Research Review 2018

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

SEI PI: **Dr. Scott McMillan**, Senior Research Scientist

CMU PI's: **Prof. Franz Franchetti**, ECE

Prof. James C. Hoe, ECE

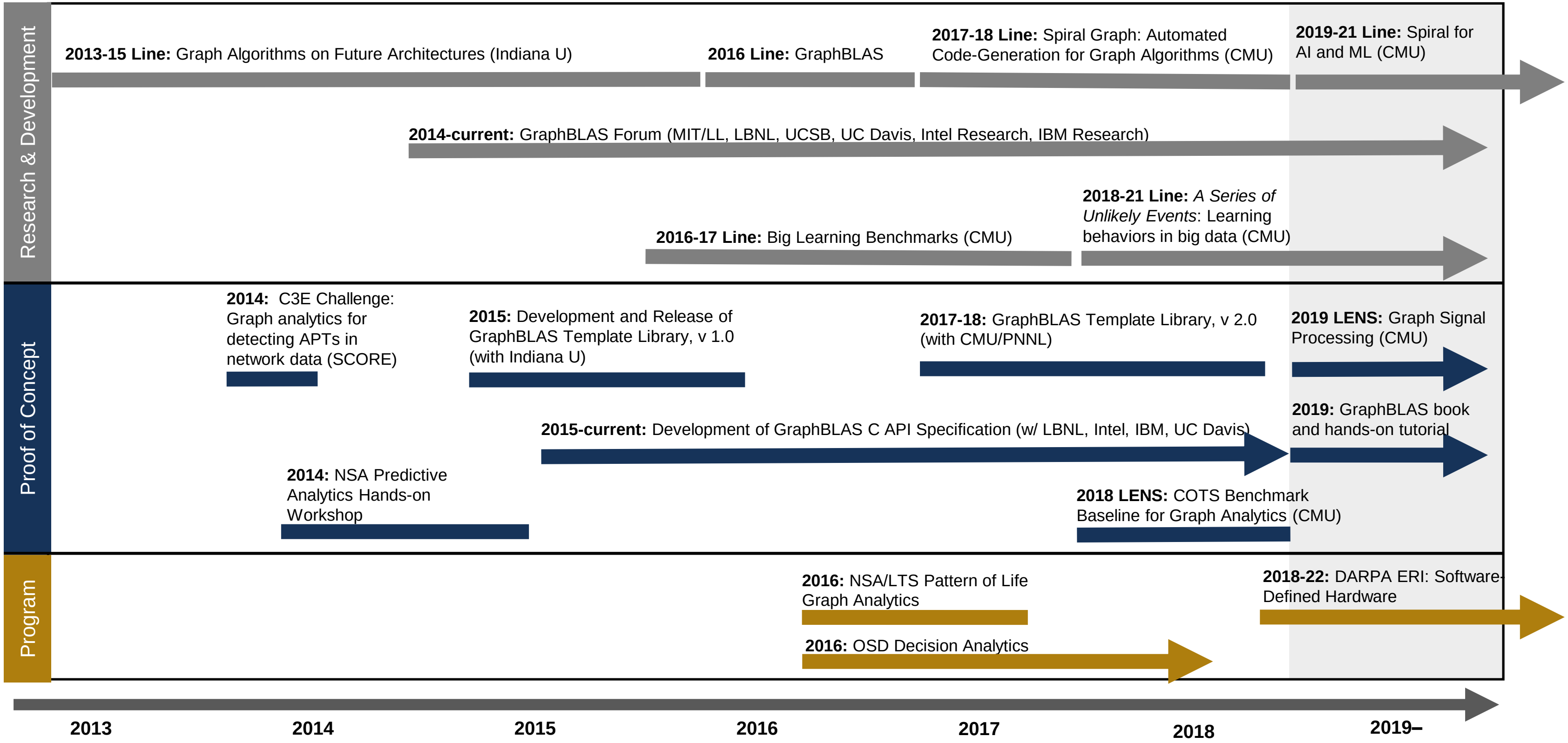
Prof. Tze Meng Low, ECE

Legal

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.

DM18-1167

Data-Intensive Computing Efforts at the SEI



SpiralGraph: Automated Code Generation for *Future-Compatible*, High-Performance 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.**

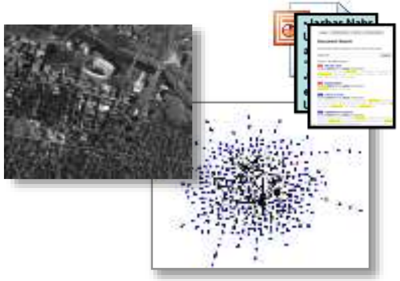
Approach:

- 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.*

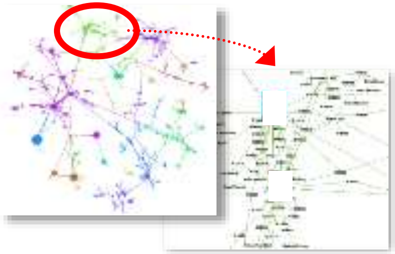
Graph Analysis Is *Important* and *Pervasive*

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

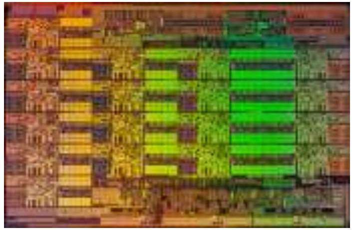
Cyber

- 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

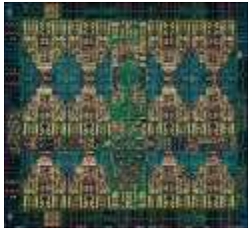
Common Goal: Detection of subtle patterns in massive graphs

Slide credit: Jeremy Kepner, et al. "Mathematical Foundations of the GraphBLAS", IEEE HPEC, Sept. 2016.

Today's Computing Landscape



Intel Xeon 8180M
2.25 Tflop/s, 205 W
28 cores, 2.5—3.8 GHz
2-way—16-way AVX-512



IBM POWER9
768 Gflop/s, 300 W
24 cores, 4 GHz
4-way VSX-3

Nvidia Tesla V100
7.8 Tflop/s, 300 W
5120 cores, 1.2 GHz
32-way SIMT

Intel Xeon Phi 7290F
1.7 Tflop/s, 260 W
72 cores, 1.5 GHz
8-way/16-way LRBni

Snapdragon 835
15 Gflop/s, 2 W
8 cores, 2.3 GHz
A540 GPU, 682 DSP, NEON

Intel Atom C3858
32 Gflop/s, 25 W
16 cores, 2.0 GHz
2-way/4-way SSSE3

Dell PowerEdge R940
3.2 Tflop/s, 6 TB, 850 W
4x 24 cores, 2.1 GHz
4-way/8-way AVX

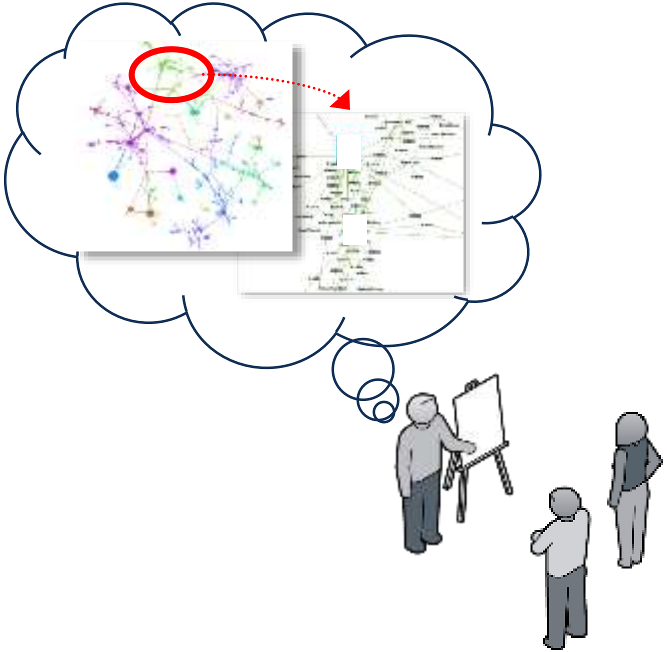
Summit
187.7 Pflop/s, 13 MW
9,216 x 22 cores POWER9
+ 27,648 V100 GPUs

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

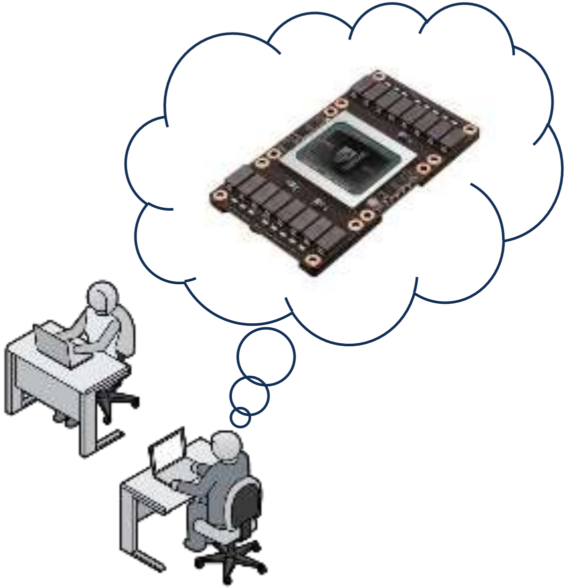
Slide credit: Franz Franchetti, "18-847G, 2018, Lecture 1: How Big is Big?"

Separation of Concerns

Separate the complexity of graph analysis from the complexity of hardware systems:

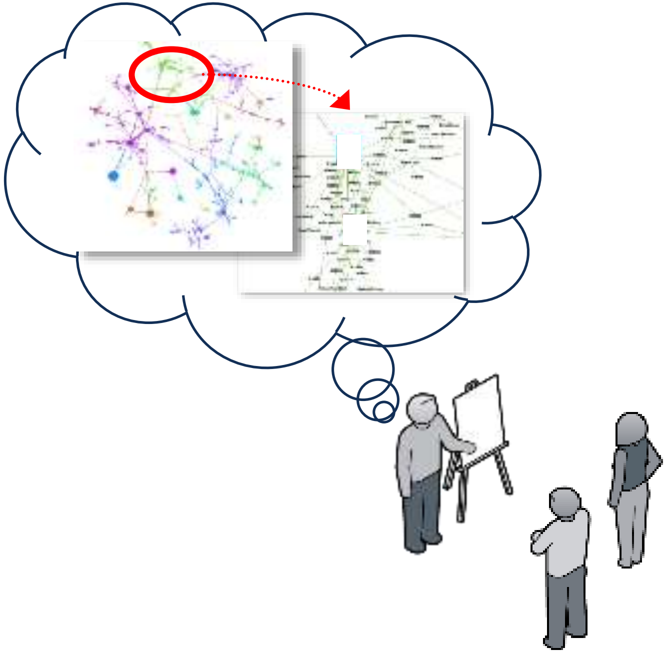


Separation of Concerns

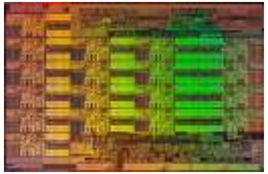


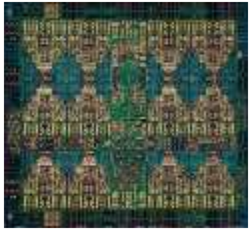
Separation of Concerns

GOAL: write once, run everywhere…fast (**with** help from hardware experts).



GraphBLAS Application Programming Interface (API)





GraphBLAS Primitives

Operation	Description
mxm, mxv, vxm	Perform matrix multiplication (e.g., breadth-first traversal)
eWiseAdd, eWiseMult	Element-wise addition and multiplication of matrices (e.g., graph union, intersection)
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)
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)
build	Build a matrix representation from row, column, value tuples
extractTuples	Extract the row, column, value tuples from a matrix representation

http://graphblas.org "A. Buluc, T. Mattson, S. McMillan, J. Moreira, C. Yang, "The GraphBLAS C API Specification, v 1.0.0," May 2017, updated May 2018.

GraphBLAS Ecosystem: One Year Later

GraphBLAS Forum: https://graphblas.org

GraphBLAS C API, v. 1.2.0

SuiteSparse

ATM

IBM-GraphBLAS

GraphBLAS Ecosystem: One Year Later

GraphBLAS Forum: https://graphblas.org

GraphBLAS C API, v. 1.2.0
GraphBLAS C++ API (proposal)

SuiteSparse

ATM

IBM-GraphBLAS

gbtl

Carnegie Mellon University

Pacific Northwest National Laboratory

Proudly Operated by Battelle Since 1965

GraphBLAS Ecosystem: One Year Later

GraphBLAS Forum: https://graphblas.org

GraphBLAS C API, v. 1.2.0
GraphBLAS C++ API (proposal)

SuiteSparse

IBM-GraphBLAS

gbtl

Carnegie Mellon University

Pacific Northwest NATIONAL LABORATORY

gpu-GraphBLAS

UC DAVIS UNIVERSITY OF CALIFORNIA

GraphBLAS on EMU

Carnegie Mellon University

UMBC

GraphBLAS Ecosystem: One Year Later

GraphBLAS Forum: https://graphblas.org

Carnegie Mellon University GraphBLAS Test Framework

GraphBLAS C API, v. 1.2.0
GraphBLAS C++ API (proposal)

SuiteSparse

ATM

IBM-GraphBLAS

gbtl

Carnegie Mellon University

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

gpu-GraphBLAS

UC DAVIS
UNIVERSITY OF CALIFORNIA

GraphBLAS on EMU

Carnegie Mellon University

UMBC

GraphBLAS Ecosystem: One Year Later

GraphBLAS Forum: https://graphblas.org

Carnegie Mellon University **GraphBLAS Test Framework**

pyGB

Python Wrapper around gbtl

Carnegie Mellon University W

GraphBLAS C API, v. 1.2.0

GraphBLAS C++ API (proposal)

SuiteSparse ATM

IBM-GraphBLAS

gbtl Carnegie Mellon University Pacific Northwest NATIONAL LABORATORY Proudly Operated by Battelle Since 1965

gpu-GraphBLAS

UCDAVIS UNIVERSITY OF CALIFORNIA

GraphBLAS on EMU

Carnegie Mellon University UMBC

GraphBLAS Ecosystem: One Year Later

GraphBLAS Forum: https://graphblas.org

gbtl
Algorithms Repository
Carnegie Mellon University

Carnegie Mellon University **GraphBLAS Test Framework**

redisgraph
redislabs

pyGB
Python Wrapper around gbtl
Carnegie Mellon University W

GraphBLAS C API, v. 1.2.0
GraphBLAS C++ API (proposal)

SuiteSparse ATM

IBM-GraphBLAS

gbtl Carnegie Mellon University Pacific Northwest NATIONAL LABORATORY Proudly Operated by Battelle Since 1965

gpu-GraphBLAS
UCDAVIS UNIVERSITY OF CALIFORNIA

GraphBLAS on EMU
Carnegie Mellon University UMBC

GraphBLAS Ecosystem: One Year Later

GraphBLAS Forum: https://graphblas.org

Algorithms Repository

Carnegie Mellon University

Carnegie Mellon University GraphBLAS Test Framework

redisgraph

redislabs

pyGB

Python Wrapper around gbtl

Carnegie Mellon University

GraphBLAS C API, v. 1.2.0

GraphBLAS C++ API (proposal)

SuiteSparse

IBM-GraphBLAS

gpu-GraphBLAS

UC DAVIS UNIVERSITY OF CALIFORNIA

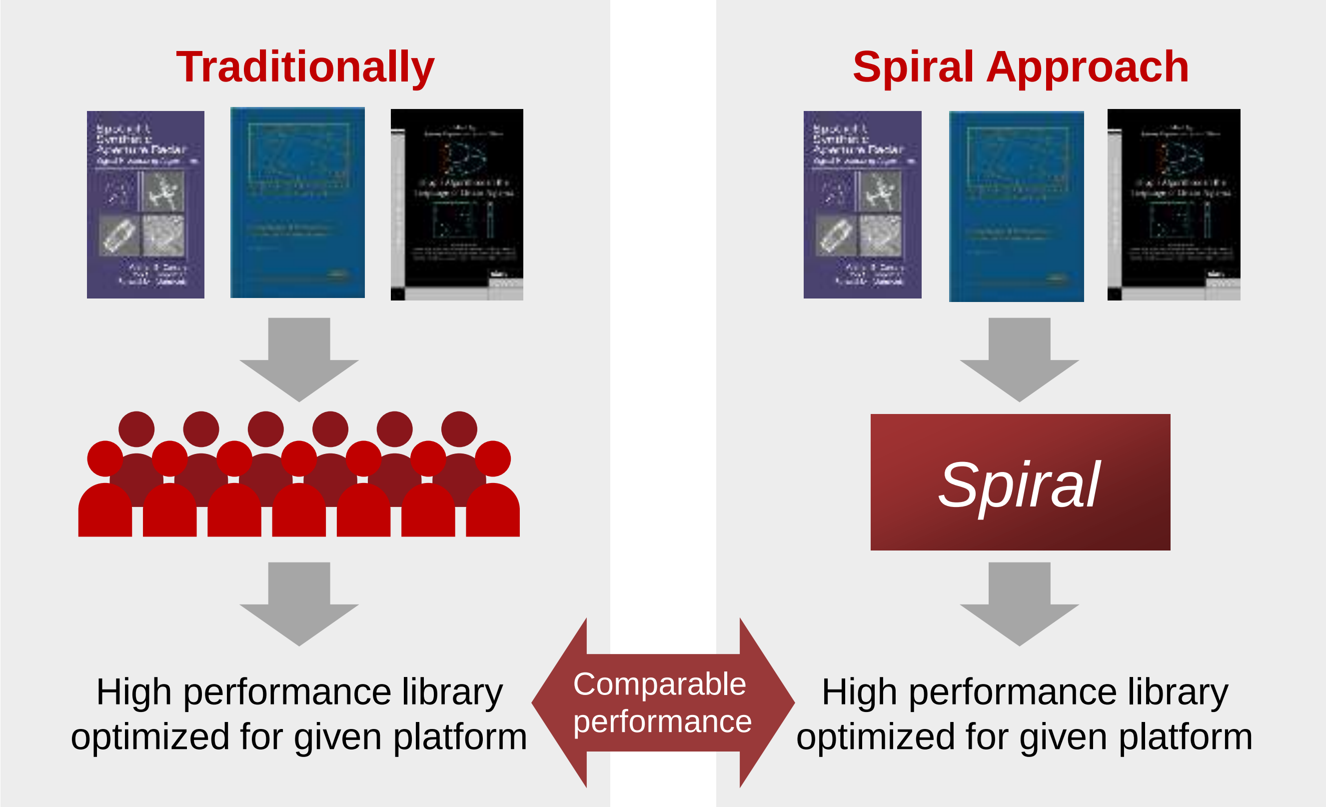
GraphBLAS on EMU

Carnegie Mellon University

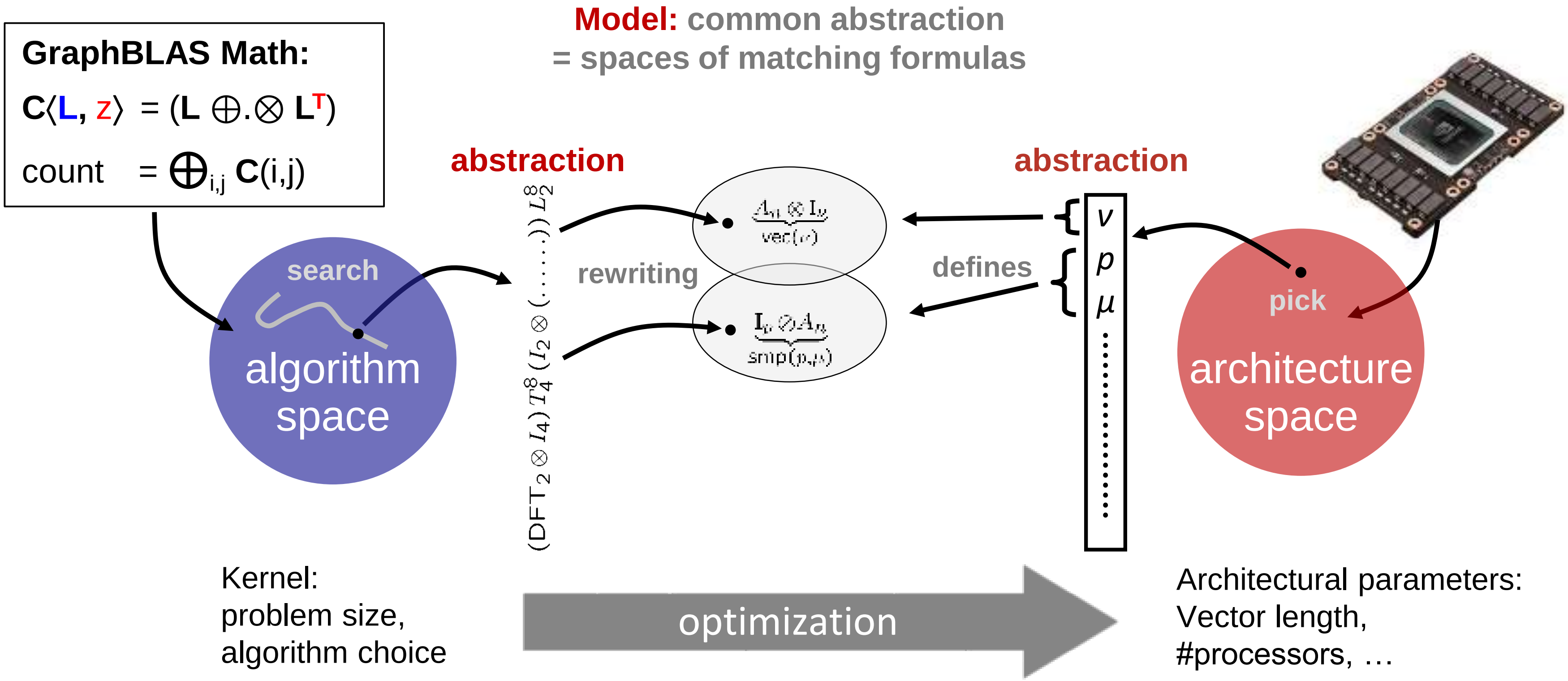
UMBC

Optimizing this is still difficult, time-consuming, and costly.

What is Spiral?



Spiral: Platform-Aware Formal Program Synthesis



GraphBLAS Primitives: The Math

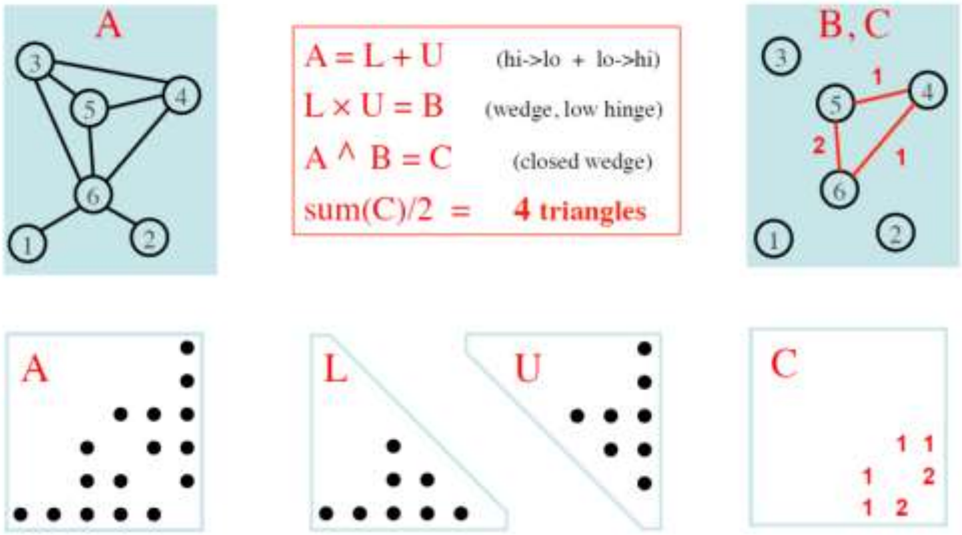
Operation	Mathematical Description	Output	Inputs
mxm	$\mathbf{C}\langle \neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^{\mathrm{T}} \oplus.\otimes \mathbf{B}^{\mathrm{T}})$	**C**	¬, **M**, z, ⊙, **A**, **T**, ⊕.⊗, **B**, **T**
mxv, (vxm)	$\mathbf{c}\langle \neg\mathbf{m}, z\rangle = \mathbf{c} \odot (\mathbf{A}^{\mathrm{T}} \oplus.\otimes \mathbf{b})$	**c**	¬, **m**, z, ⊙, **A**, **T**, ⊕.⊗, **b**
eWiseMult	$\mathbf{C}\langle \neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^{\mathrm{T}} \otimes \mathbf{B}^{\mathrm{T}})$	**C**	¬, **M**, z, ⊙, **A**, **T**, ⊗, **B**, **T**
eWiseAdd	$\mathbf{C}\langle \neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^{\mathrm{T}} \oplus \mathbf{B}^{\mathrm{T}})$	**C**	¬, **M**, z, ⊙, **A**, **T**, ⊕, **B**, **T**
reduce (row)	$\mathbf{c}\langle \neg\mathbf{m}, z\rangle = \mathbf{c} \odot [\oplus_j \mathbf{A}^{\mathrm{T}}(:,j)]$	**c**	¬, **m**, z, ⊙, **A**, **T**, ⊕
apply	$\mathbf{C}\langle \neg\mathbf{M}, z\rangle = \mathbf{C} \odot f(\mathbf{A}^{\mathrm{T}})$	**C**	¬, **M**, z, ⊙, **A**, **T**, *f*
transpose	$\mathbf{C}\langle \neg\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A}^{\mathrm{T}}$	**C**	¬, **M**, z, ⊙, **A (T)**
extract	$\mathbf{C}\langle \neg\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A}^{\mathrm{T}}(\mathbf{i},\mathbf{j})$	**C**	¬, **M**, z, ⊙, **A**, **T**, **i**, **j**
assign	$\mathbf{C}\langle \neg\mathbf{M}, z\rangle (\mathbf{i},\mathbf{j}) = \mathbf{C}(\mathbf{i},\mathbf{j}) \odot \mathbf{A}^{\mathrm{T}}$	**C**	¬, **M**, z, ⊙, **A**, **T**, **i**, **j**
build (meth.)	$\mathbf{C} = \mathbb{S}^{mxn}(\mathbf{i},\mathbf{j},\mathbf{v},\odot)$	**C**	⊙, m, n, **i**, **j**, **v**
extractTuples (meth.)	$(\mathbf{i},\mathbf{j},\mathbf{v}) = \mathbf{A}$	**i,j,v**	**A**

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, **¬** - structural complement, z – clear output, ⊕ monoid/binary function, ⊕.⊗ semiring, **blue** – optional parameters, **red** – optional modifiers

S. McMillan, et al., “Design and Implementation of the GraphBLAS Template Library (GBTL),” SIAM Annual Meeting (AN16), July 2016.
Updated IEEE HPEC Conference, Sep 2017.

Spiral: Platform-Aware Formal Program Synthesis

$$\#\Delta = \frac{1}{6}\, tr(\mathbf{A}^3)$$
$$= \|\mathbf{L} \,.\!*\, (\mathbf{L} * \mathbf{L}^T)\|_1$$



$$\mathbf{C}\langle \mathbf{L}, z\rangle = (\mathbf{L} \oplus . \otimes \mathbf{L}^T)$$
$$\#\Delta = \bigoplus_{i,j} \mathbf{C}(i,j)$$

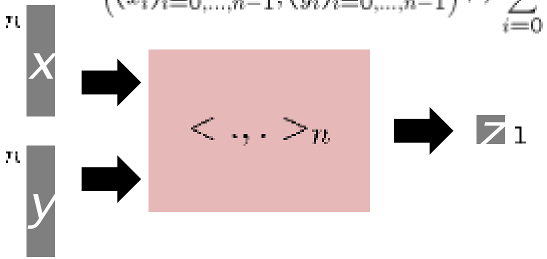
```
int triangle_count(Matrix const &L)
{
  Matrix C(L.nrows(), L.ncols());
  mxm(C, L, NoAccumulate(), ArithmeticSemiring<int>(),
      L, transpose(L));

  int count = 0;
  reduce(count, NoAccumulate(), PlusMonoid<int>(), C);
  return count;
}
```

SPIRAL's Math Framework

High Level Operators

$< .,. >_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$

$$\left((x_i)_{i=0,\dots,n-1}, (y_i)_{i=0,\dots,n-1}\right) \mapsto \sum_{i=0}^{n-1} x_i y_i$$



Basic Operators

$\text{Pointwise}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}^n$

$$(x_i)_i \mapsto f_0(x_0) \oplus \dots \oplus f_{n-1}(x_{n-1})$$

$\text{Atomic}_{f(.,.)} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

$$(x, y) \mapsto f(x, y)$$

$\text{Pointwise}_{n \times n, f_i} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$

$$\left((x_i)_i, (y_i)_i\right) \mapsto f_0(x_0, y_0) \oplus \dots \oplus f_{n-1}(x_{n-1}, y_{n-1})$$

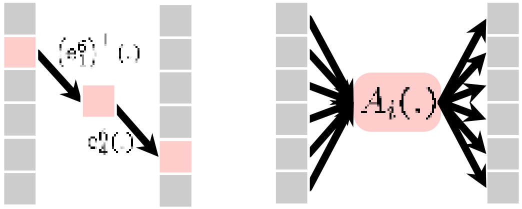
$\text{Reduction}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}$

$$(x_i)_i \mapsto f_{n-1}(x_{n-1}, f_{n-2}(x_{n-2}, f_{n-3}(\dots f_0(x_0, \text{id}())\dots)$$

Loop Abstraction

$$\coprod_{i=0}^{n-1} : (D \to R)^n \to (D \to R)$$

$$A_i \mapsto (x \mapsto A_0(x) \sqcup \dots \sqcup A_{n-1}(x))$$



Rule Based Compiler

$$\text{Code}\left(y = (A \circ B)(x)\right) \to \left\{\texttt{decl}(t), \text{Code}\left(t = B(x)\right), \text{Code}\left(y = A(t)\right)\right\}$$

$$\text{Code}\left(y = \left(\sum_{i=0}^{n-1} A_i\right)(x)\right) \to \left\{y := \vec{0}, \texttt{for}(i = 0..n-1)\ \text{Code}\left(y += A_i(x)\right)\right\}$$

$$\text{Code}\left(y = (e_i^n)^\top(x)\right) \to y[0] := x[i]$$

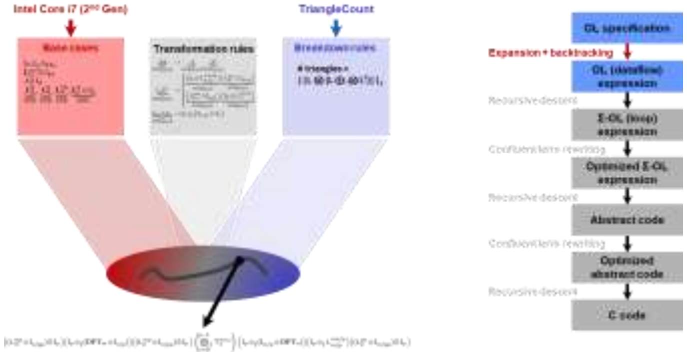
$$\text{Code}\left(y = e_i^n(x)\right) \to \left\{y = \vec{0}, y[i] := x[0]\right\}$$

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

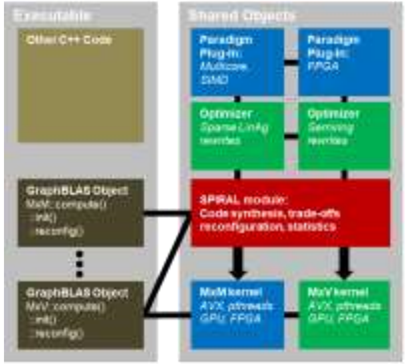
Leverages DARPA HACMS

SPIRAL Internals: Autotuning and Code Generation

Autotuning in Constraint Space



SPIRAL as JIT and GraphBLAS Optimizer



Source Code

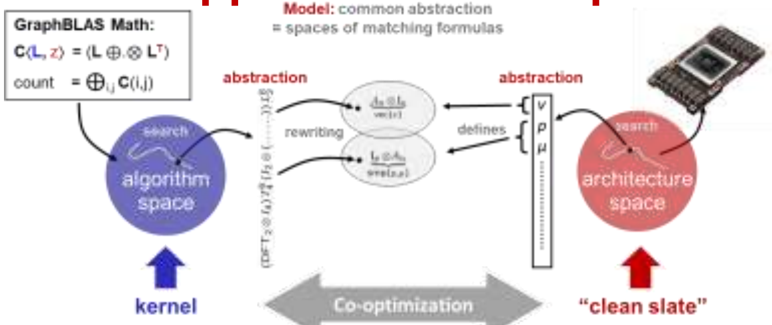
- C++, GraphBLAS calls, other supported libraries
- Code = specification, not program

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

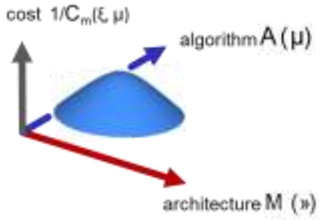
Formal Approach To Co-Optimization



Leverages DARPA DESA and PERFECT

Algorithm/Architecture Co-Optimization

Design Space



"What is the right architecture for my application?"
"What architecture features are good for my application?"

Optimization Problem

$$(\hat{A}, \hat{M}) = \text{argmin}_{\theta,\xi} C_m(A(\theta), M(\xi))$$

- Algorithm
- Architecture
- Cost function $C_m(\xi, \mu)$
- Parameters: ξ, μ
- Metric m: power, runtime,…

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

Graph Algorithms in Spiral

sr:	Arithmetic semiring
X:	Input matrix in CSR or CSC format
X.N:	Number of vertices in the graph
Accum:	Accumulation/Reduction function
Accum_X:	Accumulation over an input range
Dot:	Dot product

Problem Specification:

TriangleCount()

Algorithm Choice:

Accum_VMV(TriangleCount())

$$\Delta = \Delta + \tfrac{1}{2}\alpha_{10}A_{00}\alpha_{01}$$

Algorithm Derivation:

```
BB(
 Accum(i4, 1, X.N-1,
  Accum_X(i6, [ i4, 0 ], i4,
   Dot([ i6, add(i4, V(1)) ], [ i4, add(i4, V(1)) ],
              sub(sub(X.N, i4), V(1)))
)))
```

Abstract Code:

```
program(
 func(TVoid, "transform", [ res, IJ ],
     decl([ i6, j131, j1765, j1m31, j231, j2m31, jm32, rf63, rf64 ],
       chain(
         assign(deref(res), V(0)),
         loopf(i4, 1, 262110,
           chain(
             assign(rf63, V(0)),
             assign(j1765, add(V(262112), IJ, nth(IJ, i4))),
             assign(jm32, add(V(262112), IJ, nth(IJ, add(i4, V(1))))),
             loopw(logic_and(lt(j1765, jm32), lt(deref(j1765), V(0))),
               assign(j1765, add(j1765, V(1)))
             ),
             loopw(logic_and(lt(j1765, jm32), lt(deref(j1765), i4)),
               …
               // dot product
               …
             ),
             assign(deref(res), add(deref(res), rf63))
))))))
```

C Code:

```
void tc(int *res, int *IJ) {
  for(...) {
    // VMV product
  }
}
```

It Works…

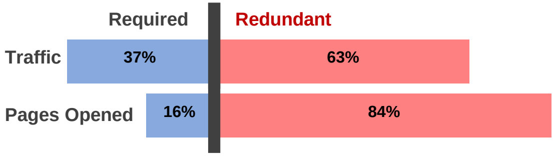
```
spiral> t := TriangleCount();
TriangleCount()
spiral> rt := RandomRuleTree(t, opts);
Accum_VMV_FLAME2( TriangleCount() )
spiral> srt := SumsRuleTree(rt, opts);
BB(
  Accum(i1, 1, 262110,
    Accum_X(i3, [ i1, 0 ], i1,
      Dot([ i3, add(i1, V(1)) ], [ i1, add(i1, V(1)) ], sub(sub(V(262111), i1), V(1)))
    )
  )
)
spiral> cs := CodeSums(srt, opts);
program(
   chain(
      func(TVoid, "init", [  ],
         chain()
      ),
      func(TVoid, "transform", [ res, IJ ],
         decl([ i3, j1, j11, j1m1, j21, j2m1, jm1, rf1, rf2 ],
            chain(
               assign(deref(res), V(0.0)),
               loopf(i1, 1, 262110,
                  chain(
                     assign(rf1, V(0.0)),
                     assign(j1, add(V(262112), IJ, nth(IJ, i1))),
                     assign(jm1, add(V(262112), IJ, nth(IJ, add(i1, V(1))))),
                     loopw(logic_and(lt(j1, jm1), lt(deref(j1), V(0))),
                        assign(j1, add(j1, V(1)))
                     ),
                     loopw(logic_and(lt(j1, jm1), lt(deref(j1), i1)),
                        chain(
                           assign(i3, deref(j1)),
                           assign(rf2, V(0.0)),
                           assign(j11, add(V(262112), IJ, nth(IJ, i3))),
                           assign(j1m1, add(V(262112), IJ, nth(IJ, add(i3, V(1))))),
                           assign(j21, add(V(262112), IJ, nth(IJ, i1))),
                           assign(j2m1, add(V(262112), IJ, nth(IJ, add(i1, V(1))))),
                           loopw(logic_and(lt(j11, j1m1), lt(deref(j11), add(i1, V(1)))),
                              assign(j11, add(j11, V(1)))
```

```
spiral> PrintCode("tc", cs, opts);

/*
 * This code was generated by Spiral 1.4.0, www.spiralgen.com
 * Copyright (c) 2017, SpiralGen, Inc.
 * All rights reserved.
 *
 * This source code is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This source code is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * For an alternate commercial license contact info@spiralgen.com.
 *
 */

#include <include/omega64.h>

void init_tc() {
}

void tc(int  *res, int  *IJ) {
    int  *j1, *j11, *j1m1, *j21, *j2m1, *jm1;
    int i3, rf1, rf2;
    *(res) = 0.0;
    for (int i1 = 1; i1 < 262110; i1++) {
        rf1 = 0.0;
        j1 = (262112 + IJ + IJ[i1]);
        jm1 = (262112 + IJ + IJ[(i1 + 1)]);
        while (((((j1 < jm1))) && (((*(j1) < 0))))) {
            j1 = (j1 + 1);
        }
        while (((((j1 < jm1))) && (((*(j1) < i1))))) {
            i3 = *(j1);
            rf2 = 0.0;
            j11 = (262112 + IJ + IJ[i3]);
            j1m1 = (262112 + IJ + IJ[(i3 + 1)]);
```

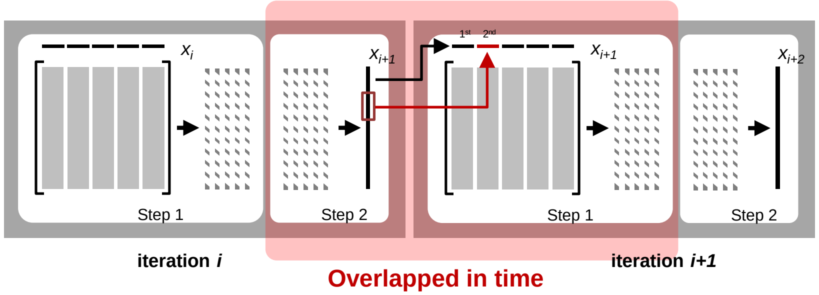
PageRank Acceleration: Comparison against GPU

PageRank

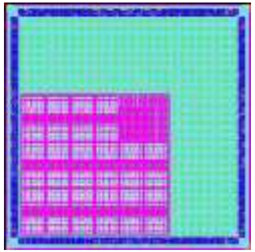
Memory Bound SpMV



Two-Step SpMV and Iteration Overlap



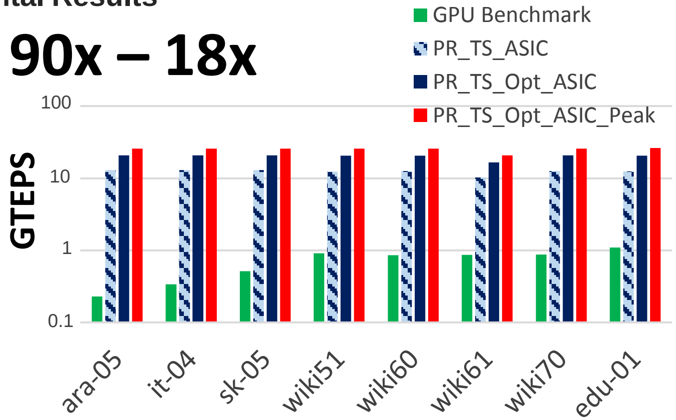
Custom Hardware Platform



16nm FinFET ASIC **Stratix10 FPGA**

Experimental Results

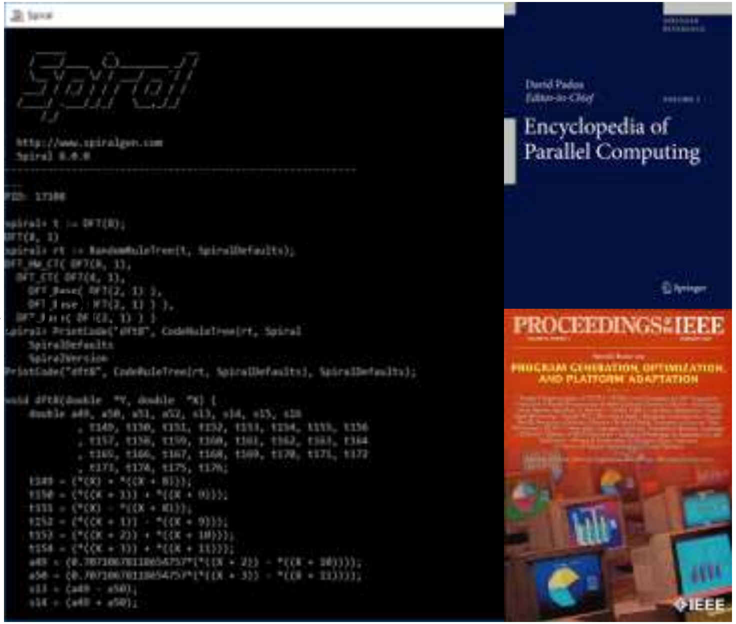
90x – 18x



Open Source Spiral: CMU/ECE and SEI Partnership

- Open Source SPIRAL available
  - non-viral license (BSD)
  - Initial version, effort ongoing to open source whole system
  - Commercial support via SpiralGen, Inc.
- Developed over 20 years
  - Funding: DARPA (OPAL, DESA, HACMS, PERFECT, BRASS), NSF, ONR, DoD HPC DOE, CMU SEI, Intel, Nvidia, Mercury
- Open sourced under DARPA PERFECT
- Ongoing Partnership between SEI and ECE

www.spiral.net



Summary and Future Work

- GraphBLAS C API Specification is complete (and no longer provisional)
  - Two conformant implementations in C (SuiteSparse and IBM)
  - C++ API proposed with a complete implementation
  - Python bindings under development
  - Algorithm development using the API continues (30+ completed)
- Development of performant code generation and data structures continues

- Goals for FY19 and beyond:
  - Expand to other data-intensive domains: **machine learning and AI**
  - Co-design targeted hardware platforms
    - Reconfigurable hardware: FPGAs, DARPA HIVE/SDH hardware
    - Incorporate resource constraints: cost, size, weight and power (CSWAP)

- 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