AI编译器-系列之前端优化

代数化简

ZOMI

Talk Overview of Frontend Optimizer

I. AI 编译器前端优化

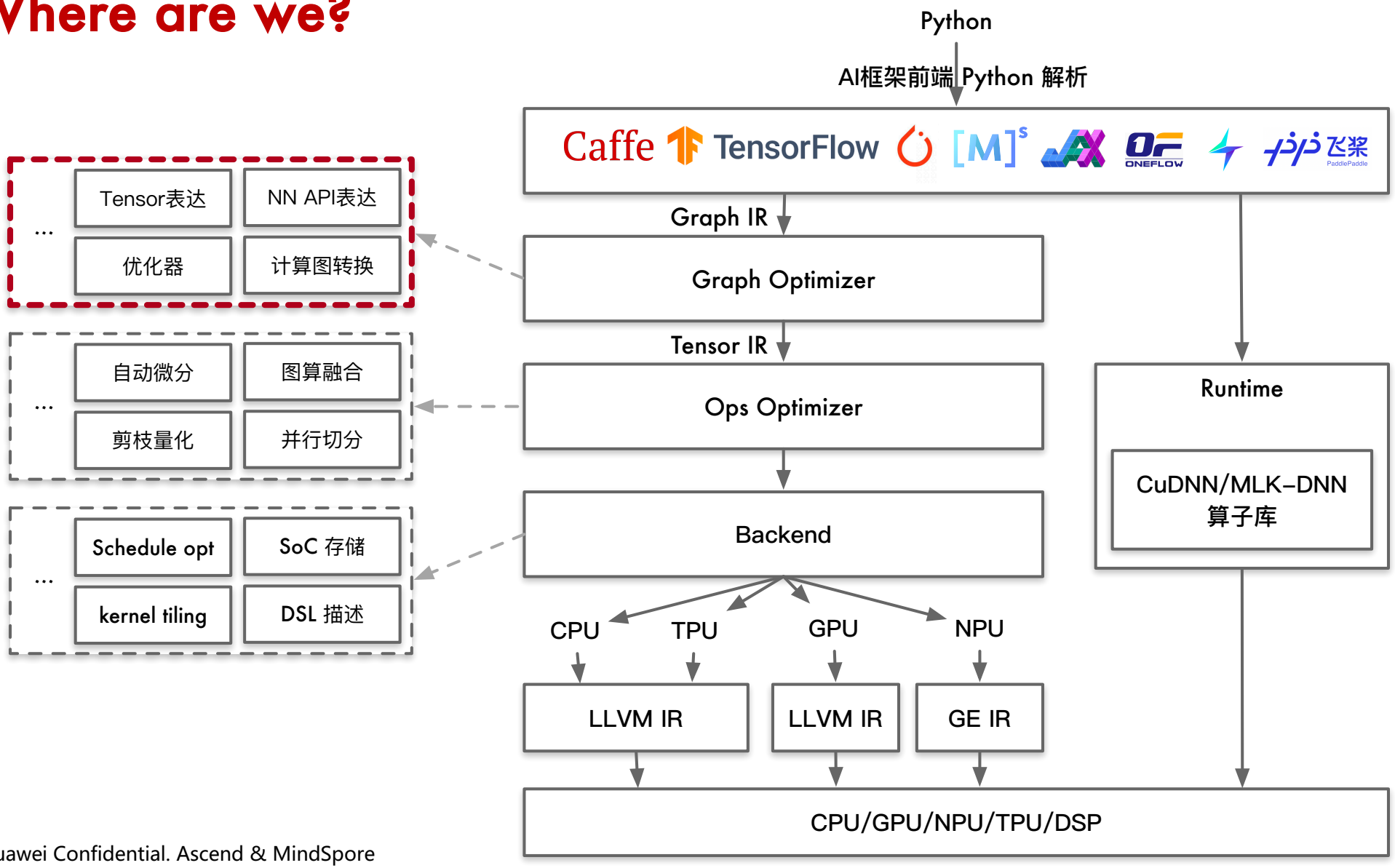
- 图层 - Graph IR
- 算子融合 - OP Fusion
- 布局转换 - Layout Transform
- 内存分配 - Memory Allocation
- 常量折叠 – Constant Fold
- 公共子表达式消除 - CSE
- 死代码消除 - DCE
- 代数简化 – Algebraic Reduced

Talk Overview

algebraic reduced – 代数化简

- 算术化简
- 运行化简
- 广播化简

Where are we?



Principle

- 代数化简的目的是利用交换率、结合律等规律调整图中算子的执行顺序，或者删除不必要的算子，以提高图整体的计算效率。

Algorithm

代数化简可以通过子图替换的方式完成，具体实现：

- 可以先抽象出一套通用的子图替换框架，再对各规则实例化。
- 可以针对每一个具体的规则实现专门的优化逻辑。

算术化简

通过利用代数之间算术运算法则，在计算图中可以确定优化的运算符执行顺序，从而用新的运算符替换原有复杂的运算符组合。

算术化简 I

- 结合律化简：

$$(A \circledast B)^{-1} \odot ((A \circledast B)C)^{-1} \rightarrow (A \circledast B)^{-2} \odot C$$

$$Recip(A) \odot Recip(A \odot B) \rightarrow Square\big(Recip(A)\big) \odot B$$

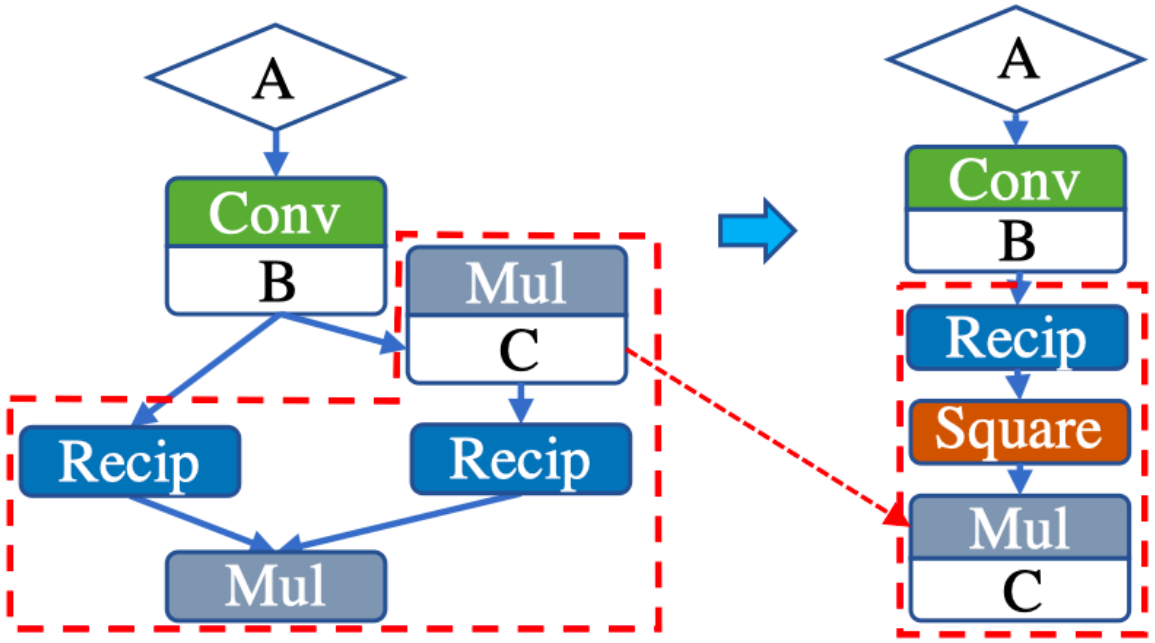
$$(A \odot \sqrt{B}) \odot (\sqrt{B} \odot C) \rightarrow A \odot B \odot C$$

$$(A \odot ReduceSum(B)) \odot (ReduceSum(B) \odot C) \rightarrow A \odot Square(ReduceSum(B)) \odot C$$

算术化简 I

- 结合律化简：

$$(A \circledast B)^{-1} \odot ((A \circledast B)C)^{-1} \rightarrow (A \circledast B)^{-2} \odot C$$



算术化简 II

- 提取公因式、分配律化简：

$$A \odot C + A \odot B \rightarrow (\mathrm{B} + C) \odot A$$

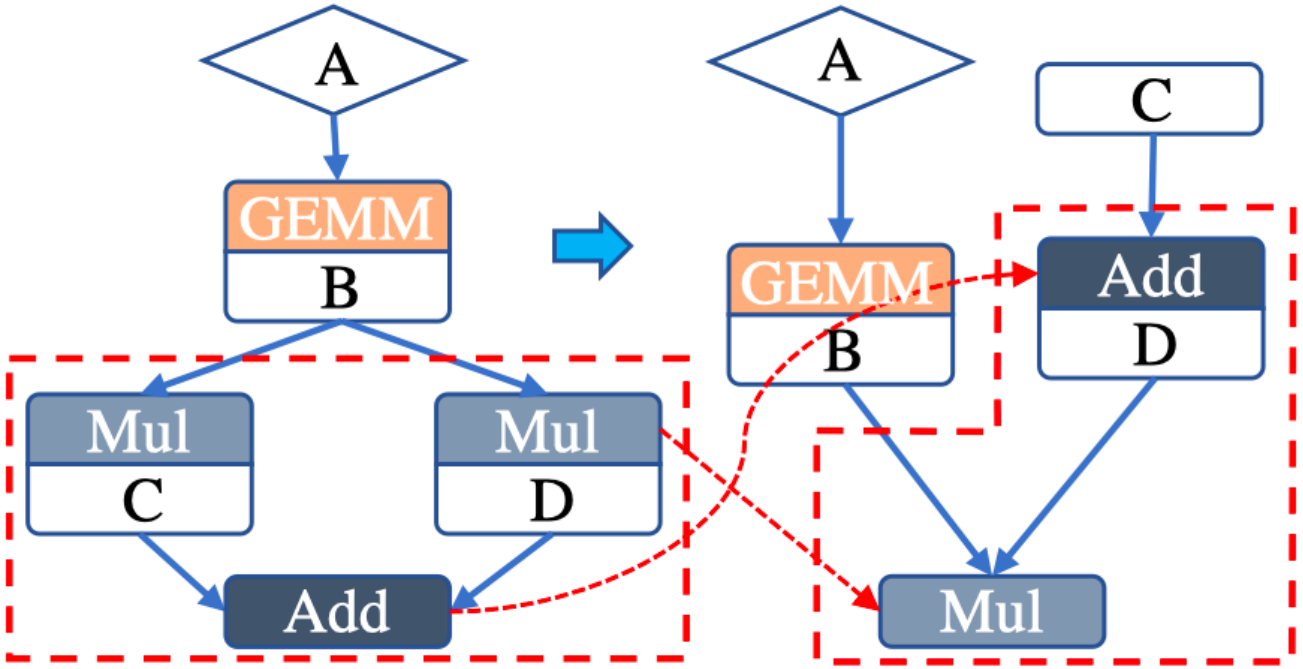
$$A + A \odot B \rightarrow A \odot (B + 1)$$

$$Square(A + B) - (A + B) \odot C \rightarrow (A + B) \odot (A + B - C)$$

算术化简 II

- 提取公因式、分配律化简：

$$(A \cdot B) \odot D \ + \ (A \cdot B) \odot C \rightarrow (A \cdot B) \odot (C + D)$$

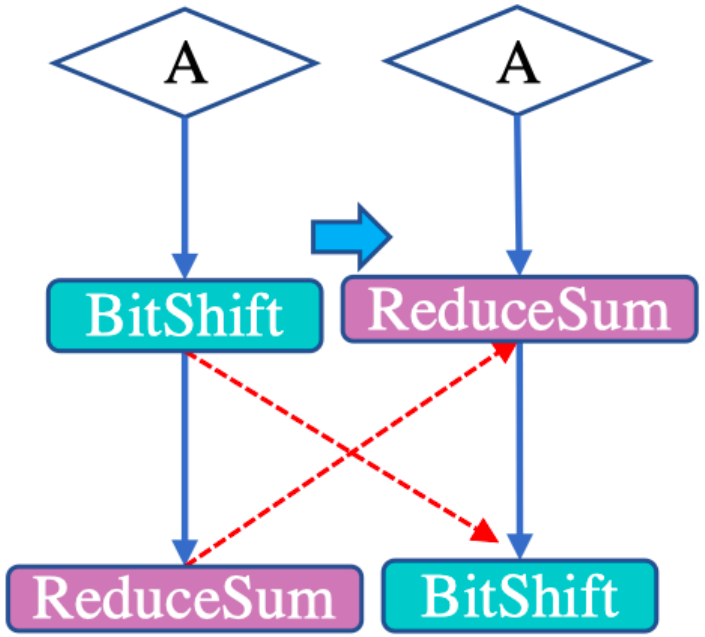


算术化简 III

- 交换律化简：

$$ReduceSum(BitShift(A)) \rightarrow BitShif\ t(ReduceSum(A))$$

$$ReduceProd(Exp(A)) \rightarrow Exp(ReduceSum(A))$$



DNNFusion: accelerating deep neural networks execution with advanced operator fusion.

Property	Without graph rewriting		With graph rewriting	
	Graph structure in equation	#FLOPS	Graph structure in equation	#FLOPS
Associative	$Recip(A) \odot Recip(A \odot B)$	$4 * m * n$	$Square(Recip(A)) \odot B$	$3 * m * n$
	$(A \odot \sqrt{B}) \odot (\sqrt{B} \odot C)$	$5 * m * n$	$A \odot B \odot C$	$2 * m * n$
	$Abs(A) \odot B \odot Abs(C)$ †	$4 * m * n$	$Abs(A \odot C) \odot B$	$3 * m * n$
	$(A \odot ReduceSum(B)) \odot (ReduceSum(B) \odot C)$ ¶	$5 * m * n$	$A \odot Square(ReduceSum(B)) \odot C$	$3 * m * n + m$
Distributive	$A \odot C + A \odot B$	$3 * m * n$	$(A + B) \odot C$	$2 * m * n$
	$A + A \odot B$	$2 * m * n$	$A \odot (B + 1)$	$2 * m * n$ §
	$Square(A + B) - (A + B) \odot C$	$5 * m * n$	$(A + B) \odot (A + B - C)$	$3 * m * n$
Commutative	$A \odot B$	$m * n$	$B \odot A$	$m * n$ ‡
	$ReduceSum(BitShift(A))$ ¶	$2 * m * n$	$BitShift(ReduceSum(A))$	$m * n + m$
	$ReduceProd(Exp(A))$ ¶	$2 * m * n$	$Exp(ReduceSum(A))$	$m * n + m$

§ Although #FLOPS is not reduced, A is loaded once instead of twice.

† First use commutative property to swap B and $Abs(C)$, then apply associative property.

‡ Even though this pattern has no #FLOPS gains, it can enable further optimization, e.g the case of †.

¶ #FLOPS is calculated by assuming the reduction of ReduceSum/ReduceProd is along with the inner-most dimension.

运行化简

减少运算或者执行时候，冗余的算子或者算子对

运行化简

- 对逆函数等于自身函数的对合算子化简，如取反、倒数、逻辑非、矩阵转置：

$$f\big(f(x)\big) = x$$

$$f(x) = f^{-1}(x)$$

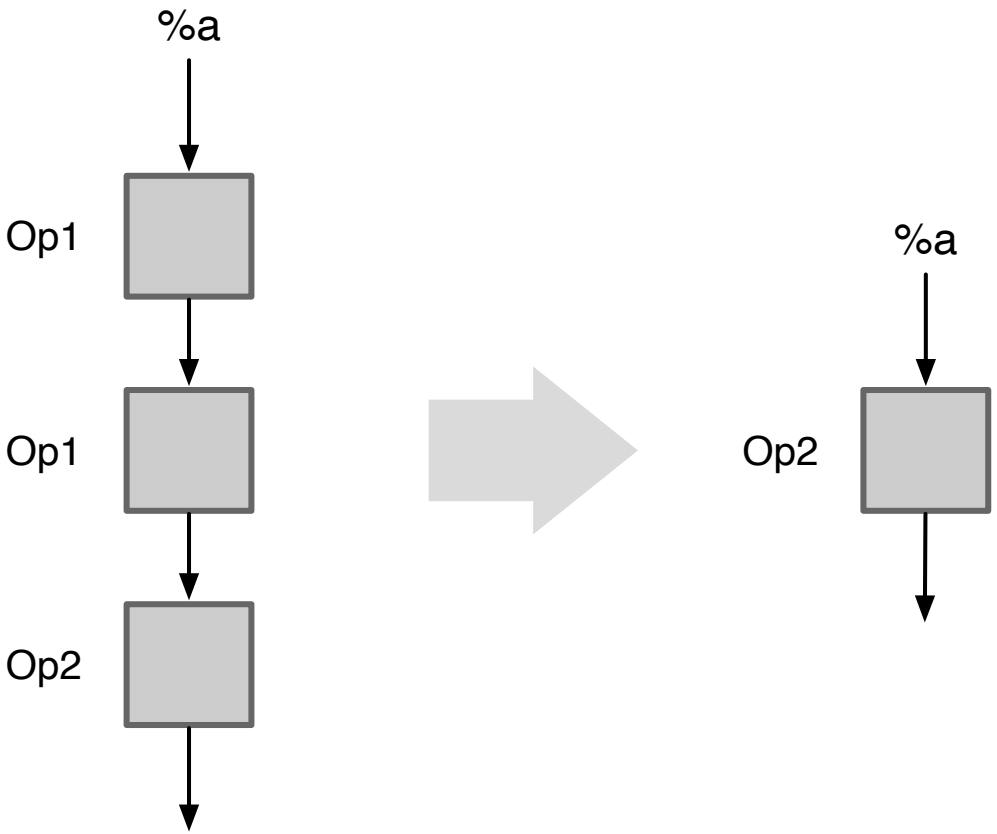
- 幂等算子化简，作用在某一元素两次与一次相同：

$$f\big(f(x)\big) = f(x)$$

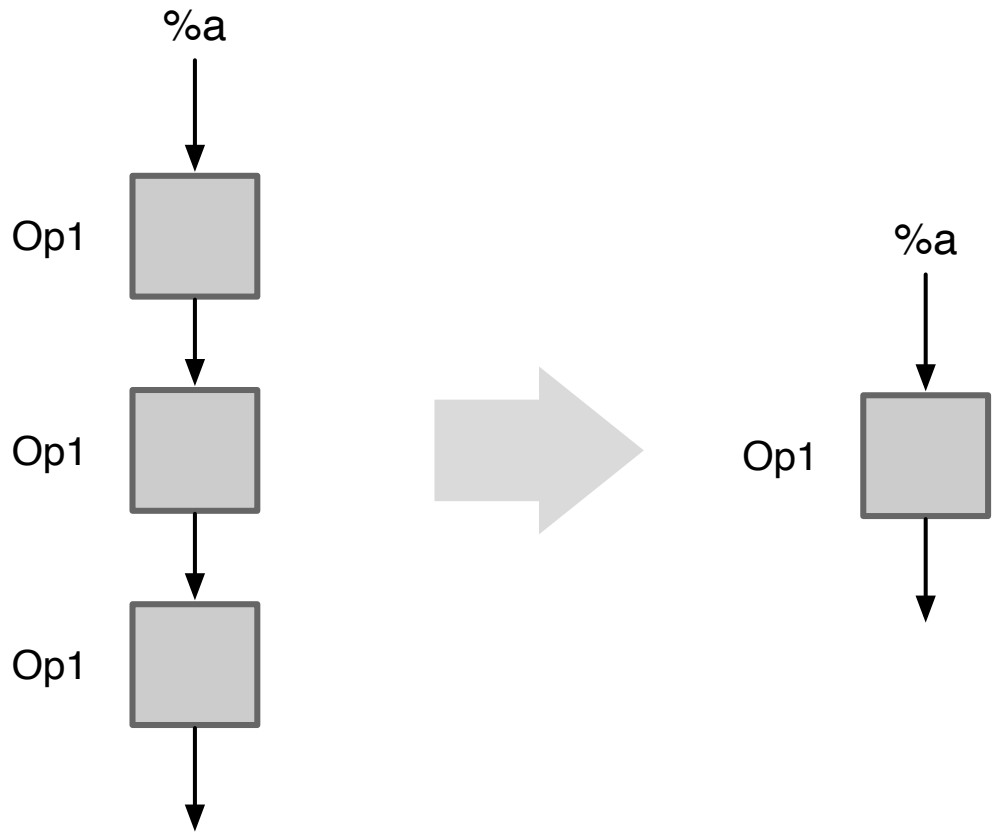
$$Reshape(Reshape(x, shape1), shape2) \rightarrow Reshape(x, shape2)$$

运行化简

- 对合算子化简：



- 幂等算子化简：



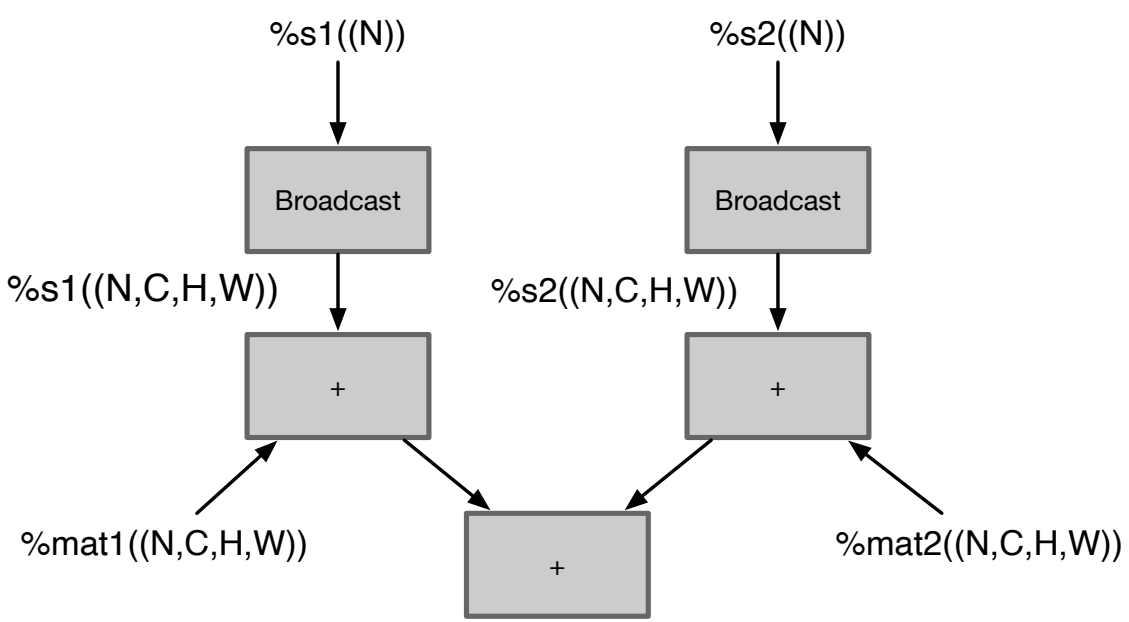
广播化简

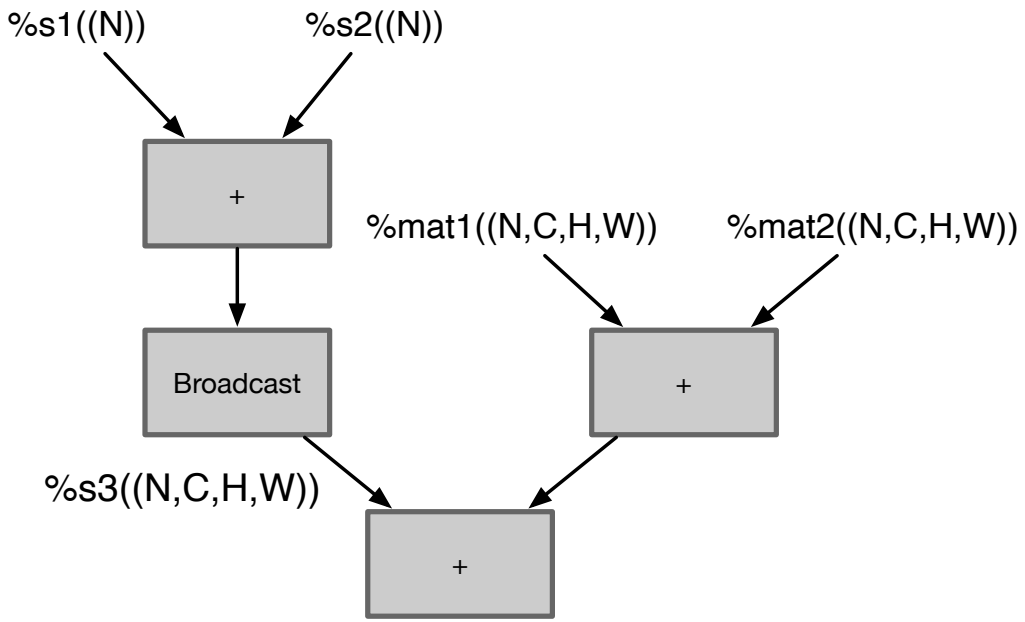
多个张量形状 Shape 不同，需要进行广播将张量的形状拓展为相同 shape 再进行运算，化简为最小计算所需的广播运算数量。

广播化简

- 位置替换：

$$(\boldsymbol{Mat_1} + S_1) + (\boldsymbol{Mat_2} + S_2) \rightarrow (\boldsymbol{Mat_1} + \boldsymbol{Mat_2}) + (S_1 + S_2)$$





Reference

1. Niu, Wei, et al. "DNNFusion: accelerating deep neural networks execution with advanced operator fusion." Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. 2021