中国科学技术大学
University of Science and Technology of China

深度学习 编译器

汇报人：陈曦

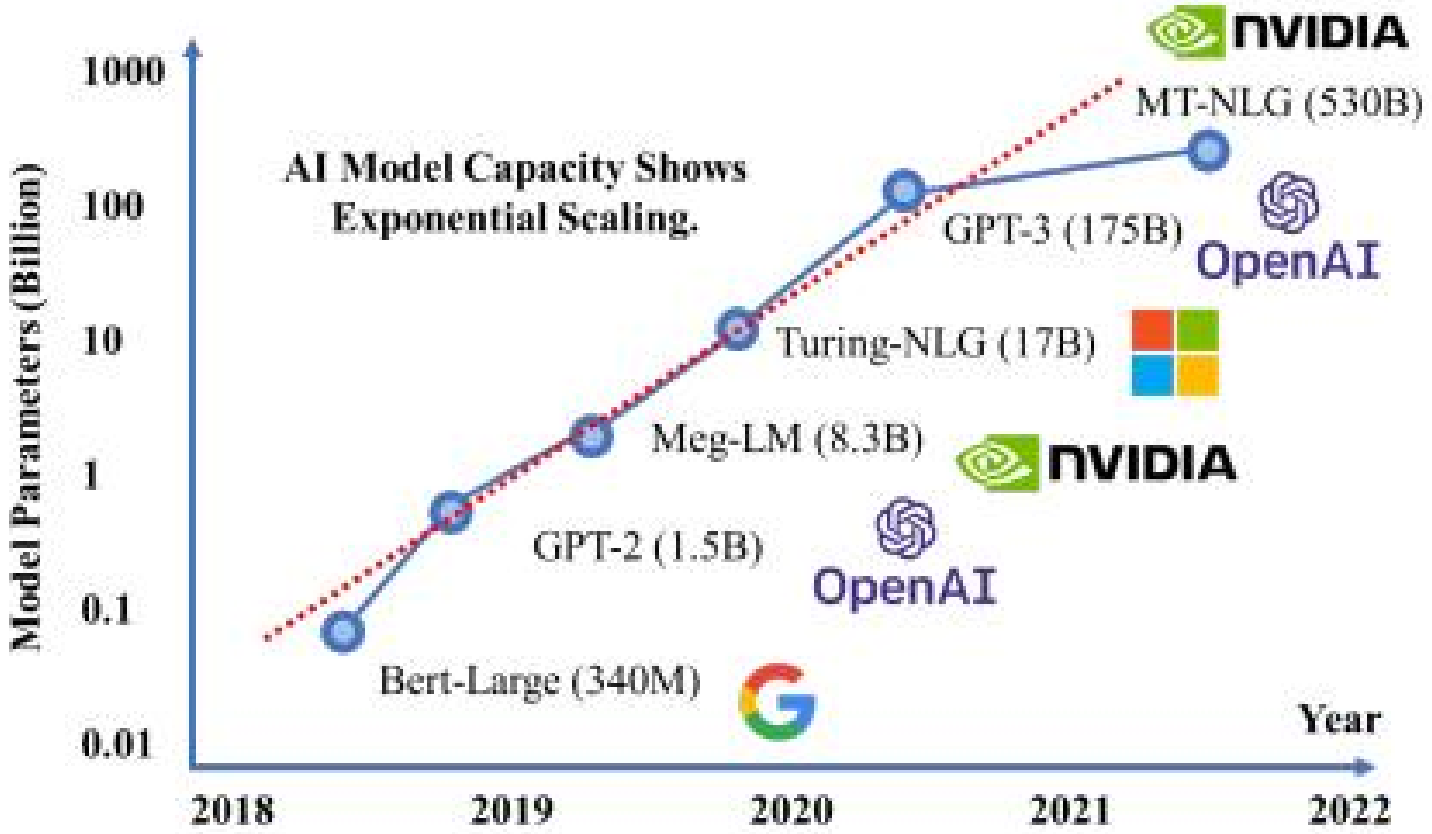
目录:

模型规模的爆炸：

•模型规模越大，“计算图 + 内存 + 调度”越复杂，性能瓶颈越来自体系结构，而不是单个算子。

•传统框架只能优化算子，而大规模模型需要：
- •跨算子融合（graph fusion）
- •内存规划（memory planning）
- •自动调度与 kernel 生成
- •跨硬件优化

•深度学习编译器应运而生

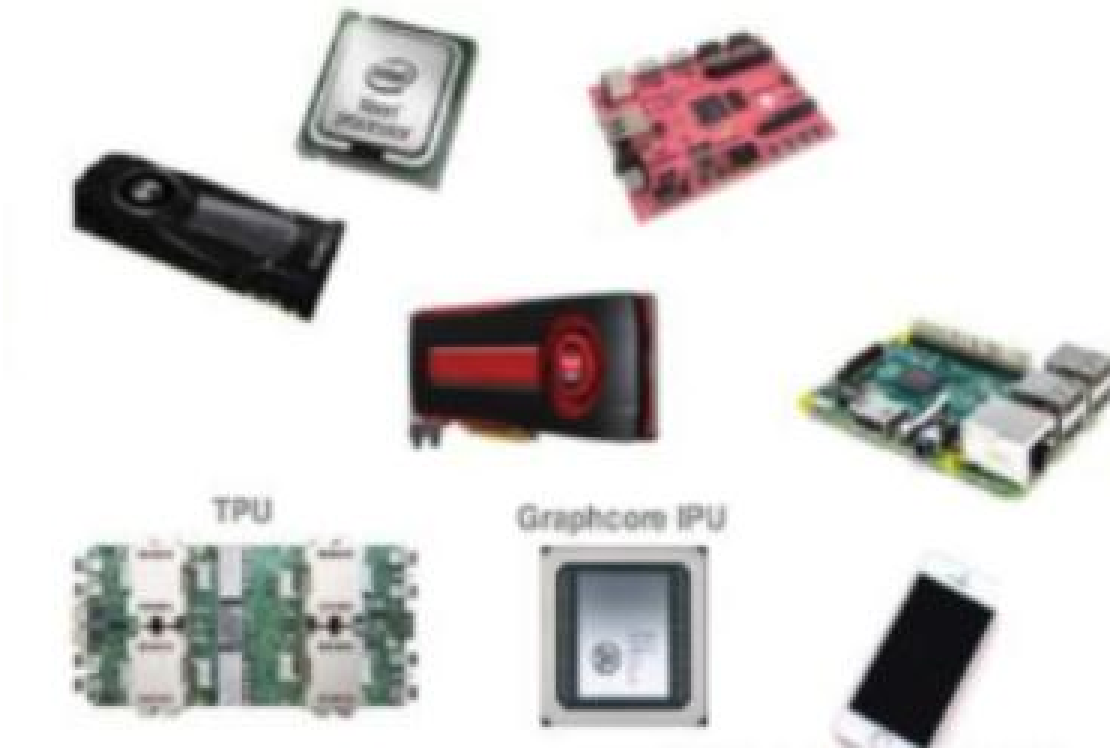


硬件平台异构化：

•如今硬件平台越来越丰富。
CPU, GPU, TPU, FPGA, DSP等等都可以用来进行模型训练。然而，**不同硬件的kernel写法完全不同**。

•人工维护**几十种设备+上千种算子**的优化---->不可能实现

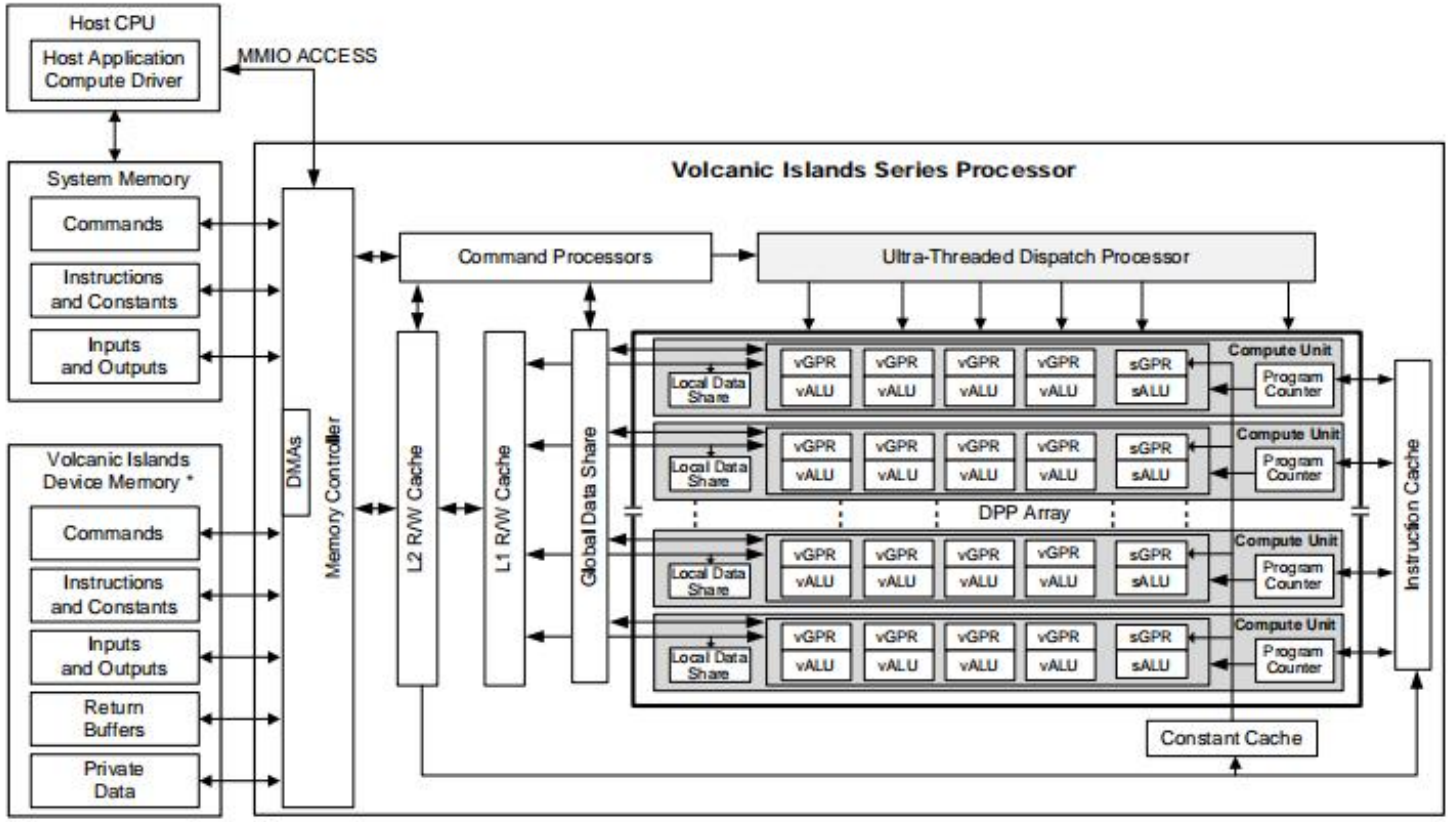
•深度学习编译器：**提供了跨硬件的抽象层。**



深度学习硬件

GPU：

- GCN并行微架构
  - 并行处理阵列
  - 命令处理器
  - 内存控制器
- 计算单元（CU）
- SALU：
  - 共享SGPR寄存器堆。
  - 执行控制流（循环控制、条断）。
  - Wave内线程锁步执行（SIMT
- VALU：
  - 私有VGPR寄存器堆。
  - 由4×SIMD16单元组成，执行运算。
  - Wave内线程并行执行。

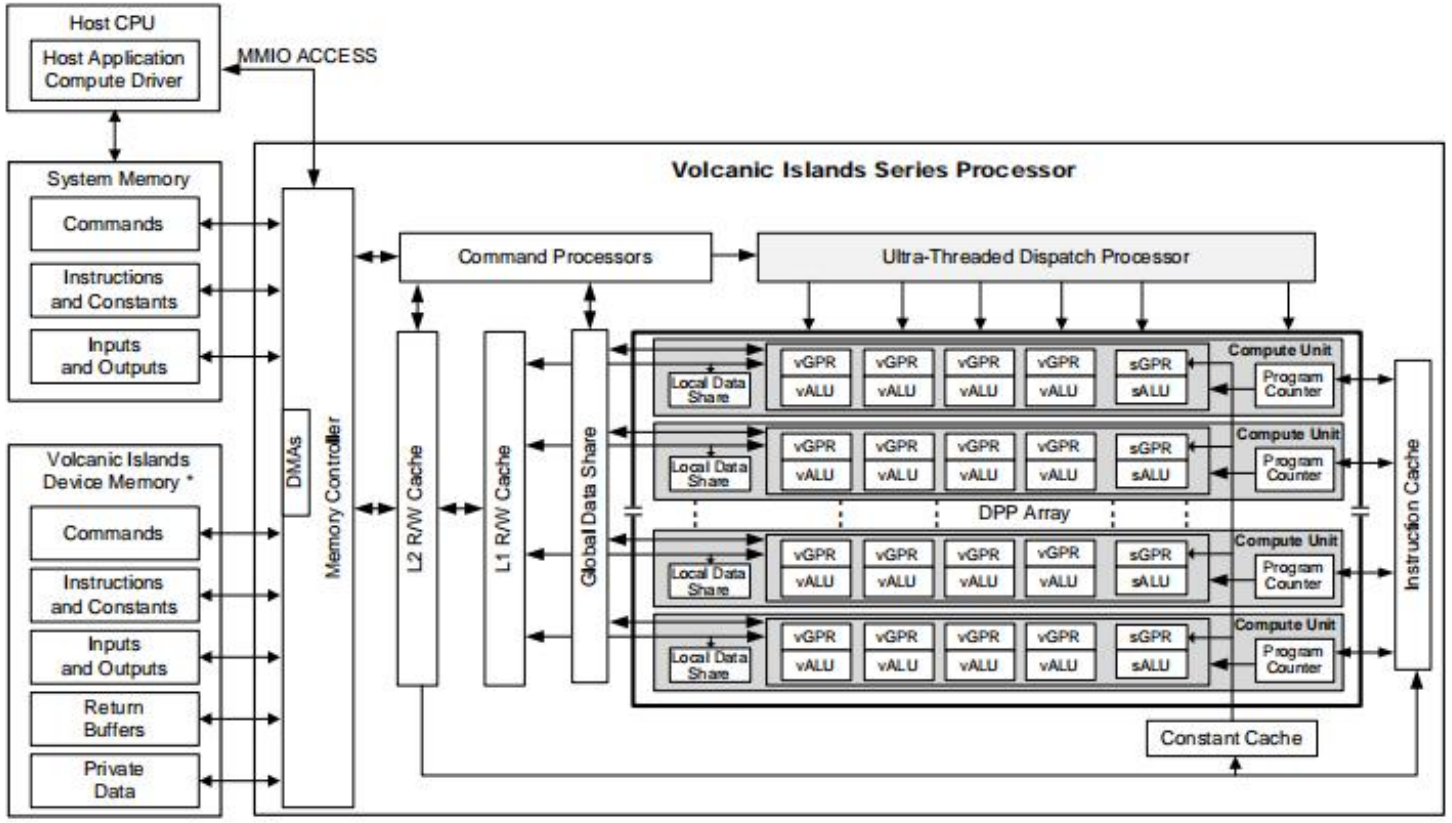


*Discrete GPU – Physical Device Memory; APU – Region of system for GPU direct access

深度学习硬件

AMD GPU线程模型

- work-item
  - 线程：最小执行单元
  - 硬件映射：一个ALU
  - 数量：一个CU中包含64个AL
- wavefront
  - 最小调度单元
  - 硬件映射：一个CU
  - 数量：一个wavefront中包含
    work-item
  - 特性：内部所有线程“锁步”
    同一命令
- work-group
  - 任务单元
  - 数量关系：一个work-group
    个wavefront
  - 特性：共享LDS内存，由硬件
    到单个CU执行
- kernel
  - 全局任务
  - 数量关系：包含多个work-group



深度学习硬件

AMD GPU内存层次结构

- 寄存器
  - SGPR：Wave内线程共享，供SALU使用
  - VGPR：线程私有，用于VALU计算
- 局部共享
  - LDS（64KB）：位于CU内部，是工作组内线程共享的SRAM(用于线程同步与数据交换)
  - L1 Cache（16KB）：向量/标量指令的共享缓存，降低访存延迟
- 全局共享
  - GDS（64KB）：跨CU的线程通信
  - L2 Cache：CU共享的统一缓存
  - 显存（DRAM）：主存访问，由内存控制器统一调度

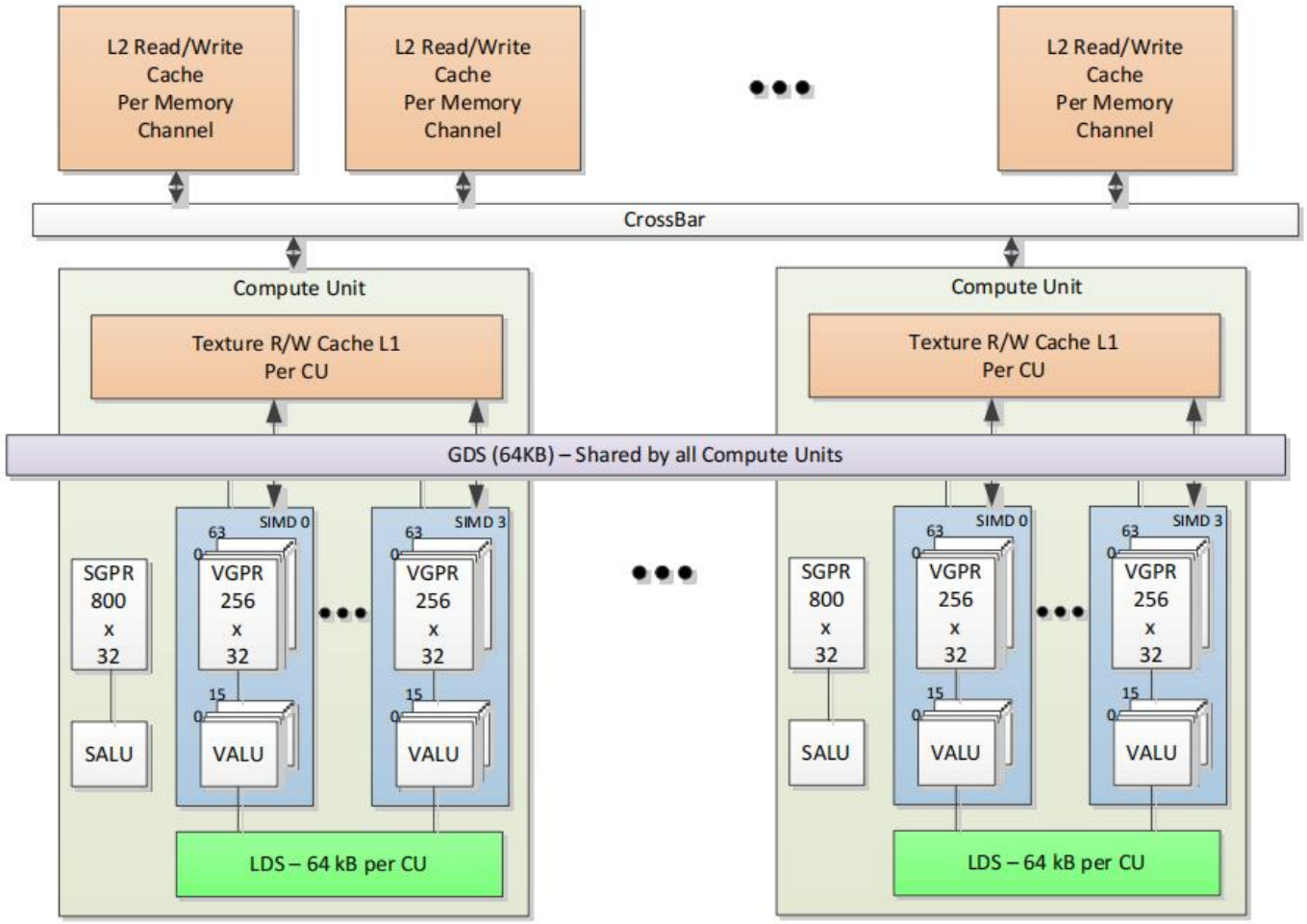
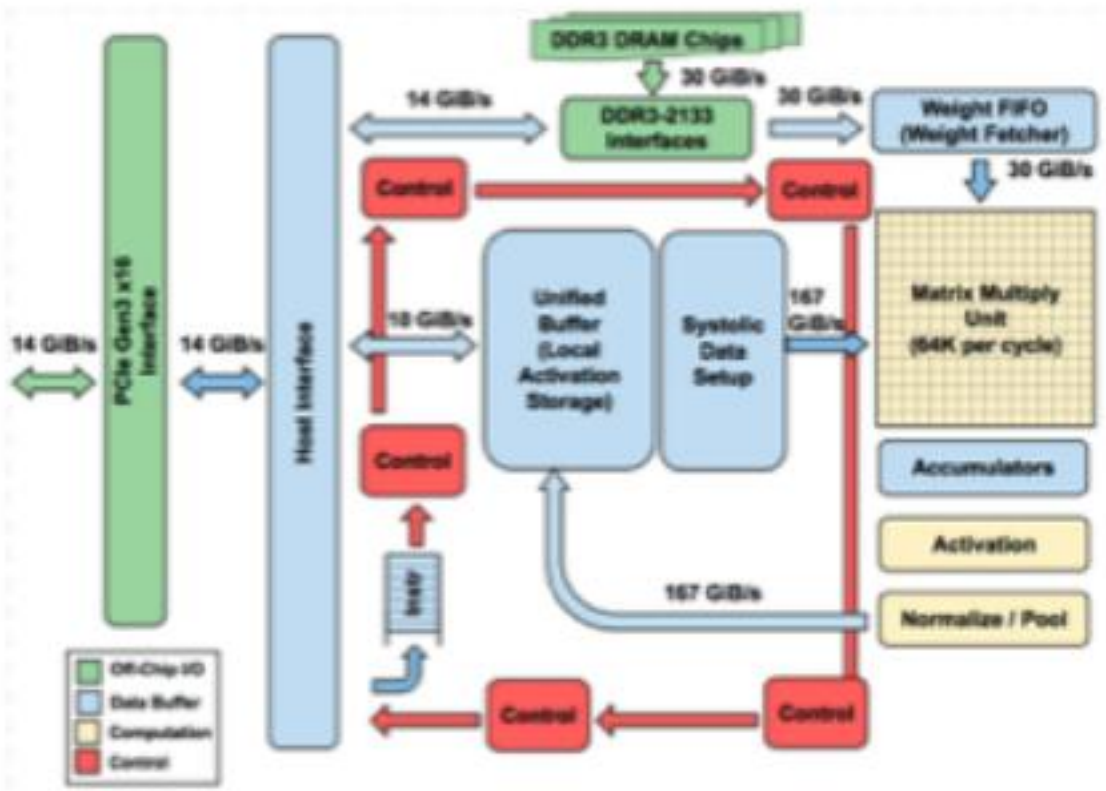
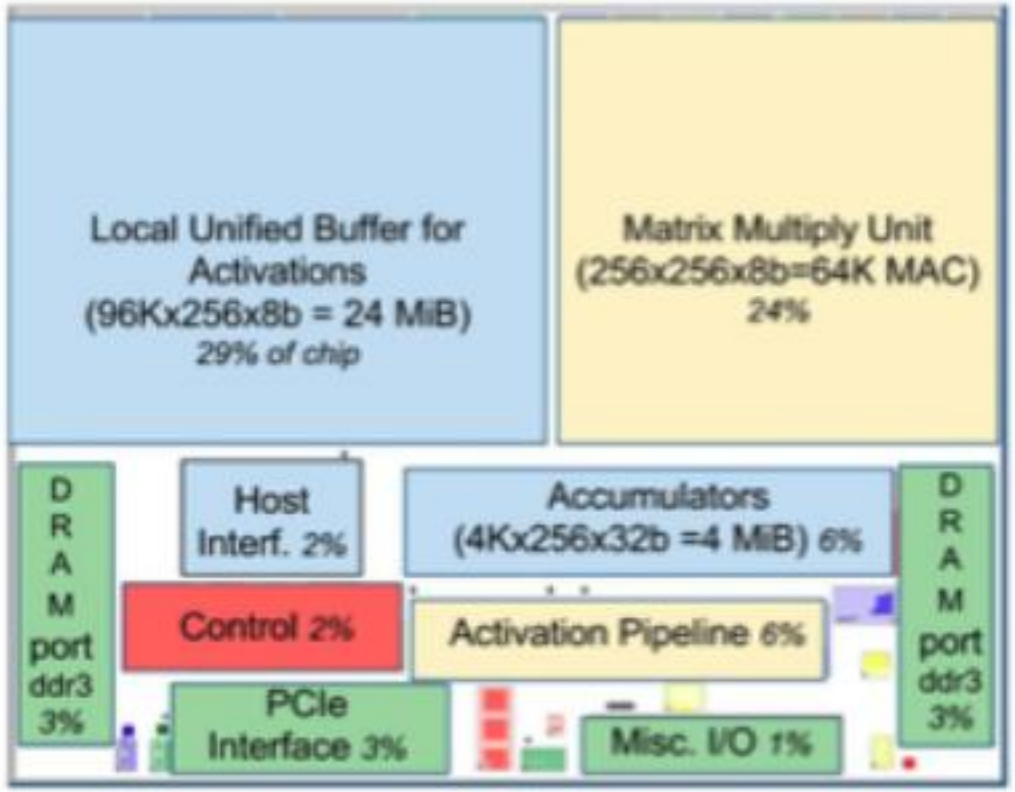


Figure 2.1 Shared Memory Hierarchy on the AMD GCN Generation 3 Series of Stream Processors

深度学习硬件

TPU1





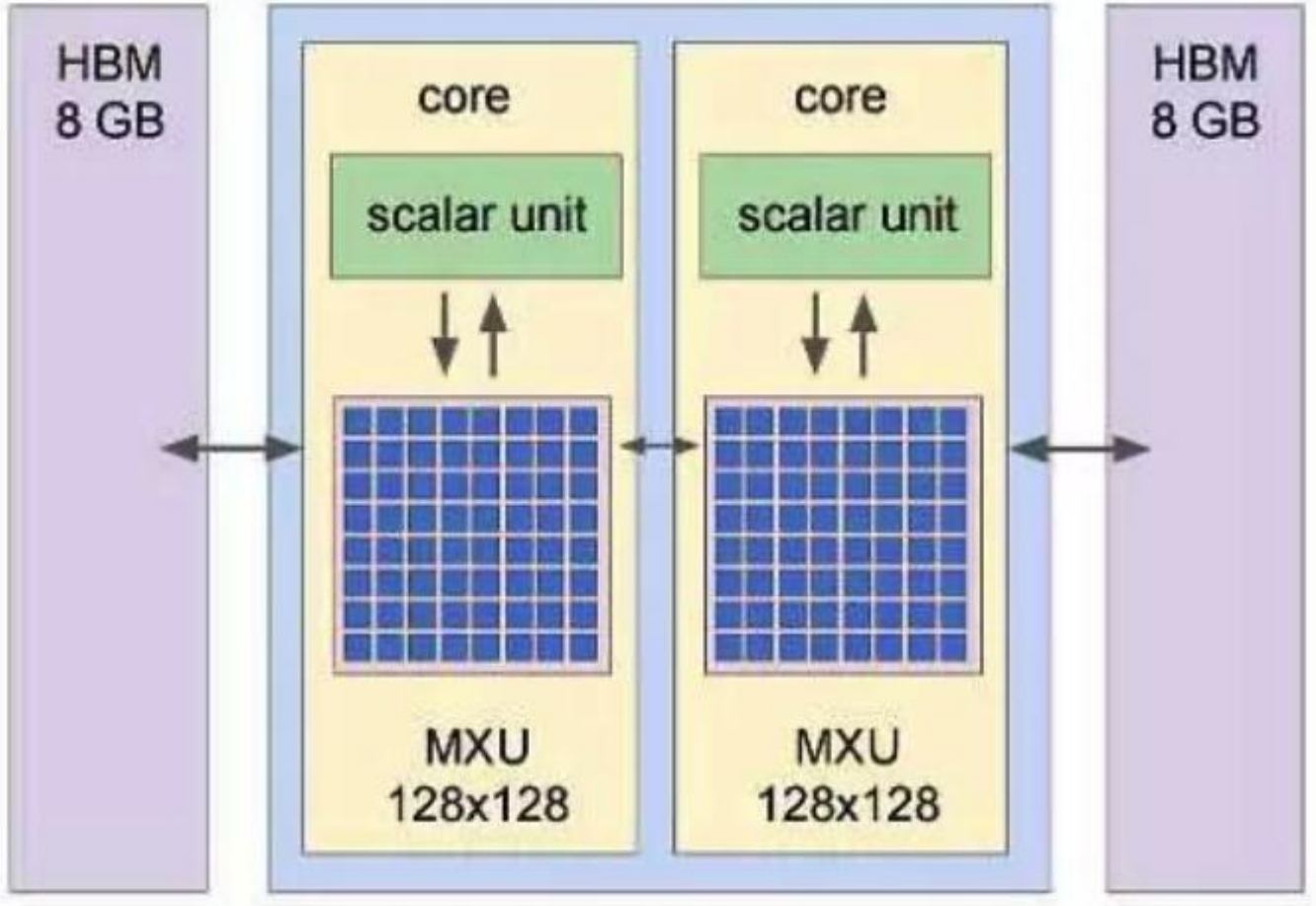
如图展示的是TPU1的架构，可支持16bit/8bit运算，主要完成矩阵-矩阵乘、矩阵-向量乘、向量-向量乘等功能，来支持MLP（多层感知机）、LSTM和CNN等深度学习算法。

深度学习硬件

TPU2

- **计算粒度**：将脉动矩阵规模从256X256缩减至128X128，如图所示，这样每次控制和调度单位为原来的四分之一，减少数据导入导出延迟的同时，更有利于提升单片的计算效率；

- **算力伸缩**：将TPU1的PCIE通信模式转变为板级通信模式，实现多板互联和高密度算力集群，采用类似芯片级分布式的方式实现算力扩展，降低任务分配、同步的难度和通信成本。

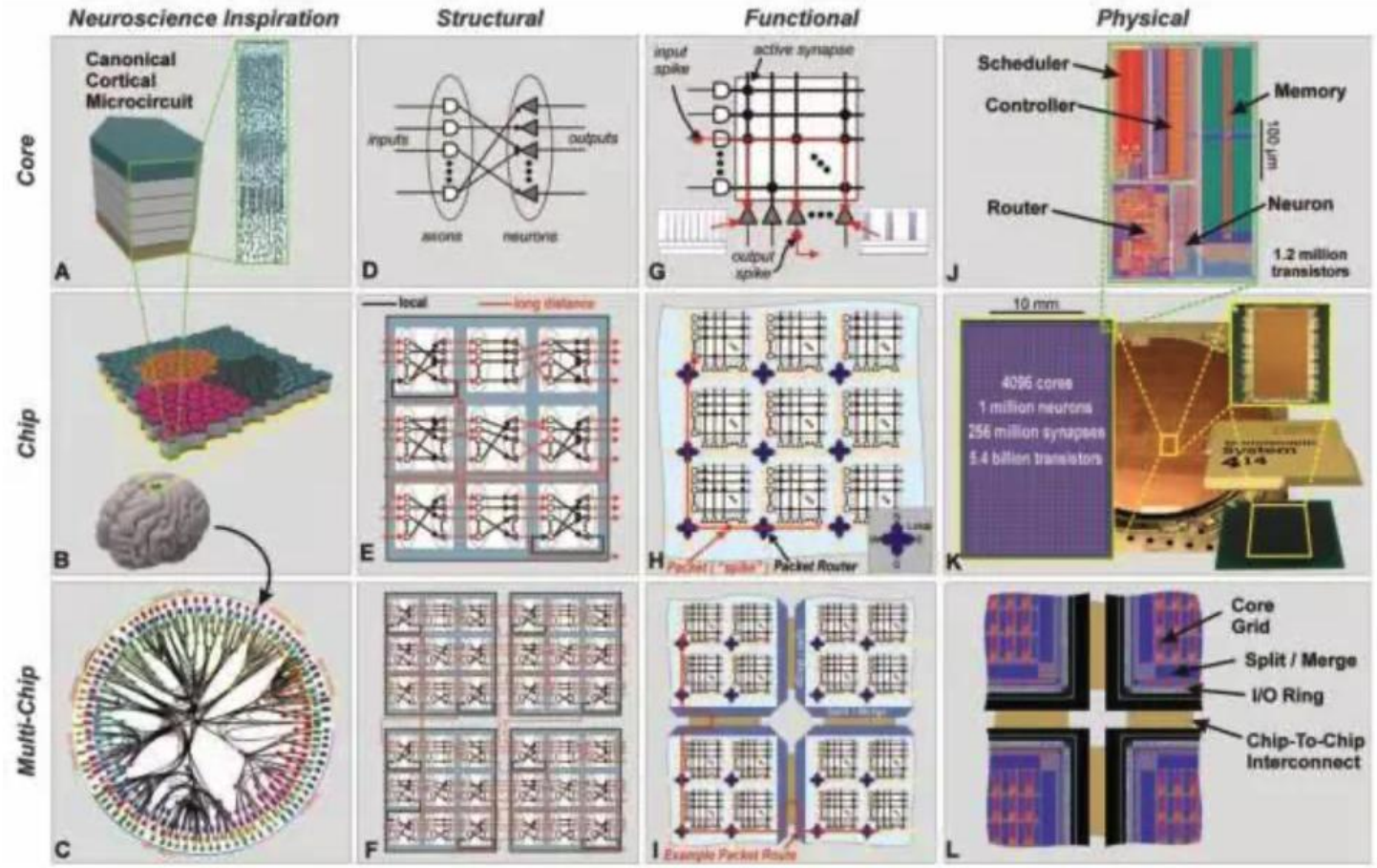


深度学习硬件

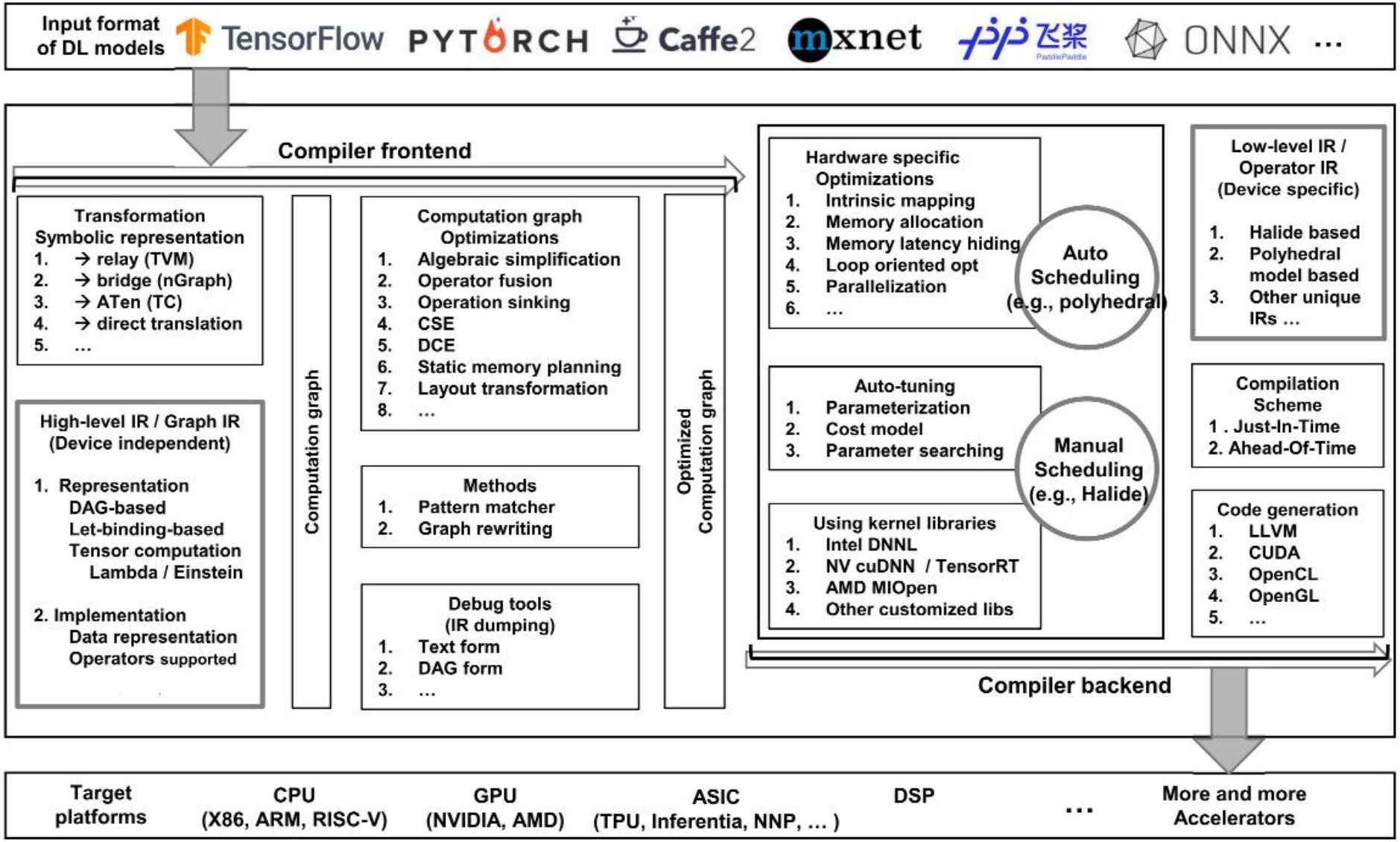
TrueNorth

- IBM也有推出类脑ASIC TrueNorth，早在2006年即开始立项，将时钟频率降低至1KHz，以注重低功耗应用。其芯片内节点的互联方式可编程实现，在学术界和国防项目的嵌入式前端的视觉识别上有相关应用，且可以海量芯片互联以实现更大规模的任务；

- 有意思的特点是仿照大脑结构，神经元可以**同时存储和处理数据**



深度学习编译器体系结构



深度学习编译器组件

graph IR：

•**基于有向无环图的中间表示。**在深度学习编译器[中，有向无环图的节点表示深度学习的原子算子（如卷积、池化等），边表示张量。并且该图是无环的，没有循环，这与通用编译器的数据依赖图（DDG）不同。

- •优势：
  - •借助有向无环图计算图，深度学习编译器可以**分析各种算子之间的关系和依赖关系**，并利用它们来**指导优化**
  - •**对于数据依赖图已经有许多优化方法，**通过将深度学习的领域知识与这些算法相结合，可以对有向无环图计算图应用进一步的优化

•**基于let绑定的中间表示。**let绑定是一种通过为某些函数提供具有受限作用域的let表达式来解决语义模糊问题的方法

graph IR：

•基于DAG的表示和基于let绑定的表示示例。

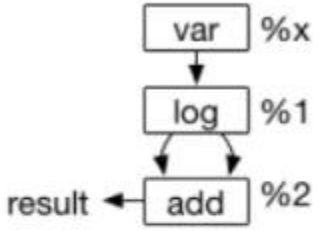
Python Code

```
x = relay.var("x")
v1 = relay.log(x)
v2 = relay.add(v1, v1)
f = relay.Function([x], v2)
```

Text Form

```
fn (%x) {
  %1 = log(%x)
  %2 = add(%1, %1)
  %2
}
```

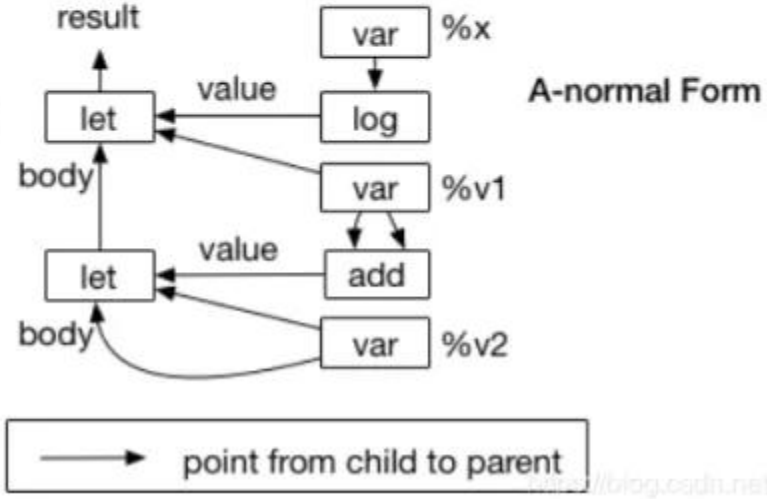
AST Structure



Dataflow

```
x = relay.var("x")
sb = relay.ScopeBuilder()
v1 = sb.let("v1", relay.log(x))
v2 = sb.let("v2", relay.add(v1, v1))
sb.ret(v2)
f = relay.Function([x], sb.get())
```

```
fn (%x) {
  let %v1 = log(%x)
  let %v2 = add(%v1, %v1)
  %v2
}
```



A-normal Form

graph IR：

- 对张量计算的表示：

  - **基于函数的**：基于函数的表示方式仅仅提供封装好的算子，Glow、nGraph 和 XLA 采用了这种方式。

  - **lambda 表达式**：lambda 表达式是一种索引公式表达式，通过变量绑定和替换来描述计算过程。**TVM 使用基于 lambda 表达式**的张量表达式来表示张量计算。在 TVM 中，张量表达式中的计算算子是由输出张量的形状以及计算规则的 lambda 表达式来定义的。

  - **爱因斯坦记号(Einstein notation)**：爱因斯坦记号，也被称为求和约定，是一种用于表示求和的记号。它在编程方面的简洁性优于 lambda 表达式。

graph IR的实现：

- 数据表示：
  - **占位符**：占位符简单来说是一个带有明确形状信息（例如，每个维度的大小）的变量，并且它会在计算的后期阶段填充值。
  - **未知（动态）形状表示**：在声明占位符时，通常支持未知的维度大小。
    - TVM 使用“Any”来表示未知维度 如，张量⟨ (Any, 3), fp32⟩
    - XLA 使用“None”来达到相同的目的 如，tf.placeholder(“float”, [None, 3]))
    - nGraph 使用其“PartialShape”类。
  - **数据布局**：数据布局描述了张量在内存中的组织方式，通常是从逻辑索引到内存索引的一种映射。
  - **边界推断**：在深度学习编译器中编译深度学习模型时，边界推断用于确定迭代器的边界。

graph IR的实现：

- 支持的算子：
  - **广播**：广播算子可以复制数据，并生成形状兼容的新数据
  - **控制流**：在表示复杂且灵活的模型时，控制流是必需的。诸如循环神经网络（RNN）和强化学习（RL）等模型依赖于递归关系和数据相关的条件执行
  - **求导**：算子 Op 的求导算子将 Op 的输出梯度和输入数据作为其输入，然后计算 Op 的梯度。
  - **定制算子**：它允许程序员为特定目的定义自己的算子。对定制算子提供支持可提高深度学习编译器的可扩展性。
    - TVM 的用户只需描述计算过程和调度，并声明输入 / 输出张量的形状。此外，定制算子通过钩子函数集成 Python 函数，这进一步减轻了程序员的负担。

graph IR与传统IR的对比：

特性	传统 IR（LLVM）	图 IR（Relay/XLA/ONNX）
表达目标	指令执行顺序	算子之间的数据依赖
程序结构	CFG	DAG
适用领域	普通程序	AI / Tensor 计算
Tensor 支持	无	支持
张量的Shape 推断	难	支持
Kernel 融合	难	简单
高层语义	丢失	保留（conv, matmul...）
优化难度	高	低
自动并行	难	易
移植性	低	高

Low-level IR：

•**基于 Halide 的中间表示：**Halide 的基本理念是**将计算和调度分离**。采用 Halide 的编译器不是直接给出特定的方案，而是**尝试各种可能的调度并选择最佳的方案**。

•**基于多面体的中间表示** ：多面体模型是深度学习编译器中采用的一项重要技术。它使用线性规划、仿射变换和其他数学方法来优化具有静态控制流的边界和分支的基于循环的代码。

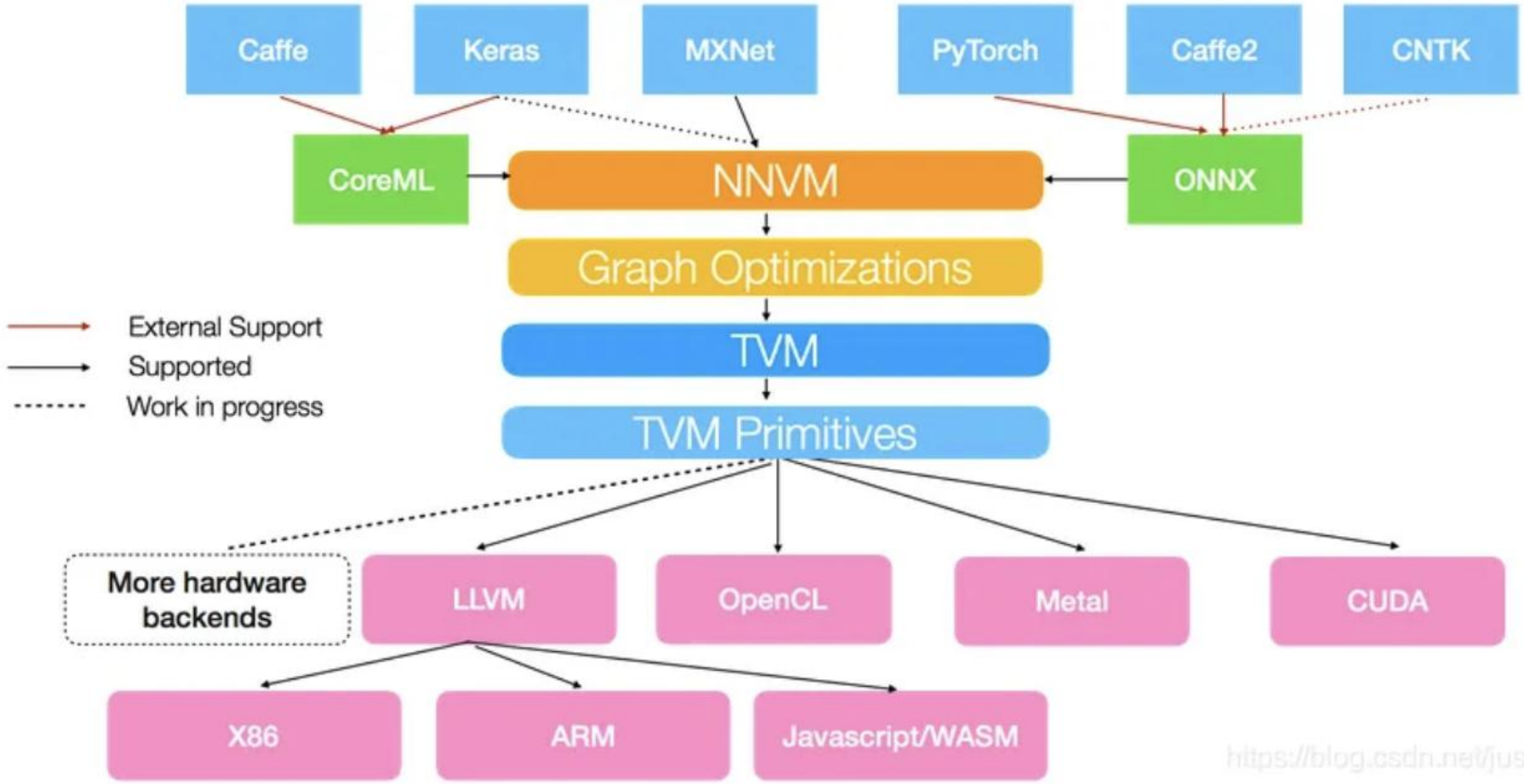
•Glow 中的低级中间表示是一种**基于指令的表达式**，对通过地址引用的张量进行操作

•MLIR 重用了 LLVM 中的许多思想和接口，并且位于模型表示和代码生成之间。**MLIR 具有灵活的类型系统，并允许多个抽象级别**，它引入了方言来表示这些多个抽象级别。每个方言由一组定义的不可变操作组成。

•XLA 的 HLO 中间表示可以被视为高级中间表示和低级中间表示，因为 **HLO 足够细粒度，可以表示特定于硬件的信息**。

深度学习编译整体流程

TVM架构为例：



节点级优化：

- **节点消除**：消除无操作节点。无操作节点通常是指，做了一个运算，但是**运算对结果没有任何影响。**

如：sum节点只有一个输入

如：Pad 节点的 padding = 0（零填充）

节点级优化：

- Zero-dimension tensor elimination：一类优化是深度学习编译器独有的，因为DL 框架会产生 shape = 0 的张量。对于这种张量的操作可以消除。
- 如：C={0,12,3} 虽然是3D张量 但是内部元素的个数是0 本质上是空张量，对于这样的C 有如下优化。

如：Pad 节点的 padding = 0（零填充）

块级优化：

- **代数简化**：一代数简化是一类利用数学性质（交换律、结合律、分配律等）来优化计算图的编译优化。
  - **代数识别**：识别计算图中可利用代数性质进行重写的模式。
    - 交换律：A + B = B + A
    - 结合律：(A + B) + C = A + (B + C)
    - 分配律：A×(B + C) = A×B + A×C
    - 矩阵转置性质：$(A^T \times B^T) = (B \times A)^T$ ← 深度学习中最常用！

优化前：

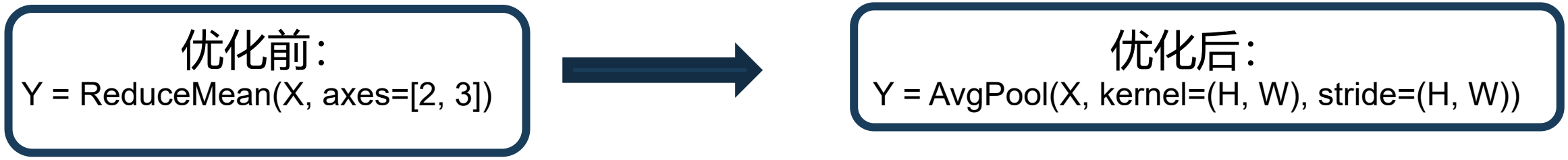
```
AT = Transpose(A)
BT = Transpose(B)
Y  = GEMM(AT, BT)
```

优化后：

```
Y = Transpose(GEMM(B, A))
```

块级优化：

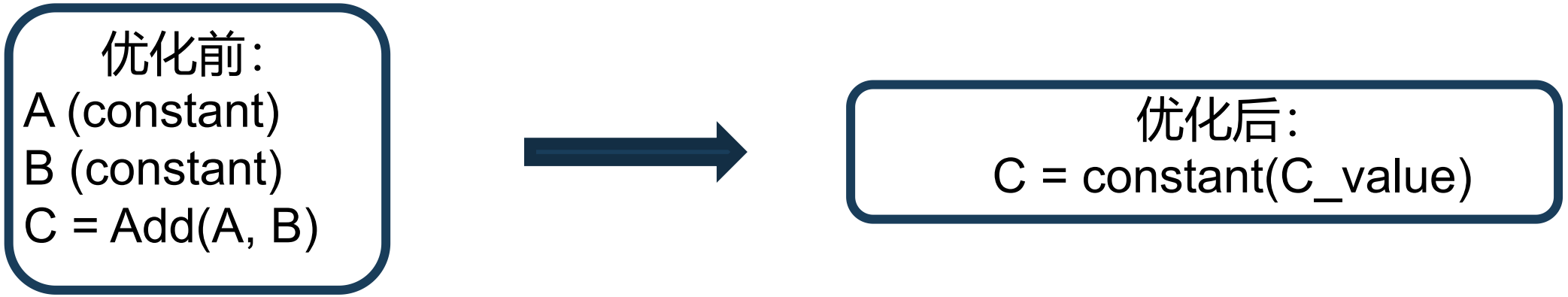
- **代数简化**：一代数简化是一类利用数学性质（交换律、结合律、分配律等）来优化计算图的编译优化。
  - **强度削弱**：用 成本更低 的算子替换 成本更高 的算子，只要数学等价。



- AvgPool 通常有更强的硬件加速（NPU/GPU/Tensor Core 友好）
- 芯片专门针对 Pooling 优化
- Glow、TensorRT 都会执行这个优化

块级优化：

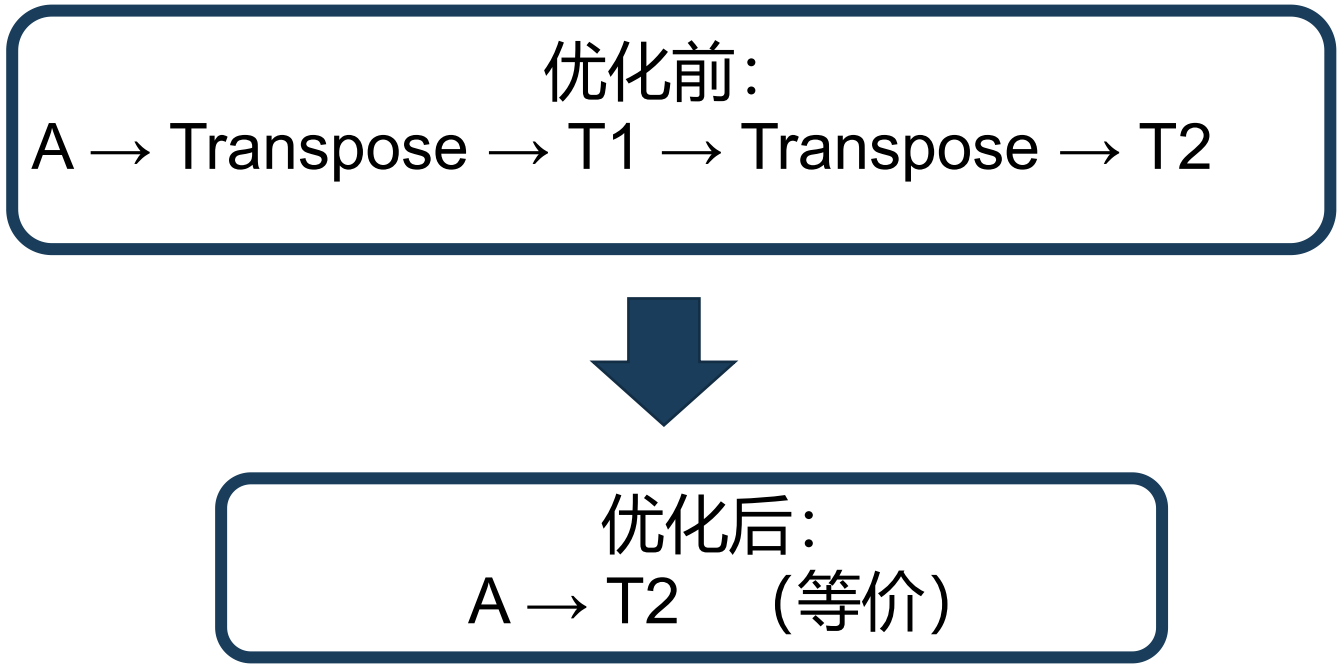
- **代数简化**：一代数简化是一类利用数学性质（交换律、结合律、分配律等）来优化计算图的编译优化。
  - **常量折叠**：如果表达式的所有输入都是常量 → 直接把计算提前在编译期算好。



- 运行时减少计算
- 去掉节点
- 图更小、执行更快

块级优化：

- **代数简化**：一代数简化是一类利用数学性质（交换律、结合律、分配律等）来优化计算图的编译优化。
  - **计算顺序优化**，在这种情况下，优化过程会根据特定特征查找并移除重塑 / 转置操作。



块级优化：

- **算子融合**：将多个相邻算子合并为一个 kernel（GPU/CPU 计算内核），减少内存访问、减少中间张量生成、减少 kernel 启动开销，从而显著提升性能。

  - **算子融合的好处**
    - 减少中间张量的分配
    - 降低内存读写压力（许多算子都属于 memory-bound）
    - 合并循环（loop fusion）减少计算开销
    - 减少 kernel launch 开销（GPU 启动一次 kernel 的成本很高）
    - 优化跨算子之间的数据依赖与共享

块级优化：

- 算子融合的类型

类型	解释	例子	是否易融合
内射型（injective）	单输入对应单输出，无依赖重排	add, relu, exp, transpose, broadcast	几乎总能融合
归约型（reduction）	输入→输出维度减少	sum, mean, argmax, softmax的reduce部分	可融合，但需满足调度条件
复杂输出可融合型（complex-out-fusable）	计算复杂但输出 shape 推导明确	conv2d, matmul, batch_norm, pooling	常作为融合“根节点”
不透明型（opaque）	内部实现不可见或过于复杂	外部函数调用、GPU 库的算子（cuDNN）	通常不可融合

块级优化：

- **算子下沉**：把某些不会改变语义的算子（如 transpose、reshape、broadcast）沿着计算图向下移动，让它更接近后续算子，以创造更多融合和代数简化的机会。

优化前：
x → transpose → relu → ...

优化后：
x → relu → transpose → ...

下沉后的好处：

- 与后续的 transpose 可能合并
- 激活函数常与卷积融合（conv + relu）
- transpose 下沉后，不阻断 conv + relu 的融合

块级优化：

- 常见可下沉算子：

可下沉算子	类型	原因
transpose	布局改变	可与其他 transpose 合并
reshape	shape-only	不改变数据顺序
broadcast	shape-only	常与 elementwise 合并
expand_dims / squeeze	shape-only	常消除
bitcast	shape-only	可合并

数据流级优化：

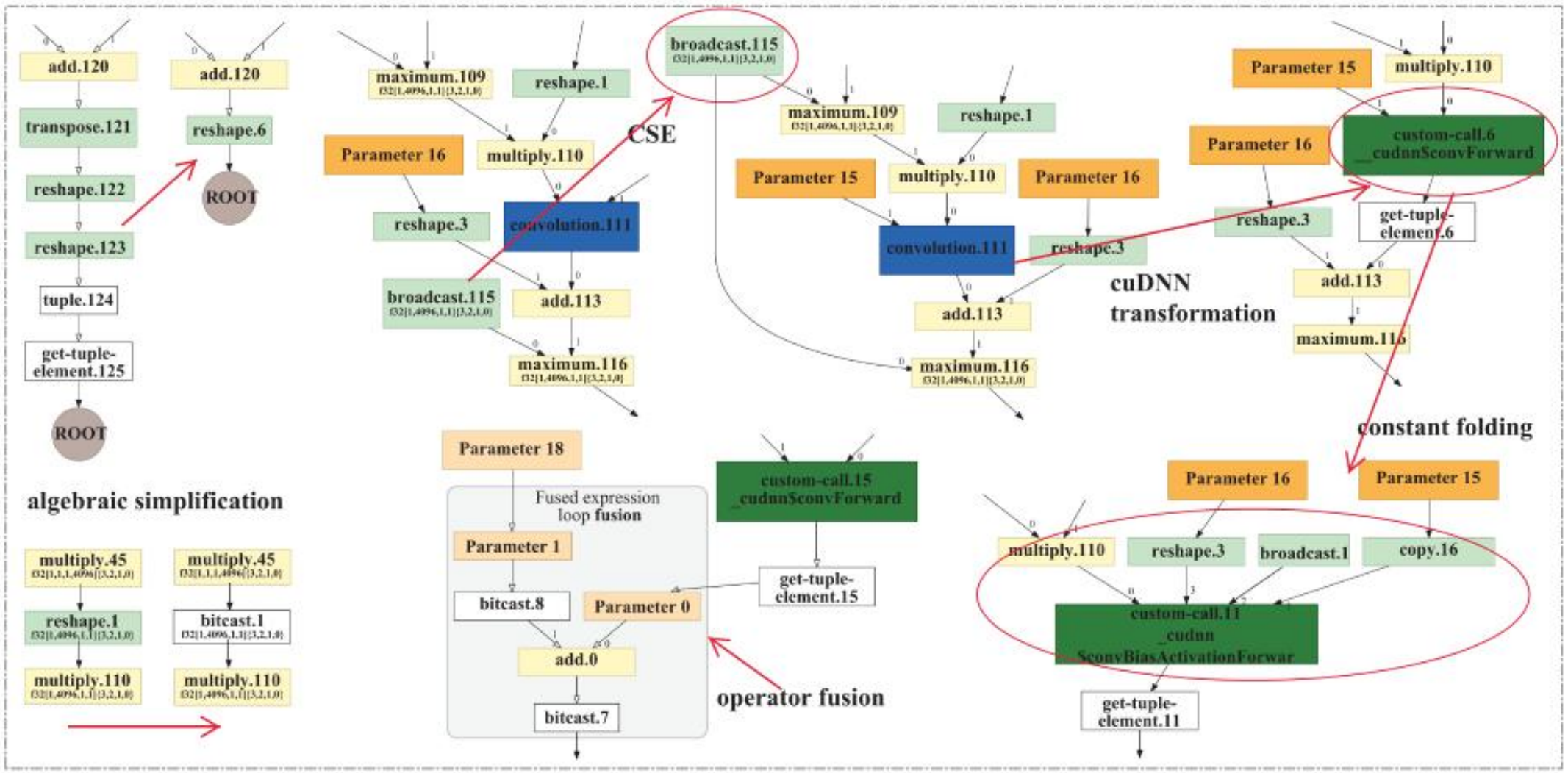
- **公共子表达式消除**：如果某个表达式 E 在图中被多次以完全相同的输入计算，而且期间这些输入没有被修改（不可变语义），那么只需计算一次，将后续出现的 E 用第一次的结果替代。

- **死代码消除（DCE ）与死存储消除（DSE）**：删除那些计算结果既不影响最终输出也无副作用的子图 / 指令。DSE：删除永远不会被读取的内存写操作。

- **静态内存规划**：在编译期分析所有中间张量的生存期（liveness），尽可能复用物理内存缓冲区以降低峰值内存使用。常见策略有：
  - 原地内存共享（in-place）：允许某个算子的输出覆盖它的某个输入（如果语义允许）。
  - 标准内存共享（buffer reuse）：两个张量在时间上不重叠时复用同一内存块。

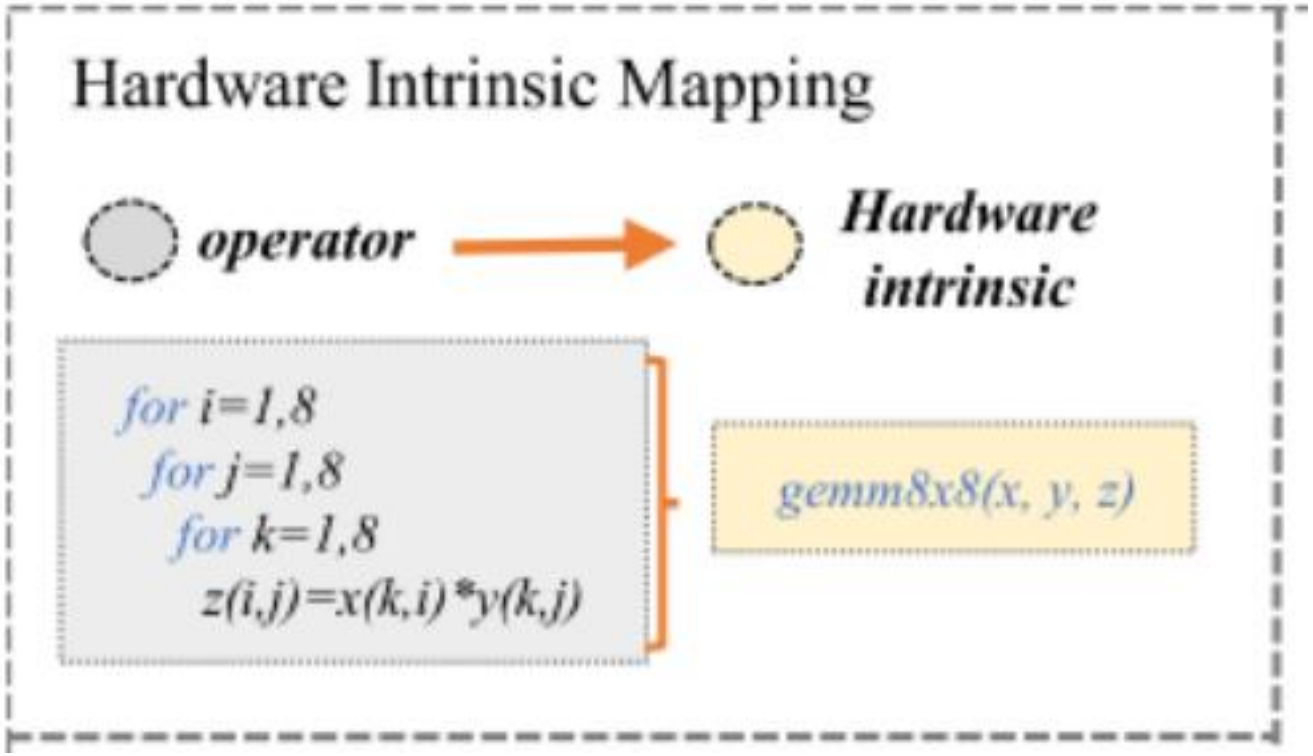
- **布局转换**:为每个硬件/算子选择最优的数据内存布局（layout），并在必要位置插入布局转换节点。

深度学习编译器前端优化



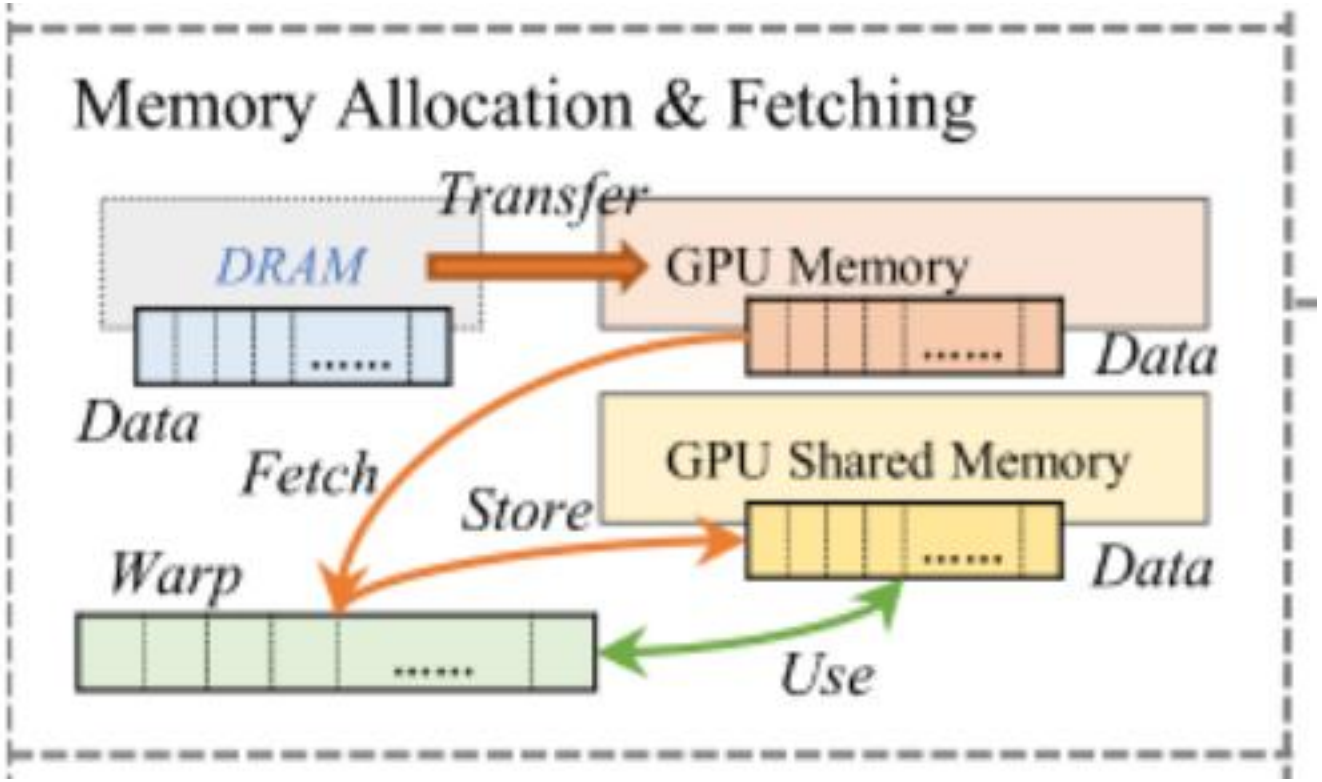
内联映射：

- 硬件内在映射可以将一组特定的低级中间表示指令转换为在硬件上已高度优化的内核



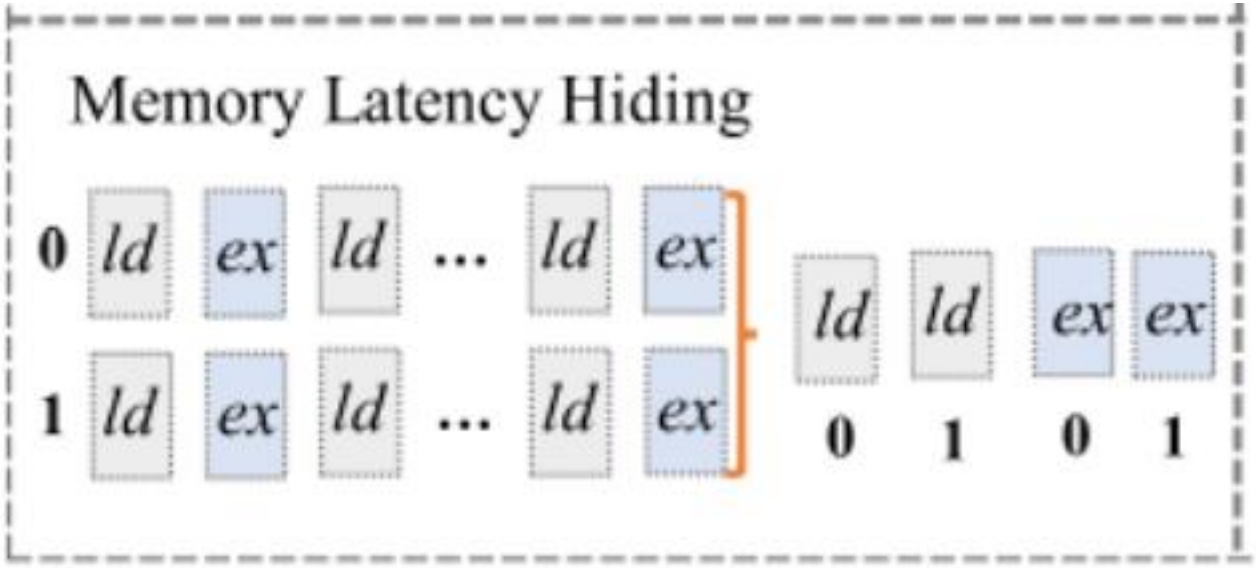
内存层次优化：

- DRAM → GPU Global Memory → Shared Memory → Registers
  - 数据转移（Transfer）：把训练的大张量从 DRAM（主存）搬到 GPU global memory。
  - Fetch / Store
    GPU kernel 内部：
    - 从 global memory fetch
    - 存到 shared memory（更快）
    - 再从 shared memory 在 warp 内使用（更快）
- **合理的访存布局**：编译器会调整 index，使得 warp 中线程连续访问连续地址，减少带宽浪费。



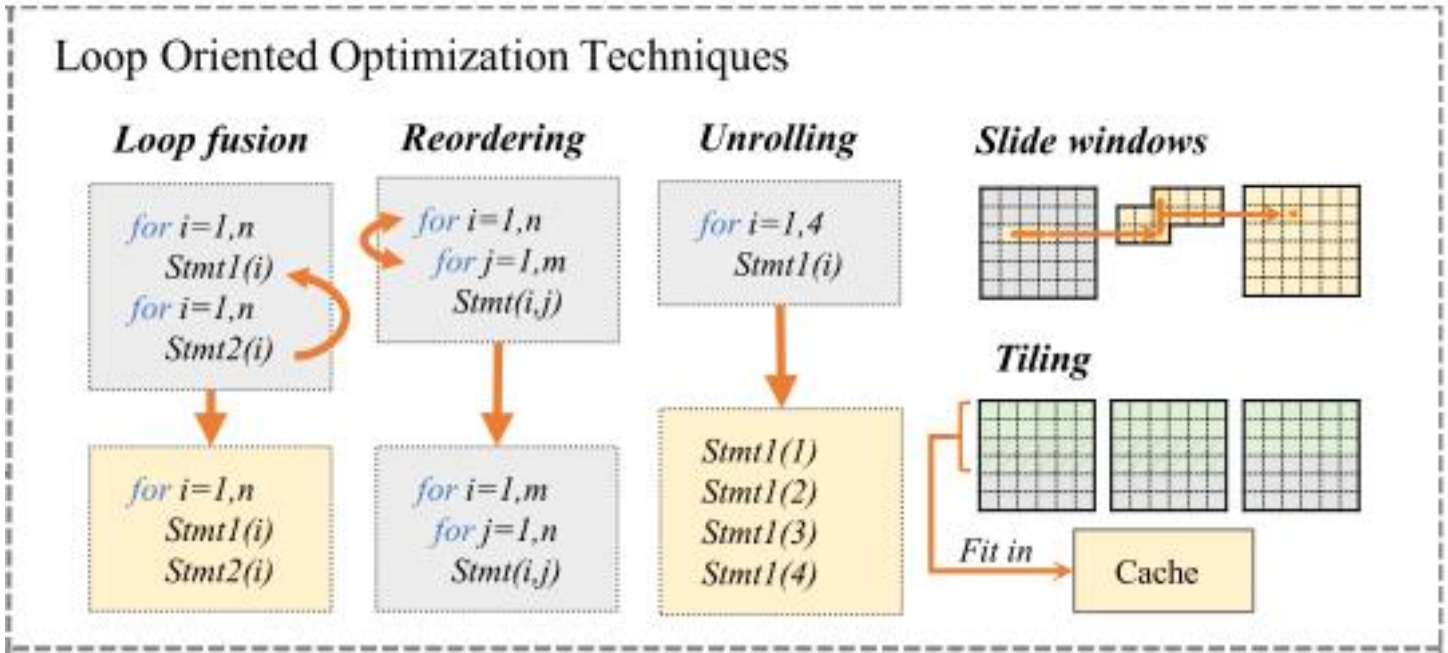
内存延迟隐藏：

- 内存延迟隐藏也是后端使用的一项重要技术，通过重新排序执行流水线来实现。由于大多数深度学习编译器支持在 CPU 和 GPU 上的并行化，内存延迟隐藏可以自然地通过硬件实现（例如，GPU 上的线程束上下文切换）。



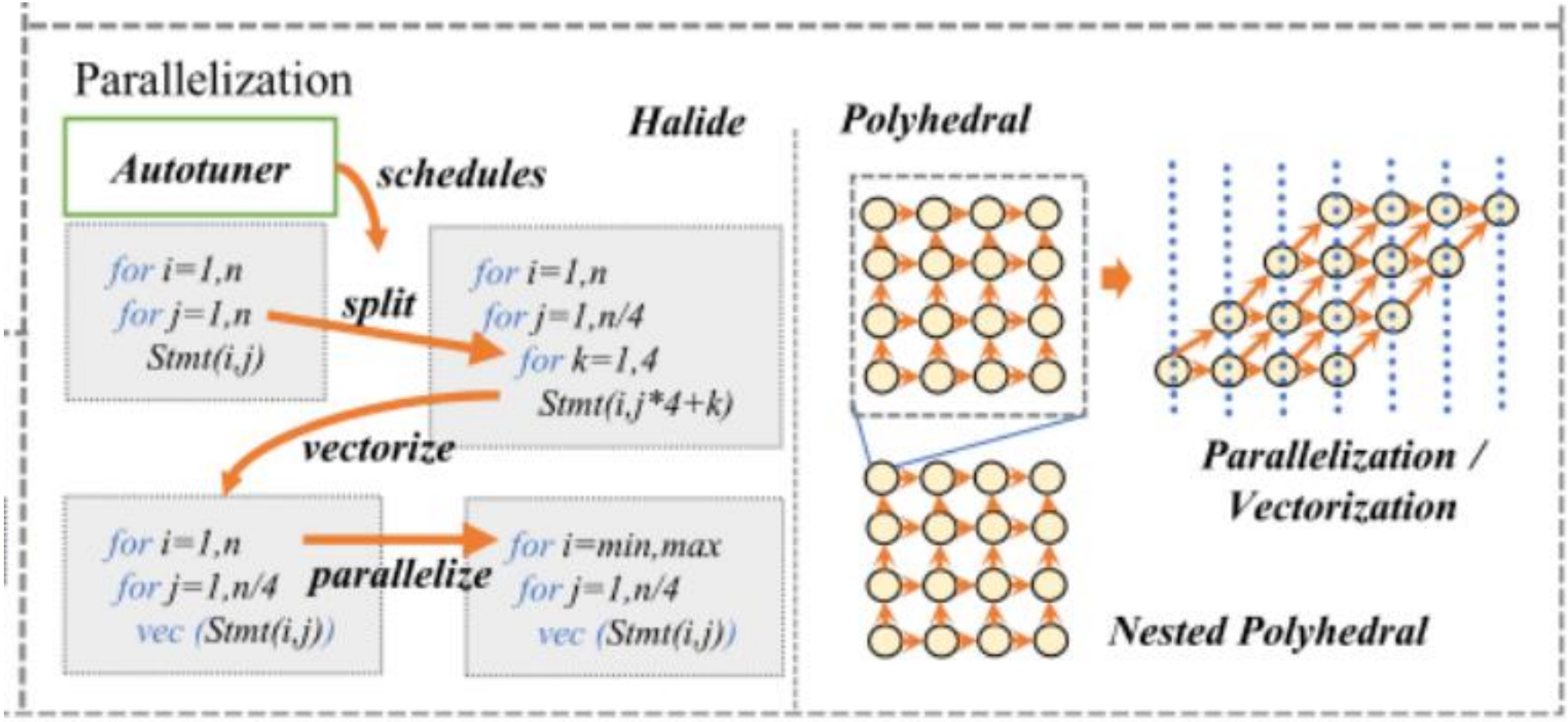
面向循环的优化：

- **循环融合**：可以融合具有相同边界的循环以实现更好的数据重用。
- **滑动窗口**：其核心概念是在需要时计算值，并即时存储这些值以便数据重用，直到不再需要这些值。
- **分块**：分块将循环分割成几个小块，因此循环被分为遍历小块的外层循环和在小块内迭代的内层循环。
- **循环重排序**：改变嵌套循环中迭代的顺序，这可以优化内存访问，从而提高空间局部性。
- **循环展开**：循环展开可以将特定的循环展开为固定数量的循环体副本，这使编译器能够应用激进的指令级并行性



并行化：

- 由于现代处理器通常支持多线程和单指令多数据（SIMD）并行性，编译器后端需要利用并行性来最大限度地提高硬件利用率以实现高性能。



自动调优：

- 由于在特定硬件优化中参数调优的搜索空间巨大，因此有必要利用自动调优来确定最优的参数配置。在本次调研所研究的深度学习编译器中，TVM、TC 和 XLA 都支持自动调优。

  参数化：

  - **数据与目标：**数据参数描述数据的规格，例如**输入形状**。目标参数描述在优化调度和代码生成过程中需要考虑的特定硬件特性和约束。
    - 例如，对于 GPU 目标，需要指定共享内存和寄存器大小等硬件参数。
  - **优化选项：**优化选项包括优化调度和相应的参数，
    - 例如面向循环的优化和分块大小。
    - 在 TVM 中，预定义的和用户自定义的调度以及参数。
    - 小批次维度是通常在 CUDA 中映射到网格维度的参数之一，并且可以在自动调优过程中进行优化。

自动调优：

- 由于在特定硬件优化中参数调优的搜索空间巨大，因此有必要利用自动调优来确定最优的参数配置。在本次调研所研究的深度学习编译器中，TVM、TC 和 XLA 都支持自动调优。

成本模型：

- **黑盒模型**：该模型只考虑最终的执行时间，而不考虑编译任务的特性。TC 采用这种模型。
- **基于机器学习的成本模型**：基于机器学习的成本模型是一种使用机器学习方法来预测性能的统计方法。它使模型能够在探索新配置时进行更新，这有助于实现更高的预测精度。TVM 和 XLA 采用这种模型，
- **预定义成本模型**：基于预定义成本模型的方法期望建立一个基于编译任务特性的完美模型，并能够评估任务的整体性能。与基于机器学习的模型相比，预定义模型在应用时产生的计算开销较小，但在每个新的深度学习模型和硬件上重新构建模型需要大量的工程工作。

自动调优：

- 由于在特定硬件优化中参数调优的搜索空间巨大，因此有必要利用自动调优来确定最优的参数配置。在本次调研所研究的深度学习编译器中，TVM、TC 和 XLA 都支持自动调优。

  搜索技术：

  - **初始化和搜索空间确定**：初始选项可以随机设置，也可以基于已知的配置来设置，
  - **遗传算法（GA）**：遗传算法将每个调优参数视为基因，将每个配置视为一个候选方案。新的候选方案通过交叉、变异和根据适应度值进行选择来迭代生成，最后，得出最优的候选方案。
  - **模拟退火算法（SA）**：模拟退火算法也是一种受退火启发的元启发式算法。它允许以递减的概率接受较差的解决方案，这可以找到近似的全局最优解，并在固定数量的迭代中避免陷入精确的局部最优解。
  - **强化学习（RL）**：强化学习通过在探索和利用之间进行权衡来学习，以在给定环境中最大化奖励。

自动调优：

- 由于在特定硬件优化中参数调优的搜索空间巨大，因此有必要利用自动调优来确定最优的参数配置。在本次调研所研究的深度学习编译器中，TVM、TC 和 XLA 都支持自动调优。

加速：

- **并行化**：加速自动调优的一个方向是并行化。
  - 如：TVM 支持交叉编译和远程过程调用（RPC），允许用户在本地机器上编译，并在多个目标上运行具有不同自动调优配置的程序。
- **配置重用**：加速自动调优的另一个方向是重用先前的自动调优配置。
  - 如：TC 通过编译缓存存储与给定配置相对应的已知最快的生成代码版本。在编译过程中，每次内核优化之前都会查询缓存，如果缓存未命中，则触发自动调优。

深度学习编译器的例子

Table 1. The comparison of DL compilers, including TVM, nGraph, TC, Glow, and XLA.

		TVM	nGraph	TC	Glow	XLA
	Developer	Apache	Intel	Facebook	Facebook	Google
Frontend	Programming	Python/C++ Lambda expression	Python/C++ Tensor expression	Python/C++ Einstein notation	Python/C++ Layer programming	Python/C++ Tensorflow interface
	ONNX support	✓ tvm.relay.frontend .from_onnx (built-in)	✓ Use ngraph-onnx (Python package)	×	✓ ONNXModelLoader (built-in)	✓ Use tensorflow-onnx (Python package)
	Framework support	tvm.relay.frontend .from_* (built-in) tensorflow/tflite/keras pytorch/caffe2 mxnet/coreml/darknet	tensorflow paddlepaddle (Use *-bridge, act as the backend)	(Define and optimize a TC kernel, which is finally called by other frameworks.) pytorch/other DLPack supported frameworks	pytorch/caffe2 tensorflowlite (Use built-in ONNXIFI interface)	Use tensorflow interface
	Training support	× Under developing (Support derivative operators now)	✓ Only on NNP-T processor	✓ (Support auto differentiation)	✓ (Limited support)	✓ Use tensorflow interface
	Quantization support	✓ int8/fp16	✓ int8 (include training)	×	✓ int8	✓ int8/int16 (Use tensorflow interface)
IR	High-/low-level IR	Relay/Halide	nGraph IR/None	TC IR/Polyhedral	Its own high-/low-level IR	HLO (Both high- and low- level)
	Dynamic shape	✓ (*Any*)	✓ (*PartialShape*)	×	×	✓ (*None*)
Optimization	Frontend opt	Hardware independent optimizations (refer to Section 4.3) Hardware specific optimizations (refer to Section 4.4) And hybrid optimizations				
	Backend opt					
	Autotuning	✓ (To select the best schedule parameters)	× (Call optimized kernel libraries, no need)	✓ (To reduce JIT overhead)	× (Additional info is already provided in IR)	✓ (On default convolution and gemm)
	Kernel libraries	✓ mkl/cudnn/cublas	✓ eigen/mkldnn/cudnn/ Others	×	×	✓ eigen/mkl/ cudnn/tensorrt
Backend	Compilation methods	JIT AOT (experimental)	JIT	JIT	JIT AOT (Use built-in executable bundles)	JIT AOT (Generate executable libraries)
	Supported devices	CPU/GPU/ARM FPGA/Customized (Use VTA)	CPU/Intel GPU/NNP GPU/Customized (Use OpenCL support in PlaidML)	Nvidia GPU	CPU/GPU Customized (Official docs)	CPU/GPU/TPU Customized (Official docs)

深度学习编译器的热点研究方向

- 动态shape和动态图支持（Dynamic shape and pre/post processing）
- 高级自动调节（Advanced auto-tuning）
- 多面体模型（Polyhedral model）
- 量化
- 隐私保护

谢谢！