实验二：基于 AVX-512 的整数量化矩阵乘法优化实验报告

姓名： 苏易文
学号： 3240103466
日期： 2025年7月6日

---

1. 实验思路

本实验优化 `uint8_t * int8_t` 整数量化矩阵乘法（`C = A * B_transposed`），采用数据重排与SIMD并行计算相结合的策略。

1.1. 性能瓶颈分析

基准 `naive_gemm` 的主要瓶颈：

1. **计算串行化**：三层循环无法利用CPU向量处理单元，计算效率低
2. **缓存不友好**：内存访问模式导致缓存利用率低下

1.2. 核心优化策略

1. **数据重排**：对B矩阵进行块转置，将原始B矩阵的 `1x4` 数据块重新排列，使相关数据在内存中连续存放
2. **AVX-512 Tiling**：
    - 将C矩阵分解为 `1x16` 小块处理，充分利用512-bit向量寄存器
    - 使用 `_mm512_broadcastd_epi32` 广播A矩阵数据块
    - 使用 `_mm512_loadu_si512` 连续加载B矩阵16个数据块
    - 使用 `_mm512_dpbusd_epi32` 并行计算16组4元素点积

---

2. 实验结果与分析

2.1. AVX-512 优化结果

实现方法	运行时间	加速比
`naive_gemm`	0.112492s	1.0× (基准)
AVX-512 优化版	**0.067263s**	**1.67×**

性能分析：

- **SIMD并行化**：向量指令大幅提升计算吞吐量
- **内存访问优化**：数据重排和分块计算改善缓存局部性

2.2. 核心代码实现

```
// AVX-512 优化核心代码
for (int i = 0; i < M; ++i) {
    for (int j = 0; j < N; j += 16) {
        __m512i c_vec_accumulator = _mm512_setzero_si512();

        for (int k = 0; k < K; ++k) {
            // 广播A矩阵数据块
            __m128i a_block_128 = _mm_cvtsi32_si128(*(const int*)
(&A[i * K * 4 + k * 4]));
            __m512i a_vec_broadcasted =
_mm512_broadcastd_epi32(a_block_128);

            // 加载B矩阵连续数据块
            __m512i b_vec_packed = _mm512_loadu_si512((__m512i
const*)&B_reshape[k * N * 4 + j * 4]);

            // 执行16组4元素点积累加
            c_vec_accumulator =
_mm512_dpbusd_epi32(c_vec_accumulator, a_vec_broadcasted,
b_vec_packed);
        }

        // 写回结果
        _mm512_storeu_si512((__m512i*)&C[i * N + j],
c_vec_accumulator);
    }
```

```
}
```

---

3. Bonus：基于 AMX 的终极优化

3.1. AMX 优化策略

Intel AMX将优化维度从一维向量提升到二维瓦片，提供专用矩阵运算硬件：

- **硬件配置**：使用 `_tile_loadconfig` 配置瓦片寄存器形状
- **二维数据流**：`_tile_loadd` 加载二维瓦片，`_tile_dpbusd` 执行瓦片矩阵乘法
- **专用引擎**：TMUL单元提供极高计算密度

3.2. AMX 优化结果

实现方法	运行时间	加速比
`naive_gemm`	3.01185s	1.0× (基准)
AMX 优化版	**0.0249321s**	**120.8×**

3.3. AMX 核心代码

```
// AMX优化核心代码
init_tile_config(M);

for (int j = 0; j < N; j += 16) {
    _tile_zero(0);  // 初始化结果瓦片

    for (int k = 0; k < K * 4; k += 64) {
        // 加载A瓦片 (12x64)
        _tile_loadd(1, A + k, K * 4);

        // 加载B瓦片 (16x64)
        void* b_addr = (uint8_t*)B_reshape + k_block_idx * 16 * 128
+ j_block_idx * 64;
        _tile_loadd(2, b_addr, 128);

        // 执行瓦片矩阵乘法累加
        _tile_dpbusd(0, 1, 2);
    }
```

```
    // 存储结果瓦片
    _tile_stored(0, C + j, N * 4);
}
```

---

4. 思考题简答

(1) NumPy中None的作用

`None` 用于增加维度，为广播机制做准备。`(x - x_idx)[:,:,:,None]` 将形状从 `(N,H2,W2)` 变为 `(N,H2,W2,1)`，支持与 `(N,H2,W2,C)` 进行广播运算。

(2) 高级索引的shape和作用

`a[n_idx, x_idx, y_idx]` 最终形状为 `[N, H2, W2, C]`。通过广播机制组合索引数组，并行检索所有对应位置的像素数据。

(3) 向量广播机制

维度不同：第一个向量是 `[N,H2,W2,C]`，后两个是 `[N,H2,W2,1]`。广播方式是将后两个向量复制C份进行相乘。

---

5. 总结

本实验通过AVX-512和AMX两个层次的优化，展示了现代CPU向量化和专用硬件的强大威力。AVX-512实现了1.67倍加速，而AMX更是达到了120倍的惊人提升，证明了专用硬件在特定计算场景下的绝对优势。