Graph Algorithms on Future Architectures
Scott McMillan, PhD

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
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
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**
- **Shortest Path**
  - Cost Minimization
  - Max Flow
- **Connected Components**
- **PageRank**
- **Centrality**
- **Clustering**
- **Malware Distribution Networks**
- **United States Interstate Highway System**
- **Social Networks**
- **Revert graph showing editor conflict on the “Cyprus dispute” Wikipedia page, 2015**
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 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

---

**Graph Analysis is Important and Pervasive (and Difficult).**
Graph Analysis is \textit{Important} and \textit{Pervasive} (and \textit{Difficult}).

Graph data typically lacks “locality” and cannot be easily partitioned.
This makes a single graph difficult to distribute across multiple or many processes.

Two major issues:
1) Small computation to communication ratio
2) Unpredictability of data access

Executive Order

Creating a National Strategic Computing Initiative (NSCI)

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

--President Barack Obama, \textit{July 29, 2015}.

The \textit{Graph 500} effort may be helpful in this regard

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 processor boasts a multicore CPU, multicore GPU, and motion processor.

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.
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.
Motivation
- Graph Algorithms
- Heterogeneous High Performance Computing (HHPC)

The Separation of Concerns
- Library Architecture
- GraphBLAS API

Example and Results

Future Work
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

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 Mellon 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.”

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.
Software Architecture

Graph Analytic Applications

Graph Algorithms

Separation of Concerns

Graph Primitives (tuned for hardware)

Hardware 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

Semi-ring algebra defines the properties of the math to be performed.

# GraphBLAS Operations (as of 9/17/15)

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

**Key primitive data type: the sparse matrix**
Sparse Matrices Represent Graphs

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

From vertex to vertex.
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

Graph Analytic Applications

Graph Algorithms

Separation of Concerns

Graph Primitives (tuned for GPU hardware)

Hardware Architecture

Open-source release: Scheduled for November 2015
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

```c
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

```c
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, Multiply);

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

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

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

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

// Update the visited list
visited = EwiseAdd(visited, wavefront, LogicalOr);
```
Results from the Hardware API Level

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

Herb Sutter, Microsoft Research, 2011

Reminder: our Hardware Focus: GPU
Results (~250 lines of GPU code using Dynamic Parallelism)

```cpp
__device__ __forceinline__ void warp_reap(uint32_t parent, uint32_t neighbors, uusi::EdgeValue* e, gafa::BitMap * q, int *parentMap)
{
    // Very simple kernel for now.
    //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)
    {
        //do all the visits in kernel2: use parentmap as visited map
        //set not-processed threads
        if (retrieved == neighborid)
            scratch[warpid][hash] = neighborid;
        scratch[warpid][hash] = threadIdx.x + blockDim.x;
        if (scratch[warpid][hash] == threadIdx.x + blockDim.x)
            return false;
    }

    //set out-of-bounds threads' noofedges
    else
        leftovers[threadIdx.x] = 0;
    leftover_size[threadIdx.x] = (uusi::EdgeValue*)1;
    __syncthreads();
    warp_reap(d, leftovers[threadIdx.x], leftoverers[threadIdx.x], q, 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)
{
    //do all the visits in kernel2: use parentmap as visited map
    int idx = threadIdx.x + blockDim.x * blockIdx.x;
    __shared__ uusi::GPUNode n[1024];
    __shared__ uusi::GPUNode *nptrp = &n[0];
    __shared__ gafa::BitMap *q = &q[0];
    __shared__ int *parentMap;
    __shared__ int64_t *pm = (int64_t*)malloc(8 * n->getNodes());
    parentMap = (int*)malloc(4 * n->getNodes());
    printf("g500 timer: %f s, or %f ms\n", uusi::elapsed_time, uusi::elapsed_time * 1000);
    uusi::elapsed_time = toc();
}

for (int i = 0; i < n->getNodes(); i++)
{
    printf("%d \", gafa::get_bitmap_value_at(data, i));
    PM[0] = 1;
    devSync();
}
```

---

© 2015 Carnegie Mellon University

October 7–8, 2015

SEI Research Review 2015

Carnegie Mellon University

Distribution Statement A: Approved for Public Release; Distribution is Unlimited
Results: GPU Dynamic Parallelism (DP)

Over 1 billion traversed edges per second (TEPs)

Previous Results: Performance, Complexity, and Cost

Performance, Traversed Edges per sec. (TEPS)

SLOC, Relative to "Single CPU, List"
Results: Performance, Complexity, and Cost

- **Single CPU, List**
- **Single CPU, CSR**
- **Single GPU, CSR**
- **Single GPU, DP-CSR**

Performance, Traversed Edges per sec. (TEPS)

SLOC, Relative to "Single CPU, List"
Results: Performance, Complexity, and Cost

<table>
<thead>
<tr>
<th>SLOC, Relative to &quot;Single CPU, List&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x</td>
</tr>
<tr>
<td>1.2x</td>
</tr>
<tr>
<td>1.4x</td>
</tr>
<tr>
<td>1.6x</td>
</tr>
<tr>
<td>1.8x</td>
</tr>
<tr>
<td>2x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance, Traversed Edges per sec. (TEPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000,000,000</td>
</tr>
<tr>
<td>1,000,000,000</td>
</tr>
<tr>
<td>100,000,000</td>
</tr>
<tr>
<td>10,000,000</td>
</tr>
<tr>
<td>1,000,000</td>
</tr>
<tr>
<td>100,000</td>
</tr>
</tbody>
</table>

- Single CPU, List
- Single CPU, CSR
- Multi-CPU, BLAS
- Single GPU, CSR
- Single GPU, DP-CSRR

- (~$300K) Single GPU, DP-CSRR
- (~$3K) Single GPU, CSR
- (~$3K) Multi-CPU, BLAS
- (1c) Single CPU, List
- (25c) Single CPU, CSR
- (100c) Multi-CPU, BLAS
- (<$5K) Single GPU, DP-CSRR
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

- Single CPU, List
- Multi-CPU, BLAS
- Single GPU, CSR
- Single GPU, DP-CSR

Performance, Traversed Edges per sec. (TEPS)

SLOC, Relative to "Single CPU, List"
Motivation

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

Numerical Applications
- LINPACK/*LAPACK
- BLAS (tuned for hardware)
- Hardware Architecture

Graph Analytic Applications
- Graph Programming Interface (algorithms)
- GraphBLAS + Sparse Solvers (tuned for hardware)
- Hardware Architecture

Text here
Clustering
Results: Performance, Complexity, and Cost

Performance, Traversed Edges per sec. (TEPS)

- Single CPU, List
- Single CPU, CSR
- Multi-CPU, BLAS (~$300K)
- Single GPU, CSR (~$3K)

SLOC, Relative to "Single CPU, List"