2023 LLVM Developers' Meeting

Quick Talk

MLIR Dialect for GraphBLAS

Sriram Aananthakrishnan

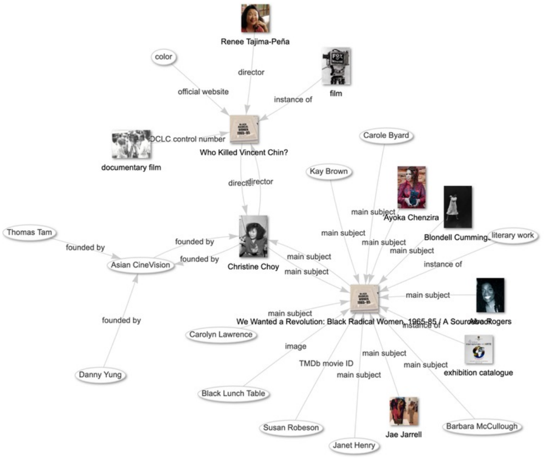
Extreme Scale Computing

Intel

intel.

Graphs Everywhere

Knowledge Graph



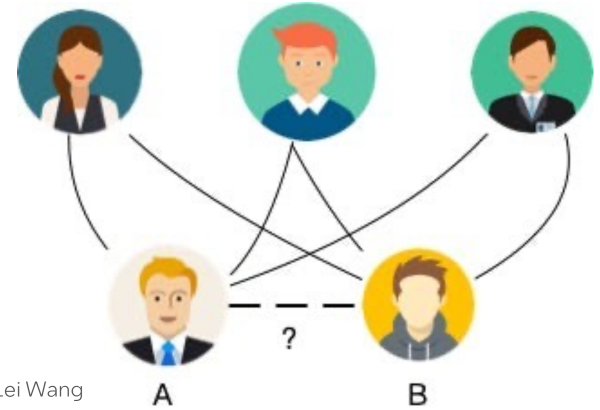
wikidata.org

Social Network

williamjturkel.net

Recommendation Systems

Link Prediction



Credit: Lei Wang

Graph Analytics

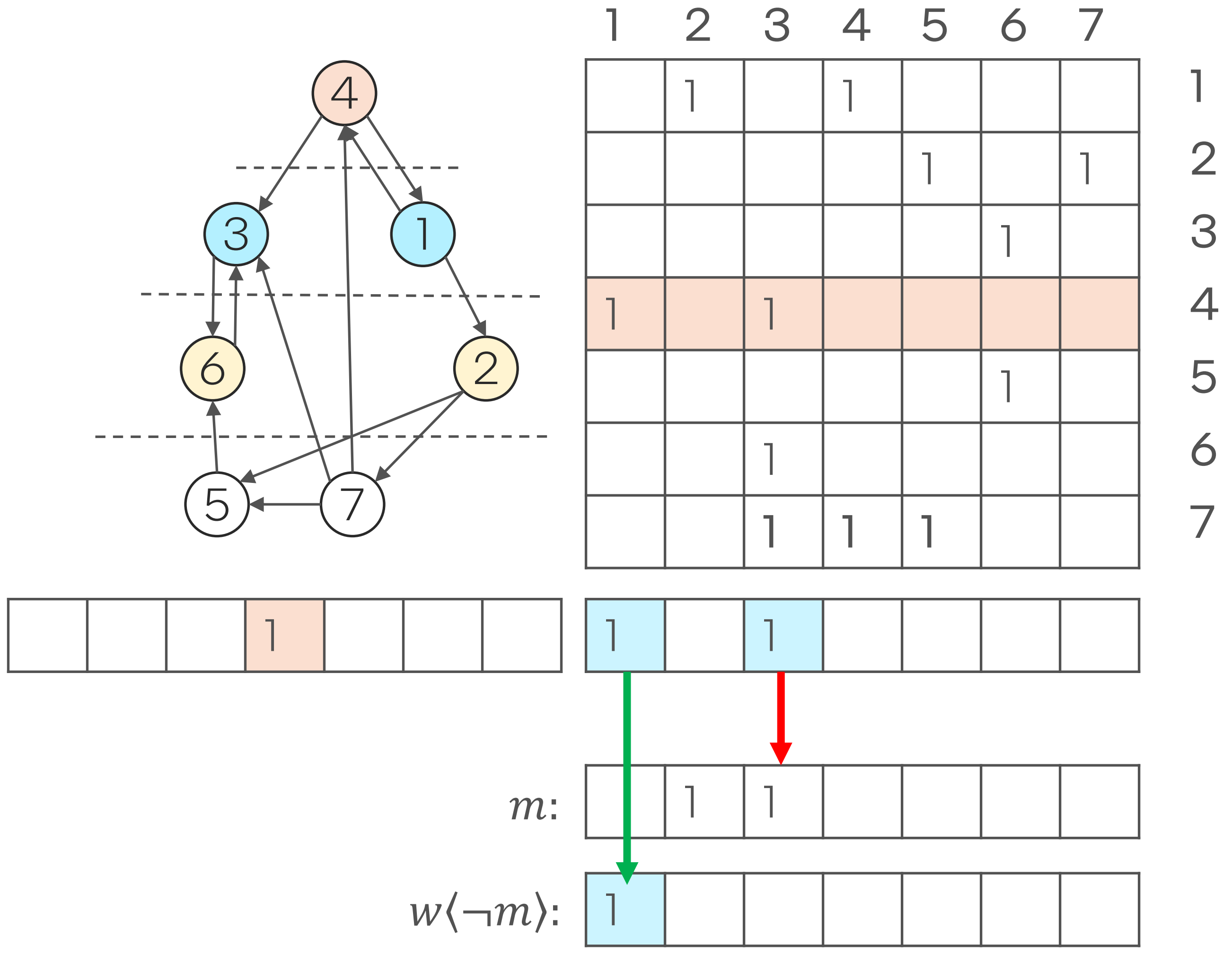
Graph Neural Networks (GNNs)

CPU GPU XPU

- Graphs are unstructured and irregular
- Difficult to parallelize and optimize across multiple platforms

Graph Analysis Using Sparse Linear Algebra

- Graphs as sparse matrices
- Vector-Matrix multiply or Matrix-Matrix multiply
- Use ∧ (⊗) instead of multiply operator and ∨ (⊕) instead of addition operator for a traversal step
- Apply transformations on final output with a write mask and an optional accumulation (⊙) operator



$$w\langle m\rangle = \mathrm{w} \odot \mathrm{v} \oplus\otimes \mathrm{A}$$

GraphBLAS

- Community driven standard
- Building blocks for graph algorithms in the language of linear algebra over algebraic semirings: $(D, \oplus, \otimes, \mathbf{0})$
  - Monoid $\oplus$ is commutative and associative with identity $\mathbf{0}$
  - Binary $\otimes$ is commutative
- Descriptors for altering the semantics e.g., transpose inputs, merge/replace output

https://graphblas.org/

Operation	Mathematical Description
mxm	$C\langle M\rangle = \mathrm{C} \odot A \oplus\!\otimes B$
mxv	$w\langle m\rangle = \mathrm{w} \odot A \oplus\!\otimes v$
vxm	$w\langle m\rangle = \mathrm{w} \odot \mathrm{v} \oplus\!\otimes \mathrm{A}$
eWiseMult	$C\langle M\rangle = \mathrm{C} \odot A \otimes B$
eWiseAdd	$C\langle M\rangle = \mathrm{C} \odot A \oplus B$
reduce	$w\langle m\rangle = \mathrm{w} \odot [\oplus_j A(,:j)]$
apply	$C\langle M\rangle = \mathrm{C} \odot f(A)$ $w\langle m\rangle = w \odot f(u)$
transpose	$C\langle M\rangle = \mathrm{C} \odot A^T$
extract	$C\langle M\rangle = \mathrm{C} \odot A(i,j)$ $w\langle m\rangle = \mathrm{w} \odot u(i)$
assign	$C\langle M\rangle(i,j) = \mathrm{C} \odot A(i,j)$ $w\langle m\rangle(i) = \mathrm{w} \odot u$

GraphBLAS in Academia & Industry

- The GraphBLAS C API Specification Version 2.0 - https://graphblas.org/docs/GraphBLAS_API_C_v2.0.0.pdf
- SuiteSparse:GraphBLAS https://people.engr.tamu.edu/davis/GraphBLAS.html
- Python Bindings
- Integrated into Julia & Matlab
- Industry: NetworkX & FalkorDB

julia

NetworkX
Network Analysis in Python

FalkorDB

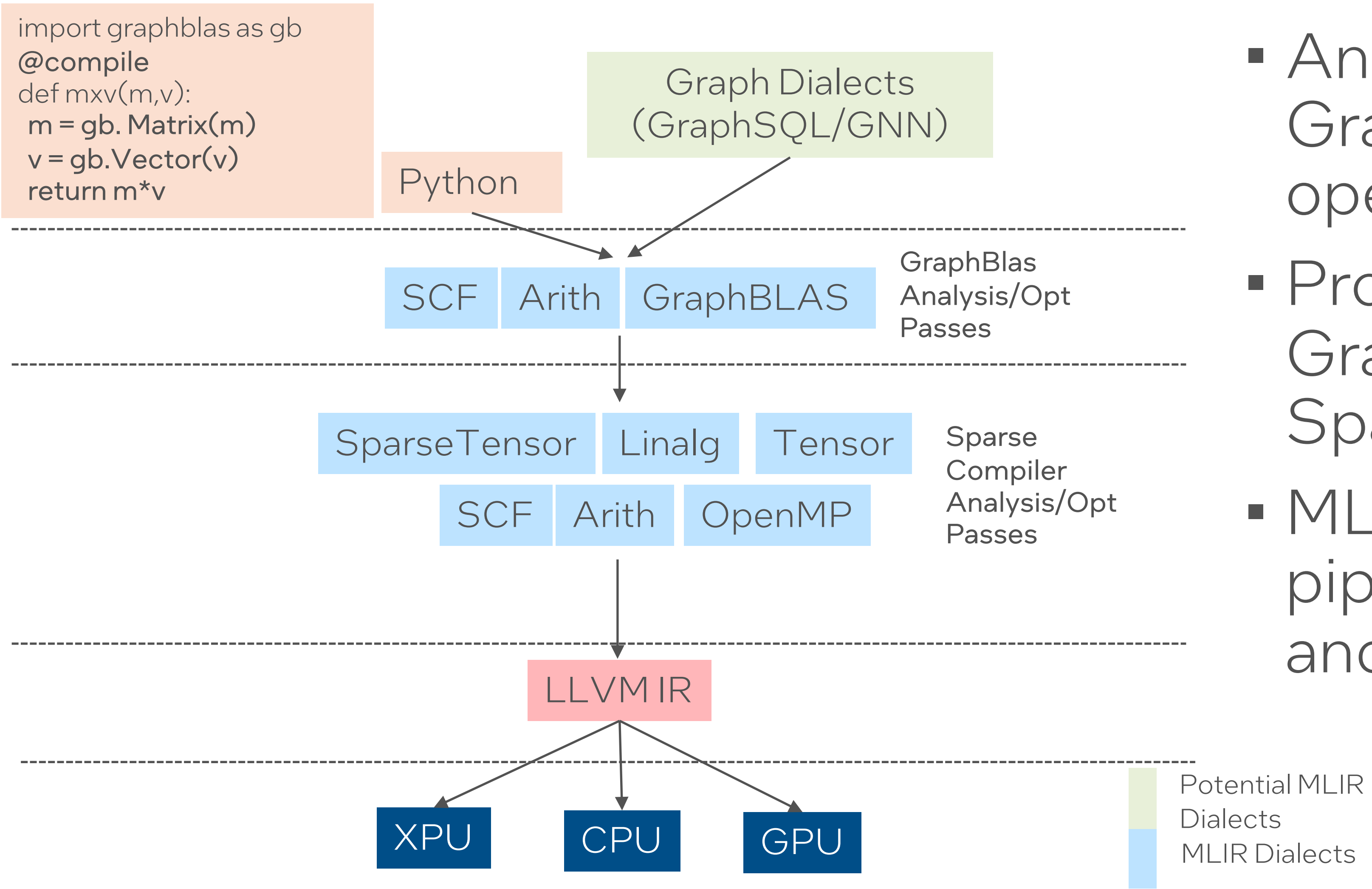
GraphBLAS Compiler

Storage		Types		Semirings		Descriptor		Algorithm		Fusion
CSR		int		$\langle +,\times \rangle$		$mask$		$Gustavson$		$assign - apply$
CSC	x	$float$	x	$\langle \vee,\wedge \rangle$	x	$\neg mask$	x	$dot - \text{prod}$	x	$mxm - \text{reduce}$
$dense$		$double$		$\langle any, sec \rangle$		$replace$		$heap$		$vxm - \text{reduce}$
$blocked$		$complex$		$\langle min, + \rangle$		$transpose$		…		…
4^4		…		$\langle +, sec \rangle$		…				…
		11^3		…		32				
				960						

$mxm\,(C\langle M \rangle = A \oplus \otimes B)$ Variants: $256 \times 11^3 \times 960 \times 32$

Algorithm 1000: SuiteSparse:GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra

GraphBLAS MLIR Dialect



- Analyze and optimize GraphBLAS DAG of operations
- Progressively lower GraphBLAS ops to SparseTensor/Linalg ops
- MLIR sparse compiler pipeline for CPU, GPU and XPU

Anatomy of a GraphBLAS Op

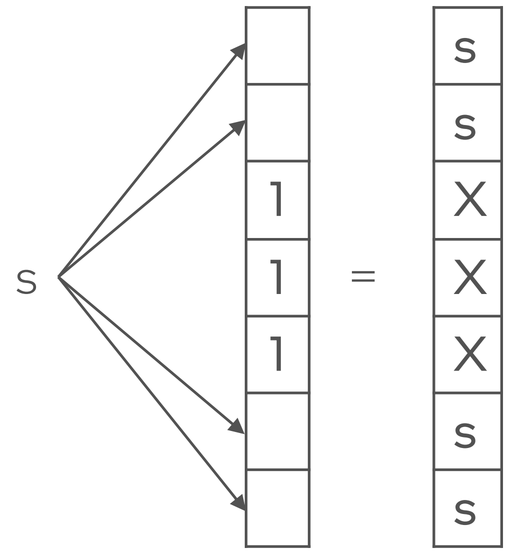
semiring descriptor

```
%w = grb.vxm <land, lor> #grb.desc<RC> %u, %A, %m:
(tensor<?xi64, #SV>,
 tensor<?x?xi64, #CSR>,
 tensor<?xi64, #SV>) →
 tensor<?xi64, #SV>
```

Inputs: %u, %A, (Optional) %m
Output: %w
Semiring: <land, lor>
Descriptor: #grb.desc<RC>

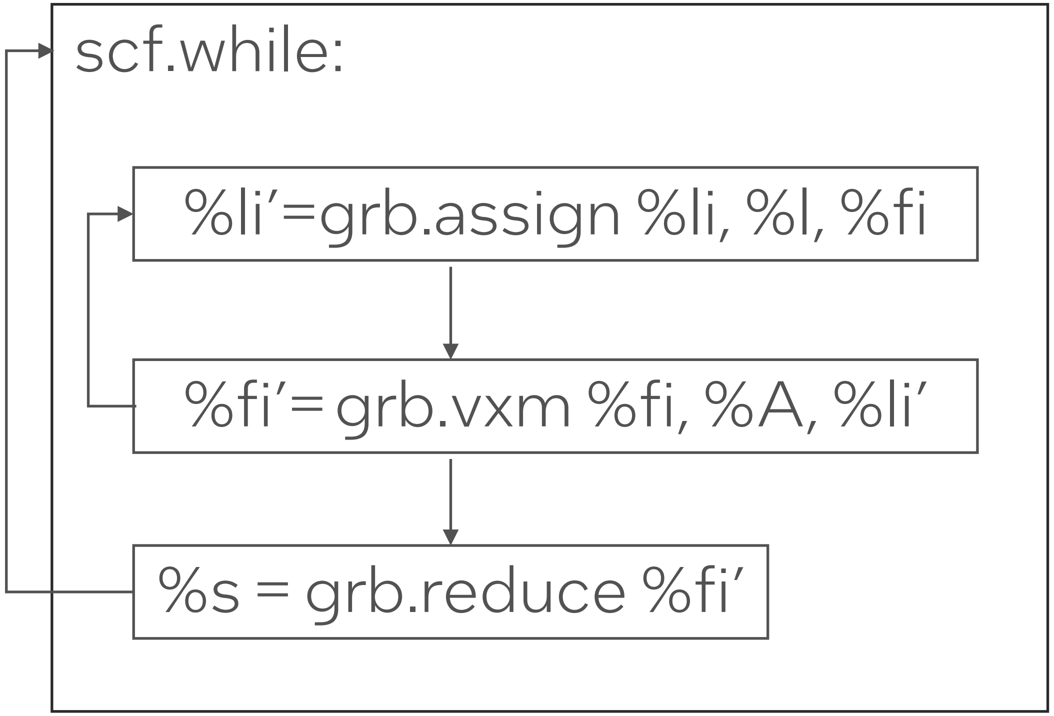
Progressive Lowering

```
%l = grb.assign #grb.desc<C>%l, %d, %m :
(tensor<?xi64, #SV>, i64, tensor<?xi64, #SV>) → tensor<?xi64, #SV
```




```
%l = linalg.generic #attr
 ins(%m) outs(%l)
^bb(%mi,%li):
   %u = sparse_tensor.unary %mi
    present {}
    absent{
     sparse_tensor.yield %d
    }
   linalg.yield %u
}
```

Operator Fusion



- Opportunity to fuse $vxm - reduce$
- Optimize memory when fusing $mxm - reduce$ when output matrix only use is $reduce$

MLIR GraphBLAS : Current Status

- MLIR representation for a subset of operations in GraphBLAS dialect
- Progressive lowering of GraphBLAS ops to Linalg and SparseTensors
- Lowering focuses on semirings, mask, ¬mask and sparse tensor dialect handles multiple storage formats
- Sparse compilation pipeline to OpenMP/LLVM
- End-to-end code generation for Breadth-First Search (BFS)

Learnings & Future Directions

- Learnings
  - Progressive lowering minimizes burden by allowing top level to focus on algorithm and GraphBLAS specific variants
  - Builder design pattern for Linalg/SparseTensor

- Future Directions
  - Expand support to all operations in GraphBLAS standard
  - Operator fusion
  - Vectorization for CPUs
  - GPU/XPU code generation

intel®