Automated Code Generation for High-Performance, Future-Compatible Graph Libraries

Dr. Scott McMillan, Senior Research Scientist
CMU PI: Prof. Franz Franchetti, ECE
Copyright 2017 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.


DM17-0774
Data-Intensive Computing Efforts at the SEI

- **2013-15**: Line: Graph Algorithms on Future Architectures
- **2016**: Line: GraphBLAS
- **2017-18**: Line: Spiral Graph: Automated Code-Generation for Graph Algorithms

**2014-current**: GraphBLAS Forum (Jeremy Kepner MIT/LL, chair)

**2016-17**: Line: Big Learning Benchmarks

**2014**: C3E Challenge: Graph analytics for detecting APTs in network data (SCORE)

**2015**: Development and Release of GraphBLAS Template Library, v 1.0 (with Indiana U)

**2015-current**: Development of GraphBLAS C API Specification (w/ LBNL, Intel, IBM, UC Davis)

**2016**: NSAILTS Pattern of Life Graph Analytics

**2016**: OSD Decision Analytics

**2016**: Development and Release of GraphBLAS Template Library, v 1.0 (with Indiana U)

**2017-18**: GraphBLAS Template Library, v 2.0 (with CMU/PNNL)

**2018**: GraphBLAS book and hands-on tutorial

- **2018**: LENS: COTS Benchmark Baseline for Graph Analytics

Research & Development

- **2013**: Data-Intensive Computing

Program

- **2018-21**: Line: A Series of Unlikely Events: Learning behaviors in big data

- **2014**: NSA: Predictive Analytics Hands-on Workshop

- **2015**: Development and Release of GraphBLAS Template Library, v 1.0 (with Indiana U)

- **2016**: OSD Decision Analytics

- **2017**: GraphBLAS Template Library, v 2.0 (with CMU/PNNL)

- **2018**: GraphBLAS book and hands-on tutorial

Proof of Concept

- **2015**: Development and Release of GraphBLAS Template Library, v 1.0 (with Indiana U)

- **2016**: NSAILTS Pattern of Life Graph Analytics

- **2016**: OSD Decision Analytics
Automated Code Generation for High-Performance, Future-Compatible Graph Libraries

Problem:

• Heterogeneous high-performance computing (HHPC) architectures are becoming more complex (the NSCI push to exascale).
• Graph algorithms are difficult to program efficiently even on today’s hardware architectures.
• Exascale trend: Programming these systems will be much more difficult.\(^1\)

Solution:

• Create an automated code generation tool that produces high-performance graph algorithm implementations for specified hardware.
• Make graph algorithms performance-portable and future-compatible.

Approach:

• Create formal abstractions of graph algorithms and primitives (build on GraphBLAS).
• Extend formal abstractions of chosen hardware architectures (build on Spiral and DARPA HACMS, DESA, PERFECT, BRASS).
• Create tool for mapping graph algorithms to hardware architectures for efficient code generation of data-intensive applications.

\(^1\)FACT SHEET: National Strategic Computing Initiative, 29 July 2015.
Graph Analysis is *Important* and *Pervasive*

**ISR**
- Graphs represent entities and relationships detected through multi-INT sources
- 1,000s – 1,000,000s tracks and locations
- GOAL: Identify anomalous patterns of life

**Social**
- Graphs represent relationships between individuals or documents
- 10,000s – 10,000,000s individual and interactions
- GOAL: Identify hidden social networks

**Cyber**
- Graphs represent communication patterns of computers on a network
- 1,000,000s – 1,000,000,000s network events
- GOAL: Identify cyber attacks or malicious software

Common Goal: Detection of subtle patterns in massive graphs

Today’s Computing Landscape

**Intel Xeon E5-2699v3**
662 Gflop/s, 145 W
18 cores, 2.3 GHz
4-way/8-way AVX2

**IBM POWER8**
384 Gflop/s, 200 W
12 cores, 4 GHz
2-way/4-way VMX/VSX

**NVIDIA Tesla P100**
10.6 Tflop/s, 250 W
3584 cores, 1.48 GHz
64-way SIMT

**Intel Xeon Phi**
1.2 Tflop/s, 300 W
61 cores, 1.24 GHz
8-way/16-way LRBni

**Qualcomm Snapdragon 810**
10 Gflop/s, 2 W
4 cores, 2.5 GHz
A330 GPU, V50 DSP, NEON

**Intel Atom C2750**
29 Gflop/s, 20 W
8 cores, 2.4 GHz
2-way/4-way SSSE3

**Dell PowerEdge R920**
1.34 Tflop/s, 850 W
4x 15 cores, 2.8 GHz
4-way/8-way AVX

**IBM BlueGene/Q**
10 Pflop/s, 8 MW
48k x 16 cores, 1.6 GHz
4-way QPX

1 Gflop/s = one billion floating-point operations (additions or multiplications) per second

Separation of Concerns

Separate the complexity of graph analysis from the complexity of hardware systems:
### GraphBLAS Primitives

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxm, mxv, vxm</td>
<td>Perform matrix multiplication (e.g., breadth-first traversal)</td>
</tr>
<tr>
<td>eWiseAdd, eWiseMult</td>
<td>Element-wise addition and multiplication of matrices (e.g., graph union, intersection)</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>apply</td>
<td>Apply unary function to each element of matrix (e.g., edge weight modification)</td>
</tr>
<tr>
<td>reduce</td>
<td>Reduce 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>
<tr>
<td>build</td>
<td>Build a matrix representation from row, column, value tuples</td>
</tr>
<tr>
<td>extractTuples</td>
<td>Extract the row, column, value tuples from a matrix representation</td>
</tr>
</tbody>
</table>

Separation of Concerns

GOAL: write once, run everywhere…fast (with help from hardware experts).
What is Spiral?

Traditionally

Spiral Approach

High performance library optimized for given platform

Comparable performance

High performance library optimized for given platform
Spiral: Platform-Aware Formal Program Synthesis

GraphBLAS Math:
\[ C(L, z) = (L \oplus \otimes L^T) \]
\[ \text{count} = \bigoplus_{i,j} C(i,j) \]

Model: common abstraction
- \( \text{spaces of matching formulas} \)

Algorithm space
- search
- rewriting
- optimization
- Kernel: problem size, algorithm choice

Architecture space
- pick
- \( \nu \), \( p \), \( \mu \)
- \( \text{Architectural parameters: Vector length, } \# \text{processors, ...} \)

Abstracting spaces:
- \( L, z \)
- \( \bigoplus \), \( \otimes \), \( \oplus \)
- \( \text{vec}(\nu) \)
- \( \text{scal}(p, \mu) \)
### GraphBLAS Primitives: The Math

<table>
<thead>
<tr>
<th>Operation</th>
<th>Mathematical Description</th>
<th>Output</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxm</td>
<td>$C(\neg M, z) = C \odot (A^T \oplus \otimes B^T)$</td>
<td>$C$</td>
<td>$\neg, M, z, \odot, A, T, \oplus \otimes, B, T$</td>
</tr>
<tr>
<td>mxv, (v xm)</td>
<td>$c(\neg m, z) = c \odot (A^T \oplus \otimes b)$</td>
<td>$c$</td>
<td>$\neg, m, z, \odot, A, T, \oplus \otimes, b$</td>
</tr>
<tr>
<td>eWiseMult</td>
<td>$C(\neg M, z) = C \odot (A^T \otimes B^T)$</td>
<td>$C$</td>
<td>$\neg, M, z, \odot, A, T, \otimes, B, T$</td>
</tr>
<tr>
<td>eWiseAdd</td>
<td>$C(\neg M, z) = C \odot (A^T \otimes B^T)$</td>
<td>$C$</td>
<td>$\neg, M, z, \odot, A, T, \otimes, B, T$</td>
</tr>
<tr>
<td>reduce (row)</td>
<td>$c(\neg m, z) = c \odot [\oplus_j A^T(:,j)]$</td>
<td>$c$</td>
<td>$\neg, m, z, \odot, A, T, \oplus$</td>
</tr>
<tr>
<td>apply</td>
<td>$C(\neg M, z) = C \odot f(A^T)$</td>
<td>$C$</td>
<td>$\neg, M, z, \odot, A, T, f$</td>
</tr>
<tr>
<td>transpose</td>
<td>$C(\neg M, z) = C \odot A^T$</td>
<td>$C$</td>
<td>$\neg, M, z, \odot, A (T)$</td>
</tr>
<tr>
<td>extract</td>
<td>$C(\neg M, z) = C \odot A^T(i,j)$</td>
<td>$C$</td>
<td>$\neg, M, z, \odot, A, T, i, j$</td>
</tr>
<tr>
<td>assign</td>
<td>$C(\neg M, z)(i,j) = C(i,j) \odot A^T$</td>
<td>$C$</td>
<td>$\neg, M, z, \odot, A, T, i, j$</td>
</tr>
<tr>
<td>build (meth.)</td>
<td>$C = \mathcal{S}_{m \times n}(i,j,v,\odot)$</td>
<td>$C$</td>
<td>$\odot, m, n, i, j, v$</td>
</tr>
<tr>
<td>extractTuples (meth.)</td>
<td>$(i,j,v) = A$</td>
<td>$i,j,v$</td>
<td>$A$</td>
</tr>
</tbody>
</table>

Notation: i, j – index arrays, v – scalar array, m – 1D mask, other bold-lower – vector (column), M – 2D mask, other bold-caps – matrix, T – transpose, $\neg$ – structural complement, z – clear output, $\odot$ monoid/binary function, $\oplus \otimes$ semiring, blue – optional parameters, red – optional modifiers.
SPIRAL’s Math Framework

High Level Operators

\[ \langle \cdot, \cdot \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \]

\[ \sum_{i=0}^{n-1} x_i y_i \]

Basic Operators

Pointwise : \( \mathbb{R}^n \to \mathbb{R}^n \)

\[ f \Rightarrow f_0(x_0) \odot \cdots \odot f_n(x_n, 1) \]

Atomic : \( \mathbb{R} \times \mathbb{R} \to \mathbb{R} \)

\[ (x, y) \Rightarrow f(x, y) \]

Pointwise : \( \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n} \)

\[ (x_{ij}, y_{ij}) \Rightarrow f_0(x_0, y_0) \odot \cdots \odot f_n(x_n, 1, y_n, 1) \]

Reduction : \( \mathbb{R}^n \to \mathbb{R} \)

\[ (x_i) \Rightarrow f_u(x_n, 1, f_{n-1}(x_{n-1}, 2, f_{n-2}(x_2, 1), \ldots, f_0(x_0, id(0))) \ldots) \]

Loop Abstraction

\[ \bigcup_{i=0}^{n-1} : (D \to R)^n \to (D \to R) \]

\[ A_i \Rightarrow (x \mapsto A_0(x) \cup \cdots \cup A_{n-1}(x)) \]

Rule Based Compiler

\[ \text{Code} \left( y = (A \circ B)(x) \right) \Rightarrow \{ \text{decl}(t), \text{Code} \left( t = B(x) \right), \text{Code} \left( y = A(t) \right) \} \]

\[ \text{Code} \left( y = \left( \sum_{i=0}^{n-1} A_i \right)(x) \right) \Rightarrow \{ y := \bar{0}, \text{for}(i = 0..n-1) \text{ Code}(y+ = A_i(x)) \} \]

\[ \text{Code} \left( y = (e^n_i) \top(x) \right) \Rightarrow y[0] := x[i] \]

\[ \text{Code} \left( y = e^n_i(x) \right) \Rightarrow \{ y = \bar{0}, y[i] := x[0] \} \]

\[ \text{Code} \left( y = \text{Atomic}_f(x) \right) \Rightarrow y[0] := f(x[i]) \]

Leverages DARPA HACMS
Autotuning in Constraint Solution Space

Intel Core i7 (2nd Gen)

Base cases

Transformation rules

TriangleCount

Breakdown rules

# triangles = || |L ⊗ (L ⊕ L^T)| ||_1

Expansion + backtracking

OL specification

OL (dataflow) expression

Σ-OL (loop) expression

Optimized Σ-OL expression

Abstract code

Optimized abstract code

C code

Recursive descent

Confluent term rewriting

Recursive descent

Confluent term rewriting

Recursive descent

OL (dataflow) expression

Abstract code

Optimized abstract code

C code

Recursive descent

Confluent term rewriting

Recursive descent

OL specification

Expanding and backtracking

OL (dataflow) expression

Optimized Σ-OL expression

Abstract code

Optimized abstract code

C code
Long-Range Goal: SPIRAL as JIT and GraphBLAS Optimizer

Source Code
- C++, GraphBLAS calls, other supported libraries
- Code = specification, not program

SPIRAL Module
- Acts as JIT, delayed execution engine, Inspector/executor
- Implements telescoping language ideas
- Rewrites code into better algorithms
- Compiles to range of platforms CPU, GPU, FPGA
- Plug-in mechanism for post deployment reconfiguration and update

Leverages DARPA BRASS
Formal Approach To Co-Optimization

GraphBLAS Math:
\[ C(L, z) = (L \oplus \otimes L^T) \]
\[ \text{count} = \bigoplus_{i,j} C(i,j) \]

Model: common abstraction
= spaces of matching formulas

Co-optimization

"clean slate"

Leverages DARPA DESA and PERFECT
Long Range Goal: Algorithm/Architecture Co-Optimization

Design Space Optimization Problem

- **Algorithm**
- **Architecture**
- **Cost function** $C_m(\xi, \mu)$

**Parameters:** $\xi, \mu$

**Metric $m$:** power, runtime, …

**Task:** Find $\xi$ and $\mu$ s.t. $C_m(\xi, \mu)$ is minimal

$$(\hat{A}, \hat{M}) = \arg\min_{\theta, \xi} C_m(\mathcal{A}(\theta), \mathcal{M}(\xi))$$

“What is the right architecture for my application?”
“What architecture features are good for my application?”
Summary and Future Work

• GraphBLAS C API Specification is complete
  - Mathematical descriptions of primitive operations are complete
  - Algorithm development using the API is in progress (dozens completed)
• Exploration of performant code generation and data structures has begun
• Goals for FY18:
  - Integrate primitives and necessary “knowledge” into Spiral code generation technology
  - Target different hardware platforms
    • Multi-core CPUs
    • Accelerators: FPGAs, Graphics Processing Units (GPUs)
  - Generate code for algorithms in benchmark and Challenge problems
    • Graph 500: breadth-first search, shortest paths
    • DARPA HIVE Graph Challenge: subgraph isomorphism
• Long-Range Goal: Co-synthesis of hardware and software
Contact Information

Presenter / SEI PI
Dr. Scott McMillan
Senior Research Scientist
Email: smcmillan@sei.cmu.edu

Presenter / CMU PI
Prof. Franz Franchetti
ECE Department
Email: franzf@ece.cmu.edu