Linear Algebraic Graph Algorithms for Back End Processing

Jeremy Kepner, Nadya Bliss, and Eric Robinson

MIT Lincoln Laboratory

This work is sponsored by the Department of Defense under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Outline

- **Introduction**
- Power Law Graphs
- Graph Benchmark
- Results
- Summary

- *Post Detection Processing*
- *Sparse Matrix Duality*
- *Approach*

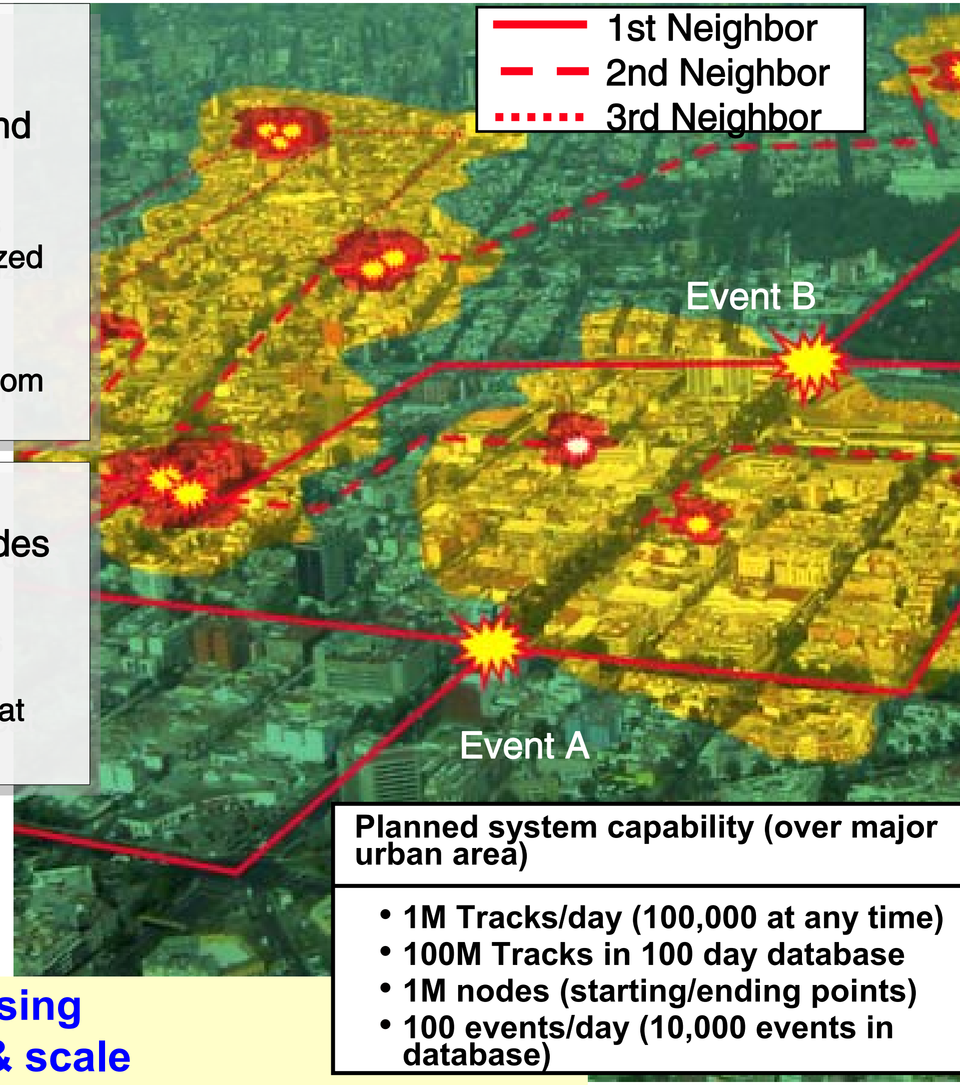
Statistical Network Detection

Problem: Forensic Back-Tracking

- Currently, significant analyst effort dedicated to manually identifying links between threat events and their immediate precursor sites
  - Days of manual effort to fully explore candidate tracks
  - Correlations missed unless recurring sites are recognized by analysts
  - Precursor sites may be low-value staging areas
  - Manual analysis will not support further backtracking from staging areas to potentially higher-value sites

Concept: Statistical Network Detection

- Develop graph algorithms to identify adversary nodes by estimating connectivity to known events
  - Tracks describe graph between known sites or events which act as sources
  - Unknown sites are detected by the aggregation of threat propagated over many potential connections



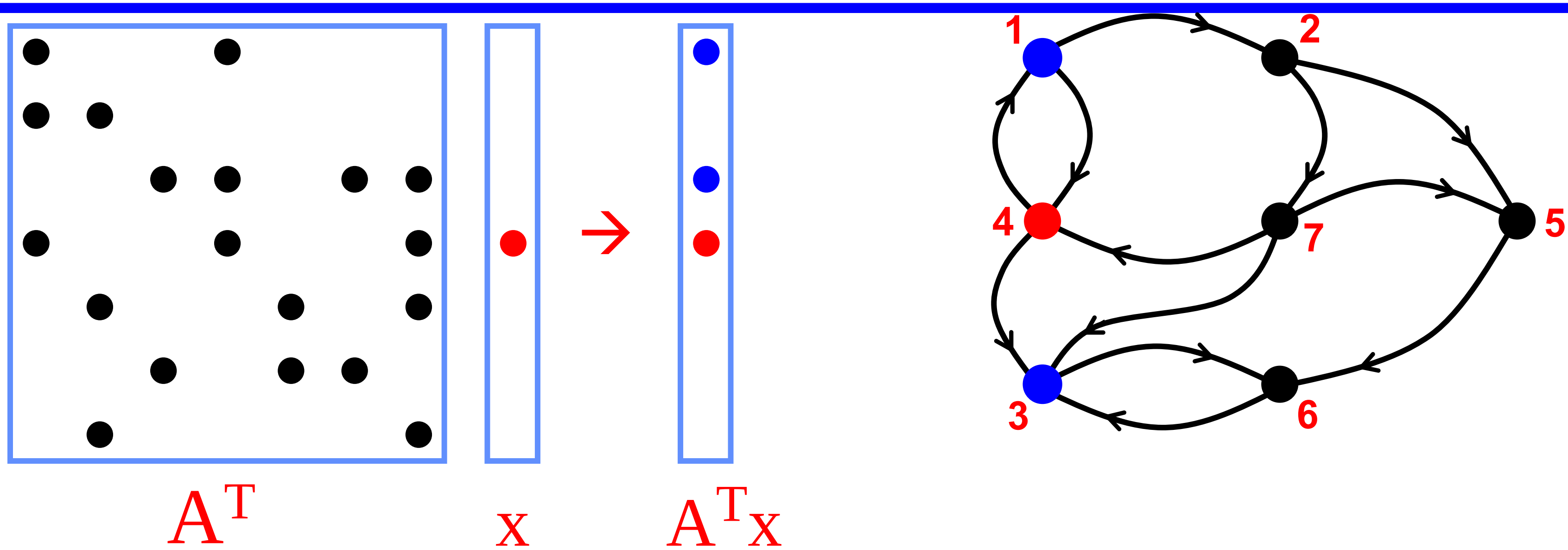
Planned system capability (over major urban area)

- 1M Tracks/day (100,000 at any time)
- 100M Tracks in 100 day database
- 1M nodes (starting/ending points)
- 100 events/day (10,000 events in database)

Computationally demanding graph processing

- – ~ 10^6 seconds based on benchmarks & scale
- – ~ 10^3 seconds needed for effective CONOPS (1000x improvement)

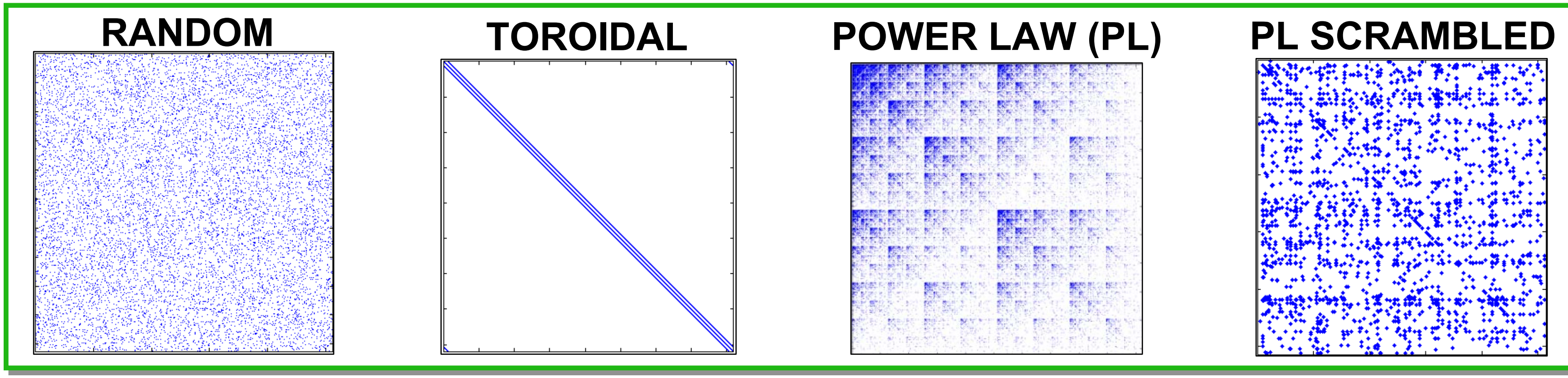
Graphs as Matrices



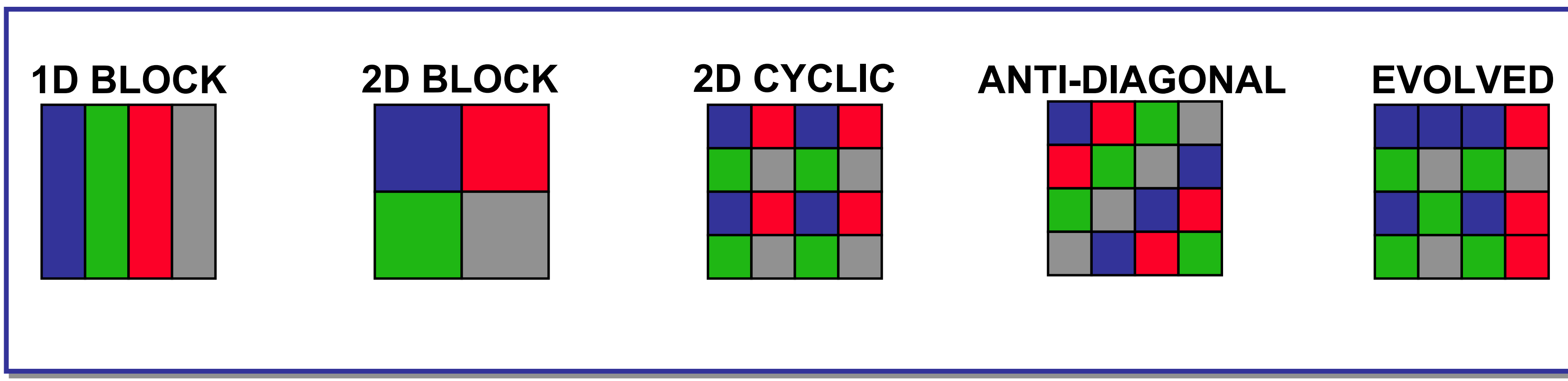
- **Graphs can be represented as a sparse matrices**
  - **Multiply by adjacency matrix → step to neighbor vertices**
  - **Work-efficient implementation from sparse data structures**
- **Most algorithms reduce to products on semi-rings: C = A “+”.“x” B**
  - **“x” : associative, distributes over “+”**
  - **□“+” : associative, commutative**
  - **Examples: +.* min.+ or.and**

Distributed Array Mapping

Adjacency Matrix Types:



Distributions:



Sparse Matrix duality provides a natural way of exploiting distributed data distributions

Algorithm Comparison

Algorithm (Problem)	Canonical Complexity	Array-Based Complexity	Critical Path (for array)
Bellman-Ford (SSSP)	$\Theta(mn)$	$\Theta(mn)$	$\Theta(n)$
Generalized B-F (APSP)	NA	$\Theta(n^3 \log n)$	$\Theta(\log n)$
Floyd-Warshall (APSP)	$\Theta(n^3)$	$\Theta(n^3)$	$\Theta(n)$
Prim (MST)	$\Theta(m+n \log n)$	$\Theta(n^2)$	$\Theta(n)$
Borůvka (MST)	$\Theta(m \log n)$	$\Theta(m \log n)$	$\Theta(\log^2 n)$
Edmonds-Karp (Max Flow)	$\Theta(m^2n)$	$\Theta(m^2n)$	$\Theta(mn)$
Push-Relabel (Max Flow)	$\Theta(mn^2)$ (or $\Theta(n^3)$)	$O(mn^2)$	?
Greedy MIS (MIS)	$\Theta(m+n \log n)$	$\Theta(mn+n^2)$	$\Theta(n)$
Luby (MIS)	$\Theta(m+n \log n)$	$\Theta(m \log n)$	$\Theta(\log n)$

Majority of selected algorithms can be represented with array-based constructs with equivalent complexity.

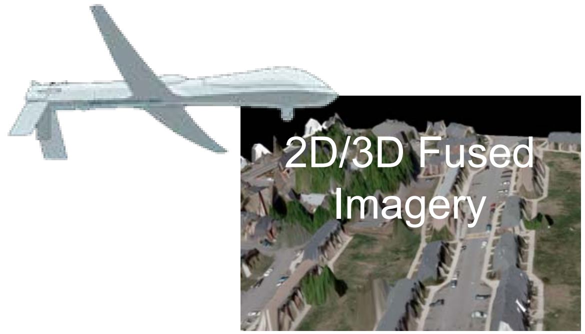
($n = |V|$ and $m = |E|$.)

A few DoD Applications using Graphs

FORENSIC BACKTRACKING

- Identify key staging and logistic sites areas from persistent surveillance of vehicle tracks

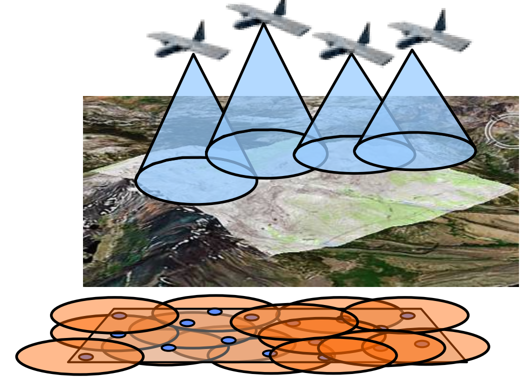
DATA FUSION



- Bayes nets for fusing imagery and ladar for better on board tracking

TOPOLOGICAL DATA ANALYSIS

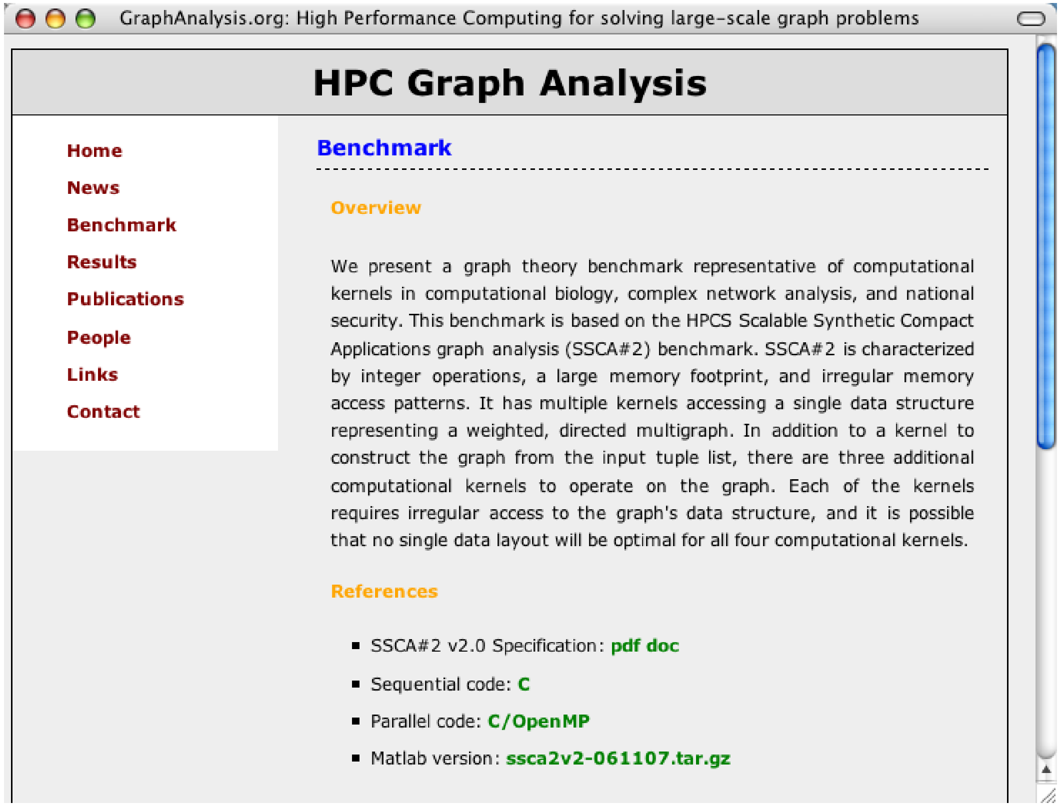
- Higher dimension graph analysis to determine sensor net coverage [Jadbabaie]



Application	Key Algorithm	Key Semiring Operation
• Subspace reduction	• Minimal Spanning Trees	$\mathbf{X}$ +.* $\mathbf{A}$ +.* $\mathbf{X}^T$
• Identifying staging areas	• Betweenness Centrality	$\mathbf{A}$ +.* $\mathbf{B}$
• Feature aided 2D/3D fusion	• Bayesian belief propagation	$\mathcal{A}$ +.* $\mathcal{B}$ (**$\mathcal{A}$, $\mathcal{B}$ tensors**)
• Finding cycles on complexes	• Single source shortest path	$\mathbf{D}$ min.+ $\mathcal{A}$ (**$\mathcal{A}$ tensor**)

Approach: Graph Theory Benchmark

- Scalable benchmark specified by graph community
- Goal
  - Stress parallel computer architecture
- Key data
  - Very large Kronecker graph
- Key algorithm
  - Betweenness Centrality

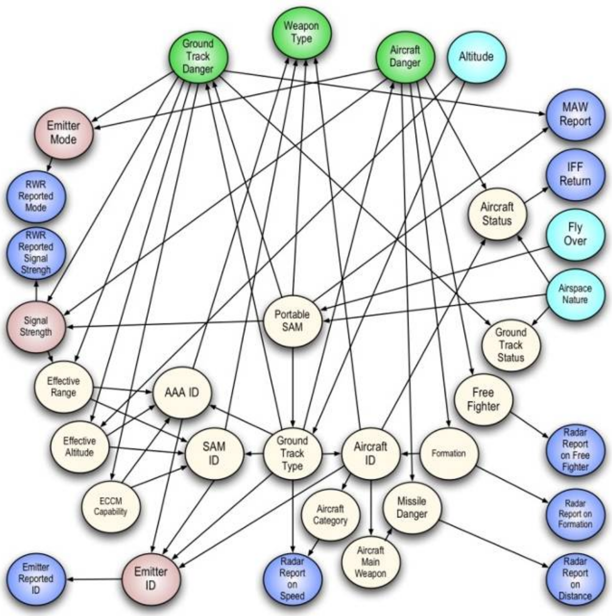


- Computes number of shortest paths each vertex is on
  - Measure of vertex “importance”
  - Poor efficiency on conventional computers

Outline

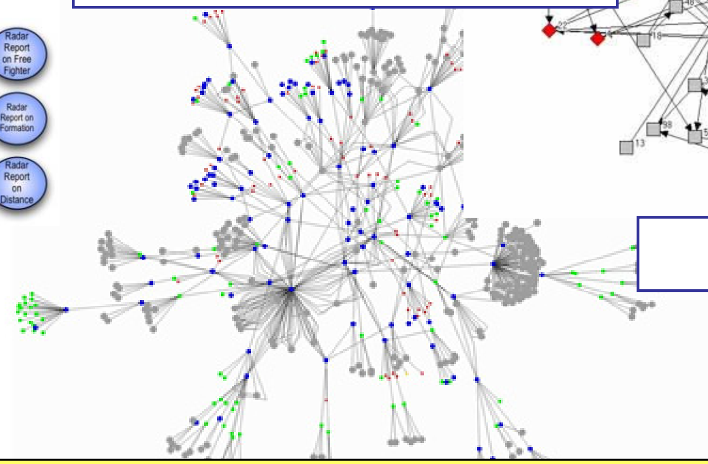
- **Introduction**
- **Power Law Graphs**
  - *Kronecker Model*
  - *Analytic Results*
- **Graph Benchmark**
- **Results**
- **Summary**

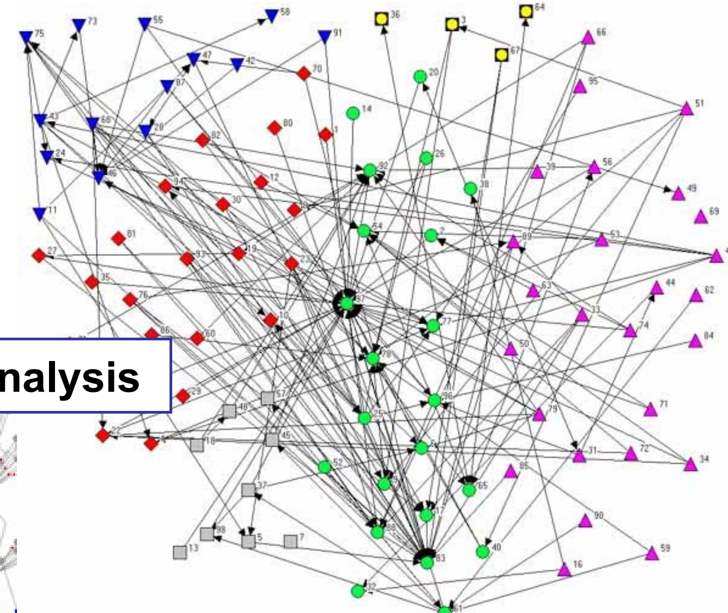
Power Law Graphs



Target Identification

Social Network Analysis



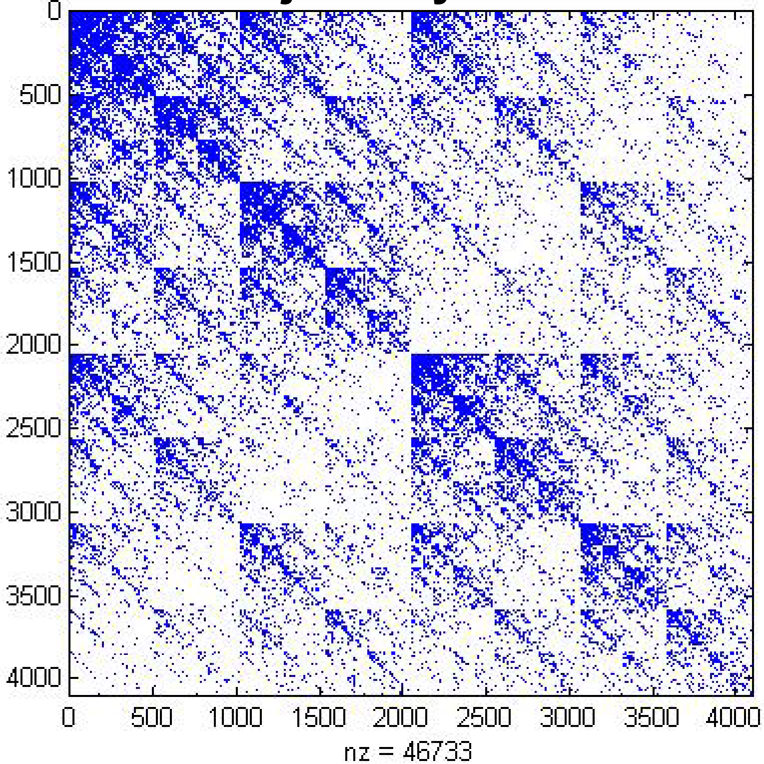


Anomaly Detection

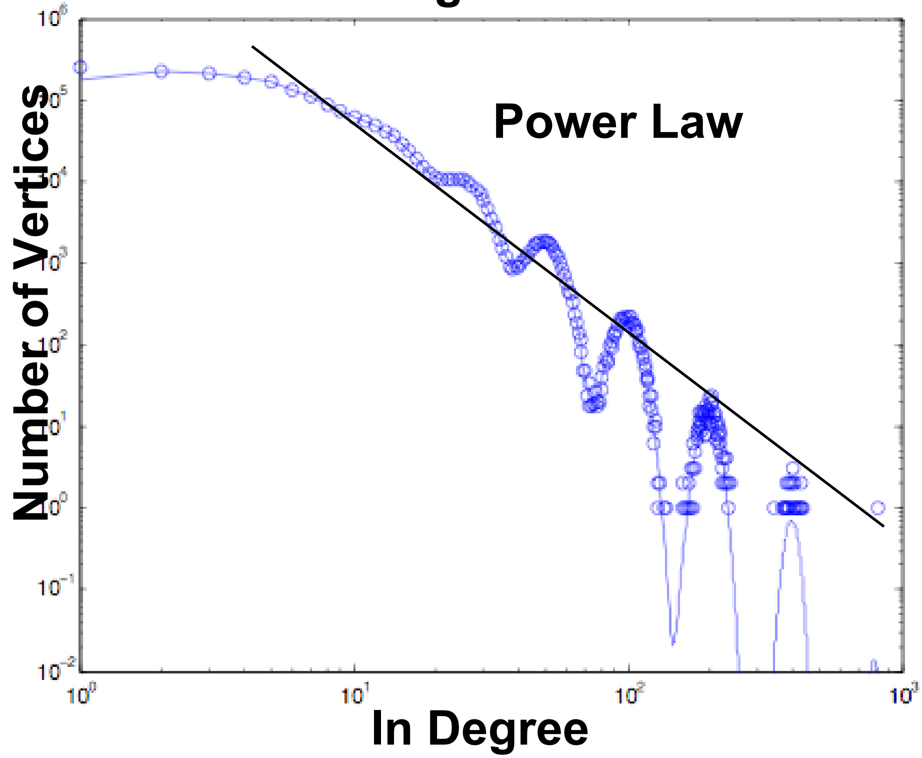
- Many graph algorithms must operate on power law graphs
- Most nodes have a few edges
- A few nodes have many edges

Modeling of Power Law Graphs

Adjacency Matrix



Vertex In Degree Distribution



- **Real world data (internet, social networks, …) has connections on all scales (i.e power law)**
- **Can be modeled with Kronecker Graphs: $G^{\otimes k} = G^{\otimes k-1} \otimes G$**
  – **Where "⊗"denotes the Kronecker product of two matrices**

Kronecker Products and Graph

Kronecker Product

- Let B be a $N_B x N_B$ matrix
- Let C be a $N_C x N_C$ matrix
- Then the Kronecker product of B and C will produce a $N_B N_C x N_B N_C$ matrix A:

$$A = B \otimes C = \begin{pmatrix} b_{1,1}C & b_{1,2}C & \dots & b_{1,M_B}C \\ b_{2,1}C & b_{2,2}C & \dots & b_{2,M_B}C \\ \vdots & \vdots & & \vdots \\ b_{N_B,1}C & b_{N_B,2}C & \dots & b_{N_B,M_B}C \end{pmatrix}$$

Kronecker Graph (Leskovec 2005 & Chakrabati 2004)

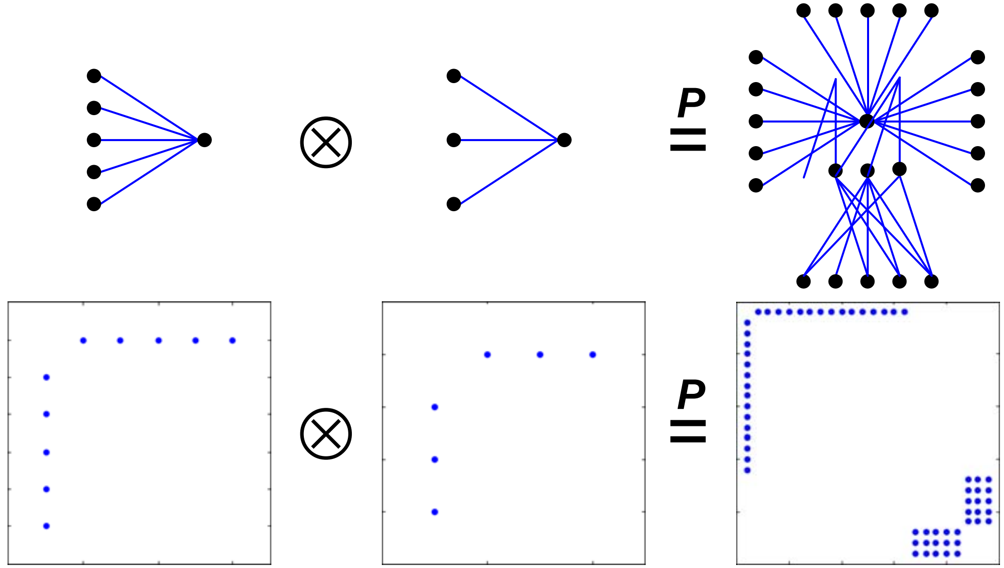
- Let G be a NxN adjacency matrix
- Kronecker exponent to the power k is:

$$G^{\otimes k} = G^{\otimes k-1} \otimes G$$

Kronecker Product of a Bipartite Graph

$\overset{P}{=}$

Equal with the right permutation



$$B(5,1) \otimes B(3,1) \overset{P}{=} B(15,1) \cup B(3,5)$$

- **Fundamental result [Weischel 1962] is that the Kronecker product of two complete bipartite graphs is two complete bipartite graphs**
- **More generally**

$$B(n_1, m_1) \otimes B(n_2, m_2) \overset{P}{=} B(n_1 n_2, m_1 m_2) \cup B(n_2 m_1, n_1 m_2)$$

Degree Distribution of Bipartite Kronecker Graphs

- **Kronecker exponent of a bipartite graph produces many independent bipartite graphs**

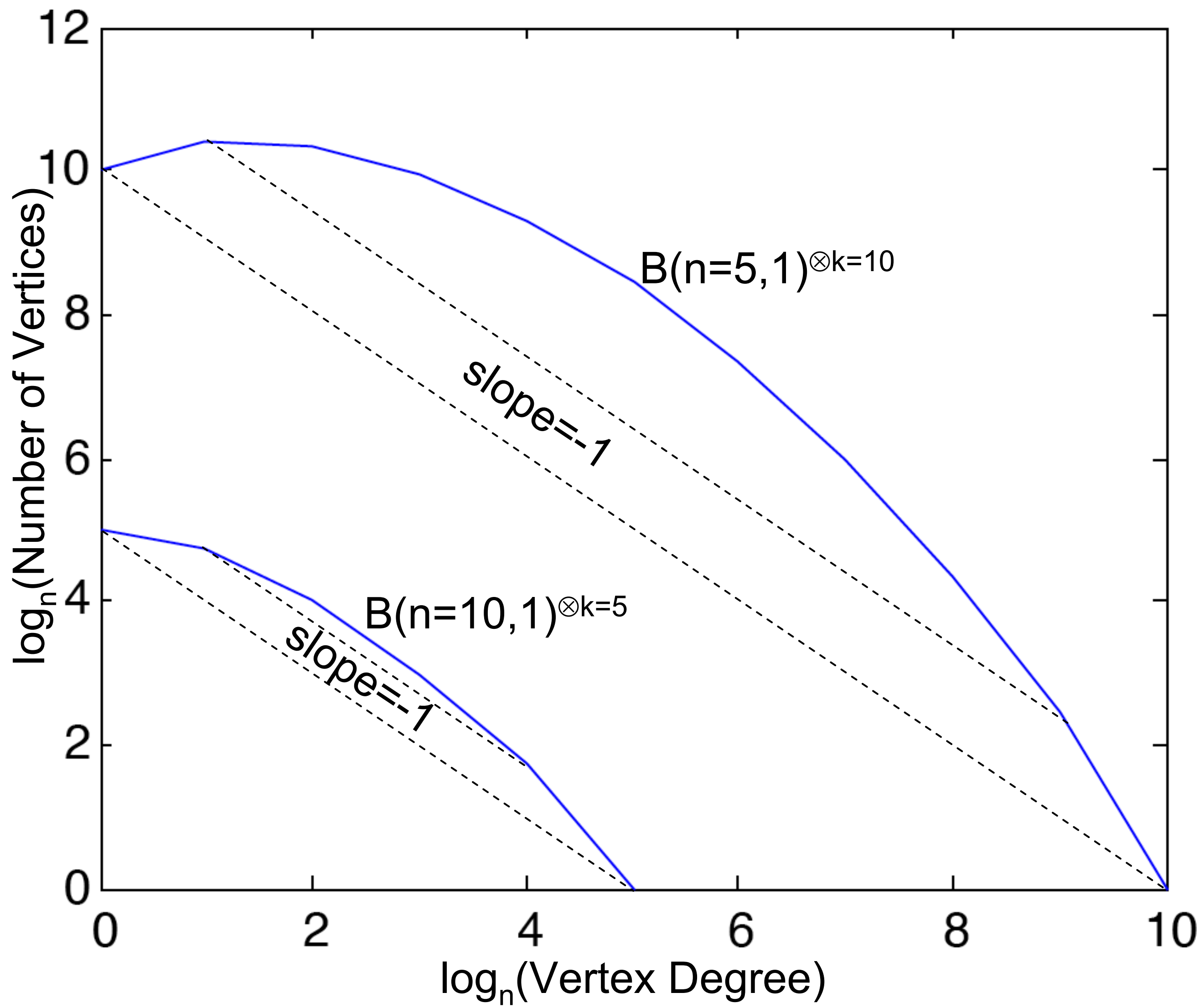
$$B(n,m)^{\otimes k} \overset{P}{=} \bigcup_{r=0}^{k-1} \bigcup^{\binom{k-1}{r}} B(n^{k-r}m^r, n^r m^{k-r})$$

- **Only k+1 different kinds of nodes in this graph, with degree distribution**

$$Count[Deg = n^r m^{k-r}] = \binom{k}{r} n^{k-r} m^r$$

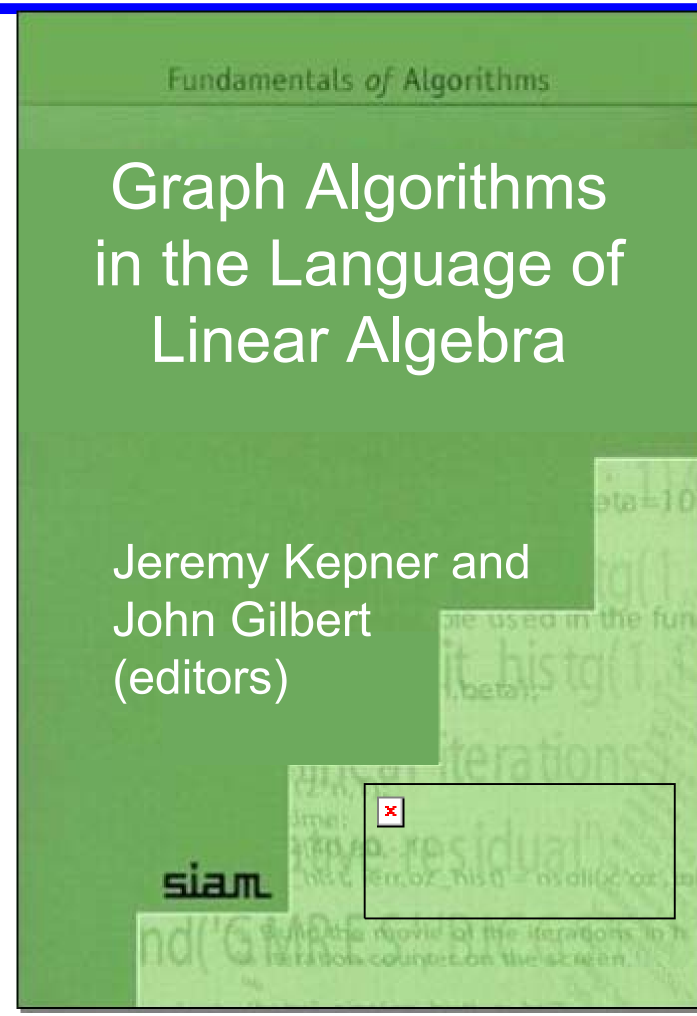
Explicit Degree Distribution

- **Kronecker exponent of bipartite graph naturally produces exponential distribution**
- **Provides a natural framework for modeling “background” and “foreground” graph signatures**
- **Detection theory for graphs?**



Reference

- Book: "Graph Algorithms in the Language of Linear Algebra"
- Editors: Kepner (MIT-LL) and Gilbert (UCSB)
- Contributors
  - Bader (Ga Tech)
  - Chakrabart (CMU)
  - Dunlavy (Sandia)
  - Faloutsos (CMU)
  - Fineman (MIT-LL & MIT)
  - Gilbert (UCSB)
  - Kahn (MIT-LL & Brown)
  - Kegelmeyer (Sandia)
  - Kepner (MIT-LL)
  - Kleinberg (Cornell)
  - Kolda (Sandia)
  - Leskovec (CMU)
  - Madduri (Ga Tech)
  - Robinson (MIT-LL & NEU), Shah (UCSB)

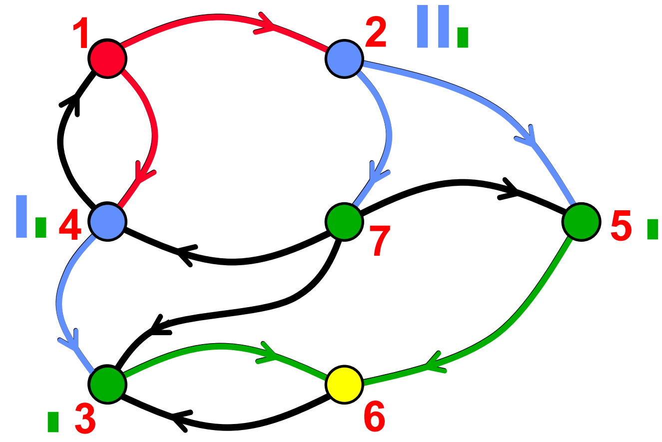


Outline

- Introduction
- Power Law Graphs
- **Graph Benchmark**
  - *Post Detection Processing*
  - *Sparse Matrix Duality*
  - *Approach*
- Results
- Summary

Graph Processing Kernel
-Vertex Betweenness Centrality-

Betweenness centrality is a measure for estimating importance of a vertex in a graph



Algorithm Description

1. Starting at vertex *v*
- compute shortest paths to all other vertices
- for each reachable vertex, for each path it appears on, assign a token

2. Repeat for all vertices

3. Accumulate across all vertices

Vertices that appear on most shortest paths have the highest betweenness centrality measure

Rules for adding tokens (betweenness value) to vertices
- Tokens are not added to start or end of the path
- Tokens are normalized by the number of shortest paths between any two vertices

Graph traversal starting at vertex 1

1. Paths of length 1
- Reachable vertices: 2, 4

2. Paths of length 2
- Reachable vertices: 3, 5, 7
  - Add 2 tokens to: 2 (5, 7)
  - Add 1 token to: 4 (3)

3. Paths of length 3
- Reachable vertex: 6 (two paths)
  - Add .5 token to: 2, 5
  - Add .5 token to: 4, 3

Array Notation

- Data types
  - Reals: $\mathbb{R}$ Integers: $\mathbb{N}$ Booleans: $\mathbb{B}$
  - Postitive Integers: $\mathbb{N}_+$

- Vectors (bold lowercase): $\mathbf{a}$: $\mathbb{R}^{N}$
- Matrices (bold uppercase): $\mathbf{A}$: $\mathbb{R}^{NxN}$
- Tensors (script bold uppercase): $\boldsymbol{\mathcal{A}}$: $\mathbb{R}^{NxNxN}$

- Standard matrix multiplication

$$\mathbf{A}\,\mathbf{B} = \mathbf{A} \;+.*\; \mathbf{B}$$

- Sparse matrix: $\mathbf{A}$: $\mathbb{R}^{S(N)xN}$
- Parallel matrix: $\mathbf{A}$: $\mathbb{R}^{P(N)xN}$

Matrix Algorithm

$$\mathbf{c} : \mathbb{R}_+^N = \textsc{BetweennessCentrality}(\mathbf{A} : \mathbb{B}^{S(N \times N)})$$

Declare Data Structures

$$\mathfrak{T} : \mathbb{B}^{S(d_{max}) \times |\mathbf{v}| \times S(N)} \qquad \mathbf{Q} : \mathbb{N}_+^{|\mathbf{v}| \times N} \qquad \tilde{\mathbf{Q}} : \mathbb{N}_+^{|\mathbf{v}| \times S(N)}$$

$$\mathbf{W} : \mathbb{R}_+^{|\mathbf{v}| \times S(N)} \qquad \tilde{\mathbf{C}} : \mathbb{R}_+^{|\mathbf{v}| \times N} \qquad \mathbf{c}, : \mathbb{R}_+^N$$

Loop over vertices

for $\mathbf{v} \in V$

do

$d := 1 \qquad \mathfrak{T} := 0 \qquad \tilde{\mathbf{C}} := 0$

$\mathbf{T}_d := \mathbf{Q} := \tilde{\mathbf{Q}} := \mathbf{I}(\mathbf{v}, :)$

Shortest paths

while $\mathbf{T}_d \neq 0$

do

$\tilde{\mathbf{Q}} := (\tilde{\mathbf{Q}}\ \mathbf{A})\ .*\ \neg\mathbf{Q}$

$\mathbf{T}_{d+1} := \tilde{\mathbf{Q}} \qquad \mathbf{Q} \mathrel{+}= \tilde{\mathbf{Q}} \qquad d{+}{+}$

Rollback & Tally

for $\tilde{d} := d$ to 3

do

$\mathbf{W} := \mathbf{T}_{\tilde{d}}\ .*\ (1 + \tilde{\mathbf{C}})\ ./\ \mathbf{Q}$

$\tilde{\mathbf{C}} \mathrel{+}= (\mathbf{A}\ \mathbf{W}^T)^T\ .*\ \mathbf{T}_{\tilde{d}-1}\ .*\ \mathbf{Q}$

$\mathbf{c} \mathrel{+}= \sum_v \tilde{\mathbf{c}}_v$

Sparse Matrix-Matrix Multiply

Parallel Algorithm

Change matrices to parallel arrays

$$\mathbf{c} : \mathbb{R}_+^N = \textsc{BetweennessCentrality}(\mathbf{A} : \mathbb{B}^{P_c(S(N \times N))})$$

$$\mathfrak{T} : \mathbb{B}^{S(d_{max}) \times |\mathbf{v}| \times P(S(N))} \qquad \mathbf{Q} : \mathbb{N}_+^{|\mathbf{v}| \times P(N)} \qquad \tilde{\mathbf{Q}} : \mathbb{N}_+^{|\mathbf{v}| \times P(S(N))}$$

$$\mathbf{W} : \mathbb{R}_+^{|\mathbf{v}| \times P(S(N))} \qquad \tilde{\mathbf{C}} : \mathbb{R}_+^{|\mathbf{v}| \times P(N)} \qquad \mathbf{c}, : \mathbb{R}_+^N$$

$$\textbf{for } \mathbf{v} \in V$$

$$\quad \textbf{do}$$

$$\qquad d := 1 \qquad \mathfrak{T} := 0 \qquad \tilde{\mathbf{C}} := 0$$

$$\qquad \mathbf{T}_d := \mathbf{Q} := \tilde{\mathbf{Q}} := \mathbf{I}(\mathbf{v}, :)$$

$$\qquad \textbf{while } \mathbf{T}_d \neq 0$$

$$\qquad\quad \textbf{do}$$

$$\qquad\qquad \tilde{\mathbf{Q}} := (\tilde{\mathbf{Q}} \ \mathbf{A}) \ .* \ \neg\mathbf{Q}$$

$$\qquad\qquad \mathbf{T}_{d+1} := \tilde{\mathbf{Q}} \qquad \mathbf{Q} \mathrel{+}= \tilde{\mathbf{Q}} \qquad d{+}{+}$$

$$\qquad \textbf{for } \tilde{d} := d \textbf{ to } 3$$

$$\qquad\quad \textbf{do}$$

$$\qquad\qquad \mathbf{W} := \mathbf{T}_{\tilde{d}} \ .* \ (1 + \tilde{\mathbf{C}}) \ ./ \ \mathbf{Q}$$

$$\qquad\qquad \tilde{\mathbf{C}} \mathrel{+}= (\mathbf{A} \ \mathbf{W}^T)^T \ .* \ \mathbf{T}_{\tilde{d}-1} \ .* \ \mathbf{Q}$$

$$\qquad \mathbf{c} \mathrel{+}= \textstyle\sum_v \tilde{\mathbf{c}}_v$$

Parallel Sparse Matrix-Matrix Multiply

Complexity Analysis

- Do all vertices at once (i.e. |v|=N)
  - N = # vertices, M = # edges, k = M/N
- Algorithm has two loops each containing d_{max} sparse matrix multiplies. As the loop progresses the work done is

  d=1 (2kM)
  d=2 $(2k^2M)$ - (2kM)
  d=3 $(2k^3M - 2k^2M)$ - $(2k^2M - 2kM)$
  …
- Summing these terms for both loops and approximating the graph diameter by $d_{max} \approx \log_k(N)$ results in a complexity

$$4\ k^{dmax}\ M \approx 4\ N\ M$$

- Time to execute is

$$T_{BC} \approx 4\ N\ M / (e\ S)$$

  where S = processor speed, e = sparse matrix multiply efficiency
- Official betweenness centrality performance metric is Traversed Edges Per Second (TEPS)

$$\text{TEPS} \equiv NM/T_{BC} \approx (e\ S) / 4$$

- **Betweenness Centrality tracks Sparse Matrix multiply performance**

Outline

- Introduction
- Power Law Graphs
- Graph Benchmark
- **Results**
  - *Post Detection Processing*
  - *Sparse Matrix Duality*
  - *Approach*
- Summary

Matlab Implementation

- **Array code is very compact**
- **Lingua franca of DoD engineering community**
- **Sparse matrix matrix multiply is key operation**

```
function BC = BetweennessCentrality(G,K4approx,sizeParts)
  declareGlobals;
  A = logical(mod(G.adjMatrix,8) > 0);
  N = length(A);  BC = zeros(1,N);  nPasses = 2^K4approx;
  numParts = ceil(nPasses/sizeParts);
  for(p = 1:numParts)
    BFS = [];       depth = 0;
    nodesPart = ((p-1).*sizeParts + 1):min(p.*sizeParts,N);
    sizePart = length(nodesPart);
    numPaths = accumarray([(1:sizePart)',nodesPart']…
       ,1,[sizePart,N]);
    fringe = double(A(nodesPart,:));
    while nnz(fringe) > 0
        depth = depth + 1;
        numPaths = numPaths + fringe;
        BFS(depth).G = logical(fringe);
        fringe = (fringe * A) .* not(numPaths);
    end
    [rows cols vals] = find(numPaths);
    nspInv = accumarray([rows,cols],1./vals,[sizePart,N]);
    bcUpdate = ones(sizePart,N);
    for depth = depth:-1:2
        weights = (BFS(depth).G .* nspInv) .* bcUpdate;
        bcUpdate = bcUpdate + ...
          ((A * weights')' .* BFS(depth-1).G) .* numPaths;
    end
    bc = bc + sum(bcUpdate,1);
  end
  bc = bc - nPasses;
```

Matlab Profiler Results

Profiler

File Edit Debug Window Help

Start Profiling | Run this code: RUN_graphAnalysis | Profile time: 45 sec

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
159	`fringe = (fringe * adjacencyMa...`	20	11.443 s	37.0%	
179	`bcUpdate = bcUpdate + ...`	18	9.878 s	31.9%	
178	`weights = (BFS(depth).G .* nsp...`	18	6.717 s	21.7%	
157	`numShortestPaths = numShortest...`	20	1.603 s	5.2%	
168	`nspInv = accumarray([rows,cols...`	2	0.341 s	1.1%	
All other lines			0.979 s	3.2%	
Totals			30.962 s	100%	

- **Betweenness Centrality performance is dominated by sparse matrix matrix multiply performance**

Code Comparison

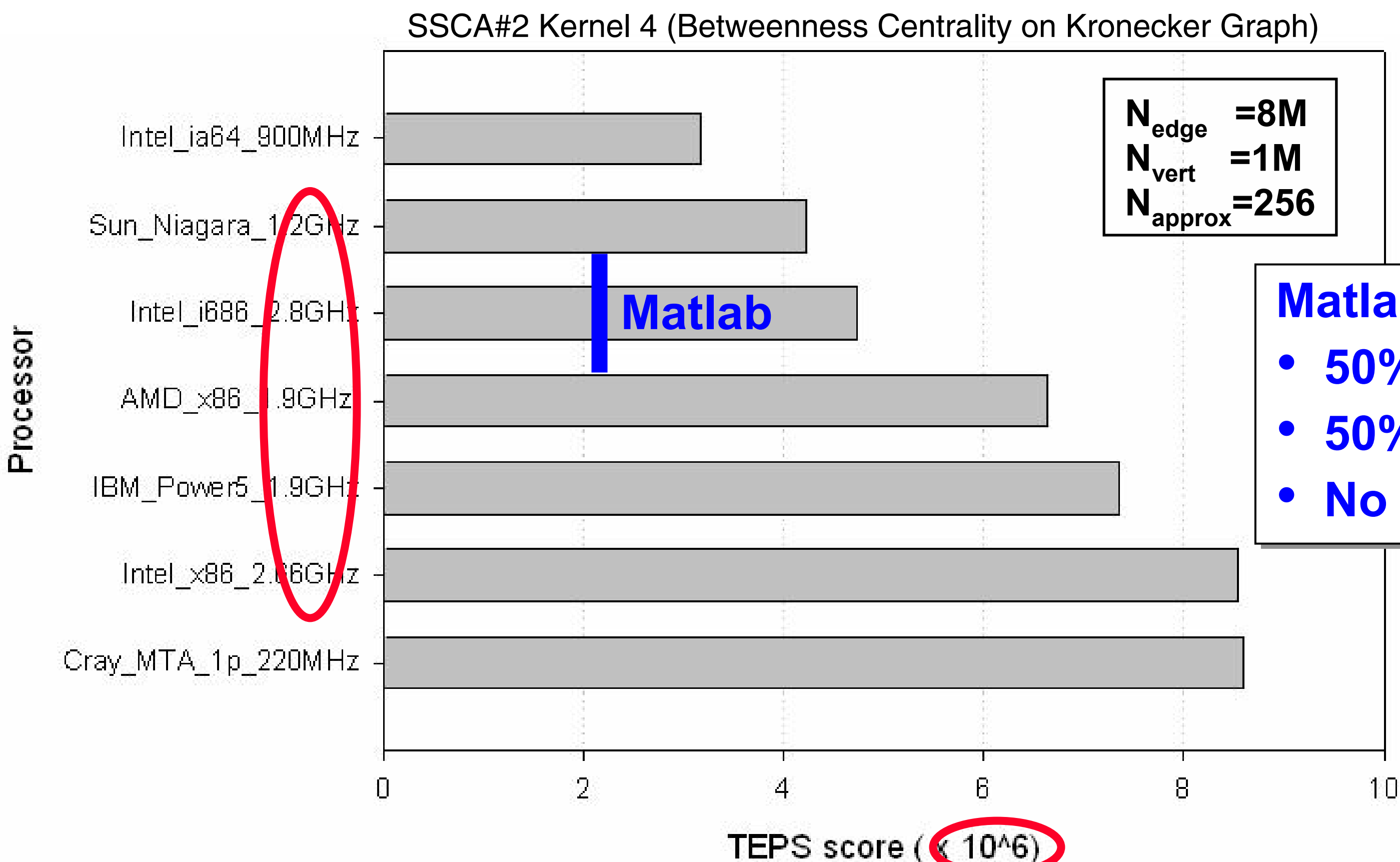
- Software Lines of Code (SLOC) are a standard metric for comparing different implementations

Language	SLOCs	Ratio to C
C	86	1.0
C + OpenMP (parallel)	336	3.9
Matlab	28	1/3.0
pMatlab (parallel)	50 (est)	1/1.7 (est)
pMatlabXVM (parallel out-of-core)	75 (est)	1 (est)

- Matlab code is small than C code be the expected amount
- Parallel Matlab and parallel out-of-core are expected to be smaller than serial C code

Betweenness Centrality Performance -Single Processor-

Data Courtesy of Prof. David Bader & Kamesh Madduri (Georgia Tech)

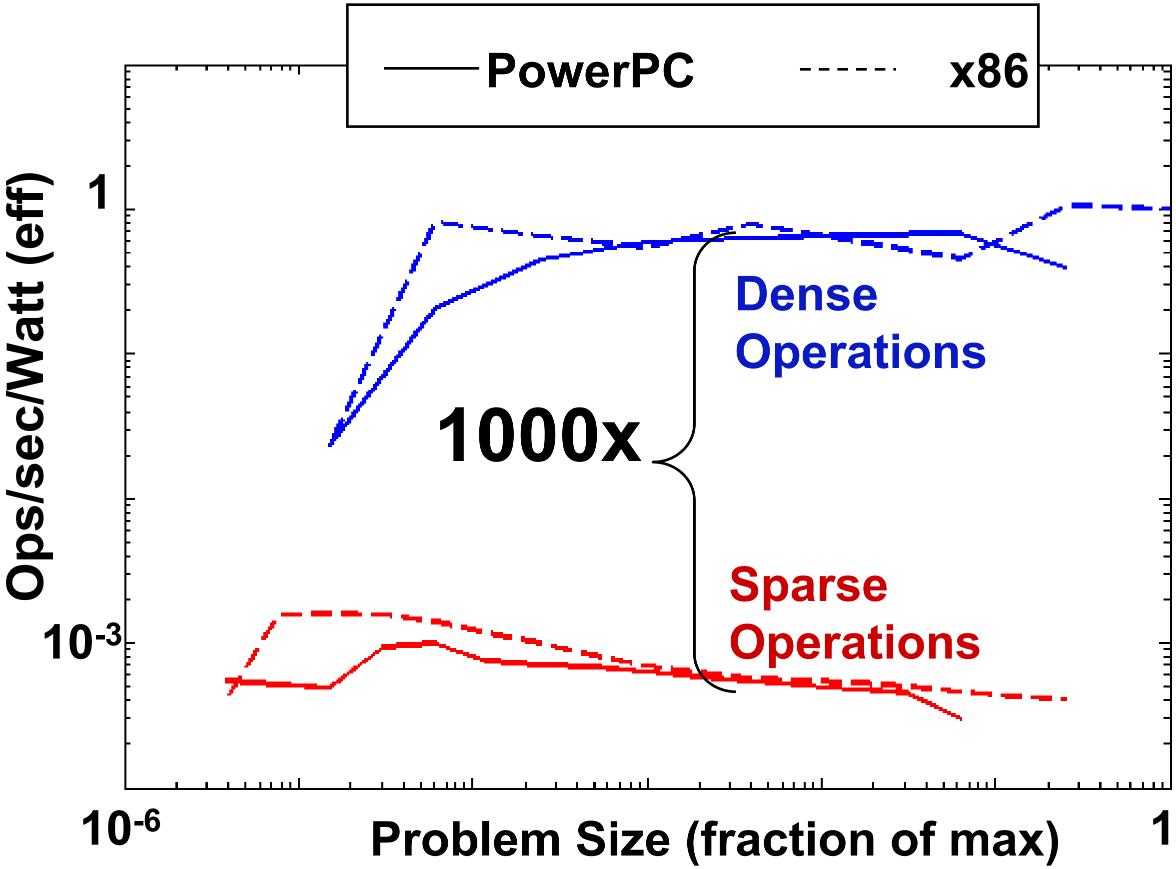


(Traversed Edges Per Second)

Matlab achieves
- **50% of C**
- **50% of sparse matmul**
- **No hidden gotchas**

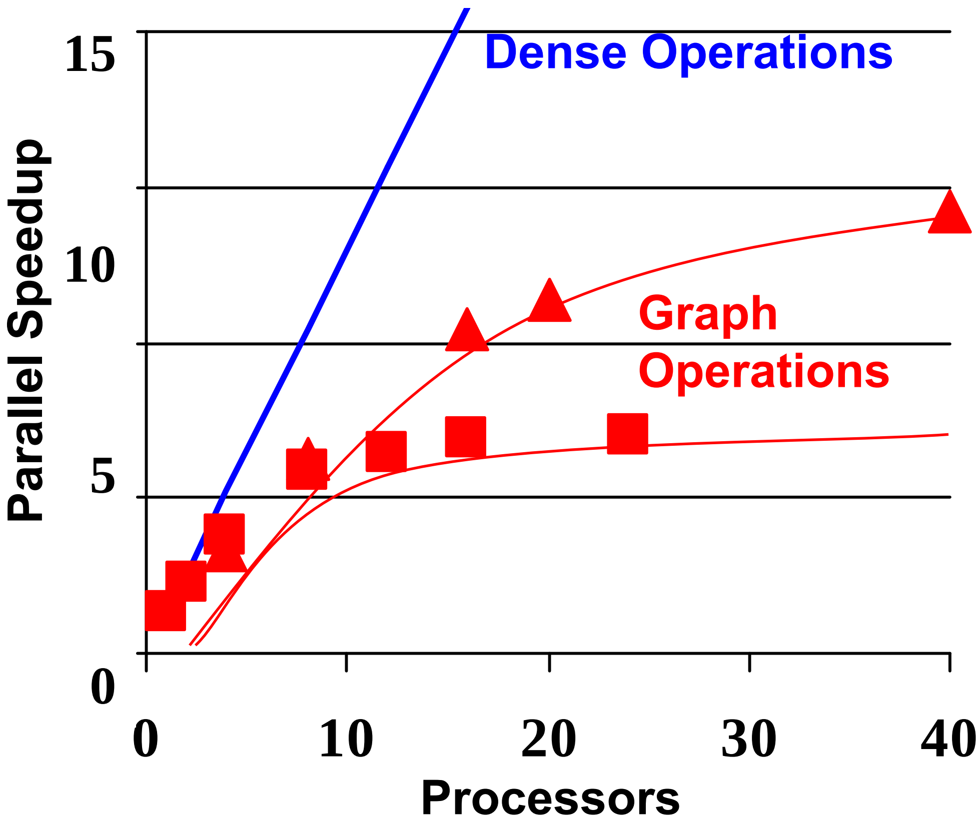
- **Canonical graph based implementations**
- **Performance limited by low processor efficiency (e ~ 0.001)**
  - **Cray Multi Threaded Architecture (1997) provides a modest improvement**

COTS Serial Efficiency



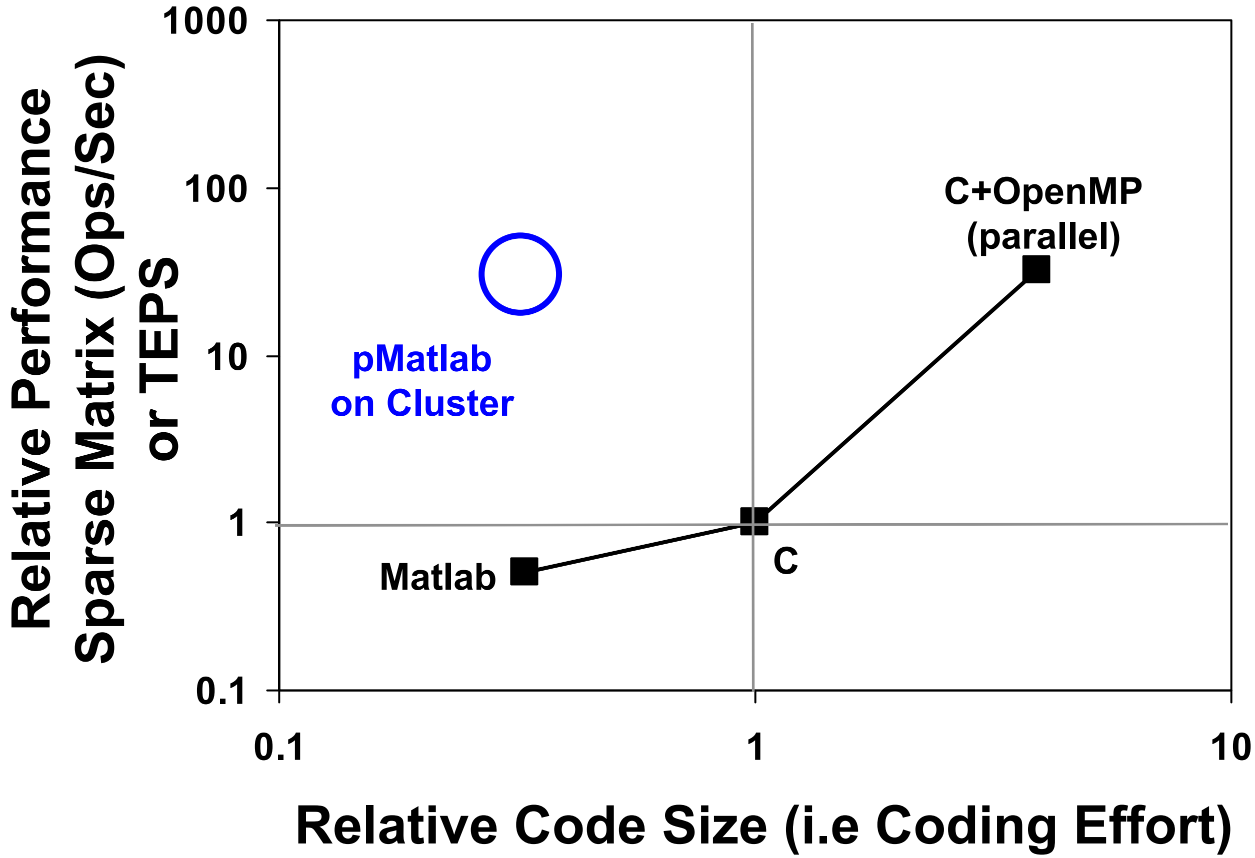
- COTS processors are 1000x more efficient on sparse operations than dense operations

Parallel Results (canonical approach)



- Graph algorithms scale poorly because of high communication requirements
- Existing hardware has insufficient bandwidth

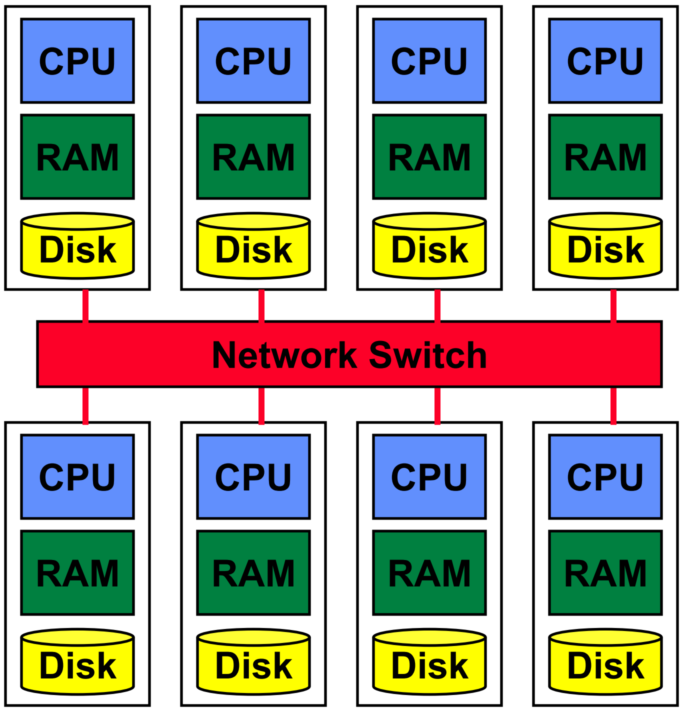
Performance vs Effort



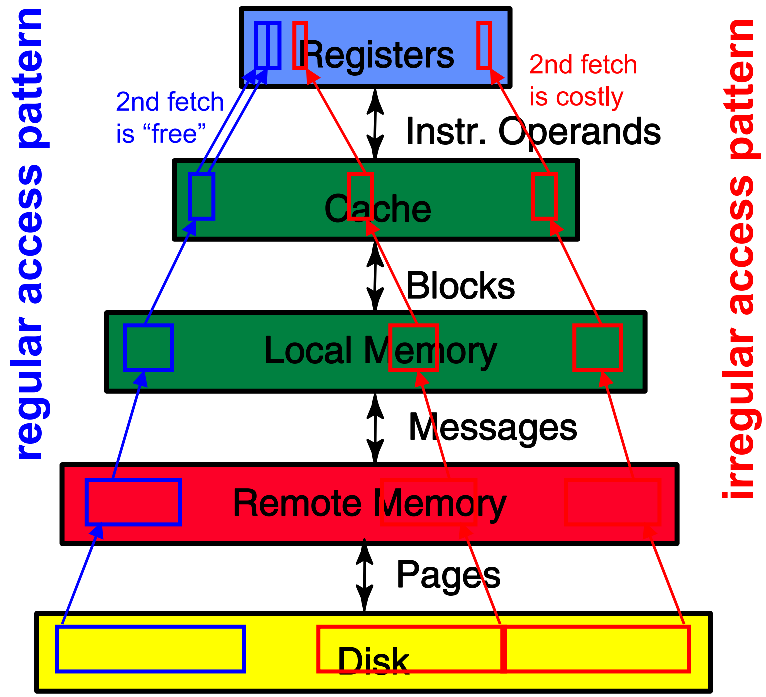
- **Array (matlab) implementation is short and efficient**
  - **1/3 the code of C implementation (currently 1/2 the performance)**
- **Parallel sparse array implementation should match parallel C performance at significantly less effort**

Why COTS Doesn't Work?

Standard COTS Computer Architecture



Corresponding Memory Hierarchy



- Standard COTS architecture requires algorithms to have regular data access patterns
- Graph algorithms are irregular, caches don't work and even make the problem worse (moving lots of unneeded data)

Summary
Embedded Processing Paradox

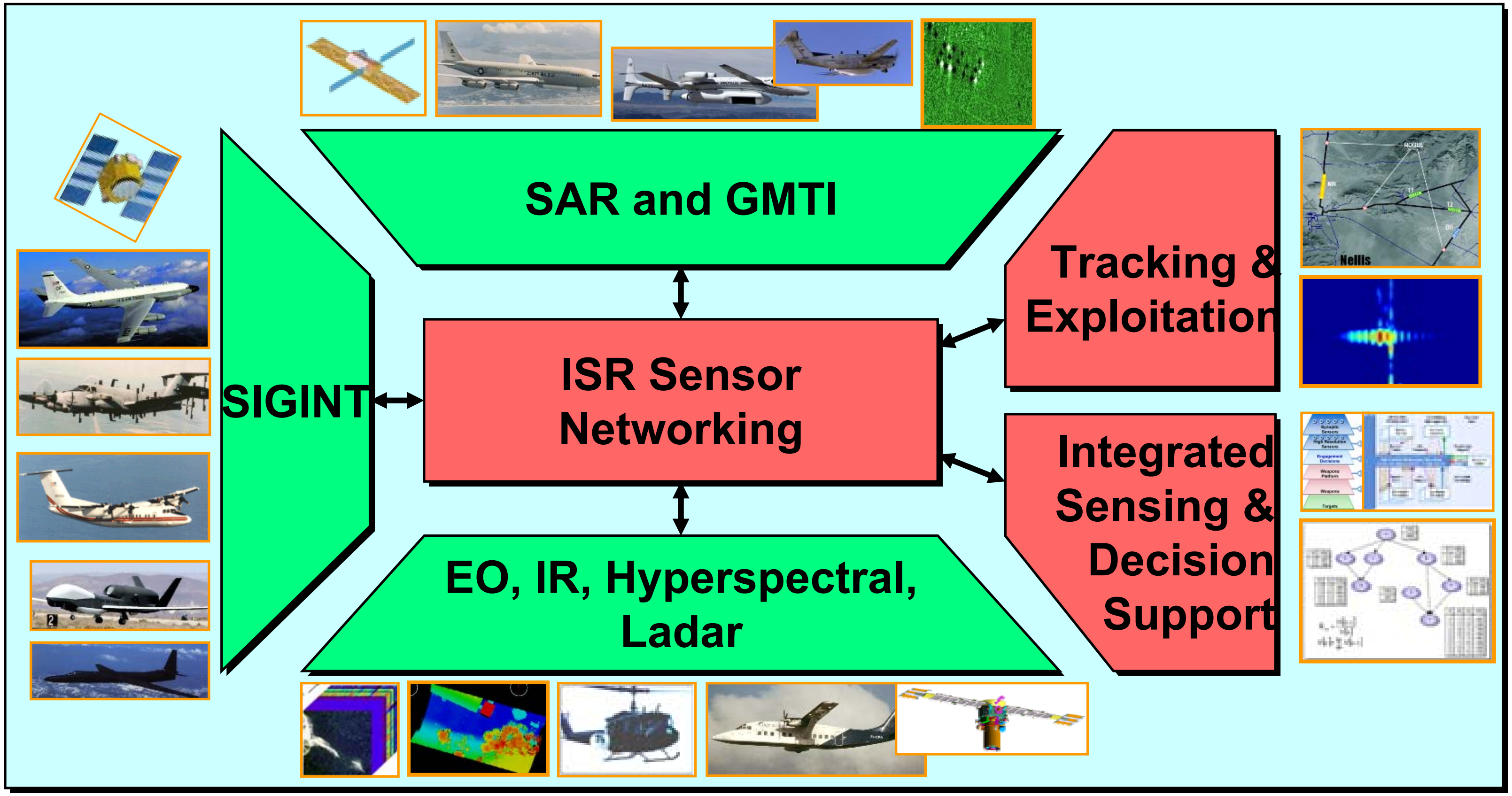
- Front end data rates are much higher
- However, back end correlation times are longer, algorithms are more complex and processor efficiencies are low
- If current processors scaled (which they don't), required power for back end makes even basic graph algorithms infeasible for embedded applications

	Front End	Back End
Data input rate	Gigasamples/sec	Megatracks/day
Correlation time	seconds	months
Algorithm complexity	O(N log(N))	O(N M)
Processor Efficiency	50%	0.05%
Desired latency	seconds	minutes
Total Power	~1 KWatt	>100 KWatt

Need fundamentally new technology approach for graph-based processing

Backup Slides

Motivation: Graph Processing for ISR



Algorithms	Signal Processing	Graph
Data	Dense Arrays	Graphs
Kernels	FFT, FIR, SVD, …	BFS, DFS, SSSP, …
Parallelism	Data, Task, …	Hidden
Compute Efficiency	10% - 100%	< 0.1%

- **Post detection processing relies on graph algorithms**
  - **Inefficient on COTS hardware**
  - **Difficult to code in parallel**

FFT = Fast Fourier Transform, FIR = Finite Impulse Response, SVD = Singular Value Decomposition
BFS = Breadth First Search, DFS = Depth First Search, SSSP = Single Source Shortest Paths