Automatic Operator Generation For Deep Learning Frameworks In The All-Scenario Context: MindSpore/AKG Architecture, Features and Challenges

Cedric Bastoul
Huawei Technologies

MindSpore HUAWEI

Scheduling AI Fused Operators For All Scenarios

AI/DL frameworks are key tools for R&D and industry

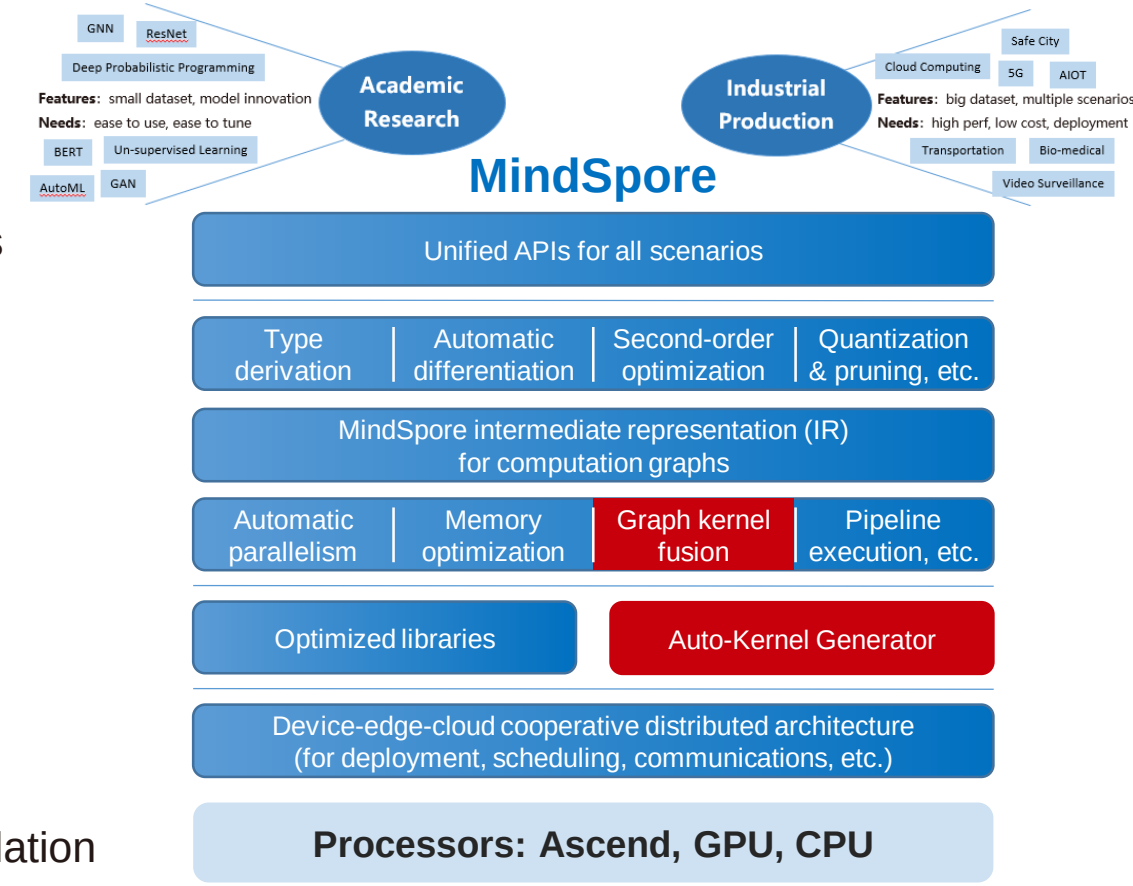
- Map high-level DNN models to efficient implementations
- Hide the complexity of target architectures to AI/DL scientists
- Support fast development of new models

Operator fusion enables critical optimizations

- Improve data locality and use of processing elements
- Minimize intermediate result storing and communication
- Reduce the cost of kernel launch on accelerators

Automatic kernel generation is required

- Complement the limited scope of optimized libraries
- Exploit the regular nature of operators with polyhedral compilation
- **Face the challenge of polyhedral optimization for all scenarios**



https://www.mindspore.cn/en

HUAWEI

Contents

HUAWEI

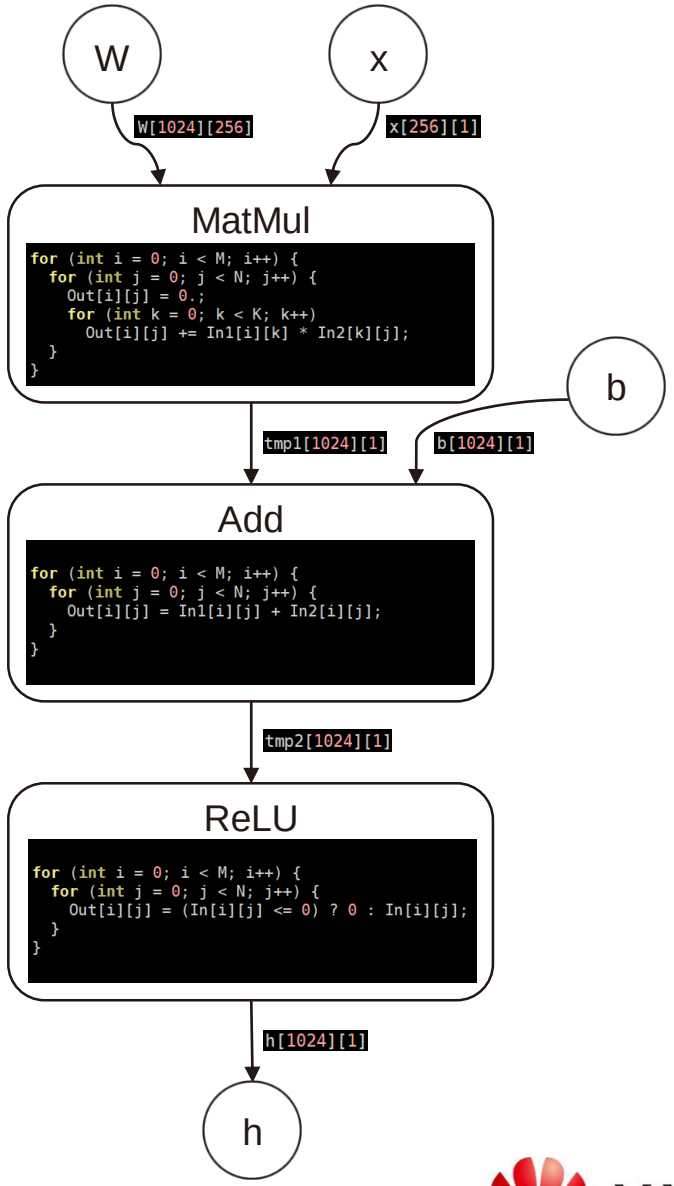
AI/DL Computational Graphs

Express numerical computation as a directed graph

- Nodes are operators with input and output tensors
  - Either framework built-in operators or user custom operators
- Edges are tensors of various shapes and types
  - Tensors may store input data, model parameters or temporaries
- Graphs are built using the framework input language
  - And/or partially automatically generated by, e.g., autodiff

Graph compilation and execution

- Compilation builds subgraphs, selects or generates operators, and prepares scheduling
- A runtime executes the graph, launching operators when their input is ready



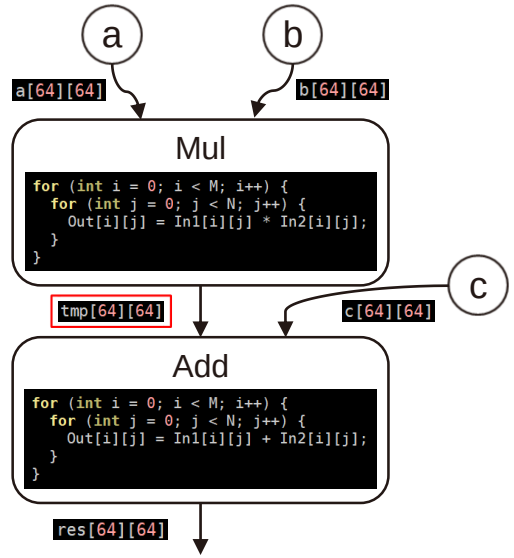
Operator Fusion

Substitution of a group of operators in the original computation graph with a new compound operator according to some construction rules

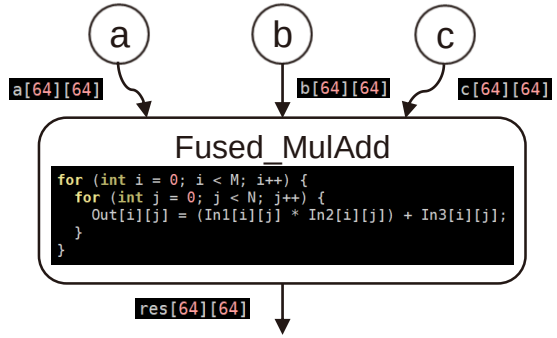
- Formula fusion: combine several operator formulas and map it to an operator (e.g., Conv + BN → Conv')
- **General fusion: combine a sub-graph into a new operator**

Main benefits

- Data access and storage optimization
  - Promote intermediate results to fast memory
  - Remove communications with host
- Better hardware usage and occupancy
  - Enable more loop-level optimization
  - Reduce the cost of kernel launch on accelerators
  - Independent operators may be merged to use available resources
- Inter-operator analysis
  - Common subexpression and dead code elimination
  - Algebraic simplifications

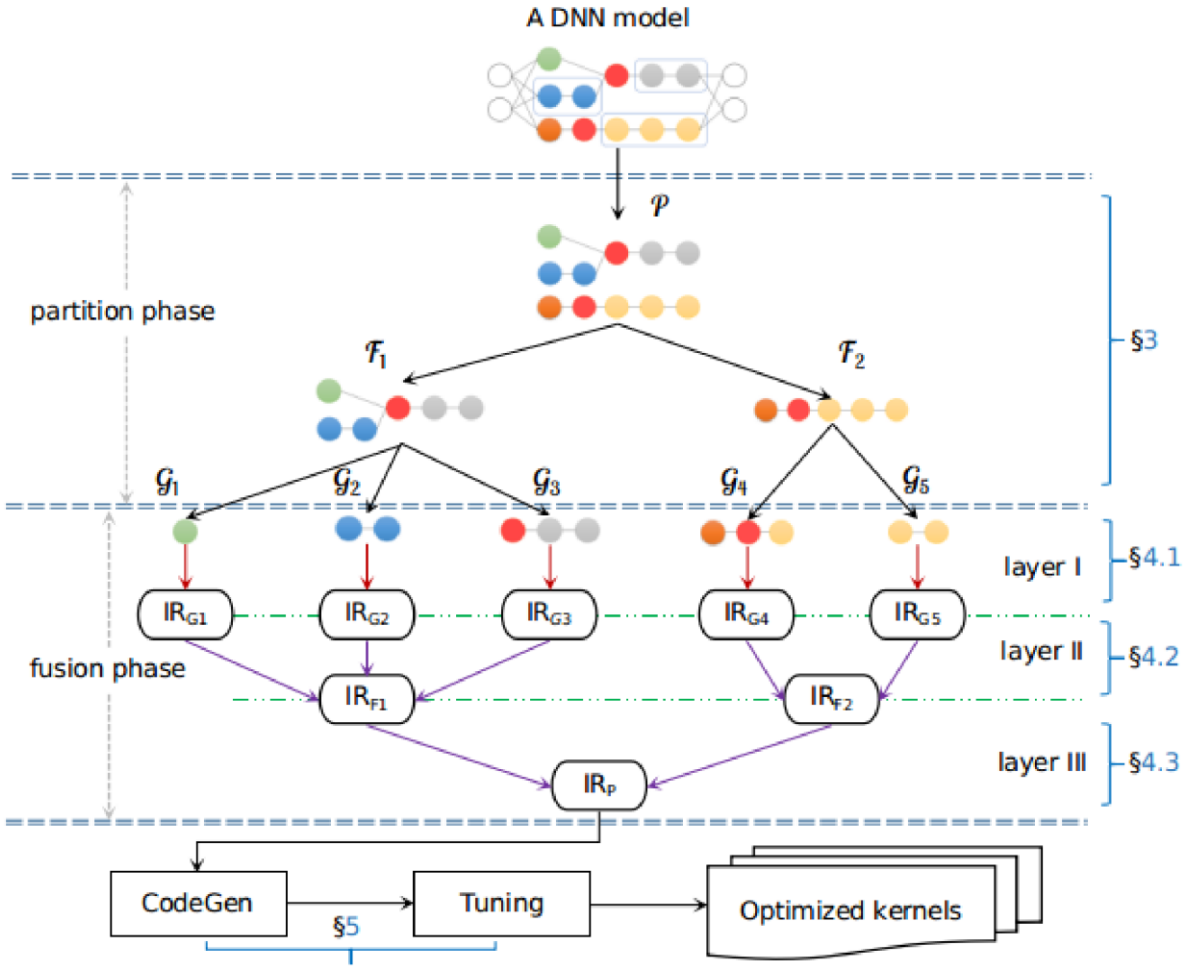


(a) Without Graph Kernel Fusion



(b) With Graph Kernel Fusion

MindSpore's Graph Kernel Approach For Operator Fusion



[Jie Zhao et al. Apollo: Automatic Partition-based Operator Fusion through Layer by Layer Optimization, MLSys 2022]

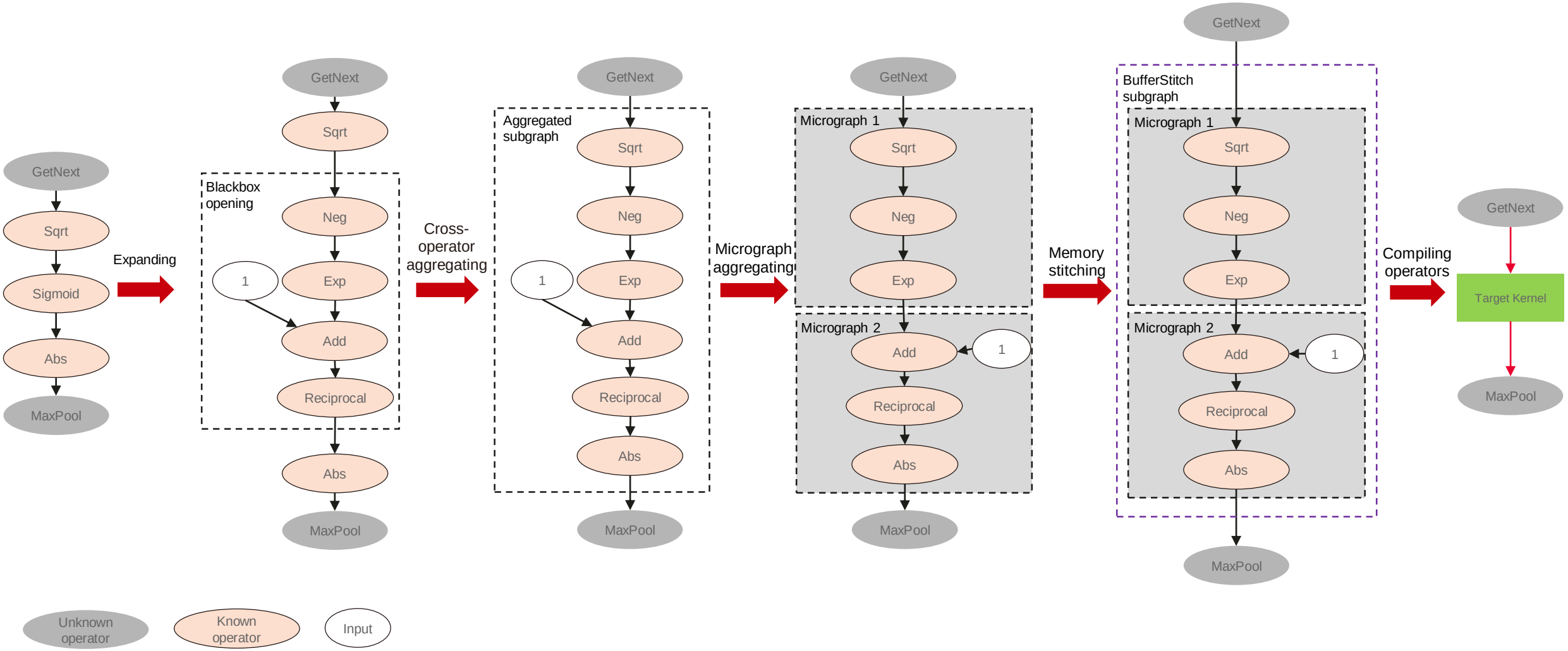
Partition phase

- Opening compound "black-box" operators
- Cross-operator aggregation according to aggregation rules (designed to work well with polyhedral scheduling)
- Fused-operator analysis (CSE, DSE, algebraic optimization, etc.)
- Split sub-graph into micro-graphs for suitable polyhedral scheduling and code generation

Fusion phase

- [Layer I] Polyhedral loop fusion and optimization with AKG
- [Layer II] Memory stitching: fusing dependent kernels for better memory reuse
- [Layer III] Parallelism stitching: fusing independent kernels for better resource usage

Operator Partition/Fusion Process Example



HUAWEI

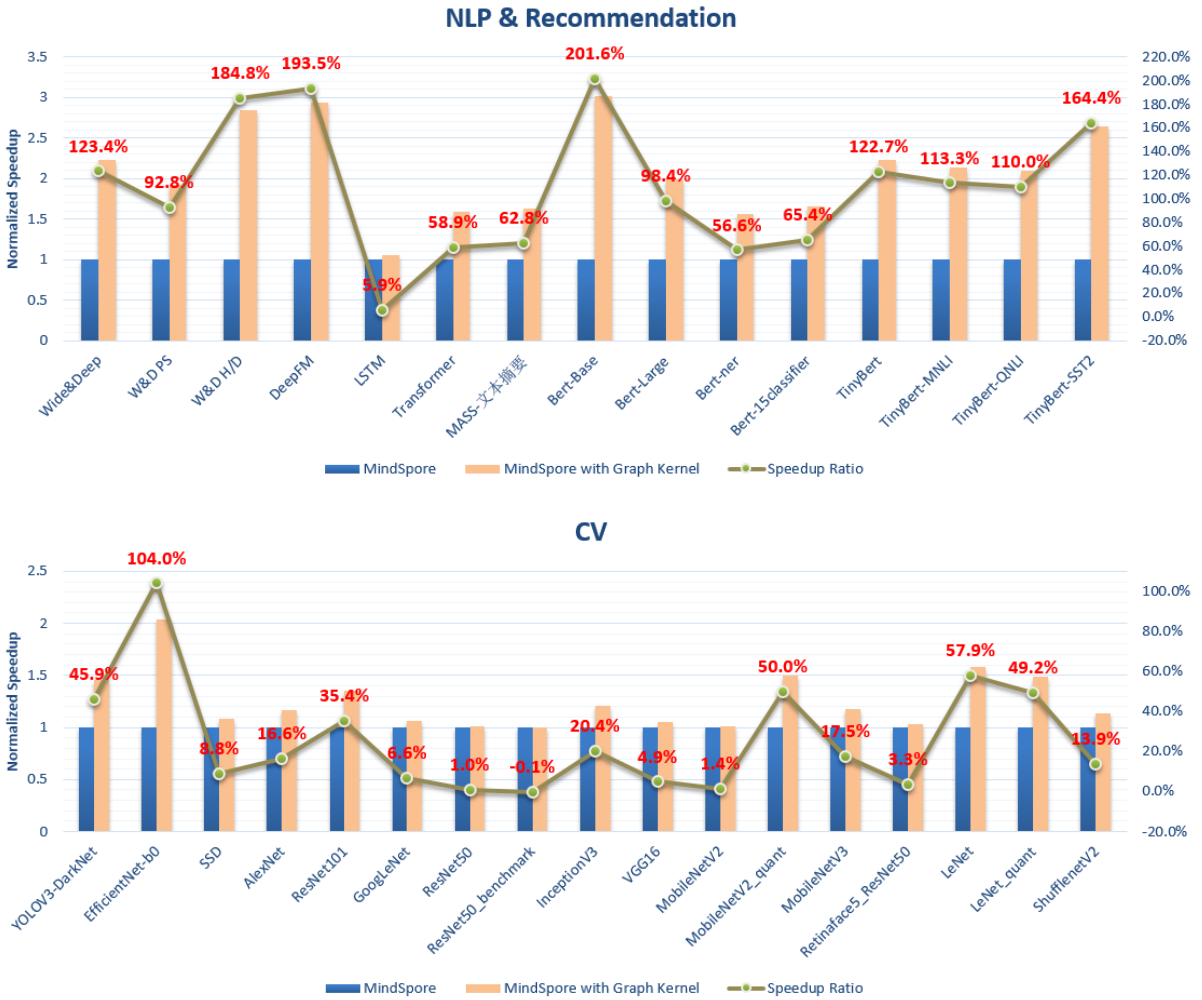
Current Graph Kernel Application Effect

GPU platform speedup average

- NLP: 96.4 %
- Recommendation: 136.6 %
- CV: 30.7 %

Fusion strategy replaces raising

- Raising to polyhedral representation is guaranteed operator-wise
  - While challenging in compilers
- But fusion strategy selection is critical for performance
  - And it may depend on the target architecture…



Multi-Target Architecture Challenge

Three architectures supported CPU + GPU + NPU

- Support widely different architectures best at different purpose
- Apply different optimization strategies as transparently as possible
- Limit software fragmentation as much as possible

A New Golden Age for Computer Architecture:
Domain-Specific Hardware/Software Co-Design, Enhanced Security, Open Instruction Sets, and Agile Chip Development

John Hennessy and David Patterson
Stanford and UC Berkeley
June 4, 2018
https://www.youtube.com/watch?v=3LVeEjsn8Ts

…And a new challenging era for compilers!

CPU

Application

- Prototypes requiring high flexibility
- Training simple/small models

Characteristics

- Exploit ILP, vectors, excellent for general programming
- Implicitly managed memory

```
void sgemm(float C[M][N], float A[M][K], float B[K][N]) {
  #pragma omp parallel for
  for (int i = 0; i < M; i++)   // Enough parallelism on i
    for (int k = 0; k < K; k++) // ikj for cache + vector
      for (int j = 0; j < N; j++)
        C[i][j] += A[i][k] * B[k][j];
}
```

GPU

Application

- Models with operators on GPU
- Training medium/larger size models

Characteristics

- Exploit SIMT, excellent for graphics
- Mixed implicitly/explicitly managed memory

```
__global__ void sgemm(float* C, float* A, float* B) {
  int i = blockIdx.x;  // Exploit all thread parallelism
  int j = threadIdx.x; // Here i on blocks, j on threads
  float sum = 0.0f;
  for (int k = 0; k < K; k++)
    sum += A[i * K + k] * B[k * N + j];
  C[i * N + j] = sum;
}
```

NPU

Application

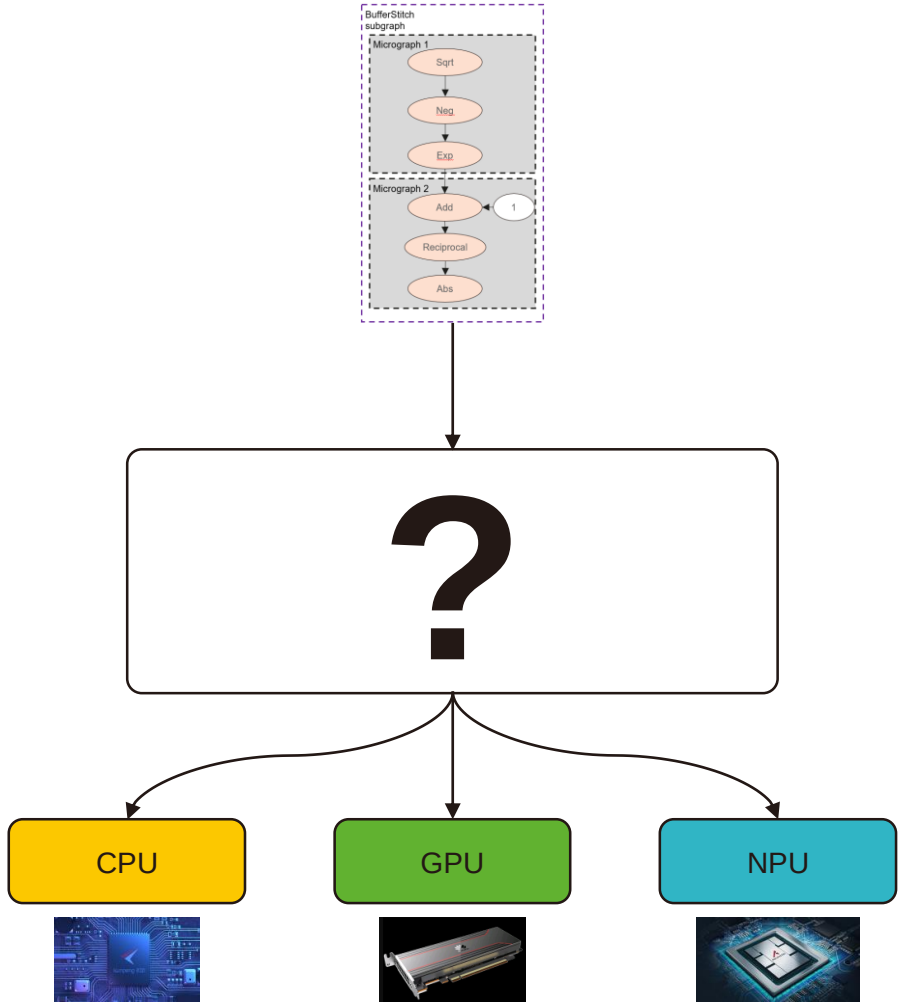
- Models with lots of matrix multiplications
- Training huge models

Characteristics

- Matrix and vector processing
- Explicitly managed memory

```
// Just use the intrinsic!
mad(C, A, B, M, K, N);
```

How To Bridge This?



Contents

HUAWEI

AKG: Auto Kernel Generator

Three supported input

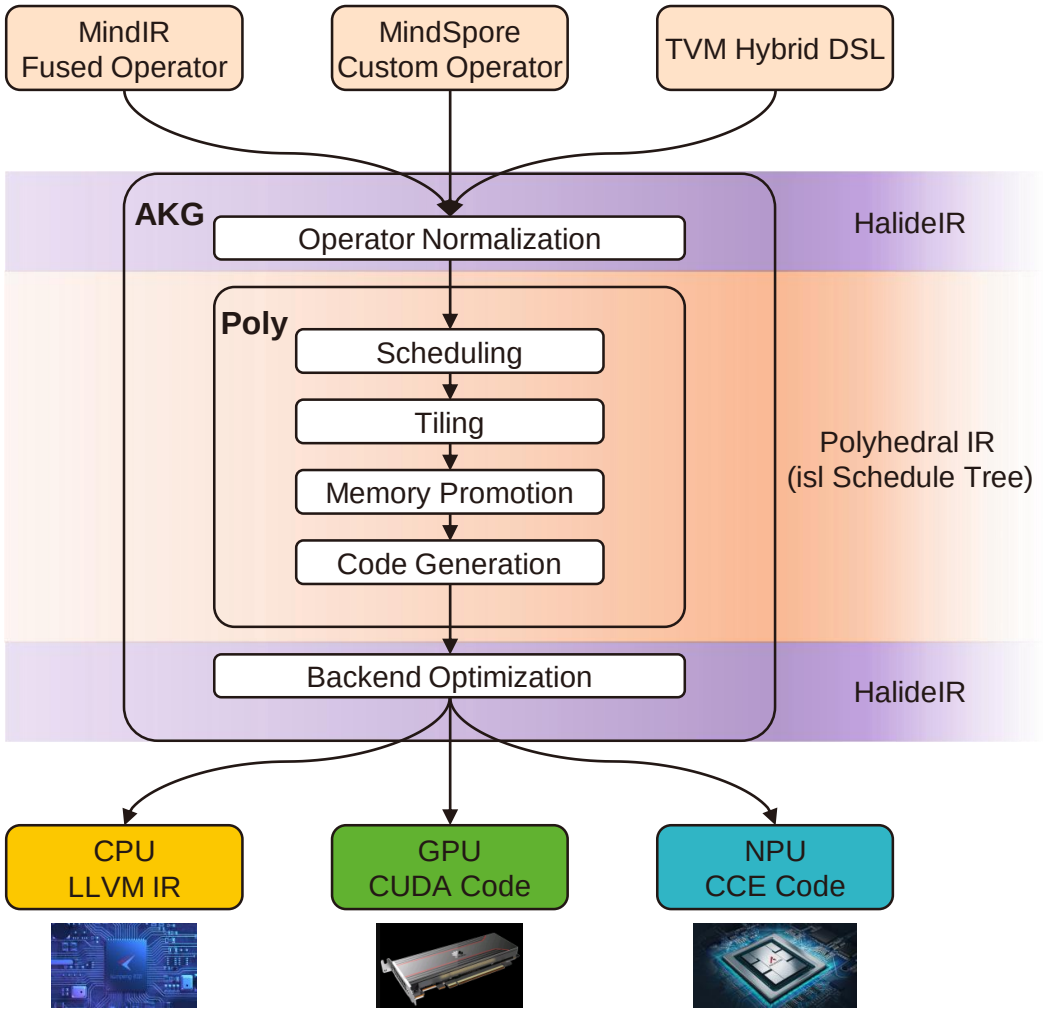
- Fused operators from MindSpore higher level
- Custom operators for user-defined operators (e.g., for scientific computing)
- TVM Hybrid DSL for fast design using built-in operators

Three main phases

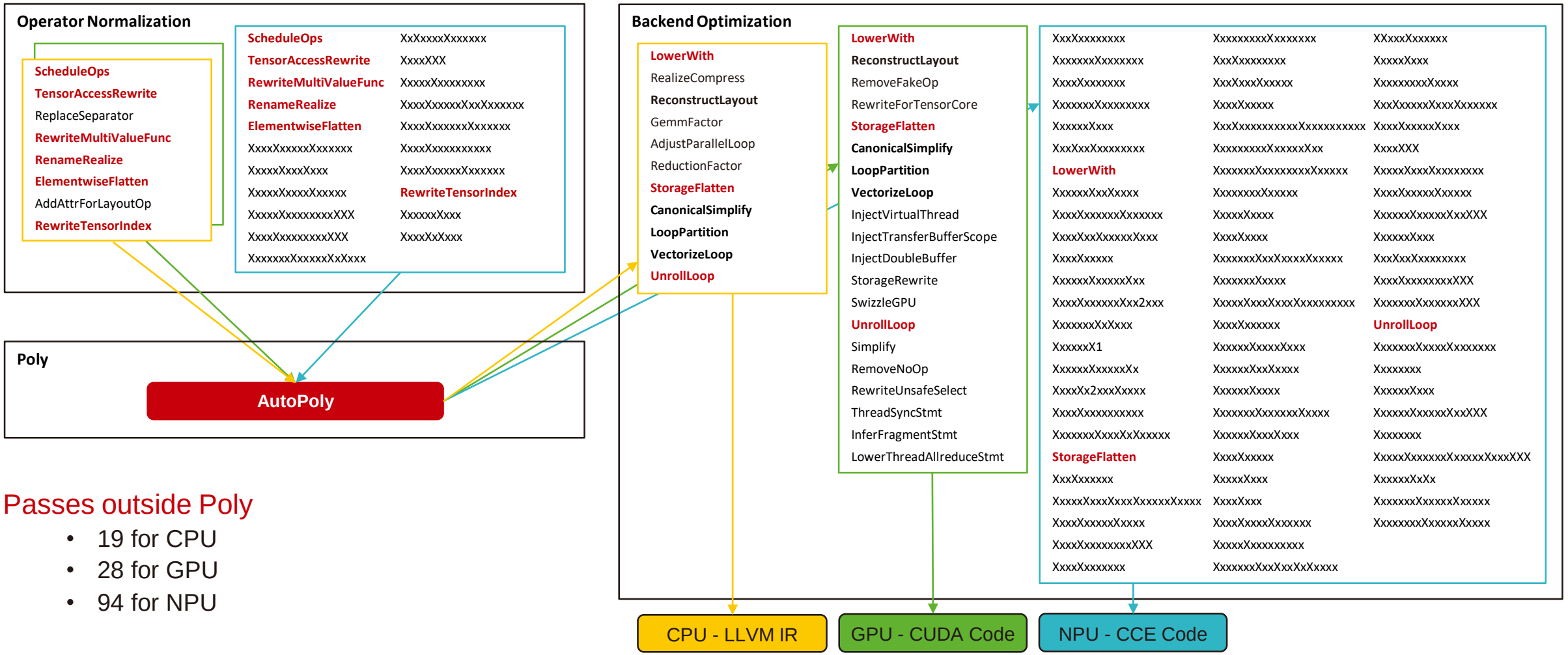
- Operator normalization to prepare polyhedral processing
  - Comply to linear constraints (e.g., embed complex conditionals)
  - Reduce complexity (e.g., inlining, CSE, fusion)
- Automatic polyhedral scheduling (e.g., loop optimization, thread parallelization, tiling, vectorization, memory mapping)
- Backend optimization (e.g., TensorCore acceleration, storage flatten, double buffering, synchronization insertion)

Three backends CPU + GPU + TPU

- Currently CPU, NVIDIA V100/A100, Ascend910



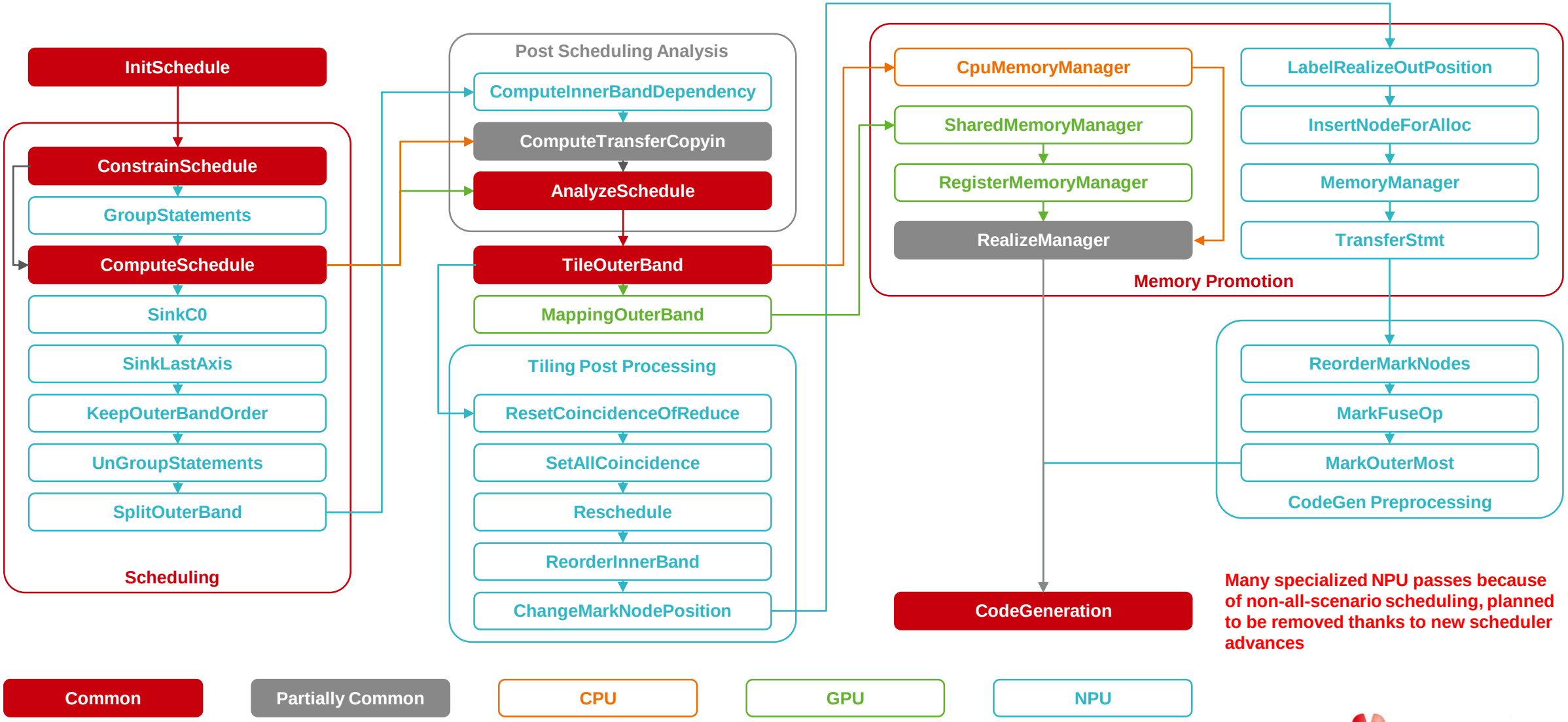
AKG Pass Overview (Current Dev Version)



Passes outside Poly

- 19 for CPU
- 28 for GPU
- 94 for NPU

AutoPoly Pass Overview (Current Dev Version)



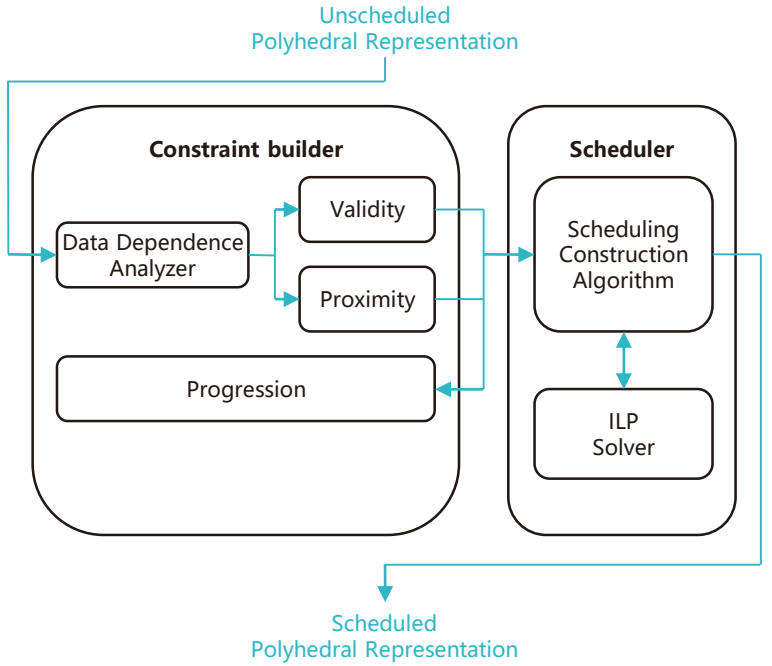
All Scenarios Polyhedral Scheduling Challenges

Polyhedral scheduling is responsible for critical actions/decisions

- Parallelism extraction: expose parallel loops
- Permutability extraction: expose “tilable” loops
- Loop-level optimization: data locality, access pattern, etc.

Limitations related to *all scenarios* context

- Current (Pluto/isl) initial scheduling is domain and target independent
  - Fixed iteration scheduling algorithm for outer parallelism and inner data-locality
  - Requires local rescheduling passes to address specific optimization
- Lack of constraints injection mechanism and prioritized optimization:
  - Cannot specify optimization constraints and priorities for DSA features
  - Cannot overcome the limitations of affine cost models and constraints



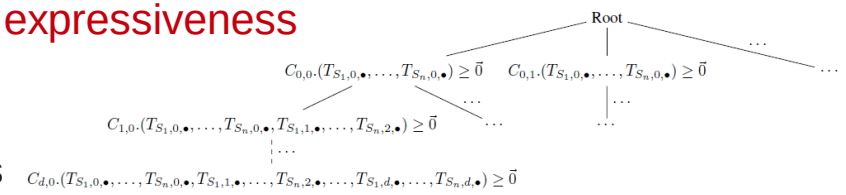
Our proposal

- **A structured solution to incorporate optimization constraints decided by (possibly) non-linear approaches**
- **An application to optimize load/store vectorization on GPU**

Structured Constraint Injection “MindTricks”

Specify desirable scheduling constraints with high expressiveness

- Dedicated “influence constraint tree” abstraction
- Cross-constrain any statement at any dimension
- Enable multiple prioritized optimization scenarios

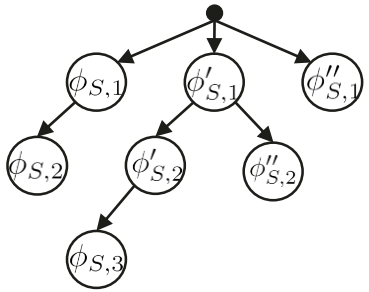


Scheduling algorithm respecting the most profitable scenario

- Influenced scheduling construction: depth-first search for a constrained solution

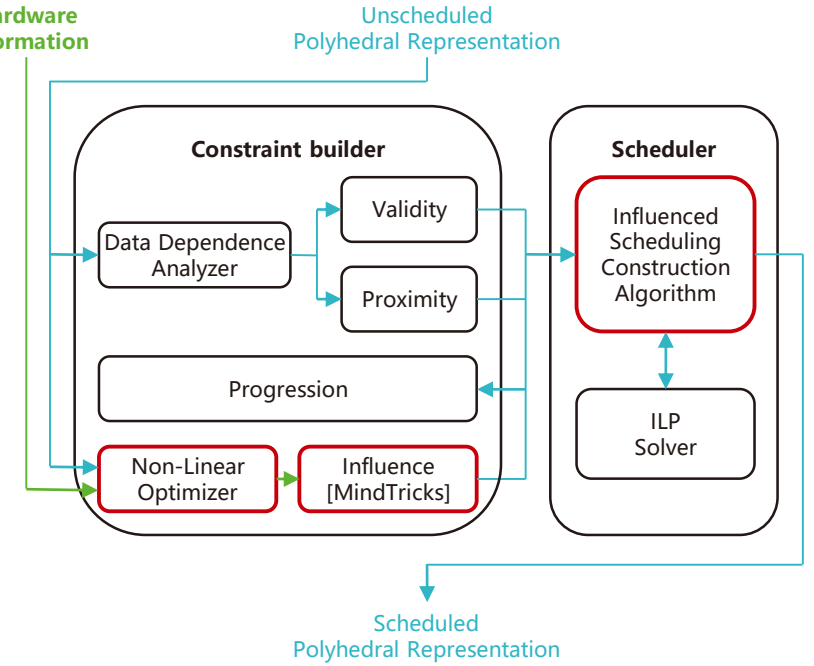
Pluto
Depth is known, trivial termination management

Pluto + MindTricks
Depth depends on injected constraints, adapted progression



Key points:
- Backtracking mechanism with priority management
- Progression constraints neutralized when they conflict with injected constraints
- Termination detection based on dependency and injected constraint satisfaction

- Strictly respects mathematical correctness



- **Construction of the influence constraint tree done by specialized, possibly non-linear, optimizer**
- **AI fused operators hold few dependencies and are likely to satisfy most profitable scenarios**

Application To GPU Fused-Operators LD/ST Vectorization

Non-linear optimization objectives

- Identify best innermost dimensions to prepare use of explicit vector types
- Identify best following dimension organization to maximize coalescing
- Identify statements that should be scheduled together

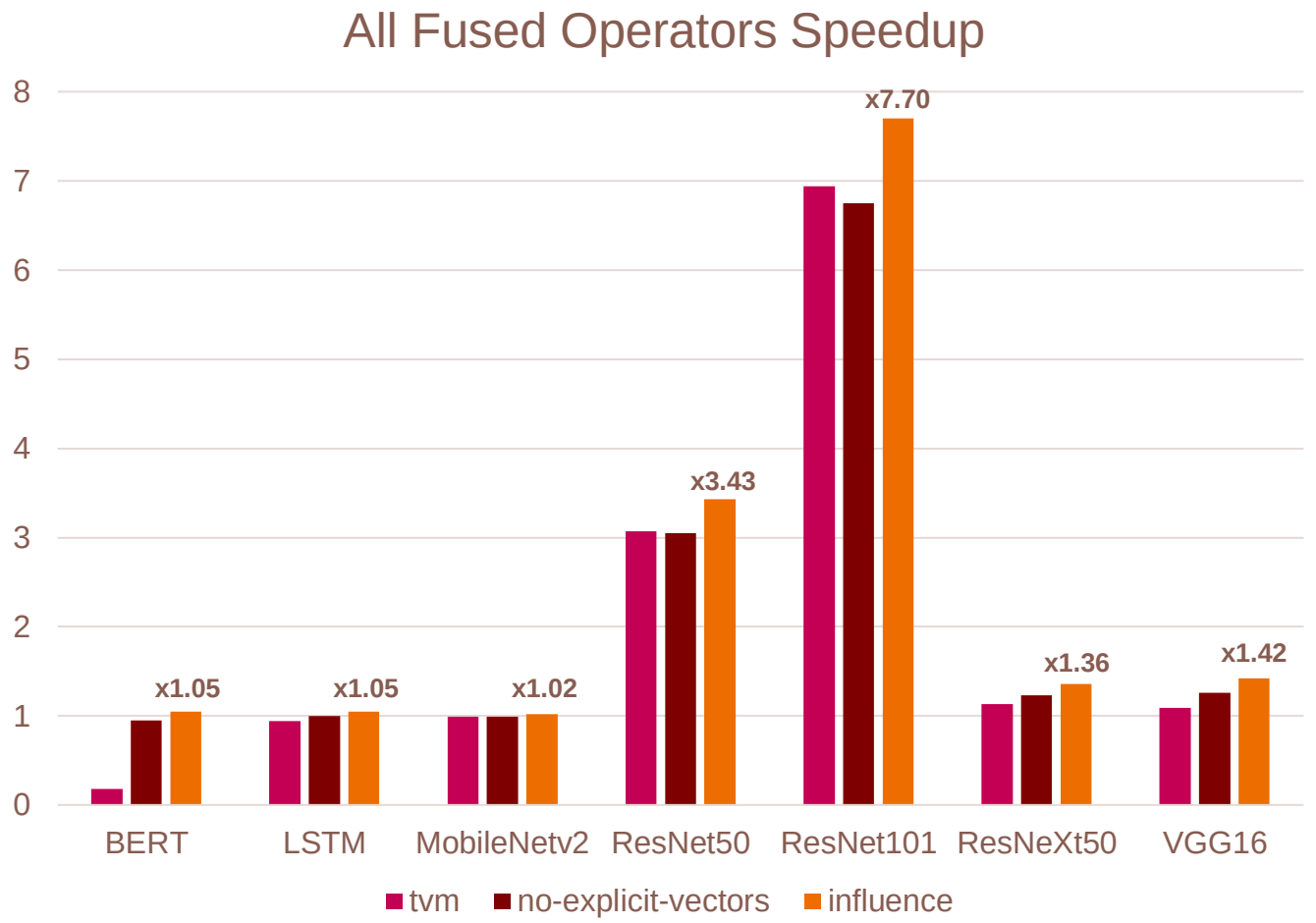
$$cost(W, D, A, L, d) = \overbrace{w_1|V_w| + w_2|V_r|}^{\text{store or load vectorization}} + \underbrace{\frac{w_3}{M} + w_4|C|}_{\text{short memory jumps}} + \overbrace{\frac{w_5 FN}{L}}^{\text{high contribution to thread number}}$$

Experimental alternatives

- **Baseline (1)**: standard isl scheduling
- **tvm**: manual hand-tuned scheduling
- **no-explicit-vectors**: influenced without explicit vectorization
- **influence**: influenced + vectorization

Key results

- Influenced operators: 50% to 90%
- 1.7x geomean speedup
- Highest impact on transpose operators



[Bastoul et al. Optimizing GPU Deep Learning Operators with Polyhedral Scheduling Constraint Injection, CGO 2022]

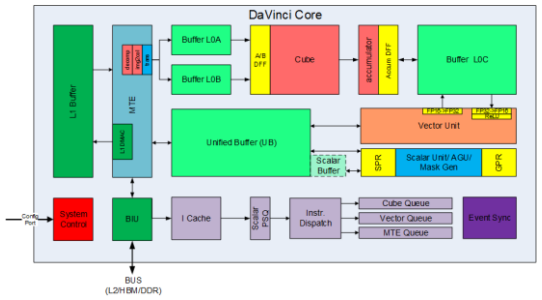
Symbolic Tiling

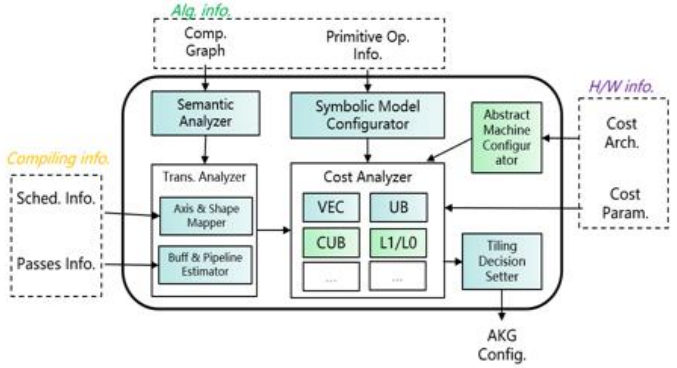
Determining the best tile size as critical as usual

- Only rectangular tiles are considered
- AKG embeds both tuning and automatic systems
- Not only classic problems…
  - Hierarchical caches of various properties
  - Data type and shape
  - Data locality, reuse and alignment
  - Vectorization and parallelization
- …But new challenges related to NPU support
  - Special computing unit (Cube) and instructions (Reduce)
  - Synergy of computing units (Scalar + Vector + Cube)
  - Programmable buffers
  - Pipeline-able computations and communications

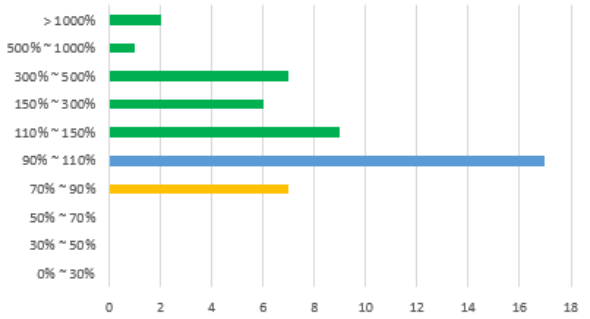
Our approach

- **Cost-equivalent abstract machine**
- **Symbolic analyzer of behaviors**
  - Memory peak usage
  - Component allocation
  - Pipeline, parallelization and vectorization
- **Real-time tile-size optimizer**
  - Multi-dimensional resources repartition
  - Multi-component resources repartition
  - Granularity





Speedup of Fused Ops in GPT Neural Network

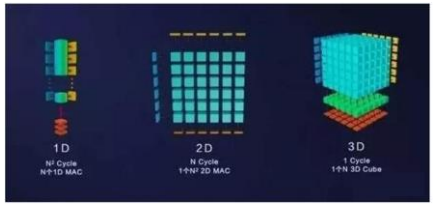


Contents

HUAWEI

Recent Call: DSA-Oriented Algorithms for Fused Operators

Benefits: In scenarios such as scientific computing and DNN training, operator fusion and optimization based on automatic scheduling can strongly speedup performance, minimize computing power, and greatly reduce engineering costs.



- In multi-architecture computing scenarios, the development and maintenance of fusion operators depend on manual scheduling and optimization, which usually require a large amount of engineering efforts. The classic polyhedral scheduling algorithms (ILP-based Pluto/Feautrier/isl), which leverage linear cost functions and fixed-priority cascading search, can balance the outer loop parallelism and inner loop data locality of operators. In the traditional parallel architecture (such as CPU/Legacy GPU), automatic scheduling of high-performance fused operators can be achieved.
- The multi-dimensional AI accelerators (3D-Matrix/2D-Vector/1D-Scalar) of DSAs allow deeper and wider operators fusion and optimization. However, the scheduling space expands and the time and space scheduling constraints increase dramatically. This leads to long modeling and solving in classical polyhedral scheduling algorithms, which becomes the bottleneck of operator fusion in scenarios such as HPC and DNN training.

References:
[1] Cutting to the Core of Pseudo-Boolean Optimization: Combining Core-Guided Search with Cutting Planes Reasoning. In AAAI 2021.
[2] Fast linear programming through transprecision computing on small and sparse data. In OOPSLA 2020.
[3] Polyhedral auto-transformation with no integer linear programming. In PLDI 2018.

Technical Challenges

- **[Challenge 1: Flexibility]** Extending the polyhedral scheduling model capability, supporting time and space (linear/non-linear optimization) scheduling constraints of multi-dimensional accelerator architectures, and supporting heuristic search algorithms with customizable priorities, and achieving configurable modeling to support domain-specific architectures.
- **[Challenge 2: Scalability]** Optimizing the scheduling space to adapt to specific scenarios and solving the scheduling space expansion in the classic polyhedral scheduling models.
- **[Challenge 3: ILP Solving]** Implementing an ILP solver suitable for polyhedral scheduling models to minimize the solving time while ensuring correctness, so as to speedup scheduling space searching.

Current Situation

- The classic polyhedral scheduling algorithms do not support DSA-oriented scheduling constraints or multi-priority search algorithms. The scheduling results require a large amount of manual optimization and modification.
- The classic polyhedral scheduling algorithms use a one-size-fits-all modeling, resulting in huge scheduling space in typical DNN networks (ResNet/Bert/GPT3) were operators are fused.
- Polyhedral scheduling is about multi-objective optimization. Generally, the solver is called hundreds of times. Each solving process needs to be completed within milliseconds. However, mainstream solvers still require further improvements to meet this requirement.

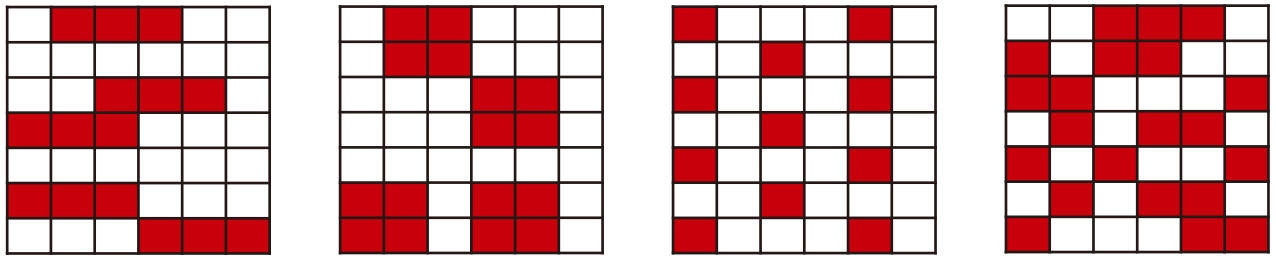
Technical Requirements

- **DSA hardware constraint modeling**: extend polyhedral scheduling algorithm to support time and space scheduling constraints and implement a heuristic search that supports user-defined priorities for DSA hardware such as Ascend 910.
- **Second-level solution for polyhedral scheduling**: achieve adaptive modeling, reduce the scheduling space to one third of the original and improve solution performance by 10 times in typical DNN (ResNet/Bert/GPT3) fusion scenarios so that the solution process of the scheduling algorithm can be completed in seconds and theoretically optimal solution can be achieved in 90% of the scenarios.

Challenge 4: Accelerated Sparse Computation Support

Structured tensor sparsity support for performance improvement

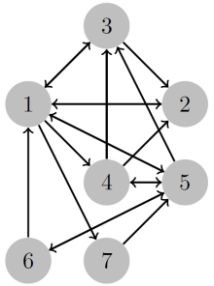
- Opportunities to limit overhead w.r.t. unstructured/random sparsity
- Regular enough for polyhedral compilation ☺



1D blocked | 2D blocked | Strided | Block-balanced (row)

Unstructured sparsity support to address particular problem classes, e.g., scientific computing or graph neural networks

- Graph neural networks (GNNs) include the processing of the graph adjacency matrix, sparse by nature



$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

- How to handle sparsity support in an optimized way?

Challenge 5: Trading Accuracy For Speed, The Wild Wild Way

Optimizing compilation community is strong about preserving semantics, but…

AI/DL is full of approximations

- Usage of low-precision floating point arithmetic
- Hyper-parameter selection (batch size, epoch, initialization, random attention matrices in transformers…)
- Quantization, sparsification, pruning
- Dropout layers
- Models built with lots of alchemy/art

Taking advantage of AI/DL models robustness

- Weak dependence analysis, weak synchronization
- Auto-adjusted shapes (no safe padding but plain removal of computation space parts, e.g., non-full tiles)
- Code generation restricted to computation core (special cases for convolution borders, really?)
- What is actually critical? How far can we go?

[Poly + Approximation: Schmitt et al. Automatic adaptive approximation for stencil computations. CC 2019]

Challenge 6: Rare Polyhedral Compilation Expertise

Industry has a high interest in the polyhedral technology

Tensor Comprehensions

Facebook – Tensor Comprehensions – The Next 700 Accelerated Layers: From Mathematical Expressions of Network Computation Graphs to Accelerated GPU Kernels, Automatically, TACO 2019

plaidML

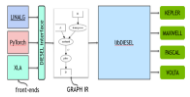
Intel – Stripe – Tensor Compilation via the Nested Polyhedral Model, 2019

MLIR

Google – MLIR – A Compiler Infrastructure for the End of Moore's Law, 2020

MindSpore

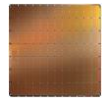
Huawei – MindSpore – AKG: Automatic Kernel Generation for Neural Processing Units using Polyhedral Transformations, PLDI 2021



NVIDIA – Diesel – DSL for linear algebra and neural net computations on GPUs, MAPL 2018

R-Stream

Qualcomm (Reservoir Labs) – R-stream – Automatic Mapping and Optimization to Kokkos with Polyhedral Compilation. HPEC 2020



Cerebras - Generating SIMD Instructions for Cerebras CS-1 using Polyhedral Compilation Techniques, IMPACT 2020

Expertise resource is scarce

> Would an explosion of Polyhedral DL frameworks be good ?
> - Expect it to be limited by small pool of polyhedral experts
>
> **Benoît Meister – IMPACT keynote 2019**

- Efforts required to promote and teach polyhedral compilation
- New doctors needed (wanted!) along with their fundamental research

Contents

HUAWEI

Conclusions And Invitation

New polyhedral framework for AI/DL

- Raising challenge (from AST to polyhedral representation) from usual compilers is fully solved
  - Replaced by more scientifically interesting operator fusion strategy selection
- But all other challenges are even bigger!
  - Scalability is a top concern with much more statements, tensor dimensions and iteration domain dimensions
  - Extracting parallelism isn't heroic but exploiting it for performance is the real issue
- Innovation related to the all-scenario context
  - Scheduling is influenced by injecting specific optimization constraints decided by non-linear optimization
    - Specific "MindTricks" influence constraint tree abstraction and adapted scheduling construction
    - Additional constraints and priorities may be adapted to input problem and target architecture
    - Enable better global optimization and simplify the compiler design by avoiding rescheduling passes
    - 1.7x geomean speedup over classic polyhedral scheduling on GPU
  - Symbolic approach for determining best tile sizes and reduce/remove the need for tuning

A space for (polyhedral) breakthrough opportunities

- Already efficient but many open challenges: flexibility, scalability, sparsity, approximation, etc.
- Open source, access to end-to-end AI/DL scenarios and usecases
- You are kindly invited to test, compare, contribute ☺

  https://gitee.com/mindspore/akg

Thank you.

This presentation and recording belong to Huawei Technologies Co. Ltd.
No distribution is allowed without Huawei Technologies Co. Ltd.’s permission

Contact: cedric.bastoul@huawei.com

Bring digital to every person, home and organization for a fully connected, intelligent world.