Graph Algorithms on Future Architectures

Scott McMillan, PhD

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0002857

Graph Algorithms on Future Architectures

Fast, efficient graph analysis is **important** and **pervasive**.

Heterogeneous hardware is ~~coming~~ **here**.

We have built a library that helps developers use both.

Research question: Can a set of **primitives and operations** be defined that will **separate the concerns** between graph analytic application development and the increasing complexity of the underlying hardware?

Release library as open source.

Schedule/Items/Contents

Motivation

- Graph Algorithms
- Heterogeneous High Performance Computing (HHPC)

The Separation of Concerns

- Library Architecture
- GraphBLAS API

Example and Results

Future Work

Schedule/Items/Contents

Motivation

- **Graph Algorithms**
- **Heterogeneous High Performance Computing (HHPC)**

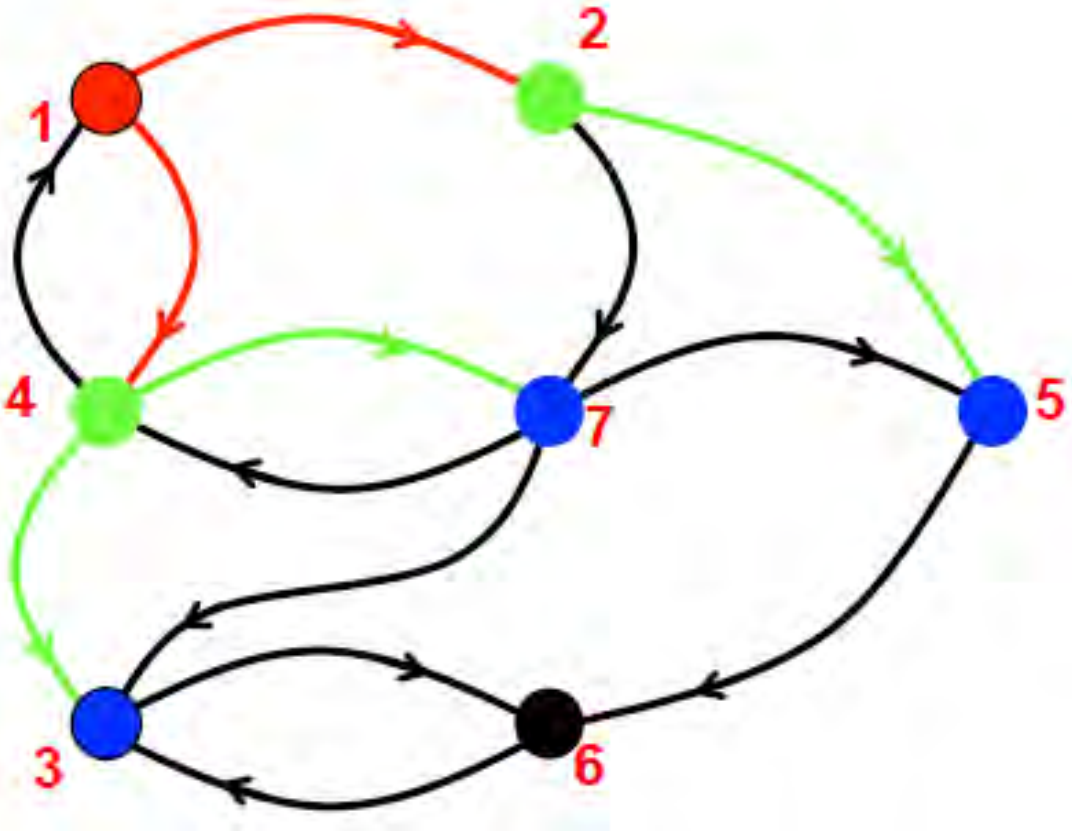
The Separation of Concerns

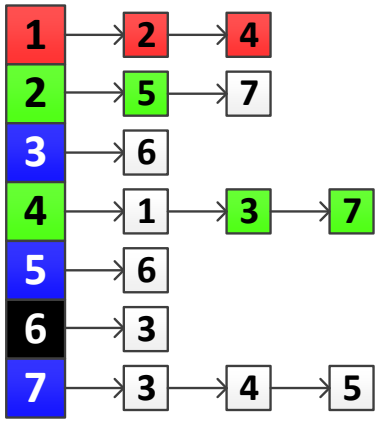
- **Library Architecture**
- **GraphBLAS API**

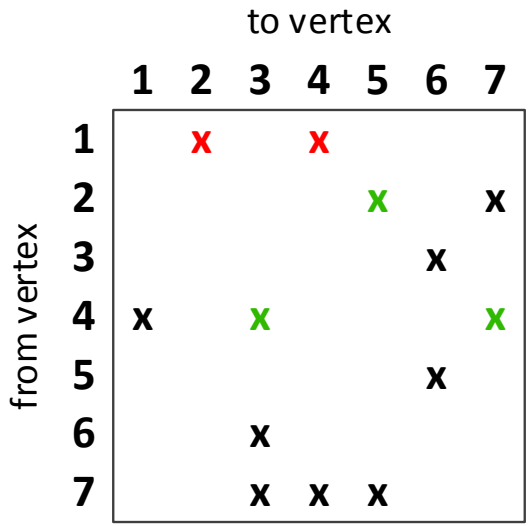
Example and Results

Future Work

Graph Analysis is Important and Pervasive. Reminder: Graphs

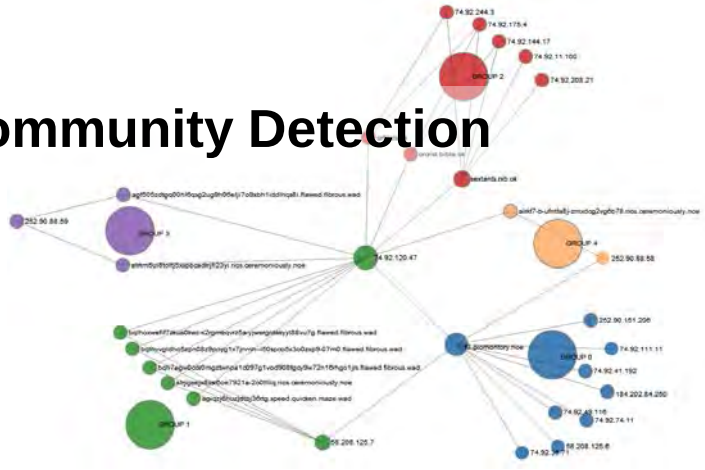






Graph Analysis is *Important* and *Pervasive*.

Community Detection

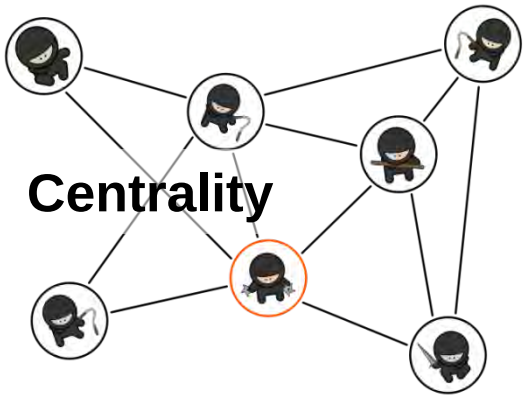


APT Detection in Computer Networks, C3E, 2013

Shortest Path
Cost Minimization
Max Flow

United States Interstate Highway System

Centrality

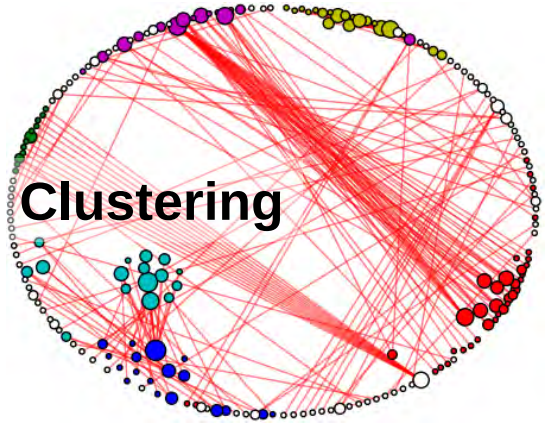


Malware Distribution Networks

Connected Components
PageRank

Social Networks

Clustering



Revert graph showing editor conflict on the "Cyprus dispute" Wikipedia page, 2015

Graph Analysis is *Important* and *Pervasive* (and *Difficult*).

Graph data typically lacks "locality" and cannot be easily partitioned into isolated sub-graphs or sub-problems.

This makes it difficult to distribute computations on graphs over multiple or many processors.

Two major implications

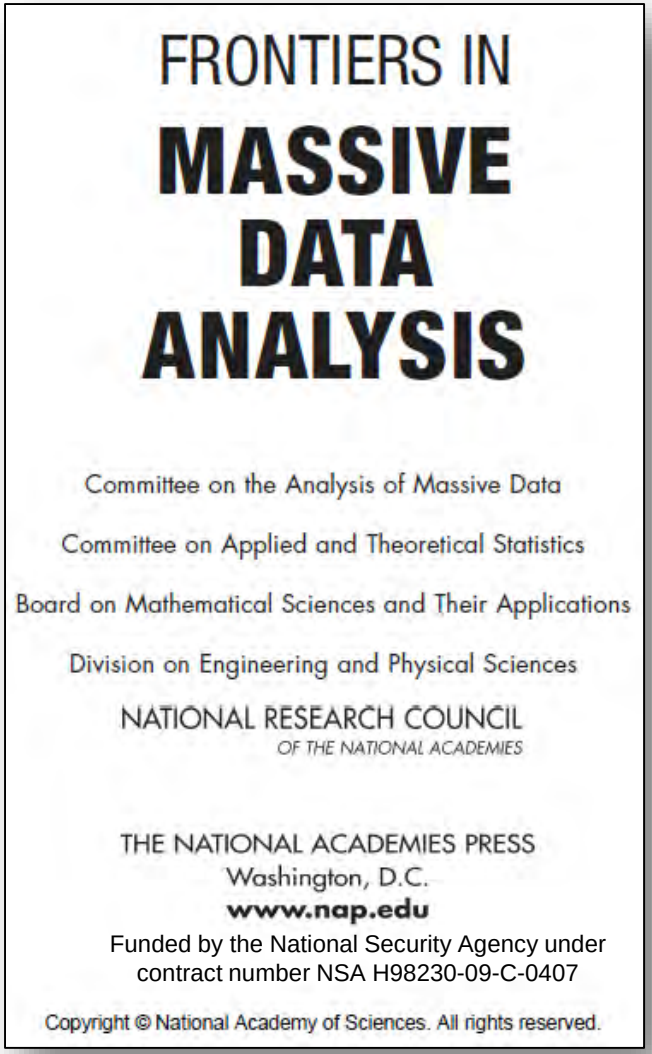
- Small *computation to communication ratio*
- Unpredictability of data access

The Challenge of Primitives: Develop a Middleware for Large-Scale Graph Analytics

From the computer systems perspective, it would be very helpful to identify a set of primitive algorithmic tools that

1) provide a framework to express concisely a broad scope of computations;
2) allow programming at the appropriate level of abstraction; and
3) are applicable over a wide range of platforms, hiding architecture-specific details from the users.

The **Graph 500** effort may be helpful in this regard
-- *Frontiers in Massive Data Analysis,* NRC, 2013.



Graph Analysis is *Important* and *Pervasive* (and *Difficult*).

Graph data typically lacks "locality" and cannot be easily partitioned

This makes i
multiple or m

Two major in

- Sma
- Unp

**The Challen
Scale Graph**

From the co
identify a set

1) provide
comput
2) allow pr
3) are app
architec

The **Graph 500** effort may be helpful in this regard
-- *Frontiers in Massive Data Analysis,* NRC, 2013.

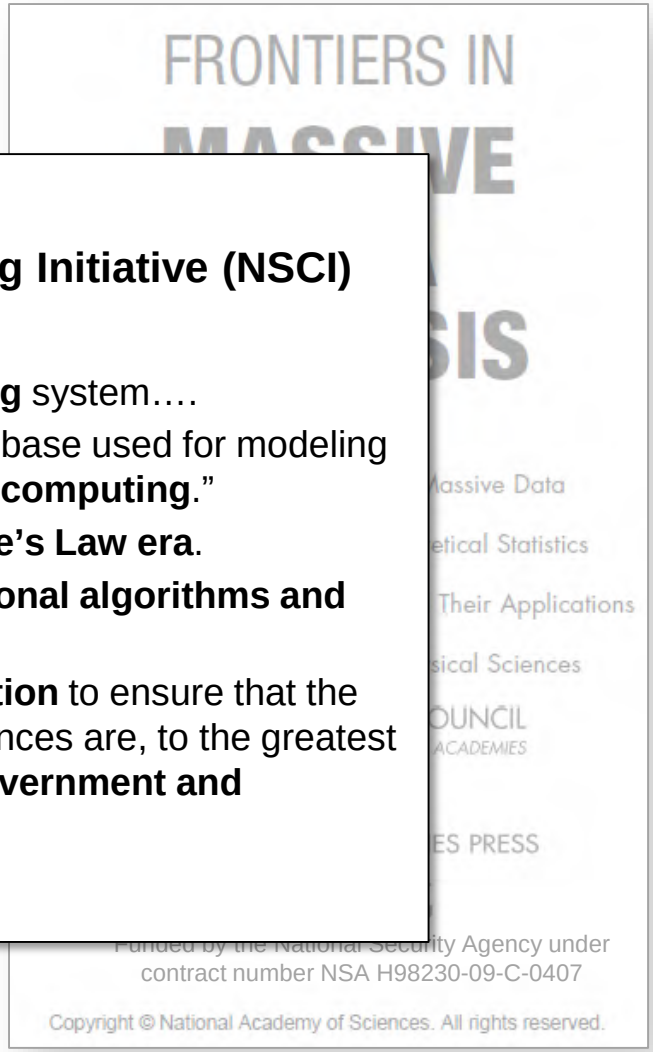
Executive Order

Creating a National Strategic Computing Initiative (NSCI)

Objectives

1) Accelerating delivery of an **exascale computing** system….
2) "Increasing coherence between the technology base used for modeling and simulation and that used **for data analytic computing**."
3) Path for future HPC systems **in the post-Moore's Law era**.
4) Addressing relevant factors such as…**foundational algorithms and software**…
5) Developing enduring **public-private collaboration** to ensure that the benefits of the research and development advances are, to the greatest extent, **shared between the United States Government and industrial and academic sectors**.

--President Barack Obama, *July 29*, 2015.



Graph Algorithms on Future Architectures

✓Fast, efficient graph analysis is **important** and **pervasive**.

Heterogeneous hardware is ~~coming~~ **here**.

We have built a library that helps developers use both.

Research question: Can a set of **primitives and operations** be defined that will **separate the concerns** between graph analytic application development and the increasing complexity of the underlying hardware?

Release library as open source.

Heterogeneous Hardware is ~~Coming~~ Here.

A8 processer boasts a multicore CPU, multicore GPU, and motion processor


TECH TIMES PERSONAL TECH BIZ

TAG Supercomputers , Top500 List , Tianhe-2

World's Fastest Supercomputer Tianhe-2 is Still No.1 a Year Later

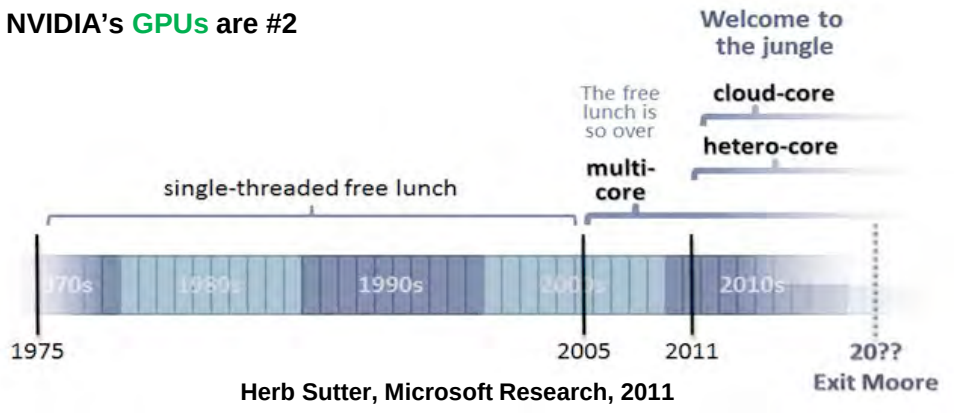
By Rhodi Lee, Tech Times | November 18, 11:21 PM

The fastest
computer in
China and it
supercompu

The Top500
the 500 fast
world and u
a year. It has
2 in the top
time in a rov

Intel's Xeon Phi accelerator holds on to top spot; NVIDIA's GPUs are #2

Our Hardware Focus: GPU



Herb Sutter, Microsoft Research, 2011

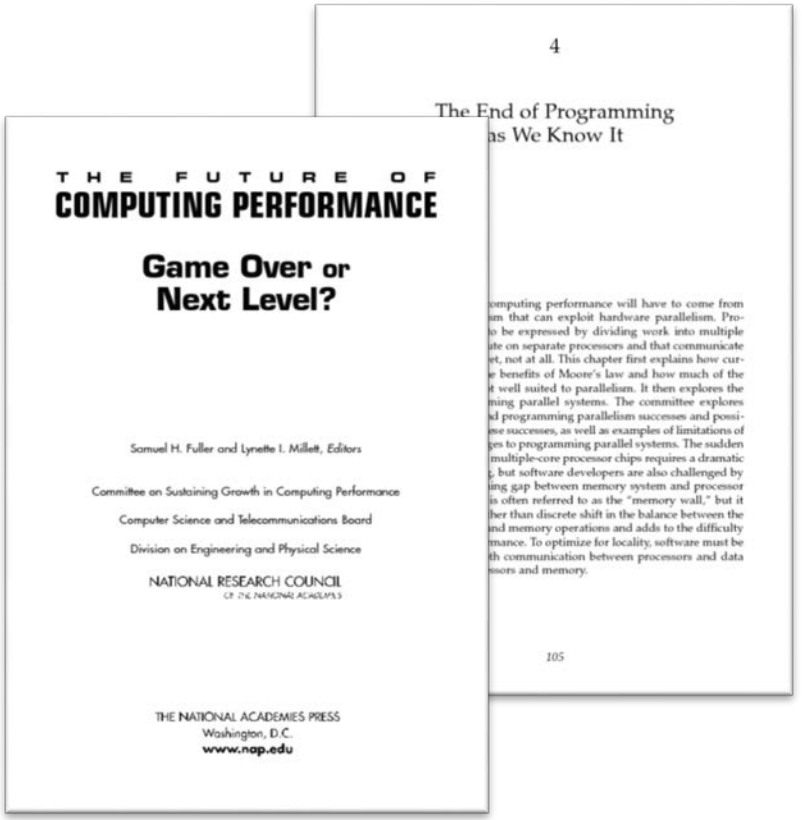
Why We Need Libraries and Frameworks to Exploit Parallel Hardware Architectures

"Future growth in computing performance will have to come from software parallelism that can exploit hardware parallelism.

Programs will need to be expressed by dividing work into multiple computations that execute on separate processors and that communicate infrequently or, better yet, not at all."

"The sudden shift from single-core to multiple-core processor chips requires a dramatic change in programming"

Simplifying the task of parallel programming requires software abstractions that provide powerful mechanisms for synchronization, load balance, communication, and locality … while hiding the underlying details**.**



The Future of Computing Performance
National Research Council of the National Academies

Graph Algorithms on Future Architectures

✓ Fast, efficient graph analysis is **important** and **pervasive**.

✓ Heterogeneous hardware is ~~coming~~ **here**.

We have built a library that helps developers use both.

Research question: Can a set of **primitives and operations** be defined that will **separate the concerns** between graph analytic application development and the increasing complexity of the underlying hardware?

Release library as open source.

Schedule/Items/Contents

Motivation

- Graph Algorithms
- Heterogeneous High Performance Computing (HHPC)

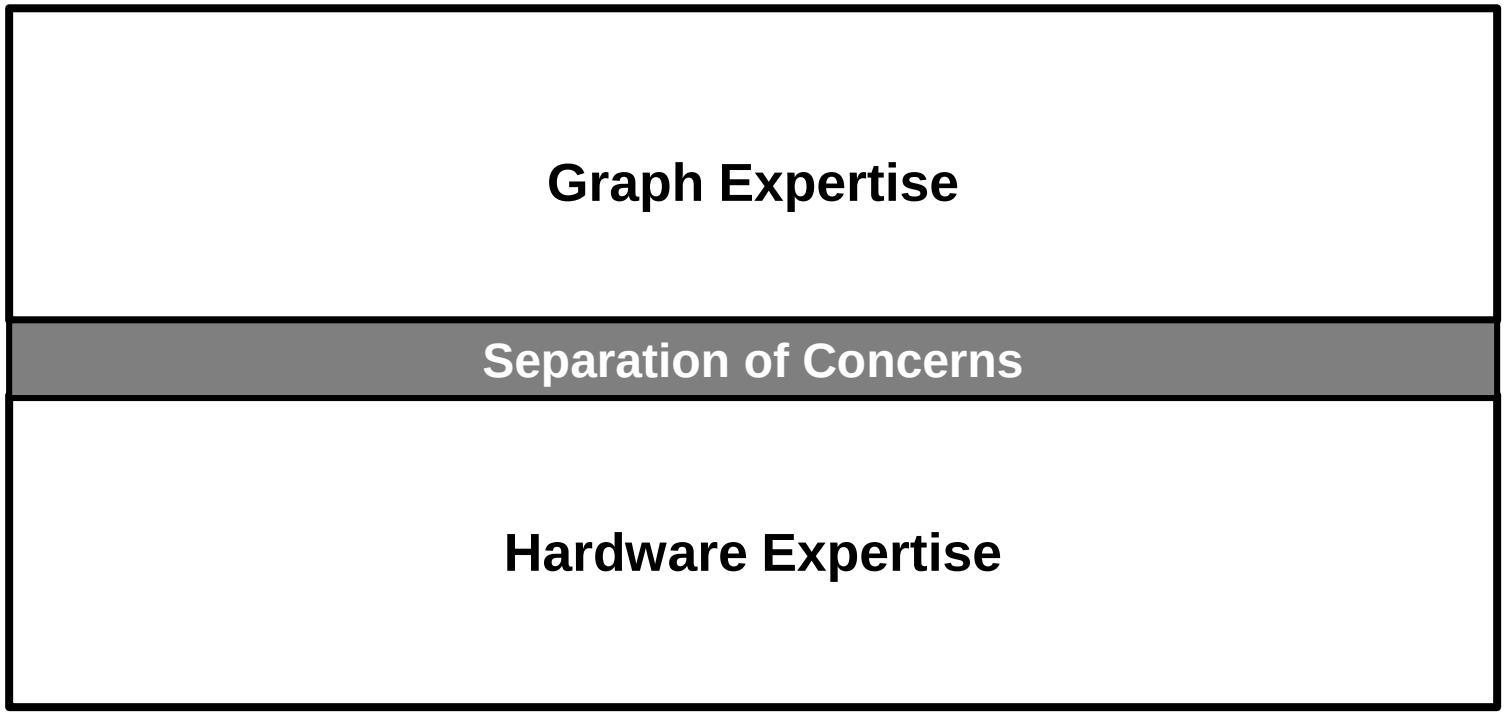
The Separation of Concerns

- **Library Architecture**
- **GraphBLAS API**

Example and Results

Future Work

Separation of Concerns



Research question: Can a **set of primitives and operations** be defined that will **separate the concerns** between graph analytic application development and the increasing complexity of the underlying hardware?

Motivation for Approach

Home page: http://graphblas.org

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

"It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of ***a standard set of primitive building blocks****. This paper is a position paper defining the problem and* ***announcing our intention to launch an open effort to define this standard****."*

Presented at the IEEE High Performance Extreme Computing Conference. Waltham, MA, Sept. 2013.

Our Collaborators: Indiana University

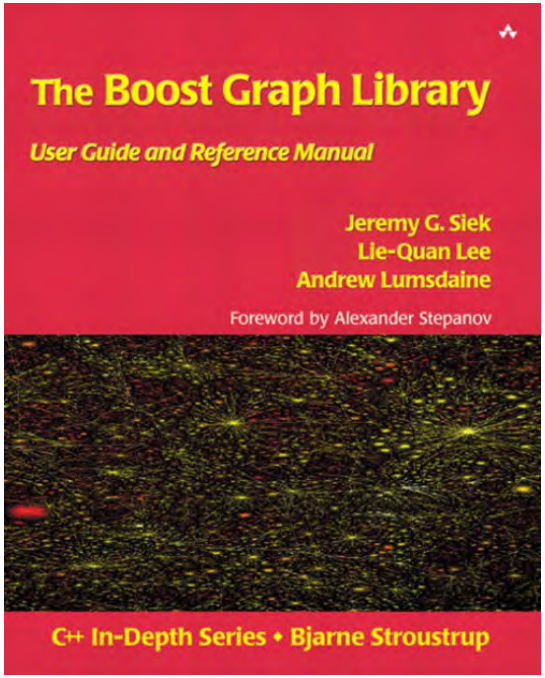
Andrew Lumsdaine

DIRECTOR, CENTER FOR RESEARCH IN EXTREME SCALE TECHNOLOGIES (CREST)

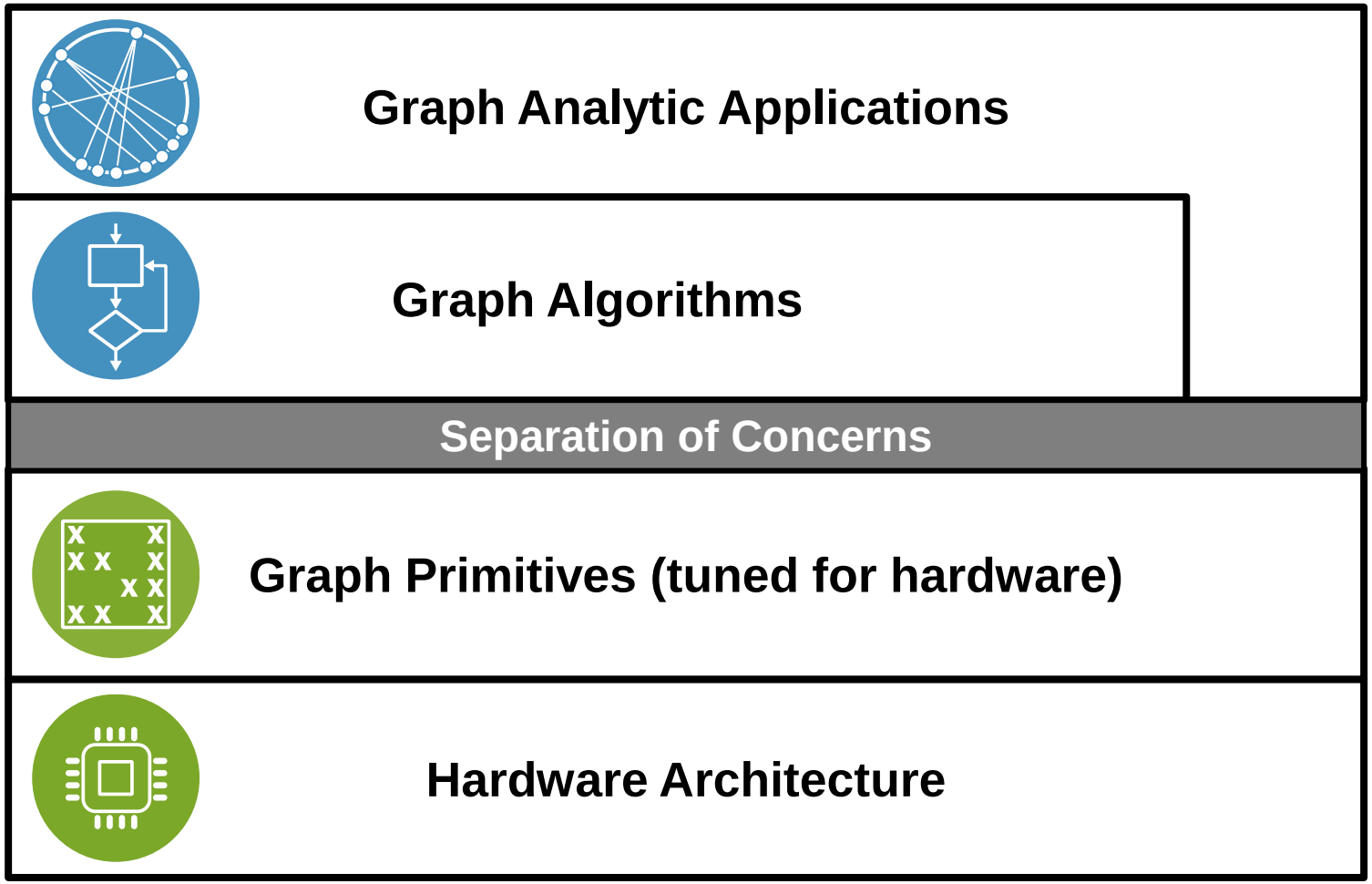
Dr. Andrew Lumsdaine is director of the Center for Research in Extreme Scale Technologies, associate director of the Digital Science Center, and a professor of computer science at Indiana University. His research interests include computational science and engineering, parallel and distributed computing, software engineering, generic programming, mathematical software, and numerical analysis.

Lumsdaine is a member of ACM, IEEE, and SIAM, as well as the MPI Forum, the BLAS technical forum, and the ISO C++ standards committee. He was previously a faculty member in the Department of Computer Science and Engineering at the University of Notre Dame. Lumsdaine received his PhD in electrical engineering and computer science from MIT.

INDIANA UNIVERSITY



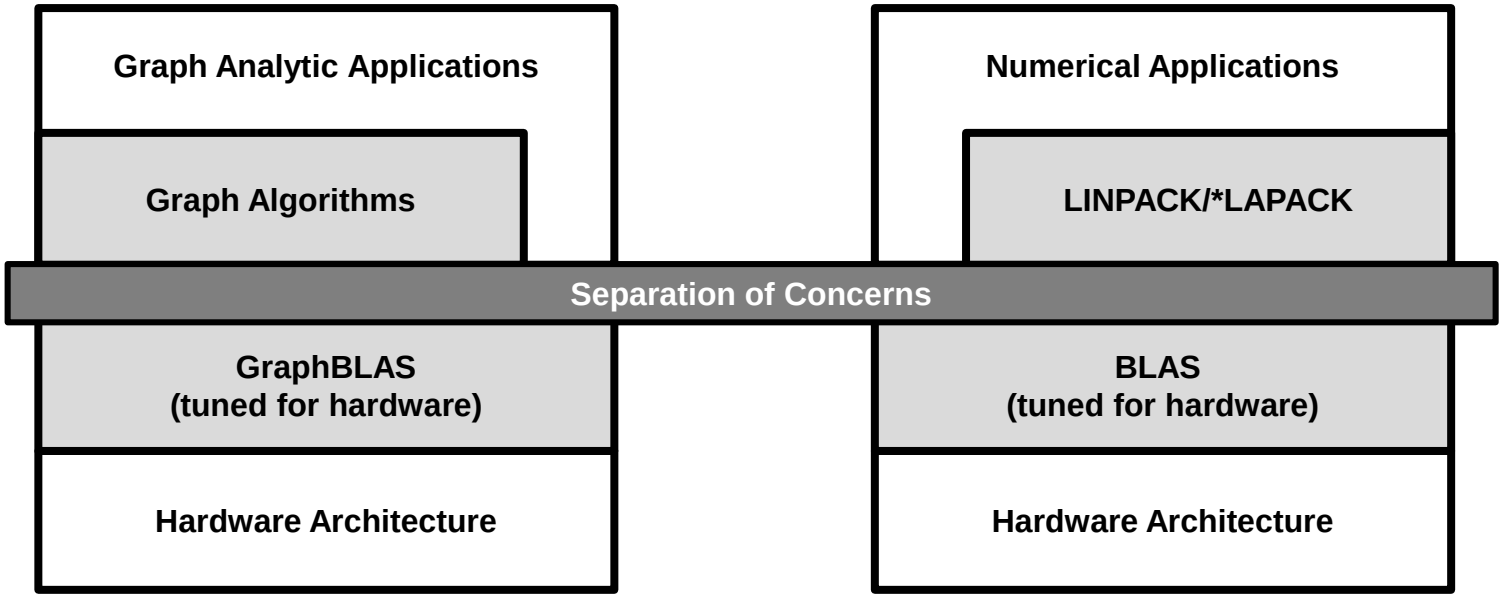
Software Architecture



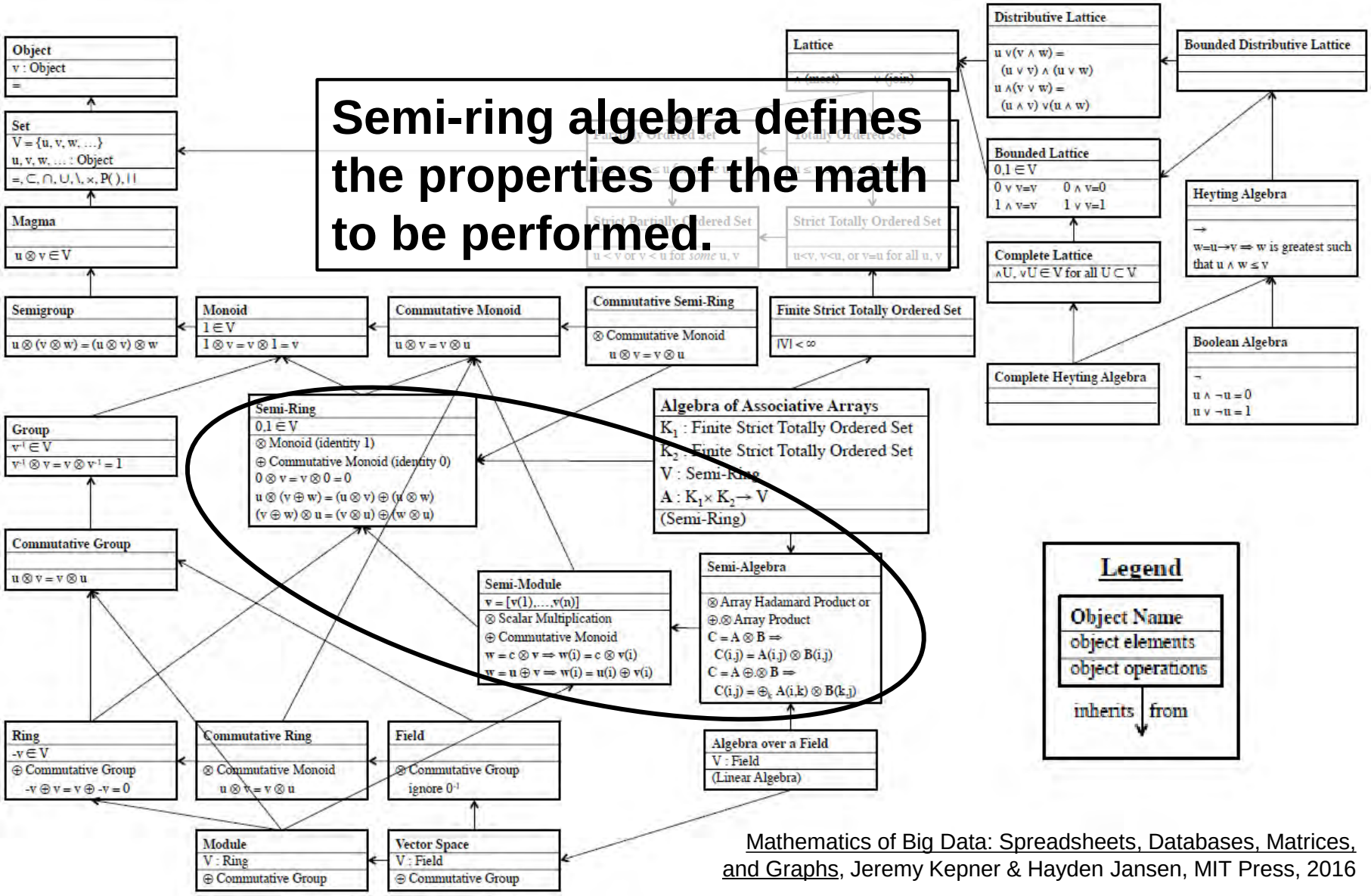
Components of this Research

Separation of Concerns (inspired by linear algebra)

- GraphBLAS movement within the graph analytics research community
- Defines a programming interface base on semi-ring algebra
- Similar to BLAS interface defined in the 1970s for Scientific Computing



Mathematics of Big Data



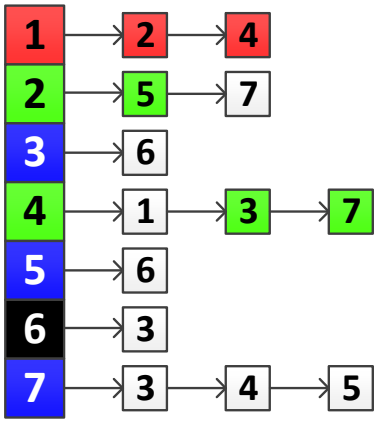
Mathematics of Big Data: Spreadsheets, Databases, Matrices, and Graphs, Jeremy Kepner & Hayden Jansen, MIT Press, 2016

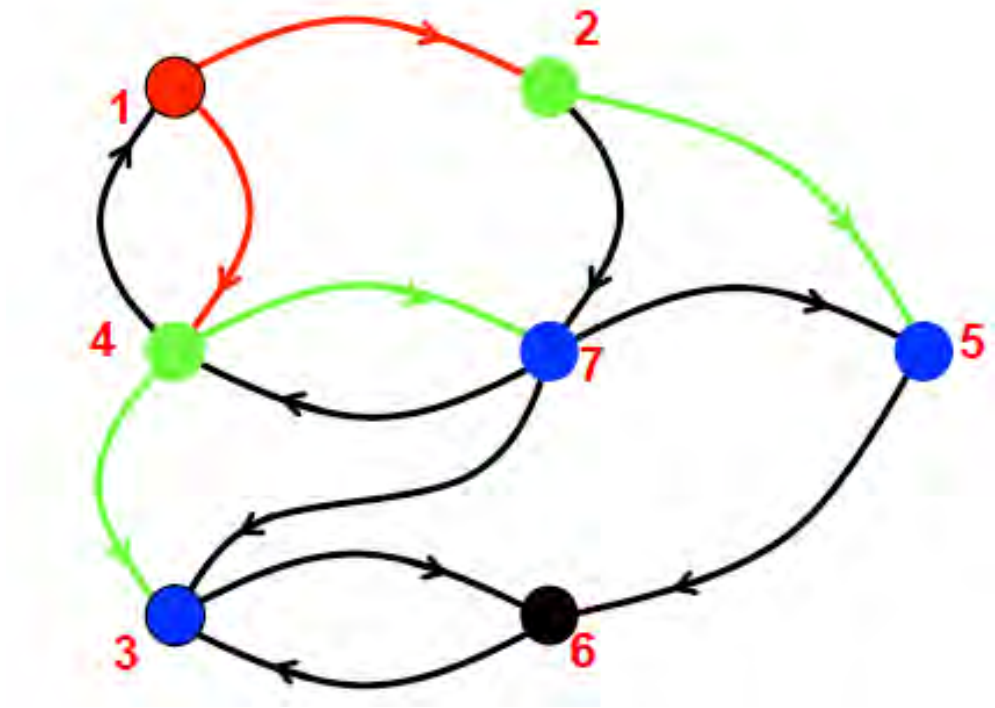
GraphBLAS Operations (as of 9/17/15)

Function Name	Description
BuildMatrix	Build a sparse matrix from row, column, value tuples
ExtractTuples	Extract the row, column, value tuples from a sparse matrix
MxM, MxV, VxM	Perform sparse matrix *multiplication* (e.g., BFS traversal)
Extract	Extract a sub-matrix from a larger matrix (e.g., sub-graph selection)
Assign	Assign to a sub-matrix of a larger matrix (e.g., sub-graph assignment)
EwiseAdd, EwiseMult	Element-wise *addition* and *multiplication* of matrices (e.g., graph union, intersection)
Apply	Apply *unary function* to each element of matrix (e.g., edge weight modification)
Reduce	*Reduce* along columns or rows of matrices (vertex degree)
Transpose	Swaps the rows and columns of a sparse matrix (e.g., reverse directed edges)

Key primitive data type: the sparse matrix

Sparse Matrices Represent Graphs

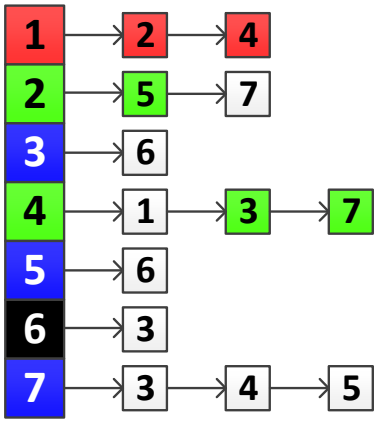




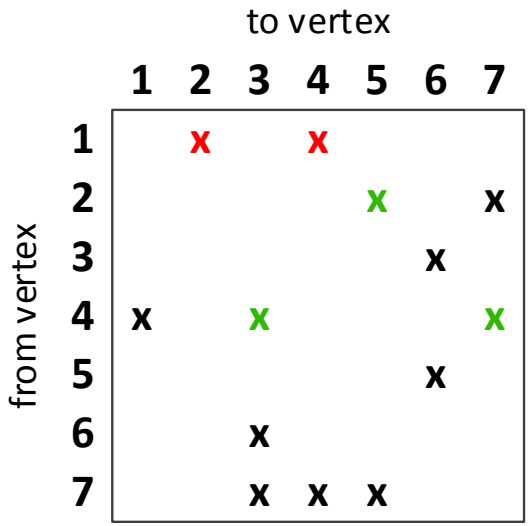
to vertex

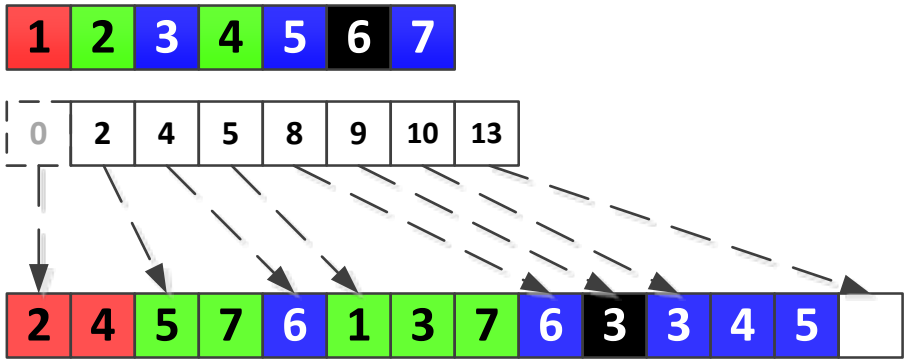
from vertex	1	2	3	4	5	6	7
1		x		x			
2					x		x
3						x	
4	x		x				x
5						x	
6			x				
7			x	x	x		

Sparse Matrices: Efficient Storage Formats



- Storage data structures is an active area of research.
- Efficient structures are tied intimately to memory architecture.
- Example: **Compressed Sparse Row (CSR)** use O(V) and O(E) dense arrays that help with multi-level cache hierarchies:



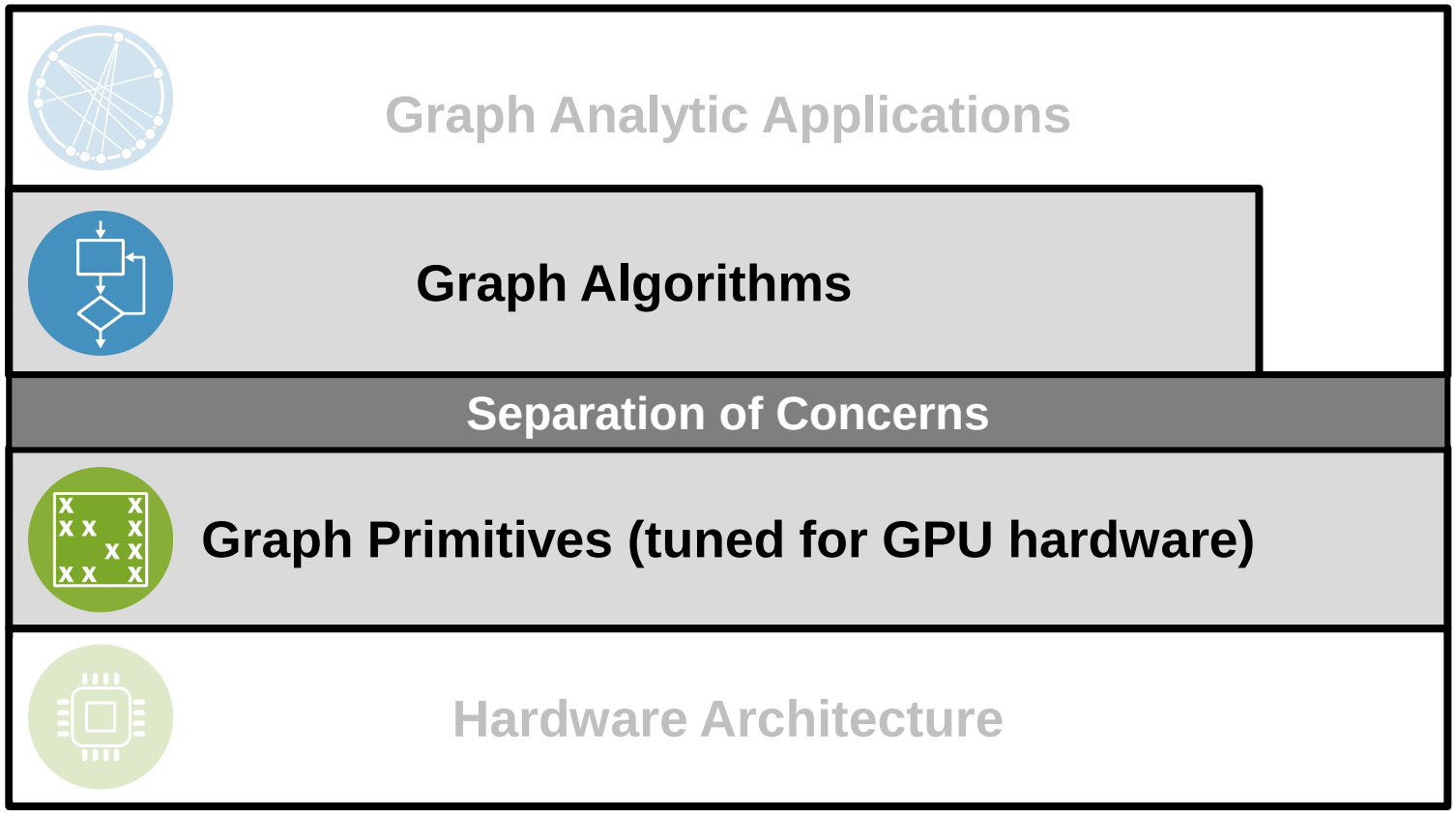


GraphBLAS Algorithms

“Classes” of algorithms built on GraphBLAS operations:

- Traversals: Breadth-First Search (BFS)
- Shortest Path/Cost Minimization (SSSP)
- Community Detection/Clustering
- Connected Components
- (Minimum) Spanning Tree
- Maximum Flow
- PageRank
- Metrics: diameter, betweenness centrality, triangle counting, etc.

Software Library Release



Open-source release: Scheduled for November 2015

Schedule/Items/Contents

Motivation

- Graph Algorithms
- Heterogeneous High Performance Computing (HHPC)

The Separation of Concerns

- Library Architecture
- GraphBLAS API

Example and Results

Future Work

Example

Breadth-first search in **five** GraphBLAS calls

```
void bfs(SparseMatrix const     &graph,          // sparse adjacency matrix
         WavefrontVector         wavefront,      // called with root (row vector)
         LevelVector            &level)
{
    visited = wavefront;
    level_val = 0;

    while (!wavefront.empty())
    {
        // traverse one level from current wavefront
        wavefront   = VxM(wavefront, graph, LogicalSemiring);

        // compute which from the next level have NOT been visited before
        not_visited = Apply(visited, LogicalNot);
        wavefront   = EWiseMult(not_visited, wavefront, LogicalAnd);

        // Assign the level to all newly visited vertices
        level_val++;
        level      += EwiseMult(wavefront, level_val, Mutiply);

        // Update the visited list
        visited     = EwiseAdd(visited, wavefront, LogicalOr);
    }
}
```

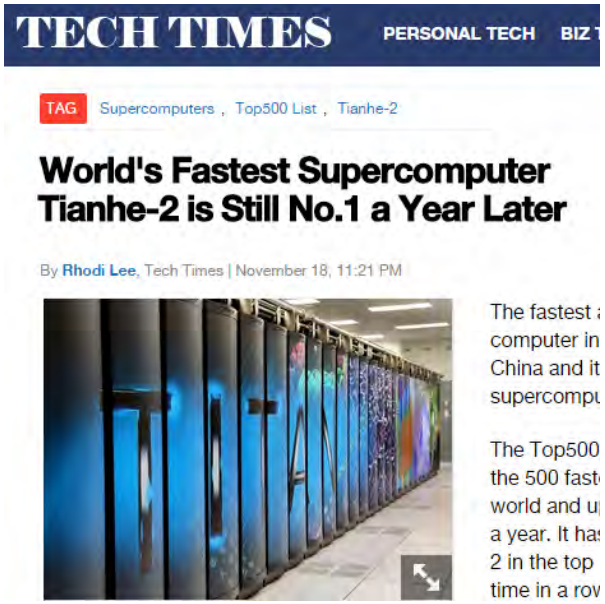
Example

Breadth-first search in **three** GraphBLAS calls with masks

```
void bfs(SparseMatrix const     &graph,          // sparse adjacency matrix
         WavefrontVector         wavefront,      // called with root (row vector)
         LevelVector            &level)
{
    visited = wavefront;
    level_val = 0;

    while (!wavefront.empty())
    {
        // traverse one level from current wavefront
        wavefront   = VxM(wavefront, graph, LogicalSemiring, mask=Not(visited));




        // Assign the level to all newly visited vertices
        level_val++;
        level      += EwiseMult(wavefront, level_val, Mutiply);

        // Update the visited list
        visited     = EwiseAdd(visited, wavefront, LogicalOr);
    }
}
```

Results from the Hardware API Level

A8 processer boasts a multicore CPU, multicore GPU and motion processor


TECH TIMES PERSONAL TECH BIZ

TAG Supercomputers , Top500 List , Tianhe-2

World's Fastest Supercomputer Tianhe-2 is Still No.1 a Year Later

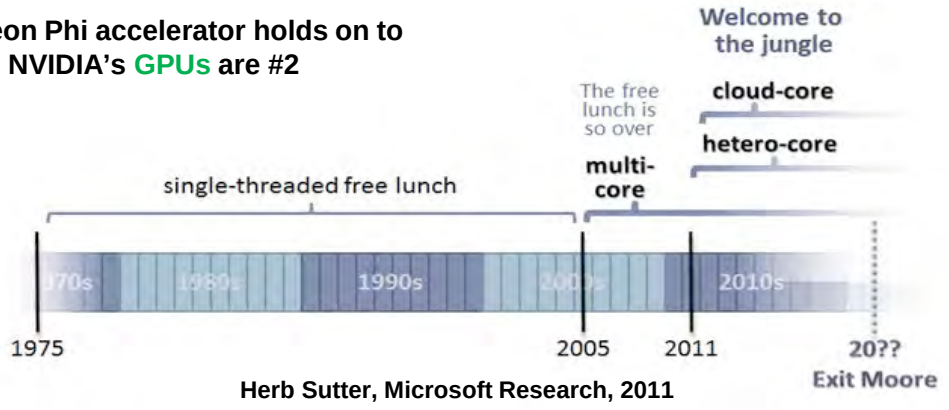
By Rhodi Lee, Tech Times | November 18, 11:21 PM

The fastest
computer in
China and it
supercompu

The Top500
the 500 fast
world and u
a year. It has
2 in the top
time in a rov

Intel's Xeon Phi accelerator holds on to top spot; NVIDIA's GPUs are #2

Reminder: our Hardware Focus: GPU

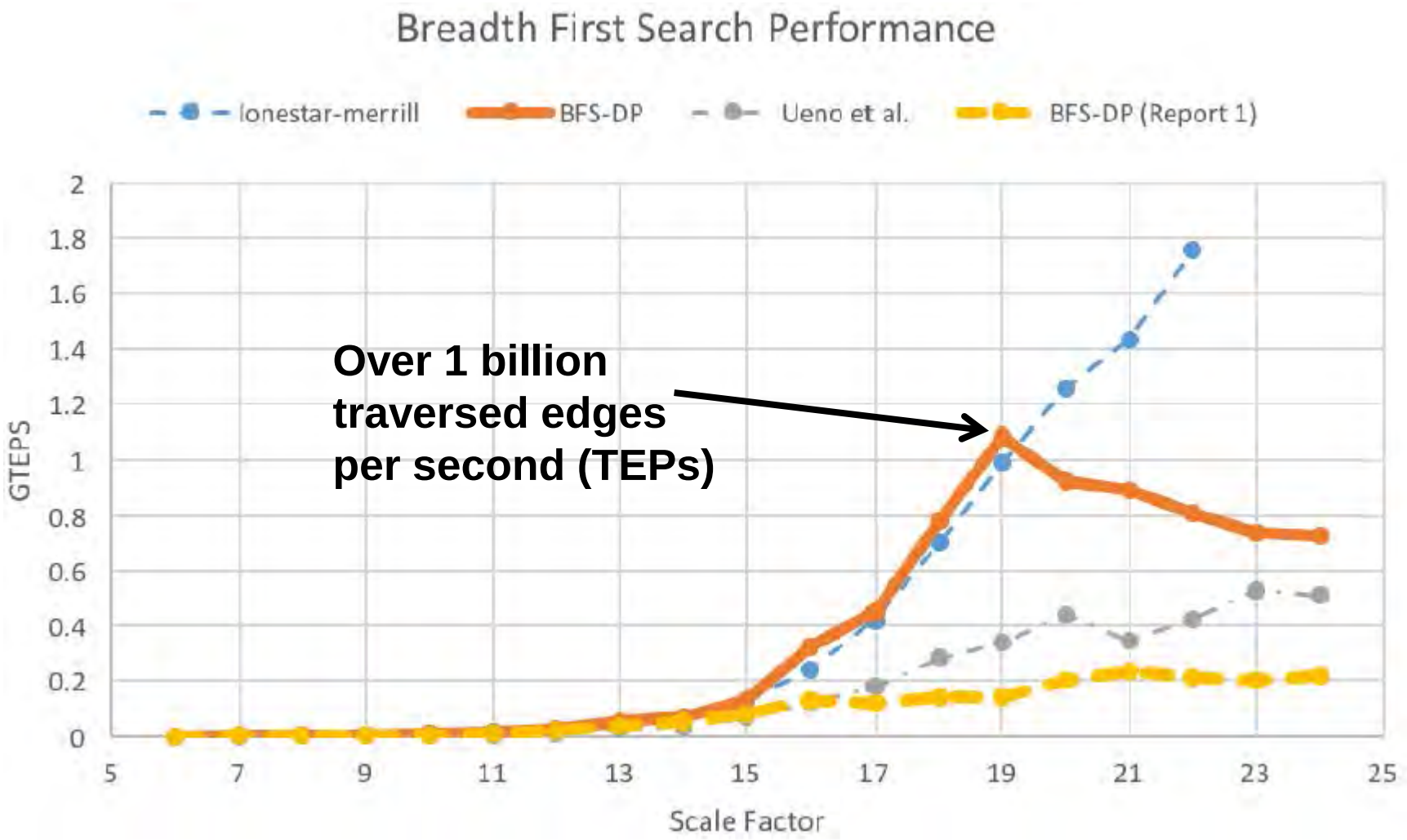


Herb Sutter, Microsoft Research, 2011

Results (~250 lines of GPU code using Dynamic Parallelism)

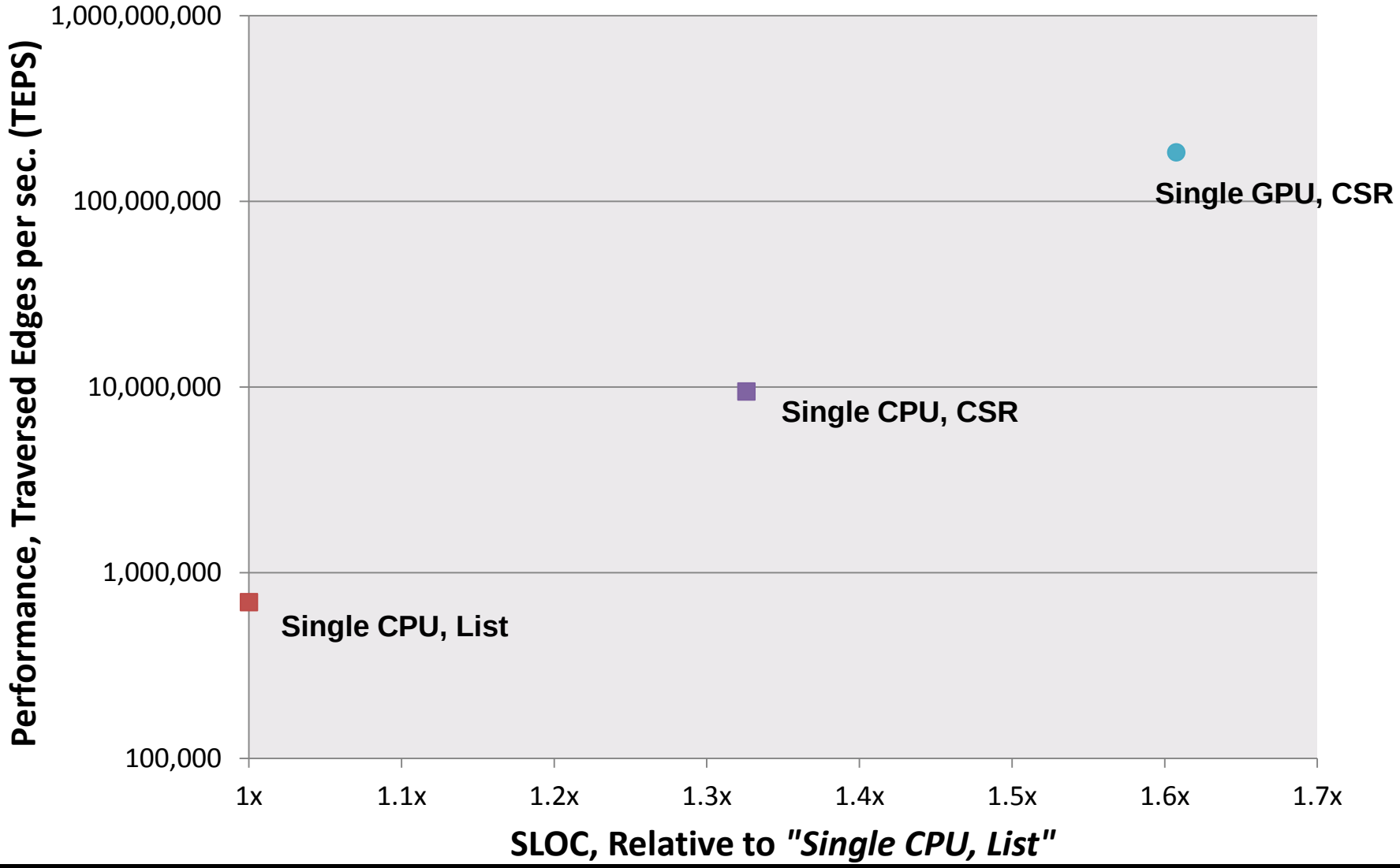
```
__device__ __forceinline__ bool warp_cull(int neighborid)
{
    volatile __shared__ uint32_t scratch[WARP_SIZE][128];
    uint16_t warpid=threadIdx.x / (WARP_SIZE);
    uint32_t hash=neighborid&127;
    scratch[warpid][hash] = neighborid;
    uint32_t retrieved=scratch[warpid][hash];
    if (retrieved == neighborid){
        scratch[warpid][hash] = threadIdx.x;
        if (scratch[warpid][hash] != threadIdx.x) {
            return true;
        }
    }
    return false;
}

__device__ __forceinline__ void warp_reap(uint32_t parent, uint32_t neighbors, uusi::EdgeValue* e, gafa::BitMap * q, int *parentMap){
    __shared__ uint32_t comm[WARP_SIZE][3];
    __shared__ uusi::EdgeValue* evs[WARP_SIZE];
    uint16_t laneid=threadIdx.x & (WARP_SIZE-1);
    uint16_t warpid=threadIdx.x / (WARP_SIZE);
    while(__any(neighbors)) {
        //per warp: one write will succeed
        if (neighbors) { comm[warpid][0]=laneid; }

        //winner descr:
        if (comm[warpid][0] == laneid) {
            comm[warpid][1] = neighbors;
            comm[warpid][2] = parent;
            evs[warpid] = e;
            if ((uint64_t)e == 1 || (uint64_t)e==2) {
                printf("not right at thread %d, neighbors=%d, code %d\n",parent,neighbors,e);
                blkSync();
            }
            neighbors=0;
        }
        while(comm[warpid][1]){
            if(comm[warpid][1] >= 32){
                comm[warpid][1]-=32;
                uusi::EdgeValue *edgevalue=evs[warpid];
                uint32_t node=edgevalue[laneid].dst;
                //if not marked in parentmap, and not in queue:
                if (1 != gafa::get_bitmap_value_at(q,node) &&
                    -1 == atomicGet(parentMap,node)) {
                    gafa::set_bitmap_value_at(q, node, 1);
                    atomicExch(parentMap+node,comm[warpid][2]);
                }
                //increment edgevalue pointer:
                evs[warpid]+=32;
            }
            else if(laneid<comm[warpid][1]){
                comm[warpid][1]=0;
                uusi::EdgeValue *edgevalue=evs[warpid];
                uint32_t node=edgevalue[laneid].dst;
                //if not marked in parentmap, and not in queue:
                if (1!=gafa::get_bitmap_value_at(q,node) &&
                    -1==atomicGet(parentMap,node)){
                    gafa::set_bitmap_value_at(q, node, 1);
                    atomicExch(parentMap+node,comm[warpid][2]);
                }
            }
        }
    }
}

//very simple kernel for now.
__global__ void kernel1(uusi::GPUNode* nodes, uint32_t size, gafa::BitMap * q1, gafa::BitMap * q2, int *parentMap){
    int idx = threadIdx.x+blockDim.x*blockIdx.x;
    __shared__ uusi::EdgeValue* leftovers[1024];
    __shared__ uint32_t leftover_size[1024];
    //fillshmemaddrs:
    leftover_size[threadIdx.x]=0;
    leftovers[threadIdx.x] = (uusi::EdgeValue*)0x3;
    if(idx<size){
        if(gafa::get_bitmap_value_at(q1,idx) && atomicGet(parentMap,idx)!=-1) {
            uint32_t edges=nodes[idx].neighbors;
            if(edges>=KERNEL_TH){
                //get remainder:
                int b,t;
                gafa::get_threads_blocks(edges,&b,&t,1024);
                kernel2<<<b,t>>>(idx,edges,nodes[idx],q2,parentMap);
            }
            else{
                leftovers[threadIdx.x]=nodes[idx].ev;
                leftover_size[threadIdx.x]=nodes[idx].neighbors;
            }
        }
        //set not-processed threads
        else {
            leftover_size[threadIdx.x]=0;
            leftovers[threadIdx.x]=(uusi::EdgeValue*)2;
        }    }//end of size limit
    //set out-of-bounds threads' noofedges
    else{
        leftover_size[threadIdx.x]=0;
        leftovers[threadIdx.x]=(uusi::EdgeValue*)1;
    }
    __syncthreads();
    warp_reap(idx, leftover_size[threadIdx.x], leftovers[threadIdx.x], q2, parentMap);
}

//do all the visits in kernel2: use parentmap as visited map
__global__ void kernel2(uint32_t parent, uint32_t size, uusi::GPUNode e, gafa::BitMap * q, int *parentMap){
    int idx = threadIdx.x+blockDim.x*blockIdx.x;
    if (idx < size) {
        uint32_t node=e.ev[idx].dst;
        //if not marked in parentmap, and not in queue:
        if (1!=gafa::get_bitmap_value_at(q,node) && -1==atomicGet(parentMap,node))
        {
            gafa::set_bitmap_value_at(q, node, 1);
            atomicExch(parentMap+node,parent);
        }
    }
}

template <typename Int>
__global__ void pm(gafa::BitMap *data, Int size) {
    for(int i=0;i<size;i++){
        printf("%d ",gafa::get_bitmap_value_at(data,i));
        blkSync();
    }
}

int64_t* run_bfs(uusi::GPUGraph n, int source){
    gafa::set_max_tpb();
    gafa::BitMap *q1,*q2;
    int t,b,count=1, k=0;
    uint32_t *count_d;
    int *parentMap, *parentMap_h;
    q1 = gafa::new_device_bitmap(n.getNodes());
    q2=gafa::new_device_bitmap(n.getNodes());
    gafa::get_threads_blocks(n.getNodes(),&b,&t,1024);
    cudaMemset((void*)(q1)+(uint8_t)(source/8),(0x01 << (source % 8)),1);
    cuMalloc(&parentMap,n.getNodes()*(sizeof(int)));
    cuMalloc(&count_d,4);
    gafa::set_value_d(parentMap,-1,n.getNodes());
    cuCopy(parentMap+source,&source,4,h2d);
    gafa::setChildLimit(256000);
    printf("blocks=%d,threads=%d\n",b,t);
    devSync();
    tic();
    while(count!=0){
        if(k%2){
            kernel1<<<b,t>>>(n.getGraph(), n.getNodes(), q1, q2, parentMap);
            devSync();
            gafa::zero(q1,bitmapSize(n.getNodes()));
            gafa::zero(count_d,1);
            or_reduction<<<b,t>>>(q2,(uint32_t)bitmapSize(n.getNodes()),count_d);
            devSync();
            cuCopy(&count,count_d,4,d2h);
            devSync();
            k++;
        }
        else{
            kernel1<<<b,t>>>(n.getGraph(), n.getNodes(), q2, q1, parentMap);
            devSync();
            gafa::zero(q2,bitmapSize(n.getNodes()));
            gafa::zero(count_d,1);
            or_reduction<<<b,t>>>(q1,(uint32_t)bitmapSize(n.getNodes()),count_d);
            devSync();
            cuCopy(&count,count_d,4,d2h);
            devSync();
            k++;
        }
    }
    uusi::elapsed_time=toc();
    printf("g500 timer: %f s, or %f ms\n",uusi::elapsed_time,uusi::elapsed_time*1000);
    parentMap_h=(int*)malloc(4*n.getNodes());
    int64_t *pm=(int64_t*)malloc(8*n.getNodes());
    devSync();
    cuCopy(parentMap_h,parentMap,sizeof(int)*n.getNodes(),d2h);
    devSync();
    for(uint32_t i=0;i<n.getNodes();i++){
        pm[i]=parentMap_h[i];
    }
    //dump parent map:
    gafa::dump_array_to_file(parentMap, n.getNodes(), "pmdump.txt");
    cudaFree(parentMap);
    cudaFree(count_d);
    cudaFree(q1);
    cudaFree(q2);
    free(parentMap_h);
    return pm;
}
```

Results: GPU Dynamic Parallelism (DP)

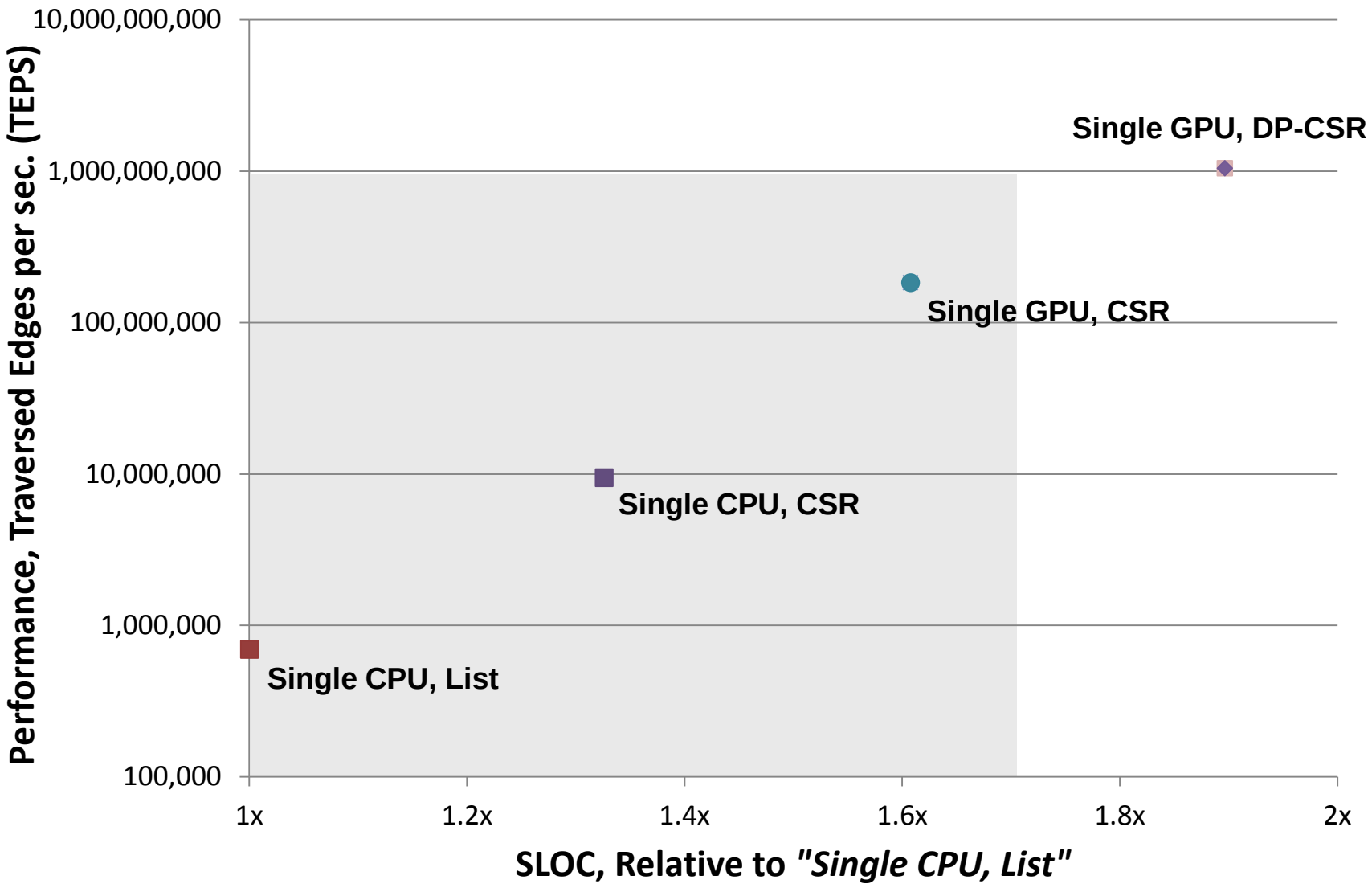


P. Zhang, et al.,, "Dynamic Parallelism for Simple and Efficient GPU Graph Algorithms," to appear in *5th IEEE Workshop on Irregular Applications: Architectures and Algorithms, Nov 2015.*

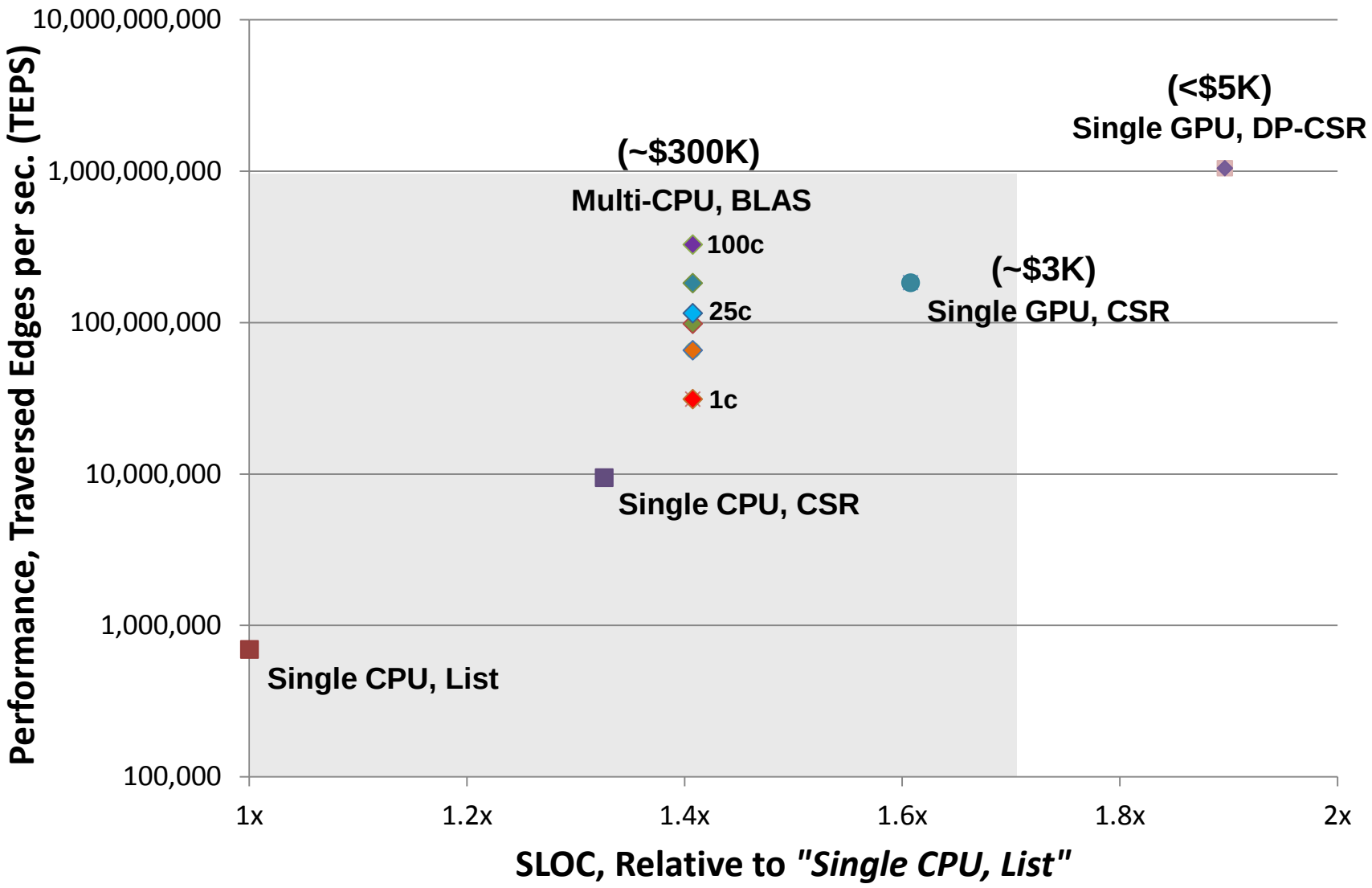
Previous Results: Performance, Complexity, and Cost



Results: Performance, Complexity, and Cost



Results: Performance, Complexity, and Cost



Schedule/Items/Contents

Motivation

- Graph Algorithms
- Heterogeneous High Performance Computing (HHPC)

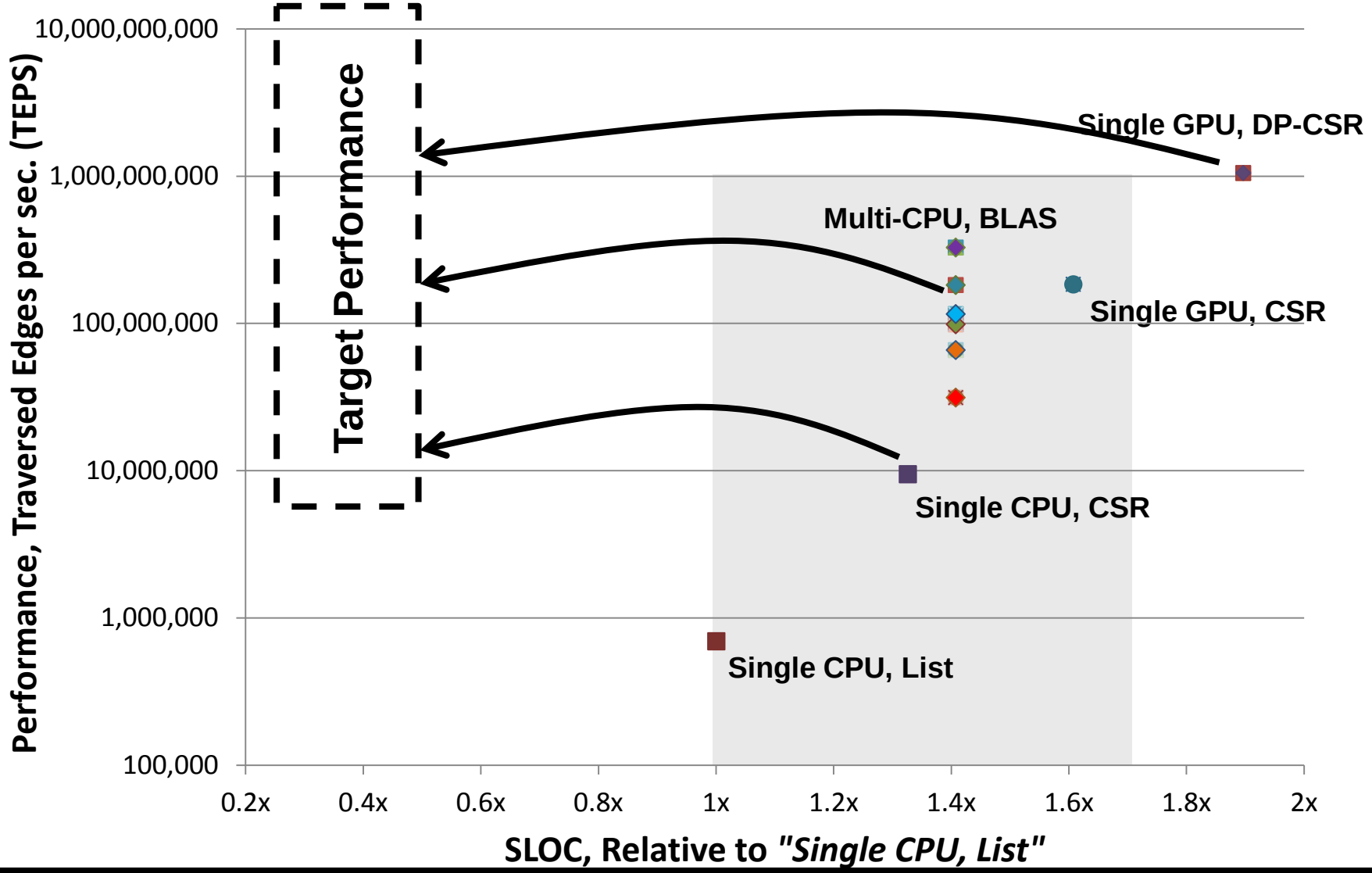
The Separation of Concerns

- Library Architecture
- GraphBLAS API

Example and Results

Future Work

Future Work: Performance, Complexity, and Cost



Future Work

Complete the GraphBLAS API Specification

- We are working on the C++ Reference Implementation.

Collaborations with special purpose hardware developers

- FPGA designers at MIT/LL
- 3D memory architectures at CMU

Incorporating Sparse Solvers

- Spectral clustering
- Principal component analysis

Contact Information

Scott McMillan

Senior Member of Technical Staff

SEI Emerging Technology Center

Telephone: +1 412-268-5156

Email: smcmillan@sei.cmu.edu

U.S. Mail

Software Engineering Institute

Customer Relations

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

USA

Web

www.sei.cmu.edu

www.sei.cmu.edu/contact.cfm

Customer Relations

Email: info@sei.cmu.edu

Telephone: +1 412-268-5800

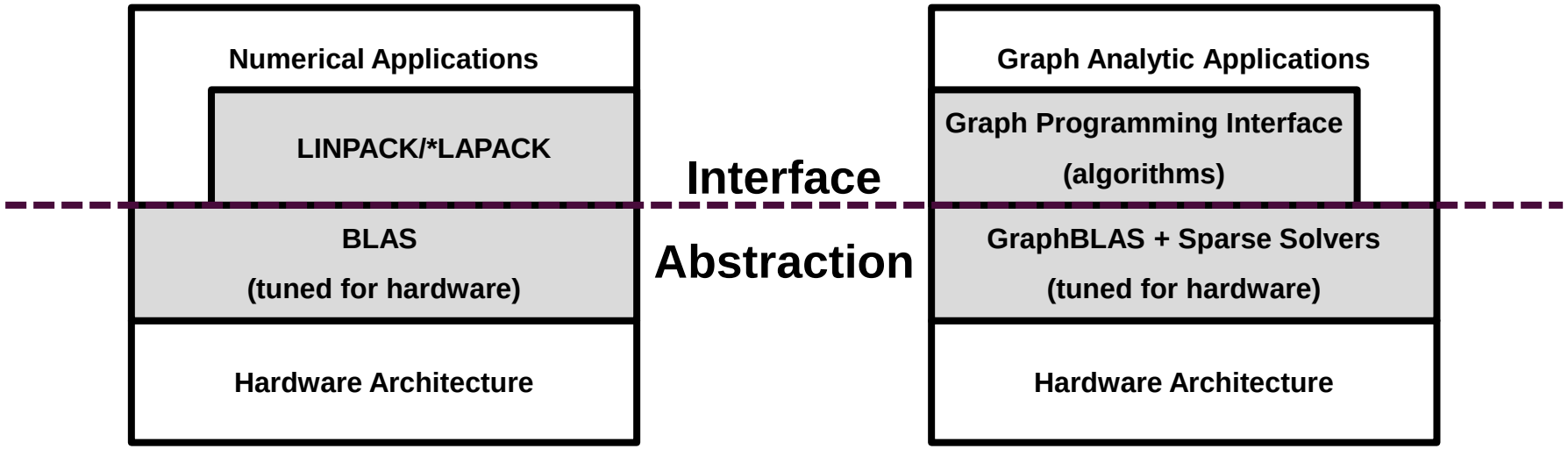
SEI Phone: +1 412-268-5800

SEI Fax: +1 412-268-6257

BACKUPS?

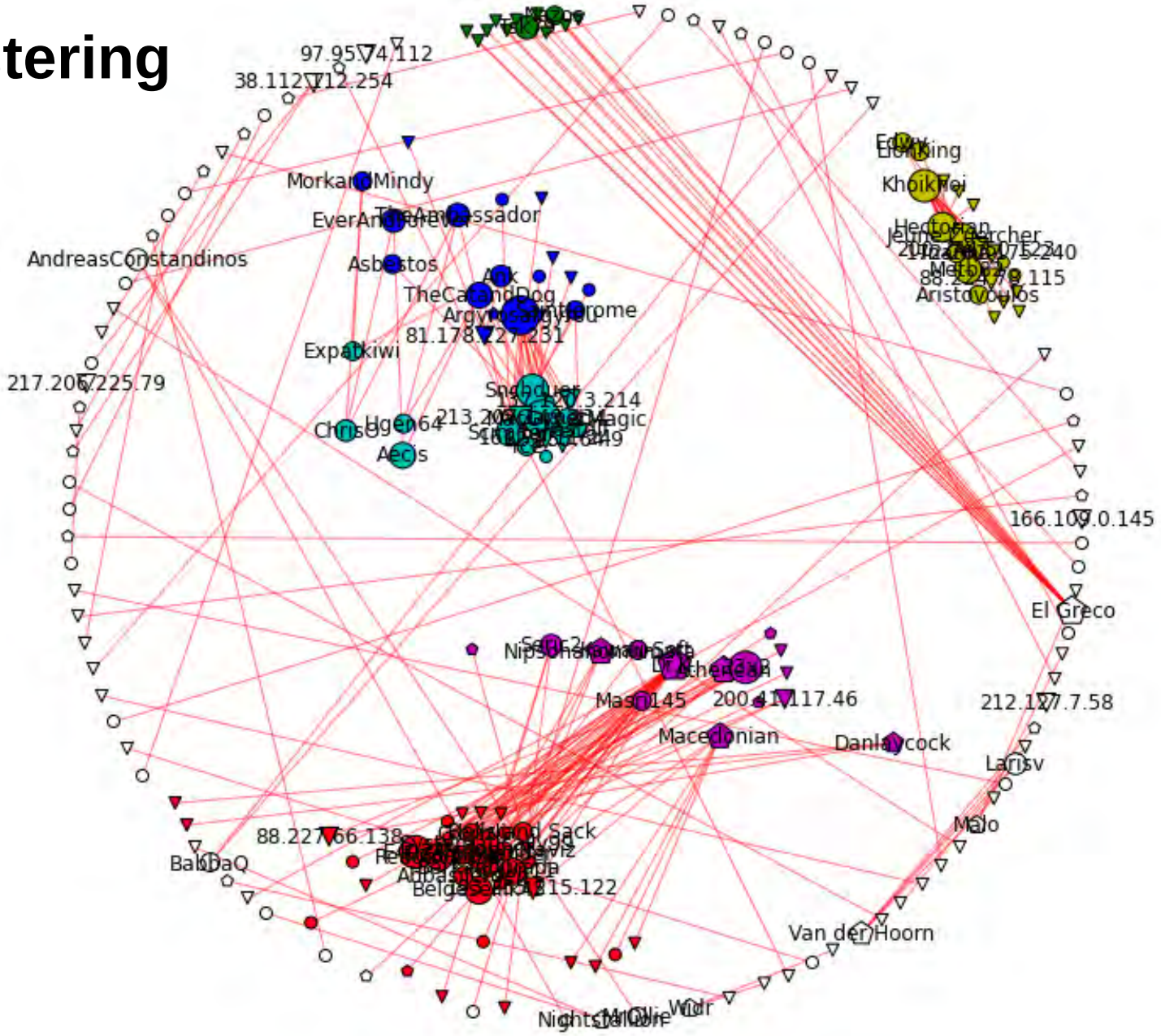
Concepts in this Research

Inspired by Linear Algebra



Text here

Clustering



Results: Performance, Complexity, and Cost