Requirement and Performance of Data Intensive, Irregular Applications

John Feo

Center for Adaptive Supercomputing Software
Pacific Northwest National Laboratory

April, 2010

Center for Adaptive Supercomputer Software

A RESEARCH CENTER

FOR

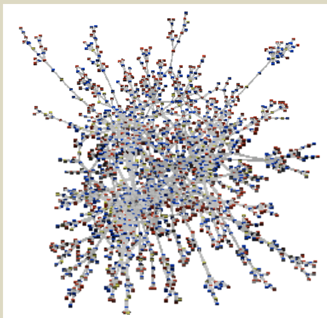
LARGE-SCALE DATA ANALYTICS

Sponsored by DOD

Proudly Operated by Battelle Since 1965

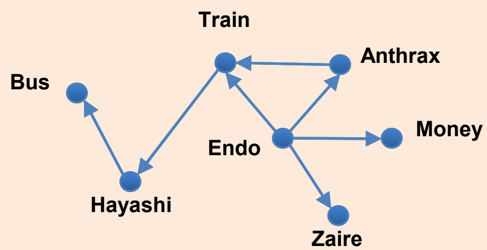
Analytic methods and applications

FaceBook - 300 M users



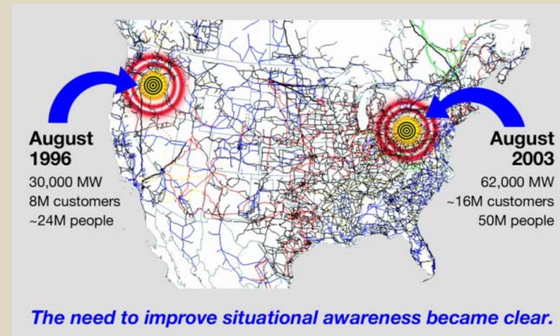
Community Activities

National Security



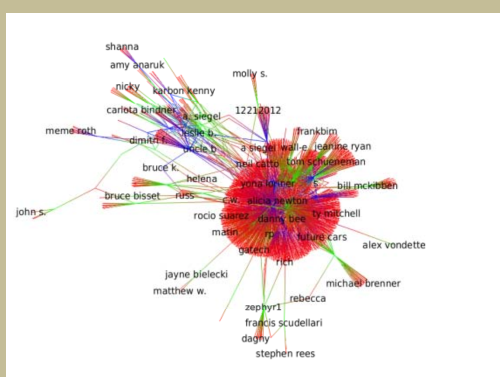
Connect-the-dots

SmartGrid



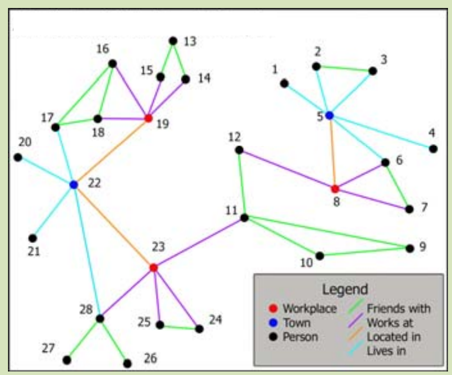
N-x contingency analysis

Blog Analysis



Community thought leaders

Semantic Web



People, Places, & Actions

Security

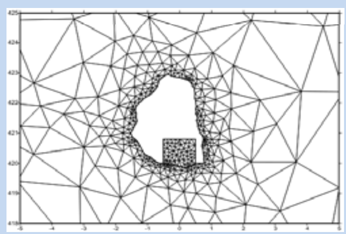
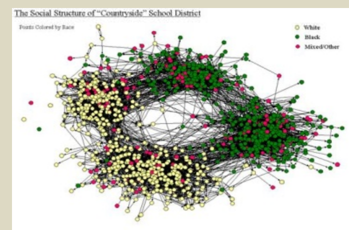
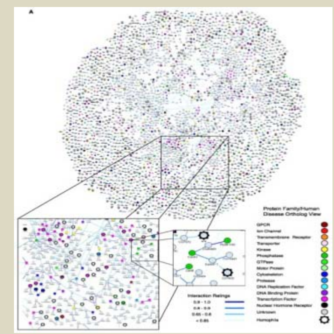
Anomaly detection

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle Since 1965

Graphs are not grids

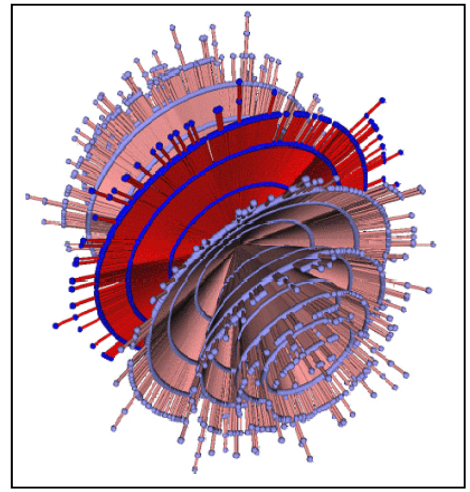
- Graphs arising in informatics are very different from the grids used in scientific computing

Scientific Grids	Graphs for Data Informatics
 	 
Static or slowly involving	Dynamic
Planar	Non-planar
Nearest neighbor communication	Communications are non-local and dynamic
Work performed per cell or node	Work performed by crawlers or autonomous agents
Work modifies local data	Work modifies data in many places

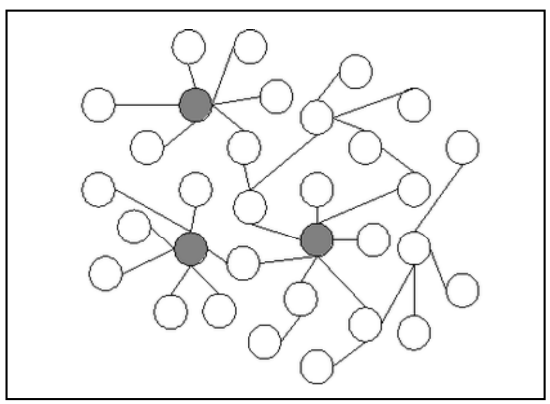
Small-world and scale-free

- In low diameter graphs
  - work explodes
  - difficult to partition
  - high percentage of nodes are visited
- In scale-free graphs
  - difficult to partition
  - work concentrates in a few nodes

"Six degrees of separation"



Large hubs are in grey



Graph methods

- Paths
  - Shortest path
  - Betweenness
  - Min/max flow
- Structures
  - Spanning trees
  - Connected components
  - Graph isomorphism
- Groups
  - Matching/Coloring
  - Partitioning
  - Equivalence
- Orderings
  - Priority
  - Topological
  - Temporal

Influential Factors

- Degree distribution
  - Normal
  - Scale-free
- Planar or non-planar
- Static or dynamic
- Weighted or unweighted
  - Weight distribution
- Typed or untyped edges

Load imbalance
Non-planar

Difficult to partition

Concurrent inserts
and deletions

Challenges

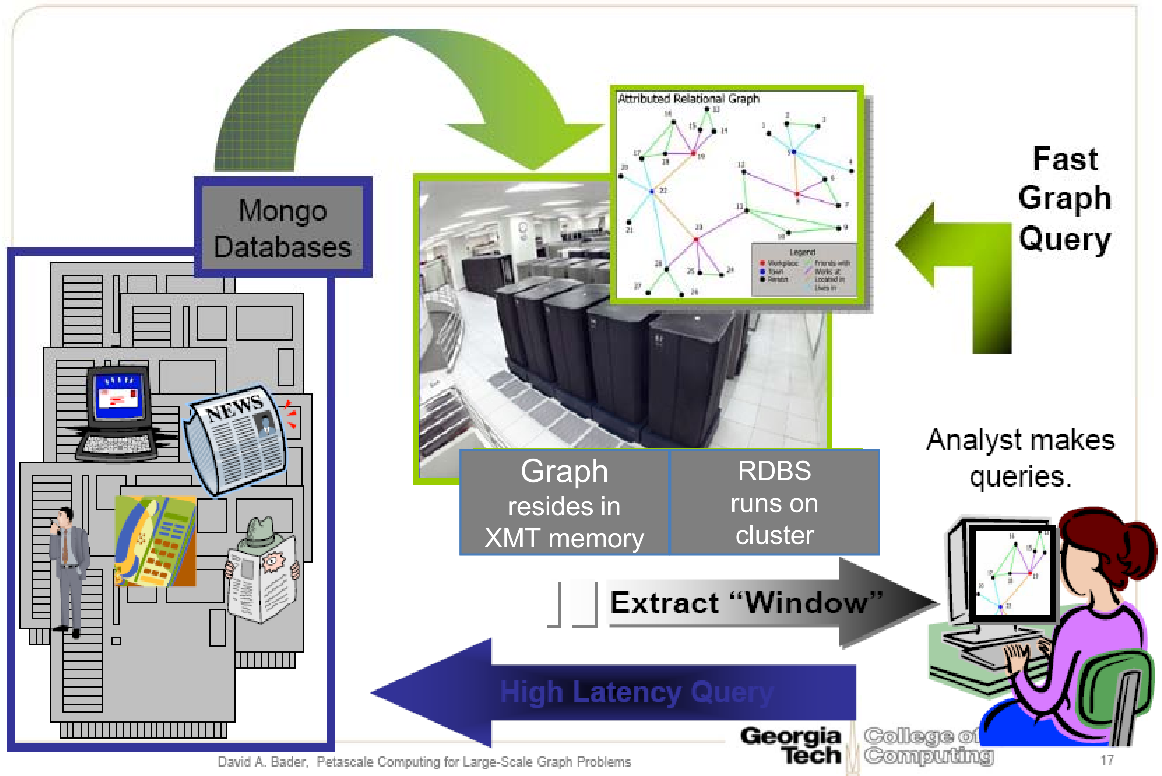
- Problem size
  - Ton of bytes, not ton of flops
- Little data locality
  - Have only parallelism to tolerate latencies
- Low computation to communication ratio
  - Single word access
  - Threads limited by loads and stores
- Synchronization points are simple elements
  - Node, edge, record
- Work tends to be dynamic and imbalanced
  - Let any processor execute any thread

System implications

- Global address space
  - Direct loads and stores
  - MC and network support for single word accesses
- Multi-threaded processors
  - Single cycle context switching
  - Multiple outstanding loads and stores per thread
- Full-and-empty bits
- Message driven operations
  - Dynamic work queues
  - Hardware support for thread migration

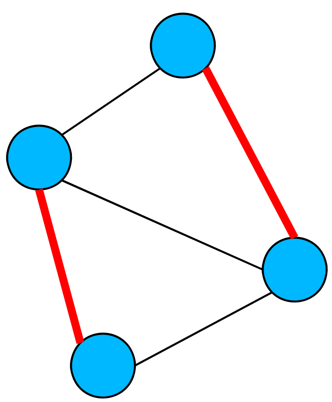
Systems for large-scale analytics

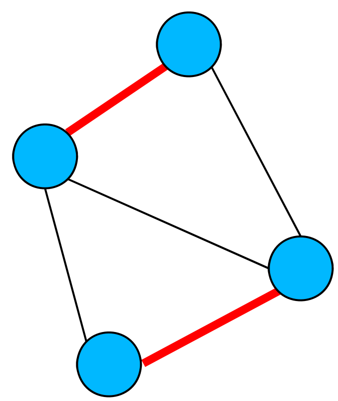
Netezza TwinFin

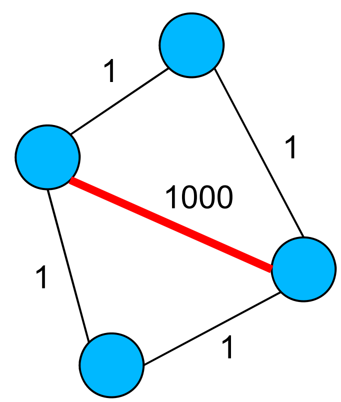


Maximum matching

- A matching M is a subset of edges such that no two edges in M are incident on the same vertex
- Maximum matching is a matching that maximizes some cost function
  - Number of edges
  - Weights

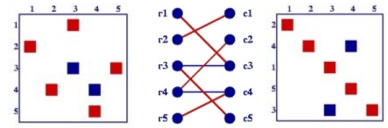


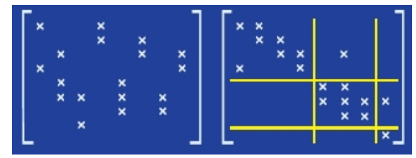


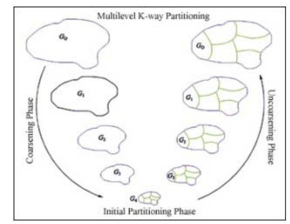


Application of matching

- Sparse linear solvers
- Block triangular form
- Graph partitioners
- Bioinformatics
- Web technology
- Sparse derivative computations
- High speed network switching

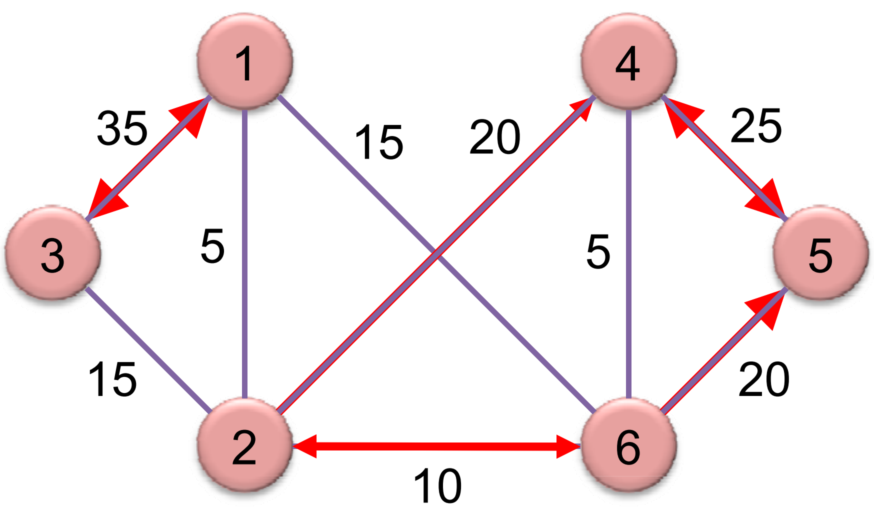






Maximum weight matching algorithms

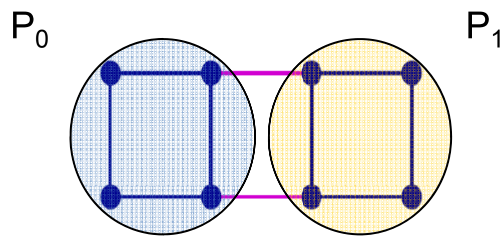
- Polynomial time algorithm first due to Edmonds
  - *Path, trees, and flower method*
- Approximate “greedy” algorithm is fast, simple, and usually give good results



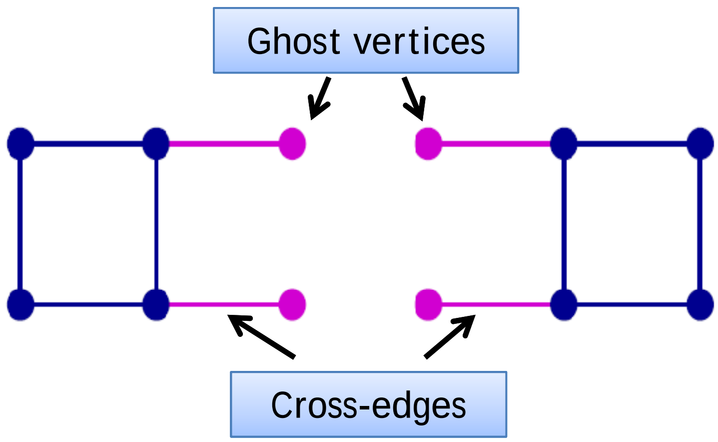
Hoepman’s Algorithm

Parallel algorithm (Halappanavar, Dobrian, and Pothen)

- Uses queues to maintain proposals and status
- On clusters (IBM Blue Gene/P)
  - Partition graph
  - Ghost cells for non-local neighbors
  - Local queues … pass as bulk messages



- On shared memory machine (Cray XMT)
  - Shared queues

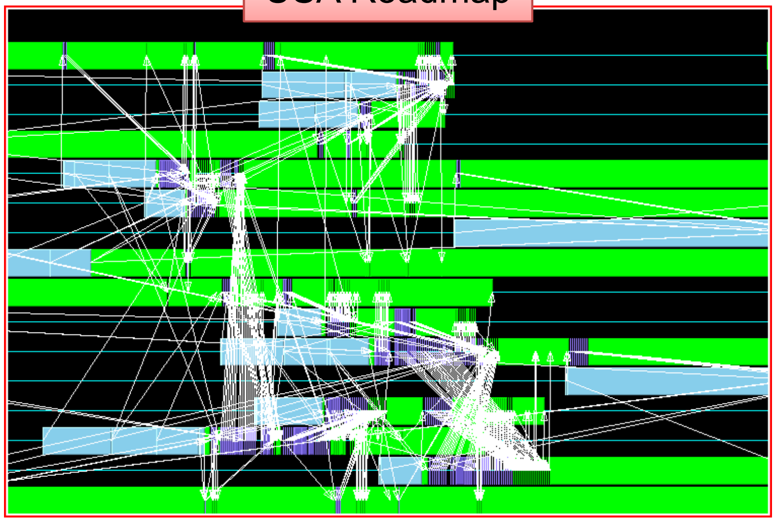


DM parallel algorithm (Halappanavar, Dobrian, and Pothen)

1. Initialize data structures
2. Phase 1: Independent Computation
   - Identify locally-dominant edges and match
   - Send messages as needed (cross-edges)
3. Phase 2: Shared Computation
   - Receive and process messages
   - Match if locally-dominant
   - Send messages as needed
   - Repeat until no more edges can be matched

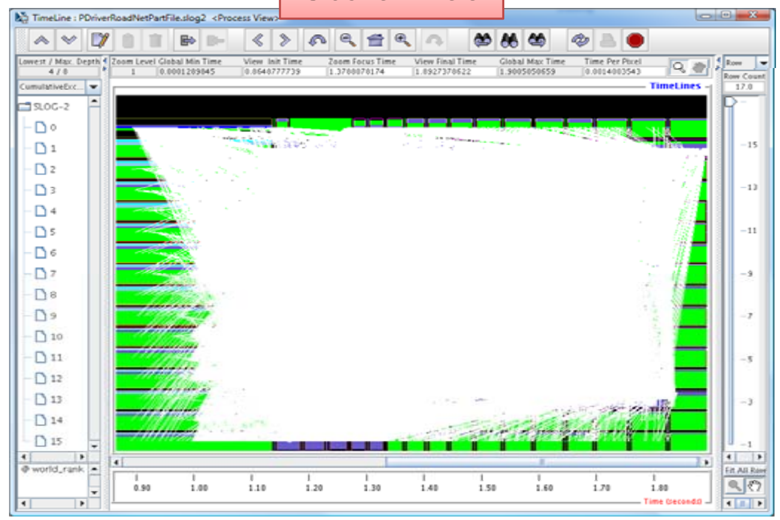
Grids, Erdős–Rényi, and Scale-Free Graphs

USA Roadmap

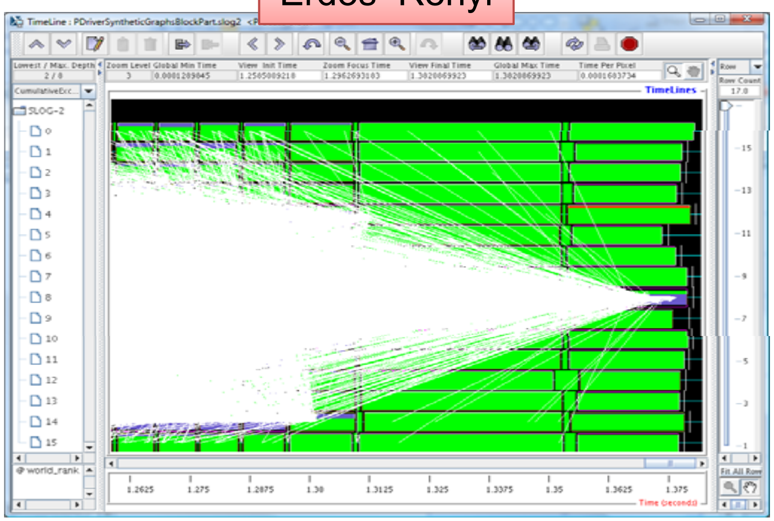


METIS Partitioner

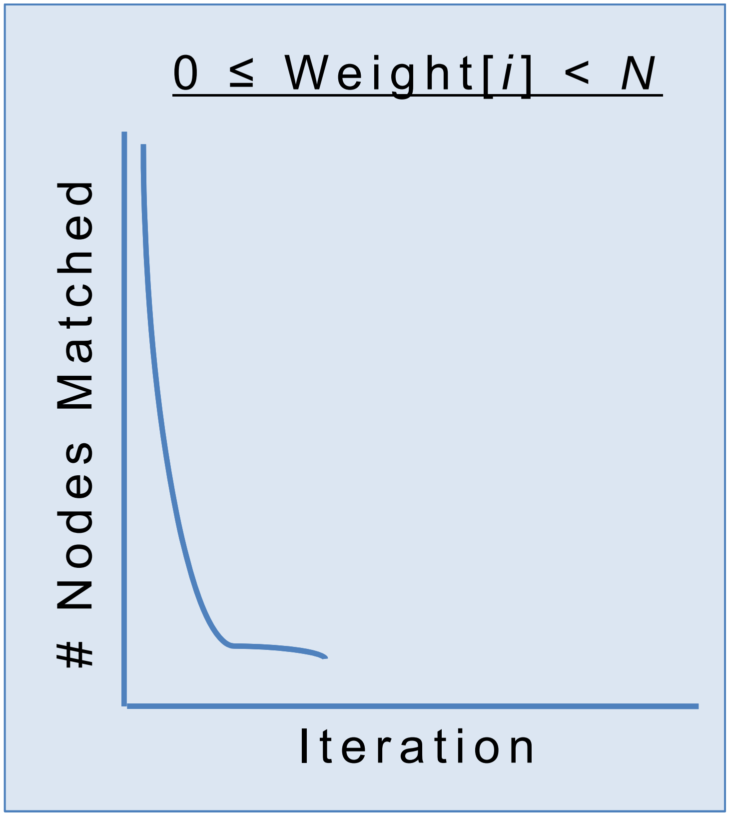
Scale-Free

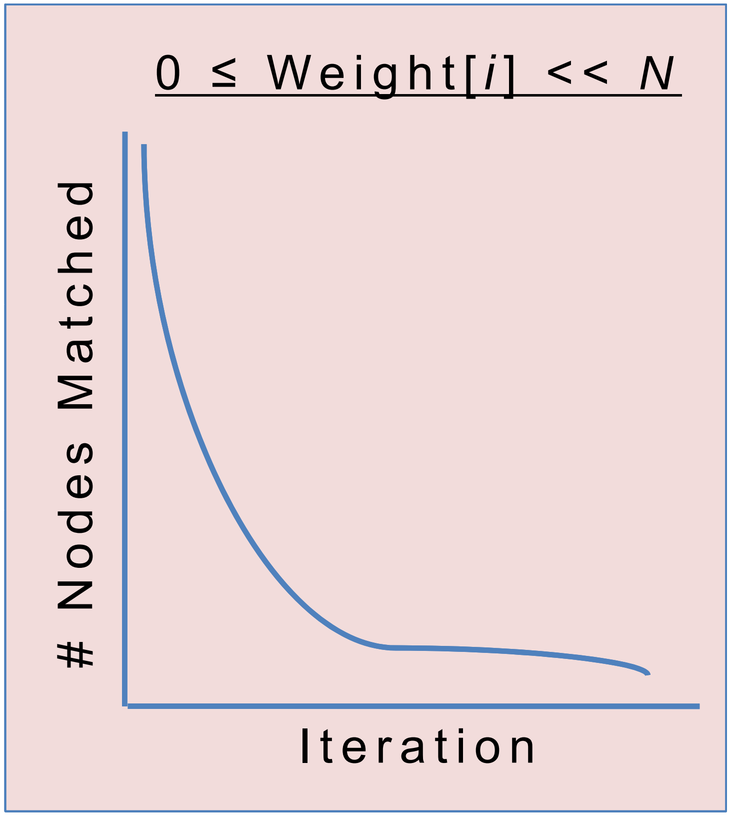


Erdős–Rényi



Effect of weight distribution



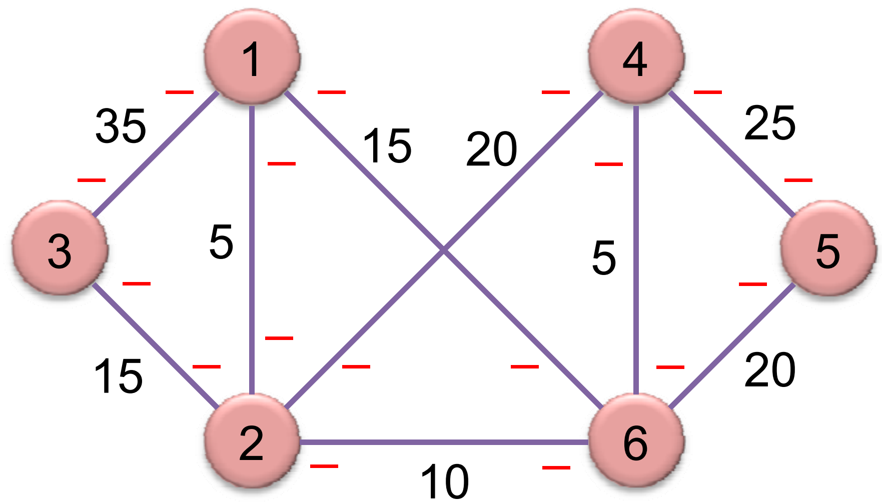


Real World

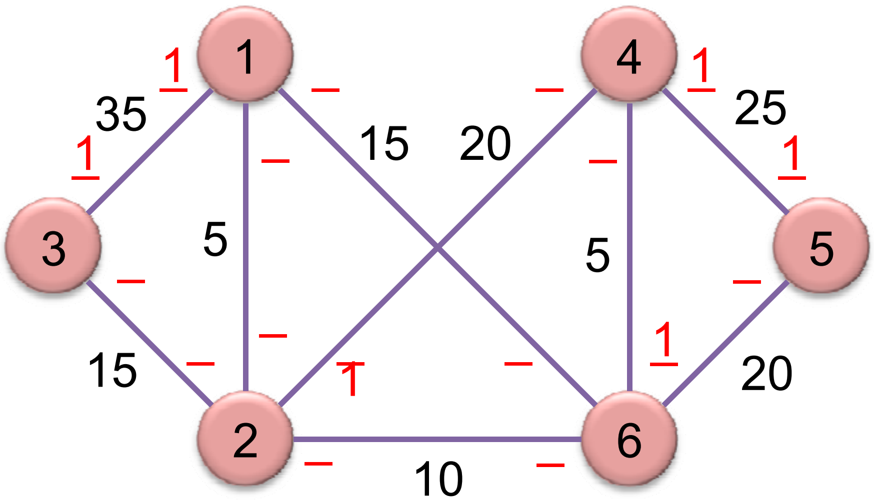
DataFlow algorithm

- Nodes propose by setting signals on edges
  - Very fine grain synchronization ➔ full/empty bits
- Performance is insensitive to structure and weights
- Watch out for deadlock !!!

Ask me later
Last deadlock took me **3 days** to debug

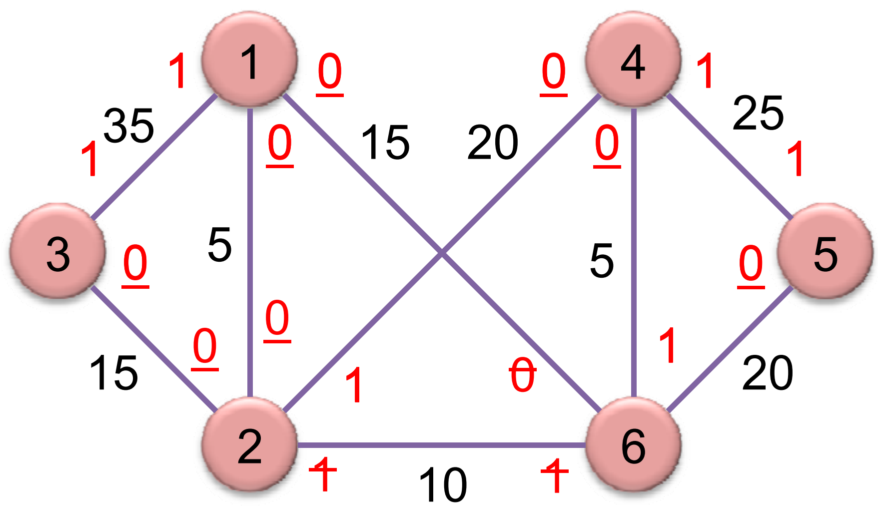


DataFlow flow



- Each node sets signal on its side of heaviest edge to 1
- Reads companion signal

DataFlow flow (cont.)



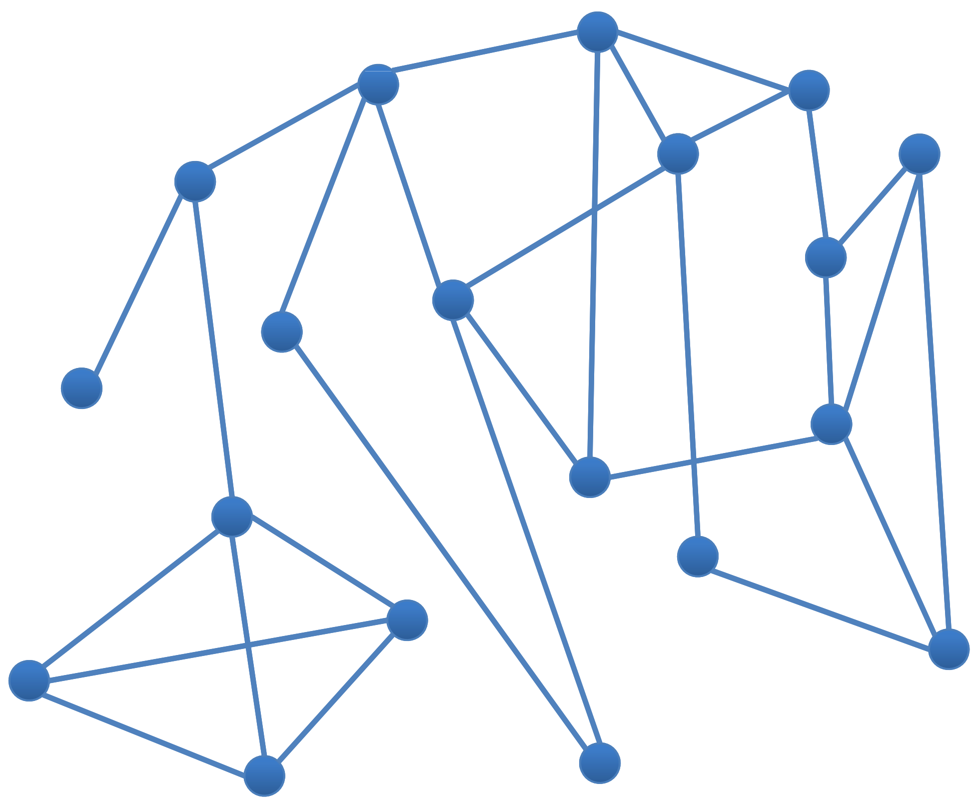
- If companion signal is 1, then set signal of other edges to 0 and stop
- else set signal on next heaviest edge to 1

Performance

64 processors

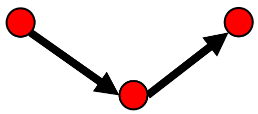
Graphz	Clusters	XMT Q	XMT DF
Square Grid, 2^{30}, 2^{30}	15.2 s (IBM BG/L)	19.9 s	9.31
US Roadmap	0.14 s (XT4)	0.5 s	0.11
RMAT, 2^{27}, 2^{30}, 2^{27}	X	5.8 s	2.12 s
RMAT, 2^{27}, 2^{30}, 2^{3}	X	8.0 s	2.12 s

When parallelism is intra-task

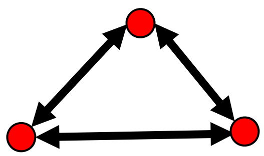


- Store graph in shared memory as accesses are non-local
- Almost no computation per access, so only direct loads and stores are efficient
- Synchronization points are nodes

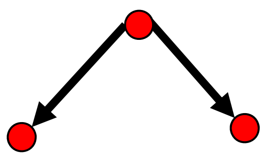
Triad Patterns Have Meaning



Transmission,
Transactions,
Chain of Command

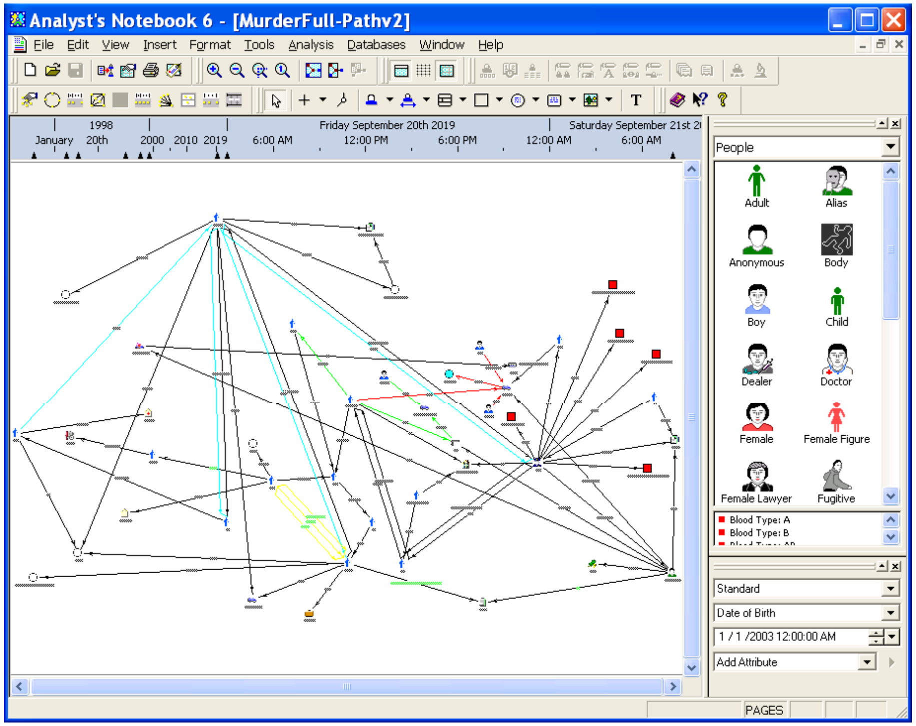


Close-Knit Groups,
Strong Dependencies



Hierarchies

Bridges,
Points of Disruption



Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by **Battelle** *Since 1965*

Subquadratic Triadic Analysis (Batagelj and Mrvar)

INPUT: *G = (V, E)*
OUTPUT: vector *Census* with frequencies of triadic types

```
1        for i := 1 to 16 do Census[i] := 0;

2        for each u ∈ V do
2.1      for each v ∈ Ê(u) do
2.1.1        if u > v then continue;
2.1.2        S := Ê(u) ∪ Ê(v) \ {v, u};
2.1.3        if uEv ∧ vEu then TriType := 3 else TriType := 2;
2.1.4        Census[TriType] += (n − |S| − 2);
2.1.4        for each w ∈ S do
2.1.4.1          if v > w ∧ !(u < w ∧ w < v ∧ ¬uÊw) continue;
2.1.4.2          TriType := Tricode(u,v,w);
2.1.4.3          Census[TriType] ++;
3         sum := 0;
4         for i := 2 to 16 do sum := sum + Census[i];
5        Census[1] := (1/6)n(n − 1)(n − 2) − sum;
```

collapse loops
“for all edges”

while loop is local to thread

Computational complexity: $O(n^2)$ for sparse graphs
$O(n^3)$ for complete graphs

XMT code for while loop

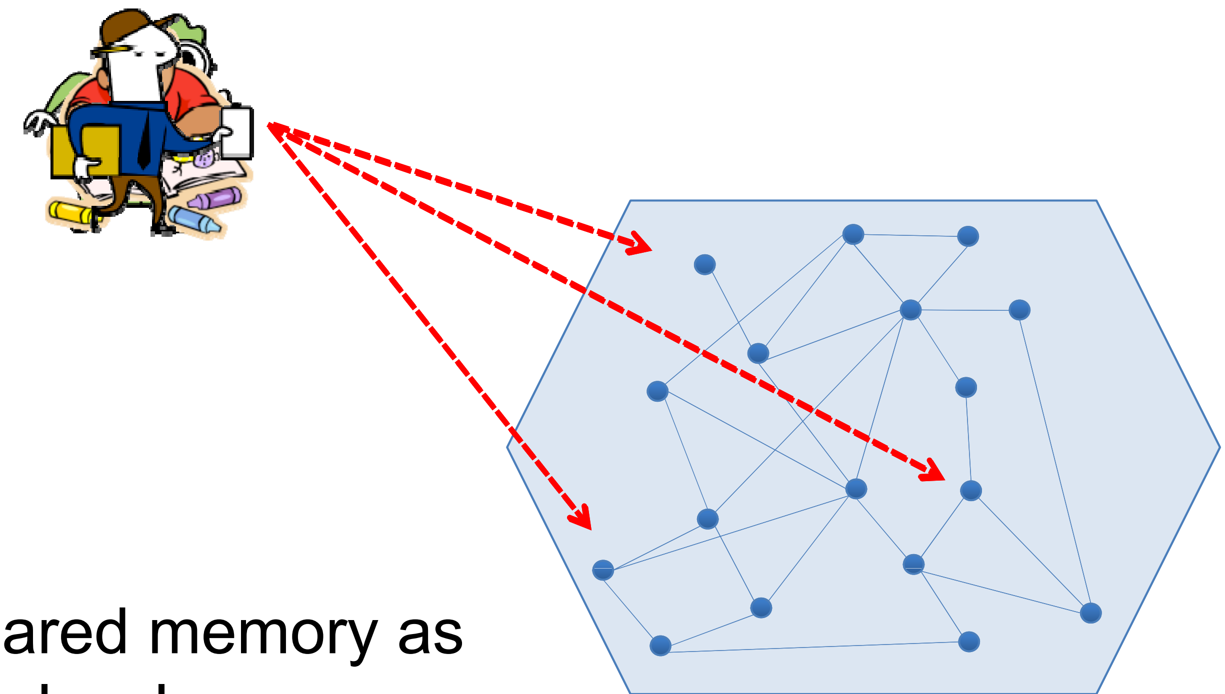
```
#pragma mta dynamic schedule
for each edge (U, V) {
  …
  while ((VV < nNodes) || (UU < nNodes)) {
    if (VV < UU) {
      if (U  < VV) {
        int VWedge = EdgeType(V_neighbors, vv)                 // Read next edgetype of V
        int code = (VWedge << 2) + VUedge;
        int type = triad_table[code];                          // Global read
        census[type] ++;                                       // Sync Global write
      }
      …
      VV = (vv >= V_count) ? nNodes : NBR(V_neighbors, vv);      // Read next neighbor of V

 } else if (VV == UU) {
      if (U < VV) {     // W is a neighbor of V, so clause 4 is false
        int VWedge = EdgeType(V_neighbors, vv);                // Read next edgetype of V
        int UWedge = EdgeType(U_neighbors, uu);                // Read next edgetype of U
        int code = (UWedge << 4) + (VWedge << 2) + VUedge;
        int type = triad_table[code];                          // Global read
        census[type] ++;                                       // Sync Global write
      }
      …
      VV = (vv >= V_count) ? nNodes : NBR(V_neighbors,vv);       // Read next neighbor of V
      UU = (uu >= U_count) ? nNodes : NBR(U_neighbors, uu);      // Read next neighbor of U
    } else {
    …
} } }
```

Optimizing thread reads

- Threads often access remote data structures in order
  - V and U neighbor list
  - V and U edgetype list
- Move multiple words when appropriate
  - Saves BW
  - Saves power
  - Reduces latency
- Hardware? Compiler? Language?
  - CASS is developing cycle accurate simulator
  - CASS is investigating compiler and runtime system mods

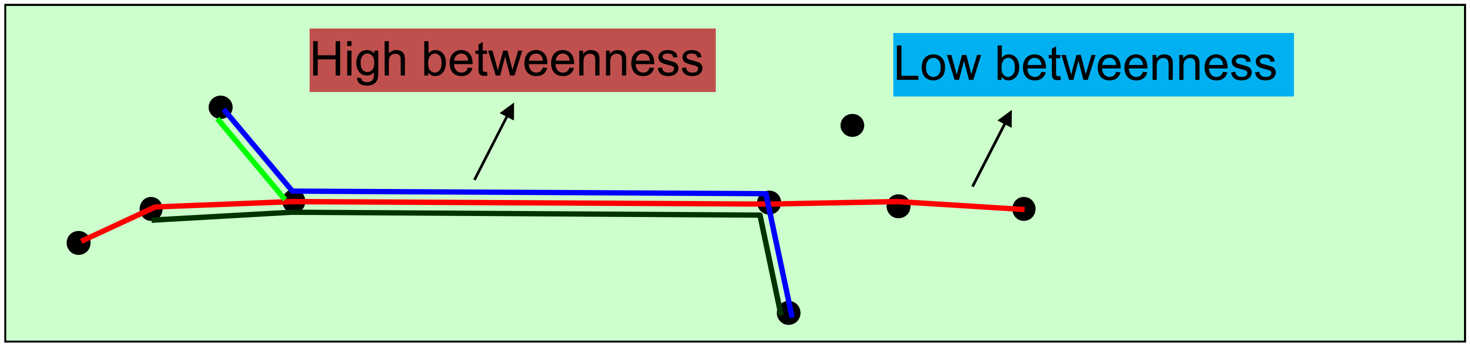
When parallelism is inter-task



- Store graph in shared memory as accesses are non-local
- Threads maintain private work space
  - Store in nearby memory
  - Coherency and synchronization hardware is unnecessary

Contingency analysis for the Power Grid

- Use BC to identify critical power transmission lines



- Execute variation of Dijskstra's shortest path algorithm for all pairs of nodes
  - Embarrassingly parallel, but tasks are irregular and use the whole graph

Thread private data

- Each thread maintains
  - Array of records
  - Heap of visited nodes
  - Previous node list
- Data is thread private, so no need
  - … to store in global memory
  - … to provide coherence
  - … to provide synchronization
- Provide local memory for thread private data

Nearby memory on XMT

- XMT has both global memory and nearby memory
  - Average latency to global memory is 740 clock cycles
  - Average latency to nearby memory is 90 clock cycles
- MMAP() provides access to nearby memory
- CASS is developing
  - APIs for managing nearby memory
  - Parallel-aware allocator for nearby memory
  - Compiler modifications to recognize thread private data structures

Performance gains

- Western US WECC power grid
  - 14,000 nodes,1,400 sources

Processors	All Global	Global/Nearby	Improvement
2	28.69 s (29%)	16.69 s (47%)	**1.72x**
4	15.00 s (27%)	8.85 s (45%)	**1.69x**
8	7.93 s (26%)	4.69 s (43%)	**1.69x**
16	4.24 s (24%)	2.44 s (39%)	**1.74x**

Time Utilization

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle Since 1965

Summary

- The new HPC is irregular and sparse
  - There are commercial and consumer applications
  - If the applications are important enough, machines will be built
- HPC is too large and too diverse for “one size fits all”
- Develop hybrid computing systems and programming models