DRAM Row Activation Cost Model (Hybrid Approach)
DRAM 行激活开销模型（混合方法）

PIM Optimizer Team

2025 年 12 月 31 日

1 模型总览 (Overview)

本模型采用 ** 混合方法 (Hybrid Approach)** 来计算 DRAM 的 Row Activation 开销。它结合了 ** 高精度模拟查表 (Simulation Lookup)** 和 ** 解析公式 (Analytical Formula)**，以在保证精度的同时维持 ILP 求解的可行性。

总开销公式如下：

$$TotalCost = \underbrace{(1 - z_{aligned}) \cdot Cost_{Seq}}_{\text{顺序布局开销}} + \underbrace{z_{aligned} \cdot Cost_{Aligned}}_{\text{对齐布局开销}} + \underbrace{Cost_{BlockCrossing}}_{\text{输入块跨越开销}} \tag{1}$$

其中：

- $z_{aligned} \in \{0, 1\}$：二进制变量，表示是否选择 `row_aligned` 布局。
- $Cost_{Seq}$：在 `sequential` (tiled) 布局下的基础开销（通过查表计算）。
- $Cost_{Aligned}$：在 `row_aligned` 布局下的基础开销（通过公式计算）。
- $Cost_{BlockCrossing}$：Input Tensor 特有的、由于滑动窗口跨越 Layout Block 边界导致的额外开销（通过 GCD 公式计算）。

2 1. 顺序布局开销 (Sequential Layout Cost)

对于紧凑的 `sequential` 布局，由于 Tile 之间可能存在复杂的地址交错和复用，纯公式难以精确描述。因此，我们采用 ** 查表法 (Lookup Table)**。

2.1 1.1 混合模拟 (Hybrid Simulation)

数据来源是 `precompute_row_acts.py`。该脚本执行以下步骤生成查找表：

2.1.1 A. 地址流生成 (Address Stream Generation)

针对给定的输出 Tile 尺寸 $(P_{tile}, Q_{tile}, C_{tile})$，首先根据卷积公式计算输入 Tile 尺寸 (H_{in}, W_{in})：

$$H_{in} = (P_{tile} - 1) \times stride + (R - 1) \times dilation + 1$$

随后，模拟器生成访问该 Tile 所需的逻辑地址序列。

- **逻辑地址**：基于 Tensor 的线性化索引。
- **物理映射**：将逻辑地址映射到 DRAM 的物理坐标 $(Row, Column, Bank, Rank)$。对于 `sequential` 布局，地址通常是连续的，但当跨越 DRAM Row Size (例如 1KB 或 2KB) 时，物理 Row Index 会发生变化。

2.1.2 B. 微观轨迹模拟 (Micro-Trace Simulation)

为了捕捉 Tile 内部以及 Tile 之间的 Row Buffer 行为，我们模拟了一组连续的 Tile 访问序列（通常为 64 个 Tile）。

1. **状态维护**：模拟器维护当前 Row Buffer 中打开的 Row Index (Row_{open})。
2. **访问处理**：对于地址流中的每一个内存请求 req：
   - 如果 $req.RowIndex \neq Row_{open}$：发生 **Row Buffer Miss**。
   - 动作：关闭旧行 (Precharge)，激活新行 (Activate)。
   - 计数：$Activations \leftarrow Activations + 1$。
   - 更新：$Row_{open} \leftarrow req.RowIndex$。
3. **跨 Tile 效应**：通过模拟连续的 64 个 Tile，模型能够捕捉到前一个 Tile 的末尾与后一个 Tile 的开头是否位于同一 DRAM Row，从而精确计算 **Inter-Tile Locality**。

2.1.3 C. 关于 L1 Tile 的假设 (Assumption on L1 Tile)

目前的微观模拟假设 L2 Tile 内部是按照给定的 `loop_order` (如 C-H-W) 进行逐元素访问的。它**没有显式模拟 L1 Buffer Tile** 的分块循环。

- 如果硬件的数据加载/计算顺序是严格遵循 L1 Tiling (例如先处理完一个 32×32 的 L1 Block，再处理下一个)，当前的逐行/逐通道扫描可能是一种近似。
- 然而，对于 DRAM Row Activation 而言，只要 L2 Tile 的填充是主导因素（例如 DMA 线性搬运），这种近似通常是足够的。如果 L1 访问顺序导致了剧烈的地址跳变 (Thrashing)，当前模型可能低估开销。

2.1.4 D. 术语解释：DMA 线性搬运与地址连续性

这里提到的"DMA 线性搬运"是指现代加速器中常见的 ** 解耦 (Decoupled)** 访问模式。这也回答了关于地址连续性的问题：

- **地址连续性**：是的，在计算 $Cost_{Seq}$ 时，我们假设 DRAM 中的数据采用了 **Tiled Layout**（分块存储）。这意味着一个 L2 Tile 所需的数据在物理内存中是 ** 连续存储 **（或高度局部化）的。
- **DRAM → L2 Buffer**：因此，DMA 控制器可以按照地址递增的顺序（线性）批量读取整个 L2 Tile。这种连续读取最大化了 Row Buffer Hit 率。
- **对比 L1 访问**：数据进入片上 L2 Buffer 后，计算单元 (PE) 可能会以复杂的顺序（如 L1 Tiling）反复读取这些数据，但这发生在片上，不再影响 DRAM Row Activation。

2.1.5 E. 关于 L1 Tile 与边界跨越 (L1 Tile & Boundary Crossing)

用户可能会问：**“如果一个 L1 Tile 恰好处于 Block 的分界线上，在处理该 L1 Tile 时岂不是会产生大量的 Row Activation?”**

答案是肯定的，这正是本模型设计的核心逻辑之一：

- **捕捉高开销**：如果 L2 Tile 的尺寸或位置导致其内部包含了 Block 边界，并且访问模式（如 Micro-Trace 模拟的逐行扫描）在边界处反复横跳，模拟器会如实记录下大量的 Row Activation。
- **保守策略 (Conservative Strategy)**：目前的 Micro-Trace 假设 L2 Tile 的加载是简单的逐行/逐通道扫描 (Flat Scan)，而不假设硬件会智能地按照 L1 Tile 的块状顺序去加载（Block-wise Load）。
- **优化结果**：由于这种“边界跨越”在模拟中会产生极高的 Cost，ILP 优化器会倾向于 ** 避免 ** 这种情况。它会选择让 L2 Tile 的尺寸与 DRAM Block 尺寸 ** 对齐 **（例如 W_{tile} 是 $Block_W$ 的整数倍），从而在物理上消除边界跨越。
- **结论**：虽然我们没有显式模拟 L1 Tile，但通过对“跨界行为”施加高额惩罚（基于保守的扫描假设），模型成功迫使优化器找到了对齐良好的配置。在对齐的配置下，L1 Tile 自然也不会跨越边界，或者边界效应被最小化。

2.1.6 F. 平均化 (Averaging)

最终的平均开销计算如下：

$$AvgCost_{entry} = \frac{\text{Total Activations over 64 Tiles}}{64}$$

该值通常在 $[1.0, 2.0]$ 之间，表示平均每个 Tile 需要多少次 Row Activation。

- 1.0 表示完美的 Row Buffer Hit（所有数据都在同一行，或与前一个 Tile 连续）。
- > 1.0 表示 Tile 内部跨越了行边界，或者 Tile 之间存在跳跃。

2.2 1.2 ILP 查表约束

ILP 模型通过引入二进制变量 z_{entry} 来选择最优的 **Row Buffer (L2) Tile** 配置。
需要注意的是，**复用惩罚 (Reuse Penalty)** 仅应作用于 **Crossing Tile**。

- **Safe Tile**：在 Row 内部，重复访问 (Reuse) 只会产生 Row Buffer Hit，开销不变。
- **Crossing Tile**：跨越了 Row 边界，重复访问会导致 Row Buffer Ping-Pong 切换，开销随 Reuse 次数线性增加。

因此，修正后的计算公式为：

$$Cost_{Seq} = \sum_{i \in Table} z_i \cdot N_{tiles} \cdot [1 + (AvgCost_i - 1) \cdot ReusePenalty] \tag{2}$$

其中：

- z_i：如果选择了表中的第 i 种配置，则为 1。

- N_{tiles}：**Row Buffer (L2) Tile 的总数量**。
- 1：基础开销 (Base Cost)，即每个 Tile 至少需要一次激活。
- $AvgCost_i - 1$：模拟得到的额外开销 (Crossing Overhead)，代表跨行概率。

2.3 1.3 复用惩罚 (Reuse Penalty)

如果循环顺序 (Loop Order) 导致连续访问的 Tile 在物理空间上不连续（例如先切换 Channel 而不是先切换 Row），则会产生额外的开销。

$$ReusePenalty = \prod_{j \in IrrelevantDims} Bound_j^{x_j} \tag{3}$$

这确保了只有当循环顺序与数据布局匹配时，才能享受到查表得到的低开销。

3 2. 对齐布局开销 (Row Aligned Layout Cost)

在 `row_aligned` 布局中，数据的每一行（或每个 Channel）都被强制填充 (Padding) 到 DRAM Row 的边界。

- **特点**：消除了 Tile 内部的 Row Crossing，因为每个 Tile 都从新的 Row 开始（或对齐的位置）。
- **计算**：开销主要取决于总数据量除以 Row Size，加上对齐带来的浪费。

$$Cost_{Aligned} = \text{TotalBytes}_{padded}/\text{RowSize} \cdot ReusePenalty \tag{4}$$

4 3. 输入块跨越开销 (Input Block Crossing Cost)

这是针对 Input Tensor 的特殊项。由于卷积的 ** 滑动窗口 (Sliding Window)** 特性，Input Tile 可能会跨越 **Layout Block** 的边界（注意：这是逻辑 Block 边界，不同于 DRAM Row 边界）。

这部分开销使用 **GCD 解析公式 ** 计算：

$$Cost_{BlockCrossing} = N_{tiles} \cdot P_{block\_cross} \cdot C_{penalty} \tag{5}$$

其中跨越概率 P_{block_cross} 由 GCD 周期性分析得出：

$$P_{block\_cross} = \frac{\text{CrossCount}}{\text{Period}} = \frac{\text{Period} - \lceil (BlockH - TileH + 1)/g \rceil}{\text{Period}} \tag{6}$$

其中 $g = \gcd(Step, BlockH)$，$Period = BlockH/g$。

5 4. 验证方法与结果 (Validation Methodology & Results)

为了验证本模型的准确性，我们将 ILP 模型的预测结果与基于周期的 Ground Truth 模拟器 (`TraceGenerator`) 进行了对比。

5.1 4.1 验证策略 (Validation Strategy)

我们对 ResNet 架构中的关键层 (Conv2_x, Conv3_x, Conv4_x) 进行了全面的扫描实验。实验变量包括：

- **Tile Size (P_{tile}, Q_{tile})**: 覆盖了从 32×32 到 56×56 的多种组合。
- **对齐情况 (Alignment)**: 包含完全对齐 ($Tile\%Block == 0$) 和非对齐 ($Tile > Block$) 的情况。

5.2 4.2 关键发现：Packed vs Strided 传输

在验证初期，我们发现 ILP 模型与 GT 存在约 1.8% 的系统性误差。深入分析揭示了两者在 ** 物理内存布局 (Physical Memory Layout)** 假设上的差异：

- **ILP 模型 (Strided)**: 默认假设 Input Tile 是从大图 (Large Tensor) 中切片读取的。这意味着每读完一行 Tile，地址需要跳过图像剩余宽度 (Stride/Gap)。这种跳跃可能导致额外的 Row Activation。
- **TraceGenerator (Packed)**: 在 `sequential` 模式下，默认模拟的是 ** 紧凑 (Packed)** 的缓冲区传输，即假设 Tile 数据在内存中是连续存放的，行与行之间无空隙。

5.3 4.3 修正与对齐 (Correction & Alignment)

为了验证 ILP 核心公式的数学正确性，我们采取了以下措施来对齐两者的假设：

1. **ILP 端**: 将配置调整为 Packed 模式 (设置 $TensorWidth = TileWidth$)，消除 Stride 带来的额外开销。
2. **GT 端 (Dummy Workload)**: 构造了一个特殊的"Dummy Workload"，其输入尺寸精确等于待验证的 Input Tile 尺寸 ($H_{in} \times W_{in}$)。这强制 `TraceGenerator` 执行一次完全线性的、无 Halo 的内存传输。

5.4 4.4 最终结果 (Final Results)

经过上述对齐后，验证结果显示了极高的一致性：

- **相关性 (Correlation)**: **0.9998** (Pearson)
- **平均误差 (Mean Error)**: **0.30%**
- **最大误差 (Max Error)**: $< 3\%$

这一结果证明了：1. 本模型的 Row Activation 核心计算公式 ($Cost = N \times [1 + (Avg - 1) \times Reuse]$) 在数学上是精确的。2. 模型具有足够的灵活性，既能模拟 Packed 传输 (当前验证配置)，也能通过参数调整支持 Strided 传输 (更接近某些硬件的真实场景)。

6 总结 (Summary)

本模型与最初的简单公式 ($N \times (1 + P)$) 相比，主要改进在于：

1. **精度提升**：使用 **Hybrid Simulation** 替代了难以推导的 $P_{crossing}$ 公式，精确捕捉了 Address Discontinuity 和 Layout 细节。
2. **解耦**：将 **DRAM Row Crossing**（物理层，查表）与 **Layout Block Crossing**（逻辑层，公式）分离计算。
3. **灵活性**：通过 ILP 的二进制变量自动搜索最优的 Tile 尺寸，而不是预先固定。