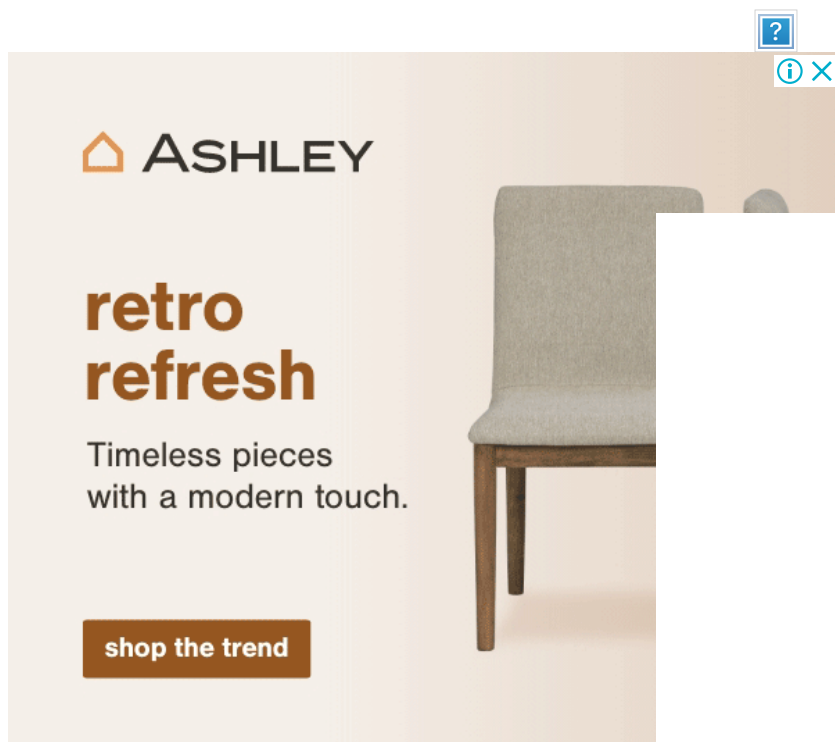




# 谭升的博客

人工智能基础



## 【CUDA 基础】5.3 减少全局内存访问

📅 2018-06-04 | 📁 [CUDA](#) | [Freshman](#) | 💬 0 | 👁

**Abstract:** 本文介绍使用共享内存进行归约，并比较全局内存归约与共享内存归约之间的性能差距

**Keywords:** 共享内存，归约

### 减少全局内存访问

逻辑是非常重要的，一旦你学会了逻辑，很多假的东西你可以轻松的识别出来，这会使你更加强大而不会被任何人或者组织洗脑。

废话少说，开始今天的博客。

使用共享内存的主要原因就是减少对全局内存的访问，来减少不必要的延迟，第三章我们学过了归约，可

以参考：

- <https://face2ai.com/CUDA-F-3-4-避免分支分化/>
- <https://face2ai.com/CUDA-F-3-5-展开循环/>

这两篇博客包含我们前面使用全局内存进行归约的各种技术，我们几天也要用其中一部分代码作为比较，来体现我们共享内存的优势。

我们要集中解决下面两个问题：

1. 如何重新安排数据访问模式以避免线程束分化
2. 如何展开循环以保证有足够的操作使指令和内存带宽饱和

本文我们通过对比研究前面的部分代码，来分析为何要使用共享内存，以及如何使用共享内存。

## 使用共享内存的并行归约

我们首先来回忆全局内存下的，完全展开的归约计算：

```
1  __global__ void reduceGmem(int * g_idata,int * g_odata,unsigned int n)
2  {
3      //set thread ID
4      unsigned int tid = threadIdx.x;
5      unsigned int idx = blockDim.x*blockIdx.x+threadIdx.x;
6      //boundary check
7      if (tid >= n) return;
8      //convert global data pointer to the
9      int *idata = g_idata + blockIdx.x*blockDim.x;
10
11      //in-place reduction in global memory
12      if(blockDim.x>=1024 && tid <512)
13          idata[tid]+=idata[tid+512];
14      __syncthreads();
15      if(blockDim.x>=512 && tid <256)
16          idata[tid]+=idata[tid+256];
17      __syncthreads();
18      if(blockDim.x>=256 && tid <128)
19          idata[tid]+=idata[tid+128];
20      __syncthreads();
21      if(blockDim.x>=128 && tid <64)
22          idata[tid]+=idata[tid+64];
23      __syncthreads();
```

```

24         //write result for this block to global mem
25         if(tid<32)
26         {
27             volatile int *vmem = idata;
28             vmem[tid]+=vmem[tid+32];
29             vmem[tid]+=vmem[tid+16];
30             vmem[tid]+=vmem[tid+8];
31             vmem[tid]+=vmem[tid+4];
32             vmem[tid]+=vmem[tid+2];
33             vmem[tid]+=vmem[tid+1];
34
35         }
36
37         if (tid == 0)
38             g_odata[blockIdx.x] = idata[0];
39
40     }

```

下面这步是计算当前线程的索引位置：

```

1  unsigned int idx = blockDim.x*blockIdx.x+threadIdx.x;

```

当前线程块对应的数据块首地址

```

1  int *idata = g_idata + blockIdx.x*blockDim.x;

```

然后是展开循环的部分，tid是当前线程块中线程的标号，主要区别于全局编号idx：

```

1  if(blockDim.x>=1024 && tid <512)
2      idata[tid]+=idata[tid+512];
3  __syncthreads();
4  if(blockDim.x>=512 && tid <256)
5      idata[tid]+=idata[tid+256];
6  __syncthreads();
7  if(blockDim.x>=256 && tid <128)
8      idata[tid]+=idata[tid+128];
9  __syncthreads();
10 if(blockDim.x>=128 && tid <64)
11     idata[tid]+=idata[tid+64];
12 __syncthreads();

```

这一步把是当前线程块中的所有数据归约到前64个元素中，接着使用如下代码，将最后64个元素归约成一个

```
1  if(tid<32)
2  {
3      volatile int *vmem = idata;
4      vmem[tid]+=vmem[tid+32];
5      vmem[tid]+=vmem[tid+16];
6      vmem[tid]+=vmem[tid+8];
7      vmem[tid]+=vmem[tid+4];
8      vmem[tid]+=vmem[tid+2];
9      vmem[tid]+=vmem[tid+1];
10 }
```

注意这里声明了一个volatile变量，如果我们不这么做，编译器不能保证这些数据读写操作按照代码中的顺序执行（参考5.1中关于编译器数据传输部分的说明），所以必须要这么做。

然后我们执行以下这段代码，虽然前面执行过了，我们还是执行以下，观察下结果：

完整的可执行代码依旧在GitHub上可以找到

github:[https://github.com/Tony-Tan/CUDA\\_Freshman](https://github.com/Tony-Tan/CUDA_Freshman)点个星星也不会很累，拜托啦😁

```
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$ sudo /usr/local/cuda/bin/nvprof ./reduce_integer_shared_memory
[sudo] tony 的密码:
==2829== NVPROF is profiling process 2829, command: ./reduce_integer_shared_memory
Using device 0: GeForce GTX 1050 Ti
with array size 16777216
grid 16384 block 1024
cpu reduce      elapsed 0.040242 ms cpu_sum: 2138842777
gpu warmup      elapsed 0.005381 ms
reduceGmem      elapsed 0.004273 ms gpu_sum: 2138842777
reduceSmem      elapsed 0.002895 ms gpu_sum: 2138842777
reduceUnroll4Smem elapsed 0.000808 ms gpu_sum: 2138842777
==2829== Profiling application: ./reduce_integer_shared_memory
==2829== Profiling result:
Type      Time(%)   Time      Calls      Avg      Min      Max      Name
GPU activities:
  80.26%   54.166ms    4    13.541ms  13.479ms  13.577ms  [CUDA memcpy HtoD]
  7.94%    5.3567ms   1    5.3567ms  5.3567ms  5.3567ms  warmup(int*, int*, unsigned int)
  6.31%    4.2577ms   1    4.2577ms  4.2577ms  4.2577ms  reduceGmem(int*, int*, unsigned int)
  4.27%    2.8817ms   1    2.8817ms  2.8817ms  2.8817ms  reduceSmem(int*, int*, unsigned int)
  1.18%    794.69us   1    794.69us  794.69us  794.69us  reduceUnroll4Smem(int*, int*, unsigned int)
  0.05%    31.584us   3    10.528us  10.528us  10.528us  [CUDA memcpy DtoH]
API calls:
  51.34%   122.84ms   2    61.422ms  271.65us  122.57ms  cudaMalloc
  22.81%   54.579ms  7    7.7971ms  30.272us  13.651ms  cudaMemcpy
  18.24%   43.656ms   1    43.656ms  43.656ms  43.656ms  cudaDeviceReset
  5.83%   13.951ms   8    1.7439ms  160.99us  5.3590ms  cudaDeviceSynchronize
  0.94%    2.2453ms   2    1.1226ms  141.55us  2.1037ms  cudaFree
  0.37%    897.13us  94    9.5430us   405ns   399.84us  cuDeviceGetAttribute
  0.35%    845.60us   1    845.60us  845.60us  845.60us  cudaGetDeviceProperties
  0.05%   127.57us   1    127.57us  127.57us  127.57us  cuDeviceTotalMem
  0.03%    80.500us   1    80.500us  80.500us  80.500us  cuDeviceGetName
  0.02%    47.931us   4    11.982us  9.1380us  18.912us  cudaLaunch
  0.01%    12.718us   1    12.718us  12.718us  12.718us  cudaSetDevice
  0.00%    4.3160us   3    1.4380us   460ns   3.2160us  cuDeviceGetCount
  0.00%    2.3140us  12    192ns    100ns   419ns    cudaSetupArgument
  0.00%    2.2460us   2    1.1230us   451ns   1.7950us  cuDeviceGet
  0.00%    1.6090us   4    402ns    265ns   649ns    cudaConfigureCall
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$
```

这里我们假装看不到别的，只看我们的核函数，可见器质性时间是4.25ms左右

然后我们对上面的代码进行改写，改写成共享内存的版本，来看代码：

```
1  __global__ void reduceSmem(int * g_idata,int * g_odata,unsigned int n)
2  {
3      //set thread ID
4      __shared__ int smem[DIM];
5      unsigned int tid = threadIdx.x;
6      //unsigned int idx = blockDim.x*blockIdx.x+threadIdx.x;
7      //boundary check
8      if (tid >= n) return;
9      //convert global data pointer to the
10     int *idata = g_idata + blockIdx.x*blockDim.x;
11
12     smem[tid]=idata[tid];
13     __syncthreads();
14     //in-place reduction in global memory
15     if(blockDim.x>=1024 && tid <512)
16         smem[tid]+=smem[tid+512];
17     __syncthreads();
18     if(blockDim.x>=512 && tid <256)
19         smem[tid]+=smem[tid+256];
20     __syncthreads();
21     if(blockDim.x>=256 && tid <128)
22         smem[tid]+=smem[tid+128];
23     __syncthreads();
24     if(blockDim.x>=128 && tid <64)
25         smem[tid]+=smem[tid+64];
26     __syncthreads();
27     //write result for this block to global mem
28     if(tid<32)
29     {
30         volatile int *vsmem = smem;
31         vsmem[tid]+=vsmem[tid+32];
32         vsmem[tid]+=vsmem[tid+16];
33         vsmem[tid]+=vsmem[tid+8];
34         vsmem[tid]+=vsmem[tid+4];
35         vsmem[tid]+=vsmem[tid+2];
36         vsmem[tid]+=vsmem[tid+1];
37
38     }
39
40     if (tid == 0)
```

```

41         g_odata[blockIdx.x] = smem[0];
42
43     }

```

唯一的不同就是多了一个共享内存的声明，以及各线程将全局写入共享内存，以及后面的同步指令：

```

1     smem[tid]=idata[tid];
2     __syncthreads();

```

这一步过后同步保证该线程块内的所有线程，都执行到此处后继续向下进行，这是可以理解的，因为我们的归约只针对本块内，当然如果想跨几个块执行，可能同步这里就有问题了，这个是上一节课要讨论的，这里就不过多解释了，我们接着就看到一个volatile类型的指针，指向共享内存，对最后64个归约结果进行归约，整个过程和全局内存一毛一样，只不过一个在全局内存操作，一个在共享内存操作，得到相同的结果，我们也来看一下运行结果。

```

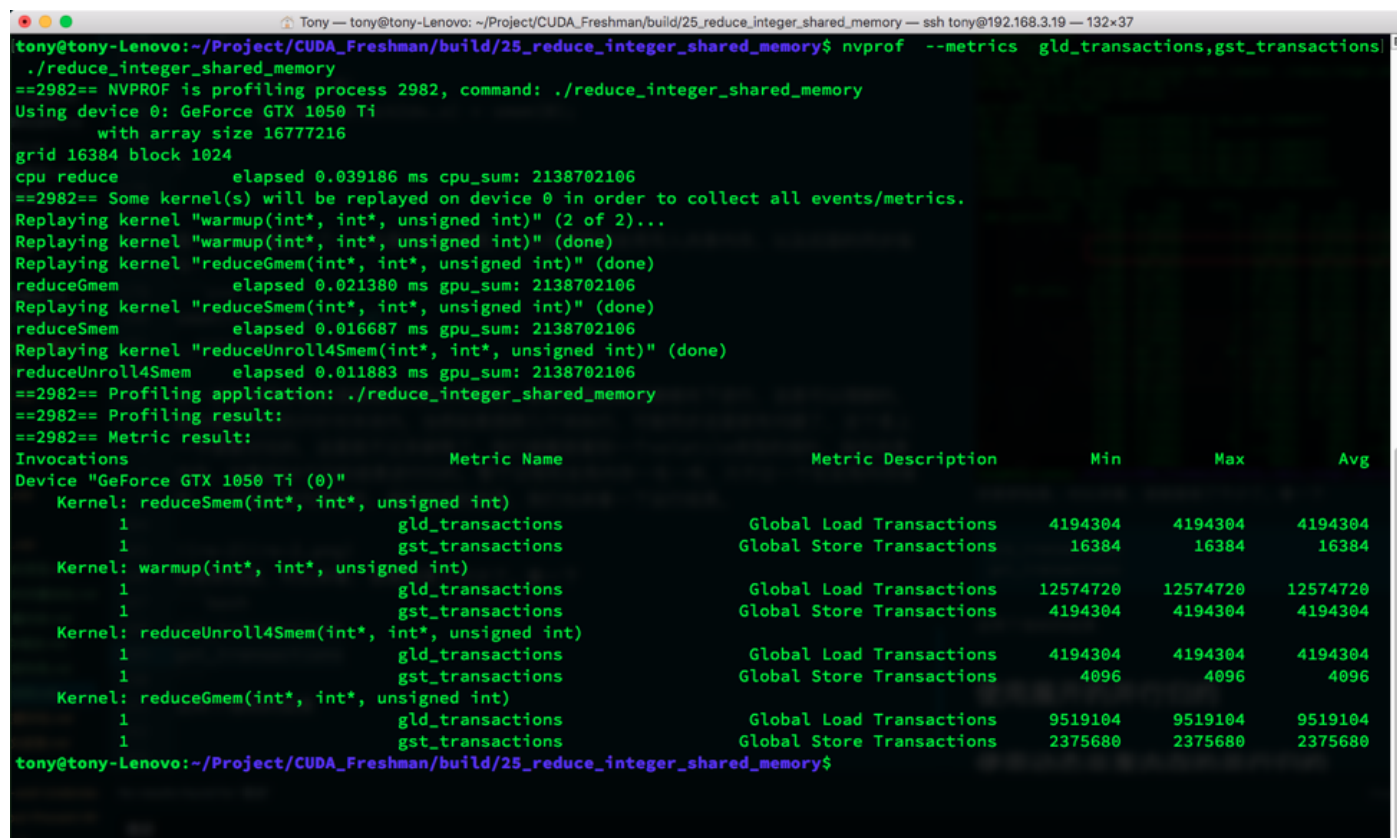
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$ sudo /usr/local/cuda/bin/nvprof ./reduce_integer_shared_memory
[sudo] tony 的密码:
==2829== NVPROF is profiling process 2829, command: ./reduce_integer_shared_memory
Using device 0: GeForce GTX 1050 Ti
with array size 16777216
grid 16384 block 1024
cpu reduce      elapsed 0.040242 ms cpu_sum: 2138842777
gpu warmup      elapsed 0.005381 ms
reduceGmem      elapsed 0.004273 ms gpu_sum: 2138842777
reduceSmem      elapsed 0.002895 ms gpu_sum: 2138842777
reduceUnroll4Smem elapsed 0.000808 ms gpu_sum: 2138842777
==2829== Profiling application: ./reduce_integer_shared_memory
==2829== Profiling result:
Type      Time(%)   Time          Calls      Avg        Min         Max      Name
GPU activities: 80.26%  54.166ms      4    13.541ms  13.479ms  13.577ms  [CUDA memcpy HtoD]
              7.94%  5.3567ms      1    5.3567ms  5.3567ms  5.3567ms  warmup(int*, int*, unsigned int)
              6.31%  4.2577ms      1    4.2577ms  4.2577ms  4.2577ms  reduceGMem(int*, int*, unsigned int)
              4.27%  2.8817ms      1    2.8817ms  2.8817ms  2.8817ms  reduceSmem(int*, int*, unsigned int)
              1.18%  794.69us      1    794.69us  794.69us  794.69us  reduceUnroll4Smem(int*, int*, unsigned int)
              0.05%  31.584us      3    10.528us  10.528us  10.528us  [CUDA memcpy DtoH]
API calls: 51.34%  122.84ms      2    61.422ms  271.65us  122.57ms  cudaMalloc
           22.81%  54.579ms      7    7.7971ms  30.272us  13.651ms  cudaMemcpy
           18.24%  43.656ms      1    43.656ms  43.656ms  43.656ms  cudaDeviceReset
           5.83%  13.951ms      8    1.7439ms  160.99us  5.3590ms  cudaDeviceSynchronize
           0.94%  2.2453ms      2    1.1226ms  141.55us  2.1037ms  cudaFree
           0.37%  897.13us      94    9.5430us  405ns    399.84us  cuDeviceGetAttribute
           0.35%  845.60us      1    845.60us  845.60us  845.60us  cudaGetDeviceProperties
           0.05%  127.57us      1    127.57us  127.57us  127.57us  cuDeviceTotalMem
           0.03%  80.500us      1    80.500us  80.500us  80.500us  cuDeviceGetName
           0.02%  47.931us      4    11.982us  9.1380us  18.912us  cudaLaunch
           0.01%  12.718us      1    12.718us  12.718us  12.718us  cudaSetDevice
           0.00%  4.3160us      3    1.4380us  460ns    3.2160us  cuDeviceGetCount
           0.00%  2.3140us      12    192ns    100ns    419ns    cudaSetupArgument
           0.00%  2.2460us      2    1.1230us  451ns    1.7950us  cuDeviceGet
           0.00%  1.6090us      4    402ns    265ns    649ns    cudaConfigureCall
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$

```

还是那张图，对比来看，速度提高了不少了。看一下

```
1  gld_transactions
2  gst_transactions
```

这两个指标的结果



```
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$ nvprof --metrics gld_transactions,gst_transactions
./reduce_integer_shared_memory
==2982== NVPROF is profiling process 2982, command: ./reduce_integer_shared_memory
Using device 0: GeForce GTX 1050 Ti
with array size 16777216
grid 16384 block 1024
cpu reduce elapsed 0.039186 ms cpu_sum: 2138702106
==2982== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Replaying kernel "warmup(int*, int*, unsigned int)" (2 of 2)...
Replaying kernel "warmup(int*, int*, unsigned int)" (done)
Replaying kernel "reduceGmem(int*, int*, unsigned int)" (done)
reduceGmem elapsed 0.021380 ms gpu_sum: 2138702106
Replaying kernel "reduceSmem(int*, int*, unsigned int)" (done)
reduceSmem elapsed 0.016687 ms gpu_sum: 2138702106
Replaying kernel "reduceUnroll4Smem(int*, int*, unsigned int)" (done)
reduceUnroll4Smem elapsed 0.011883 ms gpu_sum: 2138702106
==2982== Profiling application: ./reduce_integer_shared_memory
==2982== Profiling result:
==2982== Metric result:
Invocations Metric Name Metric Description Min Max Avg
Device "GeForce GTX 1050 Ti (0)"
Kernel: reduceSmem(int*, int*, unsigned int)
1 gld_transactions Global Load Transactions 4194304 4194304 4194304
1 gst_transactions Global Store Transactions 16384 16384 16384
Kernel: warmup(int*, int*, unsigned int)
1 gld_transactions Global Load Transactions 12574720 12574720 12574720
1 gst_transactions Global Store Transactions 4194304 4194304 4194304
Kernel: reduceUnroll4Smem(int*, int*, unsigned int)
1 gld_transactions Global Load Transactions 4194304 4194304 4194304
1 gst_transactions Global Store Transactions 4096 4096 4096
Kernel: reduceGmem(int*, int*, unsigned int)
1 gld_transactions Global Load Transactions 9519104 9519104 9519104
1 gst_transactions Global Store Transactions 2375680 2375680 2375680
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$
```

可以看出使用共享内存的，比使用全局内存的高到不知道哪里去了

## 使用展开的并行归约

可能看到上面的截图你已经知道我接下来要并行4块了，对于前面说的，使用共享内存不能并行四块，是因为没办法同步读四个块，这里我们还是用老方法进行并行四个块，就是在写入共享内存之前进行归约，4个块变成一个，然后把这一个存入共享内存，进行常规的共享内存归约：

```
1  __global__ void reduceUnroll4Smem(int * g_idata,int * g_odata,unsigned int n)
2  {
3      //set thread ID
4      __shared__ int smem[DIM];
5      unsigned int tid = threadIdx.x;
6      unsigned int idx = blockDim.x*blockIdx.x*4+threadIdx.x;
7      //boundary check
8      if (tid >= n) return;
```

```

9         //convert global data pointer to the
10     int tempSum=0;
11     if(idx+3 * blockDim.x<=n)
12     {
13         int a1=g_idata[idx];
14         int a2=g_idata[idx+blockDim.x];
15         int a3=g_idata[idx+2*blockDim.x];
16         int a4=g_idata[idx+3*blockDim.x];
17         tempSum=a1+a2+a3+a4;
18
19     }
20     smem[tid]=tempSum;
21     __syncthreads();
22     //in-place reduction in global memory
23     if(blockDim.x>=1024 && tid <512)
24         smem[tid]+=smem[tid+512];
25     __syncthreads();
26     if(blockDim.x>=512 && tid <256)
27         smem[tid]+=smem[tid+256];
28     __syncthreads();
29     if(blockDim.x>=256 && tid <128)
30         smem[tid]+=smem[tid+128];
31     __syncthreads();
32     if(blockDim.x>=128 && tid <64)
33         smem[tid]+=smem[tid+64];
34     __syncthreads();
35     //write result for this block to global mem
36     if(tid<32)
37     {
38         volatile int *vsmem = smem;
39         vsmem[tid]+=vsmem[tid+32];
40         vsmem[tid]+=vsmem[tid+16];
41         vsmem[tid]+=vsmem[tid+8];
42         vsmem[tid]+=vsmem[tid+4];
43         vsmem[tid]+=vsmem[tid+2];
44         vsmem[tid]+=vsmem[tid+1];
45
46     }
47
48     if (tid == 0)
49         g_odata[blockIdx.x] = smem[0];
50
51 }

```

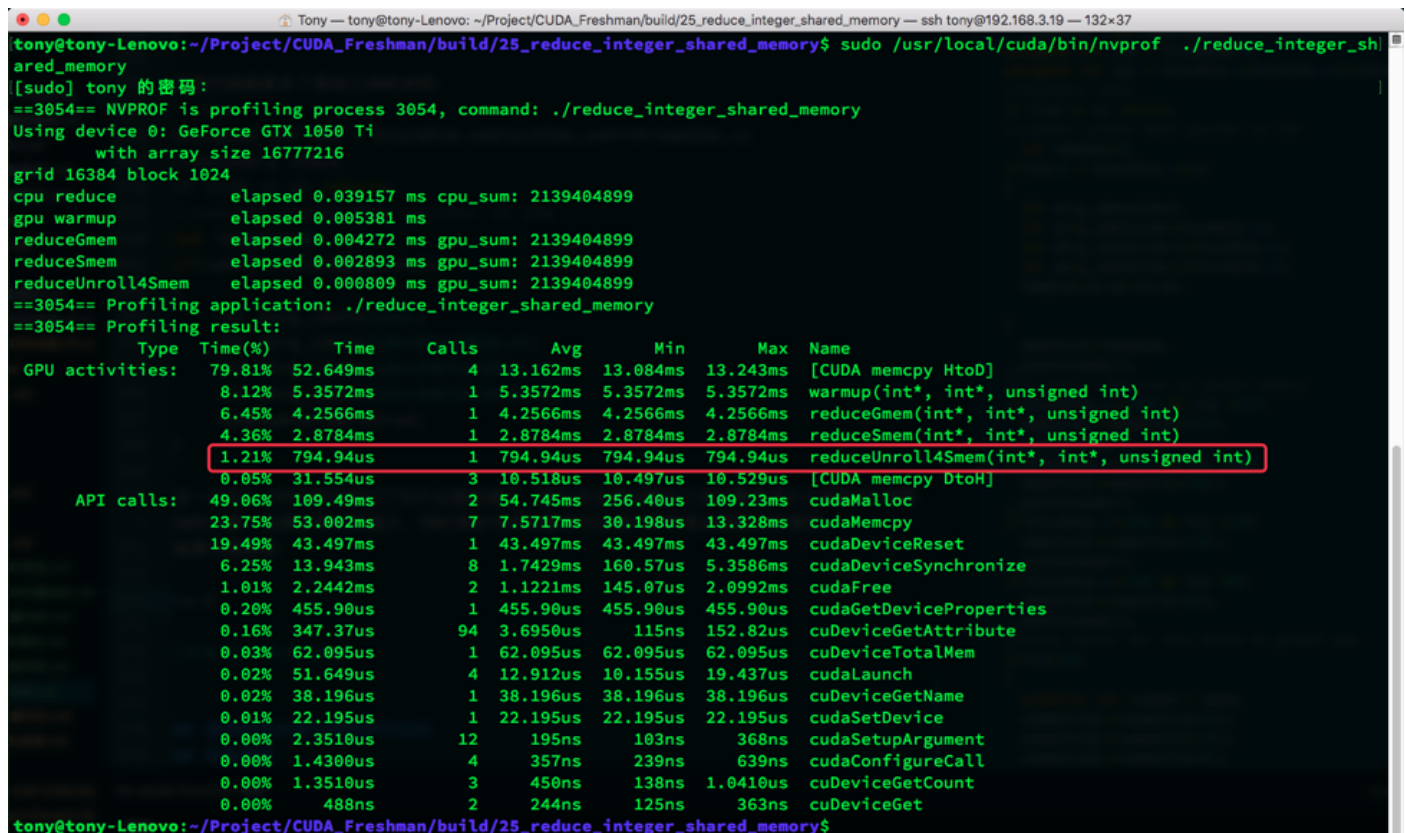


这段代码就是多了其他三块的求和：

```
1  unsigned int idx = blockDim.x*blockIdx.x*4+threadIdx.x;
2  //boundary check
3  if (tid >= n) return;
4  //convert global data pointer to the
5  int tempSum=0;
6  if(idx+3 * blockDim.x<=n)
7  {
8      int a1=g_idata[idx];
9      int a2=g_idata[idx+blockDim.x];
10     int a3=g_idata[idx+2*blockDim.x];
11     int a4=g_idata[idx+3*blockDim.x];
12     tempSum=a1+a2+a3+a4;
13 }
```

这一步在3.5中已经介绍过了为什么能加速了，因为可以通过增加三步计算而减少之前的3个线程块的计算，这是非常大的减少。同时多步内存加载也可以使内存带宽达到更好的使用。

结果可想而知：



```
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$ sudo /usr/local/cuda/bin/nvprof ./reduce_integer_sh
ared_memory
[sudo] tony 的密码:
==3054== NVPROF is profiling process 3054, command: ./reduce_integer_shared_memory
Using device 0: GeForce GTX 1050 Ti
with array size 16777216
grid 16384 block 1024
cpu reduce      elapsed 0.039157 ms cpu_sum: 2139404899
gpu warmup      elapsed 0.005381 ms
reduceGmem      elapsed 0.004272 ms gpu_sum: 2139404899
reduceSmem      elapsed 0.002893 ms gpu_sum: 2139404899
reduceUnroll4Smem elapsed 0.000809 ms gpu_sum: 2139404899
==3054== Profiling application: ./reduce_integer_shared_memory
==3054== Profiling result:
Type Time(%) Time Calls Avg Min Max Name
GPU activities: 79.81% 52.649ms 4 13.162ms 13.084ms 13.243ms [CUDA memcpy HtoD]
8.12% 5.3572ms 1 5.3572ms 5.3572ms 5.3572ms warmup(int*, int*, unsigned int)
6.45% 4.2566ms 1 4.2566ms 4.2566ms 4.2566ms reduceGmem(int*, int*, unsigned int)
4.36% 2.8784ms 1 2.8784ms 2.8784ms 2.8784ms reduceSmem(int*, int*, unsigned int)
1.21% 794.94us 1 794.94us 794.94us 794.94us reduceUnroll4Smem(int*, int*, unsigned int)
0.05% 31.554us 3 10.518us 10.497us 10.529us [CUDA memcpy DtoH]
API calls: 49.06% 109.49ms 2 54.745ms 256.40us 109.23ms cudaMalloc
23.75% 53.002ms 7 7.5717ms 30.198us 13.328ms cudaMemcpy
19.49% 43.497ms 1 43.497ms 43.497ms 43.497ms cudaDeviceReset
6.25% 13.943ms 8 1.7429ms 160.57us 5.3586ms cudaDeviceSynchronize
1.01% 2.2442ms 2 1.1221ms 145.07us 2.0992ms cudaFree
0.20% 455.90us 1 455.90us 455.90us 455.90us cudaGetDeviceProperties
0.16% 347.37us 94 3.6950us 115ns 152.82us cuDeviceGetAttribute
0.03% 62.095us 1 62.095us 62.095us 62.095us cuDeviceTotalMem
0.02% 51.649us 4 12.912us 10.155us 19.437us cudaLaunch
0.02% 38.196us 1 38.196us 38.196us 38.196us cuDeviceGetName
0.01% 22.195us 1 22.195us 22.195us 22.195us cudaSetDevice
0.00% 2.3510us 12 195ns 103ns 368ns cudaSetupArgument
0.00% 1.4300us 4 357ns 239ns 639ns cudaConfigureCall
0.00% 1.3510us 3 450ns 138ns 1.0410us cuDeviceGetCount
0.00% 488ns 2 244ns 125ns 363ns cuDeviceGet
```

```
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$ nvprof --metrics gld_transactions,gst_transactions ./reduce_integer_shared_memory
==2982== NVPROF is profiling process 2982, command: ./reduce_integer_shared_memory
Using device 0: GeForce GTX 1050 Ti
with array size 16777216
grid 16384 block 1024
cpu reduce elapsed 0.039186 ms cpu_sum: 2138702106
==2982== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Replaying kernel "warmup(int*, int*, unsigned int)" (2 of 2)...
Replaying kernel "warmup(int*, int*, unsigned int)" (done)
Replaying kernel "reduceGmem(int*, int*, unsigned int)" (done)
reduceGmem elapsed 0.021380 ms gpu_sum: 2138702106
Replaying kernel "reduceSmem(int*, int*, unsigned int)" (done)
reduceSmem elapsed 0.016687 ms gpu_sum: 2138702106
Replaying kernel "reduceUnroll4Smem(int*, int*, unsigned int)" (done)
reduceUnroll4Smem elapsed 0.011883 ms gpu_sum: 2138702106
==2982== Profiling application: ./reduce_integer_shared_memory
==2982== Profiling result:
Invocations Metric Name Metric Description Min Max Avg
Device "GeForce GTX 1050 Ti (0)"
Kernel: reduceSmem(int*, int*, unsigned int)
1 gld_transactions Global Load Transactions 4194304 4194304 4194304
1 gst_transactions Global Store Transactions 16384 16384 16384
Kernel: warmup(int*, int*, unsigned int)
1 gld_transactions Global Load Transactions 12574720 12574720 12574720
1 gst_transactions Global Store Transactions 4194304 4194304 4194304
Kernel: reduceUnroll4Smem(int*, int*, unsigned int)
1 gld_transactions Global Load Transactions 4194304 4194304 4194304
1 gst_transactions Global Store Transactions 4096 4096 4096
Kernel: reduceGmem(int*, int*, unsigned int)
1 gld_transactions Global Load Transactions 9519104 9519104 9519104
1 gst_transactions Global Store Transactions 2375680 2375680 2375680
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$
```

吞吐量指标：

```
1 nvprof --metrics dram_read_throughput ./reduce_integer_shared_memory
```

```
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$ nvprof --metrics dram_read_throughput ./reduce_integer_shared_memory
==3107== NVPROF is profiling process 3107, command: ./reduce_integer_shared_memory
Using device 0: GeForce GTX 1050 Ti
with array size 16777216
grid 16384 block 1024
cpu reduce      elapsed 0.039230 ms cpu_sum: 2139220657
gpu warmup      elapsed 0.007911 ms
reduceGmem      elapsed 0.005178 ms gpu_sum: 2139220657
reduceSmem      elapsed 0.003508 ms gpu_sum: 2139220657
reduceUnroll4Smem elapsed 0.001648 ms gpu_sum: 2139220657
==3107== Profiling application: ./reduce_integer_shared_memory
==3107== Profiling result:
==3107== Metric result:
Invocations      Metric Name      Metric Description      Min      Max      Avg
Device "GeForce GTX 1050 Ti (0)"
Kernel: reduceSmem(int*, int*, unsigned int)
1      dram_read_throughput      Device Memory Read Throughput      24.169GB/s      24.169GB/s      24.169GB/s
Kernel: warmup(int*, int*, unsigned int)
1      dram_read_throughput      Device Memory Read Throughput      5.9627GB/s      5.9627GB/s      5.9627GB/s
Kernel: reduceUnroll4Smem(int*, int*, unsigned int)
1      dram_read_throughput      Device Memory Read Throughput      86.484GB/s      86.484GB/s      86.484GB/s
Kernel: reduceGmem(int*, int*, unsigned int)
1      dram_read_throughput      Device Memory Read Throughput      15.600GB/s      15.600GB/s      15.600GB/s
tony@tony-Lenovo:~/Project/CUDA_Freshman/build/25_reduce_integer_shared_memory$
```

无论是指标还是运行速度，都有非常显著的提升。

我们这里总结下展开的优势：

- I/O得到了更多的并行，就是我上面说的更好的利用带宽，增加了吞吐量
- 全局内存存储事务减少到  $\frac{1}{4}$ ，这个主要针对最后一步，将结果存入全局内存
- 整体性能巨幅提升

## 使用动态共享内存的并行归约

然后我们看一下动态版本，其实动态版本没啥可看的，只是写法上有点不同，把宏改成核函数配置参数，注意其单位是字节就好，也就是不要忘了sizeof()就行了。

这里不啰嗦了。

## 有效带宽

对比一下数据，回顾一下我们的有效带宽，其计算公式(4.4中有详细介绍)：

$$\text{有效带宽} = \frac{(\text{读字节数} + \text{写字节数}) \times 10^{-9}}{\text{运行时间}} \tag{1}$$

可以研究三个核函数的有效带宽，这里就不再一个个计算了，因为这个在4.4中已经教大家做了，我们可以自己算一下，并得出结论

## 总结

本文主要高速大家如何使用共享内存加速归约，以及结合了共享内存的展开能更高的提高效率，注意线程块内的同步，这个是重要的。

本文作者：谭升

本文链接：<https://face2ai.com/CUDA-F-5-3-减少全局内存访问/>

版权声明：本博客所有文章除特别声明外，均采用 [CC BY-NC-SA 4.0](#) 许可协议。转载请注明出处！

### 相关文章

- [【CUDA 基础】3.5 展开循环](#)
- [【CUDA 基础】5.0 共享内存和常量内存](#)
- [【Julia】整型和浮点型数字](#)
- [【Julia】变量](#)
- [【Julia】开始使用Julia](#)

[# 归约](#) [# 共享内存](#)

帮帮点一点（不用付费哒~）

顶一下