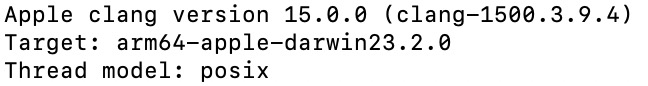
# **arm64优化卷积计算**

# **汇编与接口技术实验报告**

1. 实验环境

编译器：clang version 15.0.0

cpu：Apple M1 pro

操作系统：MacOS

1. 流程

常规实验：参考《人工智能应用场景下基于汇编语言的性能优化案例》，其中提到两种思路，思路一：**循环展开卷积运算**（不改变输入的存储方式—），思路二：**Im2col+sgemm**（改变输入的存储方式）。

由于k\_size = 4的卷积核并不常用（一方面偶数核难以对称实现填充；另一方面使用4\*4卷积核的GAN模型黄金时代已逝，打不过众多扩散模型、流模型变体了），所以我们对书中的实验设计进行调整，如下：

**思路一进行两组组优化尝试：**

①手动展开k\_size=3卷积运算（C\_loop\_Kernel3x3、asm\_loop\_Kernel3x3）

针对3\*3的卷积核，基于Armv8架构展开乘法、加法运算过程。（手动展开关于卷积核的两层循环过程，针对最常用的卷积尺寸。对特定模型，此种方法直接进行卷积核运算展开确有一定的实用意义）

②基于循环展开任意尺寸卷积核运算（C\_loop\_Kernel\_any、asm\_loop\_Kernel\_any）

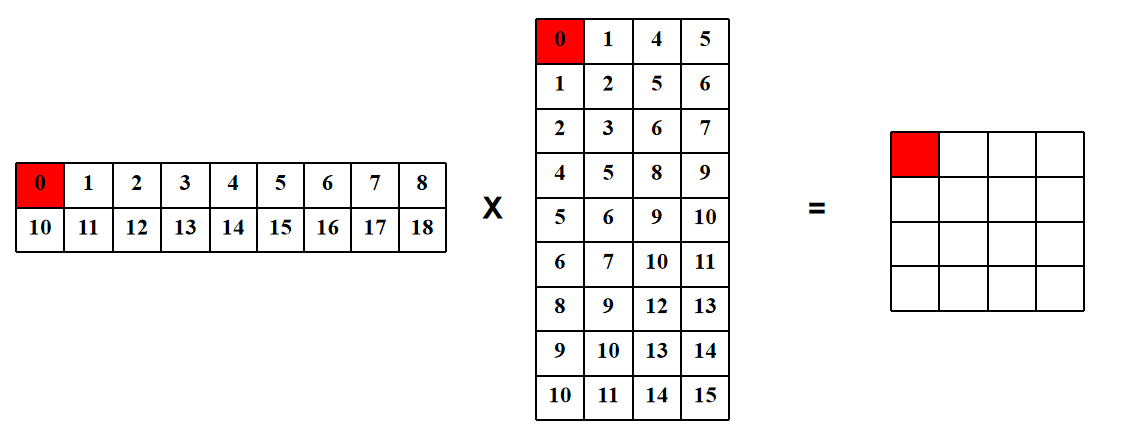
根据向量寄存器特性结合k\_size=4的展开实现过程，将整体卷积运算的循环基于卷积1\*4进行展开，不足以进行按照4\*4个元素展开的部分分别基于行列进行1\*4、4\*1形式展开以及不展开运算，展开计算部分主要基于向量寄存器以及SIMD指令的使用进行优化。

**思路二进行三组优化尝试：**

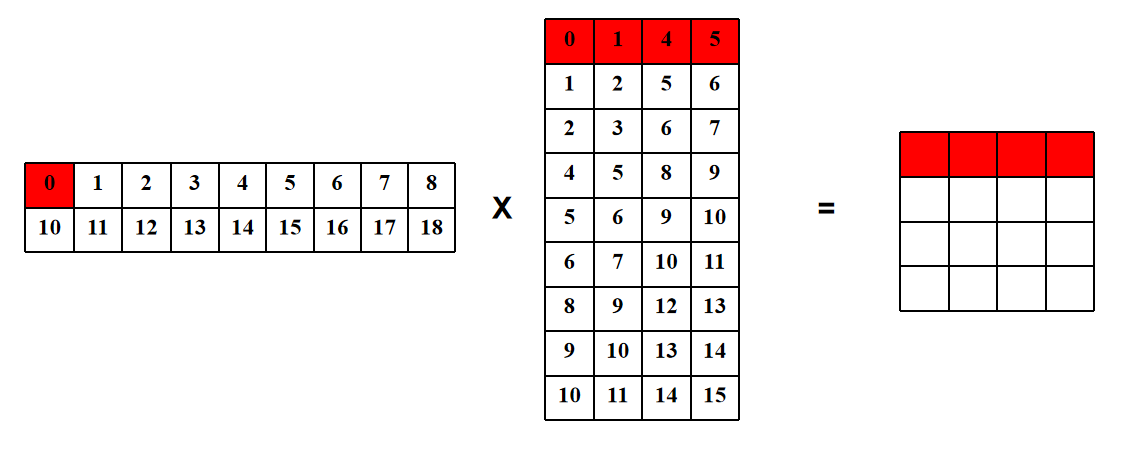
因为卷积过程存在的重复多次乘法的特性，从而根据其运算过程的特点转换为两个矩阵相乘的形式，由此将卷积优化问题转化为矩阵乘法优化问题。此种方式需要改变输入图像以及卷积核的存储方式，在这个过程增加了额外内存空间的使用。这种方式的重点在于矩阵乘法如何进一步优化。首先将输入图像转变存储方式，根据卷积核的尺寸将原本的输入图像进行Im2col转变，将输入图像与卷积核的卷积运算过程变成两个矩阵的乘法形式，基于最一般的矩阵相乘的形式进行循环展开并结合向量寄存器的使用，通过增加并行性计算提升时间。

①Im2col+sgemm+未进行矩阵乘法展开（C\_Sgemm\_op1）

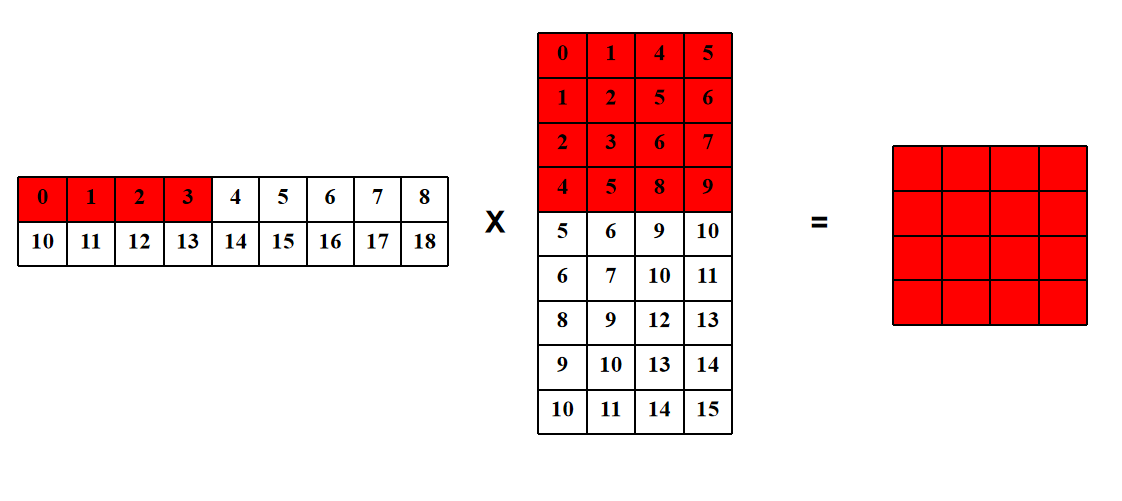
将输入图像按照Im2col转换存储方式，进行简单矩阵乘法运算过程。



②Im2col+sgemm+基于1\*4进行矩阵乘法展开（C\_Sgemm\_op4、asm\_Sgemm\_op4）

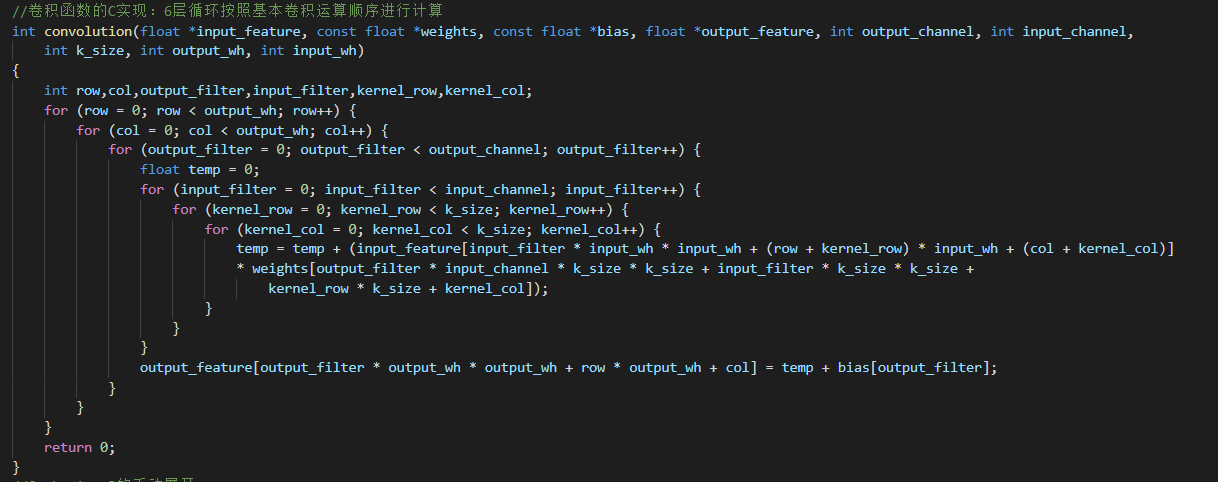


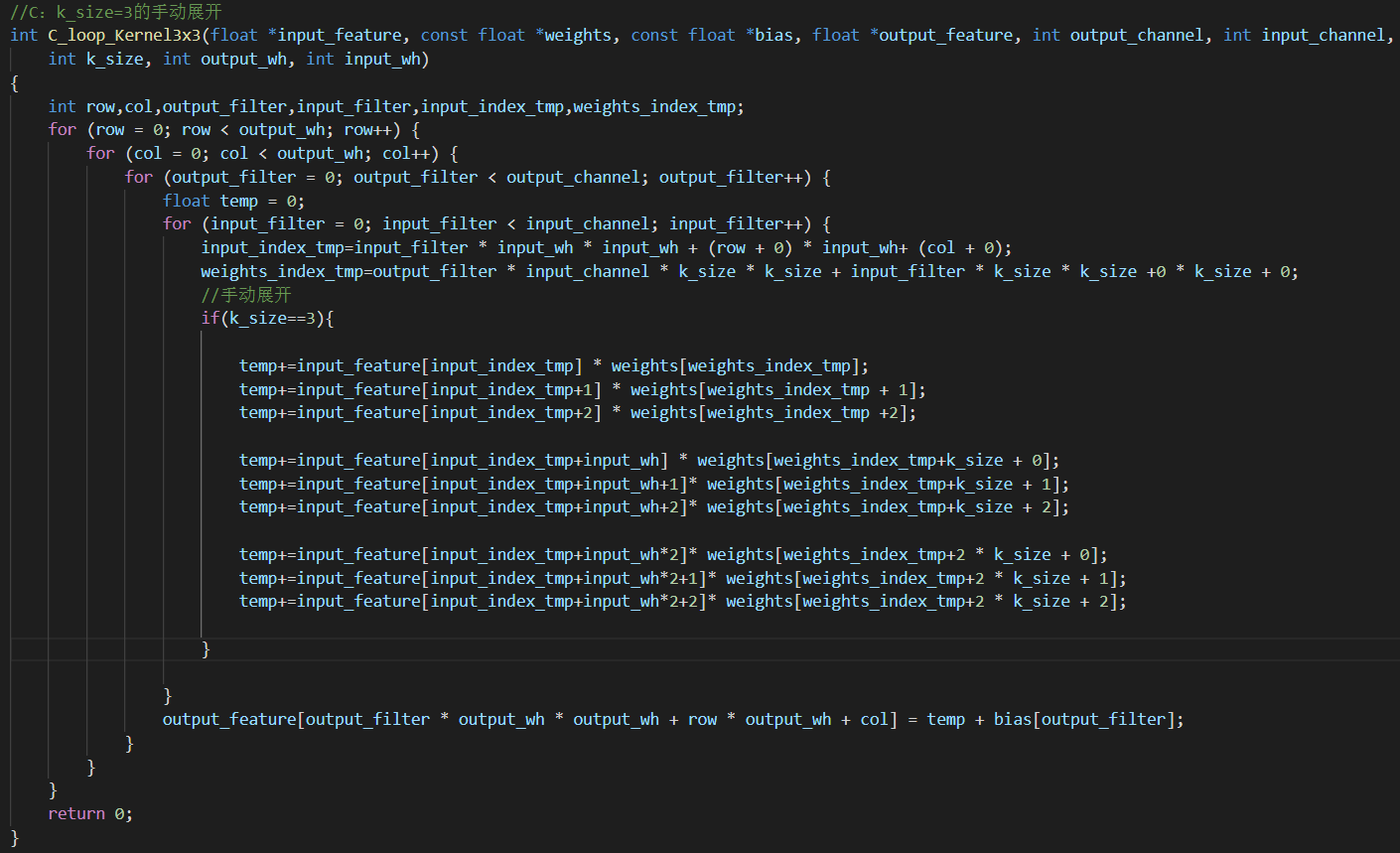
③Im2col+sgemm+基于4\*4进行矩阵乘法展开的汇编优化实现（C\_Sgemm\_op16、asm\_Sgemm\_op16）



1. 实验过程

初始c实现，C\_loop\_Origin.c：

思路一①手动展开的卷积核C实现，C\_loop\_Kernel3x3.c：

思路一①手动展开的卷积核汇编实现，asm\_loop\_Kernel3x3.c

思路一②基于循环展开的卷积核C实现，C\_loop\_Kernel\_any.c

思路一②基于循环展开的卷积核汇编实现，asm\_loop\_Kernel\_any.c

思路二：

思路二①Im2col+sgemm+未进行矩阵乘法展开，C\_Sgemm\_op1.c

思路二②Im2col+sgemm+基于1\*4进行矩阵乘法展开，C\_Sgemm\_op4.c

思路二②Im2col+sgemm+基于1\*4进行矩阵乘法展开，asm\_Sgemm\_op4.c

思路二③Im2col+sgemm+基于4\*4进行矩阵乘法展开的汇编优化实现，C\_Sgemm\_op16.c

思路二③Im2col+sgemm+基于4\*4进行矩阵乘法展开的汇编优化实现，asm\_Sgemm\_op16.c

1. 实验结果

思路一：

|  |  |  |  |
| --- | --- | --- | --- |
| （ms） | filter=3x3 | | |
|
| 尺寸  实现 | 256x256x1 | 640x640x1 | 640x640x3 |
|
| 初始C | 55.841 | 228.234 | 613.783 |
|
| C\_loop | 32.452 | 141.053 | 619.289 |
|
| asm\_loop | 14.781 | 72.5 | 136.354 |
|
| C加速比 | **1.72** | **1.618** | **0.991** |
|
| asm加速比 | **3.779** | **3.148** | **4.501** |
|

|  |  |  |  |
| --- | --- | --- | --- |
| （ms） | filter=5x5 | | |
|
| 尺寸  实现 | 256x256x1 | 640x640x1 | 640x640x3 |
|
| 初始C | 96.761 | 575.479 | 1714.208 |
|
| C\_loop | 245.745 | 1447.494 | 4265.446 |
|
| asm\_loop | 88.991 | 420.947 | 1203.940 |
|
| C加速比 | **0.393** | **0.398** | **0.402** |
|
| asm加速比 | **1.087** | **1.367** | **1.424** |
|

|  |  |  |  |
| --- | --- | --- | --- |
| (ms) | filter=7x7 | | |
|
| 尺寸  实现 | 256x256x1 | 640x640x1 | 640x640x3 |
|
| 初始C | 194.297 | 1133.369 | 3345.442 |
|
| C\_loop | 223.884 | 1425.128 | 4246.648 |
|
| asm\_loop | 162.442 | 909.062 | 2682.791 |
|
| C加速比 | **0.868** | **0.795** | **0.788** |
|
| asm加速比 | **1.196** | **1.247** | **1.247** |
|

解析：由于现在的编译器自带强大的优化功能，对于更简单的初始C实现代码，编译器有更大的优化空间，这就是C\_loop效果有时不及初始C实现的原因。另外可以看到存在卷积核不同但耗时相同的情况，这是因为在本实验中padding均为0，这意味着更大的卷积核算出来的feature map更小，自然计算量就不能简单用卷积核来衡量了。但asm版本恒优于C实现，这是汇编强大功能的体现。

思路二：

|  |  |  |  |
| --- | --- | --- | --- |
| 3x3 | | | |
|
| 尺寸  实现 | 256x256x1 | 640x640x1 | 640x640x3 |
|
| 初始C | 55.841 | 228.234 | 613.783 |
|
| C\_op1 | 56.503 | 243.732 | 659.374 |
|
| C\_op4 | 30.837 | 145.237 | 355.254 |
|
| C\_op16 | 22.675 | 85.494 | 179.758 |
|
| asm\_op4 | 7.191 | 37.107 | 84.667 |
|
| asm\_op16 | 10.476 | 39.632 | 63.147 |
|
| ms | | | |

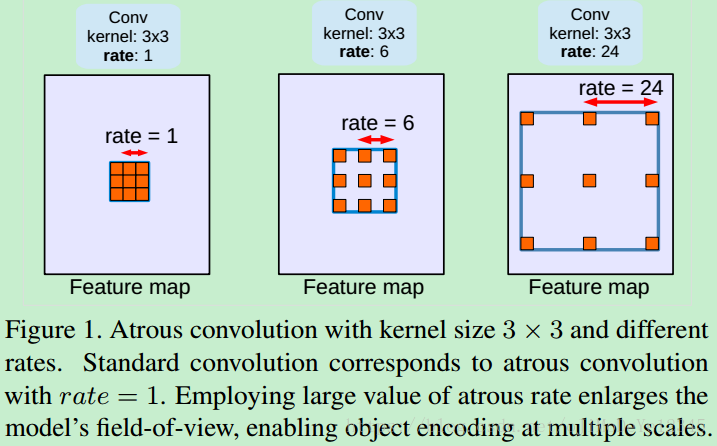
续表：

|  |  |  |  |
| --- | --- | --- | --- |
| C\_op1加速比 | 0.988 | 0.936 | 0.931 |
|
| C\_op4加速比 | 1.811 | 1.571 | 1.728 |
|
| C\_op16加速比 | 2.463 | 2.670 | 3.414 |
|
| asm\_op4加速比 | **7.765** | **6.151** | **7.249** |
|
| asm\_op16加速比 | **5.330** | **5.759** | **9.720** |

解析：可见思路二的优化方法比思路一更好，asm版本仍在展示其优秀性能。

1. 附加实验

通过汇编语言加速空洞卷积计算。



rate = 2; 使用ms

|  |  |  |  |
| --- | --- | --- | --- |
| 尺寸  实现 | 256x256x1 | 640x640x1 | 1280x1280x1 |
|
| C | 4.5 | 19.6 | 70.9 |
|
| asm | 1.1 | 7.6 | 19.8 |
|
| 加速比 | **4.091** | **2.579** | **3.580** |
|

解析：实现了空洞卷积的显著加速。