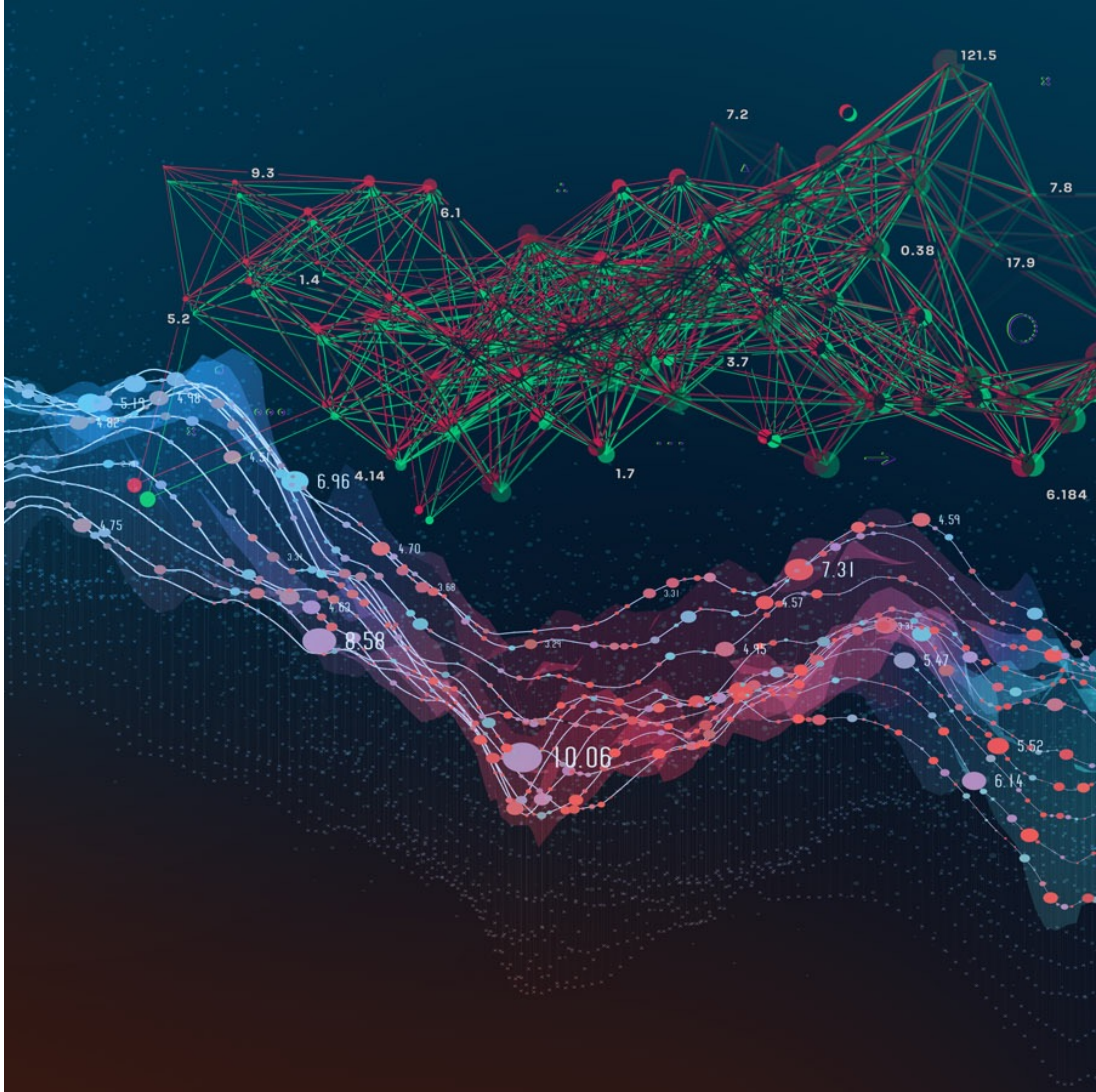
Pacific Northwest NATIONAL LABORATORY

COMET: Domain Specific COMpiler for Extreme Targets in Multi-Level IR

October 25, 2022

Gokcen Kestor, **Rizwan Ashraf**, Zhen Peng, Ruiqin Tian, Luanzheng Guo, Ryan Friese

Pacific Northwest National Laboratory



U.S. DEPARTMENT OF ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

Agenda

Time (CDT)	Topic	Presenter	Duration
9.00-9.10	Logistic	G. Kestor	10 minutes
9.10-9.50	Dense and Sparse Tensor Algebra	G. Kestor	40 minutes
9.50-10.30	Hands-on Session	G. Kestor	40 minutes
10.30-11.00	Break		
11.00-11.10	Kernel Fusion	G. Kestor	10 minutes
11.10-11.20	Semiring Support	R. Ashraf	10 minutes
11.20-11.35	FPGA Codegen	R. Ashraf	15 minutes
11.35-12.15	Hands-on Session	R. Ashraf	40 minutes
12.15-12.30	Conclusions and Q&A	All	15 minutes

COMET Kernel Fusion

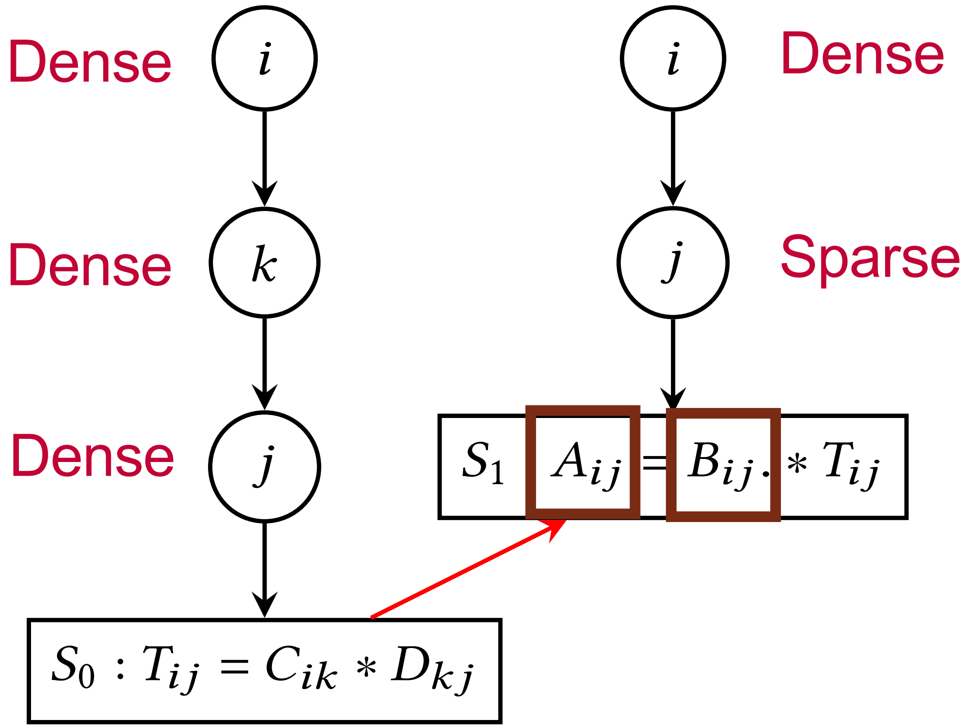
April 4th, 2023

Redundancy-Aware Kernel Fusion[1]

- Compound sparse tensor expressions are commonly used in many domains, including graph analytics, machine learning, and other scientific domain.
- Efficient kernel fusion is needed to eliminate redundant computation
- An expression can be broken into basic operator and each operator can be represented as an index tree

$$A_{ij} = B_{ij} .* (C_{ik} * D_{kj})$$

SDDMM (Sampled Dense-Dense Matrix Multiplication)



[1] ReACT: Redundancy-Aware Code Generation for Tensor Expressions. Tong Zhou, Ruiqin Tian, Rizwan A Ashraf, Roberto Gioiosa, Gokcen Kestor, Vivek Sarkar. 2022 31st International Conference on Parallel Architectures and Compilation Techniques (PACT). October 2022.

Redundancy-Aware Fusion

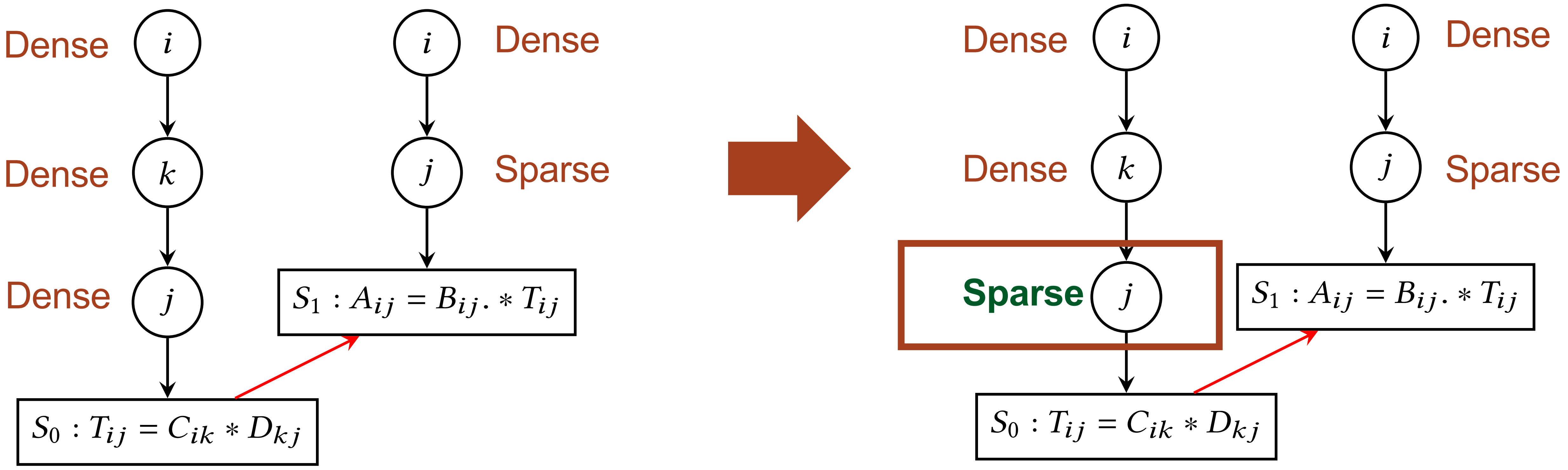
- Two step approach
  1. Propagate sparse iteration spaces
     ✓eliminate dead-value redundancy
  2. Trie-like fusion
     ✓eliminate reduction and loop invariant redundancy

- Memory optimizations
  - Reduce the size of the intermediate tensors wherever possible

Propagate sparse iteration spaces

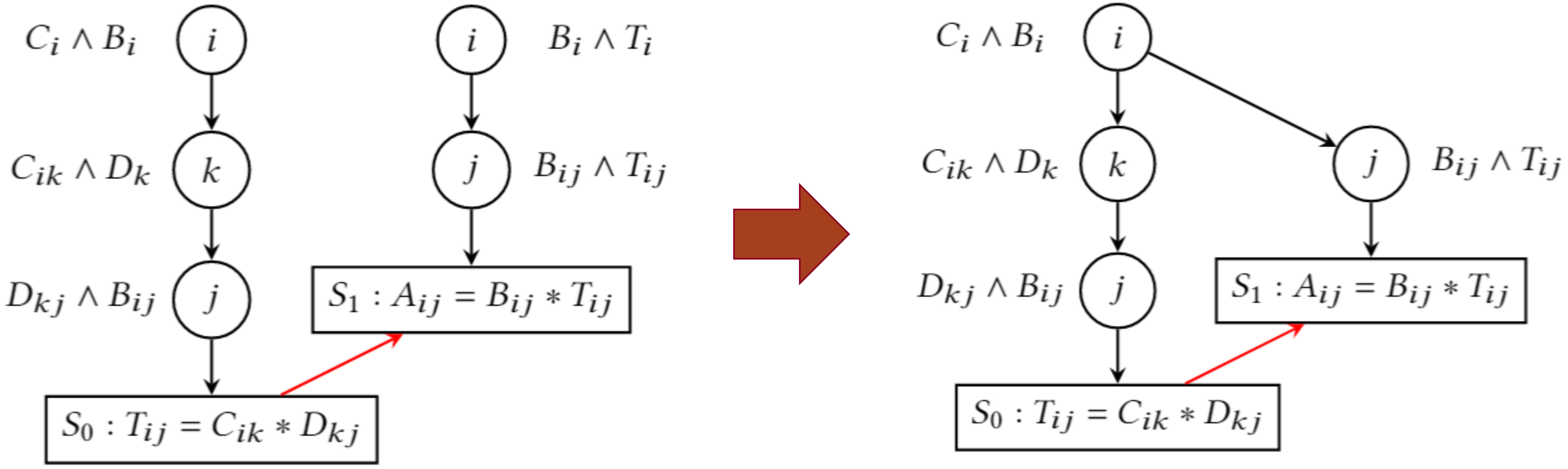
Propagate sparsity from consumer to producer

SDDMM $$A_{ij} = B_{ij} .* (C_{ik} * D_{kj})$$



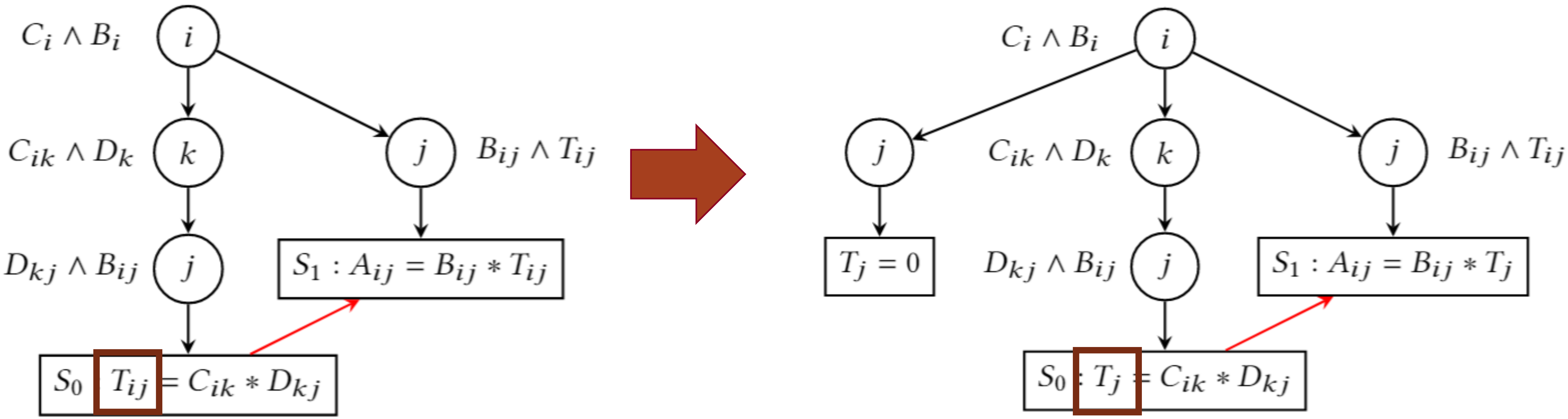
Trie-like fusion

- Fusion is like inserting a key into a trie (prefix tree)



Memory optimization

- Reduce the size of the intermediate tensors wherever possible
- For example SDDMM, we can reduce T_{ij} to an 1-D array

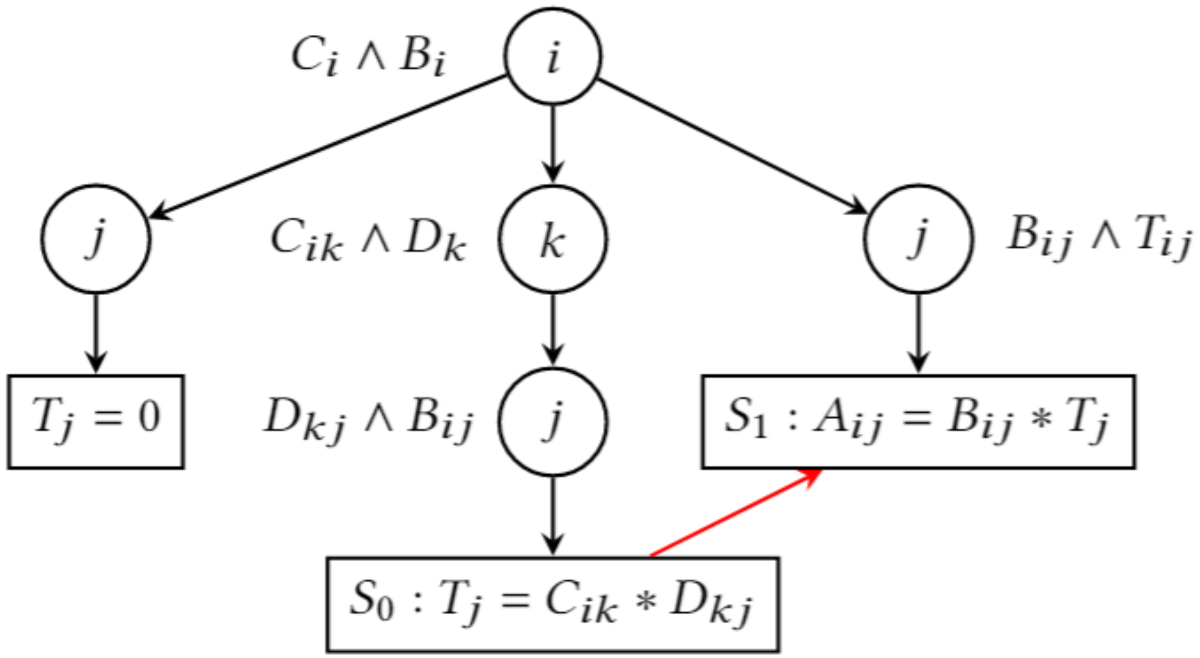


2-D tensor **1-D tensor**

Low-Level codegen

- General loop structure can be generated by traversing the tree
- Generate dense/sparse loop ranges depending on if the loop is dense or sparse

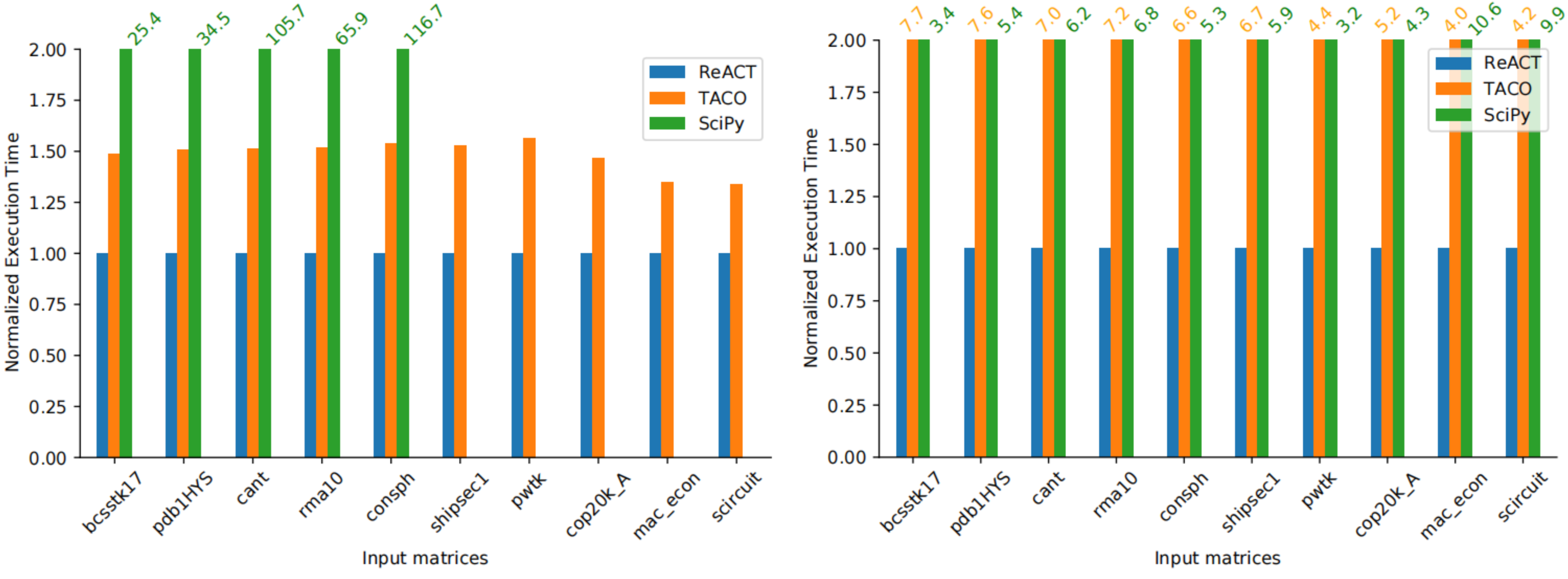
Final index tree




```
auto T = new double[D2_dimension]();
int jT = 0;
for (int32_t i = 0; i < C1_dimension; i++) {
  for (int32_t k = 0; k < D1_dimension; k++) {
    int32_t kC = i * C2_dimension + k;
    jT = 0;
    for (int32_t jB = B2_pos[i]; jB < B2_pos[(i + 1)]; jB++) {
      int32_t j = B2_crd[jB];
      int32_t jD = k * D2_dimension + j;
      T[jT] += C_vals[kC] * D_vals[k * D2_dimension + j];
      jT++;
    }
  }

  jT = 0;
  for (int32_t jB = B2_pos[i]; jB < B2_pos[(i + 1)]; jB++) {
    int32_t j = B2_crd[jB];
    A_vals[jB] += B_vals[jB] * T[jT];
    T[jT] = 0;
    jT++;
  }
}
```

SDDMM and GNN results (16 cores)



(a) SDDMM (NK=64)

(b) GNN-kernel1 (NH=256, NJ=16)

- TACO code generation produces maximal fused code, which introduces redundant computation

COMET Semiring Support

Semiring

- A semiring is an algebraic structure that allows us to perform special operations beyond addition and multiplication to elements in a generic matrix multiplication
- Some constructs found in graph algorithms cannot be represented by standard linear algebra operations but can by semirings
  - Better programming expressiveness
  - Better performance
- Semirings are included in representative libraries like GraphBLAS and Intel MKL

Represent in **semirings** by GraphBLAS:

```
LAGr_mxv (mngp, NULL, GrB_MIN_UINT32,
GxB_MIN_SECOND_UINT32, S, gp, NULL);
```

Connected Components: Hooking & shortcutting

Represent in **normal linear algebra operations** by TACO:

```
// for all edges do hooking and linking
for (int i=0; i < coors.size(); i++) {
    int v_idx = coors[i].x, u_idx = coors[i].y;
    // update u, v:
    v(v_idx) = 1; u(u_idx) = 1;
    // calculate f_u, f_v, ff_u
    f_u(n) = F(n, m) * u(m);
    f_v(p) = F(p, q) * v(q);
    ff_u(x) = F(x, y) * f_u(y);
    if (equall(f_u, ff_u) && larger(f_u, f_v)){
        // hooking - assign f_v to ff_u
        int fu_idx = getF(F, u_idx);
        assignColumn(F, v_idx, fu_idx);
    }
}
// shortcutting
for (int j = 0; j < F.getDimension(1); ++j) {
    int j_f = getF(F, j);
    int j_ff = getF(F, j_f);
    if (j_f != j_ff) {
        assignColumn(F, j_f, j);
    }
}
```

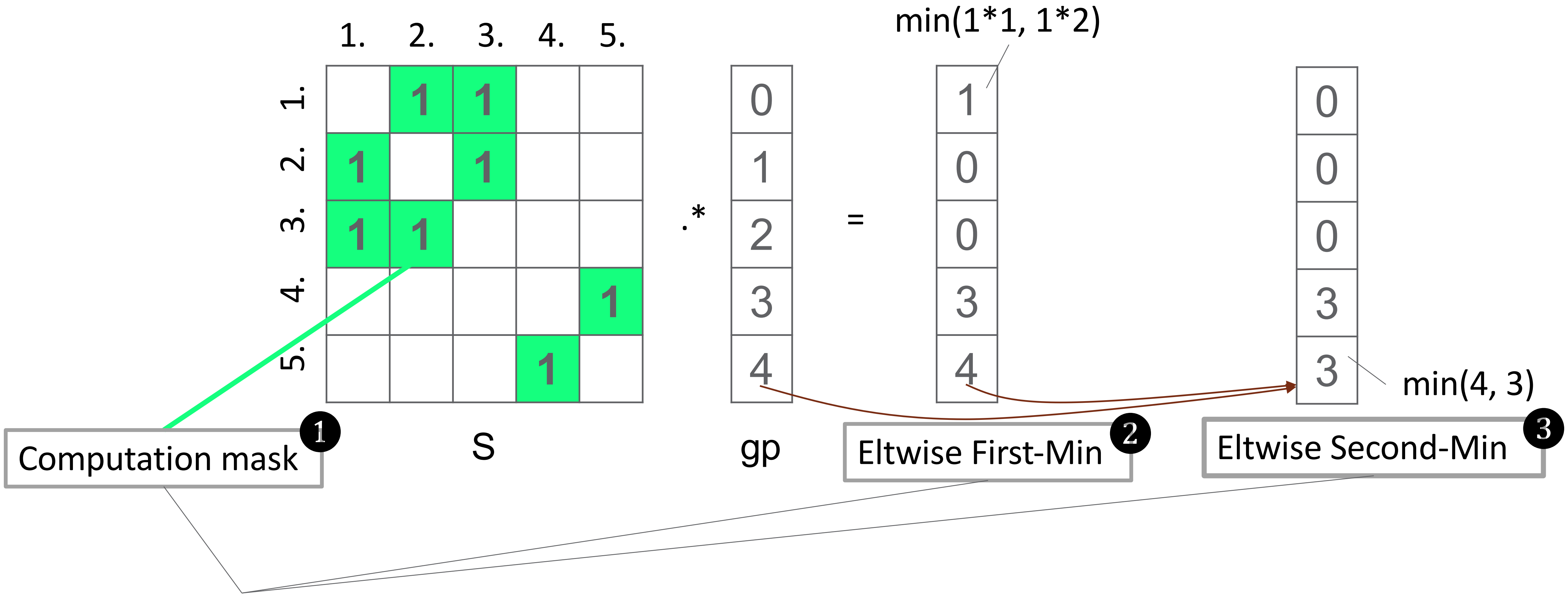
Cannot represent in LA

Cannot represent in LA

Cannot represent in LA

Challenges to represent semirings in LA

- *MIN_SECOND* Semirings cannot be represented in normal Linear Algebra



Cannot be represented in normal LA, and must be represented in special LA

Some semiring examples in DSL

```
def main() {
    #IndexLabel Declarations
    IndexLabel [a] = [?];
    IndexLabel [b] = [?];
    IndexLabel [c] = [?];

    #Tensor Declarations
    Tensor<double> A([a, b], {CSR});
    Tensor<double> B([b, c], {CSR});
    Tensor<double> C([a, c], {CSR});

    #Tensor Data Initialization
    A[a, b] = read_from_file(0);
    B[b, c] = read_from_file(1);

    #PlusTimes semiring
    C[a, c] = A[a, b] @(+,*) B[b, c];
}
```

```
def main() {
    #IndexLabel Declarations
    IndexLabel [a] = [?];
    IndexLabel [b] = [?];


    #Tensor Declarations
    Tensor<double> A([a, b], {CSR});
    Tensor<double> B([a, b], {CSR});

    #Tensor Data Initialization
    A[a, b] = read_from_file(0);
    B[a, b] = read_from_file(1);


    #Min monoid
    C[a, b] = A[a, b] @(min) B[a, b];
}
```

Semiring Operations* in COMET

Semirings	Operation	Explanation
Lor-land	s(|, &)	'lor' means logical OR; 'land' means logical AND.
Min-first	s(min, first)	'min' means the minimal value; 'first' means first(x, y) = x: output the value of the first in the pair.
Plus-times	s(+,x)	'+' means addition; 'x' means multiplication.
Any-pair	s(any, pair)	'any' means "if there is any; if yes return true". 'pair' means pair(x, y) = 1: x and y both have defined value at this intersection.
Min-plus	s(min, +)	'min' means the minimal value; '+' means addition.
Plus-pair	s(+, pair)	'+' means addition; 'pair' means pair(x, y) = 1: x and y both have defined value at this intersection.
Min-second	s(min, second)	'min' means the minimal value; 'second' means second(x, y) = x: output the value of the second in the pair.
Plus-second	s(+, second)	'+' means addition; 'second' means second(x, y) = x: output the value of the second in the pair.
Plus-first	s(+, first)	'+' means addition; 'first' means first(x, y) = x: output the value of the first in the pair.

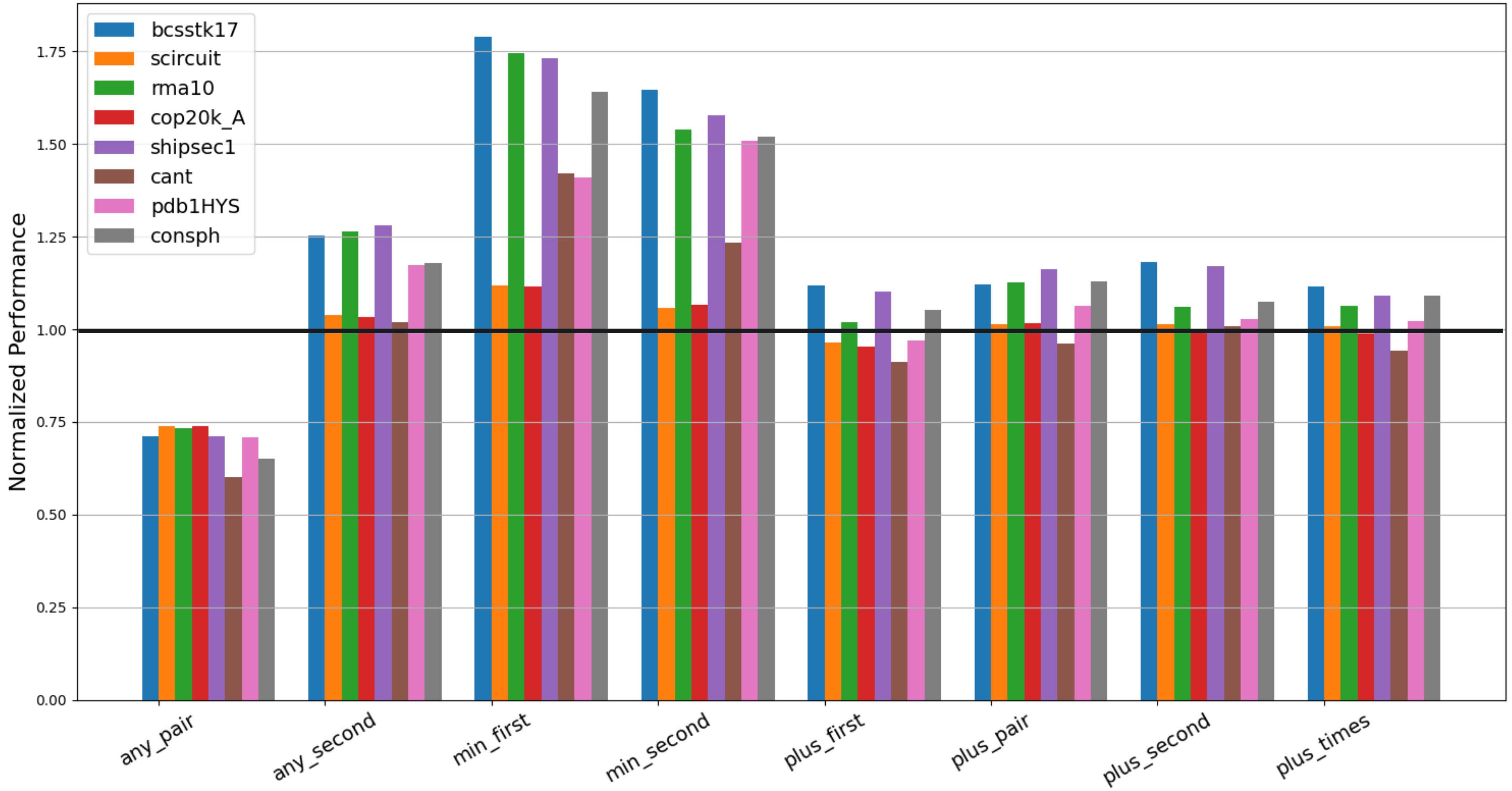
* COMET supports arbitrary combinations of operators for semiring

Semiring Operations per application

	Semirings	Operation	Description
BFS	Lor-land	s(|, &)	Compute traversal level for each vertex
	Min-first	s(min, first)	Compute parent for each vertex
	Plus-mul	s(+,x)	Number of paths
	Any-pair	s(any, pair)	Reachability
	Min-plus	s(min, +)	Shortest path
SSSP	Min-plus	s(min, +)	Shortest path without mask (Bellman-Ford Algorithm)
TC	Plus-pair	s(+, pair)	Number of triangles
CC	Min-second	s(min, second)	Hooking and shortcutting
PR	Plus-second	s(+, second)	Outbound pagerank score
BC	Plus-first	s(+, first)	Accumulate path count

Semiring Performance (unjumbled)

- A method returns a matrix in an *unjumbled* state, with indices sorted
  - if the matrix will be immediately exported in unjumbled form, or
  - if the matrix is provided as input to a method that requires it to not be jumbled

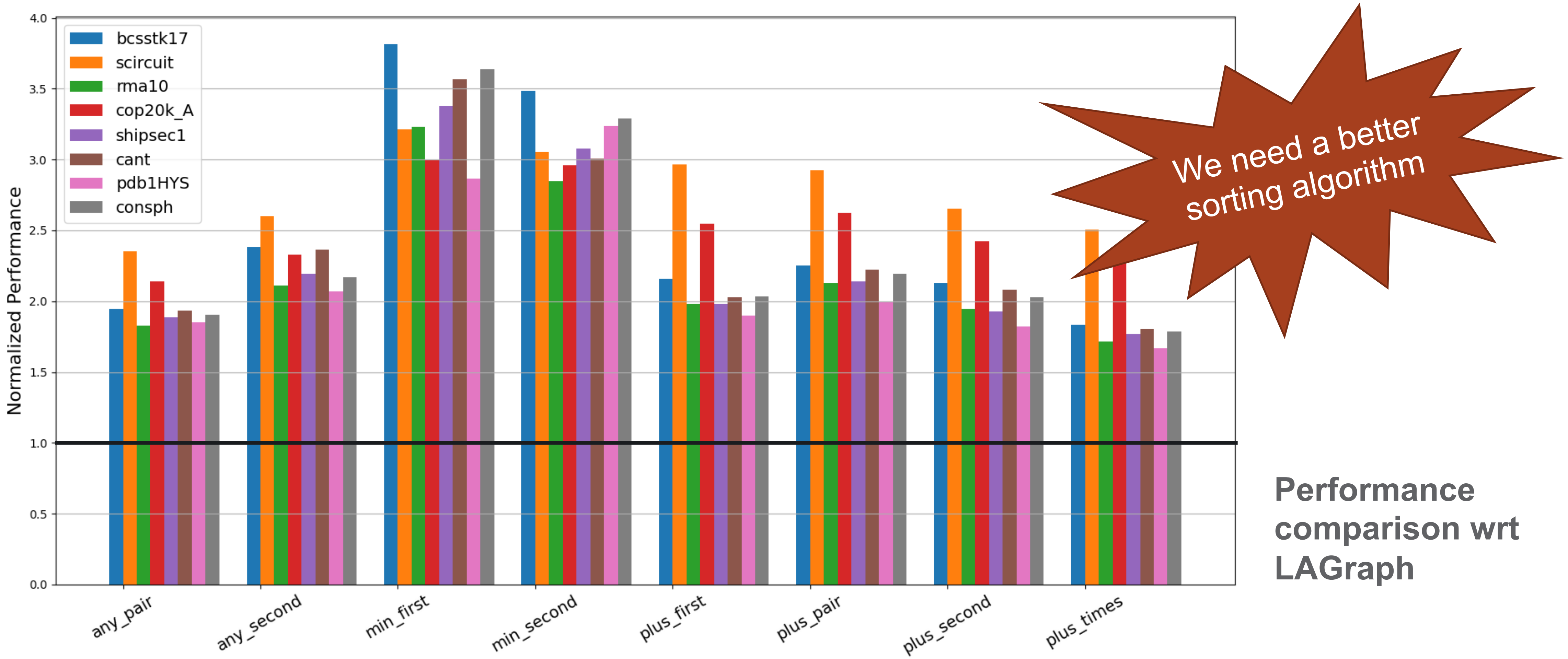


Performance comparison wrt LAGraph[1]

[1] Tim Mattson; Timothy A. Davis; Manoj Kumar; Aydin Buluc; Scott McMillan; Jose Moreira; Carl Yang. "LAGraph: A Community Effort to Collect Graph Algorithms Built on Top of the GraphBLAS", IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2019.

Semiring Performance (jumbled)

- A method returns a matrix in a *jumbled* state, with indices out of order
  - If some methods can tolerate jumbled matrices on input, the sorting of the indices is left pending



Performance comparison wrt LAGraph

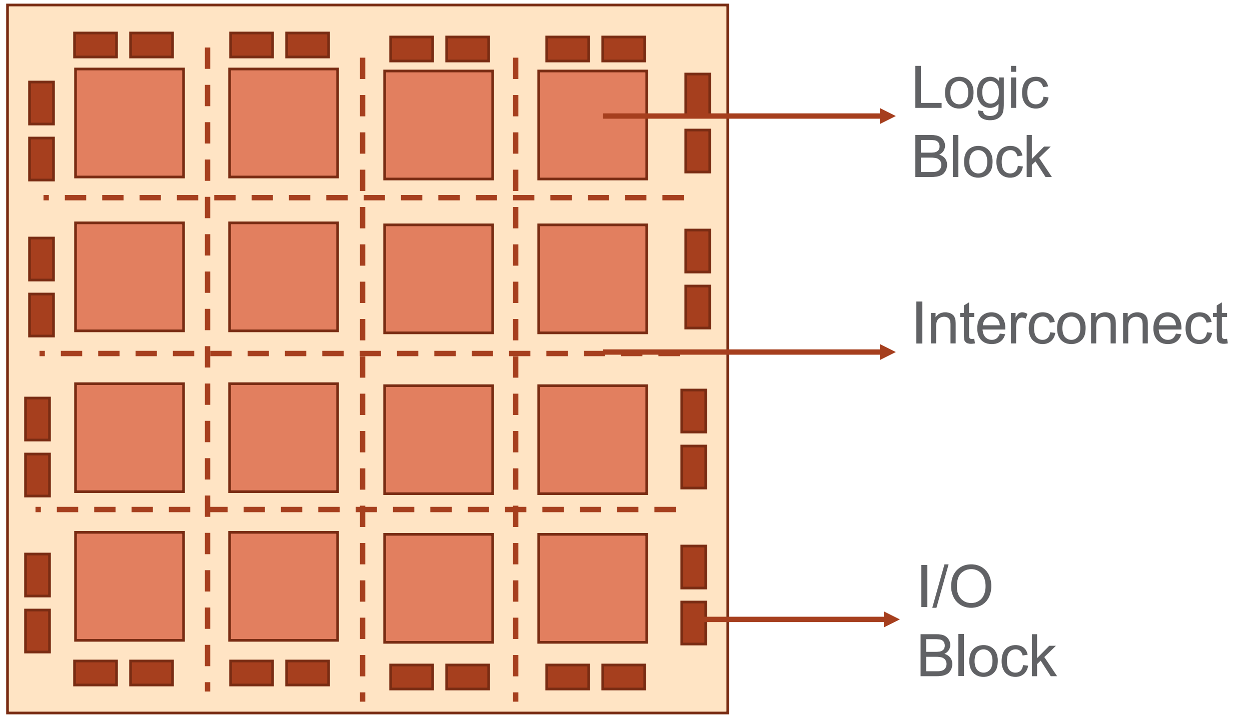
Pacific Northwest
NATIONAL LABORATORY

COMET Heterogenous Target

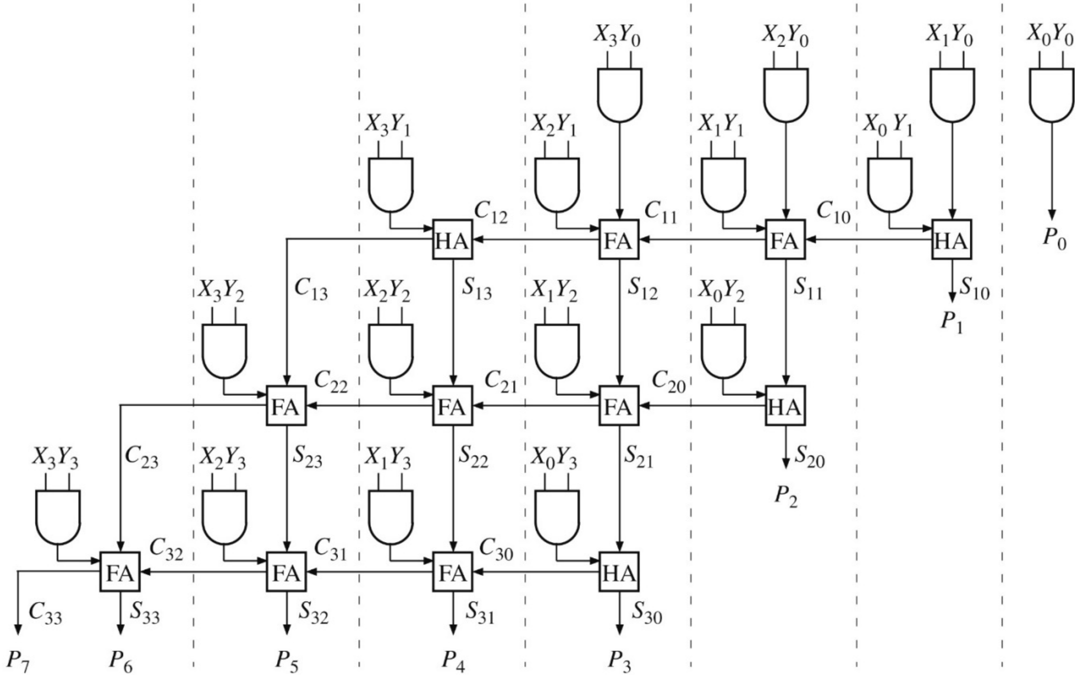
Pacific Northwest NATIONAL LABORATORY

What does an FPGA Accelerator look like?

- Field Programmable Gate Arrays (FPGAs) are reconfigurable hardware devices that can be programmed to provide desired functionality.
  - Challenge: The accelerator design needs to be synthesized (blank slate).
  - Gain: Energy efficient compared to a GPU.
  - Potential: High degree of parallelism.



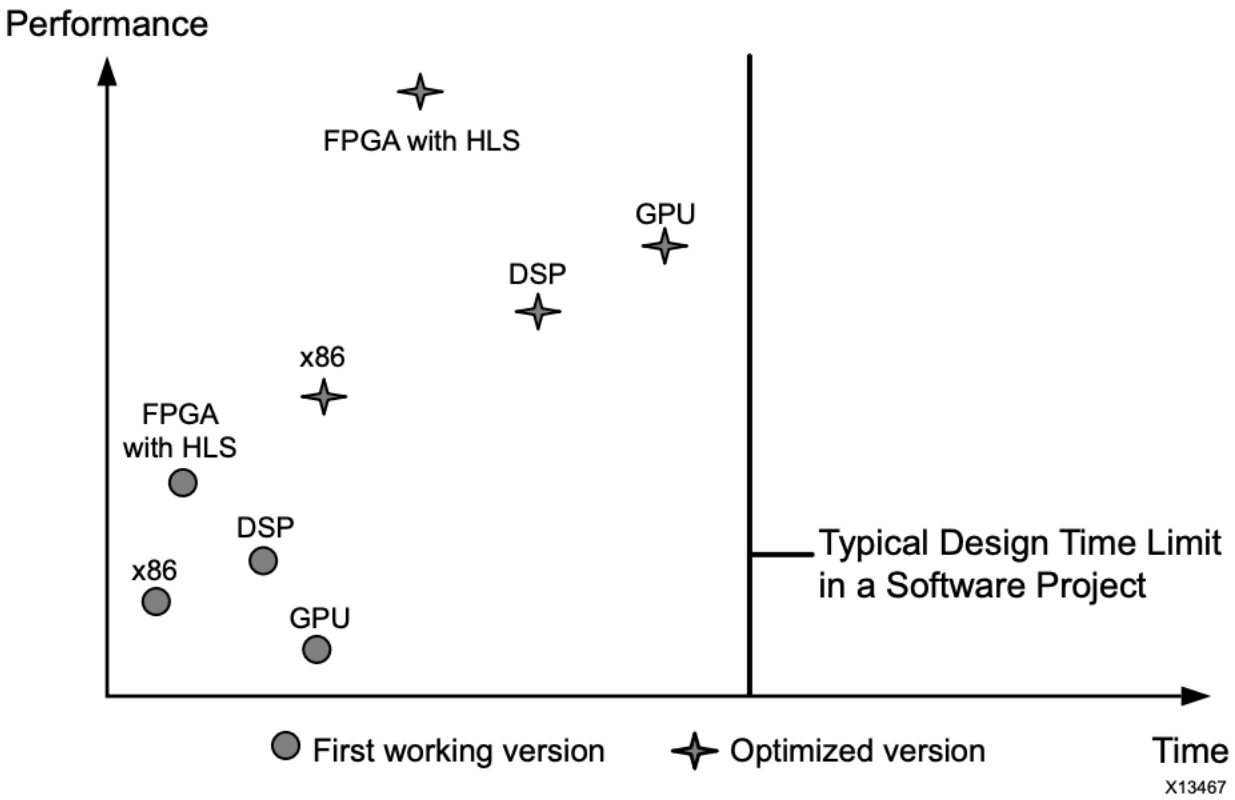
An FPGA Block Diagram



4x4 Array Multiplier (Logic gates and adders)

FPGA Programming Model

- The FPGA programming model is centered on register-transfer level description instead of C/C++ (or other high-level languages).
  - Time required to develop RTL or design for FPGA is high compared to CPU/GPU.
  - The design philosophy is like regular IC than a CPU/GPU.
  - Advances in HLS compiler provides the ability to port C/C++ code to FPGAs.

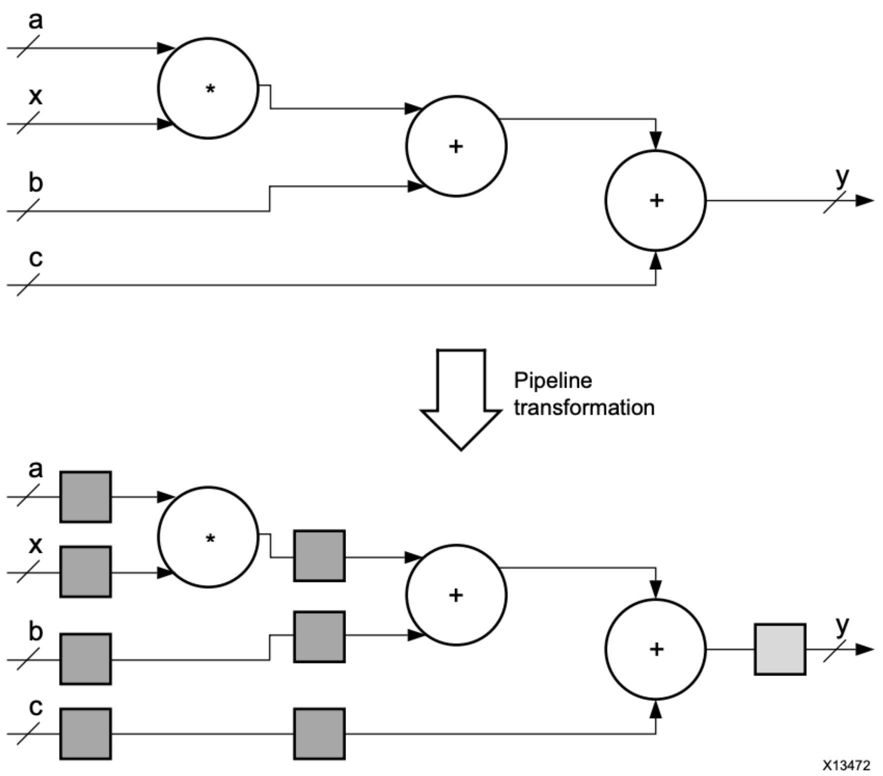


Xilinx Doc: UG998 (v1.1)

FPGA Parallelism

- FPGA is an inherently parallel processing fabric capable of implementing any logical and arithmetic function that can run on a CPU.
  - Scheduling: group operations in same clock cycle.
  - Pipelining: stages run in parallel.
  - Dataflow: parallel execution of functions in a single program.

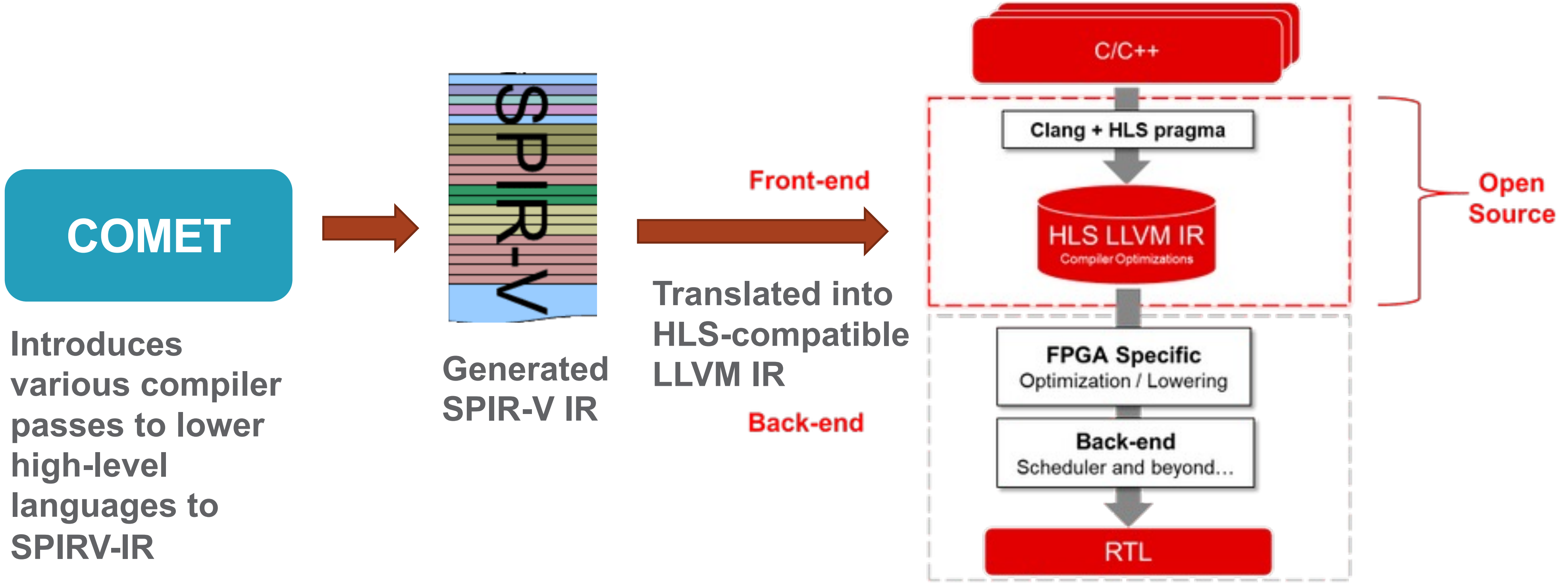
FPGA implementation of
y = (a * x) + b + c



Xilinx Doc: UG998 (v1.1)

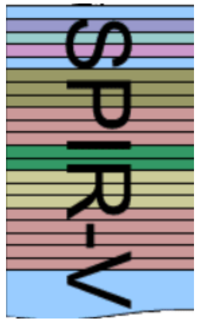
Agile Development of FPGA Accelerators for HPC

- The methodology is based on automated generation of novel hardware concepts starting from a high-level description of the algorithm.



Heterogeneous Devices

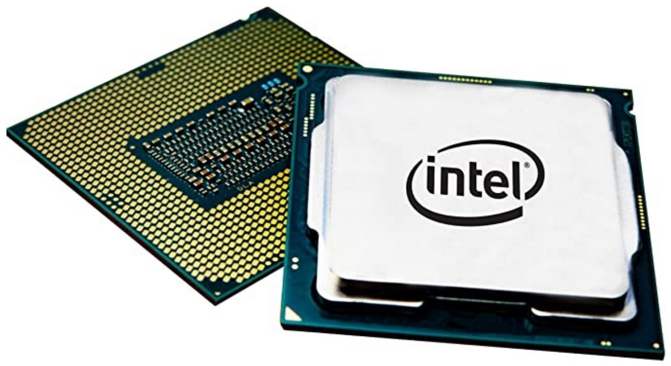
- SPIR-V generated by COMET can be used to program FPGAs, GPUs, and CPUs.



Generated SPIR-V IR

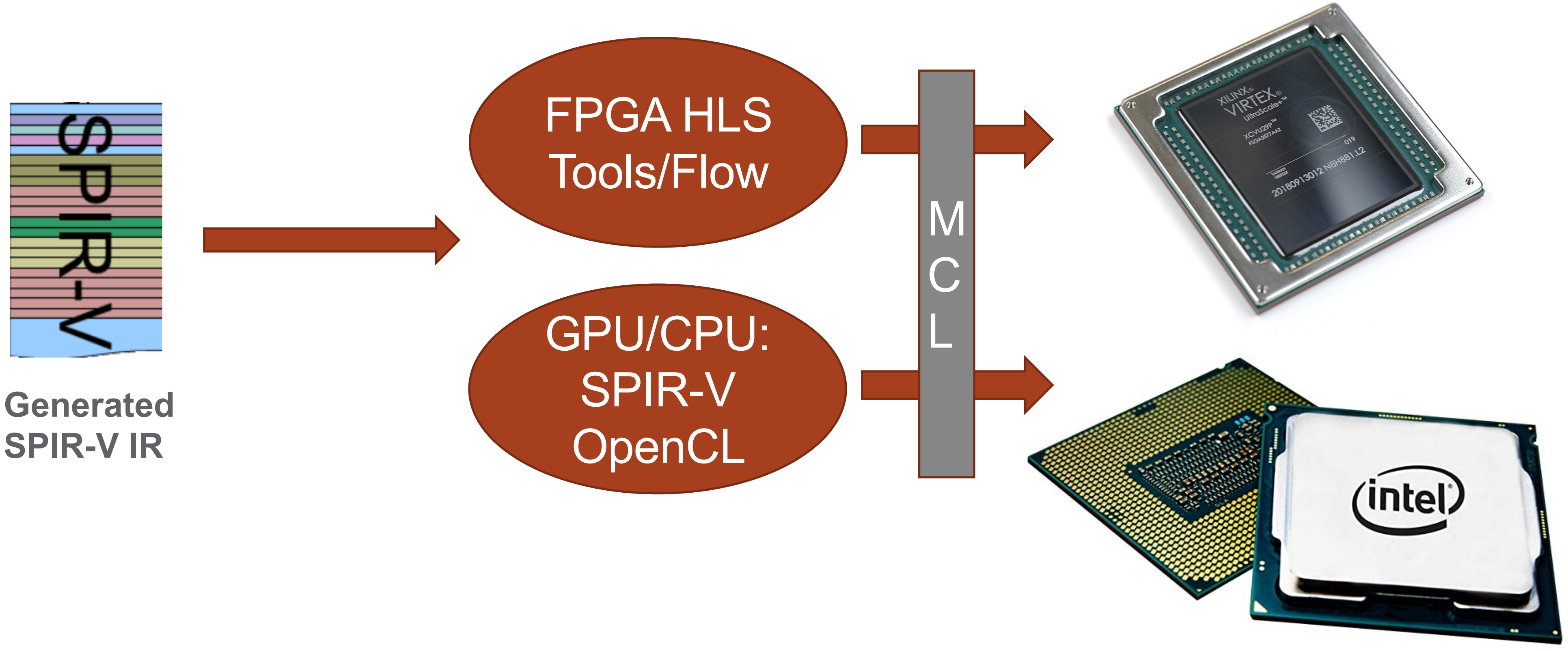
FPGA HLS Tools/Flow

GPU/CPU: SPIR-V OpenCL



Heterogeneous Devices: Minos Computing Lib.

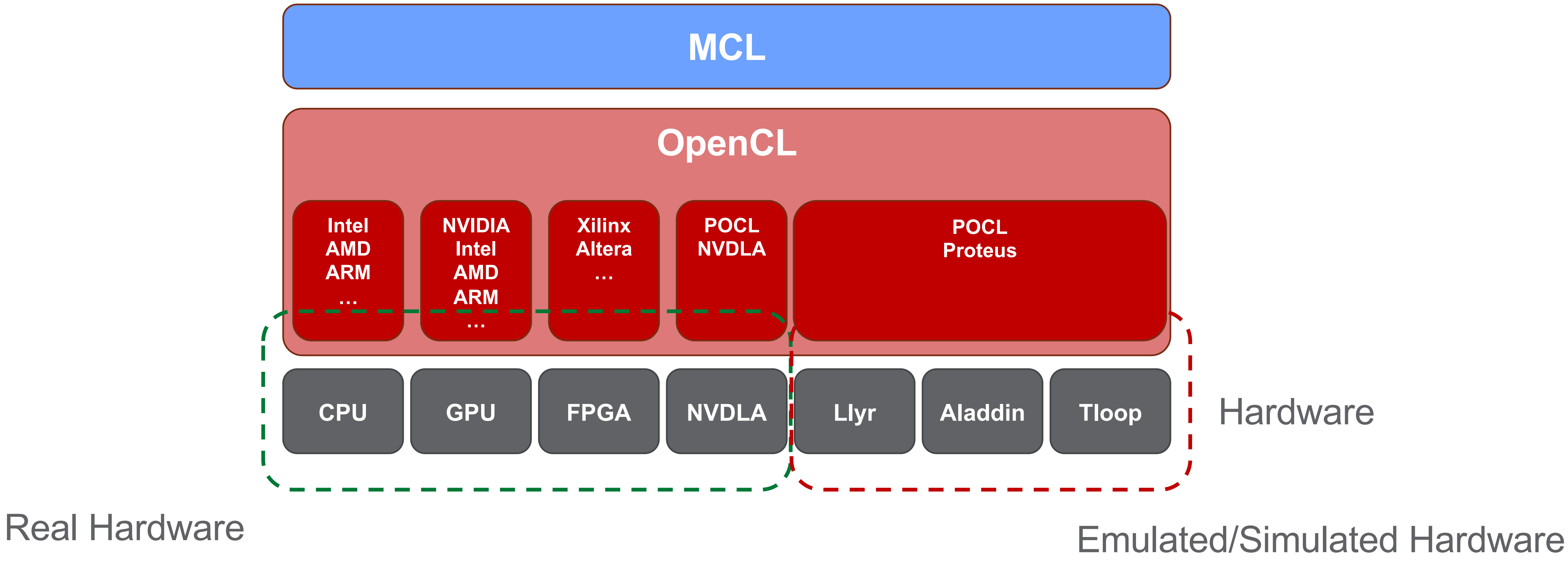
- MCL[1] is an asynchronous task-based programming model and runtime.
  - Orchestrates compute and data among available devices (checks for resources, runtime compilation of kernel code, data locality, power constraints)



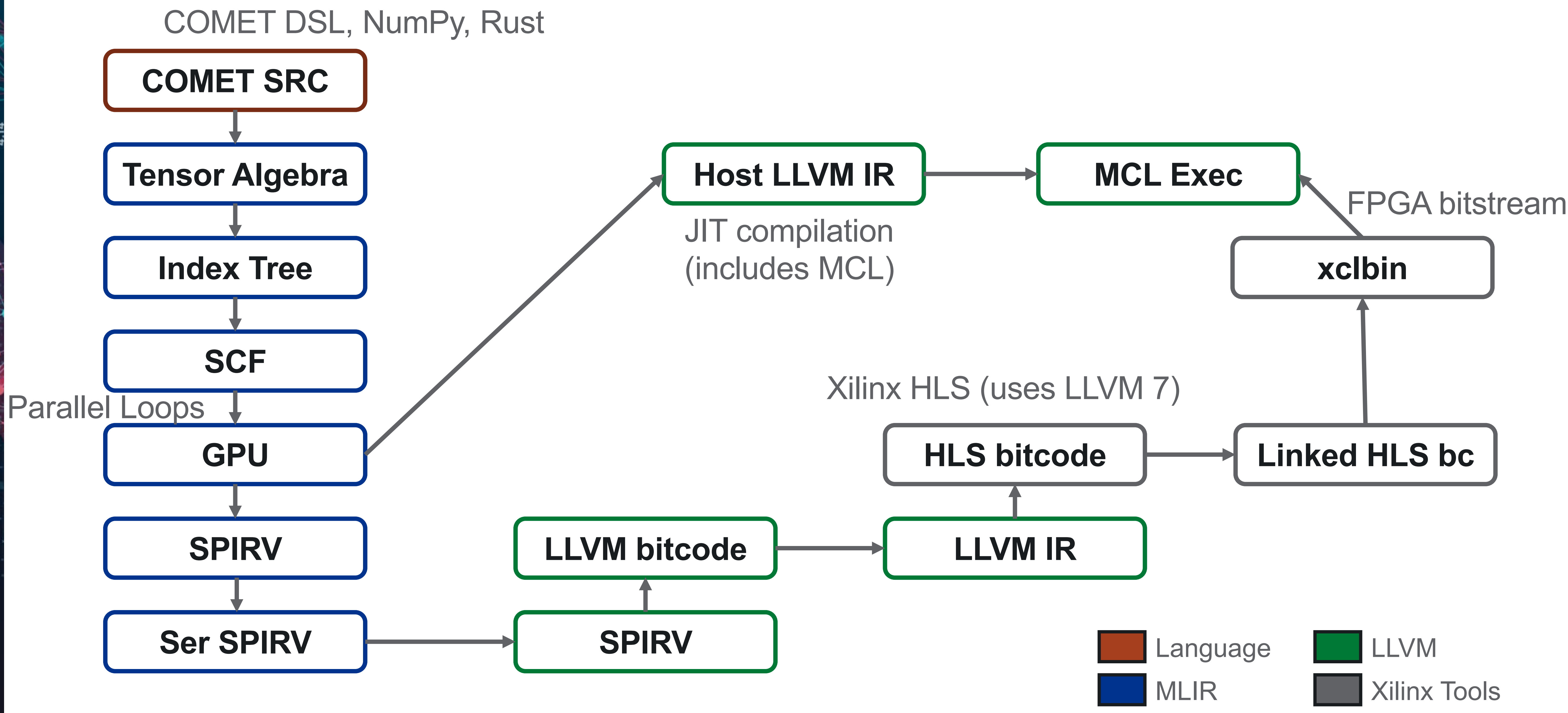
[1] Roberto Gioiosa, Burcu O. Mutlu, Seyong Lee, Jeffrey S. Vetter, Giulio Picierro, and Marco Cesati. 2020. The Minos Computing Library: efficient parallel programming for extremely heterogeneous systems. In Proceedings of the 13th Annual Workshop on General Purpose Processing using Graphics Processing Unit (GPGPU '20)

Minos Computing Library (MCL)

- MCL is based on the OpenCL programming paradigm.
  - Compute-intensive kernels are off-loaded onto accelerators driven from the host.
  - Available as open-source: https://github.com/pnnl/mcl



Workflow for Programming FPGAs Using COMET



COMET Hands-on Session

April 4th, 2022

Case_spgemm: SpGEMM

```
def main() {
    #IndexLabel Declarations
    IndexLabel [a] = [?];
    IndexLabel [b] = [?];
    IndexLabel [c] = [?];

    #Tensor Declarations
    Tensor<double> A([a, b], {CSR});
    Tensor<double> B([b, c], {CSR});
    Tensor<double> C([a, c], {CSR});

    #Tensor Data Initialization
    A[a, b] = read_from_file(0);
    B[b, c] = read_from_file(1);

    #Tensor Contraction
    C[a, c] = A[a, b] * B[b, c];
}
```

- Sparse matrix-sparse matrix multiplication, the output matrix is **sparse**
- Apply workspace transformations

```
$COMET_OPT                  \
    --convert-ta-to-it  \
    --opt-comp-workspace \
    --convert-to-loops  \
    ../benchs/$fname &> ../IRs/$fname-loop.mlir
```

Execution time: 1.083105

Demo: Case_spgemm

```
ops — kest268@bluesky:~/projects/DuoMO/COMET/tutorial/pact22/testcases — ssh bluesky — 102×28
...ps — kest268@bluesky:~/projects/DuoMO/COMET/tutorial/pact22/testcases — ssh bluesky    ~/projects/duomo/COMET/tutorial/pact22/testcases — -zsh
(base) [kest268@bluesky testcases]$
```

Case_spgemm_dense: SpGEMM

```
def main() {
    #IndexLabel Declarations
    IndexLabel [a] = [?];
    IndexLabel [b] = [?];
    IndexLabel [c] = [?];

    #Tensor Declarations
    Tensor<double> A([a, b], {CSR});
    Tensor<double> B([b, c], {CSR});
    Tensor<double> C([a, c], {Dense});

    #Tensor Data Initialization
    A[a, b] = read_from_file(0);
    B[b, c] = read_from_file(1);
    C[a, c] = 0.0;

    #Tensor Contraction
    C[a, c] = A[a, b] * B[b, c];
}
```

- Sparse matrix-sparse matrix multiplication, the output matrix is **dense**
- No need to apply workspace transformations

```
$COMET_OPT                  \
    --convert-ta-to-it      \
    --convert-to-loops      \
    ../benchs/$fname &> ../IRs/$fname-loop.mlir
```

Execution time: 0.506258

Demo: Case_spgemm_dense

```
ops — kest268@bluesky:~/projects/DuoMO/COMET/tutorial/pact22/testcases — ssh bluesky — 102×28
...ps — kest268@bluesky:~/projects/DuoMO/COMET/tutorial/pact22/testcases — ssh bluesky    ~/projects/duomo/COMET/tutorial/pact22/testcases — -zsh
(base) [kest268@bluesky testcases]$
```

Case_f1: GNN without fusion

```
def main() {
    #IndexLabel Declarations
    IndexLabel [i] = [?];
    IndexLabel [k] = [?];
    IndexLabel [j] = [16];
    IndexLabel [h] = [16];

    Tensor<double> B([i, k], {CSR});
    Tensor<double> C([k, h], {Dense});
    Tensor<double> D([h, j], {Dense});
    Tensor<double> A([i, j], {Dense});
    Tensor<double> T([i, h], {Dense});

    #Tensor Data Initialization
    B[i, k] = read_from_file(0);
    C[k, h] = 1.2;
    D[h, j] = 3.4;
    A[i, j] = 0.0;
    T[i, h] = 0.0;


    #GNN
    T[i, h] = B[i, k] * C[k, h];
    A[i, j] = T[i, h] * D[h, j];

}
```

GNN code generation **without fusion**

```
$COMET_OPT                  \
    --convert-ta-to-it      \
    --convert-to-loops      \
    ../benchs/$fname &> ../IRs/$fname-loop.mlir
```

Execution time: 0.056588

Demo: Case_f1


kest268 — kest268@bluesky:~/projects/DuoMO/COMET/tutorial/pact22 — ssh bluesky — 105×29
(base) [kest268@bluesky pact22]$

Case_f2: GNN with fusion

```
def main() {
    #IndexLabel Declarations
    IndexLabel [i] = [?];
    IndexLabel [k] = [?];
    IndexLabel [j] = [16];
    IndexLabel [h] = [16];

    Tensor<double> B([i, k], {CSR});
    Tensor<double> C([k, h], {Dense});
    Tensor<double> D([h, j], {Dense});
    Tensor<double> A([i, j], {Dense});
    Tensor<double> T([i, h], {Dense});

    #Tensor Data Initialization
    B[i, k] = read_from_file(0);
    C[k, h] = 1.2;
    D[h, j] = 3.4;
    A[i, j] = 0.0;
    T[i, h] = 0.0;


    #GNN
    T[i, h] = B[i, k] * C[k, h];
    A[i, j] = T[i, h] * D[h, j];

}
```

GNN code generation **with fusion**

```
$COMET_OPT                   \
     --convert-ta-to-it      \
     --opt-partial-fusion    \
     --convert-to-loops      \
     ../benchs/$fname &> ../IRs/$fname-loop.mlir
```

Execution time: 0.040478

1.39x Speedup

Pacific Northwest
NATIONAL LABORATORY

Demo: Case_f2

```
kest268 — kest268@bluesky:~/projects/DuoMO/COMET/tutorial/pact22/testcases — ssh bluesky — 105×29
(base) [kest268@bluesky testcases]$
```

Case_plustimes: Semiring

```
def main() {
    #IndexLabel Declarations
    IndexLabel [a] = [?];
    IndexLabel [b] = [?];
    IndexLabel [c] = [?];

    #Tensor Declarations
    Tensor<double> A([a, b], {CSR});
    Tensor<double> B([b, c], {CSR});
    Tensor<double> C([a, c], {CSR});

    #Tensor Data Initialization
    A[a, b] = read_from_file(0);
    B[b, c] = read_from_file(1);

    #PlusTimes semiring
    C[a, c] = A[a, b] @(+,*) B[b, c];
}
```

- Plus-times semiring
- Since the output is sparse, need to apply workspace transformations

```
$COMET_OPT                  \
    --convert-ta-to-it  \
    --opt-comp-workspace \
    --convert-to-loops  \
    ../benchs/$fname &> ../IRs/$fname-loop.mlir
```

Case_minfirst: Semiring

```
def main() {
    #IndexLabel Declarations
    IndexLabel [a] = [?];
    IndexLabel [b] = [?];
    IndexLabel [c] = [?];

    #Tensor Declarations
    Tensor<double> A([a, b], {CSR});
    Tensor<double> B([b, c], {CSR});
    Tensor<double> C([a, c], {CSR});

    #Tensor Data Initialization
    A[a, b] = read_from_file(0);
    B[b, c] = read_from_file(1);

    #MinFirst semiring
    C[a, c] = A[a, b] @(min,first) B[b, c];
}
```

- Min-first semiring, used in BFS algorithm
- Since the output is sparse, need to apply workspace transformations

```
$COMET_OPT                  \
    --convert-ta-to-it   \
    --opt-comp-workspace \
    --convert-to-loops   \
    ../benchs/$fname &> ../IRs/$fname-loop.mlir
```

Pacific Northwest
NATIONAL LABORATORY

Thank you