AI Compiler

영남대학교
차세대 컴퓨터 시스템 연구실
석사과정생 이원호

MLIR: Scaling compiler infrastructure for domain specific computation

2021 IEEE/ACM International Symposium
on Code Generation and Optimization (CGO). IEEE, 2021

차세대 컴퓨터 시스템 연구실

이원호

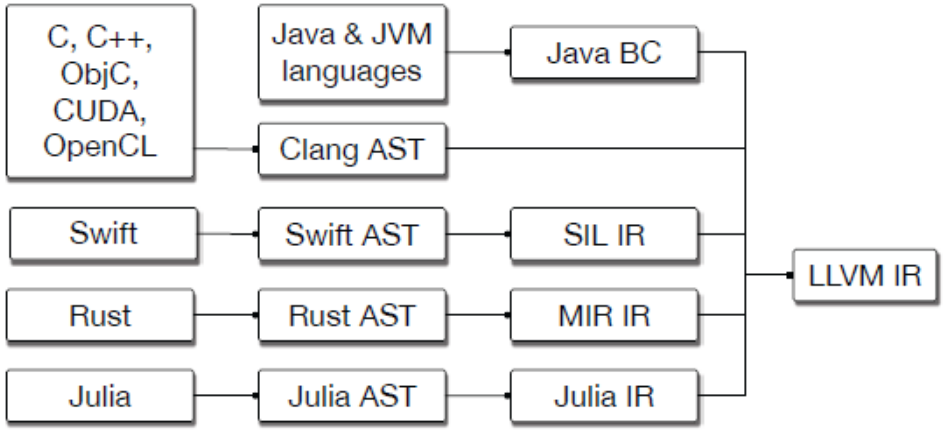
Previous works in compiler

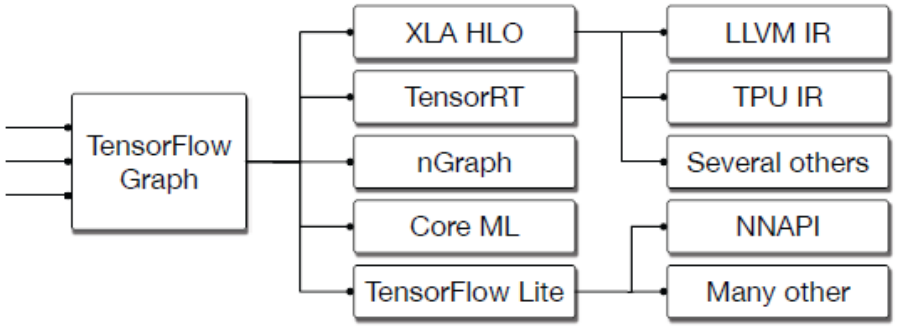
- **One size fits all**
  - 시스템에 접근하기 위해 single abstraction level을 활용
  - LLVM
    - C with vectors
  - JVM
    - Object-Oriented type system with a garbage collector
  - 더 상위 혹은 하위 레벨에서의 추상화가 더 나은 모델링을 제시하는 경우가 존재

Domain-specific IRs

- 많은 언어들 (Swift, Rust, Julia, Fortran 등)에서 특정 작업을 위해 고유 IR을 설계함

- 일반적으로 머신 러닝 시스템에서는 ML graph라는 특수한 추상화 방식을 채택함





Previous works in compiler

- Domain-specific IR 들은 항상 우선순위를 보장받을 수 없기 때문에 컴파일러의 성능을 저하시킴
  - 느린 컴파일 속도 (slow compile times)
  - 많은 구현 상의 버그 (buggy implementations)
  - 준최적 진단 품질 (suboptimal diagnostic quality)
  - 최적화 코드에 대한 부족한 디버깅 경험
    (poor debugging experience for optimized code)

MLIR design principle

- Parsimony
  - semantics, concepts, programming interface를 구성하는 데 Occam's razor 원칙 적용
  - operation, types 의 속성을 추상화해 본질적, 부수적 복잡성을 모두 활용
- Traceability
  - 정보 복구보다 정보 유지를 하려는 성질
  - 가능한 변환에 대해 단계적인 변경사항보다 규칙과 속성을 정의
- Progressivity
  - Lowering을 정해두는 게 domain-specific한 컴파일에 문제를 일으킬 수 있음
  - 요구에 따라 개인 영역을 lowering할 수 있는 다양한 transformation path를 허용
  - 이런 부분을 여러 도메인에서 재사용 가능하게 함

MLIR project

- 새로운 추상화 레벨 정의 비용을 줄이는 것에 집중
  - Static Single Assignment(SSA) 기반의 IR 데이터 구조
  - IR dialects 를 정의하기 위한 선언 시스템 제공
  - 일반적인 컴파일러가 지원하는 인프라 제공
    - 문서화
    - 구문 분석
    - 경로 관리
    - 멀티 스레드 컴파일 지원 등

Contribution

- scalable and modular compiler system의 설계 문제를 해결할 수 있음
  - Domain specific IR에 대한 선언 비용을 줄임
- 새로운 컴파일러 인프라를 통해 설계의 비용을 줄일 수 있음
  - Domain specific IR을 쉽게 활용할 수 있기 때문
- 여러 영역으로의 적용에 대한 탐색을 제공할 수 있음
  - Domain specific IR을 쉽게 활용할 수 있기 때문

Union: A unified HW-SW co-design ecosystem in MLIR for evaluating tensor operations on spatial accelerators

2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT). IEEE, 2021.

차세대 컴퓨터 시스템 연구실

이원호

Problems

- 딥러닝의 계산 요구량의 증가
  - Dataflow accelerator에 대한 수요 증가

- Domain specific accelerator는 완전히 프로그래밍할 수 없음
  - Data orchestration에 대해 여러 레벨에서 유연성을 가짐
    - Dataflow optimization 과 tiling optimization을 통해 성능 향상

- 이런 특징들로 인해 새로운 알고리즘이나 매핑 정책을 새로운 하드웨어에 적용하려 할 때 문제가 발생
  - 기존 연구들은 이를 각각 해결하려 함

Problems

- 새로운 알고리즘, 매핑 정책을 새로운 하드웨어에 적용할 때 고려할 점
  - Algorithm / Workload
  - Mapping
  - Hardware

- 기존의 연구들은 이런 요소들이 밀접하게 연결된 상태로 간주하고 하나의 문제에 대해서만 독자적으로 개선하려 함

Union

- 기존에 밀접하게 연결되어 있던 세 요소에 대해 추상화를 제공
  - Unified workload abstraction
  - Unified mapping abstraction
  - Unified hardware abstraction

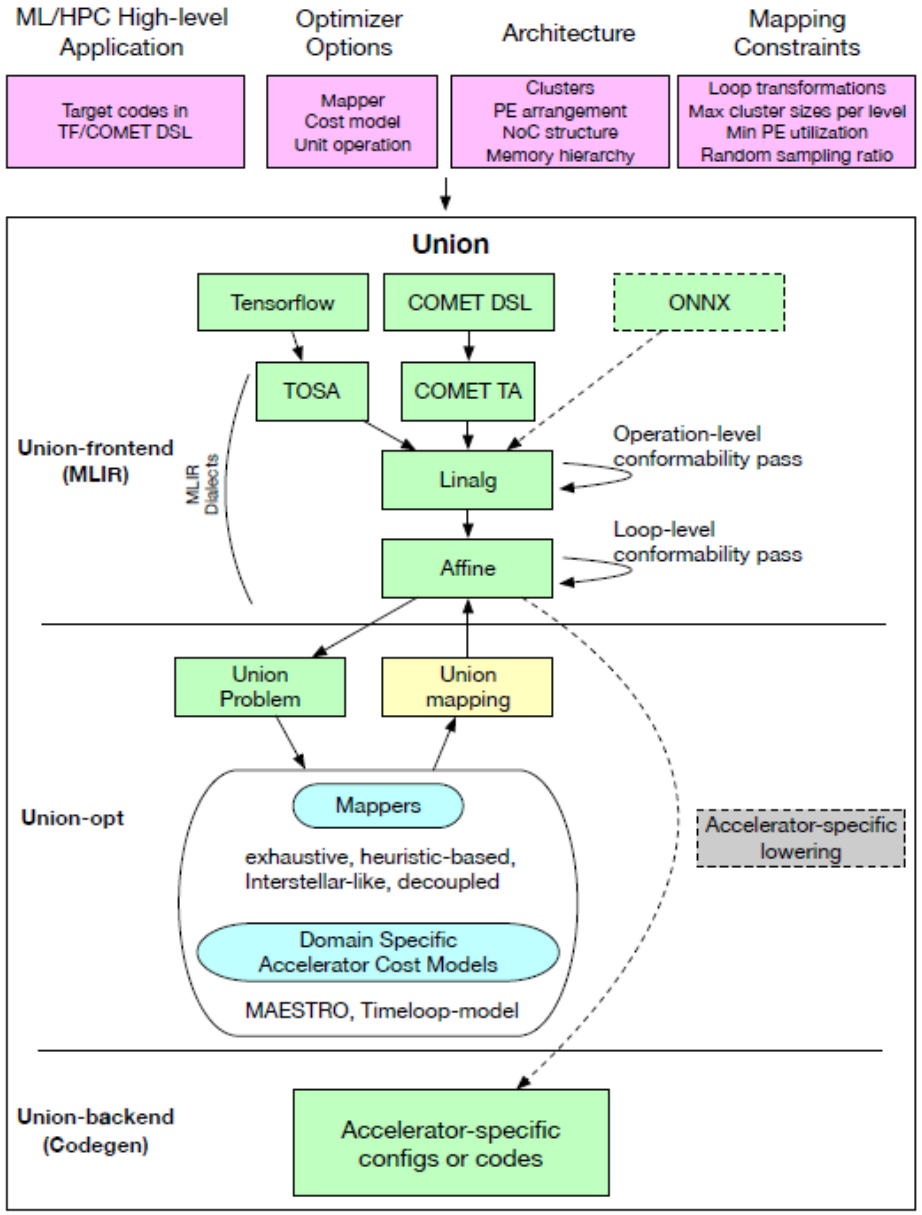


Fig. 2. Union overview. The pink boxes indicate the inputs of Union while green boxes are showing how the codes are getting lowered. Rectangles and arrows with dotted lines are out of the scope of this paper.

Union

- 기존에 밀접하게 연결되어 있던 세 요소에 대해 추상화를 제공
  - Unified workload abstraction
  - Unified mapping abstraction
  - Unified hardware abstraction

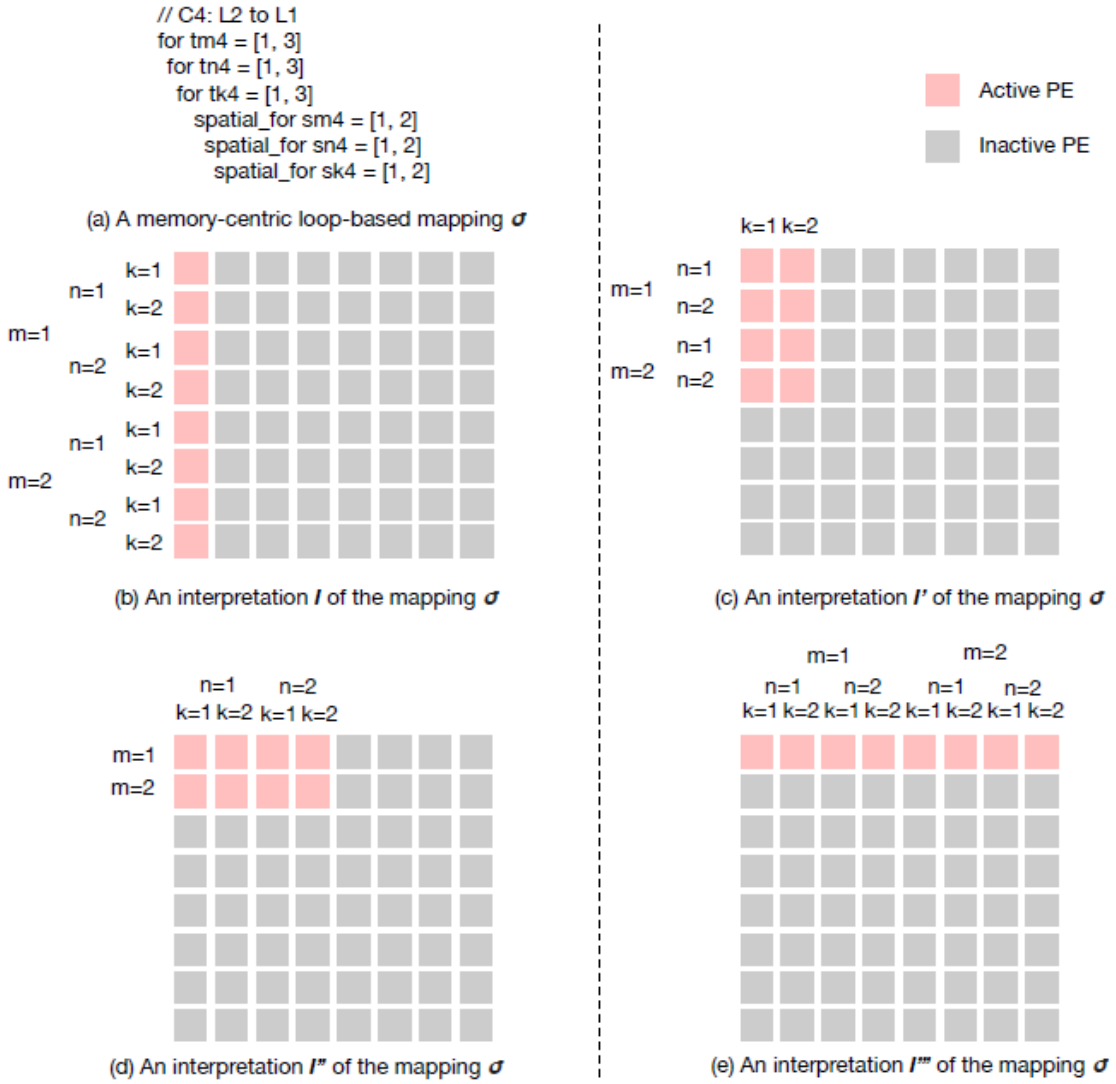


Fig. 4. An example of a memory-target loop-centric mapping and its interpretations on a 8×8 2D PE array.

```
Problem:
  Operator: GEMM

  Shape:
    Name: Example
    Dimensions: [M, N, K]
    Data-space:
      - name: Input
        Projection:
        - [ [M], [K] ]

      - name: Weight
        Projection:
        - [ [K], [N] ]

      - name: Output
        Projection:
        - [ [M], [N] ]
        Read-write: true

  Instance:
    M: 16
    N: 64
    K: 32
```

(a) Union problem

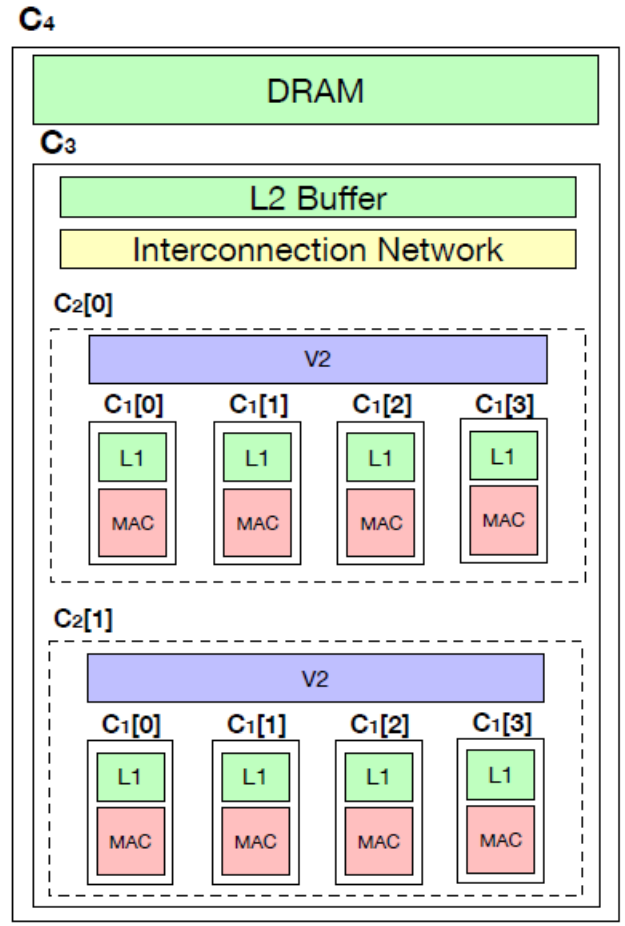
```
Name: C4
  Virtual: False
  Dimension: X
  Local:
    Memory: DRAM

  Sub-tree:
    Name: C3
      Virtual: False
      Dimension: Y
      Local:
        Memory: L2 Buffer

      Sub-tree:
        Name: C2[1…2]
          Virtual: True
          Dimension: X

          Sub-tree:
            Name: C1[1…4]
              Virtual: False
              Local:
                Memory: L1 Buffer
                Compute: MAC Unit
```

(b) Union architecture



(c) Target accelerator architecture

```
// C4: DRAM to L2
target_cluster: C4
temporal_order: MNK
temporal_tile_sizes: 16, 32, 16
spatial_tile_sizes: 16, 32, 16

// C3: L2 to V2
target_cluster: C3
temporal_order: MNK
temporal_tile_sizes: 8, 16, 8
spatial_tile_sizes: 8, 8, 8

// C2: V2 to L1
target_cluster: C2
temporal_order: MNK
temporal_tile_sizes: 8, 8, 8
spatial_tile_sizes: 8, 8, 2

// C1: L1 to MAC
target_cluster: C1
temporal_order: MNK
temporal_tile_sizes: 1, 1, 1
spatial_tile_sizes: 1, 1, 1
```

(d) Union mapping

```
// C4: DRAM to L2
for tm3 = 0
 for tn3 = 0…1
  for tk3 = 0…1
   spatial_for sm3 = 0
   spatial_for sn3 = 0
   spatial_for sk3 = 0
// C3: L2 to V2
for tm2 = 0…1
 for tn2 = 0…1
  for tk2 = 0…1
   spatial_for sm2 = 0
   spatial_for sn2 = 0…1
   spatial_for sk2 = 0
// C2: V2 to L1
for tm1 = 0
 for tn1 = 0
  for tk1 = 0
   spatial_for sm1 = 0
   spatial_for sn1 = 0
   spatial_for sk1 = 0 … 3
// C1: L1 to MAC
for tm0 = 0…7
 for tn0 = 0…7
  for tk0 = 0…1
   spatial_for sm0 = 0
   spatial_for sn0 = 0
   spatial_for sk0 = 0
```

(e) Loop nest representation

Fig. 5. Union abstractions to describe a GEMM problem with a mapping on an accelerator which is composed of a simple 2D PE array. (a) describes a Union problem for a GEMM problem and (b) describes the Union architecture of the target architecture shown in (c). (d) shows a Union mapping that shows how to map the data to the architecture to run the GEMM problem. (e) represents the Union mapping in loop nest form.

Contribution

- 여러 영역(ML, HPC)에서 텐서 연산에 대한 빠른 평가를 제공하는 plug-and-play unified ecosystem

- 텐서 연산과 spatial accelerator에 대한 매핑을 설명해 서로 다른 mapper와 cost model을 통합하는 new unified abstraction

- 연산을 일반화해 cost model을 통한 하드웨어에 대한 평가를 제공할 수 있는 operation-level / loop-level analysis

Compiler support for sparse tensor computations in MLIR

ACM Transactions on Architecture and Code Optimization (TACO), 2022

차세대 컴퓨터 시스템 연구실

이원호

Problem

- 딥러닝 모델의 크기가 증가하면서 연산에 필요한 텐서들의 희소성 또한 증가함

- 희소성은 스토리지 요구량을 줄이고, 연산 시간을 줄일 수 있음
  - 이런 희소성을 활용하기 위해 수작업으로 수정하는 건 복잡하고, 에러가 빈번히 발생할 수 있음

Propose

- 희소성을 텐서의 특징으로 취급하는 MLIR dialect 제안
  - 희소 텐서에 대한 속성, 타입, 연산, 변환 제공

- 연산의 희소성 진단 정의를 통해 희소 코드를 자동으로 생성

```
%C = linalg.matmul ins(%A, %B: tensor<2x4xf64>, tensor<4x8xf64>) -> tensor<2x8xf64>,
```



```
%C = linalg.matmul ins(%A, %B: tensor<2x4xf64, #Sparse>, tensor<4x8xf64>) -> tensor<2x8xf64>
```

Compression example

$$\vec{x} = (0, 0, 0, x_3, 0, 0, x_6, x_7, 0, 0, x_{10}, 0, 0, 0, 0, 0)$$

```
#SparseVector= #sparse_tensor.encoding<{
   dimLevelType = [ "compressed" ]
}>
tensor<16xf64, #SparseVector>
```

pointers[0]:	0	4		
indices[0]:	3	6	7	10
values:	x_3	x_6	x_7	x_{10}

Compression example

$$A = \begin{pmatrix} a_{0,0} & 0 & 0 & a_{0,3} \\ 0 & 0 & 0 & 0 \\ a_{2,0} & 0 & 0 & 0 \end{pmatrix}$$

```
#CSR = #sparse_tensor.encoding<{
  dimLevelType = [ "dense", "compressed" ]
}>
tensor<3x4xf64, #CSR>
```

pointers[1]:	0	2	2	3
indices[1]:	0	3	0	
values:	$a_{0,0}$	$a_{0,3}$	$a_{2,0}$	

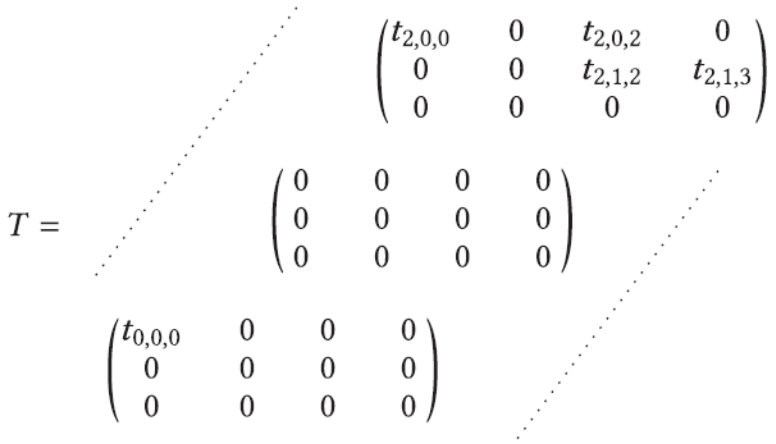
Compression example

$$A = \begin{pmatrix} a_{0,0} & 0 & 0 & a_{0,3} \\ 0 & 0 & 0 & 0 \\ a_{2,0} & 0 & 0 & 0 \end{pmatrix}$$

```
#DCSC = #sparse_tensor.encoding<{
  dimLevelType = [ "compressed", "compressed" ],
  dimOrdering  = affine_map<(i,j) -> (j,i)>
}>
tensor<3x4xf64, #DCSC>
```

pointers[0]:	0	2	
indices[0]:	0	3	
pointers[1]:	0	2	3
indices[1]:	0	2	0
values:	$a_{0,0}$	$a_{2,0}$	$a_{0,3}$

Compression example




```
#SparseTensor = #sparse_tensor.encoding<{
  dimLevelType = [ "compressed",
                   "compressed",
                   "compressed" ]
}>
tensor<3x3x4xf64, #SparseTensor>
```

pointers[0]:	0	2			
indices[0]:	0	2			
pointers[1]:	0	1	3		
indices[1]:	0	0	1		
pointers[2]:	0	1	3	5	
indices[2]:	0	0	2	2	3
values:	$t_{0,0,0}$	$t_{2,0,0}$	$t_{2,0,2}$	$t_{2,1,2}$	$t_{2,1,3}$

Propose

- Sparse compiler의 두 가지 사용법

- End-to-end Sparse Compiler Support for Array Languages
  - Sparse Dialect를 추가함으로써 새로운 컴파일러 프론트 엔드를 생성하는 작업을 희소성과 독립적으로 진행할 수 있음
- Sparse State Space Search
  - MLIR이 제공하는 파이썬 인터페이스를 통해 특정 커널에 대한 전체 희소 저장 형태와 컴파일러 최적화의 탐색 비용을 줄임

Propose

- End-to-end Sparse Compiler Support for Array Languages
  - Sparse Dialect를 추가함으로써 새로운 컴파일러 프론트 엔드를 생성하는 작업을 희소성과 독립적으로 진행할 수 있음

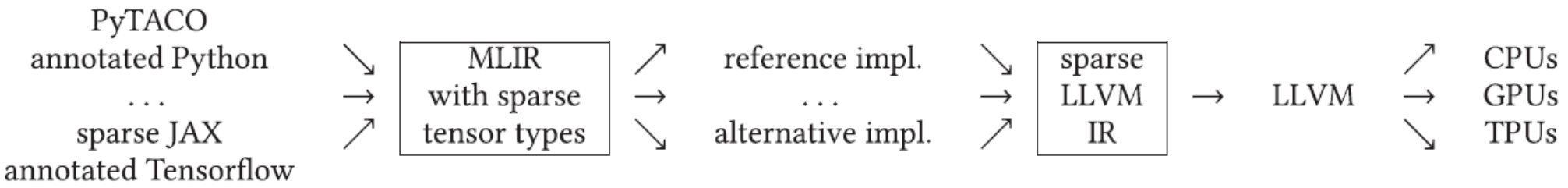


Fig. 3. Overview of retargetable sparse compiler support. Multiple front-ends map different sparse array languages, such as PyTACO, to a shared high-level intermediate representation, such as `Linalg` with sparse tensor types. Alternative sparse compiler pipelines, like the reference implementation, lower this intermediate to a form that only stores and iterates over nonzero elements. This intermediate is handed off to a retargetable back-end compiler, like LLVM.

Propose

- Sparse State Space Search
  - MLIR이 제공하는 파이썬 인터페이스를 통해 특정 커널에 대한 전체 희소 저장 형태와 컴파일러 최적화의 탐색 비용을 줄임

```
for level in [ [dense,      dense], [dense,      compressed],
               [compressed, dense], [compressed, compressed] ]:
  for ordering in [ ir.AffineMap.get_permutation([0, 1]),
                    ir.AffineMap.get_permutation([1, 0]) ]:
    for ptrWidth in [0, 8, 16, 32, 64]:
      for indxWidth in [0, 8, 16, 32, 64]:
        attr = st.EncodingAttr.get(level, ordering, ptrWidth, idxWidth)
        mlir = buildSpMM(attr)
        for opt in compilerStrategies:
          exec = compileKernel(mlir, opt)
          result = runKernel(exec, input)
          verify(result)
```

Contribution

- MLIR dialect를 통해 희소 텐서 연산 컴파일러에 대한 명세
- 다른 MLIR dialect의 sparse dialect 와 transformation의 결합
- 희소 컴파일러 최적화 측면의 비용 감소

SparTA: Deep-Learning Model Sparsity via Tensor-with-Sparsity-Attribute

16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). 2022.

차세대 컴퓨터 시스템 연구실

이원호

Problem

- 딥러닝 모델의 크기가 증가하면서 연산에 필요한 텐서들의 희소성 또한 증가함

- 희소성은 효율성과 scalability를 탐색하기 위해 가장 중요한 요소

- DNN 모델은 일반적으로 Dataflow graph (DFG) 표현으로 나타냄
  - Node = operator (하나 이상의 입출력 텐서와 함께)
- 이런 DNN operators를 sparsity pattern에 대해 커스텀 했을 때 inference latency 의 감소를 기대
- 희소성의 증가가 실제 효율성의 증가로 이어지지 못함

Problem

- 희소성의 증가가 실제 효율성의 증가로 이어지지 못함
  - 범용 희소 연산들을 위한 연산 커널이 최적이 아님
  - DNN 연산은 여러 스테이지에 걸쳐 발생 → sparse-pattern도 여러 스테이지를 걸쳐 발생함 → sparsity-aware optimization이 어려움
  - 기존 sparsity-aware optimization은 다른 기술 스택들의 도움이 필요 (framework, kernel, hardware etc)

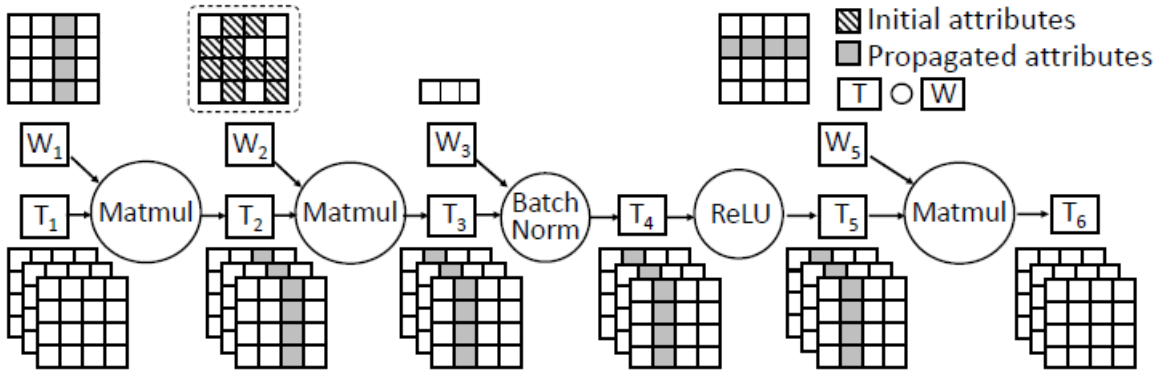


Figure 1: The sparsity attribute of one tensor can be propagated along the deep learning network.

SparTA

- Sparsity = first-class citizen
- 설계 원칙
  - 새로운 희소성에 대해 Customizable / Extensible
  - 모든 연산자 or layer에 대해 작업 가능해야 함
- TeSA 라는 새로운 추상화의 도입
  - 희소성 속성과 패턴을 포함하는 일반 텐서
  - 해당 속성을 도입해 효율적인 실행 계획 생성 가능

SparTA – Overall Architecture

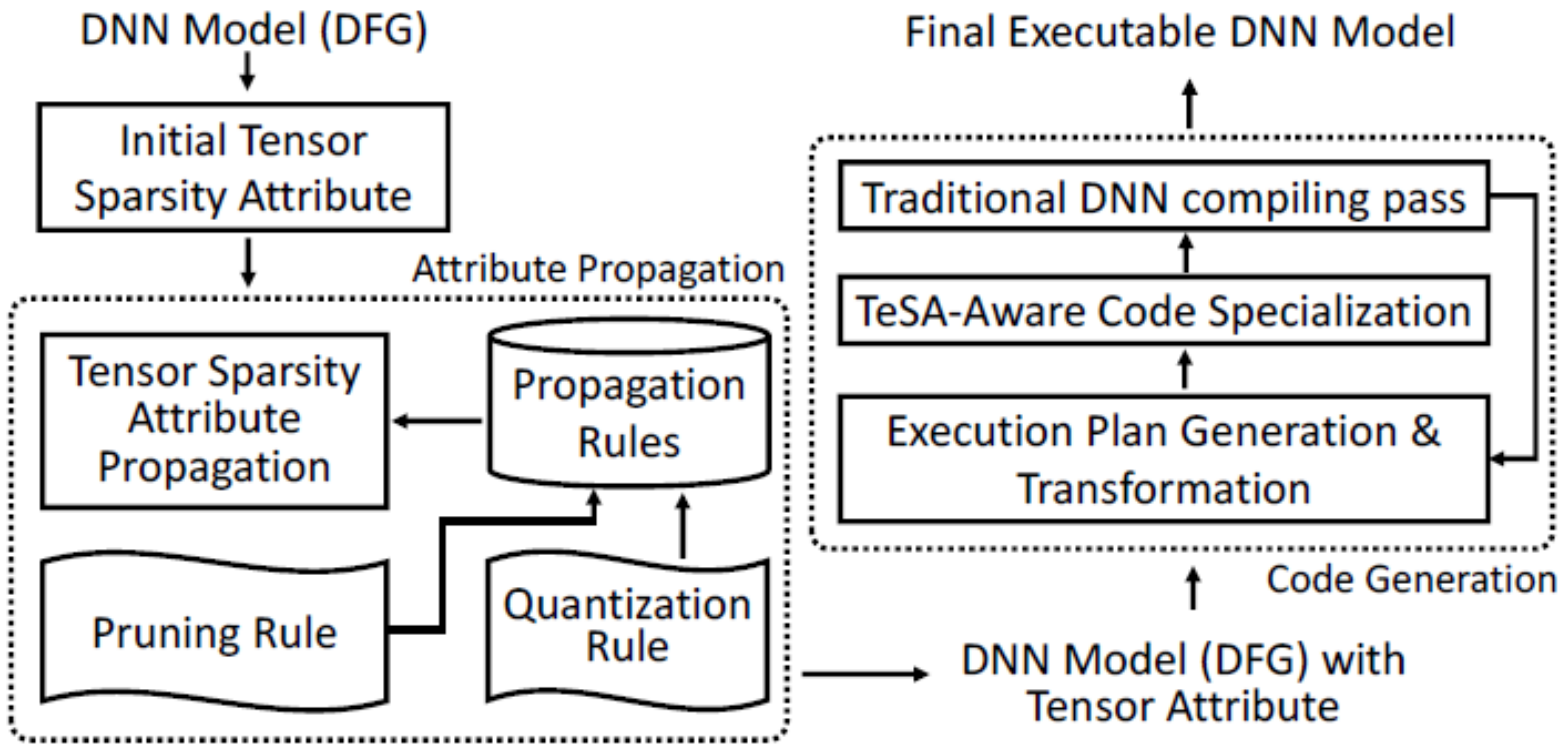


Figure 2: The system architecture of SparTA.

SparTA – Code Generation

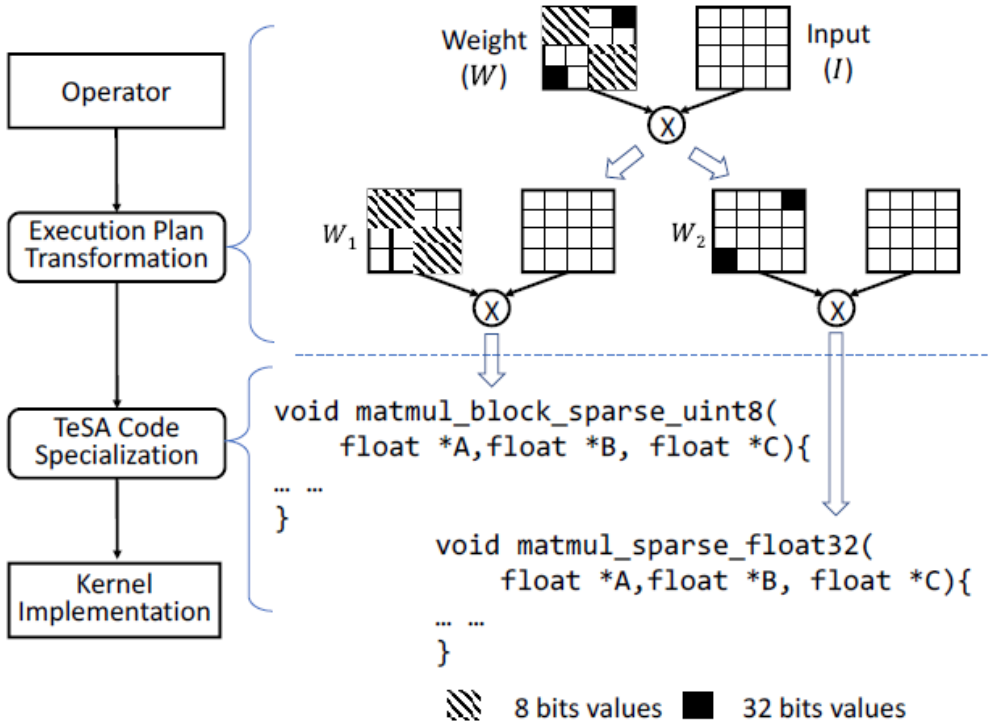


Figure 5: Two-pass compilation to generate an efficient kernel for an operator (`MatMul`).

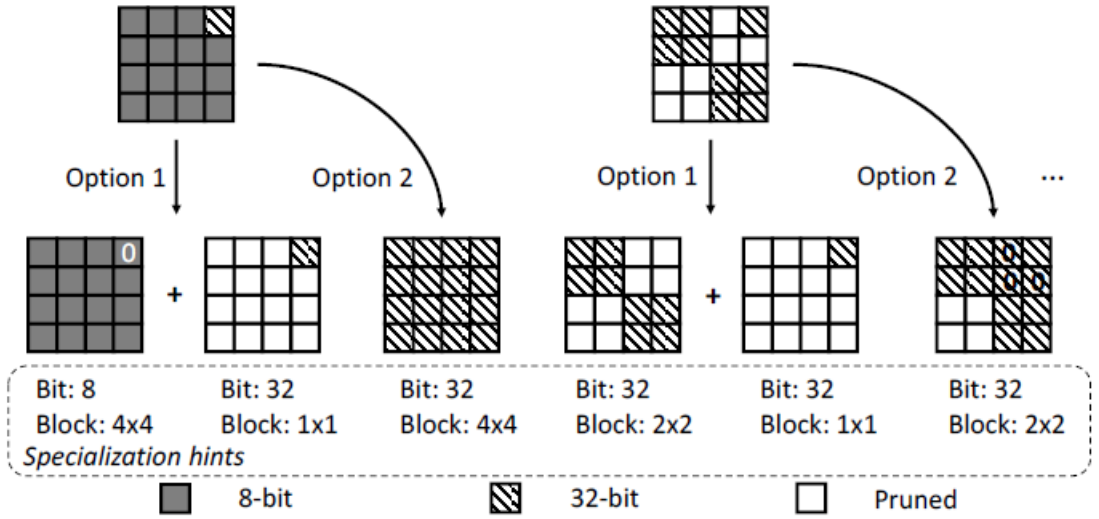


Figure 6: Multiple transformation plans produced by a transformation policy. The specialization hints are used by the second pass compilation for code specialization.

SparTA

- 앞선 컴파일 경로를 통해 더 적은 연산량을 제공하는 컴파일 패스를 선택해
- 희소성 인지 코드를 구체화함

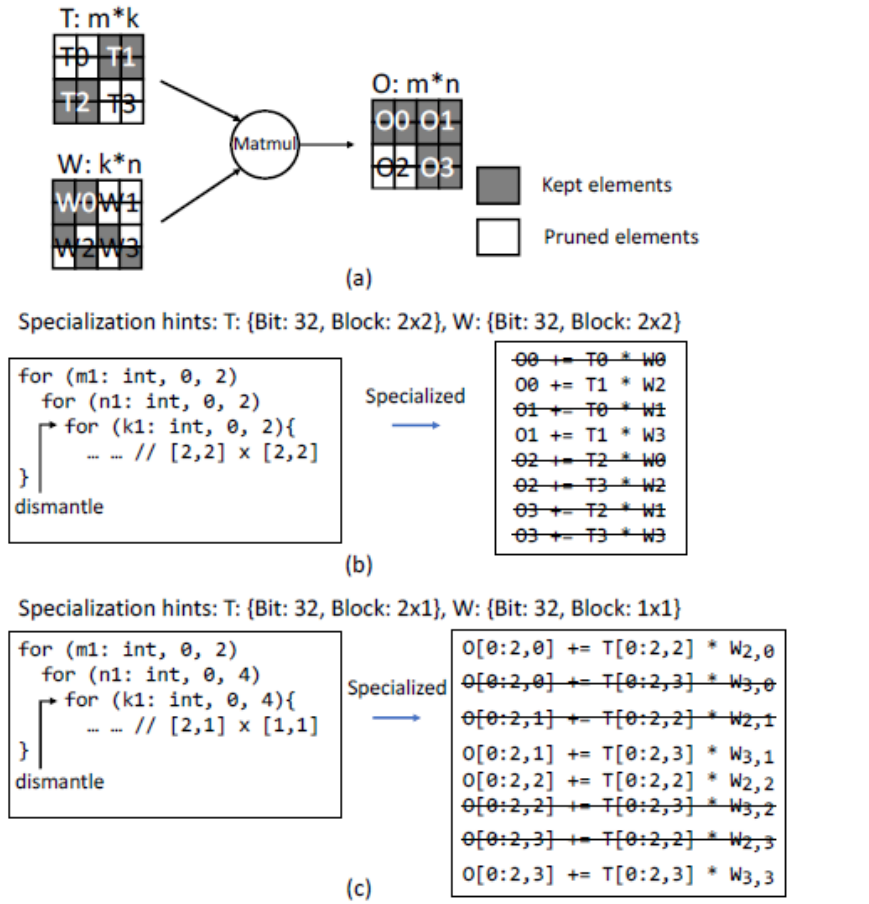


Figure 7: Sparsity-aware code specialization, leveraging specialization hints generated during execution plan transformation. (a) is a sparse Matmul, (b) and (c) are its specialized kernel code with different transformation plans. $T[x:y,z]$ denotes the elements on row x to y and column z, $W_{x,y}$ denotes the value on row x and column y of W.

Contribution

- 다양한 sparsity pattern에 대해 수용할 수 있는 설계
- 모델의 희소성을 일관된 방식으로 평가 가능

Reference

- MLIR: Scaling compiler infrastructure for domain specific computation 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). IEEE, 2021

- Union: A unified HW-SW co-design ecosystem in MLIR for evaluating tensor operations on spatial accelerators 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT). IEEE, 2021.

- Compiler support for sparse tensor computations in MLIR ACM Transactions on Architecture and Code Optimization (TACO), 2022

- SparTA: Deep-Learning Model Sparsity via Tensor-with-Sparsity-Attribute 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). 2022.