pro2

静态指标：`nums`、`block_size`

结果指标：

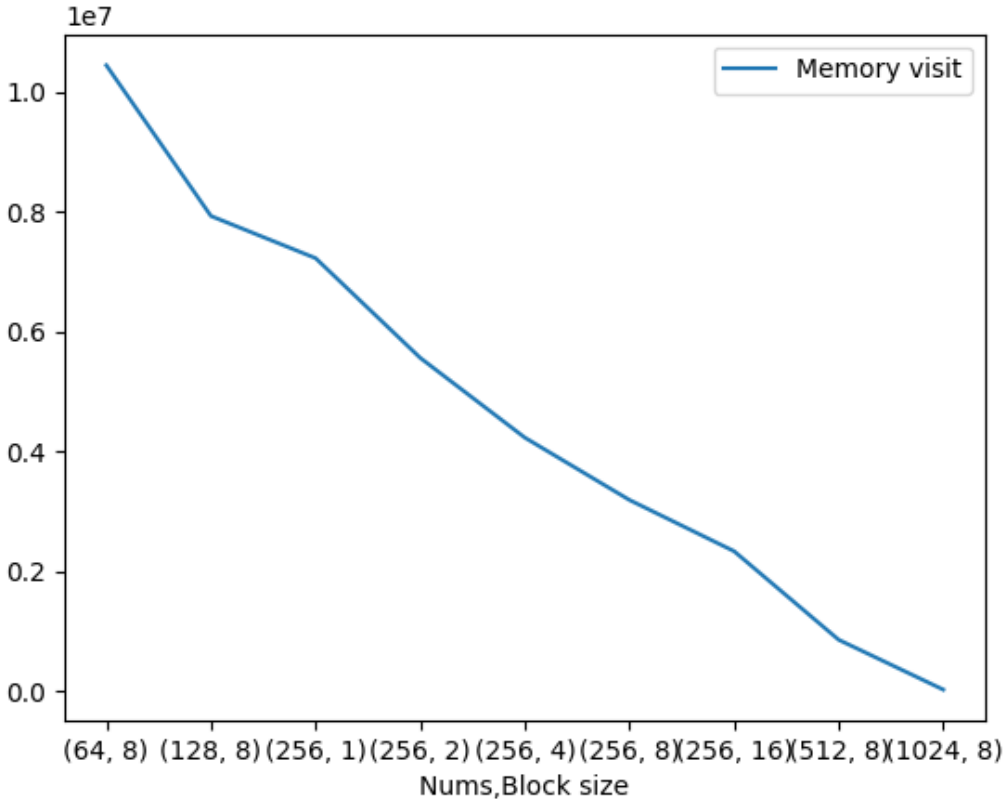
- `memory_visit`：主存访问次数（包括读写，下同）
- `cache_visit`：cache访问次数
- `cache_replace`：cache替换次数
- `cnt[]`：每个地址访问次数

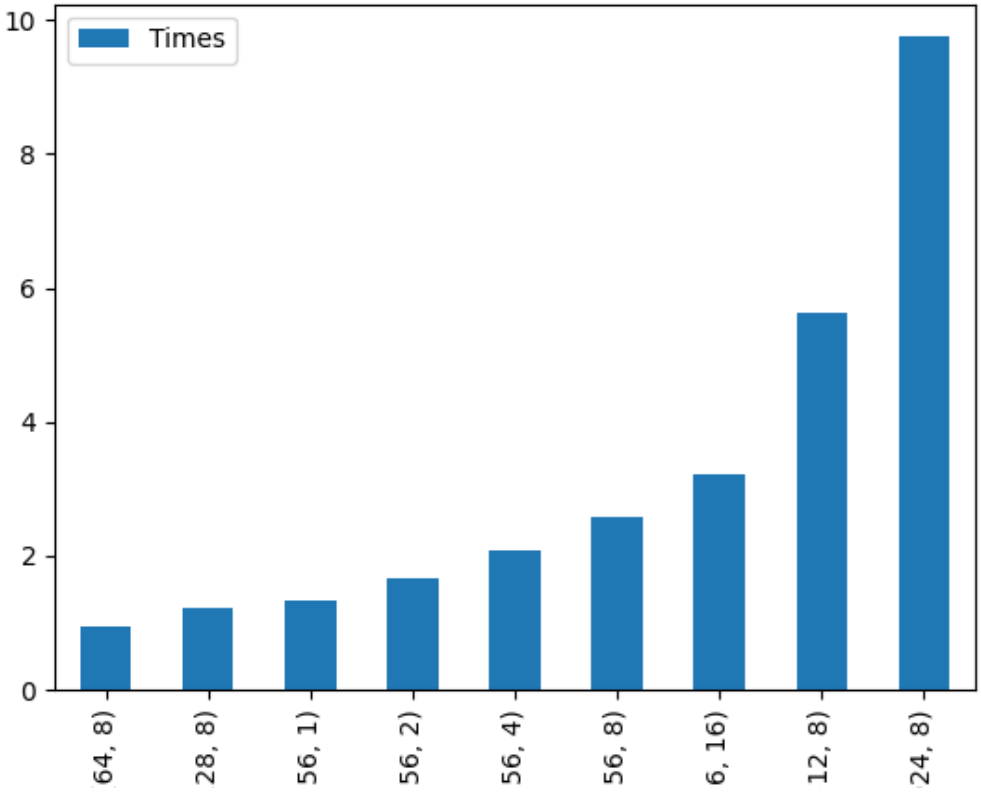
静态指标分析

Nums	Block size	Cache visit	Memory visit	Cache replace	Times
32	8	11059200	11059200	11391412	0.88
64	8	11059200	10452584	5226228	0.96
128	8	11059200	7935498	3967621	1.22
256	8	11059200	3195006	1597247	2.57
512	8	11059200	860738	429857	5.62
1024	8	11059200	28426	13189	9.75
256	1	11059200	7232164	3615826	1.33
256	2	11059200	5567678	2783583	1.66
256	4	11059200	4235946	2117717	2.07
256	8	11059200	3195006	1597247	2.57
256	16	11059200	2339638	1169563	3.21

总体访存结果分析（横坐标：Nums/Block size）：

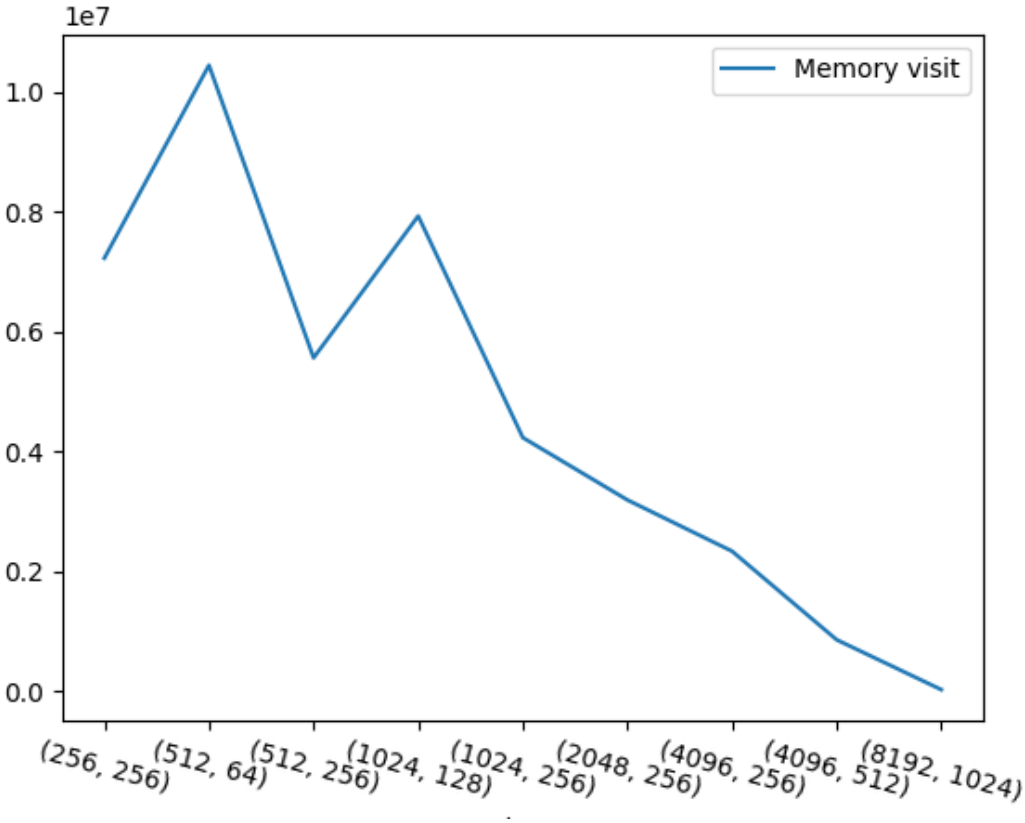
- 在Nums一定时，Memory visit与Block size成反比，加速比与Block size成正比。
- 在Block size一定时，Memory visit与Nums成反比，加速比与Nums成正比。
- Memory visit与加速比成反比，这与计算公式一致。

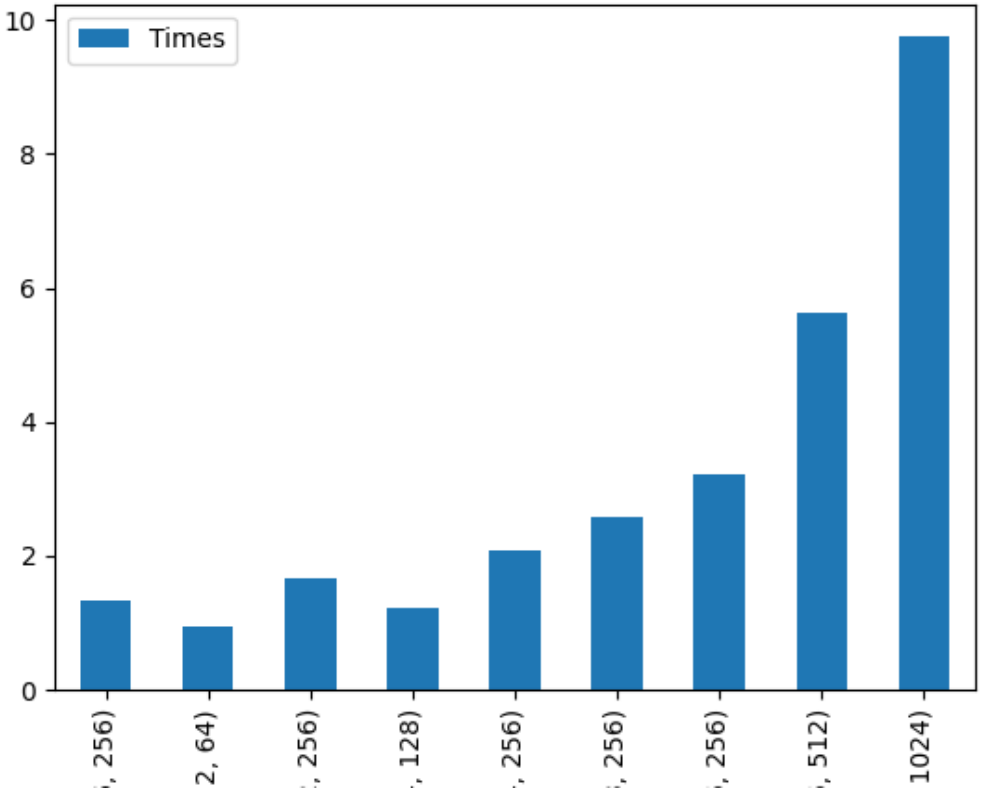




容量（Capacity）一定，结果分析（横坐标：Capacity/Nums）：

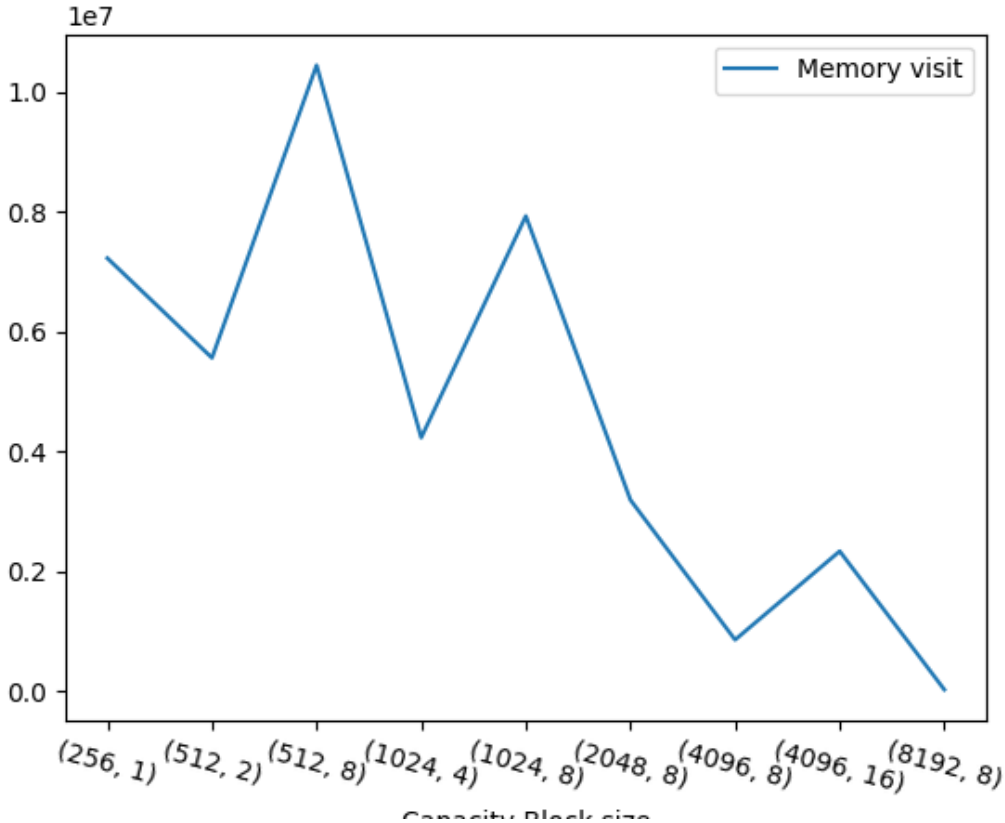
- Cache容量一定时，随着Nums的增加，开始时由于冲突缺失减少，加速比提高。但随着Block size的减少，空间局部性效果减弱，导致加速比降低。

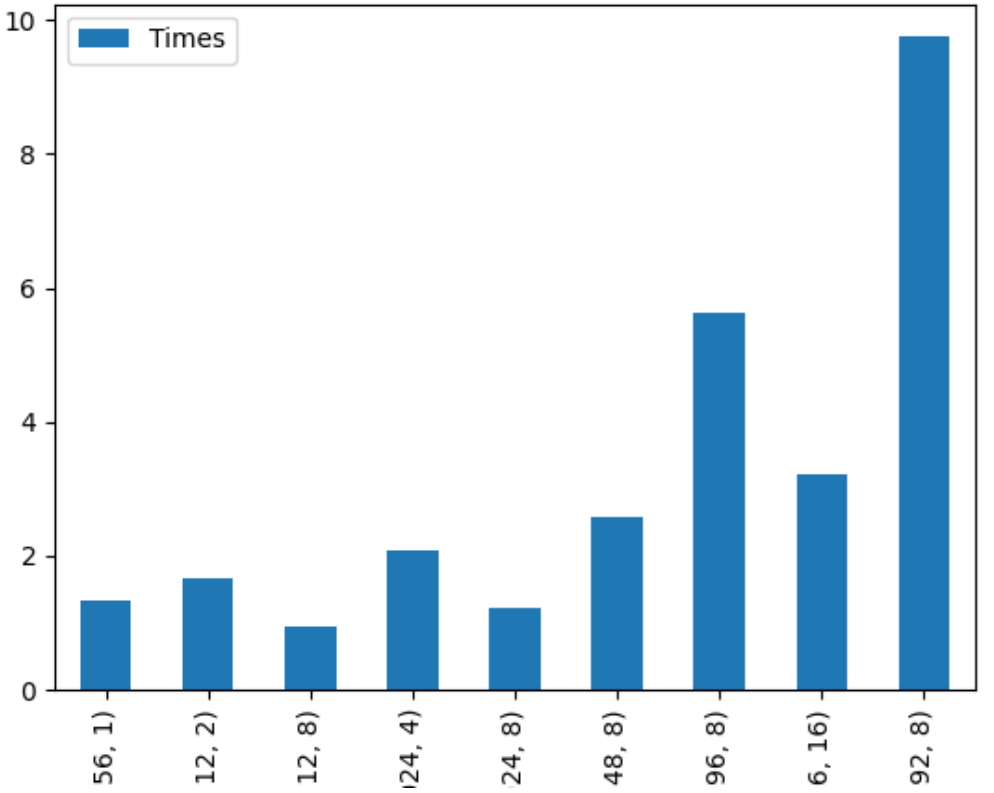




容量（Capacity）一定，结果分析（横坐标：Capacity/Block size）

- Cache容量一定时，随着Block size的增加，开始时由于冲空间局部性增加，加速比提高。但随着Nums的减少，冲突缺失增加，导致加速比降低。





总结：在容量一定时，增加Nums或Block size既会带来部分方面提升，也会导致其他方面的下降。但提升Cache容量可以在只增加成本的同时带来性能的提升。

循环方案

修改前：

```
# 修改前循环
for h in range(r_size):
    for w in range(r_size):
        for H in range(kernel_size):
            for W in range(kernel_size):
                for i in range(in_channels):
                    for j in range(out_channels):
```

```
Pass Correctness Check!
总共访存量为337.5MiB，在这过程中与主存交互字节数778.475MiB，如果不使用cache，共需与主存交互2.637GiB字节数据！
总共访问cache 11059200次，总共访问主存3188632次，假设主存的访问时间为cache的10倍，则整体访存效率提高了2.58倍！
```

修改后：

```
# 修改后循环
for i in range(in_channels):
    for j in range(out_channels):
        for h in range(r_size):
            for w in range(r_size):
                for H in range(kernel_size):
                    for W in range(kernel_size):
```

```
Pass Correctness Check!
总共访存量为337.5MiB，在这过程中与主存交互字节数10.821MiB，如果不使用cache，共需与主存交互2.637GiB字节数据！
总共访问cache 11059200次，总共访问主存44324次，假设主存的访问时间为cache的10倍，则整体访存效率提高了9.61倍！
```

修改思路：因为计算时的空间局部性最大，因此将对卷积的循环放在最内层，发挥Cache的空间局部性。

替换策略

使用FIFO策略：

```
def kickoff():
    """
    当cache满时，需要调用此函数进行替换。目前的替换方式为完全随机替换
    """
    cache_replace[0] += 1
    from random import choice
    # index = choice(list(cache.keys()))
    # 使用FIFO策略
    index = list(cache.keys())[0]
    d = cache.pop(index)
    addr = index * block_size
    write_memory(round_block(addr), d)
```

```
Pass Correctness Check!
总共访存量为337.5MiB，在这过程中与主存交互字节数10.201MiB，如果不使用cache，共需与主存交互2.637GiB字节数据！
总共访问cache 11059200次，总共访问主存41784次，假设主存的访问时间为cache的10倍，则整体访存效率提高了9.64倍！
```