

推理引擎-Kernel优化

汇编与循环优化



ZOMI



BUILDING A BETTER CONNECTED WORLD

Ascend & MindSpore

www.hiascend.com
www.mindspore.cn

Talk Overview

1. 推理系统介绍：推理系统架构 – 推理引擎架构
2. 模型小型化：CNN小型化结构 – Transform小型化结构
3. 离线优化压缩：低比特量化 – 模型剪枝 – 知识蒸馏
4. 模型转换与优化：模型转换细节 - 计算图优化
5. Kernel 优化
 - 算法优化 (Winograd / Strassen)
 - 内存布局 (NC1HWC0 / NCHW4)
 - 汇编优化 (指令与汇编)
 - 调度优化
6. Runtime 优化

推理引擎架构

- 高性能算子层
- 算子优化
 - 算子执行
 - 算子调度



汇编优化



BUILDING A BETTER CONNECTED WORLD

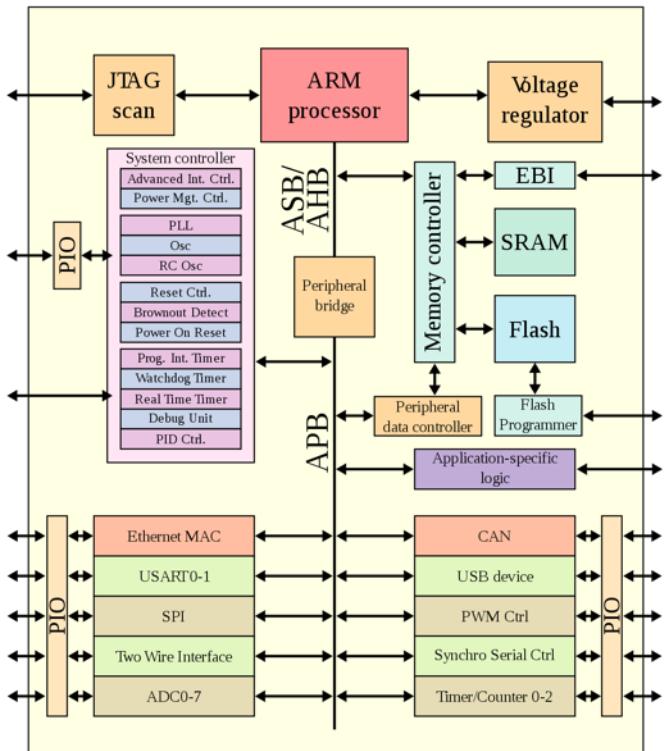
Ascend & MindSpore

4

www.hiascend.com
www.mindspore.cn

汇编优化

- 为了降低读写冗余，很多推理引擎在 CPU 中会利用汇编代码来实现算子对应的 Kernel。而汇编代码中最重要的是实现循环展开，并手排指令减少相邻数据依赖，提升 Kernel 的执行性能。

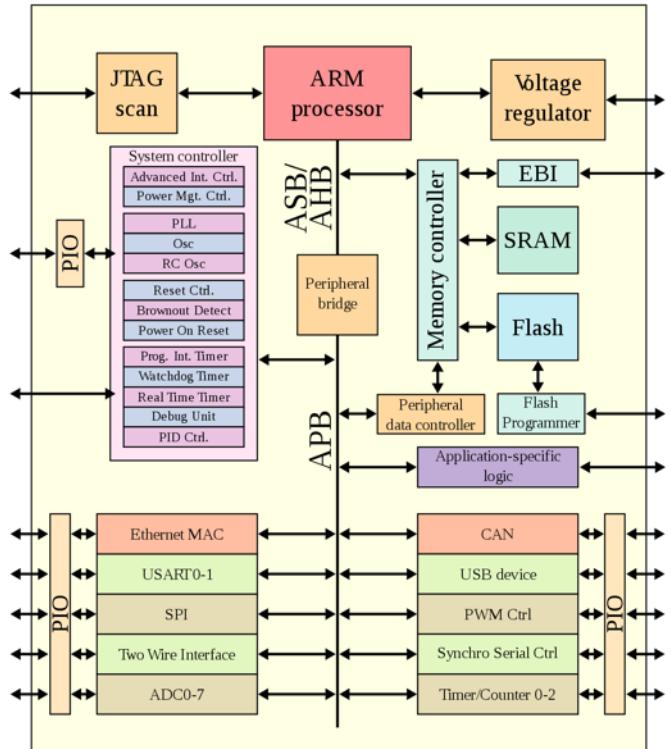


人工手排指令，减少相邻数据依赖，减少总延迟

```
MAIN_TRANSPOSE_E12
1 st1 {v20.8h, v21.8h, v22.8h, v23.8h}, [x0], #64
2 st1 {v24.8h, v25.8h, v26.8h, v27.8h}, [x0], #64
3 st1 {v28.8h, v29.8h, v30.8h, v31.8h}, [x0], #64
4
5
6 add x1, x2, x4
7 sub x5, x5, #8
8 cmp x5, #8
9 bge Body_LoopL8
10
11 cbz x5, Body_LoopLEnd
12 Body_LoopLEextra:
13     MAIN_TRANSPOSE_E12
14     cmp x5, #7 // if x5 < 7
15     blt Body_LoopLEx6 // jump to Body_LoopLEx6
```

汇编优化

- 为了降低读写冗余，很多推理引擎在 CPU 中会利用汇编代码来实现算子对应的 Kernel。而汇编代码中最重要的是实现循环展开，并手排指令减少相依数据依赖，提升 Kernel 的执行性能。



循环优化，按寄存器数量，尽可能一次时钟周期内
计算更多的数据

```
1 .macro MAIN_TRANSPOSE_E12
2 // src: [v0-v11]
3   ld1 {v0.8h}, [x1], x6
4   ld1 {v1.8h}, [x1], x6
5   ld1 {v2.8h}, [x1], x6
6   ld1 {v3.8h}, [x1], x6
7   ld1 {v4.8h}, [x1], x6
8   ld1 {v5.8h}, [x1], x6
9   ld1 {v6.8h}, [x1], x6
10  ld1 {v7.8h}, [x1], x6
11  ld1 {v8.8h}, [x1], x6
12  ld1 {v9.8h}, [x1], x6
13  ld1 {v10.8h}, [x1], x6
14  ld1 {v11.8h}, [x1], x6
15
16 TRANSPOSE_8x8 v20, v12, v23, v13
```

```
1 mov x9, #2 // sizeof(FLOAT16)
2 mul x7, x11, x7
3 mul x8, x9, x8
4 add x0, x0, x7
5 add x0, x0, x8
6 mov x2, #0
7 ldr w2, [x3, #0] // e
8
9 Body:
10 cmp w2, #12 // eP
11 blt E8
12 cmp x5, #8
13 blt Body_LoopLExtra
14 Body_LoopL8:
15     mov x2, x1
```

算子调度优化方法

- 循环展开 (Loop Unrolling)
- 循环分块 (loop tiling)
- 循环重排 (loop Reorder)
- 循环融合 (loop Fusion)
- 循环拆分 (loop Split)
- 向量化 (Vector)
- 张量化 (Tensor)
- 访存延迟 (Latency Hiding)
- 存储分配 (Memory Allocation)



循环优化 (Loop Optimization)

指令优化 (Instructions Optimization)

存储优化 (Memory Optimization)

循环优化

Loop Optimization

循环展开 Loop Unrolling

- 对循环进行展开，以便每次迭代多次使用加载的值，使得一个时钟周期的流水线上尽可能满负荷计算。在流水线中，会因为指令顺序安排不合理而导致NPU等待空转，影响流水线效率。循环展开为编译器进行指令调度带来了更大的空间。

循环展开 Loop Unrolling

```
1
2   for j = 1,2 * n
3     for i = 1,m
4       A(j) = A(j) + B(i)
5     endfor
6   endfor
7
8 # After Unrolling
9
10  for j = 1,2 * n by 2
11    for i = 1,m
12      A(j) = A(j) + B(i)
13      A(j+1) = A(j+1) + B(i)
14    endfor
15  endfor
16
```

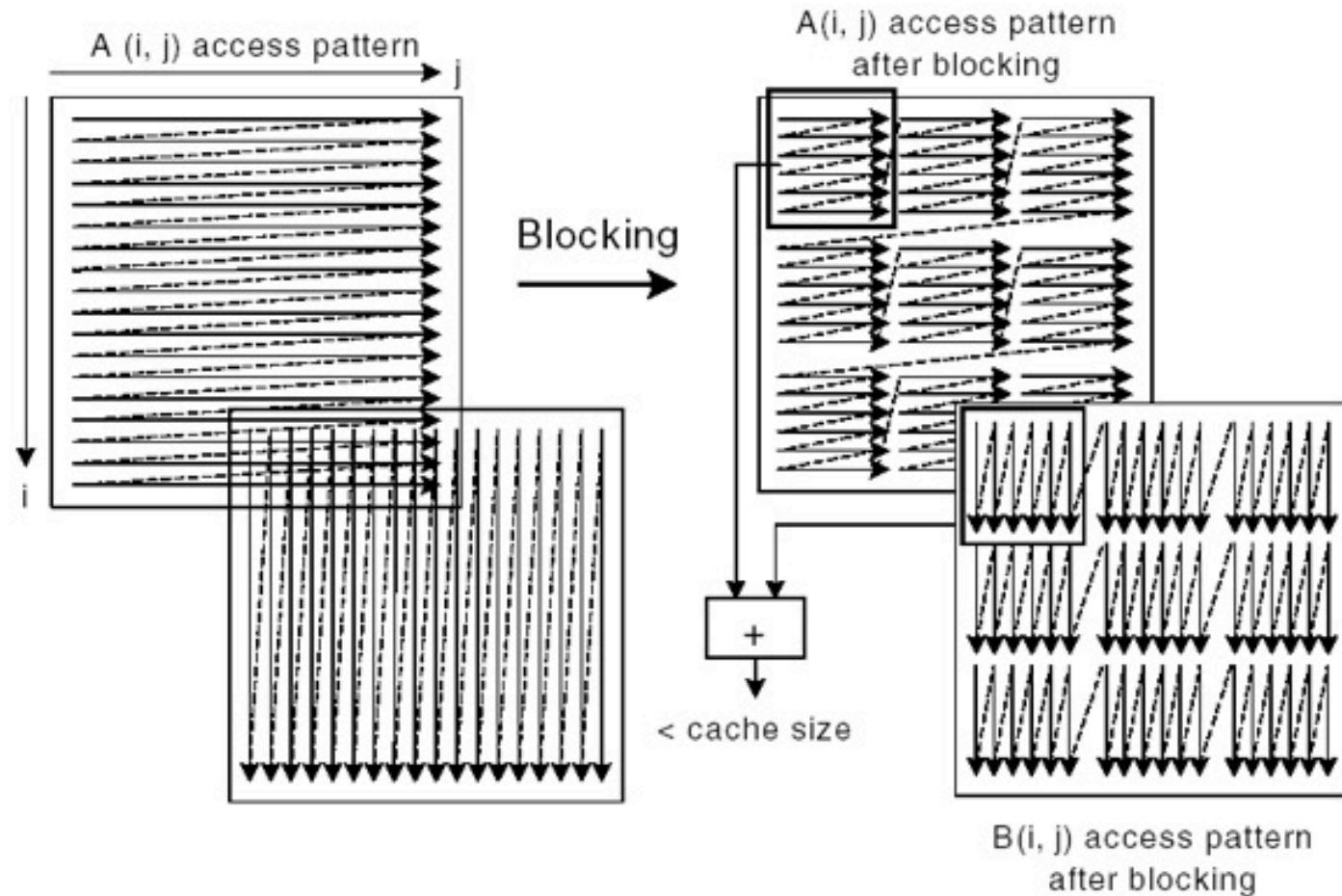
循环分块 Loop tiling

- 由于内存空间有限，代码访问的数据量过大时，无法一次性将所需要的数据加载到设备内存，循环分块能有效提高NPU cache 上的访存效率，改善数据局部性。
- 如果分块应用于外部循环，会增加计算的空间和时间局部性；分块应与缓存块一起作用，可以提高流水线的效率。

循环分块 Loop tiling

- Loop Tiling 的目的是确保一个 Cache 在被用过以后，后面再用的时候其仍然在 cache 中。
- **实现思路**：当一个数组总的数据量无法放在 cache 时，把总数据分成一个个 tile 去访问，令每个 tile 都可以满足 Cache
- **具体做法**：把一层内层循环分成 outer loop * inner loop。然后把 outer loop 移到更外层去，从而确保 inner loop 一定能满足 Cache

循环分块 Loop tiling



循环分块 Loop tiling

```
1
2  ∵ for j = 0, n
3  |   for i = 1, m
4  |   |   A(i) += B(j)
5  |   endfor
6  endfor
7
8  # After Tiling
9
10 ∵ for j_o = 0, m, T:
11 |   for i = 0, n:
12 |   |   for j_i = j_o, j_o + T:
13 |   |   |   A[i] += B[j_i]
14 |   |   endfor
15 |   endfor
16 endfor
17
```

循环重排 Loop Reorder

- 内外层循环重排，改善空间局部性，并最大限度地利用引入缓存的数据。对循环进行重新排序，以最大程度地减少跨步并将访问模式与内存中的数据存储模式对齐。

循环重排 Loop Reorder

```
1
2  for i = 1, n
3      for j = 1, m
4          | A(i,j) = B(i, j) * C(i, j)
5      endfor
6  endfor
7
8 # After Reorder
9
10 for j= 1, m
11     for i = 1, n
12         | A(i, j) = B(i, j) * C(i, j)
13     endfor
14 endfor
15
```

循环融合 Loop Fusion

- 循环融合将相邻或紧密间隔的循环融合在一起，减少的循环开销和增加的计算密度可改善软件流水线，数据结构的缓存局部性增加。

循环融合 Loop Fusion

```
2   for i = 0, n
3     A(i) = a(i) + b(i);
4     c(i) = 2 * a(i);
5   endfor
6   for i = 1, n - 1
7     D(i) = c(i) + a(i);
8   endfor
9
10 # After fusion
11
12 A(0) = a(0) + b(0);
13 c(0) = 2 * a(0);
14 A(n - 1) = a(n - 1) + b(n - 1);
15 c(n - 1) = 2 * a(n - 1);
16 for i = 1, n - 1
17   A(i) = a(i) + b(i)
18   c(i) = 2 * a(i)
19   D(i) = c(i) + a(i)
20 endfor
```

循环拆分 Loop Split

- 拆分主要是将循环分成多个循环，可以在有条件的循环中使用，分为无条件循环和含条件循环
 -

循环拆分 Loop Split

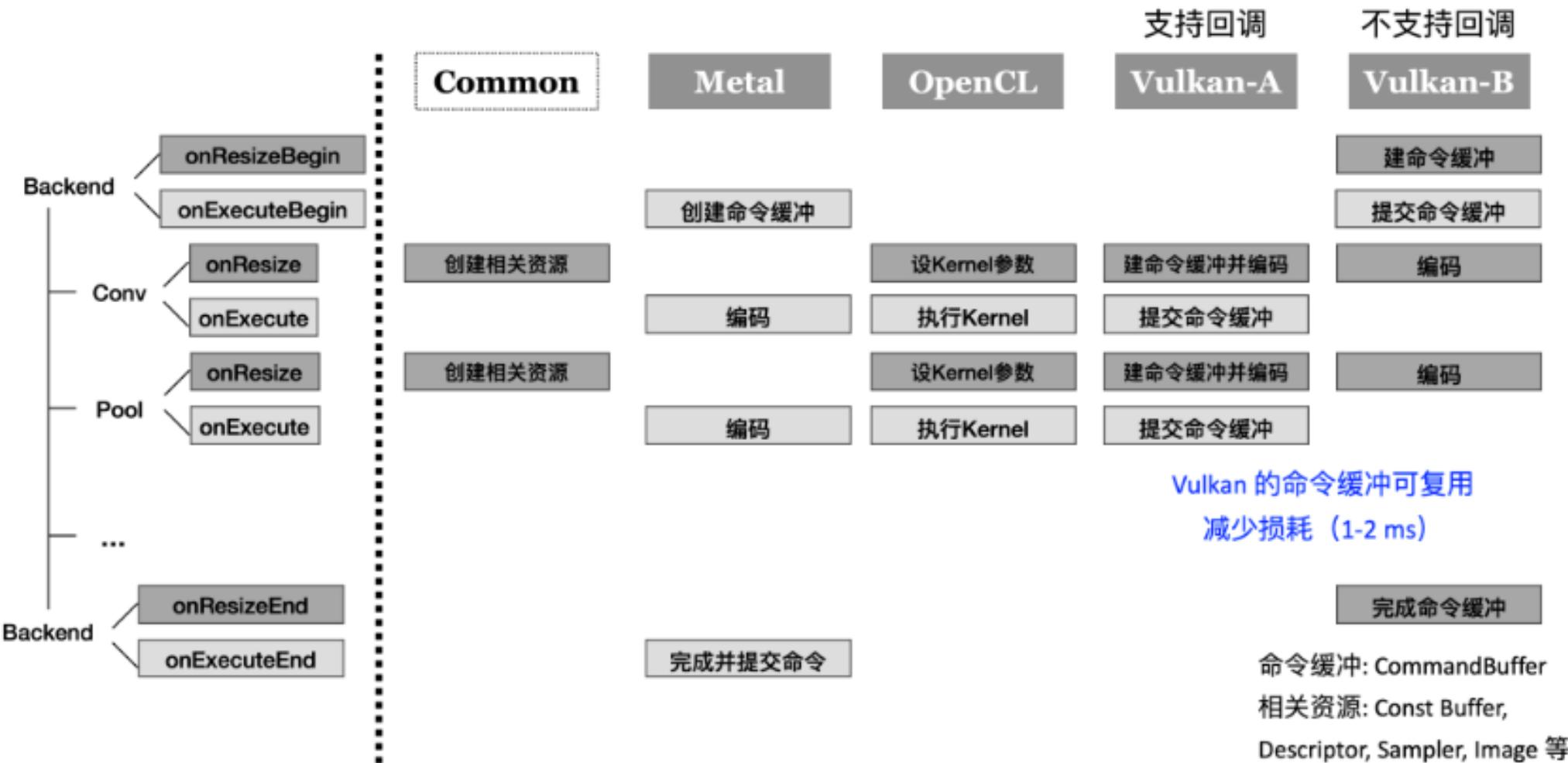
```
2  for i = 0, n
3      A(i) = a(i) + b(i)
4      c(i) = 2 * a(i)
5      if(temp[i] > data)
6          d(i) = a(i)
7  endfor
8
9 # After Split
10
11 for i = 0, n
12     A(i) = a(i) + b(i)
13     c(i) = 2 * a(i)
14 endfor
15 for i = 0, n
16     if(temp[i] > data)
17         d(i) = a(i)
18 endfor
```

调度优化

调度优化

- MNN 的预推理模块可以较好地降低调度冗余，我们把算子的执行拆分为 `onResize` 和 `onExecute` 两个部分，在预推理过程中执行 `onResize`，在推理过程中执行 `onExecute`，视各类 GPU 的 API 设计不同，可以不同程度地降低调度冗余。

调度优化



调度优化总结

1. 对于 OpenCL，可以减少 Kernel 参数设定，将计算资源的申请转移到预推理过程中；
2. 对于 Vulkan，可进一步把命令缓冲的创建全部转移到预推理中，最小化调度冗余；
3. 对于 Metal，可进一步降低命令提交频率；



BUILDING A BETTER CONNECTED WORLD

THANK YOU

Copyright©2014 Huawei Technologies Co., Ltd. All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.