thread gets four (1 + 1 to 4i).

7	3	1	1
---	---	---	---

degree

7	10	11	12
---	----	----	----

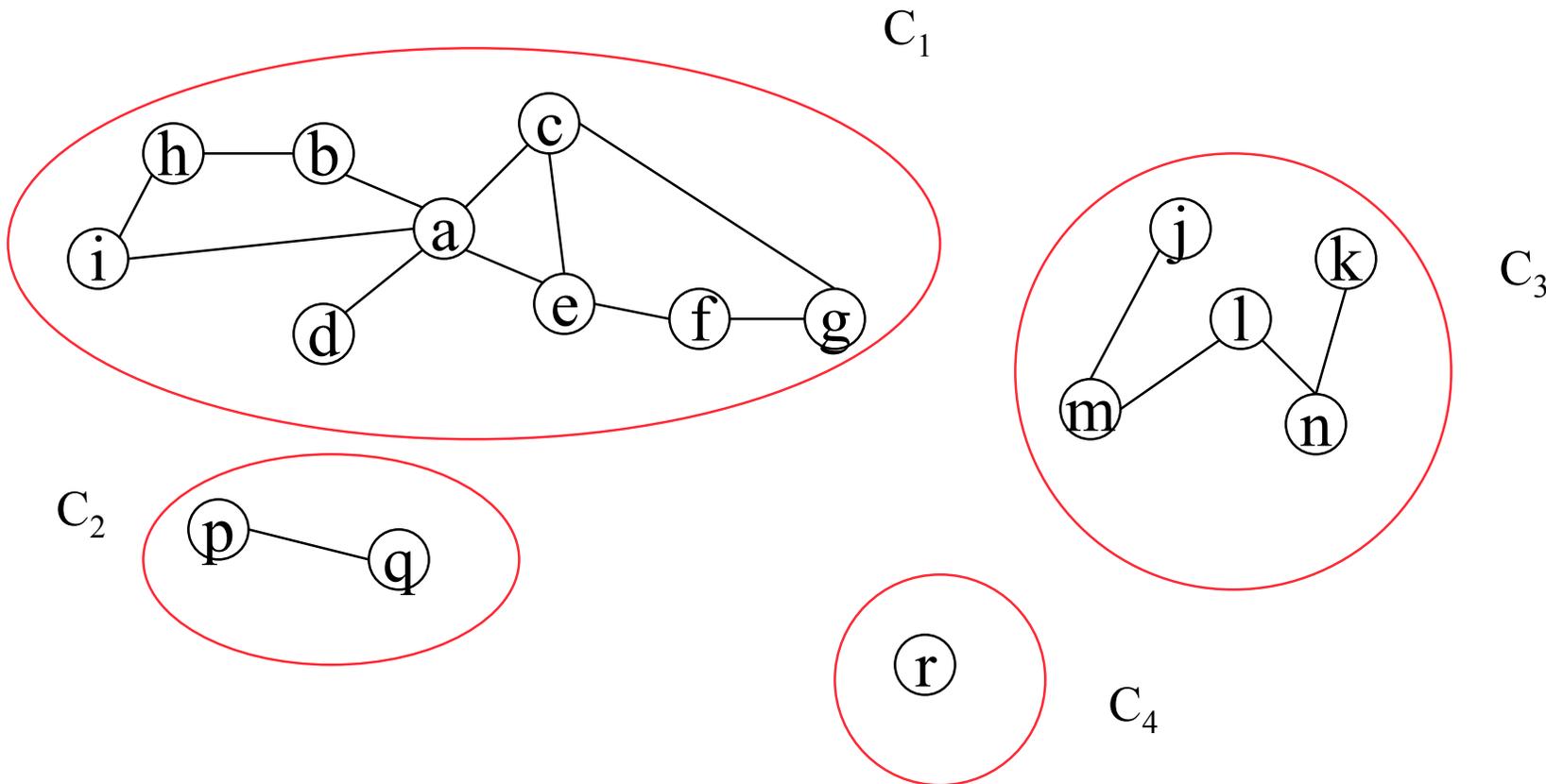
cumulative degree





# Connected Components Problem

- Give each node a label
- Two nodes have the same label if there is a path between them

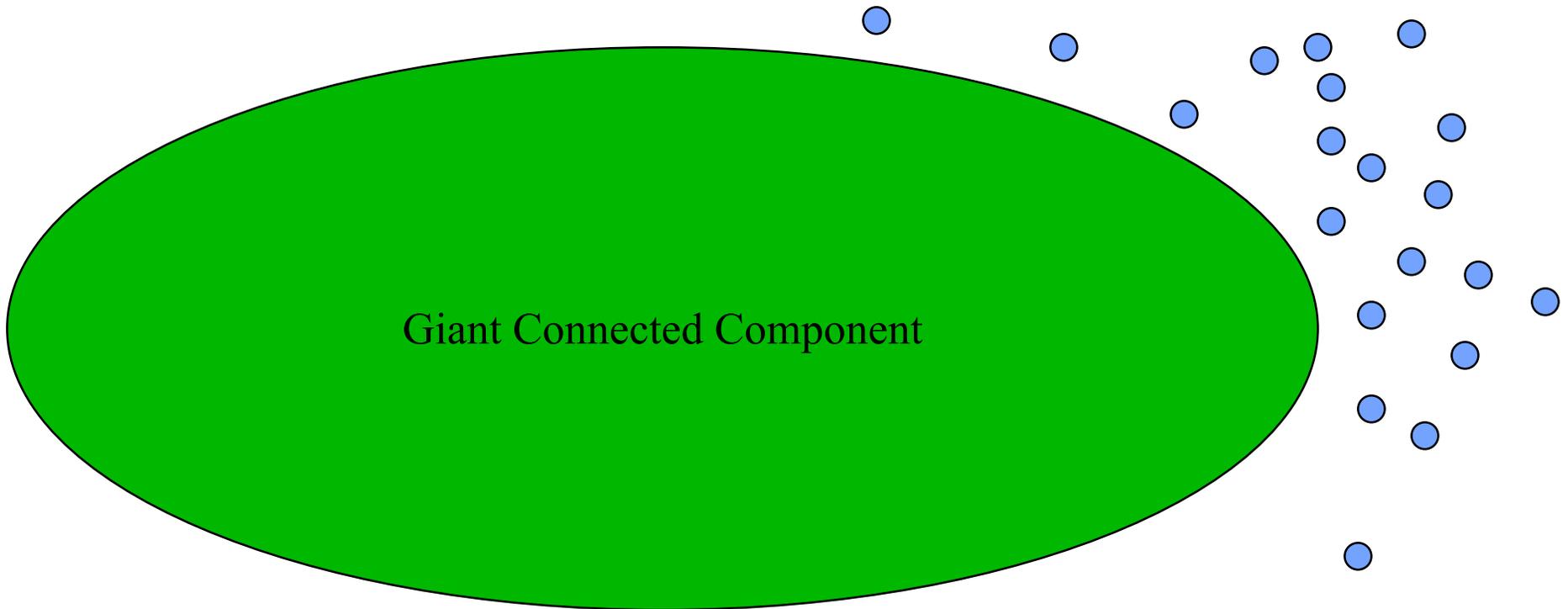




# Connected Components on XMT (Power Law)

---

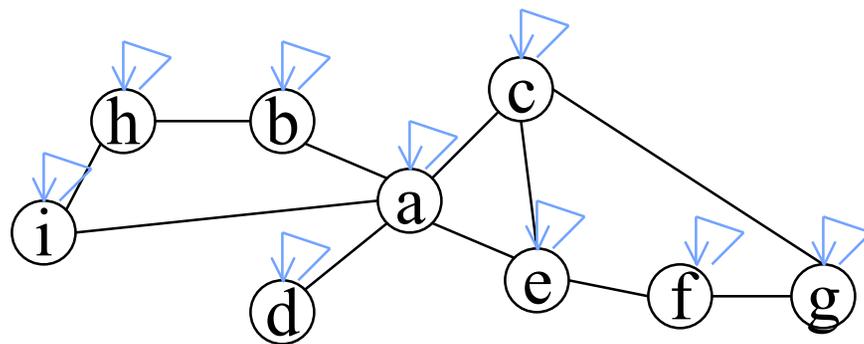
- Do a parallel BFS from the node of largest degree
  - Will likely label the largest (great) component



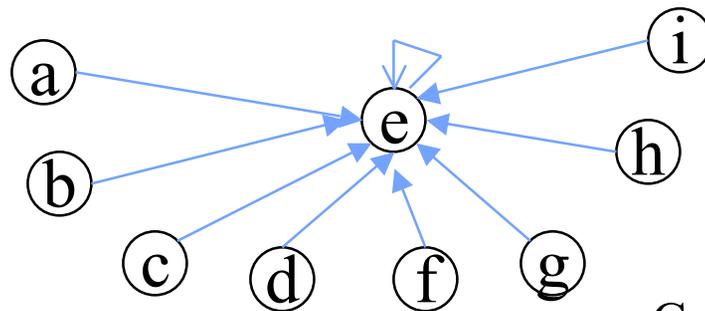


# Connected Components on XMT (Power Law)

- Clean up with Shiloach-Vishkin PRAM algorithm
- Hook with edges, pointer jumping to create a star



Start as own leader (label)



Component small.  
No significant hot spot



# Community Detection

---

- We are applying unconstrained facility location to finding communities on the XMT [See Jon Berry (MS75, 10:30) for a bit more]
- Motivated by EPA water sensor network problem
  - EPA wanted low-memory algorithms (run on PC)
  - Sensor placement problem is p-median (facility location)
  - We adapted code from COIN-OR for unconstrained facility location



# Low-Memory to Parallel

---

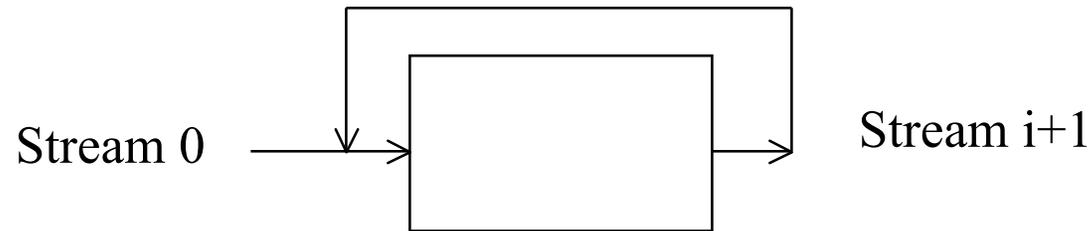
- Streaming algorithms
- Answer a question as the data set streams by
  - Use much less local space than the stream size
- Example: Watch a permutation of  $1..n$  ( $n$  known) with one number missing. You have space for one number. Determine the missing number.
- Answer: store the sum of the numbers you have seen.



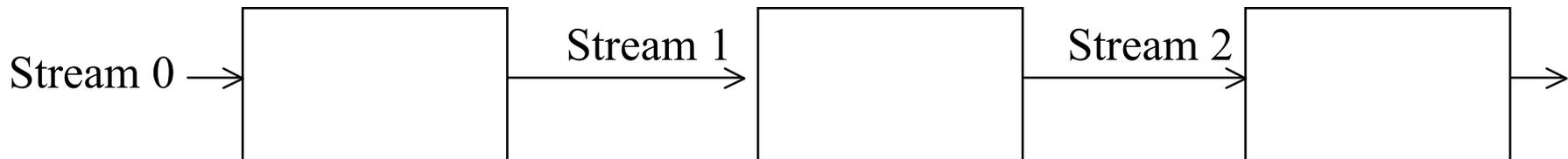
# W-Stream

---

- Read a stream, write a stream for another pass



- Unroll for a parallel machine that keeps the streams in flight/use:





# Connected Components, W-Stream

---

- Input: edges of finite graph in a stream  $(v_1, v_2), \dots, (v_i, v_j)$
- Output: (edge, label) pairs [label will be a vertex name, star]

Stream has two parts:

A: edges between partial components

- Initially the graph edges

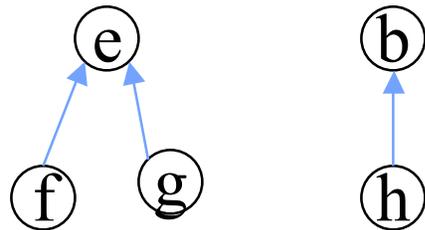
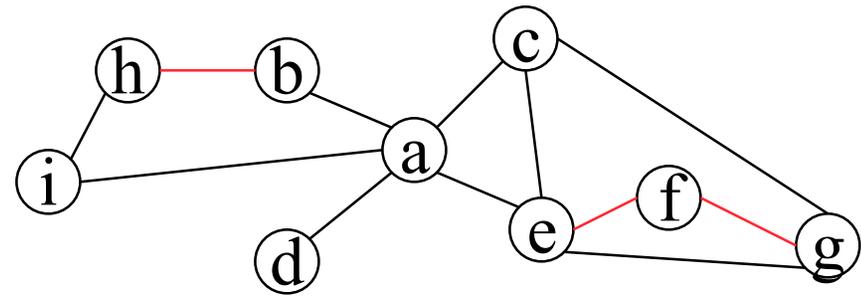
B: (node, label) pairs

- Initially  $(v_i, v_i)$  implicitly



# W-Stream Connected Components

- For each processor (stage), accept edges from the A stream and compute connected components (stars) until memory full

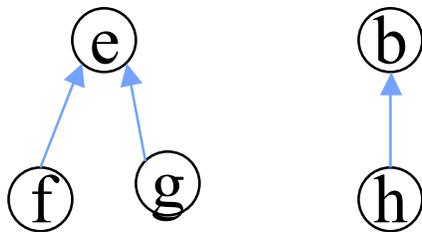
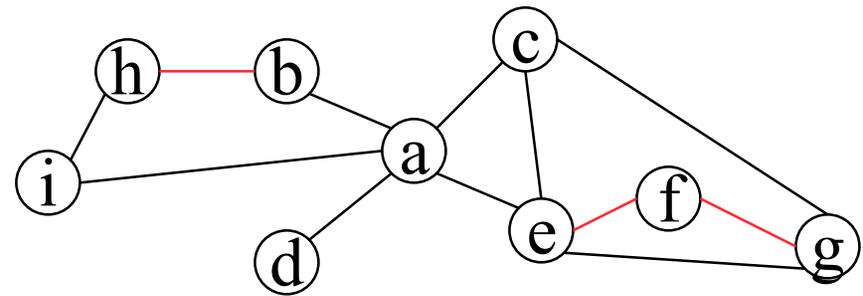




# W-Stream Connected Components

For rest of A stage

- Map known nodes to labels
- Drop intracomponent edges



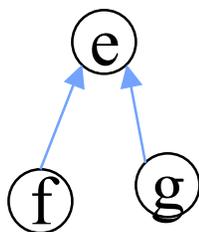
A Output: (i,b) for (i,h)  
(c,e) for (c,g)  
drop (e,g)  
rest unchanged



# Connected Components W-Stream

---

- At first stage, after last end, output A/B boundary marker and list the components:

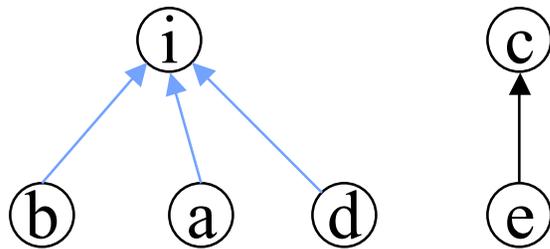
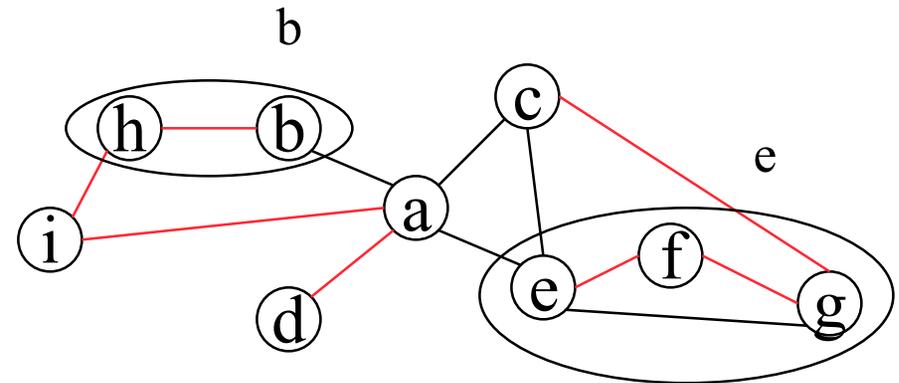


Output: (f,e), (g,e), (e,e), (h,b)



# Connected Components W-stream

- Repeat A phase
- On B update labels as necessary



Output:

A: (i,c) from (a,c) and (a,e)

B: (f,c) for (f,e)

(g,c) for (g,e)

(e,c) for (e,e)

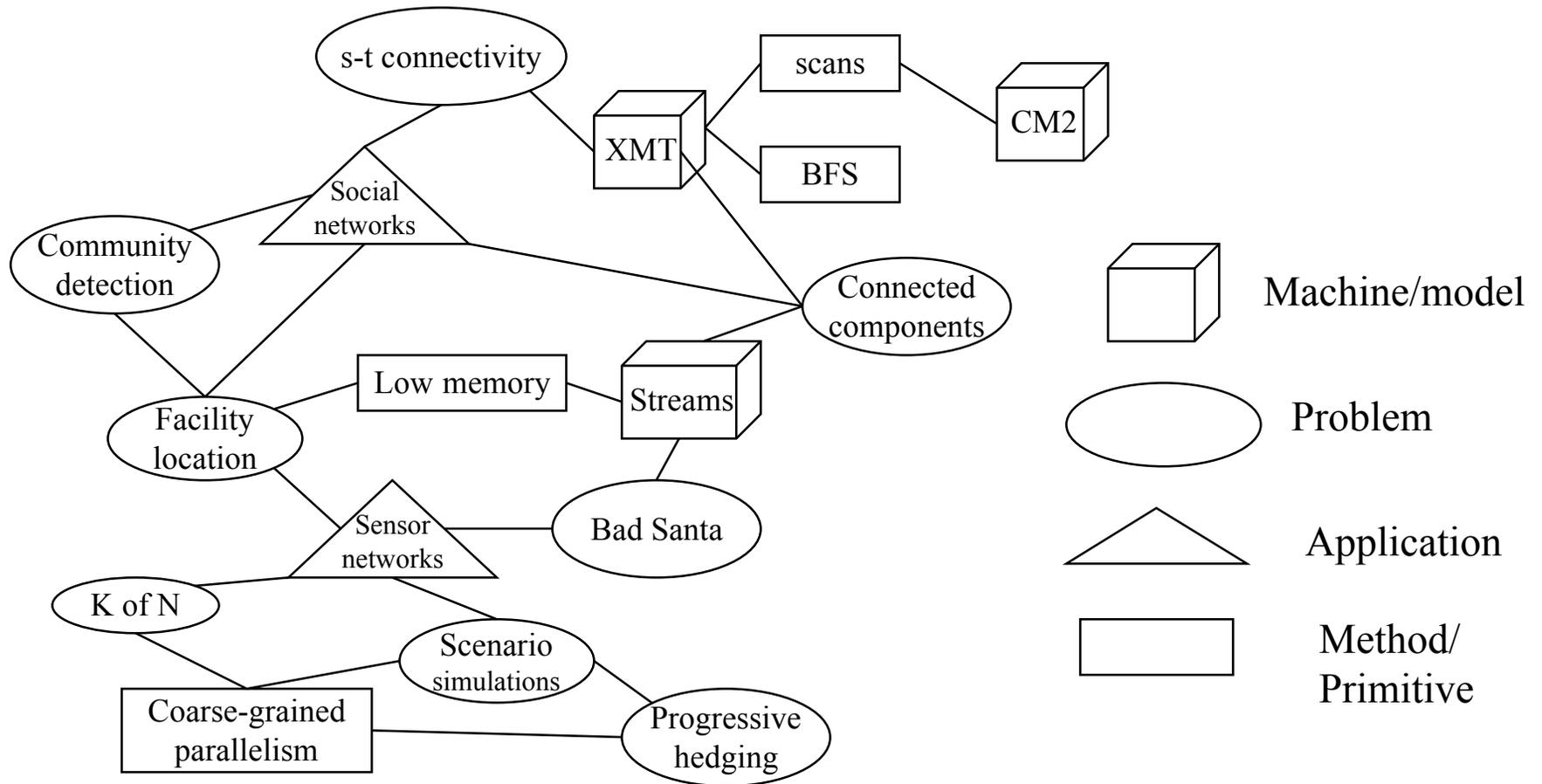
(h,i) for (h,b)

After marker: (b,i), (a,i), (d,i) (i,i) (c,c)

Note: e was done in phase A



# The interconnection of concepts



- Thanks to: Jon Berry, Bruce Hendrickson, Karen Devine, Steve Plimpton, Bill Hart, Erik Boman, Cray Inc, The US EPA, Michael Bender, Nick Edmonds, Jeremiah Willcock, David Mizell, Kamesh Madduri