高性能计算导论 第四次小作业报告

管思源 2021012702

一、测试数据处理

以下分别是不使用 Shared Memory 和使用的情况下的卷积代码执行时间、以及它们使用 Shared Memory 的加速比（两者执行时间之商）。结果以热力图的形式呈现，横轴为 X 维度上的 Thread block 大小，纵轴为 Y 维度上的 Thread block 大小，绿色代表代码执行时间越短（加速比越高），红色代表代码执行时间越长（加速比越低）。

	32	64	96	128	160	192	224	256	288	320	352	384	416	448	480	512	544	576	608	640	672	704	736	768	800	832	864	896	928	960	992	1024
1	9.39	7.61	7.62	7.63	7.64	7.66	7.65	7.67	7.66	7.70	7.78	7.73	7.88	7.81	7.75	7.74	7.98	7.93	7.93	7.87	7.82	8.13	8.06	8.07	7.98	8.01	7.96	7.92	7.84	7.91	8.00	7.84
2	7.61	7.63	7.65	7.66	7.70	7.73	7.80	7.76	7.93	7.87	8.11	8.05	8.01	7.90	7.85	7.87																
3	7.63	7.68	7.67	7.74	7.77	7.95	7.82	8.10	7.97	7.93																						
4	7.64	7.69	7.74	7.77	7.84	8.16	7.92	7.92																								
5	7.70	7.74	7.81	7.88	8.14	7.95																										
6	7.71	7.77	8.01	8.30	7.92																											
7	7.69	7.89	7.89	8.00																												
8	7.70	7.81	8.41	7.95																												
9	7.74	8.08	8.02																													
10	7.76	7.97	8.00																													
11	7.85	8.51																														
12	7.77	8.42																														
13	8.01	8.29																														
14	7.86	8.02																														
15	7.83	8.01																														
16	7.79	7.95																														
17	8.20																															
18	8.03																															
19	8.03																															
20	7.93																															
21	7.93																															
22	8.41																															
23	8.37																															
24	8.31																															
25	8.32																															
26	8.23																															
27	7.99																															
28	7.99																															
29	8.00																															
30	7.96																															
31	7.95																															
32	7.91																															

不使用 Shared Memory 的代码执行时间（ms）

	32	64	96	128	160	192	224	256	288	320	352	384	416	448	480	512	544	576	608	640	672	704	736	768	800	832	864	896	928	960	992	1024
1	10.13	6.46	5.84	5.62	5.71	5.67	5.52	5.47	5.49	5.60	5.91	5.63	6.17	5.85	5.64	5.51	6.28	6.06	6.01	5.83	5.64	6.83	6.56	6.55	6.37	6.30	6.28	6.05	5.99	5.98	5.97	5.75
2	6.54	5.51	5.48	5.24	5.40	5.41	5.65	5.26	5.86	5.51	6.55	6.17	5.91	5.66	5.48	5.23																
3	5.83	5.56	5.32	5.51	5.48	5.91	5.38	6.32	5.89	5.42																						
4	5.53	5.32	5.49	5.30	5.55	6.51	5.94	5.35																								
5	5.81	5.60	5.56	5.60	6.39	5.61																										
6	5.73	5.61	6.00	6.56	5.71																											
7	5.59	5.89	5.54	5.93																												
8	5.49	5.41	6.64	5.60																												
9	5.54	6.09	6.09																													
10	5.63	5.70	5.75																													
11	5.98	7.09																														
12	5.65	6.72																														
13	6.22	6.28																														
14	5.89	6.04																														
15	5.69	5.89																														
16	5.50	5.77																														
17	6.35																															
18	6.03																															
19	5.95																															
20	5.77																															
21	5.72																															
22	7.07																															
23	6.81																															
24	6.82																															
25	6.56																															
26	6.49																															
27	6.23																															
28	6.29																															
29	6.00																															
30	6.08																															
31	5.91																															
32	6.00																															

使用 Shared Memory 的代码执行时间（ms）

	32	64	96	128	160	192	224	256	288	320	352	384	416	448	480	512	544	576	608	640	672	704	736	768	800	832	864	896	928	960	992	1024
1	0.93	1.18	1.30	1.36	1.34	1.35	1.38	1.40	1.40	1.38	1.32	1.37	1.28	1.34	1.37	1.41	1.27	1.31	1.32	1.35	1.39	1.19	1.23	1.23	1.25	1.27	1.27	1.31	1.31	1.32	1.34	1.36
2	1.16	1.38	1.40	1.46	1.43	1.43	1.38	1.47	1.35	1.43	1.24	1.31	1.36	1.40	1.43	1.51																
3	1.31	1.38	1.44	1.41	1.42	1.35	1.45	1.28	1.35	1.46																						
4	1.38	1.45	1.41	1.47	1.41	1.25	1.33	1.48																								
5	1.33	1.38	1.40	1.41	1.27	1.42																										
6	1.35	1.38	1.34	1.27	1.39																											
7	1.37	1.34	1.42	1.35																												
8	1.40	1.45	1.27	1.42																												
9	1.40	1.33	1.32																													
10	1.38	1.40	1.39																													
11	1.31	1.20																														
12	1.38	1.25																														
13	1.29	1.32																														
14	1.33	1.33																														
15	1.38	1.36																														
16	1.42	1.38																														
17	1.29																															
18	1.33																															
19	1.35																															
20	1.37																															
21	1.39																															
22	1.19																															
23	1.23																															
24	1.22																															
25	1.27																															
26	1.27																															
27	1.28																															
28	1.27																															
29	1.33																															
30	1.31																															
31	1.34																															
32	1.32																															

使用 Shared Memory 的加速比

二、现象分析讨论

由于对观察到的现象缺乏有力的分析框架，我上网查阅了一些问答，以下是一篇知乎问答，对我的分析很有帮助，特此引用：

与 block 对应的硬件级别为 SM。SM 为同一个 block 中的线程提供通信和同步等所需的硬件资源，跨 SM 不支持对应的通信，所以一个 block 中的所有线程都是执行在同一个 SM 上的。而且因为线程之间可能同步，所以一旦 block 开始在 SM 上执行，block 中的所有线程同时在同一个 SM 中执行（并发，不是并行），也就是说 block 调度到 SM 的过程是原子的。SM 允许多于一个 block 在其上并发执行，如果一个 SM 空闲的资源满足一个 block 的执行，那么这个 block 就可以被立即调度到该 SM 上执行。具体的硬件资源一般包括寄存器、shared memory、以及各种调度相关的资源，这里的调度相关的资源一般会表现为两个具体的限制，Maximum number of resident blocks per SM 和 Maximum number of resident threads per SM ，也就是 SM 上最大同时执行的 block 数量和线程数量……

要达到[同一时间流水线上有足够多的指令]这个目的有多种方法，其中一个最简单的方法是让尽量多的线程同时在 SM 上执行。SM 上并发执行的线程数和 SM 上最大支持的线程数的比值，被称为 Occupancy，更高的 Occupancy 代表潜在更高的性能。显然，一个 kernel 的 block_size 应大于 SM 上最大线程数和最大 block 数量的比值，否则就无法达到 100% 的 Occupancy。对应不同的架构，这个比值不相同，对于 V100 、 A100、 GTX 1080 Ti 是 2048 / 32 = 64，对于 RTX 3090 是 1536 / 16 = 96，所以为了适配主流架构，如果静态设置 block_size 不应小于 96。考虑到 block 调度的原子性，那么 block_size 应为 SM 最大线程数的约数，否则也无法达到 100% 的 Occupancy，主流架构的 GPU 的 SM 最大线程数的公约是 512，96 以上的约数还包括 128 和 256，也就是到目前为止，block_size 的可选值仅剩下 128 / 256 / 512 三个值。

—— OneFlow《如何设置 CUDA Kernel 中的 grid_size 和 block_size？》[1]

对于不使用 Shared Memory 的情况，除了 32×1 的 block size，总体趋势是 block size 越小，执行时间越短，但差异并不显著。结合以上分析框架，我认为 32×1 的 block size 效率较低主要是因为其低于了 SM 上最大线程数和最大 block 数量的比值（conv1 节点为 GTX 1080，该比值为 64），从而未能占满 SM 的全部线程资源。同时，block size 越小意味着将任意 block 调度给某个 SM 对其已用资源情况的要求越少、充分分配 block 后 SM 的资源利用率越高，

[1] 如何设置 CUDA Kernel 中的 grid_size 和 block_size？ - 知乎 (zhihu.com)

从而并行效率更高、执行时间更短。

而对于使用了 Shared Memory 的情况，除了同样观察到 32×1 的 block size 执行时间很长，执行时间较短的 block size 却更多地集中在了 X、Y 维度上 thread 数更多的设置测例上。这一点也可以加速比的热力图中得到印证，即 X、Y 维度上 thread 数更大的在使用了 Shared Memory 后获得了更高的加速比。这主要是因为引入 Shared Memory 后，卷积操作中相邻格子可以共享 2/3 的计算结果，不在边界上的格子（对应 Thread）只需要计算一个格子的运算，大大减少了运算量。而由于 block（SM）之间不互通 memory，因此 thread 数较少的设定下无法充分发挥 Shared Memory 的优势减少运算量。

此外，我还观察到以上三幅图中当 X 维度为 32 或 Y 维度为 1 时，随 block size 增加的执行时间（加速比）并非较为均匀地变化，而是呈现某种规律。阅读代码发现总计算规模为 10000×10000，并不总能被两个维度的 thread 数整除，因此猜测执行时间的变化与此有关。下面计算了两个维度 block 数变化下向上取整导致的最终 thread 总数，与上图吻合较好。

Thread 数	Block 数	总 Thread 数	Thread 数	Block 数	总 Thread 数
1	10000.0	10000	32	312.5	10016
2	5000.0	10000	64	156.3	10048
3	3333.3	10002	96	104.2	10080
4	2500.0	10000	128	78.1	10112
5	2000.0	10000	160	62.5	10080
6	1666.7	10002	192	52.1	10176
7	1428.6	10003	224	44.6	10080
8	1250.0	10000	256	39.1	10240
9	1111.1	10008	288	34.7	10080
10	1000.0	10000	320	31.3	10240
11	909.1	10010	352	28.4	10208
12	833.3	10008	384	26.0	10368
13	769.2	10010	416	24.0	10400
14	714.3	10010	448	22.3	10304
15	666.7	10005	480	20.8	10080
16	625.0	10000	512	19.5	10240

17	588.2	10013	544	18.4	10336
18	555.6	10008	576	17.4	10368
19	526.3	10013	608	16.4	10336
20	500.0	10000	640	15.6	10240
21	476.2	10017	672	14.9	10080
22	454.5	10010	704	14.2	10560
23	434.8	10005	736	13.6	10304
24	416.7	10008	768	13.0	10752
25	400.0	10000	800	12.5	10400
26	384.6	10010	832	12.0	10816
27	370.4	10017	864	11.6	10368
28	357.1	10024	896	11.2	10752
29	344.8	10005	928	10.8	10208
30	333.3	10020	960	10.4	10560
31	322.6	10013	992	10.1	10912
32	312.5	10016	1024	9.8	10240

下面回答助教给出的思考题：

1. 对于这个程序，如何设置 thread block size 才可以达到最好的效果？为什么？

   答：就结果而言，应当选择 128×2 的 block size 并启用 Shared Memory。原因主要来自占满了 SM 的线程数、block 的调度较为轻量、以及 Shared Memory 在卷积操作中的优势。

2. 对于这个程序，Shared memory 总是带来优化吗？如果不是，为什么？

   答：不一定。当创建和维护 Shared Memory 的开销大于其带来的计算量的减少时，Shared Memory 反而会造成性能损失。例如在 block size 为 32×1 时，加速比 0.93 就小于 1。

3. 对于这个程序，Shared memory 在什么 thread block size 下有效果，什么时候没有？

   答：对此程序而言，应当是 X、Y 维度的 thread 数越大，Shared memory 越有效果；X、Y 维度的 thread 数越小，Shared memory 的效果越差。测试结果基本符合这一结论，但是差异并不显著，且在 X、Y 维度的 thread 数均较大的情况下，加速比反而减小了。我认为这个现象的主要原因是 Shared Memory 本身较大使得 block 调度的难度加大，SM 的利用率降低，影响了并行效率。

4. 对于这个程序，还有哪些可以优化的地方？

答：我暂时没有想到优化的地方。不过我对测试代码中 block size 的 X 维度为什么以 32 为单位有一些疑惑，于是尝试采集了 X 维度为 1~32 的数据，结果如下：

（从上至下分别为不使用 Shared Memory 的代码执行时间、使用 Shared Memory 的代码执行时间、和两者相比的加速比，热力图格式与先前规定的一致）

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	261	133	90	70	56	48	41	37	33	30	27	25	23	21	20	19	17	16	15	14	14	13	12	12	11	11	11	10	10	9.7	9.4	9.2
2	135	70	48	37	29	25	21	19	17	15	14	13	12	11	10	9.4	14	14	13	12	12	11	11	10	9.7	9.4	9	8.7	8.4	8.1	7.9	7.6
3	93	48	33	25	20	17	15	13	11	10	15	14	12	12	11	10	9.6	9	8.5	8.1	7.7	11	11	10	9.7	9.4	9	8.7	8.4	8.1	7.9	7.6
4	70	37	25	19	16	13	11	10	14	12	11	10	9.4	8.7	8.1	7.6	11	10	9.6	9.2	8.7	8.3	8	7.6	9.8	9.4	9.1	8.7	8.4	8.2	7.9	7.6
5	57	30	21	16	13	11	14	12	11	9.8	8.9	8.1	11	10	9.8	9.2	8.6	8.2	7.7	9.8	9.3	8.9	8.5	8.2	7.8	9.4	9.1	8.8	8.5	8.2	7.9	7.7
6	48	25	17	13	11	14	12	10	9	8.1	11	10	9.4	8.7	8.2	7.6	9.6	9.1	8.6	8.2	7.8	9.3	8.9	8.5	8.2	7.9	9.1	8.8	8.5	8.2	7.9	7.7
7	41	22	15	12	14	12	10	8.7	7.8	11	9.6	8.8	8.1	10	9.3	8.8	8.2	7.8	9.3	8.8	8.4	8	9.2	8.8	8.5	8.2	7.8	8.8	8.5	8.2	7.9	7.7
8	37	19	14	10	12	10	8.7	7.6	10	9.2	8.4	7.7	9.4	8.8	8.2	7.7	9.1	8.6	8.1	7.7	8.8	8.4	8.1	7.7	8.6	8.3	8	7.7	8.5	8.2	8	7.7
9	33	17	13	14	11	9	7.7	10	9.1	8.2	9.9	9.1	8.4	7.8	9.1	8.6	8.1	9.2	8.7	8.2	7.9	8.7	8.3	8	8.7	8.4	8.1	7.8	8.5	8.3	8	7.7
10	31	16	14	12	9.8	8.1	10	9.2	8.2	9.8	8.9	8.2	9.5	8.8	8.2	7.7	8.7	8.2	7.8	8.6	8.2	7.8	8.6	8.2	7.9	8.5	8.2	8	8.5	8.3	8	7.8
11	29	17	15	11	9.1	11	9.5	8.4	9.9	8.9	8.1	9.4	8.6	8	9	8.4	7.9	8.7	8.3	7.8	8.5	8.2	7.8	8.4	8.1	7.8	8.4	8.1	7.8	8.4	8.1	7.9
12	28	18	14	10	9.9	10	8.7	7.7	9.1	8.2	9.3	8.6	7.9	8.8	8.2	7.7	8.5	8	8.6	8.2	7.8	8.4	8.1	7.7	8.3	8	8.5	8.3	8	8.3	8	7.8
13	30	17	13	9.4	11	9.4	8.1	9.4	8.4	9.5	8.6	7.9	8.8	8.2	8.8	8.3	7.8	8.4	8	8.5	8.2	7.8	8.3	8	8.5	8.2	7.9	8.2	7.9	8.6	8.3	8
14	32	17	13	9.2	10	8.7	10	8.7	7.8	8.8	8	8.8	8.1	8.8	8.2	7.7	8.3	7.8	8.4	7.9	8.5	8.1	8.6	8.3	7.9	8.2	7.9	8.6	8.3	8.4	8.1	7.9
15	32	16	14	9.6	9.8	8.1	9.3	8.2	9.1	8.2	9	8.2	8.8	8.2	8.7	8.2	7.7	8.2	7.8	8.3	7.9	8.4	8	8.3	7.9	8.6	8.3	8.4	8.1	8.4	8.1	7.8
16	33	16	14	9.8	9.2	7.8	8.7	7.7	8.5	7.7	8.4	7.7	8.3	7.7	8.2	7.7	8.2	7.7	8.2	7.8	8.3	7.9	8.1	7.8	8.4	8.1	8.2	7.9	8.1	7.8	8	7.8
17	32	16	14	11	9	9.6	8.2	9	8	8.7	7.9	8.4	7.8	8.3	7.7	8.2	8.6	8.1	8.6	8.1	8.4	8	8.6	8.2	8.3	8	8.2	7.9	8.1	7.8	8.5	8.2
18	32	17	14	10	9.5	9	7.8	8.5	9.1	8.2	8.7	8	8.4	7.8	8.2	7.7	8.1	8.6	8.1	8.3	7.9	8.5	8.1	8.2	8.3	8	8.2	7.9	8.5	8.3	8.3	8
19	32	17	14	9.6	9.8	8.6	9.2	8.1	8.6	7.8	8.2	8.6	8	8.4	7.8	8.2	8.6	8.1	8.3	7.9	8.4	8.4	8.1	8.2	7.9	8.1	8.6	8.4	8.4	8.1	8.3	8
20	32	17	13	9.2	9.8	8.1	8.8	7.7	8.2	8.6	7.8	8.2	8.5	7.9	8.3	7.8	8.1	8.3	7.9	8.4	8.4	8	8.2	7.8	8	8.6	8.3	8.3	8.4	8.1	8.2	7.9
21	31	17	13	9	9.3	7.9	8.3	8.8	7.8	8.2	8.5	7.8	8.1	8.4	7.9	8.2	8.4	7.9	8.4	8.4	8	8.2	8.2	7.9	8.5	8.5	8.2	8.3	8.3	8.1	8.2	7.9
22	32	17	13	9.1	9.1	9.3	8	8.4	8.7	7.8	8.1	8.4	7.8	8.1	8.4	7.9	8	8.5	8.4	8	8.1	8.2	7.9	8.4	8.4	8.1	8.2	8.3	8	8.1	8.8	8.4
23	32	17	13	9.2	9.4	8.9	9.2	8	8.3	8.5	7.8	8	8.3	8.6	8	8.1	8.6	8.1	8.1	8.2	8.2	7.9	8.4	8.4	8	8.2	8.2	8.2	8	8.7	8.7	8.4
24	32	17	13	9.3	9.8	8.5	8.8	7.7	8	8.2	8.4	7.7	7.9	8.2	8.3	7.8	8.2	8.2	8.3	7.8	7.9	8.4	8.4	8	8.1	8.2	8.2	7.9	8.6	8.6	8.6	8.3
25	32	17	13	9.8	9.9	8.2	8.4	8.6	8.7	7.9	8.1	8.3	8.5	7.9	8	8.4	8.3	8.4	7.9	8	8.5	8.4	8.1	8.1	8.1	8.2	8.8	8.5	8.6	8.6	8.6	8.3
26	31	17	13	9.5	9.4	7.9	8.1	8.2	8.4	8.5	7.8	7.9	8.2	8.2	8.6	8	8	8	8.1	8.6	8.5	8.1	8.2	8.2	8.2	8.8	8.5	8.5	8.6	8.6	8.5	8.2
27	31	17	13	9.3	9.1	9.1	7.8	7.9	8.1	8.2	8.3	8.5	7.9	7.9	8.3	8.2	8.2	8.2	8.7	8.3	8.2	8.2	8.2	8.2	8.8	8.5	8.5	8.5	8.6	8.5	8.2	8
28	31	16	13	9.2	8.9	8.8	8.8	7.7	7.8	7.9	8	8.2	8.2	8.6	8.4	7.9	7.9	7.9	8.4	8.3	8.3	8.3	8.2	7.9	8.5	8.5	8.5	8.6	8.5	8.2	8.3	8
29	31	16	13	9.2	8.8	8.5	8.4	8.4	8.5	8.5	7.8	7.9	7.9	8.3	8.1	8.1	8.1	8.6	8.4	8.4	8.4	8	8	8.5	8.6	8.5	8.6	8.5	8.2	8.2	8.2	8
30	31	16	13	9.2	8.9	8.2	8.2	8.2	8.2	8.2	8.4	8.3	8.6	8.4	8.4	7.8	7.8	8.3	8.1	8.1	8.1	8.1	8.7	8.6	8.6	8.6	8.5	8.2	8.2	8.2	8.2	8
31	31	16	13	9.3	9	8	7.9	7.9	7.9	8	8.1	8	8.3	8.1	8.1	8	8.5	8.3	8.3	8.2	8.2	8.8	8.7	8.6	8.6	8.5	8.3	8.3	8.2	8.2	8.2	7.9
32	31	16	13	9.3	9	7.8	7.7	7.7	7.7	7.7	7.9	7.8	8.1	7.9	7.9	7.8	8.2	8.1	8	7.9	8	8.4	8.4	8.3	8.4	8.3	8	8	8	8	8	7.9

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	290	146	97	73	59	49	42	37	34	30	28	25	23	22	20	19	18	17	16	16	15	14	14	13	13	12	12	11	11	11	10	10
2	146	75	51	39	31	26	23	20	18	16	15	14	13	12	11	10	13	13	12	11	11	10	9.9	9.5	9.1	8.7	8.4	8.1	7.8	7.6	7.3	6.5
3	101	52	35	27	21	18	15	14	12	11	14	13	12	11	10	8.6	8.1	7.6	7.2	6.8	6.5	9.1	8.7	8.3	8	7.6	7.4	7.1	6.8	6.6	6.4	5.8
4	77	40	27	20	16	14	12	10	13	11	9.6	8.7	8	7.4	6.9	6.4	9.1	8.6	8.2	7.7	7.3	6.5	6.2	5.9	7.7	7.5	7.2	6.9	6.6	6.3	6	5.5
5	62	32	22	16	13	11	13	10	9.3	8.5	7.7	7	9.5	8.9	8.3	7.4	7	6.6	6.3	7.8	7.5	7.1	6.8	6.2	6	7.3	7.1	6.9	6.7	6.4	6.2	5.8
6	52	27	18	14	11	13	10	8.8	7.8	7.1	9.3	8.7	7.6	7.1	6.6	6.3	7.7	7.3	6.9	6.2	6	7.2	7	6.7	6.4	6	6.9	6.7	6.5	6.3	6.1	5.7
7	45	23	16	12	13	9.9	8.6	7.5	6.8	8.9	7.7	7.1	6.6	8	7.5	6.7	6.3	6	7.2	6.9	6.6	6.1	6.9	6.7	6.5	6.2	5.9	6.5	6.4	6.2	6	5.6
8	40	20	14	10	10	8.7	7.5	6.6	8.7	7.5	6.8	6.3	7.6	6.7	6.3	5.9	7.1	6.7	6.2	5.9	6.8	6.5	6	5.8	6.4	6.2	6	5.7	6.2	6.1	5.9	5.5
9	35	18	12	12	9.3	7.8	6.7	8.2	7.4	6.7	7.9	6.9	6.5	6.1	7.1	6.5	6.2	7	6.7	6.1	5.9	6.5	6.3	5.8	6.4	6.2	6	5.7	6.2	6.1	5.9	5.5
10	32	16	11	10	8.4	7	8.8	7.4	6.7	7.8	6.9	6.3	7.4	6.9	6.3	5.9	6.7	6.2	5.9	6.4	6.2	5.8	6.3	6.1	5.7	6.3	6.1	5.9	6.3	6.2	6	5.6
11	29	15	14	9.5	7.7	9.3	7.7	6.8	7.9	6.9	6.3	7.3	6.6	6.2	6.9	6.3	6	6.5	6.3	5.8	6.3	6	5.7	6.2	6	5.7	6.3	6.1	5.8	6.5	6.3	6
12	27	14	11	8.7	7.1	8.2	7.1	6.3	7	6.4	7.3	6.5	6.1	6.8	6.2	5.8	6.4	5.9	6.4	6.1	5.7	6.2	6	5.6	6.2	6	6.6	6.4	6.2	6.2	6	5.6
13	25	13	11	8	9.4	7.6	6.7	7.2	6.5	7.4	6.6	6.1	6.8	6.2	6.6	6.1	5.8	6.3	5.8	6.3	6.1	5.7	6.2	5.9	6.6	6.4	6.1	6.2	6	6.7	6.6	6.2
14	23	12	9.9	7.5	8.4	7.1	7.9	6.7	6.1	6.8	6.2	6.8	6.2	6.6	6.1	5.7	6.2	5.7	6.2	5.8	6.3	6.1	6.6	6.4	6.1	6.2	5.9	6.7	6.5	6.4	6.3	5.9
15	22	11	9.2	7	7.9	6.7	7.2	6.3	7.2	6.3	6.9	6.2	6.6	6.1	6.5	5.9	5.7	6.2	5.8	6.2	5.9	6.5	6.2	6.2	6	6.7	6.5	6.4	6.3	6.2	6.2	5.7
16	20	10	8.7	6.6	7.5	6.4	6.8	6	6.6	6	6.4	5.9	6.2	5.8	6	5.6	6.2	5.7	6.2	5.8	6.5	6.1	6.1	5.8	6.6	6.3	6.4	6	6.2	5.9	6	5.5
17	19	12	8.2	8.6	7.1	7.3	6.4	6.9	6.3	6.6	6.1	6.3	5.9	6.1	5.7	6	6.4	6	6.6	6.3	6.3	5.9	6.7	6.4	6.4	6.1	6.2	5.9	6	5.7	6.8	6.3
18	18	11	7.8	8.1	6.8	7	6.1	6.6	7	6.3	6.6	5.9	6.3	5.8	6.2	5.7	6.1	6.6	6.3	6.2	5.9	6.7	6.3	6.3	6.3	6.1	6.1	5.9	6.8	6.6	6.6	6
19	17	11	7.5	7.8	6.5	6.7	7.3	6.3	6.6	6	6.2	6.4	5.9	6.3	5.8	6	6.6	6.3	6.3	5.8	6.7	6.5	6.3	6.2	5.9	6	6.8	6.7	6.6	6.4	6.5	6
20	17	10	7.2	7.4	7.8	6.4	6.8	6	6.3	6.6	5.9	6	6.4	5.9	6.3	5.8	6.3	6.3	5.9	6.6	6.5	6.1	6.2	5.9	6	6.8	6.6	6.5	6.5	6.4	6.3	5.8
21	16	9.8	6.9	7.1	7.2	6.1	6.5	6.6	6	6.1	6.4	5.8	6	6.4	5.9	6.4	6.3	5.9	6.7	6.4	6.2	6.2	6.1	5.8	6.8	6.6	6.5	6.5	6.3	6.2	6.2	5.7
22	15	9.4	9.2	6.8	6.9	7.3	6.3	6.4	6.6	5.9	6	6.3	5.8	6	6.6	6.1	6	6.7	6.5	6.1	6.2	6.1	5.8	6.7	6.6	6.3	6.5	6.3	6.1	6.2	7.5	7.1
23	15	9	8.5	6.5	6.7	6.8	7.1	6.1	6.2	6.4	5.8	5.9	6.3	6.6	6.3	6	6.8	6.3	6.2	6.2	6.1	5.8	6.8	6.5	6.3	6.4	6.3	6.2	6.1	7.4	7.3	6.8
24	15	8.7	8.2	6.3	6.4	6.6	6.7	5.9	6	6	6.3	5.7	6	6.4	6.3	5.8	6.5	6.3	6.3	5.9	5.9	6.7	6.6	6.2	6.4	6.2	6.2	6	7.4	7.3	7.2	6.8
25	14	8.4	7.9	7.4	6.3	6.4	6.4	6.4	6.5	5.9	6	6.2	6.6	6.2	6	6.5	6.5	6.3	6	5.8	6.9	6.6	6.3	6.3	6.3	6.2	7.5	7.3	7.4	7.1	7	6.6
26	14	8.1	7.7	7.1	7.4	6.2	6.2	6.1	6.2	6.4	5.8	5.9	6.4	6.2	6.8	6.3	6.2	6	6.1	6.8	6.7	6.3	6.5	6.2	6.3	7.4	7.4	7.2	7.1	7	6.9	6.5
27	14	7.9	7.5	6.9	7	7	6.1	6	6	6.1	6.3	6.6	6.2	6	6.5	6.2	6.3	6.1	6.8	6.5	6.4	6.5	6.3	6.2	7.5	7.3	7.3	7	7.1	6.8	6.8	6.2
28	14	7.9	7.4	6.7	6.8	6.7	6.6	5.8	5.8	5.9	6.1	6.4	6.3	6.8	6.5	6	6	5.9	6.7	6.5	6.6	6.3	6.2	6	7.4	7.2	7.2	7	6.9	6.8	6.7	6.3
29	14	8.2	7.2	6.5	6.6	6.5	6.3	6.2	6.4	6.4	5.9	6.2	6	6.5	6.3	6	6.1	6.8	6.7	6.4	6.4	6.1	6.1	7.4	7.4	7.1	7.1	6.8	6.8	6.7	6.6	6
30	14	8.1	7	6.3	6.5	6.3	6.2	6	6.1	6.2	6.6	6.3	6.8	6.5	6.4	5.9	5.8	6.6	6.4	6.3	6.2	6.2	7.5	7.3	7.2	7	6.9	6.7	6.8	6.6	6.6	6.1
31	14	7.9	6.9	6.2	6.3	6.1	6	5.9	6	6	6.3	6	6.6	6.3	6.2	5.9	6.9	6.6	6.6	6.3	6.3	7.4	7.4	7.1	7.1	6.9	6.8	6.6	6.6	6.6	6.4	5.9
32	14	7.9	6.7	6	6.2	6	5.9	5.7	5.8	5.9	6.2	5.8	6.4	6.1	6	5.7	6.6	6.3	6.3	6	6.1	7.3	7.3	7	7	6.8	6.7	6.6	6.6	6.5	6.4	6

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	0.90	0.91	0.93	0.96	0.96	0.98	0.98	0.98	0.98	0.98	0.97	0.97	0.97	0.97	0.97	0.97	0.96	0.93	0.92	0.91	0.91	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.91
2	0.92	0.93	0.94	0.94	0.94	0.94	0.94	0.94	0.93	0.92	0.92	0.93	0.92	0.92	0.92	0.91	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.16
3	0.92	0.93	0.94	0.94	0.94	0.94	0.95	0.95	0.94	0.93	1.07	1.07	1.07	1.07	1.07	1.18	1.19	1.19	1.19	1.18	1.18	1.22	1.22	1.22	1.22	1.23	1.23	1.23	1.23	1.23	1.23	1.31
4	0.92	0.92	0.94	0.95	0.96	0.96	0.97	0.97	1.07	1.07	1.16	1.17	1.17	1.17	1.18	1.18	1.19	1.18	1.18	1.19	1.20	1.27	1.29	1.30	1.26	1.26	1.26	1.27	1.28	1.29	1.30	1.38
5	0.92	0.93	0.95	0.96	0.97	0.97	1.07	1.17	1.16	1.15	1.15	1.16	1.19	1.18	1.17	1.24	1.23	1.23	1.22	1.26	1.25	1.25	1.24	1.30	1.29	1.30	1.29	1.28	1.27	1.27	1.27	1.33
6	0.92	0.94	0.96	0.97	0.98	1.07	1.17	1.16	1.15	1.15	1.19	1.18	1.24	1.23	1.23	1.22	1.26	1.25	1.24	1.31	1.29	1.30	1.28	1.28	1.27	1.31	1.32	1.31	1.30	1.30	1.30	1.35
7	0.92	0.95	0.98	0.98	1.07	1.17	1.16	1.16	1.15	1.18	1.24	1.23	1.22	1.26	1.25	1.31	1.30	1.29	1.29	1.28	1.28	1.32	1.33	1.31	1.31	1.30	1.33	1.35	1.34	1.33	1.33	1.38
8	0.92	0.96	1.00	0.99	1.17	1.17	1.16	1.15	1.18	1.23	1.23	1.22	1.25	1.30	1.30	1.30	1.28	1.28	1.31	1.31	1.31	1.31	1.34	1.33	1.34	1.33	1.33	1.36	1.36	1.35	1.35	1.40
9	0.93	0.97	1.03	1.18	1.17	1.16	1.15	1.24	1.23	1.22	1.25	1.31	1.30	1.29	1.28	1.32	1.31	1.32	1.30	1.34	1.33	1.34	1.33	1.37	1.37	1.35	1.35	1.38	1.36	1.34	1.34	1.39
10	0.95	1.00	1.26	1.18	1.16	1.15	1.19	1.23	1.22	1.25	1.30	1.29	1.29	1.27	1.30	1.30	1.30	1.33	1.32	1.34	1.32	1.36	1.36	1.35	1.38	1.36	1.34	1.34	1.35	1.33	1.33	1.38
11	0.98	1.12	1.07	1.17	1.18	1.20	1.23	1.22	1.25	1.30	1.29	1.28	1.31	1.30	1.30	1.34	1.32	1.34	1.32	1.35	1.35	1.35	1.37	1.35	1.34	1.37	1.33	1.33	1.35	1.29	1.28	1.31
12	1.06	1.29	1.18	1.17	1.40	1.24	1.22	1.21	1.30	1.29	1.28	1.31	1.29	1.30	1.32	1.32	1.32	1.35	1.35	1.35	1.37	1.35	1.34	1.37	1.33	1.33	1.30	1.28	1.28	1.33	1.33	1.38
13	1.21	1.35	1.18	1.17	1.20	1.23	1.21	1.31	1.29	1.28	1.30	1.29	1.30	1.32	1.33	1.36	1.34	1.34	1.37	1.36	1.33	1.36	1.33	1.35	1.30	1.28	1.29	1.33	1.33	1.28	1.27	1.29
14	1.37	1.41	1.29	1.23	1.24	1.22	1.26	1.30	1.27	1.30	1.29	1.30	1.32	1.33	1.34	1.35	1.34	1.36	1.34	1.37	1.33	1.33	1.31	1.28	1.29	1.32	1.33	1.28	1.27	1.31	1.29	1.33
15	1.50	1.44	1.50	1.38	1.23	1.21	1.30	1.29	1.27	1.29	1.30	1.32	1.33	1.34	1.35	1.38	1.36	1.33	1.35	1.33	1.34	1.29	1.29	1.33	1.33	1.28	1.27	1.31	1.29	1.34	1.32	1.38
16	1.61	1.52	1.57	1.49	1.23	1.22	1.29	1.28	1.30	1.28	1.32	1.31	1.34	1.33	1.36	1.37	1.33	1.35	1.32	1.34	1.28	1.29	1.32	1.35	1.27	1.28	1.28	1.31	1.31	1.34	1.34	1.42
17	1.66	1.30	1.66	1.25	1.27	1.30	1.28	1.31	1.28	1.32	1.30	1.35	1.33	1.36	1.35	1.37	1.34	1.34	1.30	1.30	1.33	1.35	1.27	1.28	1.29	1.31	1.31	1.34	1.34	1.37	1.25	1.29
18	1.72	1.50	1.75	1.25	1.41	1.29	1.26	1.30	1.30	1.31	1.32	1.34	1.33	1.35	1.32	1.36	1.34	1.30	1.29	1.33	1.34	1.27	1.28	1.29	1.33	1.31	1.35	1.34	1.26	1.26	1.27	1.33
19	1.81	1.58	1.82	1.24	1.51	1.28	1.27	1.29	1.31	1.30	1.33	1.35	1.35	1.33	1.34	1.35	1.29	1.29	1.32	1.34	1.26	1.30	1.28	1.32	1.33	1.34	1.27	1.26	1.27	1.28	1.28	1.35
20	1.89	1.65	1.88	1.25	1.25	1.28	1.29	1.28	1.31	1.31	1.32	1.36	1.34	1.34	1.32	1.34	1.29	1.32	1.33	1.27	1.29	1.31	1.31	1.34	1.34	1.27	1.26	1.28	1.30	1.28	1.30	1.37
21	1.96	1.73	1.94	1.28	1.30	1.28	1.28	1.32	1.30	1.33	1.33	1.35	1.35	1.32	1.33	1.30	1.33	1.34	1.26	1.30	1.30	1.31	1.35	1.37	1.25	1.29	1.26	1.29	1.32	1.31	1.32	1.39
22	2.04	1.80	1.43	1.34	1.31	1.27	1.27	1.32	1.32	1.32	1.35	1.34	1.34	1.33	1.28	1.29	1.33	1.27	1.29	1.31	1.31	1.35	1.35	1.26	1.27	1.29	1.27	1.32	1.31	1.31	1.17	1.19
23	2.12	1.86	1.57	1.41	1.42	1.30	1.30	1.31	1.33	1.33	1.34	1.35	1.32	1.29	1.28	1.35	1.27	1.28	1.30	1.32	1.34	1.36	1.24	1.28	1.27	1.28	1.30	1.34	1.30	1.18	1.18	1.23
24	2.17	1.92	1.63	1.48	1.52	1.29	1.32	1.30	1.32	1.35	1.33	1.34	1.33	1.28	1.31	1.34	1.27	1.29	1.31	1.34	1.35	1.26	1.27	1.29	1.26	1.31	1.31	1.33	1.16	1.18	1.21	1.22
25	2.21	1.98	1.66	1.32	1.59	1.28	1.31	1.35	1.34	1.34	1.34	1.34	1.28	1.27	1.32	1.28	1.28	1.33	1.32	1.37	1.24	1.28	1.27	1.30	1.29	1.32	1.18	1.15	1.16	1.22	1.22	1.27
26	2.23	2.04	1.70	1.32	1.27	1.27	1.30	1.34	1.35	1.33	1.33	1.34	1.28	1.31	1.27	1.28	1.29	1.33	1.33	1.27	1.27	1.29	1.26	1.32	1.31	1.19	1.14	1.18	1.20	1.22	1.24	1.27
27	2.25	2.08	1.74	1.34	1.30	1.30	1.29	1.33	1.35	1.34	1.32	1.29	1.27	1.33	1.27	1.31	1.30	1.34	1.27	1.27	1.27	1.28	1.29	1.33	1.18	1.15	1.16	1.21	1.21	1.24	1.22	1.28
28	2.27	2.08	1.77	1.37	1.30	1.32	1.32	1.32	1.33	1.34	1.33	1.29	1.31	1.27	1.28	1.31	1.32	1.35	1.25	1.28	1.26	1.31	1.32	1.33	1.15	1.17	1.19	1.23	1.22	1.21	1.23	1.27
29	2.28	2.01	1.80	1.41	1.33	1.31	1.33	1.36	1.33	1.32	1.32	1.28	1.32	1.28	1.29	1.34	1.32	1.26	1.26	1.31	1.30	1.32	1.31	1.16	1.16	1.21	1.21	1.25	1.20	1.23	1.25	1.33
30	2.27	2.01	1.83	1.46	1.37	1.30	1.33	1.36	1.34	1.33	1.27	1.32	1.27	1.29	1.31	1.34	1.34	1.26	1.27	1.30	1.30	1.31	1.16	1.18	1.19	1.22	1.23	1.23	1.20	1.26	1.25	1.31
31	2.27	2.05	1.86	1.51	1.42	1.30	1.32	1.35	1.33	1.33	1.28	1.34	1.27	1.30	1.31	1.37	1.23	1.27	1.26	1.31	1.30	1.18	1.18	1.22	1.22	1.24	1.21	1.25	1.25	1.25	1.28	1.34
32	2.28	2.06	1.90	1.56	1.47	1.30	1.31	1.34	1.33	1.32	1.27	1.33	1.27	1.29	1.31	1.36	1.24	1.28	1.27	1.32	1.31	1.16	1.16	1.19	1.20	1.21	1.19	1.21	1.21	1.23	1.25	1.32

我首先观察到代码执行时间较短的对应 block size 有规律地出现在图中，且两张图的这一现象完全一致。同样也是在这篇知乎回答中我找到了答案：

同一个 block 中，连续的 32 个线程组成一个 warp，这 32 个线程每次执行同一条指令，也就是所谓的 SIMT，即使最后一个 warp 中有效的线程数量不足 32，也要使用相同的硬件资源，所以 block_size 最好是 32 的整数倍

我验证发现，代码执行时间较短的对应 block size 的确总线程数接近一个 32 的整数倍，我想这也是为什么提供的测试代码 X 维度以 32 为单位。

其次我还观察到使用 Shared Memory 的加速比在 X/Y 维度 thread 数较小时显著变小、而在 block size 较大时又有回落，这与先前的结论一致。但是这张图中 X 维度与 Y 维度明显不对称，这是为什么？我还没能得到确定的结论。

5. 对于任意一个给定程序，应该如何设置 thread block size？

   答：应当考虑运行的节点硬件配置（SM 上最大线程数和最大 block 数量），一般在 128 / 256 / 512 中选择。同时应考虑计算任务的性质，如矩阵运算的维数是否能被 X、Y 维度的 thread 数整除，避免增添不整除引起的额外开销。

6. 对于任意一个给定程序，应该如何决定 shared memory 的使用？

   答：首先应该考察程序的数据依赖情况和计算任务性质，如每个 thread 都可以独立完成计算、thread 之间也没有重复计算的可能，则无需使用 shared memory。其次应当考虑 shared memory 的局域性及其大小，太大的 shared memory 为 block 的调度增添了困难，因此如果每个 thread 都需要全局的所有信息，则亦没必要使用 shared memory。同时需要考虑 shared memory 与 block 划分之间的关系，如果计算任务本身可以分块进行、互不干扰或重复，则适合将这些数据块对应于一个 block、共用 shared memory。