Microsoft

人工智能系统 System for AI

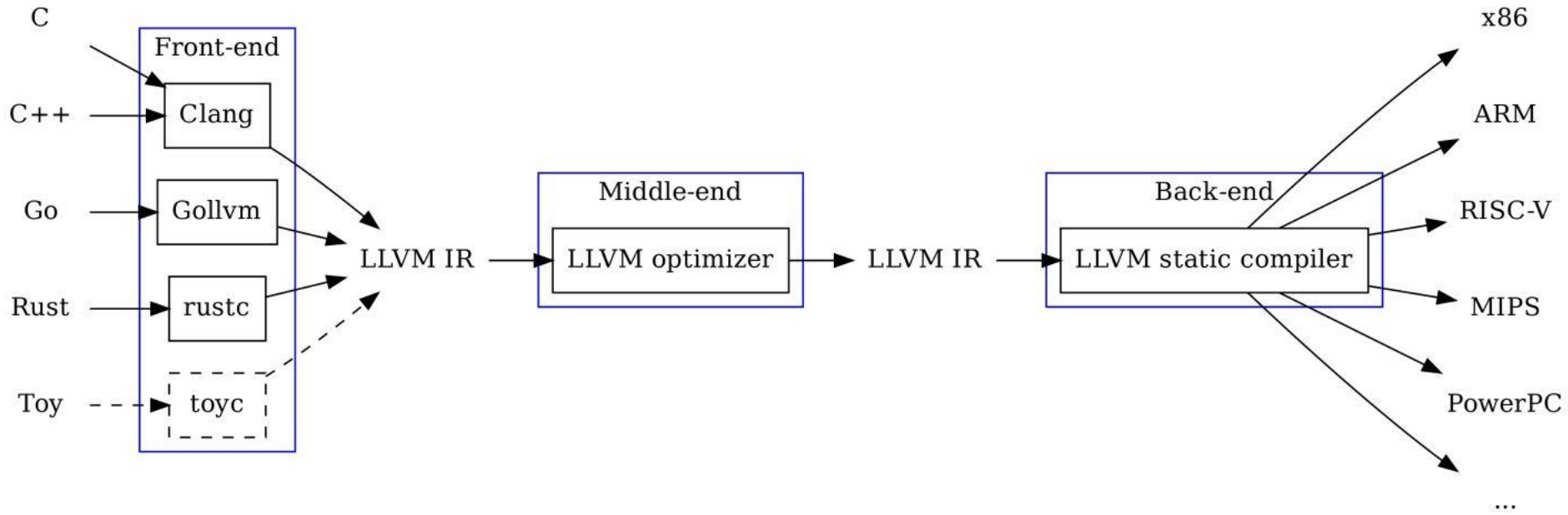
计算图的编译与优化
Computation Graph Compilation and Optimization

主要内容

- 深度神经网络编译器
- 计算图优化
- 内存优化
- 内核优化
- 调度优化

编译器

- **编译器（compiler）**是一种计算机程序，它会将某种编程语言写成的源代码（原始语言）转换成另一种编程语言（目标语言）
- 核心概念：前端、后端、中间表达、优化过程
- 举例：LLVM



神经网络编译器

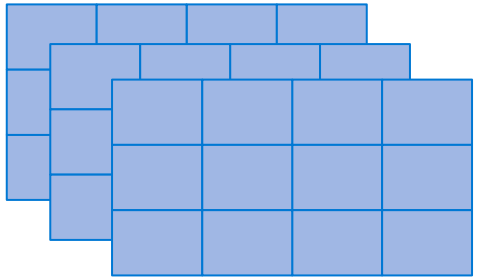
- 前端：基于Python的DSL语言
  - 如PyTorch, TensorFlow等
- 后端：神经网络加速器
  - 如GPU, CPU, FPGA, TPU, IPU等
- 中间表达：计算图、算子表达式
  - 如DAG, TVM IR
- 优化过程：
  - 计算图优化、内存优化、内核优化、调度优化

大纲

- 1. **计算图优化**
- 2. 内存优化
- 3. 内核优化
- 4. 调度优化

中间表示--计算图

- 基本数据结构：Tensor （N维数组）
  - Tensor形状： [2, 3, 4]
  - 元素类型：int, float, string, etc.

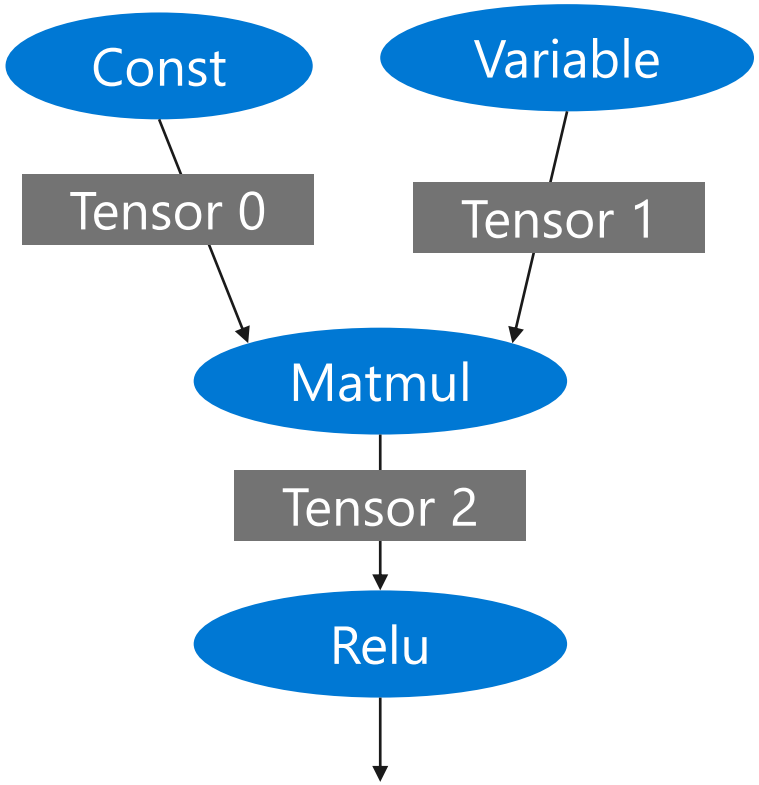


- 基本运算单元：Operator
  - 由最基本的代数算子组成
  - 每个Operator接收N个输入Tensor，并输出M个输出Tensor
  - TensorFlow中有>400个基本operator

Add	Log	While
Sub	MatMul	Merge
Mul	Conv	BroadCast
Div	BatchNorm	Reduce
Relu	Loss	Map
Tanh	Transpose	Reshape
Exp	Concatenate	Select
Floor	Sigmoid	.....

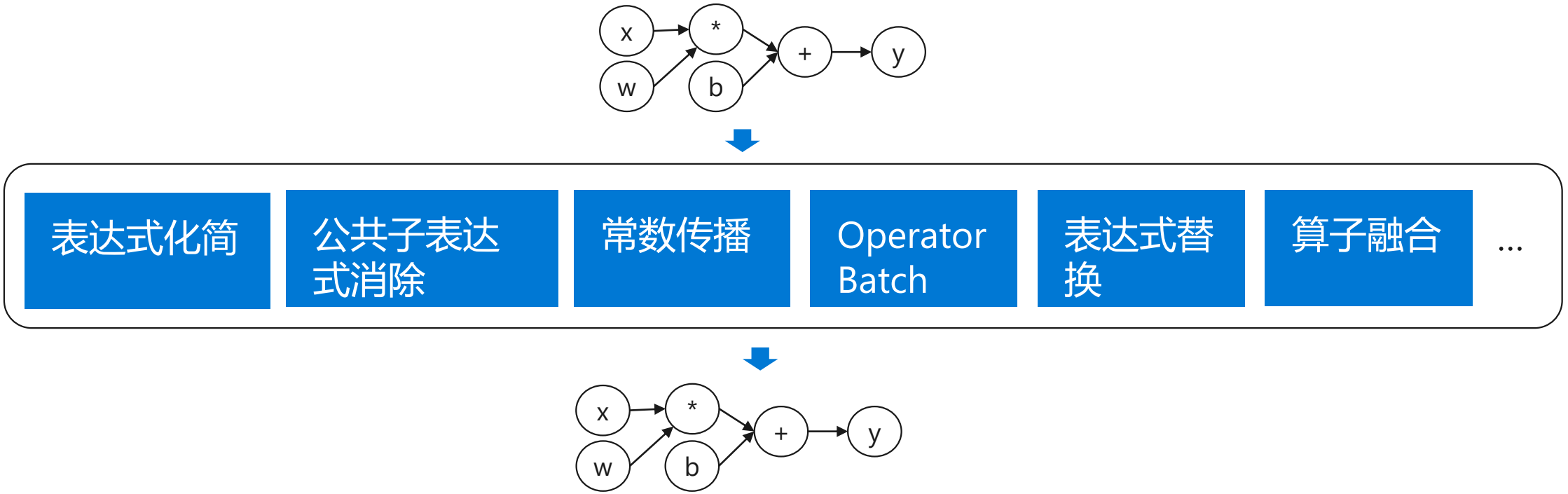
中间表示--计算图

- 用数据流图表示计算逻辑和状态
  - 节点表示Operator
  - 边表示Tensor
- 计算状态（如参数）也是Operator
  - 如Variable Operator
- 特殊的Operator
  - 如：Switch, Merge, While 等用来构建控制流
- 特殊的边
  - 如：控制边用来表示节点之间的依赖关系



图优化

- 目标：通过图的等价变换化简计算图，从而降低计算复杂度或内存开销。
- 数据流图作为深度学习框架中的高层中间表示，可以允许任何等价图优化Pass去化简计算流图或提高执行效率



图优化（1）：算术表达式化简

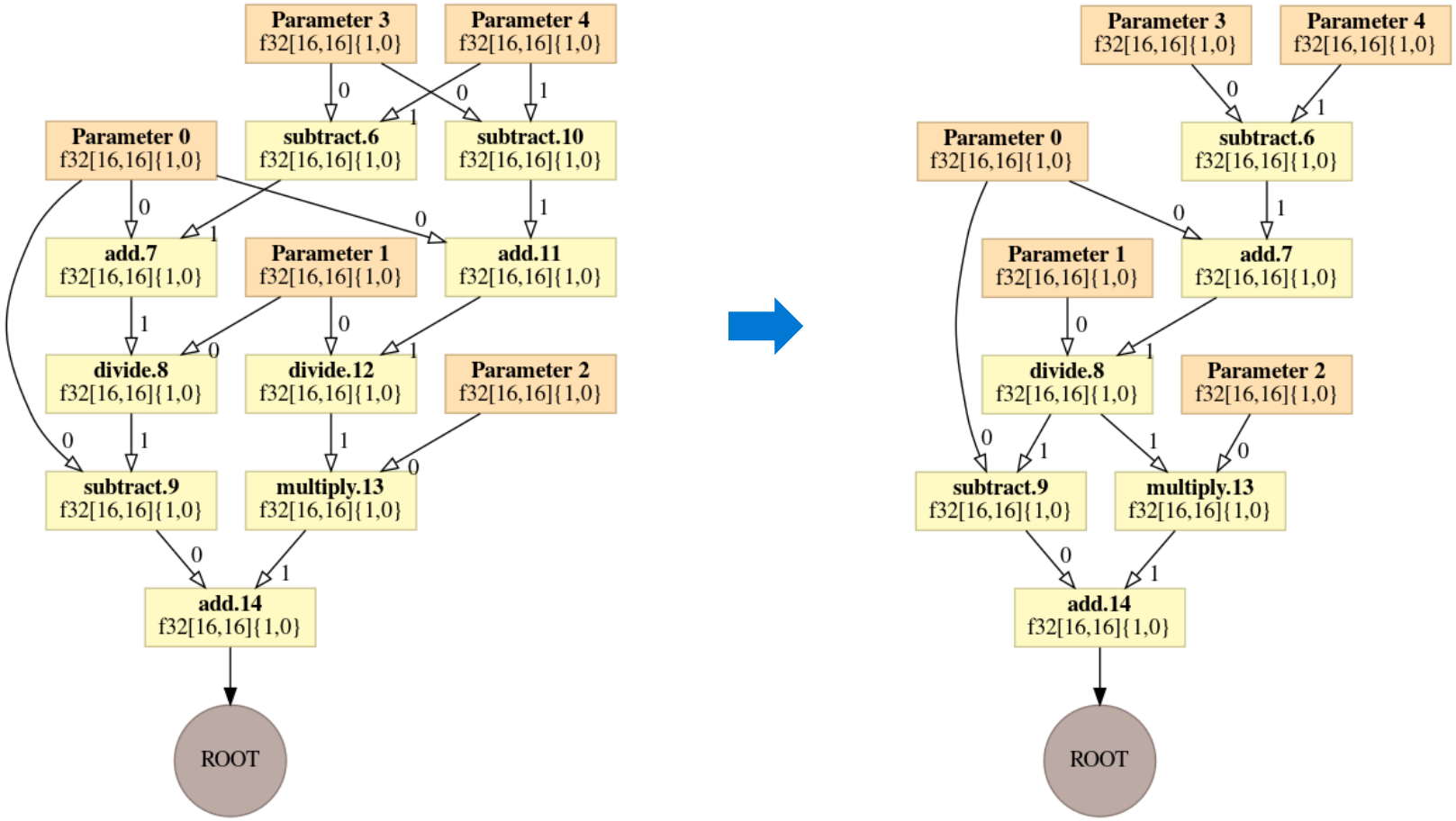
- 通过代数运算等价变换化简计算图，如：
- a * 0 -> 0
- a * broadcast(0) -> broadcast(0)
- a * 1 -> a
- a * broadcast(1) -> a
- a + 0 -> a
- a + broadcast(0) -> a
- log(exp(x)/y) -> x - log(y)

…

图优化（2）：公共子表达式消除

- 目的是将相同输入的表达式进行消除，由一个节点来代替，复用计算结果
- 思考：如何实现

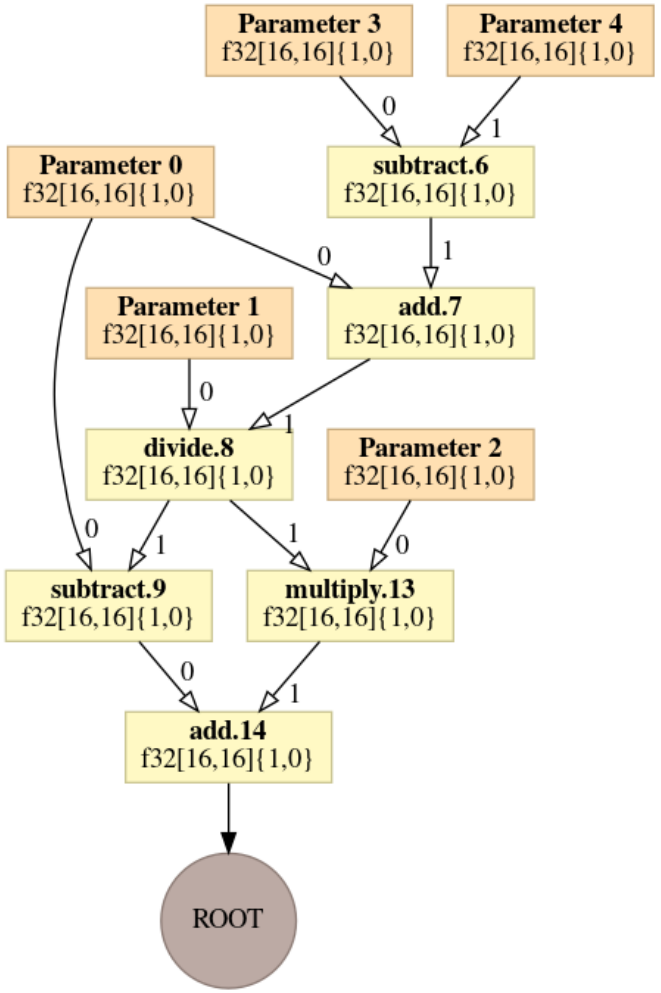
p1 / (p0 + (p3 - p4))



图优化（3）：常数传播

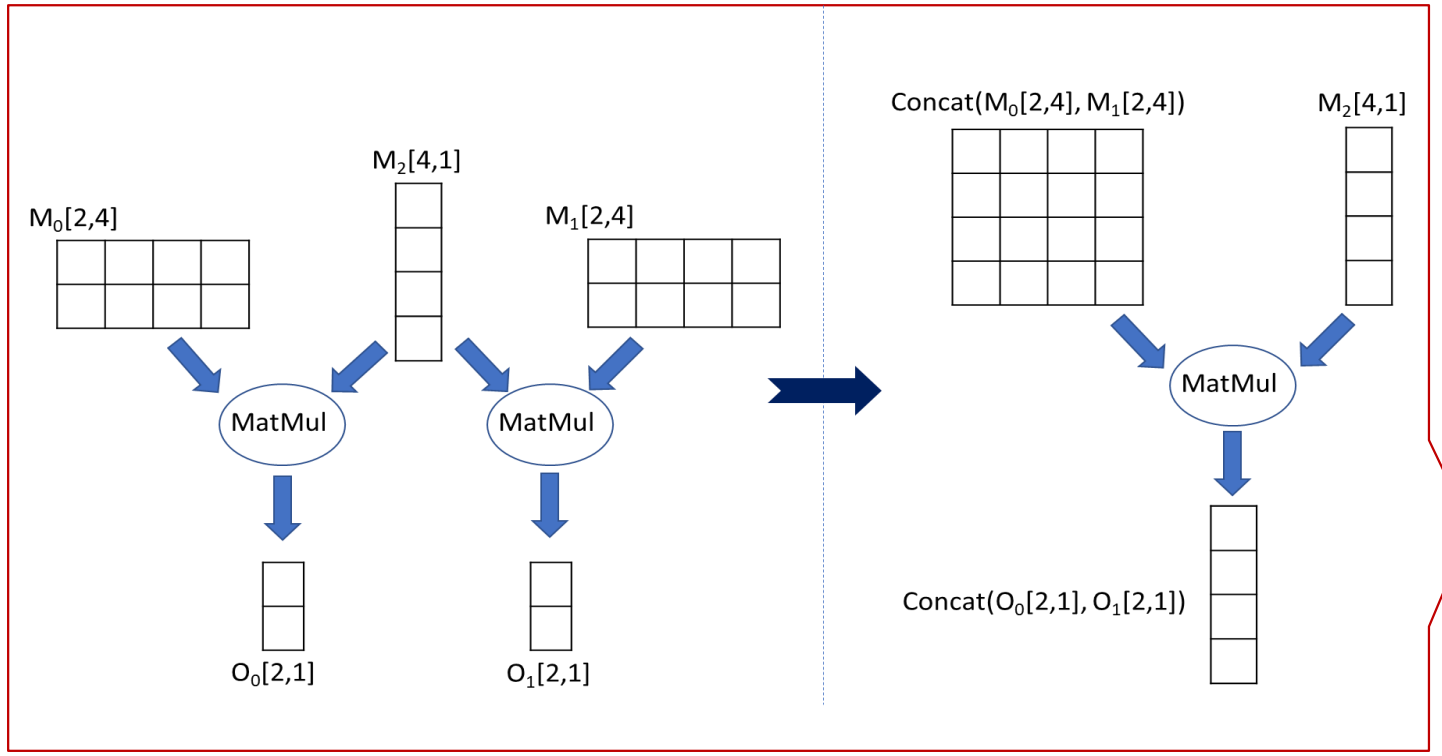
- 如果一个算子的所有输入张量都是常数的话，那么该算子的结果也为常数张量
- 在编译器计算并化简
- 例子：假设参数0和参数1为常数张量，最终的图可以化简为什么？

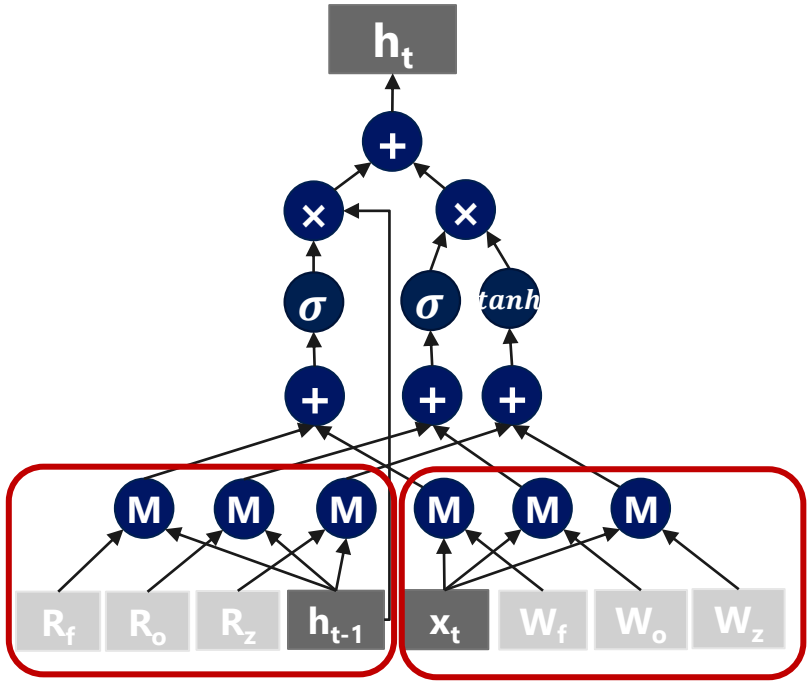
- 注意：常数传播可能会引起内存的扩张



图优化（4）：GEMM自动融合

- Batch same-type operators to leverage GPU massive parallelism

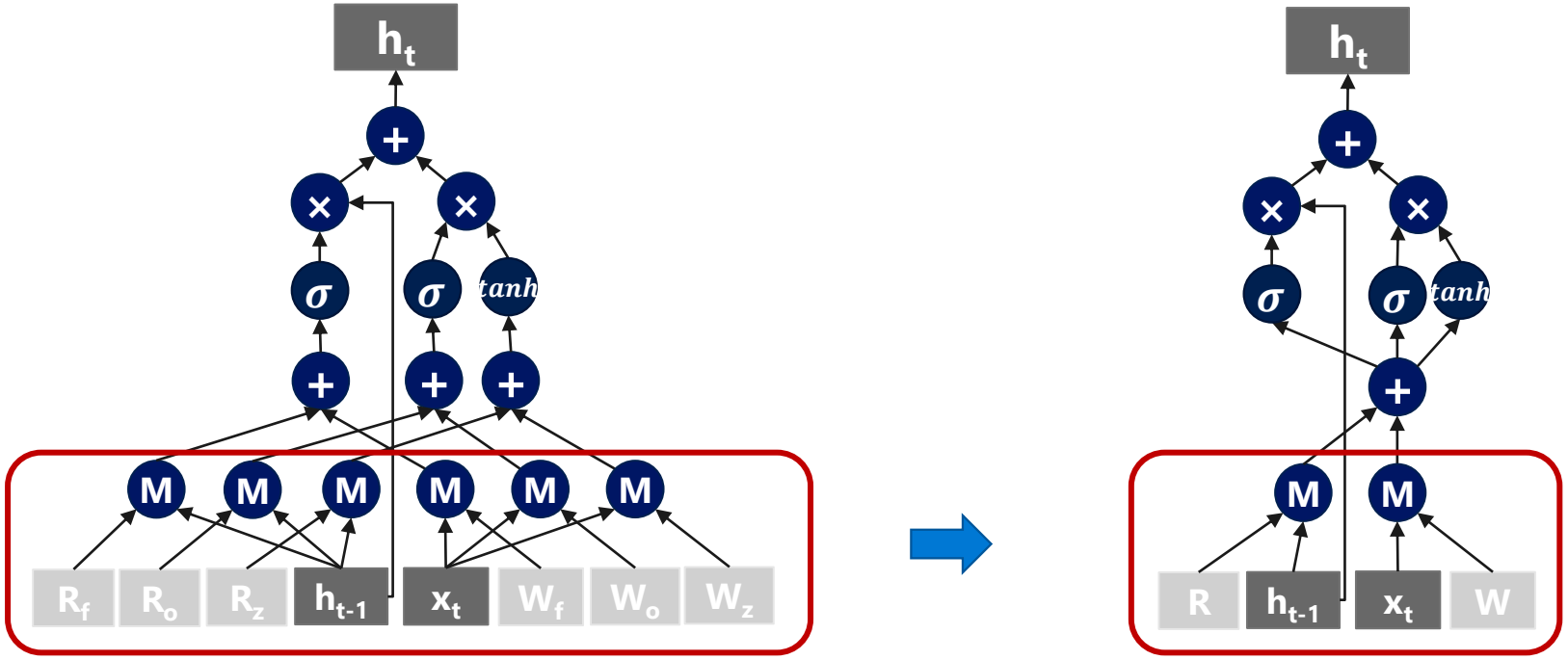




Data-flow graph of a GRU cell

图优化（4）：GEMM自动融合

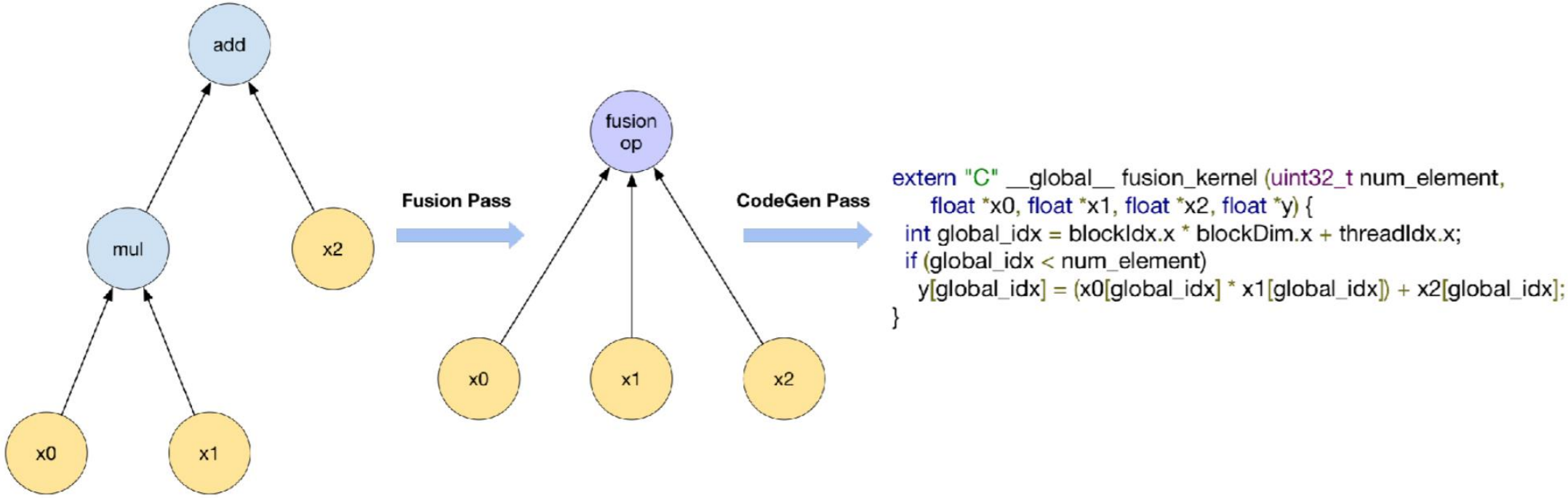
- 通过将输入张量合并成一个大的张量来实现将相同的算子合并成一个更大的算子，从而更好的利用硬件并行度



Data-flow graph of a GRU cell

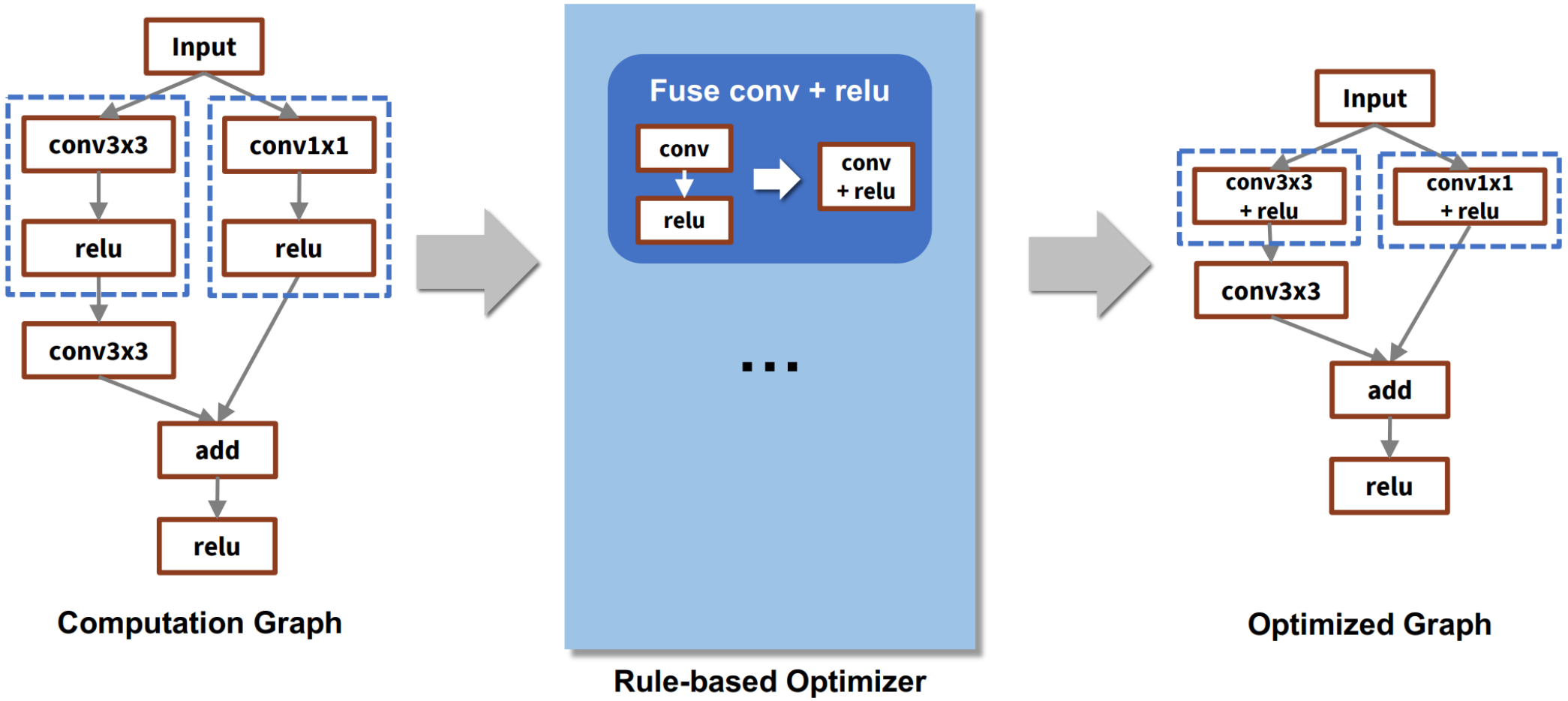
图优化（5）：算子融合

- 向量化的多个算子的操作可以合并成一个向量化操作 （）
  - 减少内核启动开销
  - 减少蹭内存的读取，提高计算密度



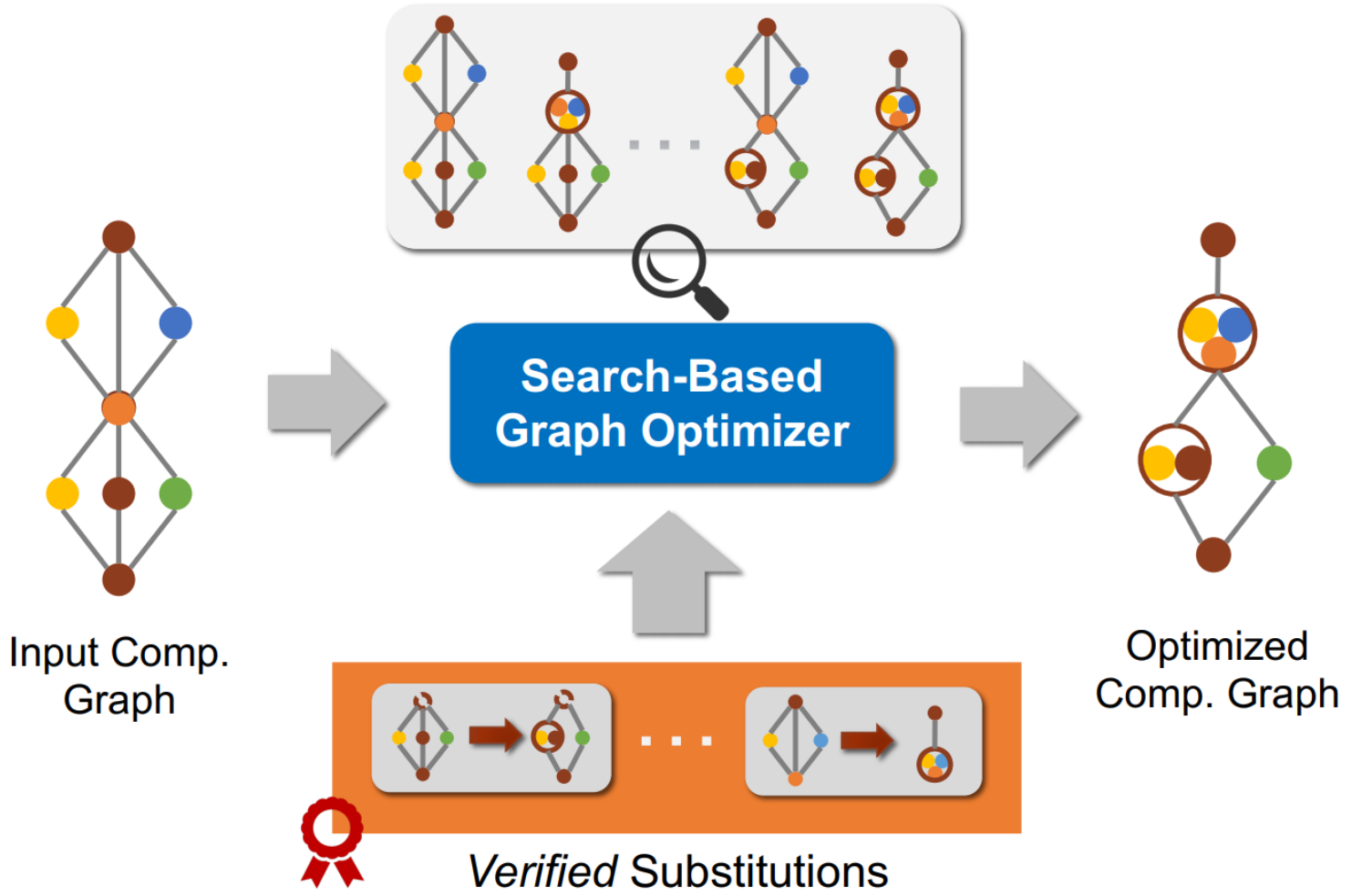
图优化（6）：子图替换

· 利用子图匹配识别出可替换的复杂子图，替换为更高效的合并算子



图优化（6）：随机子图替换

- TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions



小结：图优化的总结

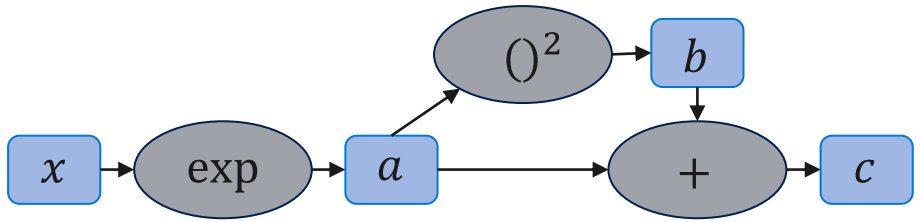
- 计算图作为深度学习编译框架的第一层中间表示
- 基于计算图的优化算法：
  - 算术表达式化简
  - 公共子表达式消除
  - 常数传播
  - 矩阵合并
  - 算子融合
  - 子图替换/随机子图替换
- 思考：
  - 你可以想到哪些其它的计算图上的优化？
  - 计算图还有哪些其它的好处？

大纲

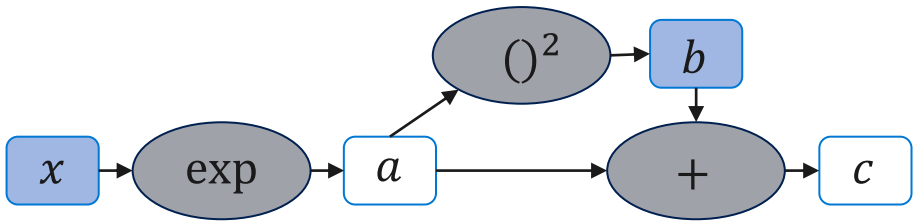
- 1. 计算图优化
- 2. **内存优化**
- 3. 内核优化
- 4. 调度优化

内存优化

- 目标：图过对计算图的变化以及张量的合理分配来降低内存使用的总量。
- 举例：



Total memory=4N



Total memory=2N

内存优化（1）：基于拓扑序的最小内存分配

- 将计算流图按照某种拓扑序进行排序
  - 如BFS，Reverse DFS等
- 按照节点的拓扑顺序依次分配其使用到的输出张量
- 当一个张量后面没有其它算子使用时，则回收到内存池
- 当所以张量分配完成后，内存池的最大分配空间就是该计算图需要的最小内存

- 拓扑序的选择会同时影响模型的计算时间和最大内存占用

内存优化（2）：根据整数线性规划求解最优内存放置

Goal

Minimize execution time

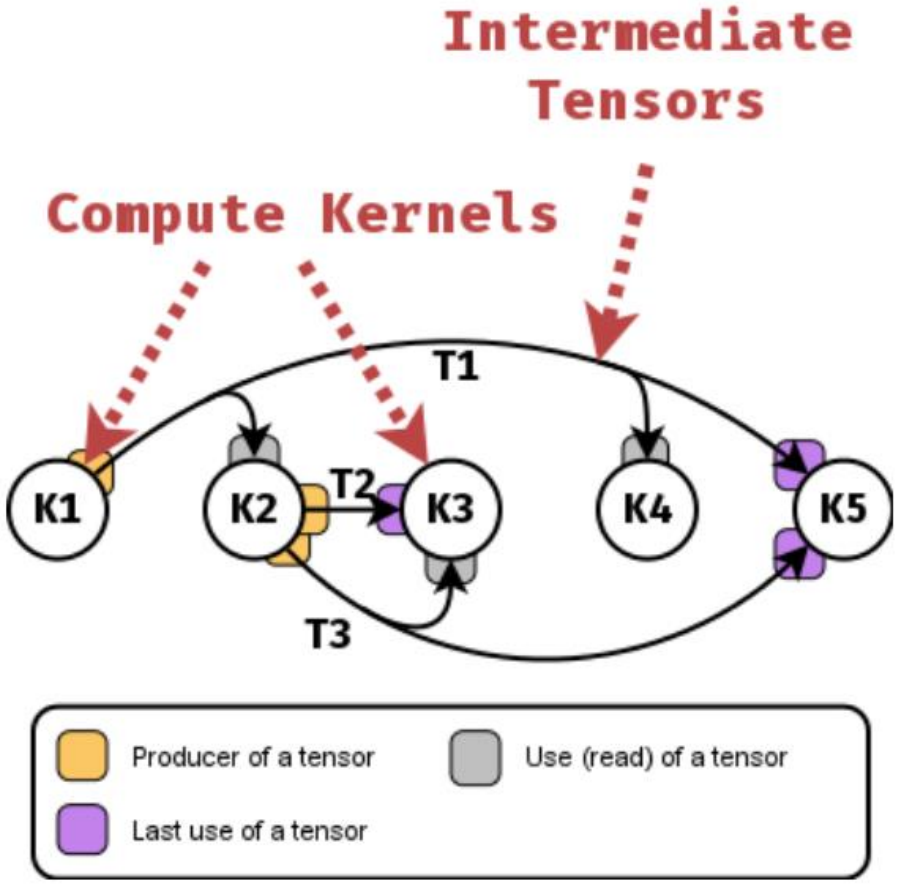
- Arbitrary computation graph
- Size constraint on **fast** memory

How

- Place tensors in **fast** or **slow** memory.
- Optimal tensor **movement**

Strategy

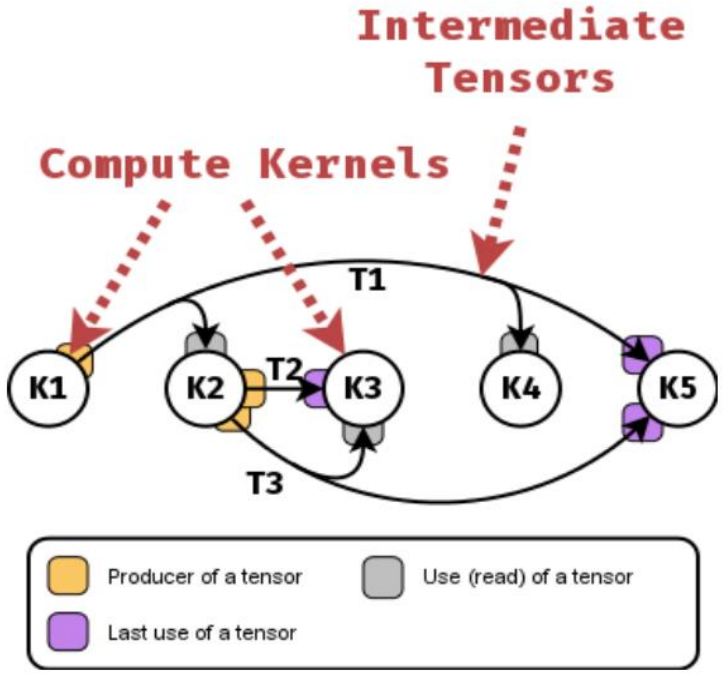
- Profile kernel performance.
- Model tensor assignment as ILP.



AutoTM: Automatic Tensor Movement in HeterogeneousMemory Systems using Integer Linear Programming

内存优化（2）：根据整数线性规划求解最优内存放置

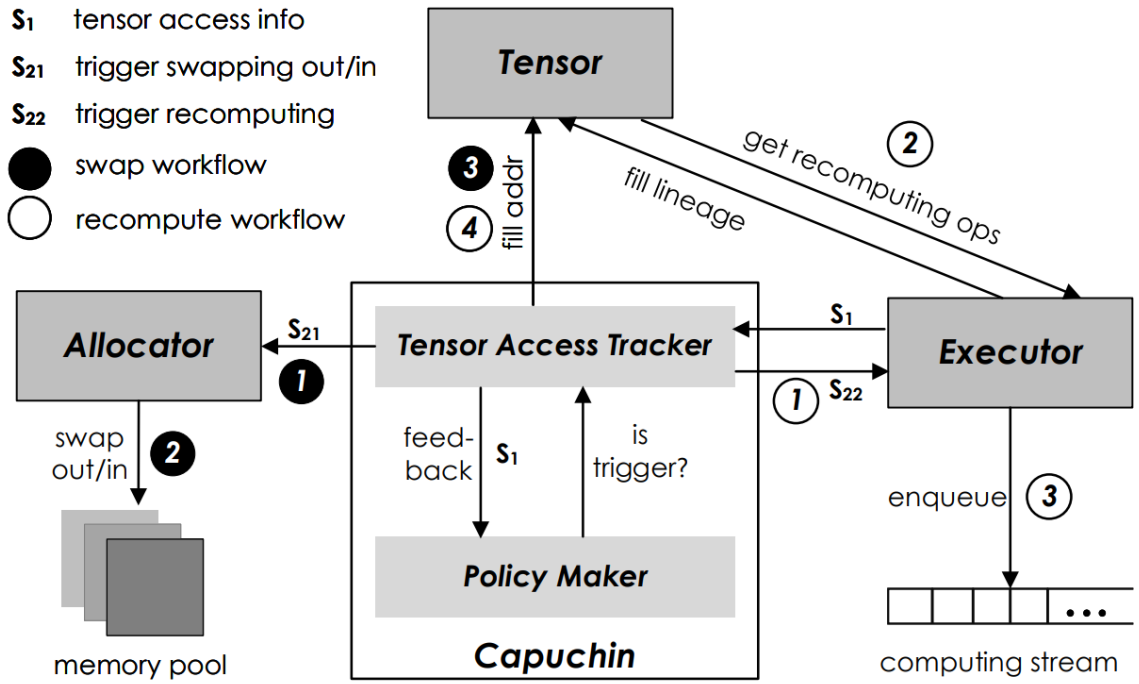
- 目标：给定任意的计算图最小化执行时间
- 约束：有限的快速内存，如GPU内存
- 变量：每一个张量是否放置在快速内存中，还是较慢的外存中（如CPU内存）
- 过程：最优化张量的移动
- 需求：每个内核计算的测量时间
- 方法：将上述优化问题建模为整数线性规划问题



AutoTM: Automatic Tensor Movement in HeterogeneousMemory Systems using Integer Linear Programming

内存优化（3）：张量换入换出与张量重计算

- DRAM存储量相对GPU显存来说比较大，可以将数据在GPU与DRAM之间进行转移，或者直接重新计算
- 举例：
  - Capuchin: Tensor-based GPU Memory Management for Deep Learning



小结：内存优化的总结

- 基于计算图的内存优化算法：
  - 基于拓扑序的最小内存分配
  - 根据整数线性规划求解最优内存放置
  - 张量换入换出与张量重计算

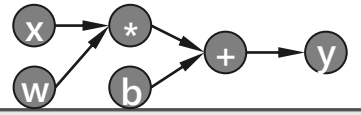
- 思考：
  - 你可以想到哪些其它的内存优化方法？

大纲

- 1. 计算图优化
- 2. 内存优化
- 3. **内核优化**
- 4. 调度优化

内核优化与内核生成

计算图IR（DAG）



计算图优化

问题: 每个后端平台针对每个算子都需要单独实现至少一个内核
考虑：编程模型、数据排布、线程模型、缓存大小等等

内核优化与内核生成


Caffe2
CNTK
计算图IR（DAG）
x
*
+
y
w
b
计算图优化
张量运算表达式
C[i] = A[i] + B[i]
算子优化与生成
intel
Xeon
processor

张量运算编译

- 核心思想：分离计算逻辑与调度逻辑
  - 通过张量运算表达式表示每个算子的通用计算逻辑
  - 通过调度语言描述算子在映射到具体硬件上时的调度空间
- 相关工作：
  - TVM, Halide, TACO, Tensor Comprehension, FlexTensor等
- 张量运算表达式：例：TVM IR
  - C = A * B
  - C = tvm.compute((m, n), lambda i, j: **tvm.sum(A[i, k] * B[k, j]),** axis=k)

其它张量运算表达式

Affine Transformation

```
out = tvm.compute((n, m), lambda i, j: tvm.sum(data[i, k] * w[j, k], k))
out = tvm.compute((n, m), lambda i, j: out[i, j] + bias[i])
```

Convolution

```
out = tvm.compute((c, h, w),
  lambda i, x, y: tvm.sum(data[kc,x+kx,y+ky] * w[i,kx,ky], [kx,ky,kc]))
```

ReLU

```
out = tvm.compute(shape, lambda *i: tvm.max(0, out(*i))
```

张量运算到代码生成

Tensor Expression Language

```
C = t.compute((m, n),
    lambda i, j: t.sum(A[i, k] * B[j, k], axis=k))
```

Key Idea:
Separation of Compute and Schedule
introduced by Halide

Schedule Optimizations

Hardware

例子：如何优化和生成一个向量加算子?

```
C = tvm.compute((n,), lambda i: A[i] + B[i])
s = tvm.create_schedule(C.op)
```

```
for (int i = 0; i < n; ++i) {
  C[i] = A[i] + B[i];
}
```

例子：如何优化和生成一个向量加算子?

```
C = tvm.compute((n,), lambda i: A[i] + B[i])
s = tvm.create_schedule(C.op)
xo, xi = s[C].split(s[C].axis[0], factor=32)
```

```
for (int xo = 0; xo < ceil(n / 32); ++xo) {
  for (int xi = 0; xi < 32; ++xi) {
    int i = xo * 32 + xi;
    if (i < n) {
      C[i] = A[i] + B[i];
    }
  }
}
```

例子：如何优化和生成一个向量加算子?

```
C = tvm.compute((n,), lambda i: A[i] + B[i])
s = tvm.create_schedule(C.op)
xo, xi = s[C].split(s[C].axis[0], factor=32)
s[C].recorder(xi, xo)
```

```
for (int xi = 0; xi < 32; ++xi) {
  for (int xo = 0; xo < ceil(n / 32); ++xo) {
    int i = xo * 32 + xi;
    if (i < n) {
      C[i] = A[i] + B[i];
    }
  }
}
```

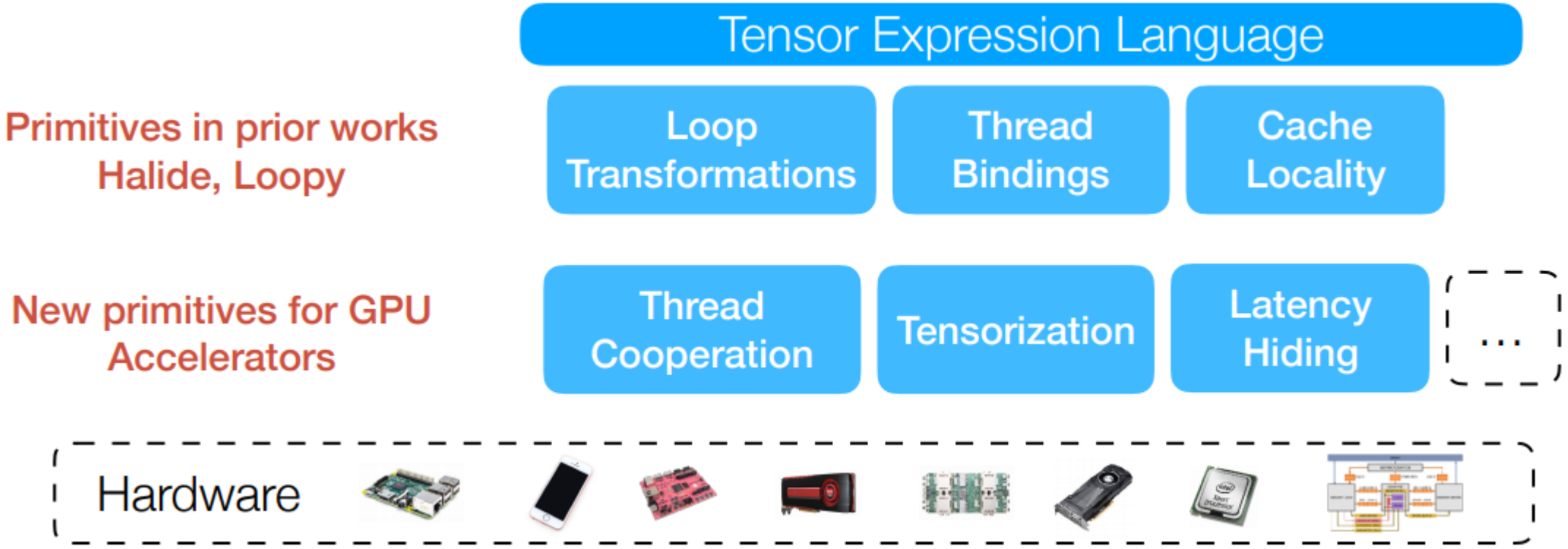
例子：如何优化和生成一个向量加算子?

```
C = tvm.compute((n,), lambda i: A[i] + B[i])
s = tvm.create_schedule(C.op)
xo, xi = s[C].split(s[C].axis[0], factor=32)
s[C].recorder(xi, xo)
s[C].bind(xo, tvm.thread_axis(“blockIdx.x”)
s[C].bind(xi, tvm.thread_axis(“threadIdx.x”)
```

```
int i = threadIdx.x * 32 + blockIdx.x;
if (i < n) {
   C[i] = A[i] + B[i];
}
```

其它算子调度优化

- 每一种优化都可能产生出多个内核代码的实现
- 利用自动机器学习（Auto Tuner）在给定时间内搜索出最高效的实现



小结：内核优化的总结

- 内核优化与内核生成
  - 算子表达式
  - 算子表示与调度逻辑的分离
  - 自动调度搜索与代码生成

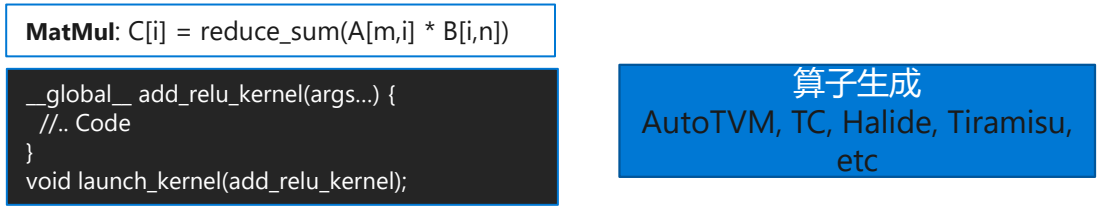
大纲

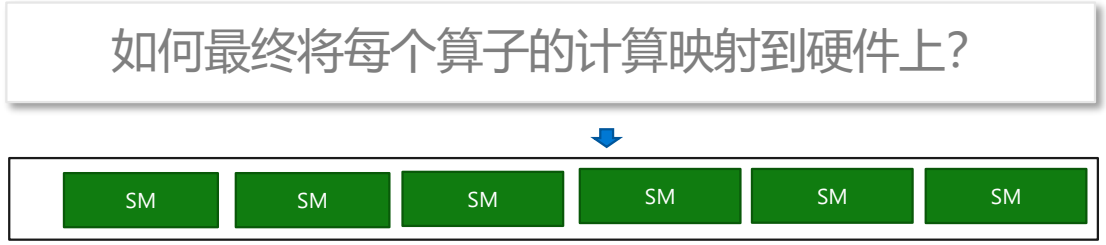
- 1. 计算图优化
- 2. 内存优化
- 3. 内核优化
- **4. 调度优化**

调度优化


ONNX
TensorFlow
PYTORCH
计算图IR（DAG）
x
*
+
y
w
b
计算图优化
内存优化
算子表达式IR
内核优化与生成
计算调度优化
CPU
GPU
XPU

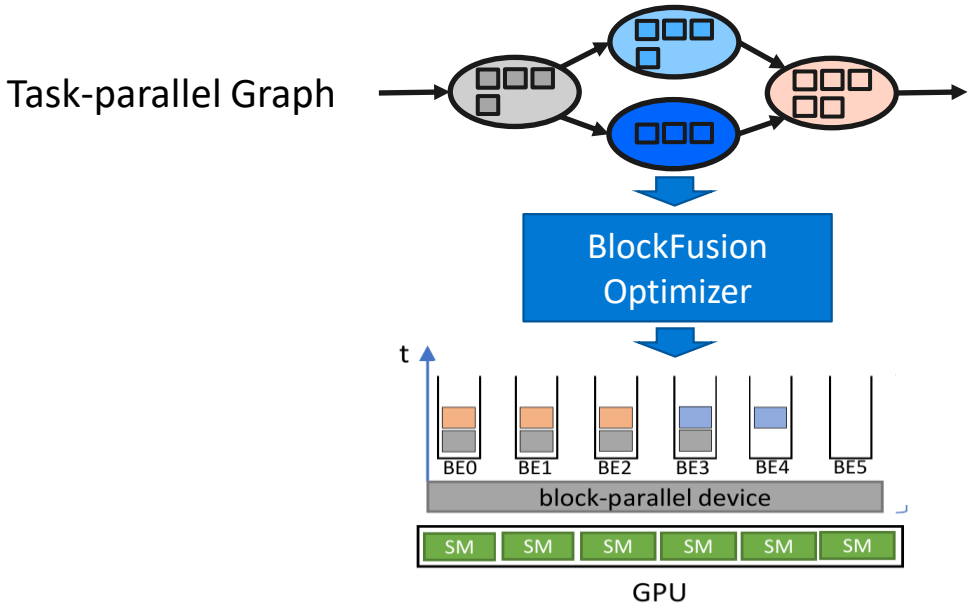

add
C[i] = A[i] + B[i]
relu
D[i] = max(C[i], 0)
add_relu
D[i] = max(A[i] + B[i], 0)
计算图优化
XLA, TVM, nGraph

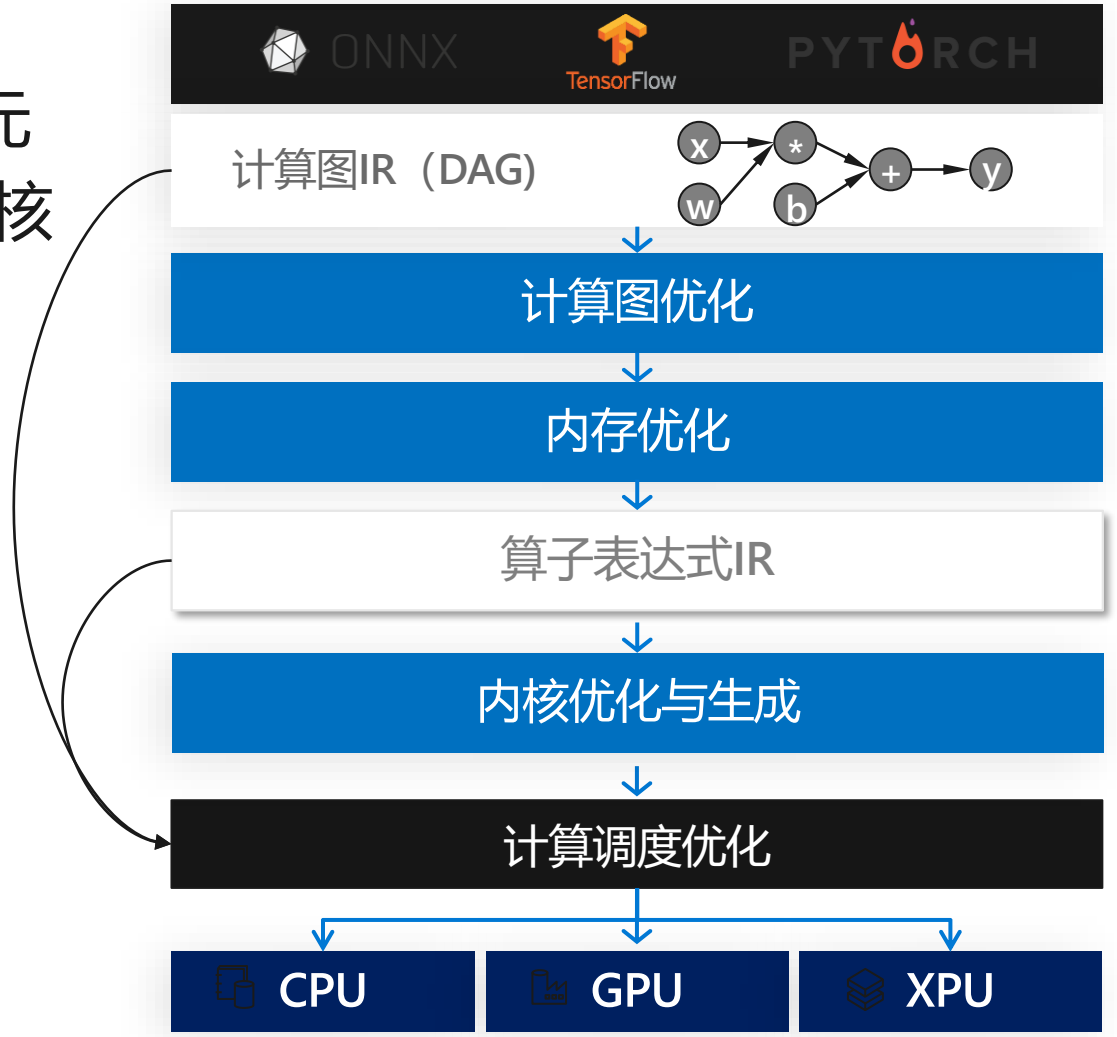

MatMul: C[i] = reduce_sum(A[m,i] * B[i,n])
__global__ add_relu_kernel(args...) {
//.. Code
}
void launch_kernel(add_relu_kernel);
算子生成
AutoTVM, TC, Halide, Tiramisu, etc


如何最终将每个算子的计算映射到硬件上?
SM
SM
SM
SM
SM
SM

NNFusion:全局计算调度优化

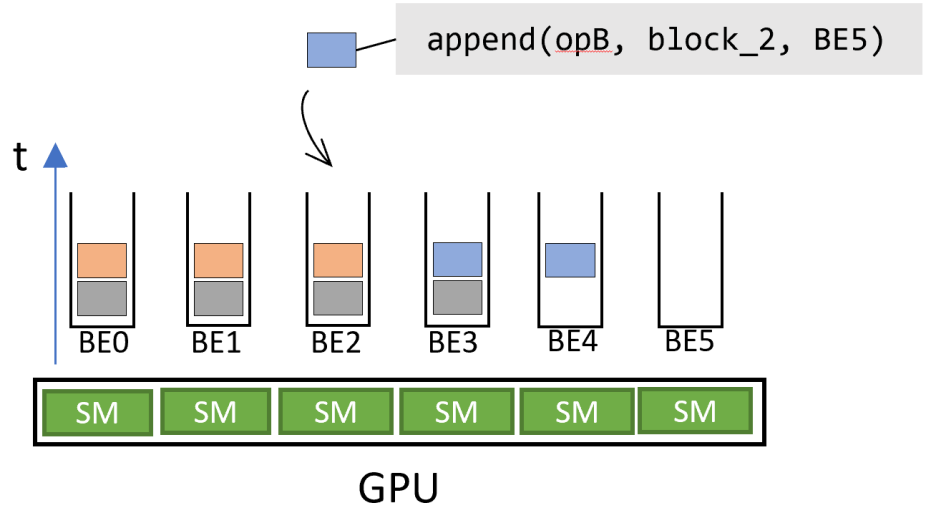
- 目标：通过将多个算子进行协同调度以及精确映射每一个算子到硬件计算单元来充分利用硬件并行度
- 中间表示：数据流图+细粒度算子并行单元
- 结果：将每个子图编译成一个硬件计算内核
  - 充分减少上层调度的开销
  - 高效利用硬件并行度

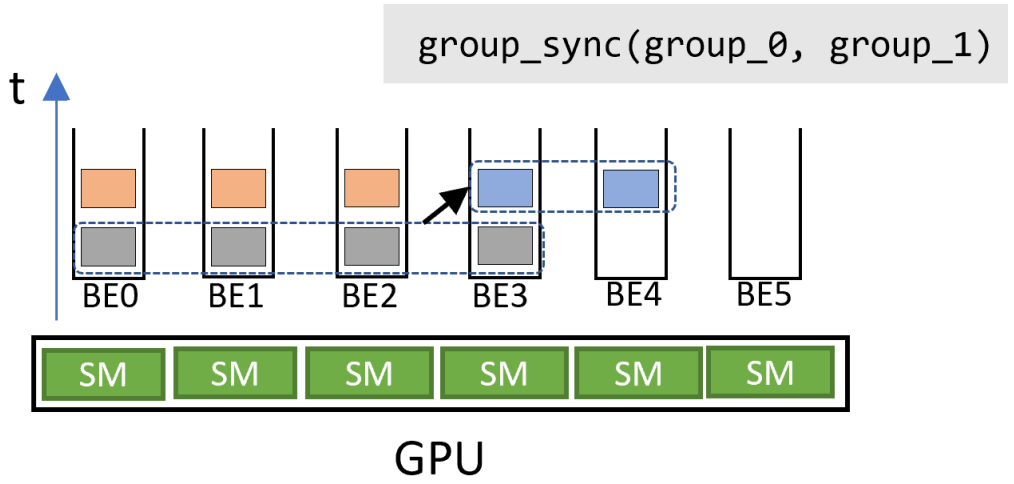




通过引出新的调度原语来支持任务级调度

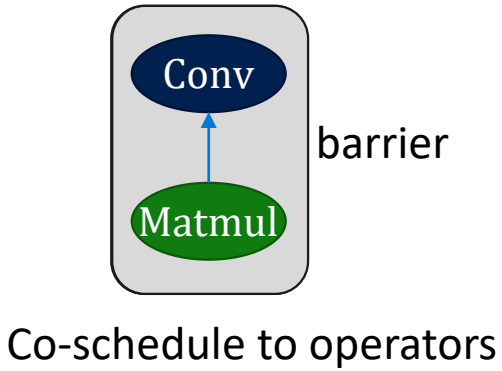
- APEEND: 将一个算子中的一个任务调度到硬件的一个计算单元上
- GROUP_SYNC: 维护任务间的依赖关系

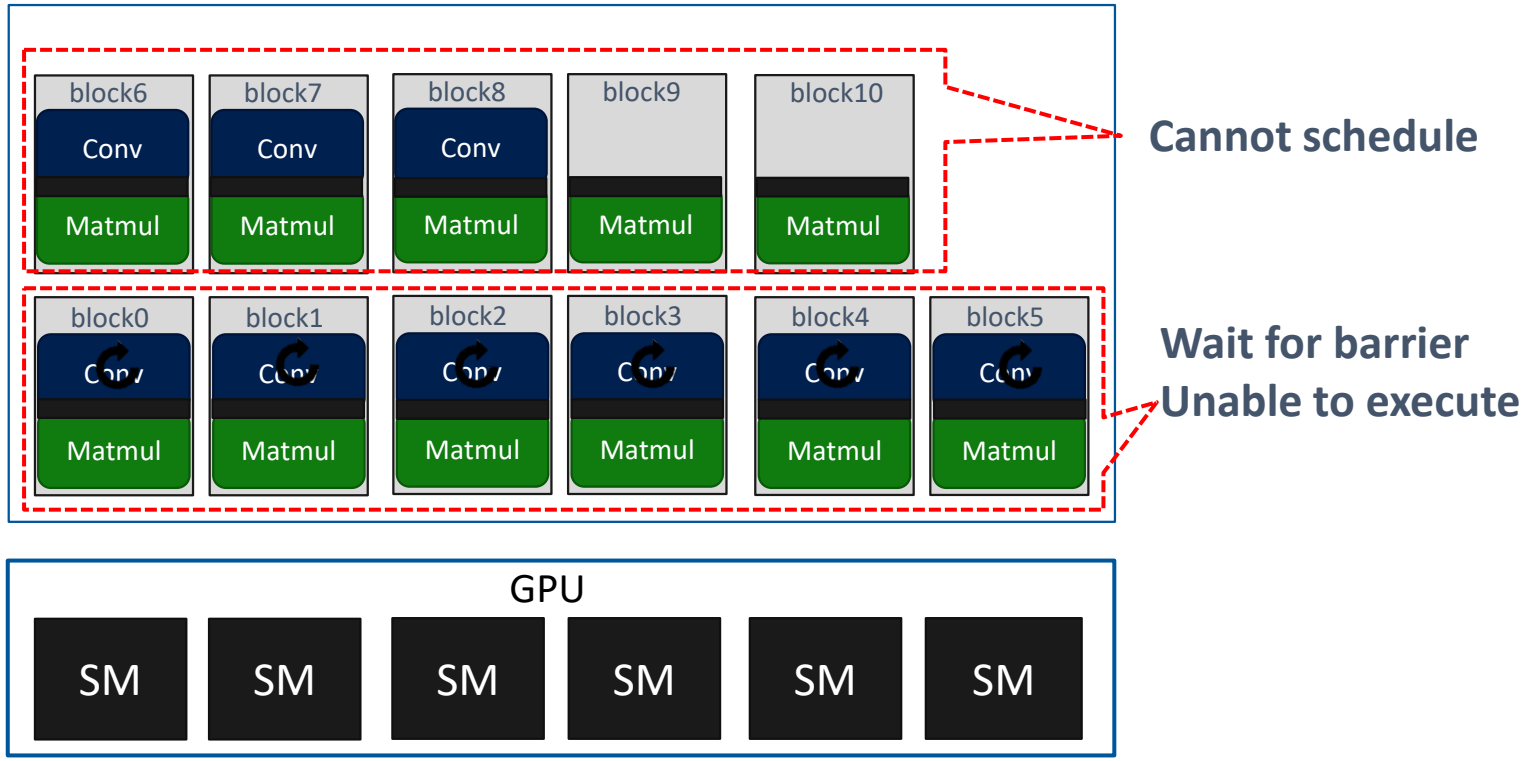




挑战：调度算子的任务到GPU上的挑战

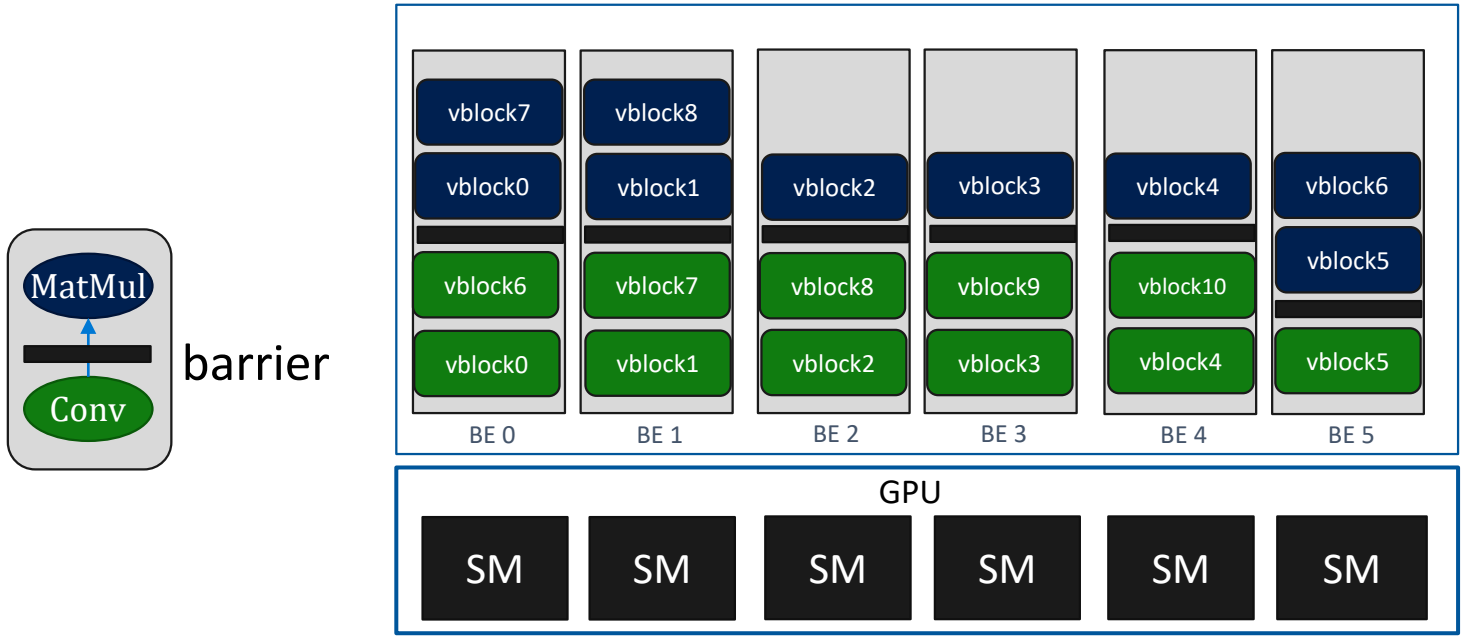
- 简单的任务级调度可能引起正确性问题()
  - 错误依赖
  - 死锁





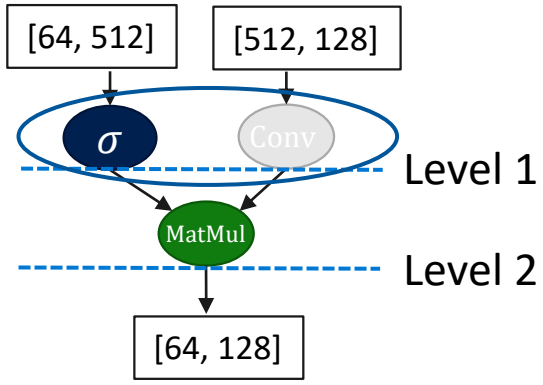
依赖关系的映射

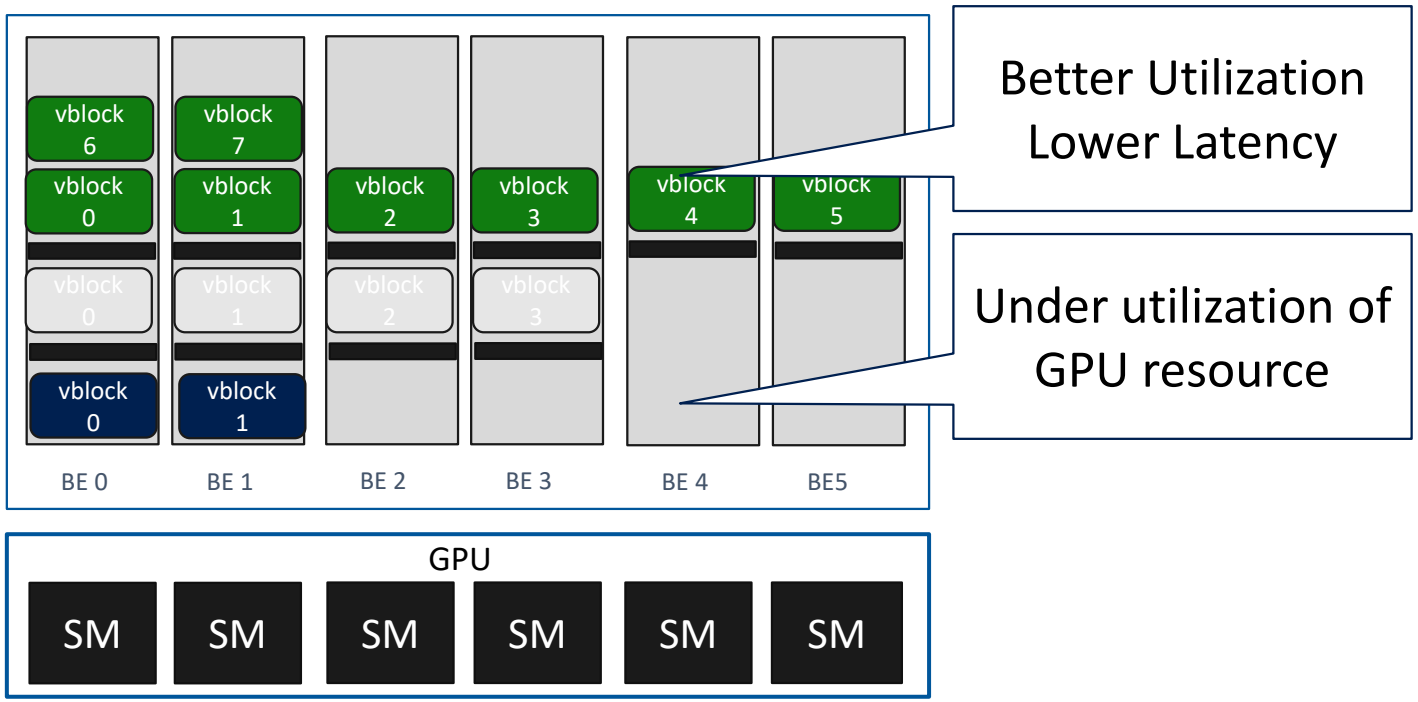
- 通过将硬件计算单元抽象到软件可控计算单元，并引入细粒度任务级同步支持来保证计算正确性



并行度的映射

- 任务级调度可以支持任意算子之间的并行调度，从而最大化硬件利用率





本次课程总结

- 深度神经网络编译器的概念与架构
  - 中间表达、前端、后端、优化过程
- 计算图优化
  - 算术表达式化简、公共子表达式消除、常数传播、矩阵合并、算子融合、子图替换/随机子图替换
- 内存优化
  - 基于拓扑序的最小内存分配、根据整数线性规划求解最优内存放置、张量换入换出与张量重计算
- 内核优化
  - 算子表达式、算子表示与调度逻辑的分离、自动调度搜索与代码生成
- 调度优化

课后作业

- 推荐补充阅读材料
- XLA, nGraph, TensorFlow, PyTorch
- TVM: An automated end-to-end optimizing compiler for deep learning
- Taso: optimizing deep learning computation with automatic generation of graph substitutions.
- Learning to optimize tensor programs
- Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions
- Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system
- Tiramisu: A polyhedral compiler for expressing fast and portable code

Lab 1 (for week 1, 2)

- Purpose
  - A simple throughout end-to-end AI example, from a system perspective
  - Understand the systems from debugger info and system logs
- Get ready
  - https://github.com/microsoft/ai-edu/ai-system/labs/1