RESEARCH REVIEW 2019

Spiral AI/ML: Co-optimization for High-Performance, Data-Intensive Computing in Resource Constrained Environments

SEI PI: **Dr. Scott McMillan**, Senior Research Scientist
CMU PI's: **Professor Franz Franchetti**, ECE
Professor Tze Meng Low, ECE
Professor James C. Hoe, ECE

DM19-1113

Team


Dr. Scott McMillan, **SEI, PI**


Prof. Franz Franchetti


Prof. Tze Meng Low


Prof. James C. Hoe

CMU, co-PI's


Dr. John Wohlbier


Dr. Jason Larkin


Elliott Binder


Mark Blanco


Paul Brouwer


Anuva Kulkarni


Dr. Fazle Sadi


Dr. Daniele Spampinato


Courtney Rankin


Sandra Sanjeev


Peter Oostema

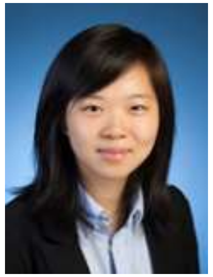

George Ralph*


Upasana Sridhar*


Arvind Srinivasan*

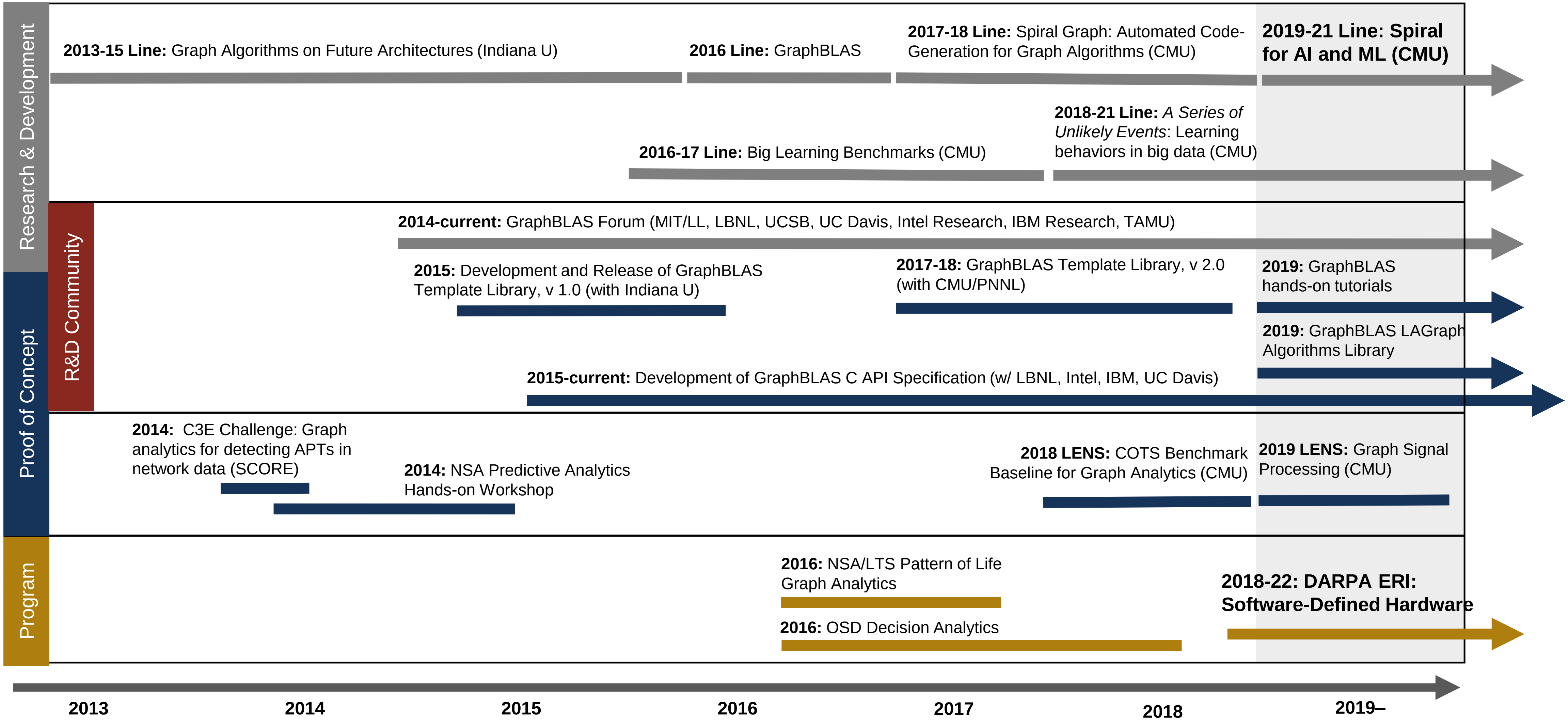

Guanglin Xu


Vadim Zaliva

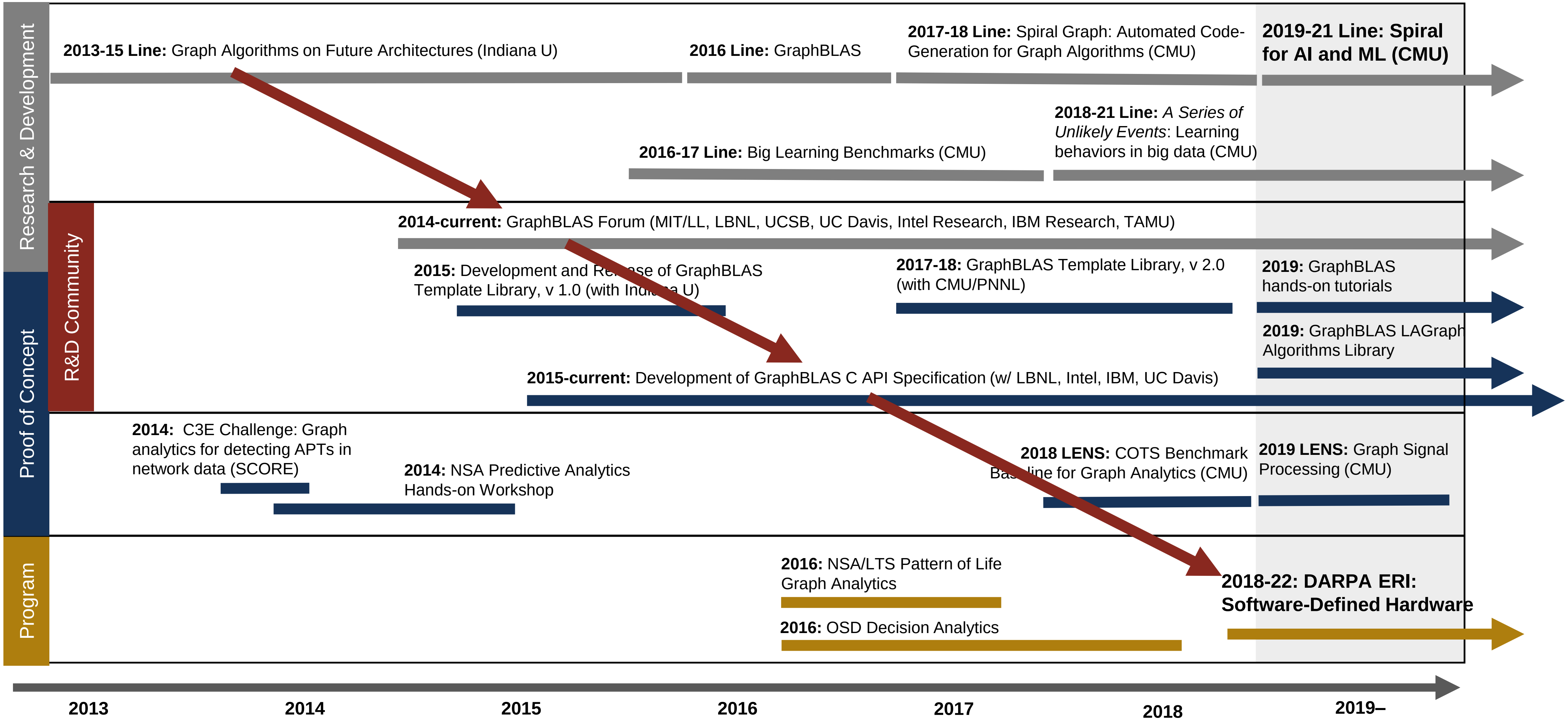

Jiyuan Zhang

* unfunded collaborators

Advanced Computing Efforts: Graphs to AI



Advanced Computing Efforts: Graphs to AI



Spiral/AIML: Co-optimization for High-Performance, Data-Intensive Computing in Resource Constrained Environments

“Rapidly delivering artificial intelligence to a combat zone won’t be easy.” Col. Drew Cukor, USMC.

Problem(s)

- Increasing complexity in computing architectures.
- Mission cost, size, weight, and power (CSWAP) constraints drive increasing use of FPGAs and ASICs (more complexity).
- Achieving performance from these platforms is hard.
- Achieving performance from data-intensive applications (graphs, ML, AI) is hard.

Solution

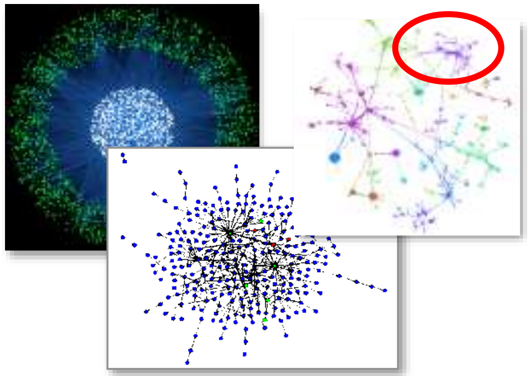
- Automatic code generation for data-intensive computations.
- Simultaneous, automatic co-optimization of hardware within CSWAP constraints.

Approach

- Identify common AI/ML/Graph computational primitives.
- Encode knowledge about graph, ML, and AI computational primitives into Spiral code-gen technology.
- Develop hardware performance models allowing Spiral to choose between components satisfying CSWAP requirements.

Data Intensive Computing Is *Important* and *Pervasive*

Graphs

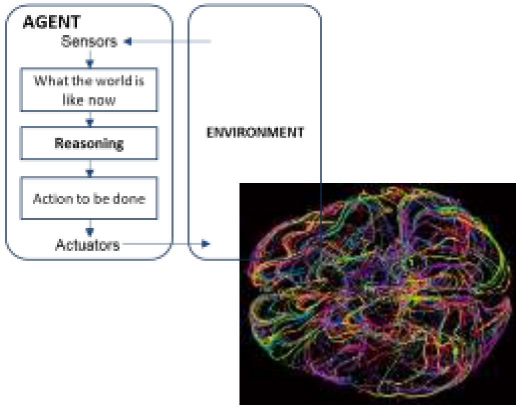


- Graphs represent entities and relationships detected through multi-INT sources.
- 1,000s – 100,000,000,000s tracks, interactions, events
- GOAL: Find clusters of similar entities or behaviors of interest.

Machine Learning

- Machine Learning encompasses techniques that exploit patterns captured in data for a given task.
- 10,000s – 1,000,000s labeled and unlabeled data
- GOAL: Provide accurate, data-driven predictions, insights, or models of observed phenomenon.

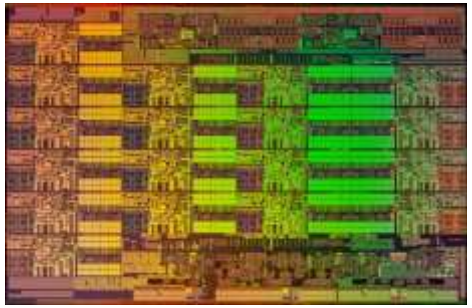
Artificial Intelligence



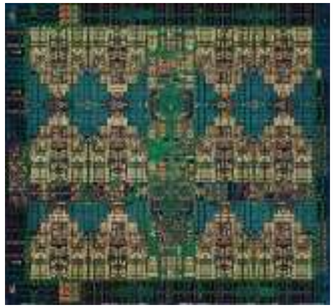
- AI is the collection of computations that make it possible to perceive, reason and act.
- Practically infinite state space and non-deterministic dynamics
- GOAL: Create intelligent agents that achieve target performance in deployed environments.

Common Goal: Timely, accurate, and actionable transformation of **massive amounts of** data into knowledge

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

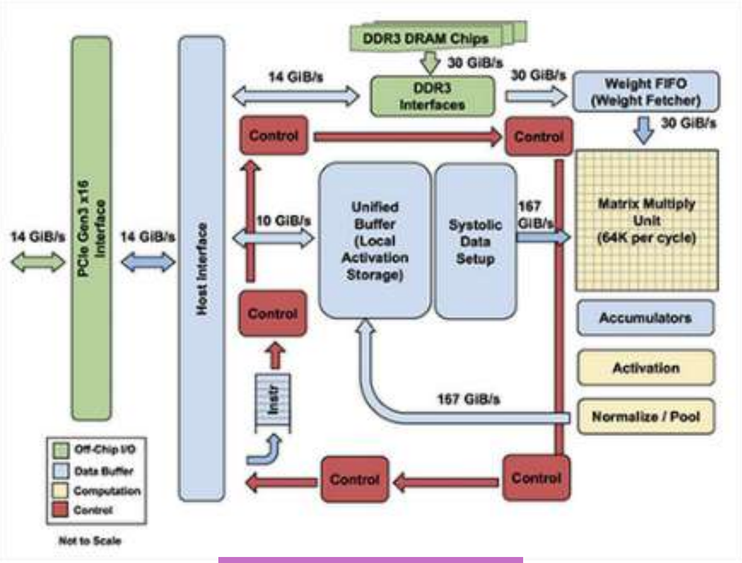
Dell PowerEdge R940
3.2 Tflop/s, 6 TB, 850 W
4x 24 cores, 2.1 GHz
4-way/8-way AVX

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

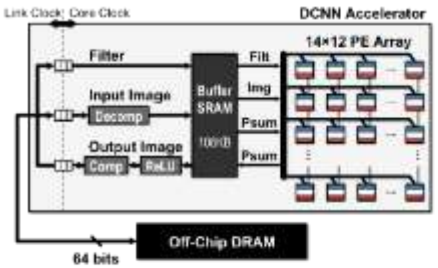
1 Gflop/s = one billion floating-point operations (additions or multiplications) per second

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

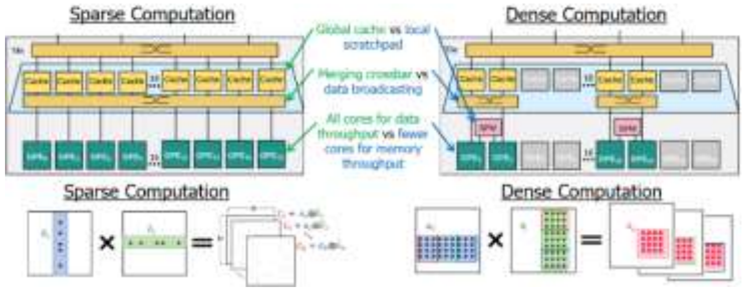
Today’s Computing Landscape…is not tomorrow’s.



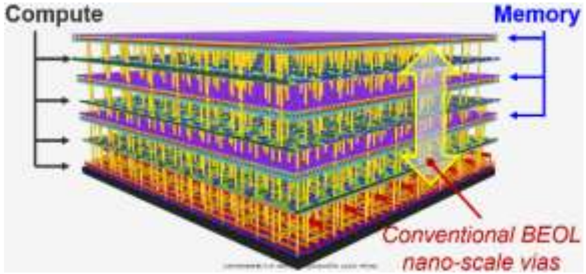
Google TPU



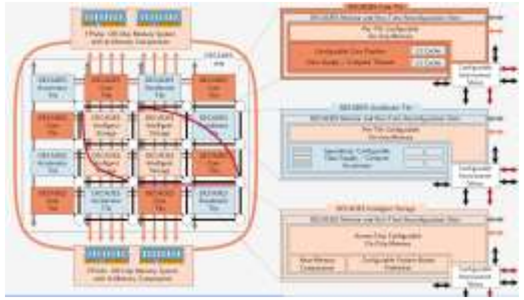
MIT EYERISS



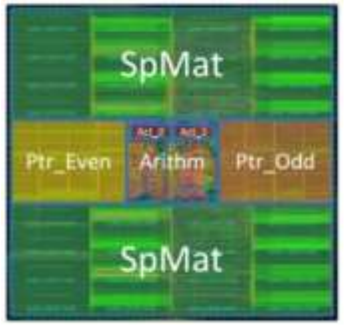
Michigan (DARPA SDH)



DARPA 3DSOC



Princeton DECADES (DARPA SDH)



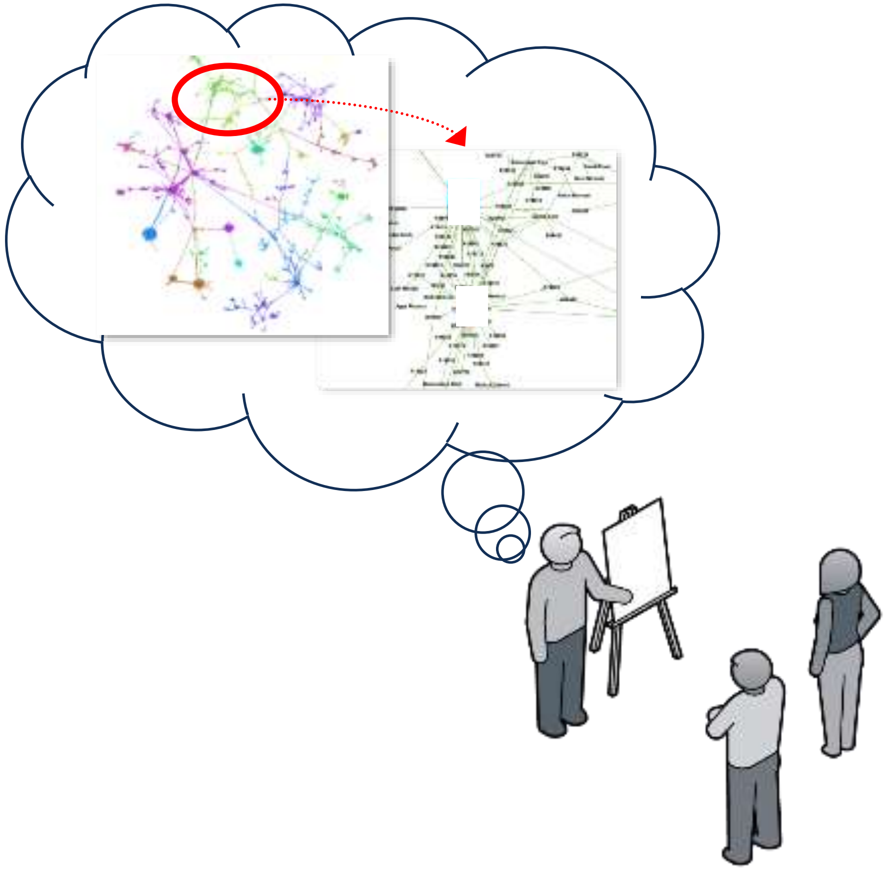
Stanford EIE

Intel PUMA (DARPA HIVE)

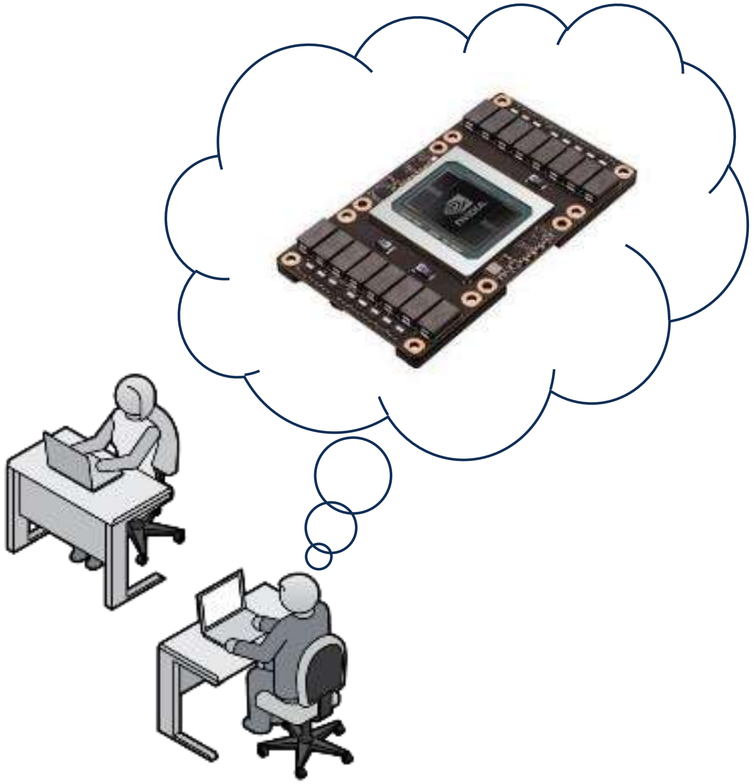
Slide sources: DARPA Electronics Resurgence Initiative (ERI) Summit, July 2019; DARPA ERI Summit, July 2018; DARPA Software Defined Hardware (SDH) Proposers Day, September 2017.

Separation of Concerns

Separate the complexity of algorithms from the complexity of hardware systems:

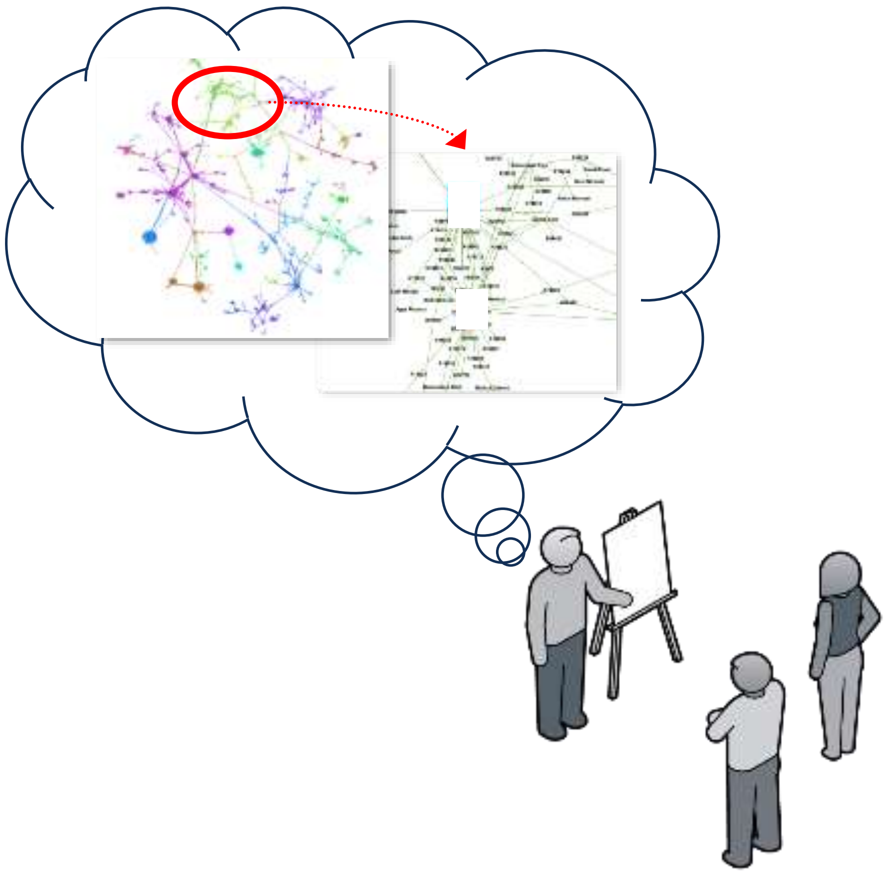


Separation of Concerns



Separation of Concerns

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



Application Programming Interface (API)

Systems for working with Graphs (Irregular Data)

TigerGraph
GraphQ
RedisGraph
PowerSwitch
HipG
GRACE
Mosaic
OrientDB
Chronos
Hama
GBTL
TurboGraph
GunRock
GPS
JCoreDB Graph
Julienne
LFGraph
GraphIn
Piccolo
GBASE
GasCL
FlockDB
Sedge
GoFFish
Ligra
combBLAS
neo4J
GraphBLAS
GoldenOrb
Titan
BPP
Graphulo
GraphBlast
GraphX
Pregelix
Chaos
xDGP
TOTEM

Options for graph processing:
Dazed and Confused

Stinger
GraphHP
GraphMP
Weaver
Grail
GraphMat
GoDB
AlegroGraph
Giraph++
GraphMap
Mosaid
DUALSIM
RASP
LCC-Graph
Gemini
GraphIT
X-stream
GraphX
Hornet
Mizan
Polymer
SuiteSparse
GridGraph
CuSha
cuGraph
PathGraph
Horton+
ArangoDB
XPregel
Galois GB
PowerGraph
Xstream
pggraphblas
GraphLab
Kineograph
HypergraphDB
pygraphblas
Pregel
InfiniteGraph
PyGBTL
GraphChi
Puma
Blogel

Third Party names are the property of their owners. Source: Scalable Graph Processing Frameworks: A Taxonomy and Open Challenges: S. Heidari, Y. Simmhan, R. N. Calheiros, and R. Buyya, ACM Comput. Surv. 51, 3, Article 60 (June 2018); Slide credit: Tim Mattson/Intel Labs, "The Growing GraphBLAS community: Progress Report", LPS Workshop on HPC Data Analytics, Sep. 2019.

Systems for working with Graphs (Irregular Data)

TigerGraph

GraphQ

RedisGraph

PowerSwitch

HipG

GRACE

Mosaic

OrientDB

Chronos

Hama

GBTL

TurboGraph

GunRock

GPS

JCoreDB Graph

Julienne

LFGraph

GraphIn

GBASE

Piccolo

GasCL

FlockDB

Sedge

GoFFish

Ligra

combBLAS

neo4J

GraphBLAS

Graphulo

GoldenOrb

Titan

BPP

GraphBlast

GraphX

Pregelix

Chaos

Graphs Algorithms in the Language of Linear Algebra

xDGP

TOTEM

Stinger

GraphHP

GraphMP

Weaver

Grail

GraphMat

GoDB

AlegroGraph

Giraph++

GraphMap

Mosaid

RASP

DUALSIM

LCC-Graph

Gemini

GraphIT

GraphX

Mizan

Hornet

X-stream

Polymer

CuSha

SuiteSparse

GridGraph

cuGraph

PathGraph

Horton+

ArangoDB

XPregel

Galois GB

PowerGraph

Xstream

pggraphblas

HypergraphDB

GraphLab

Kineograph

pygraphblas

Pregel

InfiniteGraph

PyGBTL

GraphChi

Puma

Blogel

Third Party names are the property of their owners. Source: Scalable Graph Processing Frameworks: A Taxonomy and Open Challenges: S. Heidari, Y. Simmhan, R. N. Calheiros, and R. Buyya, ACM Comput. Surv. 51, 3, Article 60 (June 2018); Slide credit: Tim Mattson/Intel Labs, “The Growing GraphBLAS community: Progress Report”, LPS Workshop on HPC Data Analytics, Sep. 2019.

GraphBLAS References

Mathematical Foundations of the GraphBLAS

IEEE HPEC 2016

Jeremy Kepner (MIT Lincoln Laboratory Supercomputing Center), Peter Aaltonen (Indiana University), David Bader (Georgia Institute of Technology), Aydın Buluç (Lawrence Berkeley National Laboratory), Franz Franchetti (Carnegie Mellon University), John Gilbert (University of California, Santa Barbara), Dylan Hutchison (University of Washington), Manoj Kumar (IBM), Andrew Lumsdaine (Indiana University), Henning Meyerhenke (Karlsruhe Institute of Technology), Scott McMillan (CMU Software Engineering Institute), Jose Moreira (IBM), John D. Owens (University of California, Davis), Carl Yang (University of California, Davis), Marcin Zalewski (Indiana University), Timothy Mattson (Intel)

Design of the GraphBLAS API for C

IEEE HPEC 2017

Aydın Buluç[†], Tim Mattson[‡], Scott McMillan[§], José Moreira[¶], Carl Yang[*,†]

[†]*Computational Research Division, Lawrence Berkeley National Laboratory*

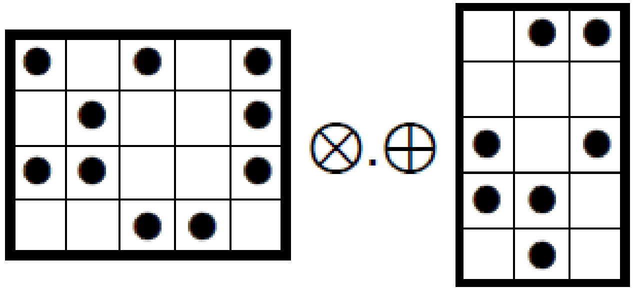
[‡]*Intel Corporation* §*Software Engineering Institute, Carnegie Mellon University* ¶*IBM Corporation*

[*]*Electrical and Computer Engineering Department, University of California, Davis, USA*

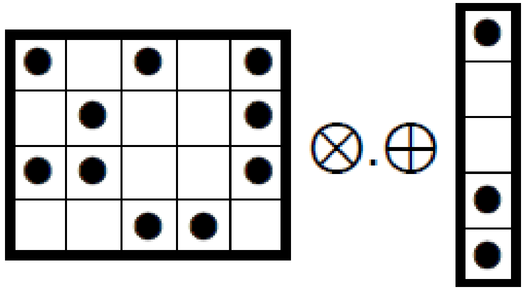
GraphBLAS Primitives

- Basic objects (opaque types)
  - Matrix, vector, algebraic structures, and ”control objects”
- Fundamental operations over these objects

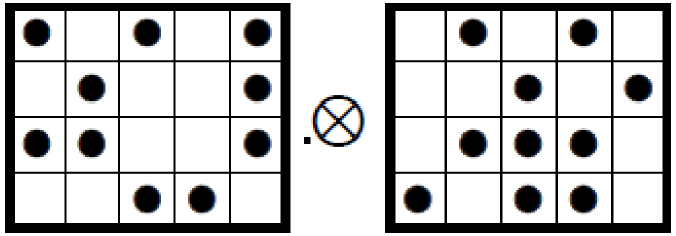
Sparse matrix times sparse matrix



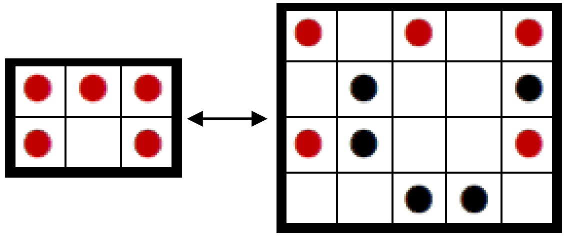
Sparse matrix times sparse vector



Element-wise multiplication (and addition)



Sparse matrix extraction (and assignment)



…plus reductions, transpose, and application of a function to each element of a matrix or vector

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, Sep 2019.

GraphBLAS Ecosystem: This year

GraphBLAS Forum: https://graphblas.org

GraphBLAS C API, v. 1.3.0

GraphBLAS C++ API (in progress)

gbtl
C++ Algorithms Repository
Carnegie Mellon University

Carnegie Mellon University GraphBLAS Test Framework

pggraphblas
PostgreSQL Wrapper

pygraphblas
Python Wrapper

LAGraph
C algorithms repository

pyGB
Python Wrapper
around gbtl
Carnegie Mellon University W

redisgraph
redislabs

Multithreaded
SuiteSparse
ATM

IBM-GraphBLAS

Galois GB
TEXAS
The University of Texas at Austin

Distributed gbtl
Lawrence Livermore National Laboratory

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

GraphBLAST (GPU)
UCDAVIS
UNIVERSITY OF CALIFORNIA

GraphBLAS on EMU
Carnegie Mellon University UMBC

GraphBLAS Ecosystem: This year

GraphBLAS Forum: https://graphblas.org

gbtl
C++ Algorithms Repository
Carnegie Mellon University

Carnegie Mellon University **GraphBLAS Test Framework**

pggraphblas
PostgreSQL Wrapper

pygraphblas
Python Wrapper

LAGraph
C algorithms repository

pyGB
Python Wrapper around gbtl
Carnegie Mellon University W

redisgraph
redislabs

GraphBLAS C API, v. 1.3.0
GraphBLAS C++ API (in progress)

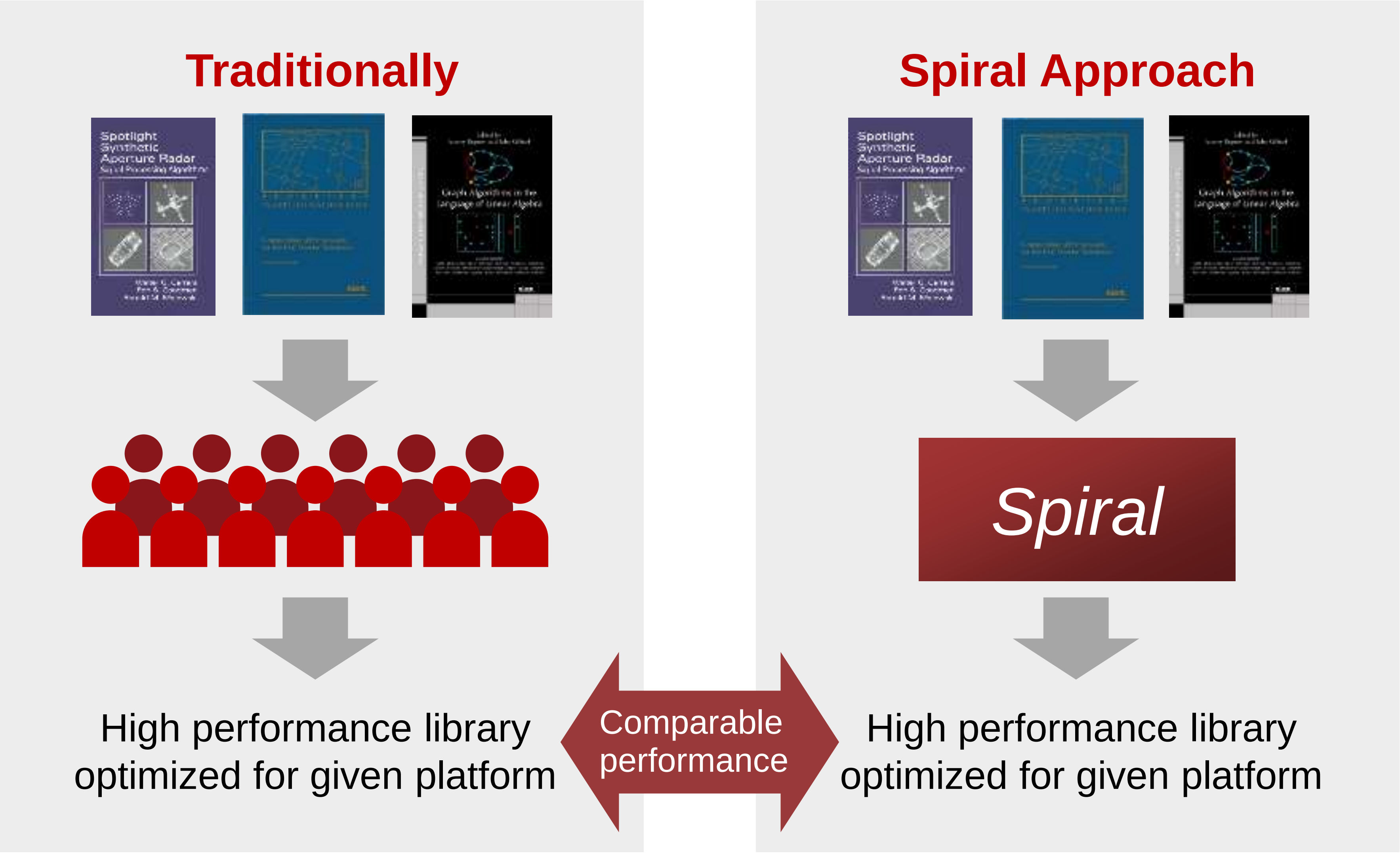
Multithreaded
SuiteSparse
A|M
IBM GraphBLAS
GaloisGB
TEXAS
Distributed gbtl
Lawrence Livermore National Laboratory
gbtl
Carnegie Mellon University
Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965
GraphBLAST (GPU)
UCDAVIS
UNIVERSITY OF CALIFORNIA
GraphBLAS on EMU
Carnegie Mellon University
UMBC

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

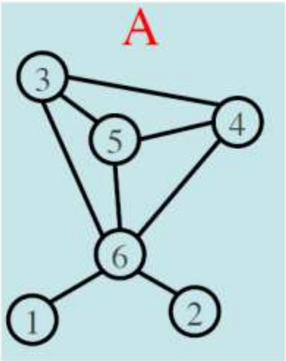
Spiral Code Generation

Prof. Franz Franchetti, CMU ECE

What is Spiral?



Spiral: Platform-Aware Formal Program Synthesis

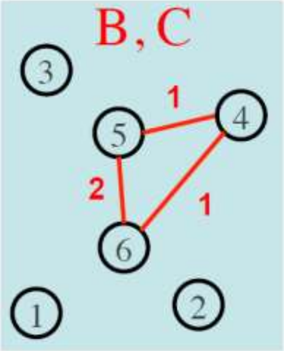


$A = L + U$ (hi->lo + lo->hi)

$L \times U = B$ (wedge, low hinge)

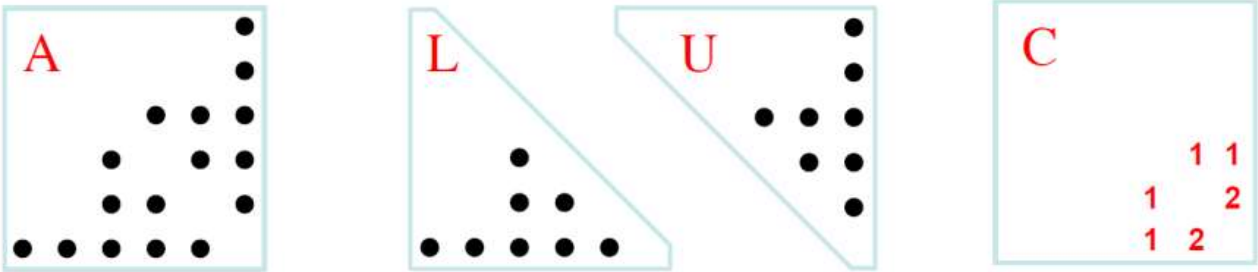
$A \wedge B = C$ (closed wedge)

$\mathrm{sum}(C)/2 =$ **4 triangles**



$$\#\Delta = \frac{1}{6}\, tr(\mathbf{A}^3)$$

$$= \|\mathbf{L} \;.\!*\; (\mathbf{L} * \mathbf{L}^T)\|_1$$



$$C\langle L, z\rangle = (L \oplus.\otimes L^T)$$

$$\#\Delta = \bigoplus_{i,j} C(i,j)$$

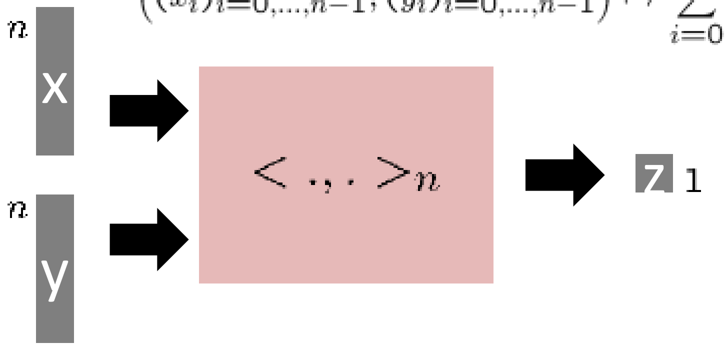
```
int triangle_count(Matrix const &L)
{
  Matrix C(L.nrows(), L.ncols());
  mxm(C, L, NoAccumulate(), ArithmeticSemiring<int>(),
      L, transpose(L));

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

SPIRAL's Math Framework

High Level Operators

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

$$\big((x_i)_{i=0,\ldots,n-1}, (y_i)_{i=0,\ldots,n-1}\big) \mapsto \sum_{i=0}^{n-1} x_i y_i$$



Basic Operators

$$\text{Pointwise}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}^n$$
$$(x_i)_i \mapsto f_0(x_0) \oplus \cdots \oplus f_{n-1}(x_{n-1})$$

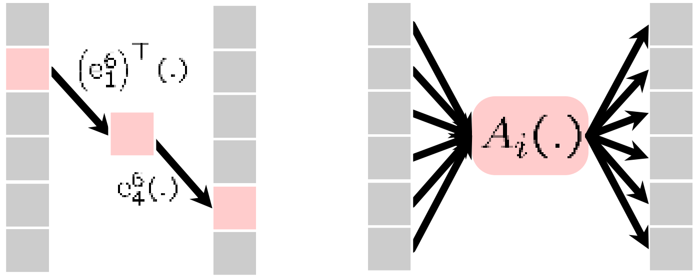
$$\text{Atomic}_{f(.,.)} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$
$$(x, y) \mapsto f(x, y)$$

$$\text{Pointwise}_{n\times n,f_i} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$$
$$\big((x_i)_i, (y_i)_i\big) \mapsto f_0(x_0, y_0) \oplus \cdots \oplus f_{n-1}(x_{n-1}, y_{n-1})$$

$$\text{Reduction}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_i)_i \mapsto f_{n-1}(x_{n-1}, f_{n-2}(x_{n-2}, f_{n-3}(\ldots f_0(x_0, \text{id}())\ldots)$$

Loop Abstraction

$$\coprod_{i=0}^{n-1} : (D \to R)^n \to (D \to R)$$
$$A_i \mapsto (x \mapsto A_0(x) \sqcup \cdots \sqcup A_{n-1}(x))$$



Rule Based Compiler

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

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

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

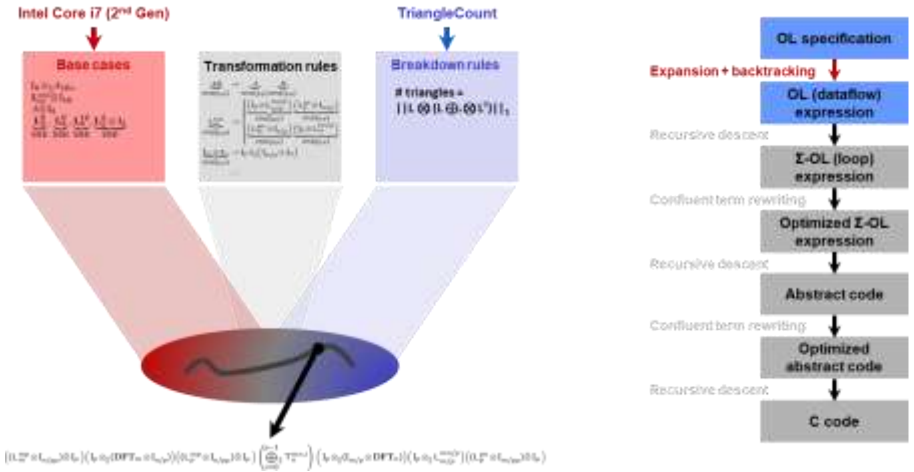
$$\text{Code}\big(y = e_i^n(x)\big) \to \big\{y = \vec{0}, y[i] := x[0]\big\}$$

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

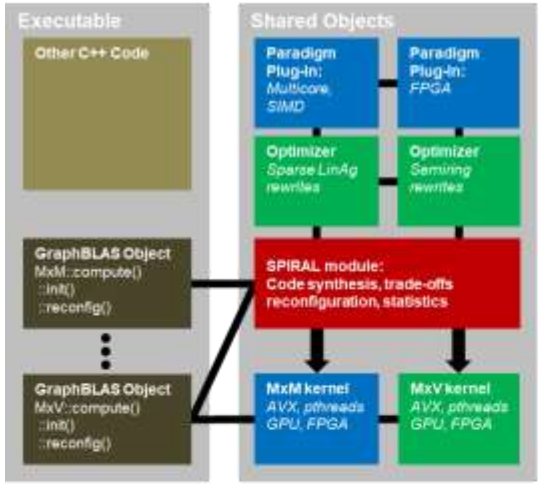
Leverages DARPA HACMS

SPIRAL Internals: Autotuning and Code Generation

Autotuning in Constraint Space



SPIRAL as JIT and GraphBLAS Optimizer



Source Code

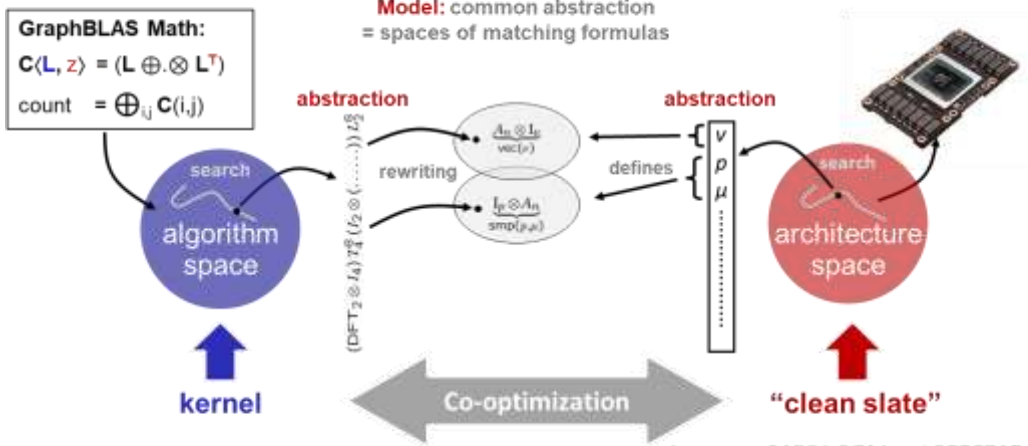
- C++, GraphBLAS calls, other supported libraries
- Code = specification, not program

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

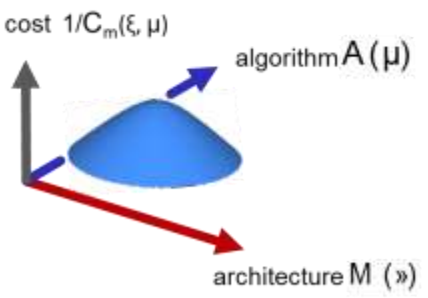
Formal Approach To Co-Optimization



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

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

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

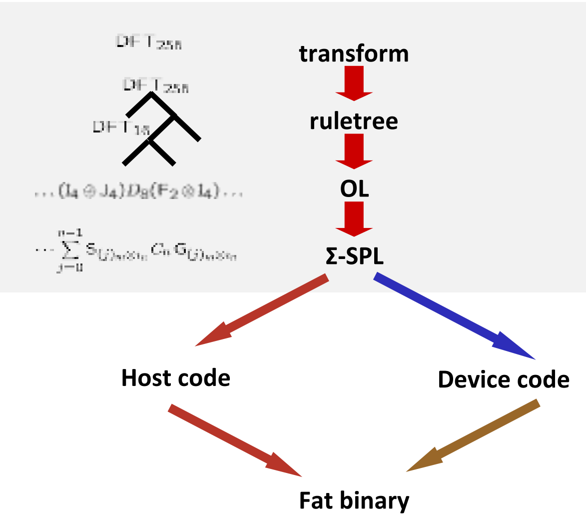
Task: Find ξ and μ s.t. $C_m(\xi, \mu)$ is minimal

Targeting FPGAs With SPIRAL

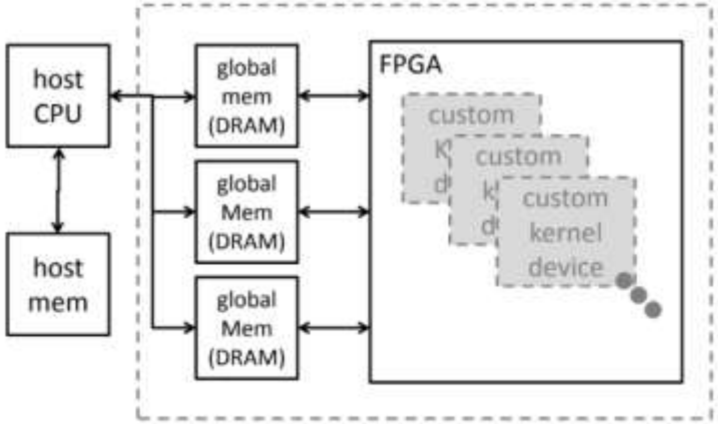
Range of Platforms

Small dev board
to multi-board server

FPGAs in SPIRAL Flow



Execution Layer: OpenCL



BUT: Not standard OpenCL Code

```
__kernel mmm(__global float *A, … *B, … *C) {
  for(int i=0; i<N; i++)
    for(int j=0; j<N; j++)
      for(in k=0; k<N; k++)
        C[i*N+j]=C[i*N+j]+A[i*N+k]*B[k*N+j]
```

- NDRange=(1, 1, 1)   never do this on GPU!!
- Arbitrary control flow (loops, if's) and dependencies
- Becomes just "regular" C-to-HW synthesis
  - pipeline and parallelize loops
  - schedule for initiation-interval, resource, etc.

Only want OpenCL's platform model and API;
"work-group" & "work-item" not too meaningful

Graph Algorithms in Spiral

Problem Specification:

TriangleCount()

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

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 for Triangle Counting and K-TRUSS

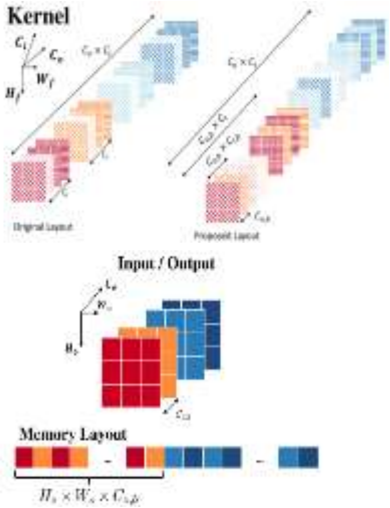
```
spiral> t := TriangleCount();
TriangleCount()
spiral> rt := RandomRuleTree(t, opts);
Accum_VMV_FLAME2( TriangleCount() )
spiral> srt := SumsRuleTree(rt, opts);
BB(
  Accum(i1, 1, 262110,
    Accum_X(i3, [ i1, 0 ], i1,
      Dot([ i3, add(i1, V(1)) ], [ i1, add(i1, V(1)) ], sub(sub(V(262111), i1), 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)))
```

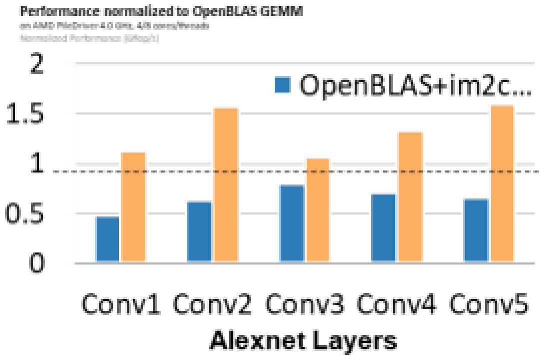
```
void ktruss(int *dEk, int k) {
  int *S = (int*)malloc(E * sizeof(int));
  int *IAk = (int*)malloc((V+1) * sizeof(int));
  int *JAk = (int*)malloc(E * sizeof(int));
  int Ek = E;

  for (int i = 0; i < V+1; i++) IAk[i] = IA_CSR[i];
  for (int i = 0; i < E; i++) JAk[i] = JA_CSR[i];

  int iter = 1;
  while (1) {
    int row = 0;
    for (int i = 0; i < Ek; i++) {
      while (IAk[row+1] == i) row++;
      int i0 = IAk[row]; int b0 = IAk[row + 1];
      int i1 = IAk[JAk[i]];
      int b1 = IAk[JAk[i] + 1];
      int res = 0;
      while (i0 < b0 && i1 < b1) {
        int v0 = JAk[i0]; int v1 = JAk[i1];
        if (v0 == v1)
          res++;
        if (v0 <= v1)
          i0++;
        if (v1 <= v0)
          i1++;
      }
      S[i] = res;
    }
  ...
```

500 lines generated C code

Current Efforts Feeding Into SPIRAL AI/ML

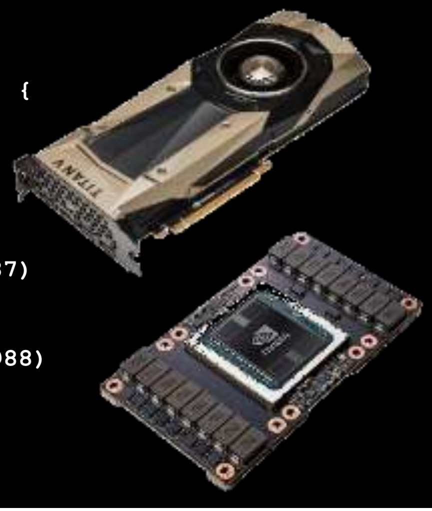
CNN/DNN Code Generation



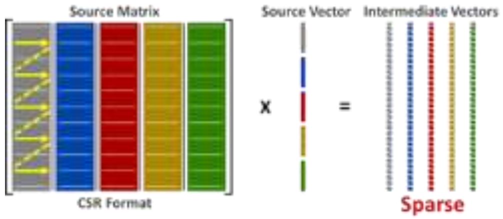


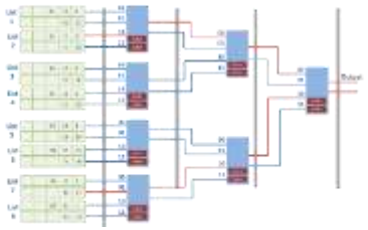
SPIRAL CUDA/OpenACC GPU Target

```
void ioprunedconv_130_0_62_72_130(double  *Y, double  *X, double * S) {

    ...
for(int i18899 = 0; i18899 <= 1; i18899++) {
    for(int i18912 = 0; i18912 <= 4; i18912++) {
        a9807 = ((2*i18899) + (4*i18912));
        a9808 = (a9807 + 1);
        a9809 = (a9807 + 52);
        a9810 = (a9807 + 53);
...
        *((104 + Y + a12569)) = ((s3983 - s3987)
            + (0.80901699437494745*t6537)
            + (0.58778525229247314*t6538));
        *((105 + Y + a12569)) = (((s3984 - s3988)
            + (0.80901699437494745*t6538))
            - (0.58778525229247314*t6537));
    }
}
```



A SPIRAL/SEI Chip in 2020/2021





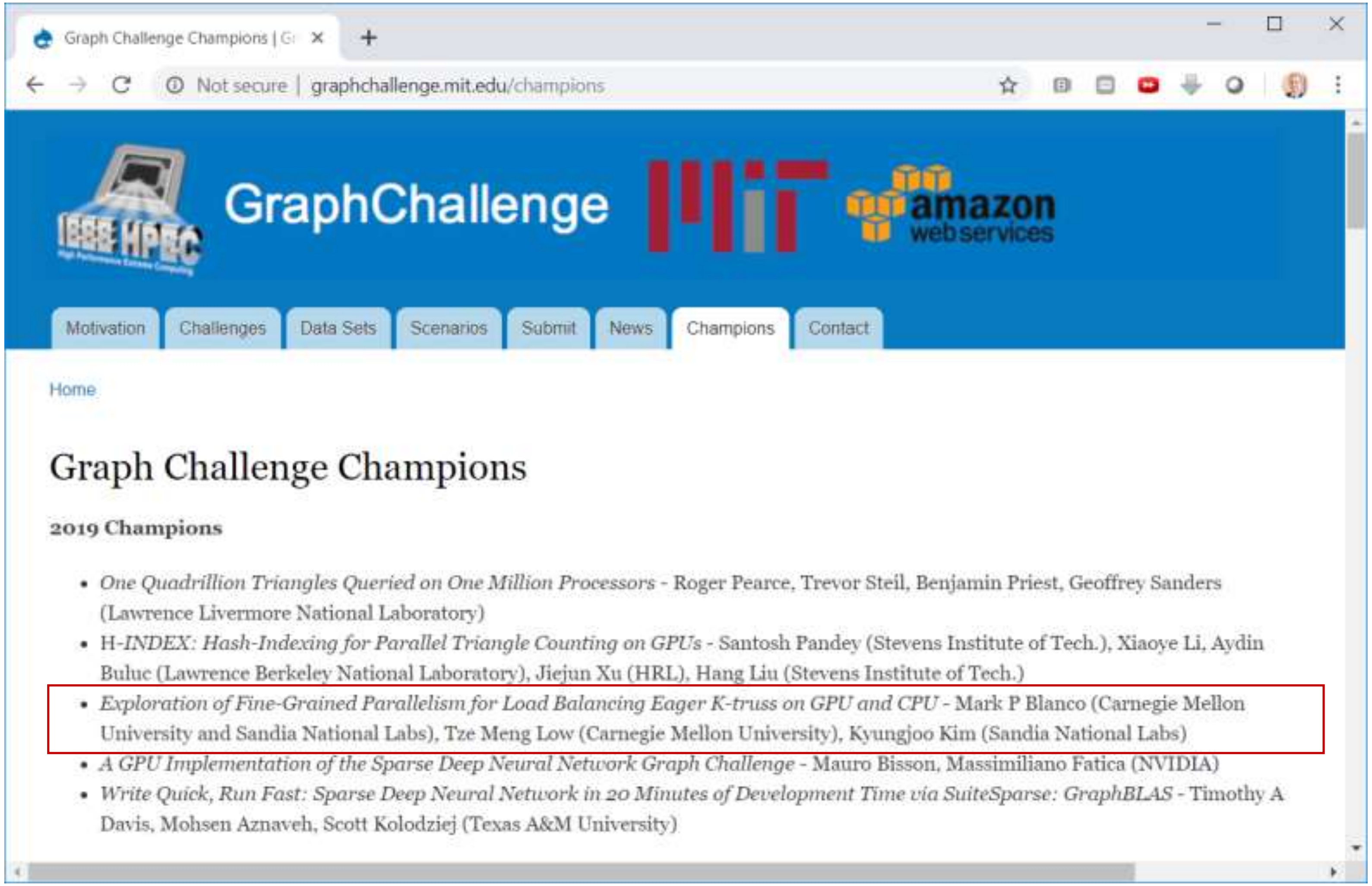
SPIRAL Library as DSL Frontend

```
fftx_plan pruned_real_convolution_plan(fftx_real *in, ….
    …
    tmp4 = fftx_create_temp_real(rank, &padded_dims);
    plans[3] = fftx_plan_guru_dft_c2r(rank, &padded_dims_batch_rank,
        &bat
    plans[4]
    p = fftx

    return p
}
```

This is a specification dressed as a program

- Needs to be clean and concise
- No code level optimizations and tricks
- Don't think "performance" but "correctness"
- *For small and in-development platforms*

Result: Graph Challenge Champions



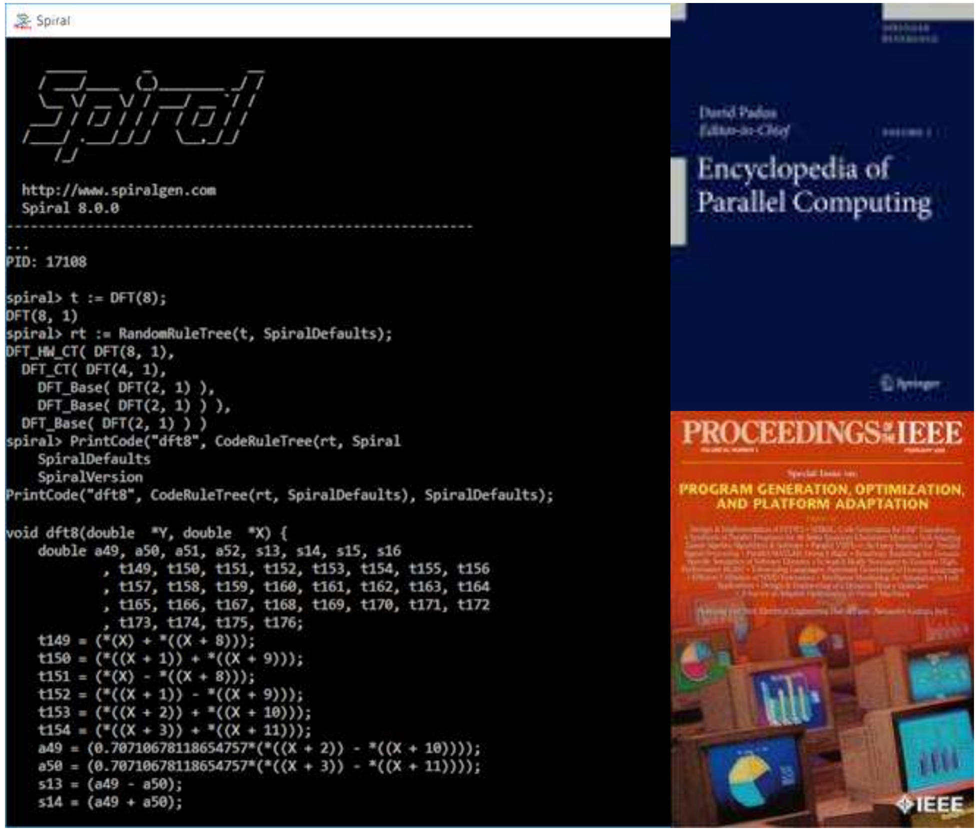
Exploration of Fine-Grained Parallelism for Load Balancing Eager K-truss on GPU and CPU

Mark Blanco, Tze Meng Low — Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, United States

Kyungjoo Kim — Center for Computing Research, Sandia National Laboratories, Albuquerque, United States

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, JPL, DOE, CMU SEI, Intel, Nvidia, Mercury
- Open sourced under DARPA PERFECT
- Ongoing Partnership between SEI and ECE



www.spiral.net

External Collaborators

Pacific Northwest National Laboratory
W

- Andrew Lumsdaine
- Marcin Zalewski
- Kevin Deweese
- Jesun Firoz

Lincoln Laboratory
Massachusetts Institute of Technology

- Jeremy Kepner

Lawrence Livermore National Laboratory

- Roger Pearce
- Trevor Steil

Berkeley Lab

- Aydin Buluc
- Benjamin Brock

UC Davis
University of California

- Carl Yang

LPS
The Laboratory for Physical Sciences

- Kaushik Velusamy
- Tyler Simon

Intel Research Labs

- Tim Mattson

IBM

- Jose Moreira
- Manoj Kumar

Emu Technology

- Janice McMahon
- Eric Hein

Texas A&M University

- Tim Davis