ROLLER: Fast and Efficient Tensor Compilation for Deep Learning

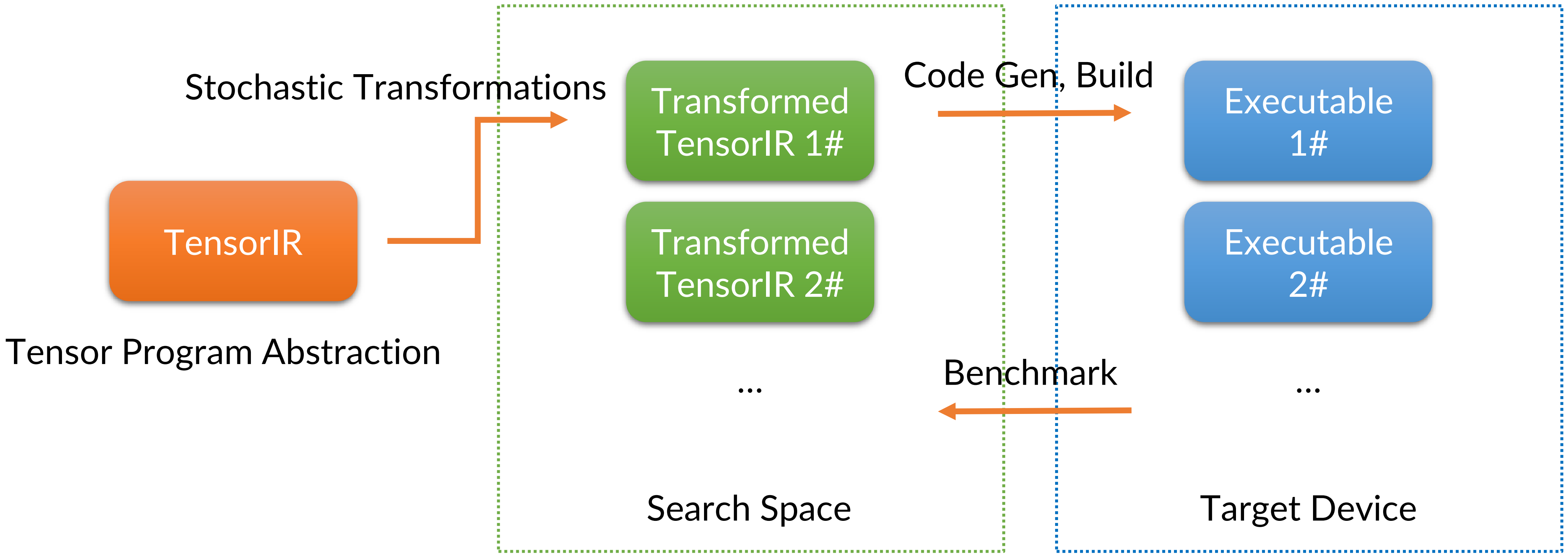
Edge System Reading Group @ SEU

田昊冬

Mar 16, 2023.

Background – MLC 算子级别优化

TVM Approach：随机变换搜索



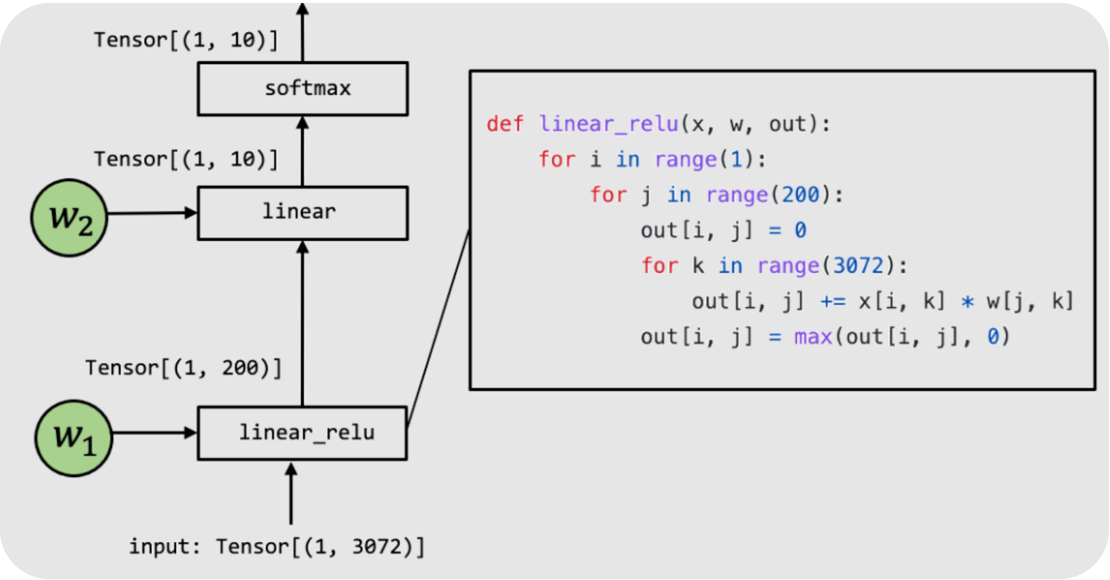
Background – MLC 算子级别优化

传统 MLC 编译/优化流程

- 传统 ML Compiler 将算子翻译为**嵌套多重循环（nested multi-level loops）**

Partitioning/fusion/reordering...

为什么不同的循环变体会导致不同的性能?



Background – MLC 算子级别优化

传统 MLC 编译/优化流程

- 传统 ML Compiler 将算子翻译为**嵌套多重循环（nested multi-level loops）**
- 对张量函数进行**随机程序变换（stochastic transformation）**
- 估算变换后程序的性能

Measure online

Measure in advance

Predict online

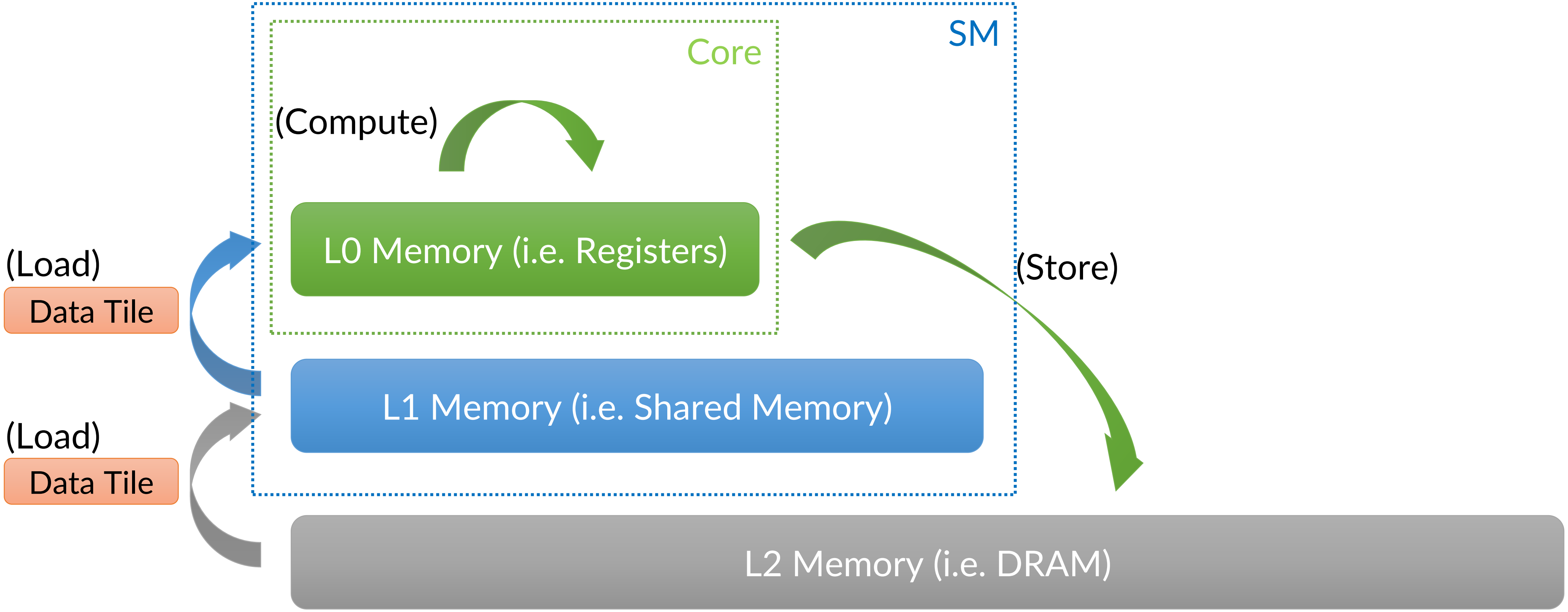
System Design – Overview

Roller 作出了哪些改进?

- 传统 ML Compiler 将算子翻译为嵌套多重循环（nested multi-level loops）
- 提出一种新的张量程序抽象 *data processing pipeline*
- 对张量函数进行随机程序变换（stochastic transformation）
- 估算变换后程序的性能

System Design – *data processing pipeline*

将计算过程抽象为 *data processing pipeline*



System Design – *data processing pipeline*

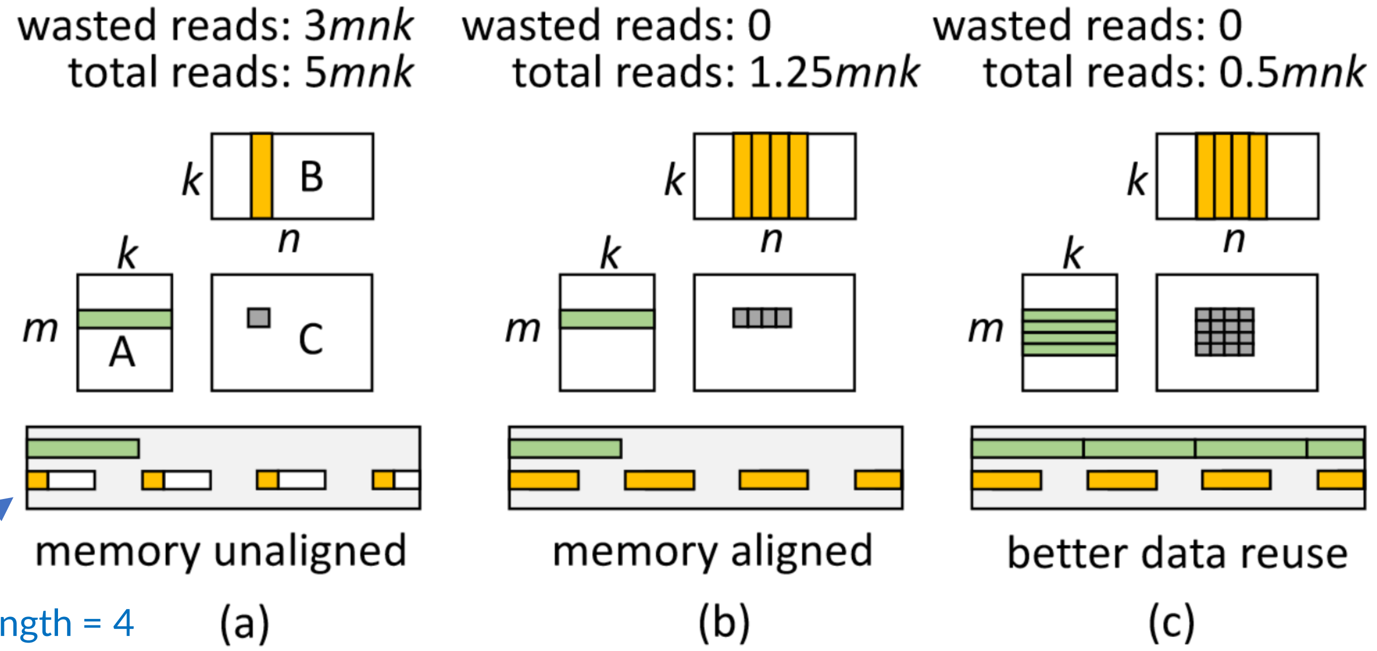
影响 *data processing pipeline* 性能的关键要素

- Tile Shape
- (Tile's) Memory Layout

$$(k + 4k) \times \frac{mn}{1} = 5mnk$$

$$(k + 4k) \times \frac{mn}{4} = 1.25mnk$$

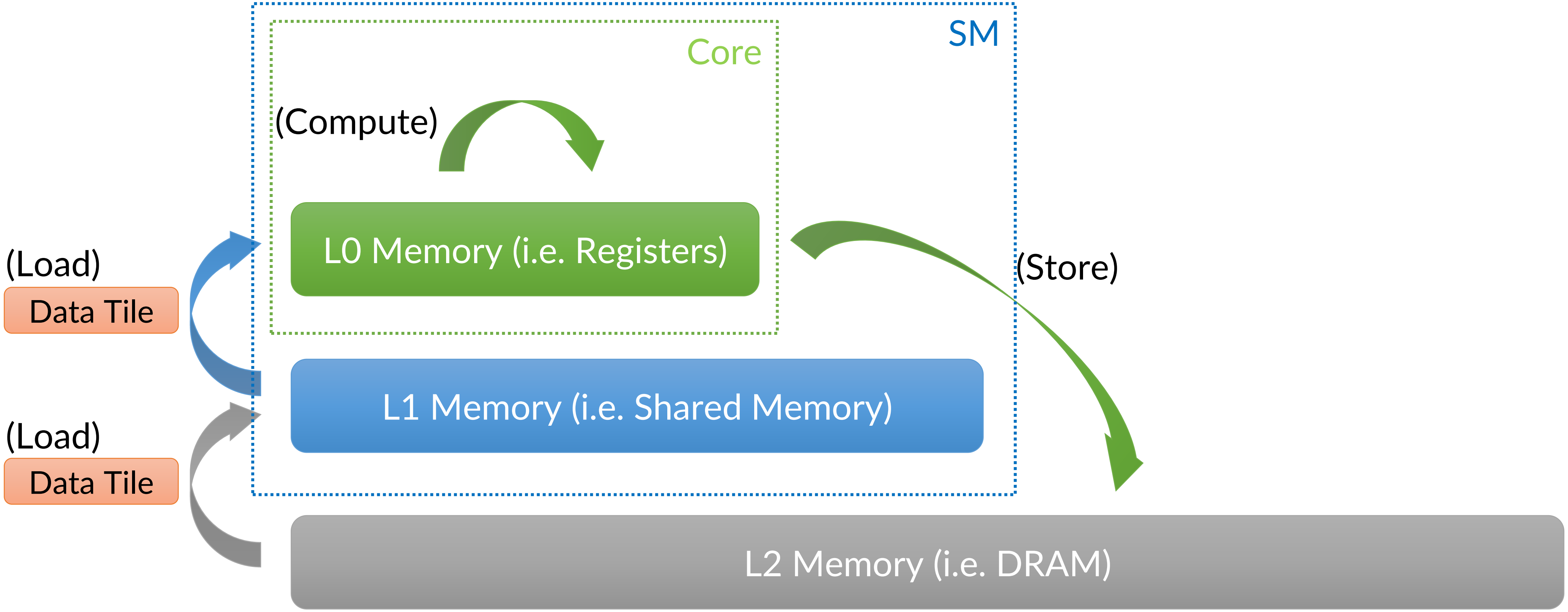
$$(4k + 4k) \times \frac{mn}{4 \times 4} = 0.5mnk$$



Memory transaction length = 4

System Design – *data processing pipeline*

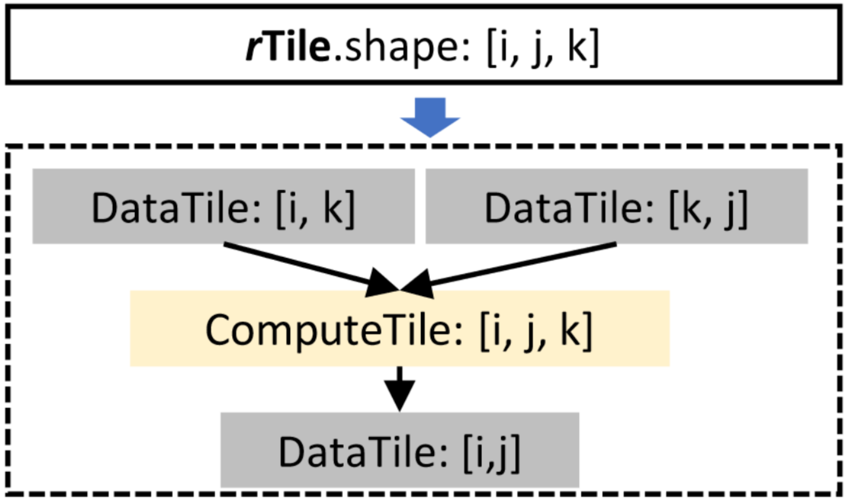
如何选取合适的 Tile Shape? 🤔



System Design – *r*Tile

精确描述 **Load, Compute & Store** 的数据处理过程

```
class rTile {
    TensorExpr expr;
    TileShape shape;
    TileShape storage_padding;
};
```



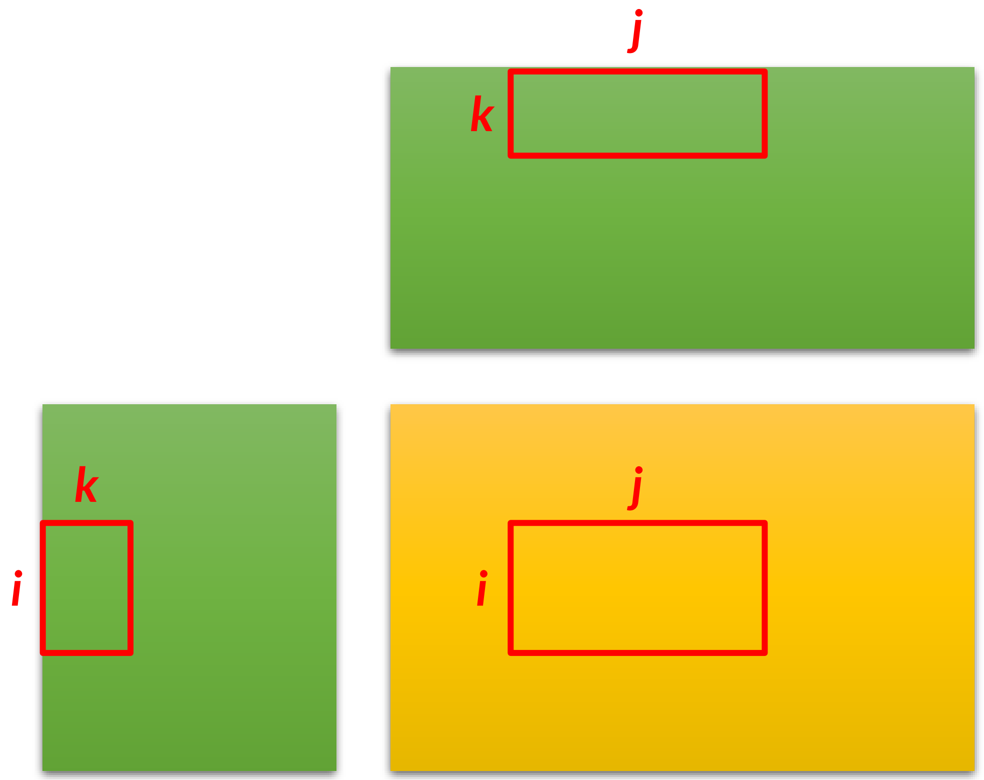

```
.expr = “C=compute((M,N),lambda i,j:sum(A[i,k]*B[k,j]))”
```

System Design – *r*Tile

精确描述 **Load, Compute & Store** 的数据处理过程

***r*Tile**.shape: [*i*, *j*, *k*]

***r*Tile**.expr: “(matmul)”



System Design – *r*Tile

*r*Tile layout 选取时的限制条件（alignment）

- DataTile 的**大小**与硬件**并行度**对齐
- DataTile 的 **leading dimension** 与 **memory transaction length** 对齐
- DataTile (with padding) 的**形状**与 **memory bank 的数量**对齐

如何避免共享内存 Bank Conflict – 知乎

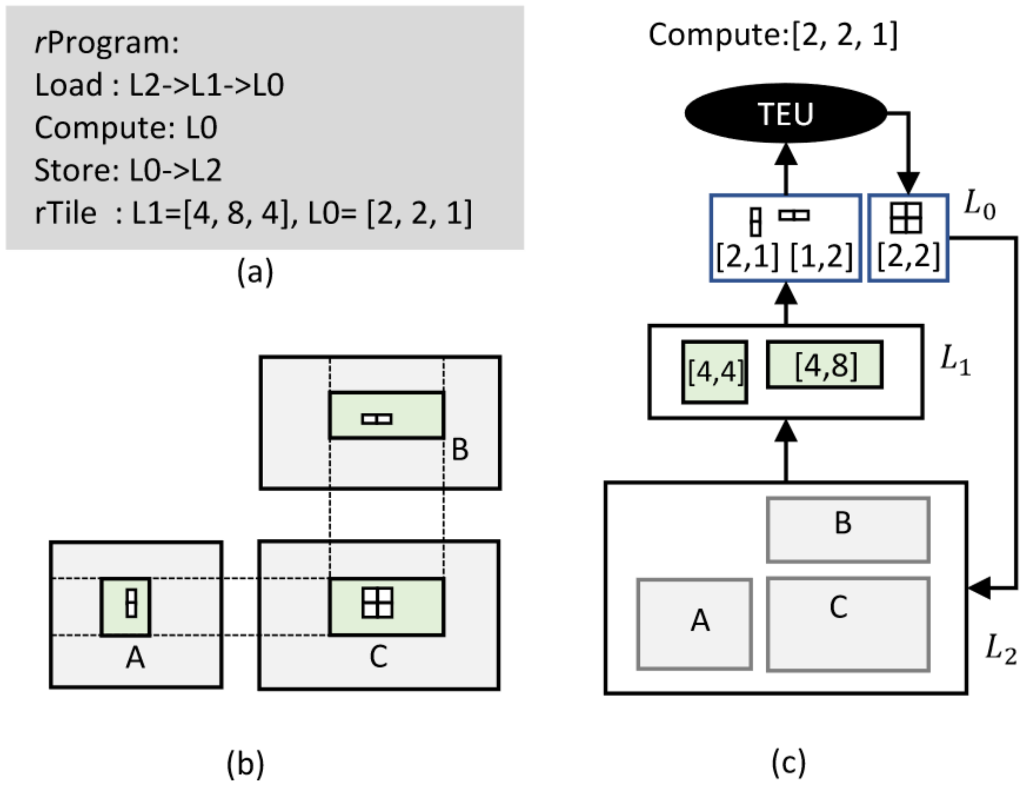
- DataTile 的**形状**与**输入 Tensor shape 的形状**对齐

为输入 Tensor 添加 padding，但需保证浪费比率 < ε

System Design – *r*Tile

用内存层次之间的 *r*Tile configuration 来描述 *r*Program

- ***r*Program** tuning ⇔ find good ***r*Tile** configuration



System Design – Overview

Roller 作出了哪些改进?

- 传统 ML Compiler 将算子翻译为嵌套多重循环（nested multi-level loops）
- 提出一种新的张量程序抽象 *data processing pipeline*
- 对张量函数进行随机程序变换（stochastic transformation）
- 采用构造算法生成 *r*Program (*r*Tile configurations)
- 估算变换后程序的性能
- 在构造算法进行过程中，使用 micro-performance model 估测性能

System Design – Tuning & cost model

构造 *r*Program 的流程

- 自上而下，在每个内存层次之间确定 *r*Tile configuration

什么样的 *r*Tile configuration 是“好”的?

System Design – Tuning & cost model

什么样的 *r*Tile configuration 是“好”的?

- 更“大”的 *r*Tile 是更“好”的（key observation）
- 令当前 *r*Tile 在其各个维度扩张（需要满足 *r*Tile layout 限制条件），得到 *r*Tiles’[]
- 对扩张前后的 *r*Tile 实例 T 和 T'，定义 T' 的 **Data reuse score** 为

$$\frac{MemTraffic(T) - MemTraffic(T')}{MemFootprint(T') - MemFootprint(T)}$$

- 按照 Data reuse score 降序尝试扩张当前的 *r*Tile

System Design – Tuning & cost model

构造 *r*Program 的流程

- 在 L0 层初始化最小 *r*Tile
- 自上而下，在每个内存层次之间尝试 enlarge *r*Tile

*r*Tile 的上限是多少？*r*Tile 扩展到多大就无需继续扩展了？

$$MemFootprint(T) < spec.MemCapacity(layer)$$

$$MemPerf(T) > MaxComputePerf(T.expr)$$

System Design – Tuning & cost model

最后一个问题：Micro-performance model

- MemFootprint(T), MemTraffic(T): statically infer from rTile.shape & .expr
- MaxComputePerf(T.expr): one-time profiling
- MemPerf(T): memory bandwidth / memory traffic

注意 micro-performance model 仅当 *r*Tile 满足**对齐**条件时有效!

System Design – Tuning & cost model

构造 *r*Program 伪代码

```
Func ConstructProg(expr:TensorExpr, dev:Device):
    T = rTile(expr);
    Results = [];
    EnlargeTile(T, dev.MemLayer(0), rProg());
Func EnlargeTile(T:rTile, mem:MemLayer, P:rProg):
    if mem.IsLowestLayer()
        Results.append(P);
        if (Results.Size() > TopK) Exit();
        Return();
    for T' : GetNextRTileShapes(T, mem) do
        if Visited(T')
            Return();
        if MemFootprint(T') > mem.Capacity()
            P.Add(mem, T);
            EnlargeTile(T, mem.Next(), P);
        else
            if MemPerf(T') > MaxComputePerf(T'.expr)
                P.Add(mem, T');
                EnlargeTile(T', mem.Next(), P);
            EnlargeTile(T', mem, P);
```

Conclusion

- 一种新的张量程序抽象：*r*Program (data processing pipeline)
- 充分对齐的 data tile：*r*Tile
- 基于 *r*Tile 的 tensor program tuning 算法：构造法
- 基于 *r*Tile 的 cost model：micro-performance model