

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

Copyright 2019 Carnegie Mellon University.

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.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM19-1113

Team



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



Prof. Franz Franchetti



Prof. Tze Meng Low
CMU, co-PI's



Prof. James C. Hoe



Dr. John Wohlber



Dr. Jason Larkin



Elliott Binder



Mark Blanco



Paul Brouwer



Anuva Kulkarni



Dr. Fazle Sadi



Dr. Daniele Spampinato



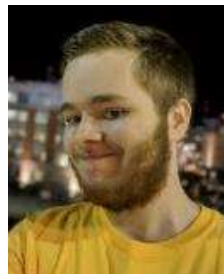
Courtney Rankin



Sandra Sanjeev



Peter Oostema



George Ralph*



Upasana Sridhar*



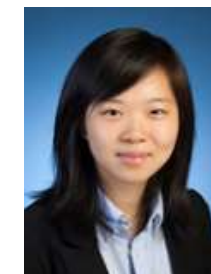
Arvind Srinivasan*



Guanglin Xu



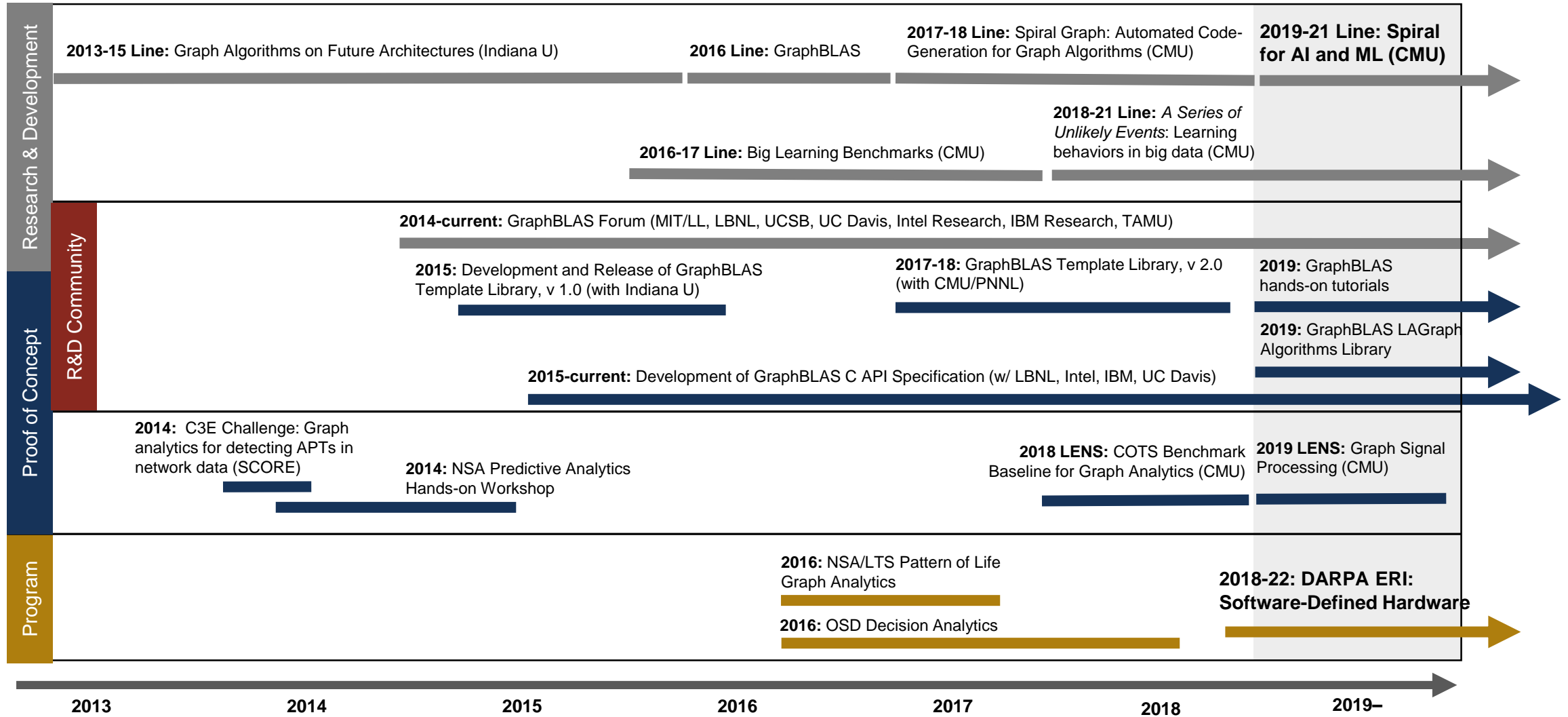
Vadim Zaliva



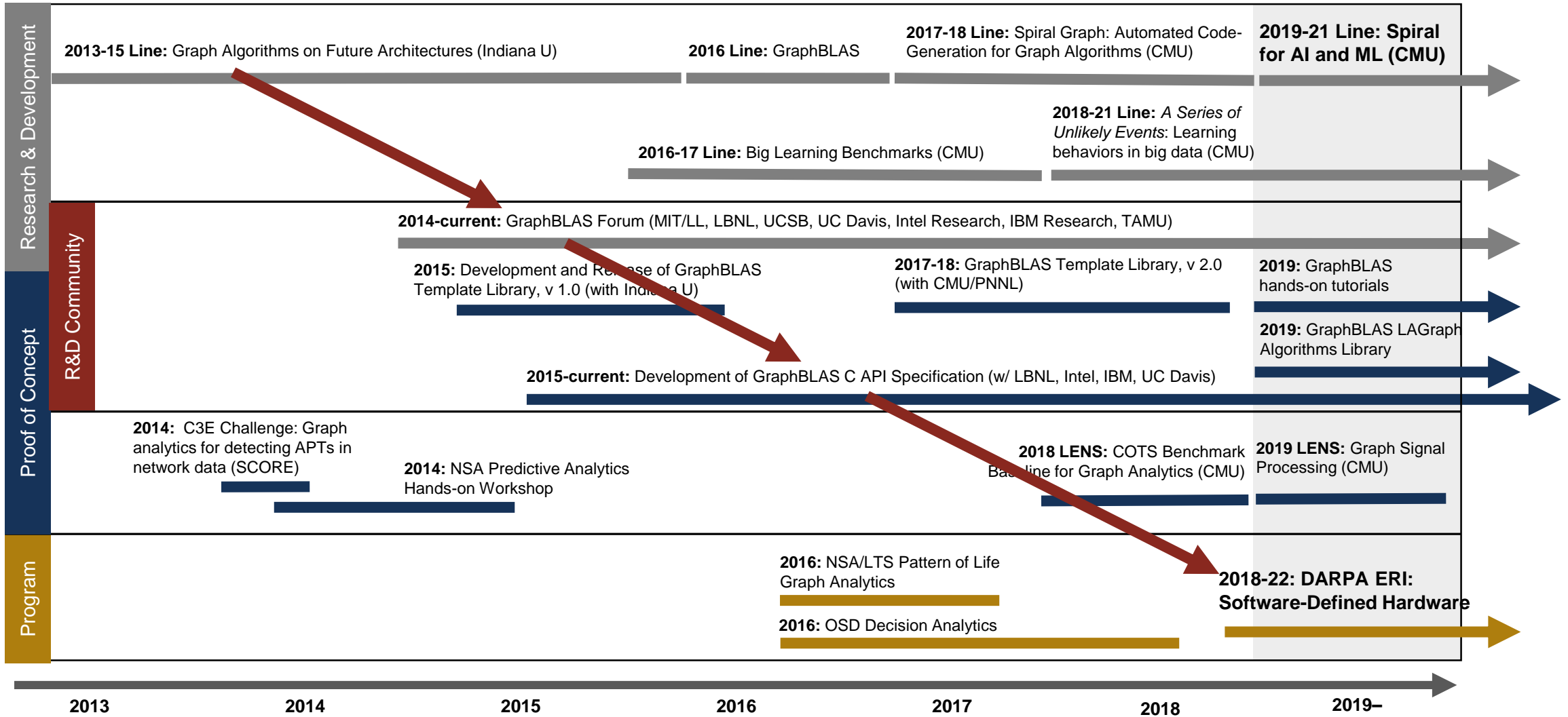
Jiyuan Zhang

* 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

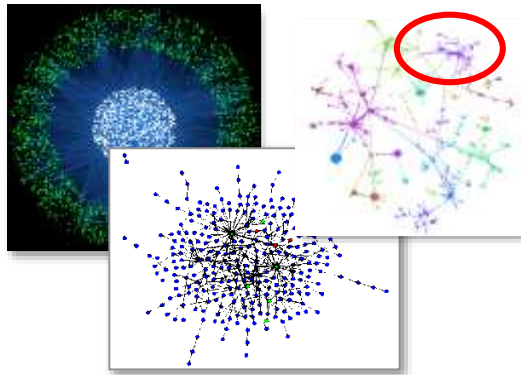
- Automatic code generation for data-intensive computations.
- Simultaneous, automatic co-optimization of hardware within CSWAP constraints.

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



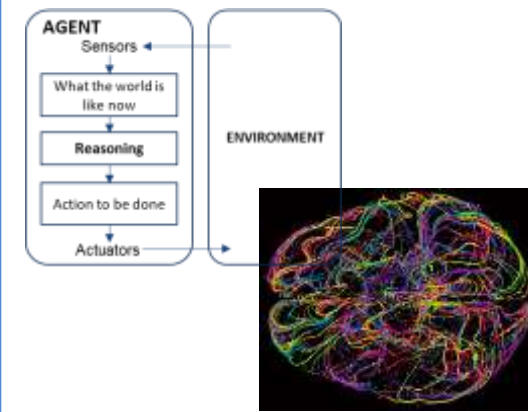
- Graphs represent entities and relationships detected through multi-INT sources.
- 1,000s – 100,000,000,000s tracks, interactions, events
- GOAL: Find clusters of similar entities or behaviors of interest.

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.

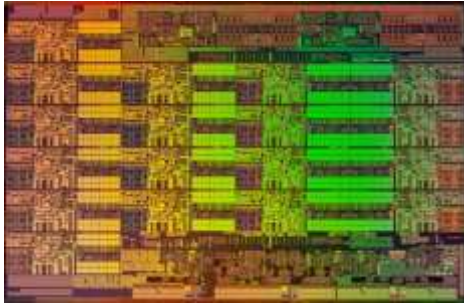
Artificial Intelligence



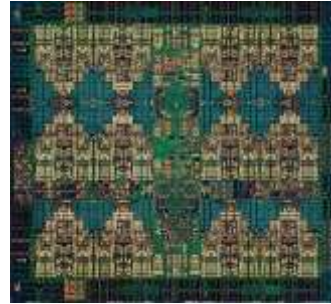
- AI is the collection of computations that make it possible to perceive, reason and act.
- Practically infinite state space and non-deterministic dynamics
- GOAL: Create intelligent agents that achieve target performance in deployed environments.

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

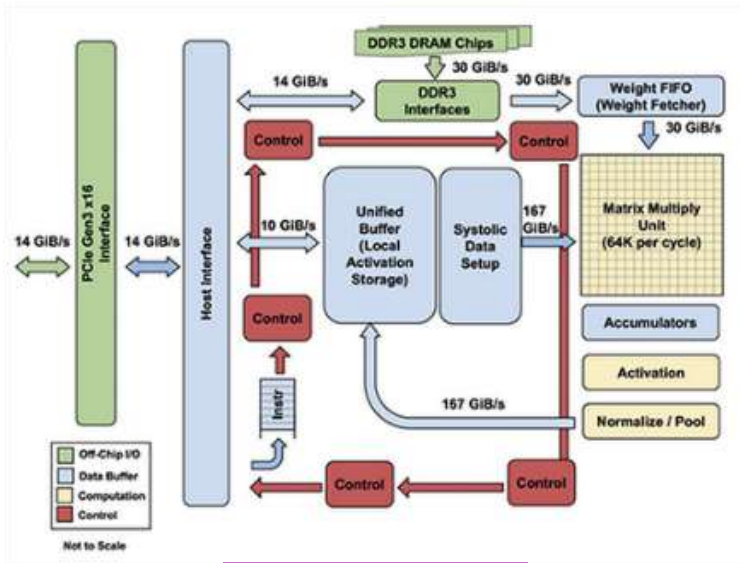


Summit
 187.7 Pflop/s, 13 MW
 9,216 x 22 cores POWER9
 + 27,648 V100 GPUs

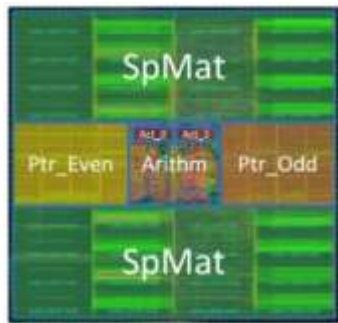
1 Gflop/s = one billion floating-point operations (additions or multiplications) per second

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

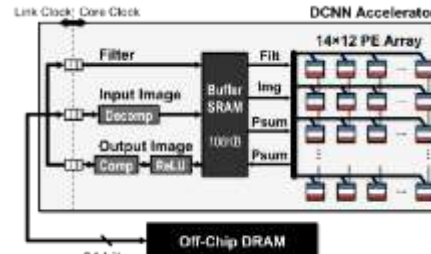
Today's Computing Landscape...is not tomorrow's.



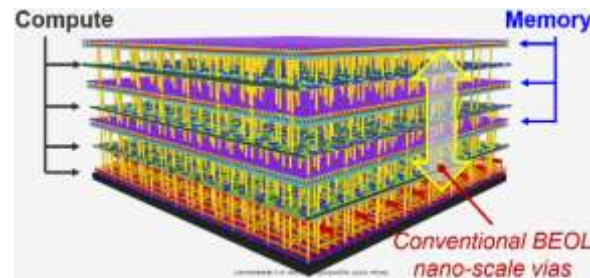
Google TPU



Stanford EIE



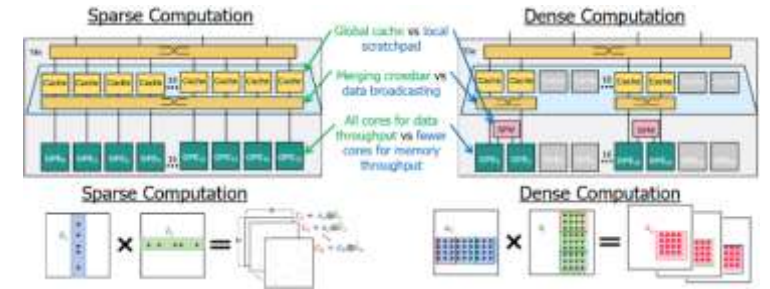
MIT EYERISS



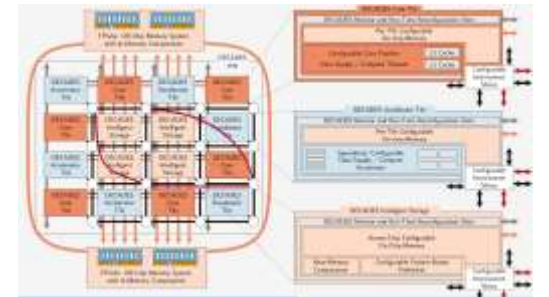
DARPA 3DSOC



Intel PUMA (DARPA HIVE)



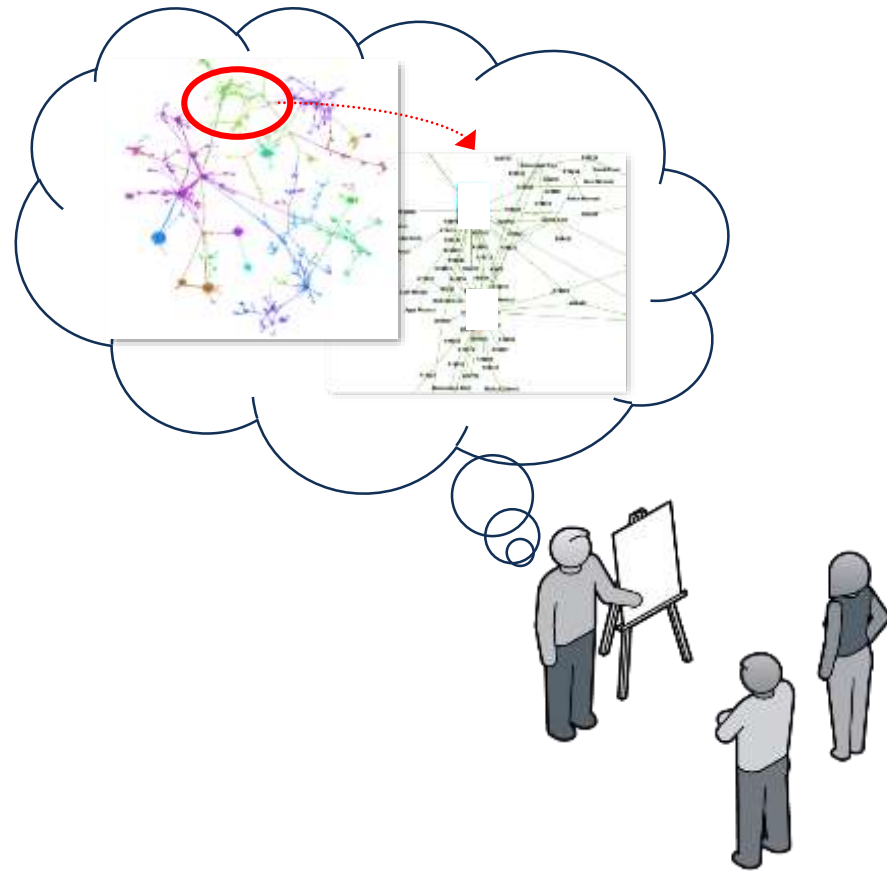
Michigan (DARPA SDH)



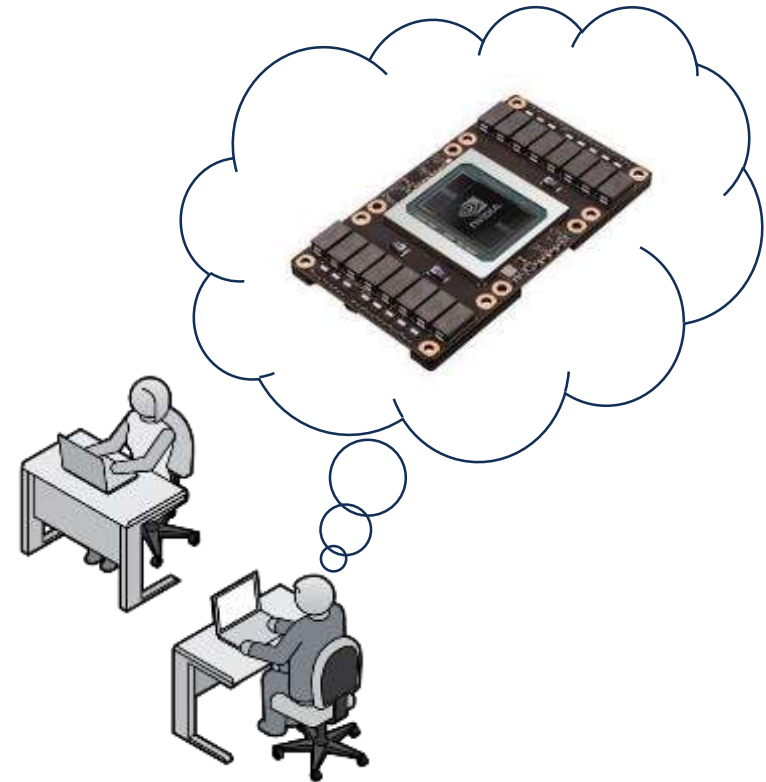
Princeton DECADES (DARPA SDH)

Separation of Concerns

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

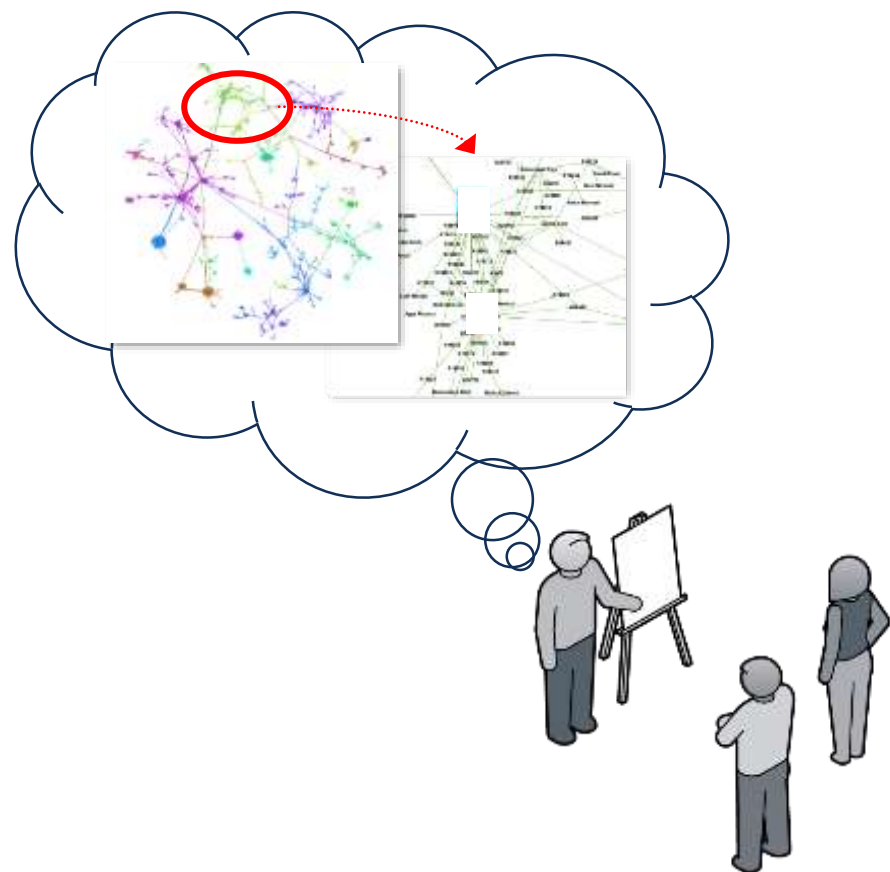


Separation of Concerns



Separation of Concerns

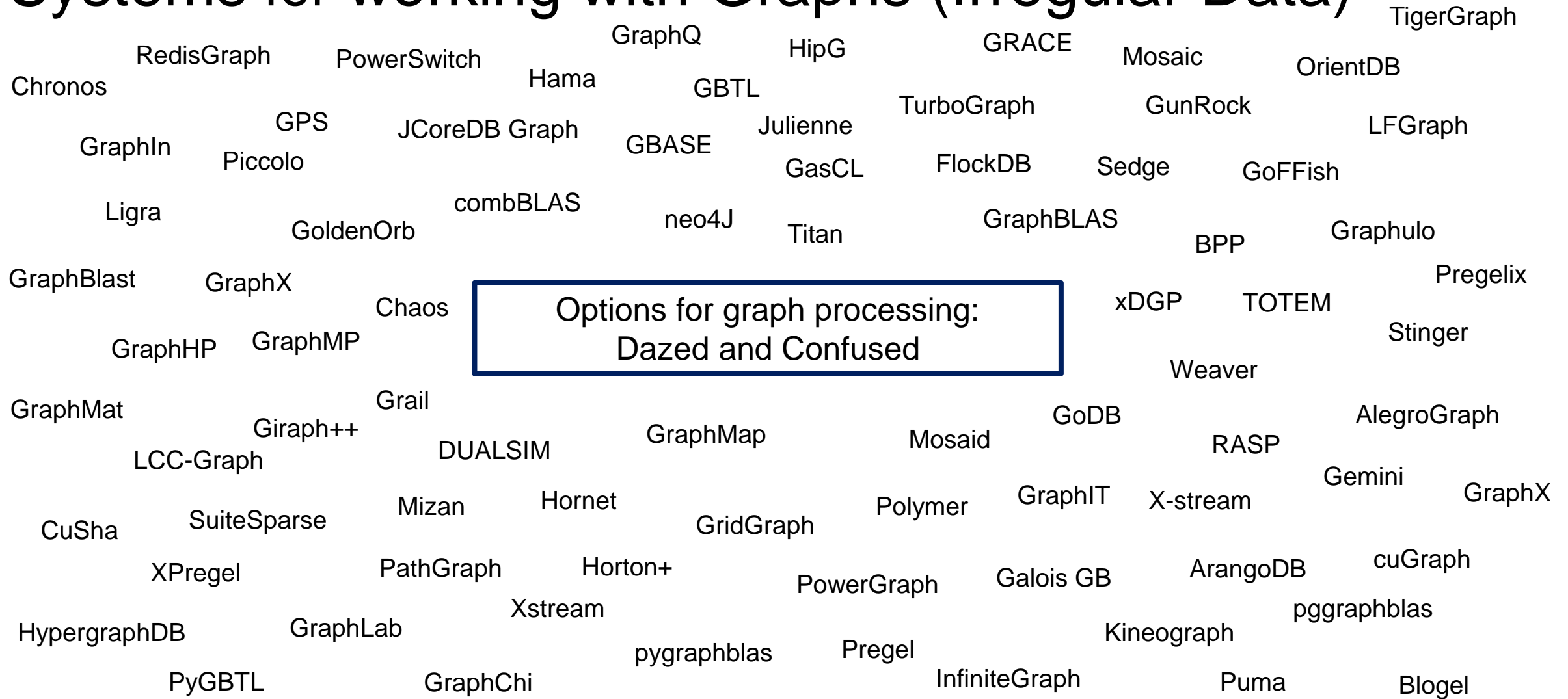
GOAL: write once, run everywhere...fast (**with** help from hardware experts).



Application Programming Interface (API)

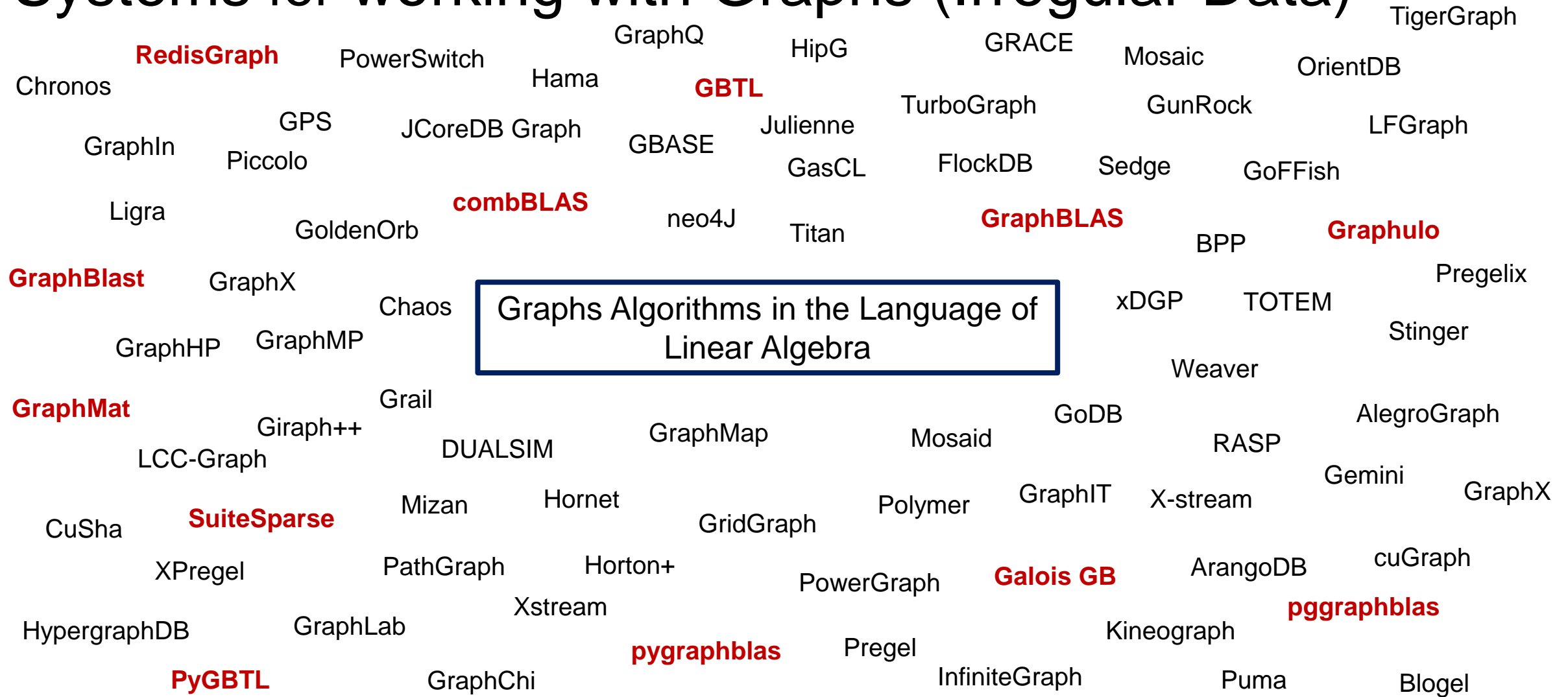


Systems for working with Graphs (Irregular Data)



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)



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*

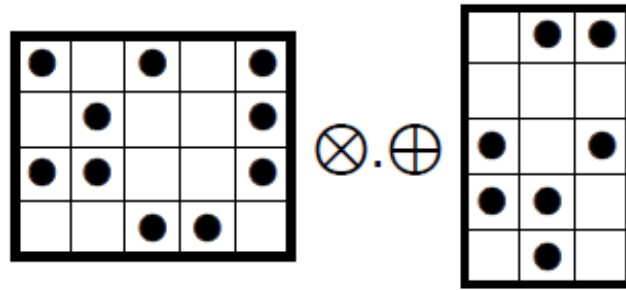
[‡]*Intel Corporation* [§]*Software Engineering Institute, Carnegie Mellon University* [¶]*IBM Corporation*

^{*}*Electrical and Computer Engineering Department, University of California, Davis, USA*

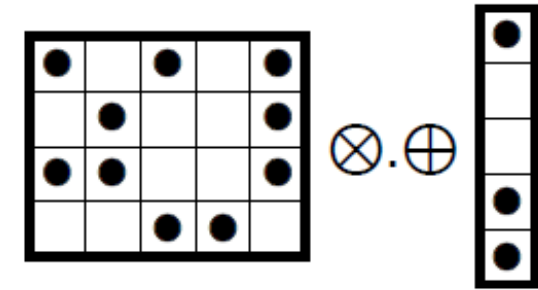
GraphBLAS Primitives

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

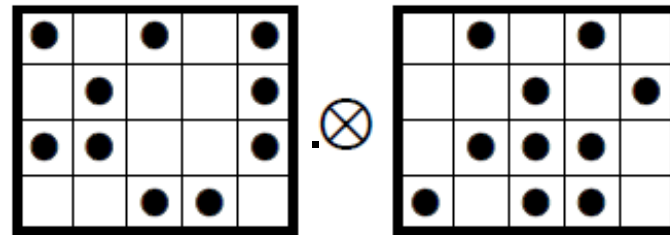
Sparse matrix times
sparse matrix



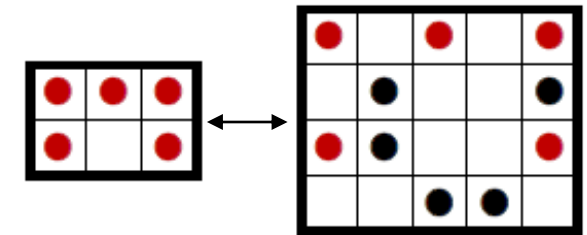
Sparse matrix times
sparse vector



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>

gblt

C++ Algorithms

Repository

Carnegie Mellon University

LAGraph

C algorithms repository

redisgraph

redislabs

Carnegie Mellon University

GraphBLAS Test Framework

pggraphblas

PostgreSQL Wrapper

pygraphblas

Python Wrapper

pyGB

Python Wrapper

around **gblt**

Carnegie Mellon University

W

GraphBLAS C API, v. 1.3.0

GraphBLAS C++ API (in progress)

Multithreaded



IBM-GraphBLAS

Galois GB



Distributed gblt



gblt

Carnegie Mellon University

Pacific Northwest NATIONAL LABORATORY

Proudly Operated by **Battelle** Since 1965

GraphBLAST (GPU)



GraphBLAS on EMU



GraphBLAS Ecosystem: This year

GraphBLAS Forum: <https://graphblas.org>

gblt

C++ Algorithms

Repository

Carnegie Mellon University

LAGraph

C algorithms repository

redisgraph

redislabs

Carnegie Mellon University

GraphBLAS Test Framework

pggraphblas

PostgreSQL Wrapper

pygraphblas

Python Wrapper

pyGB

Python Wrapper

around **gblt**

Carnegie Mellon University

W

GraphBLAS C API, v. 1.3.0

GraphBLAS C++ API (in progress)



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

GraphBLAST (GPU)

UC DAVIS UNIVERSITY OF CALIFORNIA

ATM

Galileo TEXAS

Distributed gblt Lawrence Livermore National Laboratory

gblt Carnegie Mellon University

Pacific Northwest NATIONAL LABORATORY Proudly Operated by Battelle Since 1963

GraphBLAS on EMU

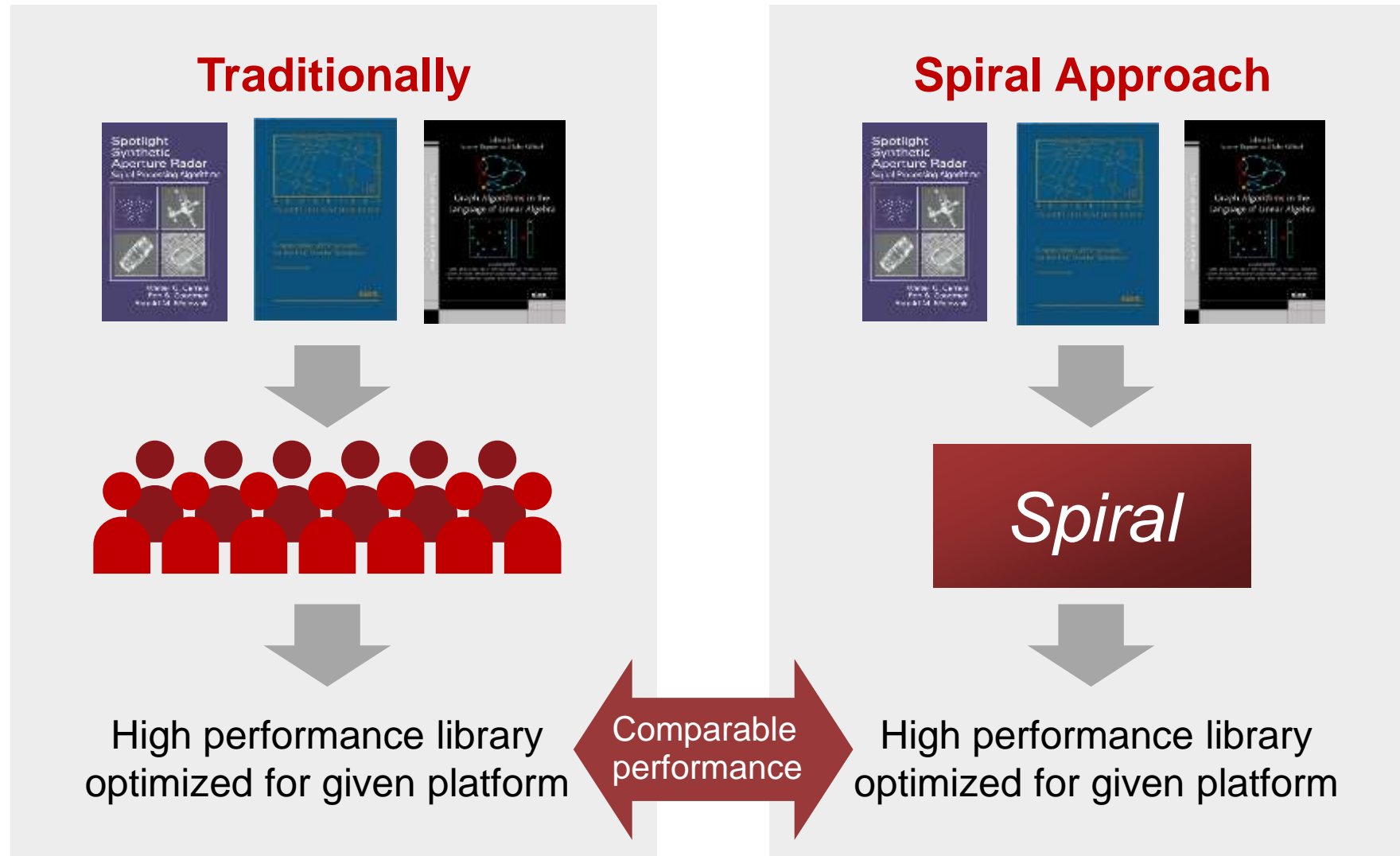
Carnegie Mellon University UMBC

Spiral Code Generation

Prof. Franz Franchetti, CMU ECE



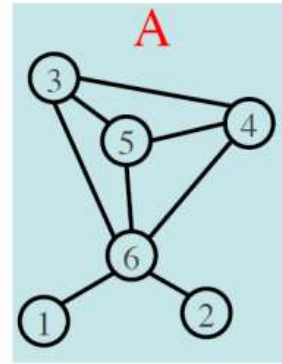
What is Spiral?



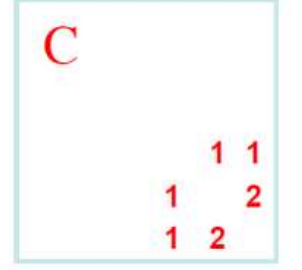
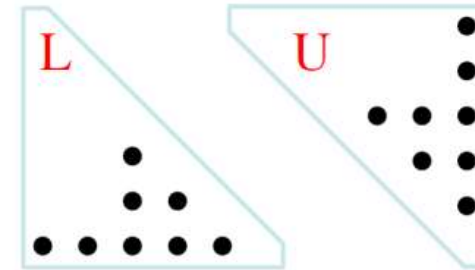
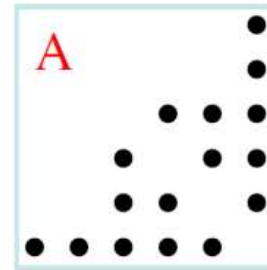
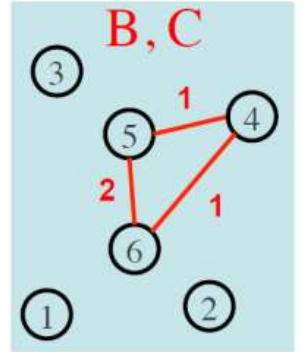
Spiral: Platform-Aware Formal Program Synthesis

$$\begin{aligned} \#\Delta &= \frac{1}{6} \text{tr}(\mathbf{A}^3) \\ &= \|\mathbf{L} \cdot (\mathbf{L} * \mathbf{L}^T)\|_1 \end{aligned}$$

$$\begin{aligned} \mathbf{C}(\mathbf{L}, z) &= (\mathbf{L} \oplus \cdot \otimes \mathbf{L}^T) \\ \#\Delta &= \bigoplus_{i,j} \mathbf{C}(i,j) \end{aligned}$$



$$\begin{aligned} \mathbf{A} &= \mathbf{L} + \mathbf{U} && (\text{hi} \rightarrow \text{lo} + \text{lo} \rightarrow \text{hi}) \\ \mathbf{L} \times \mathbf{U} &= \mathbf{B} && (\text{wedge, low hinge}) \\ \mathbf{A} \wedge \mathbf{B} &= \mathbf{C} && (\text{closed wedge}) \\ \text{sum}(\mathbf{C})/2 &= && \mathbf{4 \text{ triangles}} \end{aligned}$$

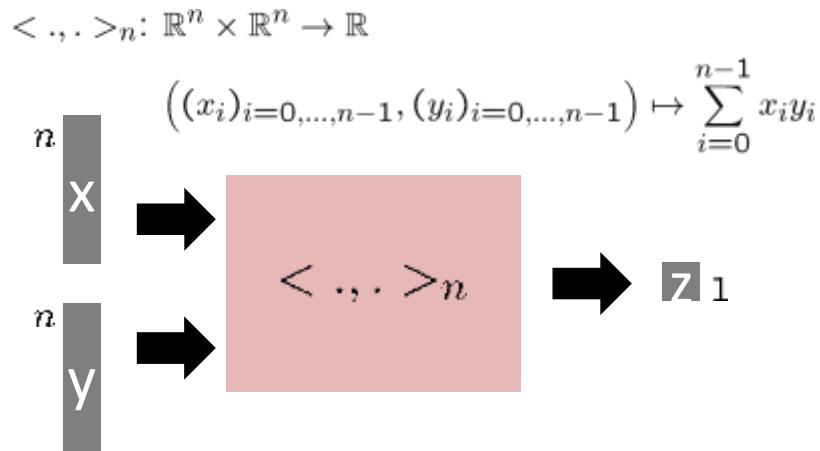


```
int triangle_count(Matrix const &L)
{
    Matrix C(L.nrows(), L.ncols());
    mxm(C, L, NoAccumulate(), ArithmeticSemiring<int>(),
        L, transpose(L));

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

SPIRAL's Math Framework

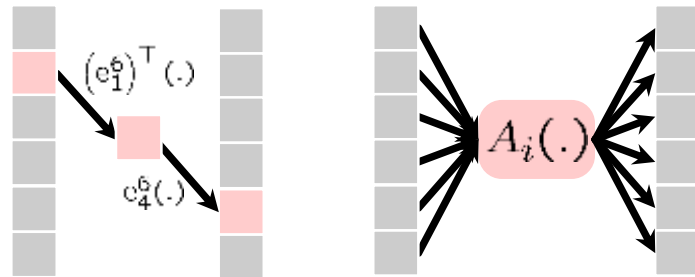
High Level Operators



Loop Abstraction

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

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



Basic Operators

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

Atomic $_{f(\cdot, \cdot)}: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$
 $(x, y) \mapsto f(x, y)$

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

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

Rule Based Compiler

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

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

$\text{Code}(y = (e_i^n)^T(x)) \rightarrow y[0] := x[i]$

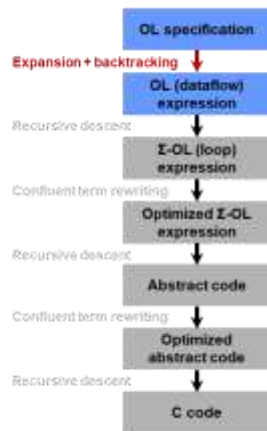
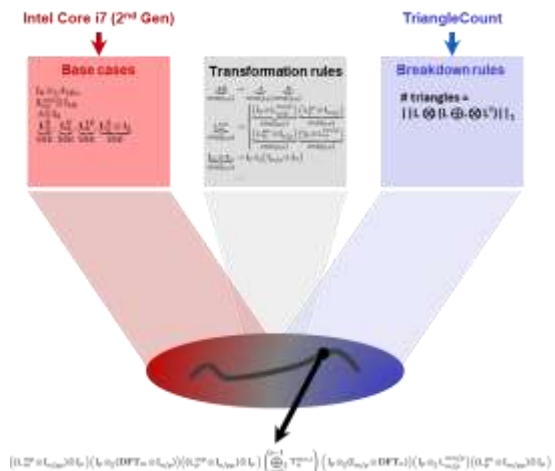
$\text{Code}(y = e_i^n(x)) \rightarrow \{y = \vec{0}, y[i] := x[0]\}$

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

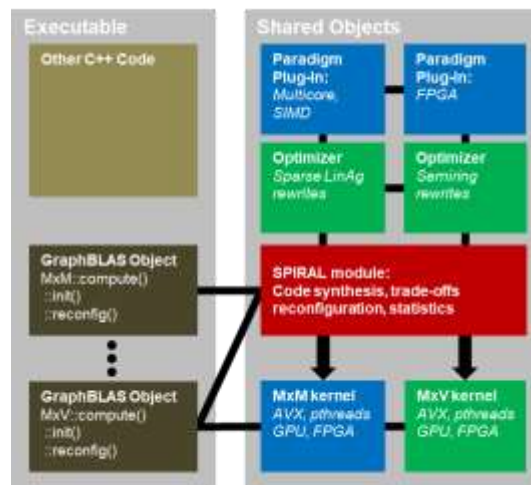
Leverages DARPA HACMS

SPIRAL Internals: Autotuning and Code Generation

Autotuning in Constraint Space



SPIRAL as JIT and GraphBLAS Optimizer



Source Code

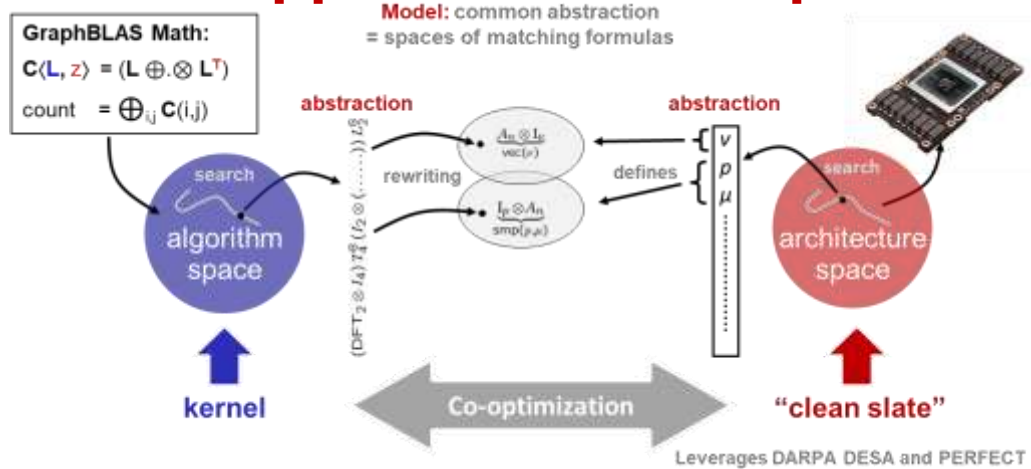
- C++, GraphBLAS calls, other supported libraries
- Code = **specification**, not program

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

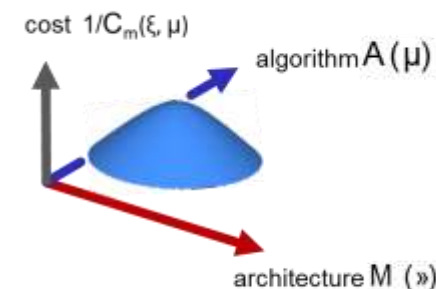
Leverages DARPA BRASS

Formal Approach To Co-Optimization



Algorithm/Architecture Co-Optimization

Design Space



Optimization Problem

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

- 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

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

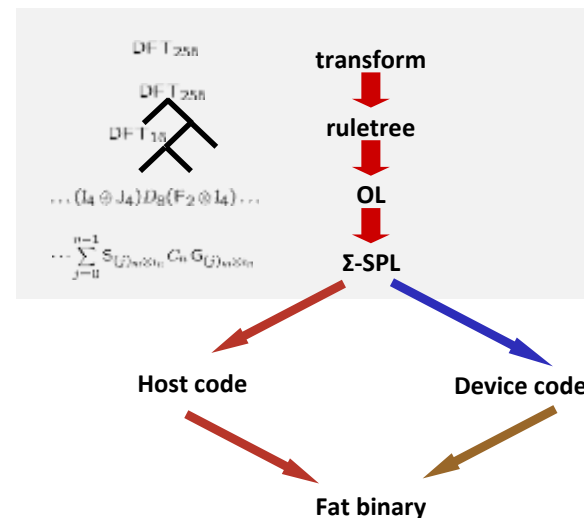
Targeting FPGAs With SPIRAL

Range of Platforms

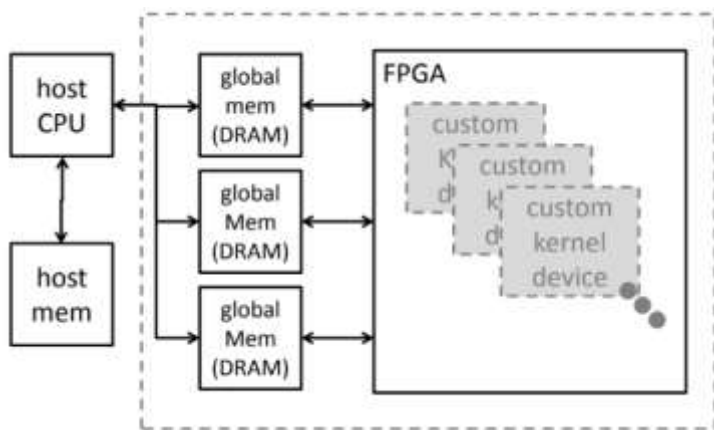


Small dev board
to multi-board server

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(int 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())



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

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];

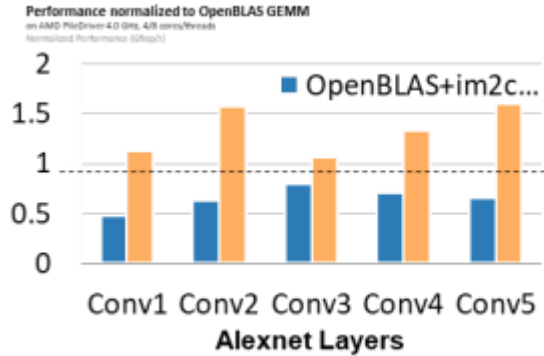
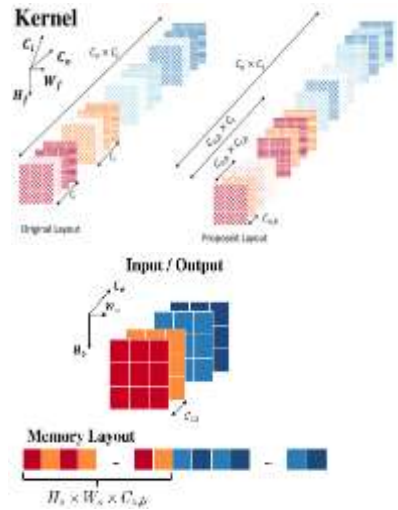
  int iter = 1;
  while (1) {
    int row = 0;
    for (int i = 0; i < Ek; i++) {
      while (IAk[row+1] == i) row++;
      int i0 = IAk[row]; int b0 = IAk[row + 1];
      int i1 = IAk[JAk[i]];
      int b1 = IAk[JAk[i] + 1];
      int res = 0;
      while (i0 < b0 && i1 < b1) {
        int v0 = JAk[i0];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

```

500 lines generated C code

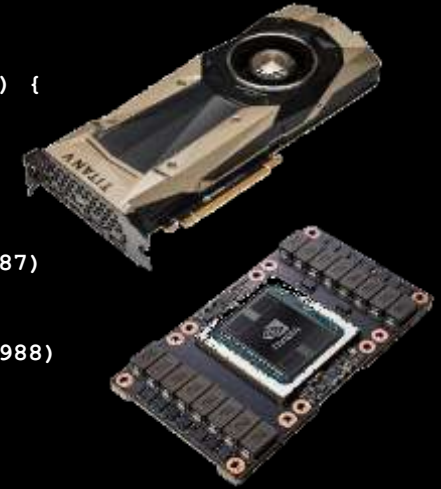
Current Efforts Feeding Into SPIRAL AI/ML

CNN/DNN Code Generation

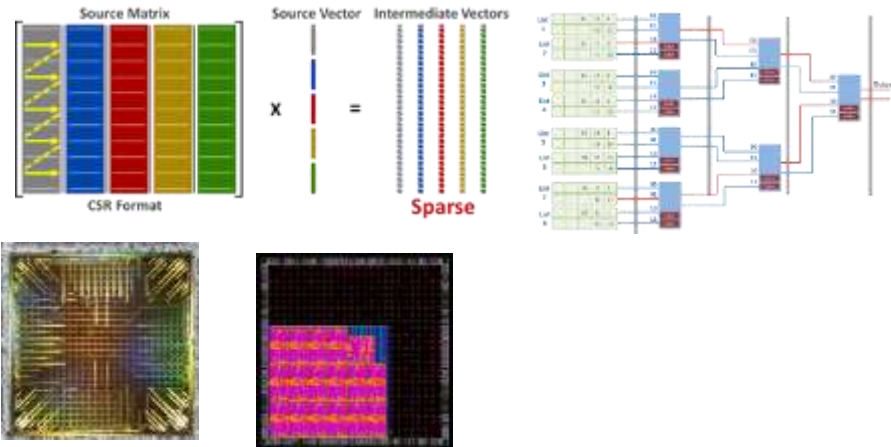


SPIRAL CUDA/OpenACC GPU Target

```
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



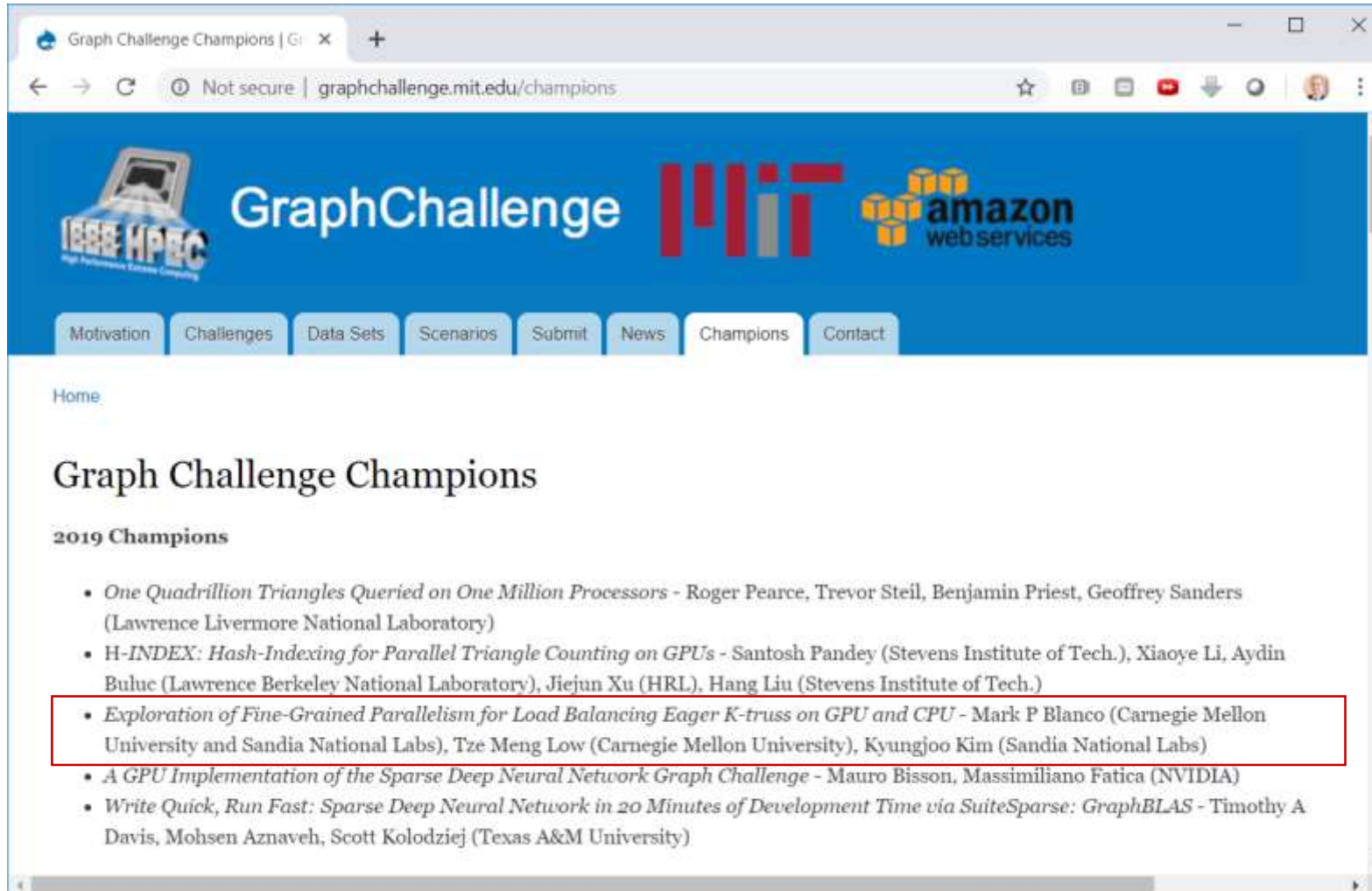
SPIRAL Library as DSL Frontend

```
fftx_plan pruned_real_convolution_plan(fftx_real *in, ...
    ...
    tmp4 = fftx_create_temp_real(rank, &padded_dims);
    plans[3] = fftx_plan_curu_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



Graph Challenge Champions | G | x +

Not secure | graphchallenge.mit.edu/champions

GraphChallenge MIT amazon web services

Motivation Challenges Data Sets Scenarios Submit News Champions Contact

Home

Graph Challenge Champions

2019 Champions

- *One Quadrillion Triangles Queried on One Million Processors* - Roger Pearce, Trevor Steil, Benjamin Priest, Geoffrey Sanders (Lawrence Livermore National Laboratory)
- *H-INDEX: Hash-Indexing for Parallel Triangle Counting on GPUs* - Santosh Pandey (Stevens Institute of Tech.), Xiaoye Li, Aydin Buluc (Lawrence Berkeley National Laboratory), Jiejun Xu (HRL), Hang Liu (Stevens Institute of Tech.)
- *Exploration of Fine-Grained Parallelism for Load Balancing Eager K-truss on GPU and CPU* - Mark P Blanco (Carnegie Mellon University and Sandia National Labs), Tze Meng Low (Carnegie Mellon University), Kyungjoo Kim (Sandia National Labs)
- *A GPU Implementation of the Sparse Deep Neural Network Graph Challenge* - Mauro Bisson, Massimiliano Fatica (NVIDIA)
- *Write Quick, Run Fast: Sparse Deep Neural Network in 20 Minutes of Development Time via SuiteSparse: GraphBLAS* - Timothy A Davis, Mohsen Azaveh, Scott Kolodziej (Texas A&M University)

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

Mark Blanco¹, Tze Meng Low¹
¹ Dept. of Electrical and Computer Engineering
 Carnegie Mellon University
 Pittsburgh, United States
 {markbl, lowt}@cmu.edu

Kyungjoo Kim²
² Center for Computing Research
 Sandia National Laboratories
 Albuquerque, United States
 {kykim}@sandia.gov

Abstract—In this work we present a performance exploration on Eager K-truss, a linear-algebraic formulation of the K-truss graph algorithm. We address performance issues related to load balancing of parallel tasks in symmetric, triangular graphs by presenting a fine-grained parallel approach to executing the support computation. This approach also increases available parallelism, making it amenable to GPU execution. We demonstrate our fine-grained parallel approach using implementations in Kokkos and evaluate them on an Intel Skylake CPU and an Nvidia Tesla V100 GPU. Overall, we observe between a 1.26-1.48x improvement on the CPU and a 9.97-16.92x improvement on the GPU due to our fine-grained parallel formulation.

Index Terms—Graph Algorithms, K-truss, Linear Algebra, Parallelism, High Performance, GPU, CPU, Kokkos, Eager K-truss, Performance Portability

Algorithm 1: Linear algebraic K-truss algorithm. Lower-case letters are vectors and capital letters are matrices.

Input: A is the adjacency matrix of input graph
Output: S represents the support of edges in A

```

1 coverage ← false
2 while not coverage do
3    $S = A^T A + A$  // Step 1: compute support
4    $M = S \geq (k-2)$  // Step 2: pruned edges
5    $A = A \wedge M$ 
6   coverage ← not(isempty(M))
7 return S

```

1. INTRODUCTION

The K-trusses of a graph G are highly connected subgraphs of G where each edge in a subgraph is an edge in at least $K-2$ distinct triangles in the subgraph [1].

In this work, we present a fine-grained parallel implementation of the Eager K-truss algorithm [2], a linear algebraic formulation of an edge-centric K-truss algorithm, on both the CPU and the GPU. The key observation is that the existing Eager K-truss algorithm uses a coarse-grained approach to parallelism by dividing the edges into blocks that are distributed between parallel workers based on the common vertex that the edges are connected to, typically the ‘source’ vertex. As each edge may be connected to different number of neighboring edges (i.e. edges that share the same vertex), the performance of this approach suffers from potential load imbalance. In addition, the Eager K-truss algorithm computes with an upper-triangular adjacency matrix, which means this load imbalance may be skewed significantly as the algorithm proceeds.

We resolve the load imbalance problem by introducing a fine-grained task unit upon which parallel workers compute edge membership in triangles - the edge support values. This, in essence, introduces parallelism within each block of edges assigned in a processing element.

In Sections II and III, we present details of our proposal algorithm. For an efficient implementation, we use a performance portable parallel programming model, Kokkos [3]. In particular, we report our K-truss performance for several Intel K values on NVIDIA V100 and Intel Skylake architectures.

2. BACKGROUND

To keep this paper self-contained, we provide a brief description of the Eager K-truss algorithm in this section. For the detailed algorithmic derivation, we refer to Low et al. [2].

A. Linear Algebraic K-truss Algorithm

Similar to many other K-truss implementations, Eager K-truss takes a two-step approach. Specifically, the first step computes the support of all edges, and the second step prunes edges whose support are below a specified threshold. This two-step approach is then repeated on the pruned graphs until no more edges can be removed.

These two steps of the K-truss algorithm can generally be expressed using linear algebraic notation as described in Algorithm 1, where A is given as the adjacency matrix of the input graph, and M is a binary matrix where $M_{ij} = 1$ when $S_{ij} \geq (k-2)$ [4]. The \wedge operator represents an element-wise multiplication of the input operands. Note that Step 1 computes matrix S , where each value at S_{ij} is the number of triangles containing the edge between nodes i and j .

B. Eager K-truss Algorithm

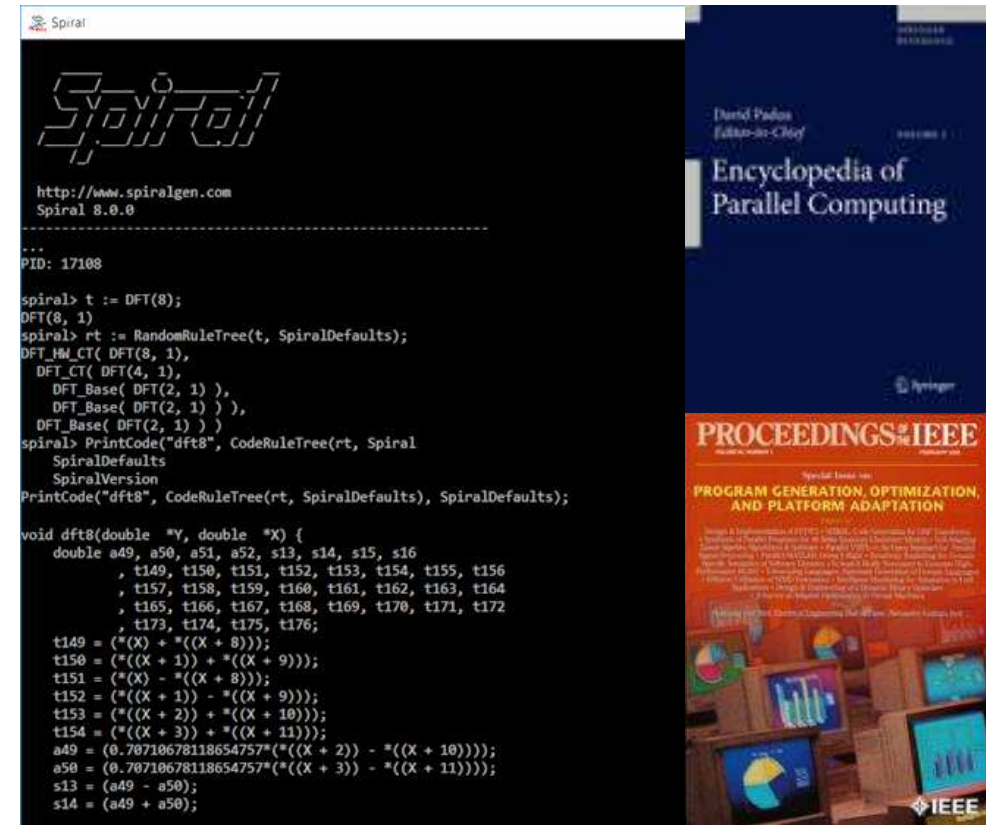
The Eager K-truss algorithm derives its name for the eager manner in which it updates the support values of all edges for each triangle that has been identified. Specifically, the

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



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



- Jeremy Kepner



- Roger Pearce
- Trevor Steil



- Aydin Buluc
- Benjamin Brock



- Carl Yang



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



- Tim Davis