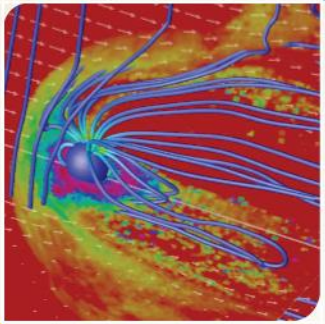
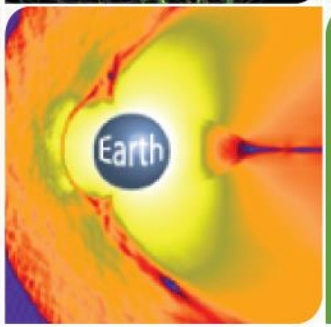
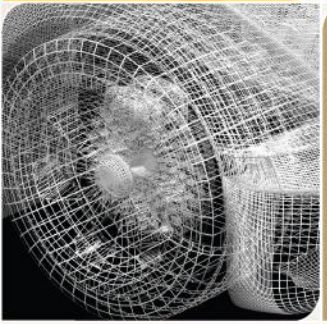




An Energy-Efficient Abstraction for Simultaneous Breadth-First Searches

Adam McLaughlin, Jason Riedy, and David A. Bader

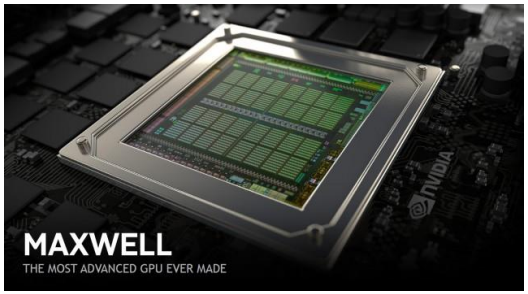
Georgia Tech College of Computing
Computational Science and Engineering

Problem

- Data is unstructured, heterogeneous, and vast
- Serious opportunities for businesses to capitalize on insight
  - Target (pregnancy scandal)
  - Explosion of popularity in ML techniques
- Interactive, real-time analyses preferred
  - Graph analytics
  - HPC to the rescue!

“Traditional” HPC is Expensive

- Tianhe-2: 17.8 MW
- Titan: 8.2 MW
- Distributed systems are overkill
  - Too much time and energy wasted on expensive communication
  - Shared memory is large enough (~1 TB)
- Leverage the high memory bandwidth of GPUs



GPUs are Challenging to Program

- Months of domain expert programmer time required to develop/optimize code
- Efforts are typically limited to a single problem, architecture, or data set
  - Little code reuse
  - Limited number of libraries
  - Opaque performance consequences

What makes GPU Computing so Difficult?

- Parallel programming challenges
  - Deadlock, synchronization, race conditions
- Architectural/Ecosystem challenges
  - Programmer managed shared memory
  - Deep knowledge of the underlying architecture required
- Challenges unique to graph analysis
  - Data dependent memory access patterns

Solution: Abstraction

- Abstract details of parallel programming from end users
- Let social scientists, analysts, etc. focus on gathering insights
- Let domain experts focus on parallel programming, architectural details
  - Encourage modularity and code reuse

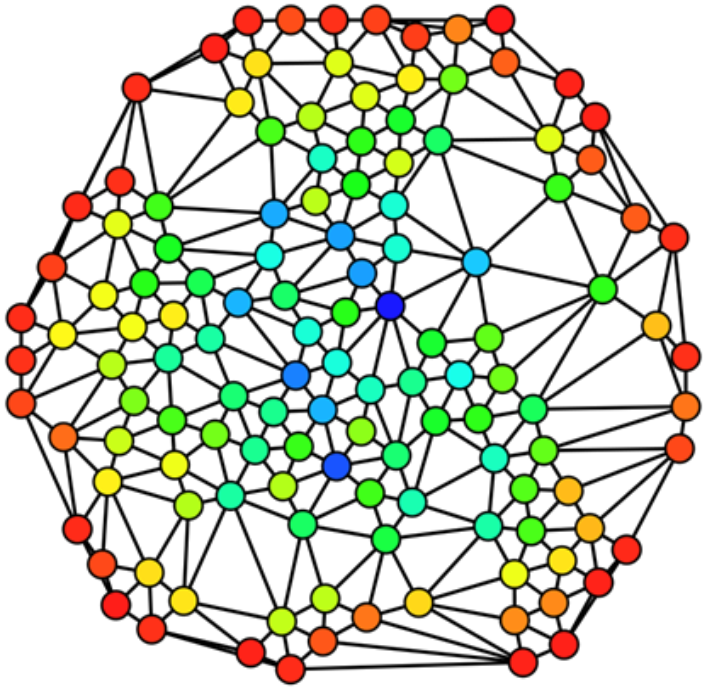
Related Abstractions

- Traversal-based
  - Ligra [Shun *et al.*], Gunrock [Wang *et al.*]
  - User applies code that operates on active vertices and provides the next frontier of vertices
  - Library efficiently transitions from one frontier to the next
- Linear-Algebraic
  - GraphBLAS [Mattson *et al.*], CUSP [Dalton *et al.*]
  - Formulate as operators on vectors and matrices
- Gather Apply Scatter (GraphLab)

The Multi-Search Abstraction

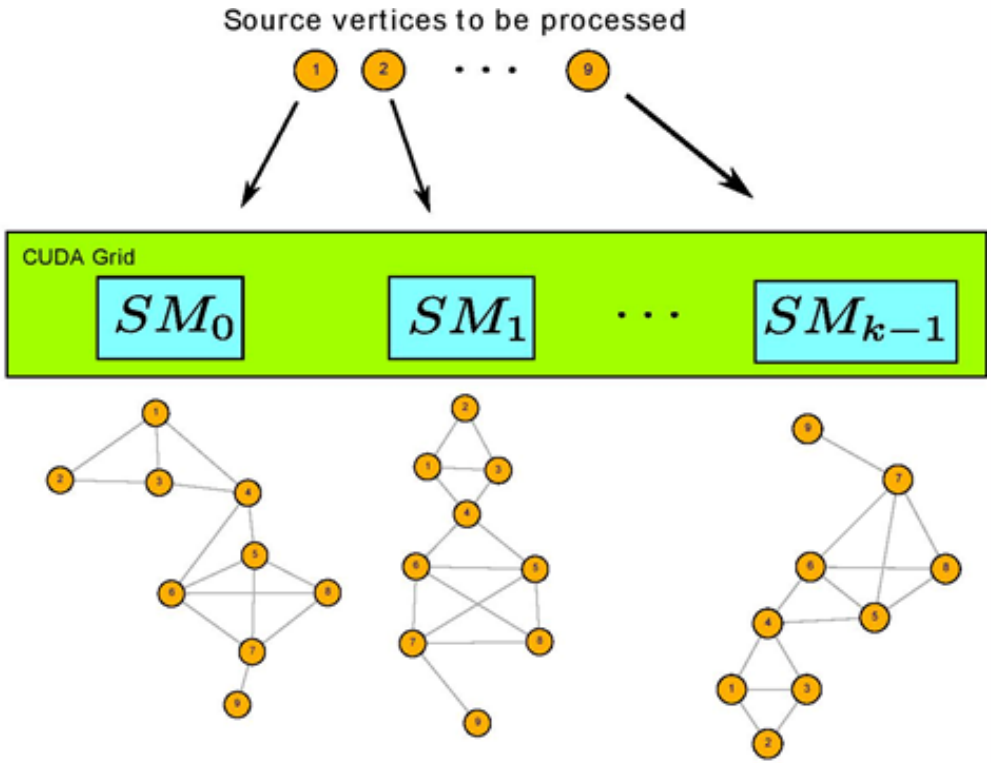
- Fits any problem requiring the simultaneous execution of many breath-first searches

1. All-Pairs Shortest Paths
2. Diameter Computations
3. Transitive Closures
4. Betweenness Centrality



What makes this abstraction different?

- Traversal based, but utilizes *coarse-grained parallelism*
  - Prior abstractions parallelize within the context of a single BFS
  - Our abstraction parallelizes *across* BFSs



Multi-Search: User Functions

- User needs to implement a small number of typically short functions:

- `init()`: Runs prior to every search
- `prior()`: Runs before every search iteration
- `visitVertex()`: When an edge `(u,v)` is traversed from source `s`, update data in terms of `u,v,s`
- `post()`: Runs after every search iteration
- `finalize()`: Runs after the completion of every search

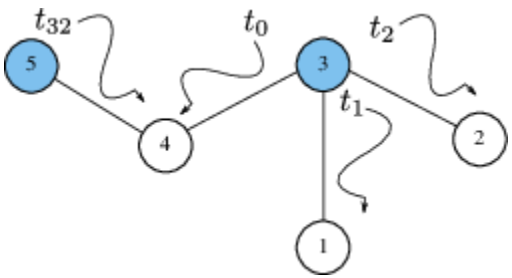
Multi-Search: Simple Example

- All-Pairs Shortest Paths

```
void init(int s)
{
        for(int k=0; k<n; k++) //For each vertex
        {
                if(k == s) d[s][k] = 0;
                else d[s][k] = INT_MAX;
        }
}
void visitVertex(int s, int u, int v, int *Q_next, int Q_next_len)
{
        if(d[s][v] == INT_MAX)
        {
                d[s][v] = d[s][u] + 1;
                t = atomicAdd(&Q_next_len,1);
                Q_next[t] = w
        }
}
```

Multi-Search: Visiting vertices

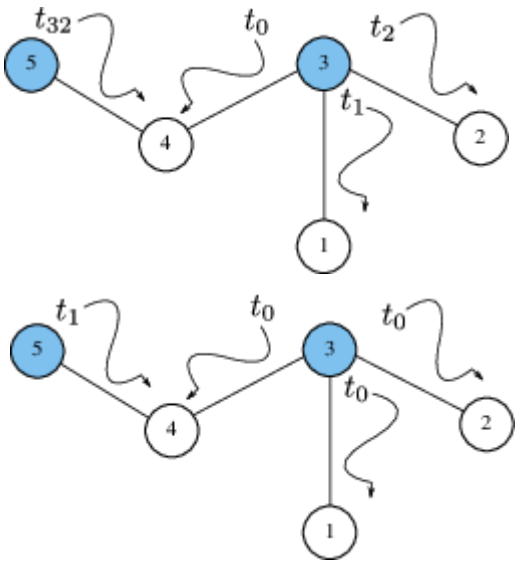
- Use a cooperative, Warp-*based* approach
- Warps concurrently expand adjacency lists of enqueued vertices
- Works great for vertices with high outdegree
  - Coalesced accesses to neighbor lists
- Underutilization for vertices with low outdegree



Multi-Search: Hierarchical Queues

- To resolve this underutilization, we can assign a thread to each enqueued vertex

- Use a thresholding approach
  - Outdegree(v) >= T → Warp processing
  - Outdegree(v) < T → Thread processing



Multi-Search: Skeleton

```
void multi_search(int u, int v, int s){
  parallel_for(int s=0; s<n; s++){ //Parallelize across SMs
    if(outdegree(i) > T) Q_small_curr.enqueue(s);
    else Q_curr.enqueue(s);
    init(s);
    barrier();
    while(!Q_curr.empty() && !Q_small_curr.empty()){
      prior();
      barrier();
      //Visit vertices in the queues...
      move(Q_curr,Q_next); //Swap queues for next iteration
      move(Q_small_curr,Q_small_next);
      barrier();
      post();
      barrier();
    }
    finalize();
  }
}
```

Multi-Search: Under the hood

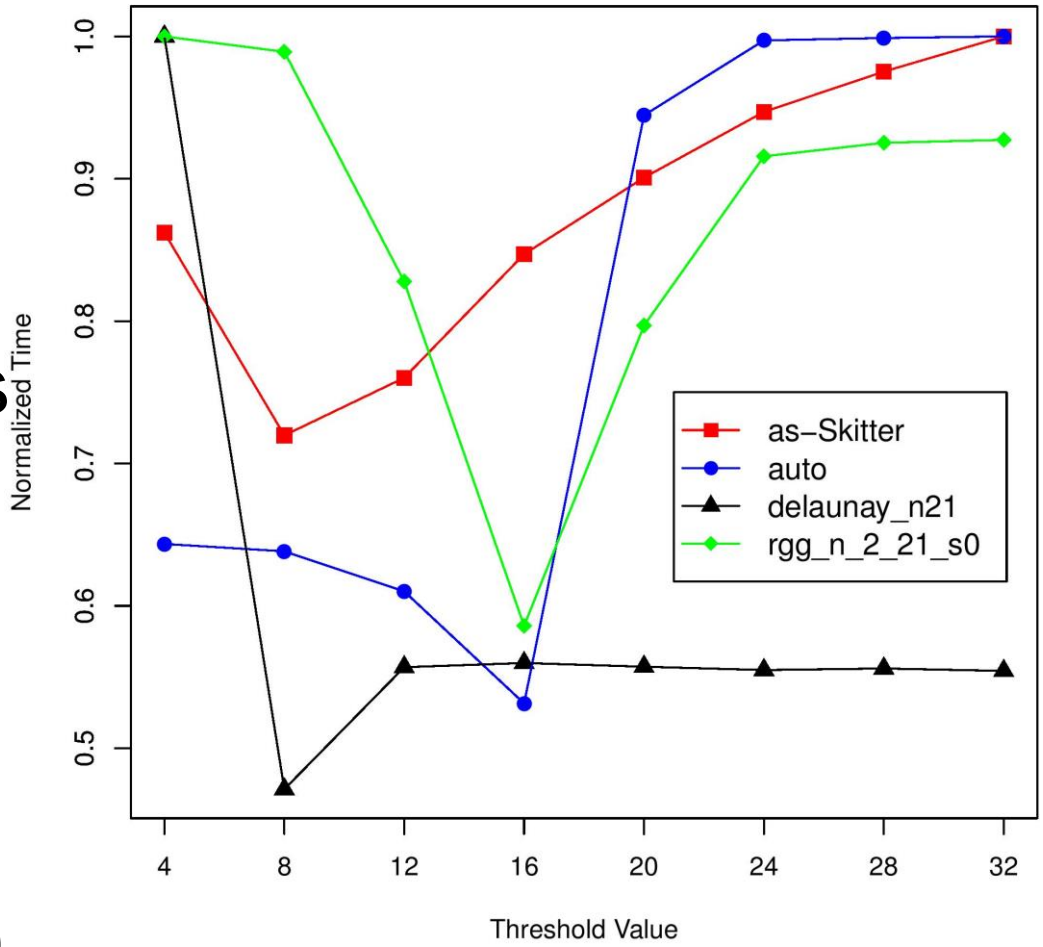
```
while(!Q_curr.empty() && !Q_small_curr.empty()){
  prior();
  barrier();
  // Visit vertices in the queues
  for(int v=0; v<Q_curr_len; v++){
    parallel_for(int w=0; w<outdegree(v); w++){ //threads
      visitVertex(s,v,w,Q_next,Q_next_len);
    }
  }
  parallel_for(int v=0; v<Q_small_curr_len; v++){ //threads
    for(int w=0; w<outdegree(v); w++){
      visitVertex(s,v,w,Q_small_next,Q_small_next_len);
    }
  }
  move(Q_curr,Q_next); //Swap queues for next iteration
  move(Q_small_curr,Q_small_next);
  barrier();
  post();
  barrier();
}
```

Experimental Setup

- NVIDIA GTX Titan, NVIDIA Tesla K40c
  - Compute capability 3.5 ("Kepler") GPUs
  - Peak theoretical memory bandwidth: 288.4 GB/s
- Titan: 14 SMs, 6GB memory, 837MHz
- Tesla K40c: 15 SMs, 12GB memory, 745MHz, 245W TDP
- Power measurement: NVML & C++11 Futures
  - Sample every 10ms

Effect of Thresholding

- T = 0: Warp
- T = ∞: Thread
- Too small: Warp occupancy suffers
- Too large: severe workload imbalances among threads
- T = 16 (Half-warp)



Benchmark Data Sets

Graph	Vertices	Edges	Graph	Vertices	Edges
333SP	3,712,815	22,217,266	*hollywood-2009*	1,139,905	115,031,232
adaptive	6,815,744	27,248,640	*kron_g500-logn19*	524,288	43,561,574
as-Skitter	1,696,415	22,190,596	*ldoor*	952,203	45,570,272
auto	448,695	6,629,222	*rgg_n_2_21_s0*	2,097,152	28,975,990
delaunay_n21	2,097,152	12,582,816	*roadNet-CA*	1,971,281	5,533,214
ecology1	1,000,000	3,996,000	*thermal2*	1,227,087	7,352,268

Timing Results (s)

Graph	Oracle	Warp-based	Hybrid	Graph	Oracle	Warp-based	Hybrid
333SP	47.8	68.0	**32.1**	*hollywood-2009*	81.6	21.3	**20.4**
adaptive	54.8	183.6	**42.8**	*kron_g500-logn19*	35.4	**16.4**	17.1
as-Skitter	27.8	12.9	**9.7**	*ldoor*	**35.4**	35.7	36.5
auto	15.6	5.48	**4.8**	*rgg_n_2_21_s0*	67.0	37.0	**23.7**
delaunay_n21	23.3	25.1	**15.0**	*roadNet-CA*	12.3	15.0	**9.2**
ecology1	8.9	29.1	**6.5**	*thermal2*	11.7	19.4	**7.7**

- Savings of 42% time on average vs. Oracle

Energy Results (J)

Graph	Oracle	Warp-based	Hybrid	Graph	Oracle	Warp-based	Hybrid
333SP	2,220	2,880	**1,083**	*hollywood-2009*	11,546	**1,355**	3,342
adaptive	8,551	6,790	**1,433**	*kron_g500-logn19*	1,180	1,195	**570**
as-Skitter	4,688	**1,053**	1,606	*ldoor*	1,829	1,855	**1,210**
auto	635	539	**159**	*rgg_n_2_21_s0*	2,851	1,908	**759**
delaunay_n21	1,425	1,495	**508**	*roadNet-CA*	2,050	**1,144**	1,539
ecology1	856	1,642	**219**	*thermal2*	996	1,311	**259**

- Savings of 62% energy on average vs. Oracle

Conclusions

- Abstraction is paramount for high-performance, reusable applications
  - Prior methods of abstraction miss out on coarse-grained parallelism
- If the amount of parallelism changes over time, the method of parallelism should change too
- Maximizing performance also tends to help save energy
  - Saved 42% time and 62% energy compared to prior work

Acknowledgment of Support

Questions

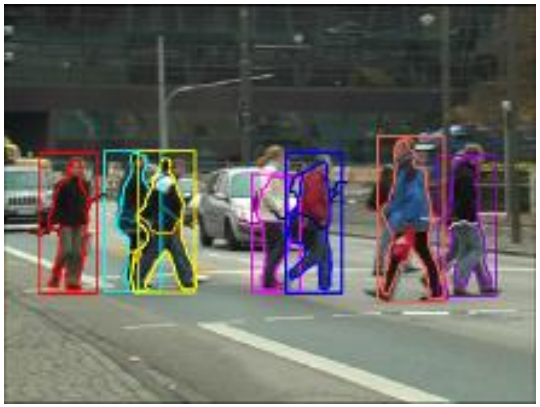
"To raise new questions, new possibilities, to regard old problems from a new angle, requires creative imagination and marks real advance in science."– Albert Einstein

https://github.com/Adam27X/graph-utils

Backup

“Traditional” HPC is Expensive

- Graphs are no longer processed by supercomputers alone
  - Embedded systems
    - Computer vision
  - Mobile devices
    - Spam detection



- Systems are becoming constrained by power and energy
  - High demand for work-efficient implementations
  - Goal: Maximize performance per Watt using GPUs

Motivation

- Real world graphs are challenging to process
  - Enormous
    - Networks cannot be manually inspected
  - Varying structural properties
    - Small-world, scale-free, meshes, road networks
      - Not a one-size fits all problem
  - Unpredictable
    - Rapidly change over time
    - Data dependent memory access patterns