这里比较了三种方法，第一种是cpu下的直接求和，后两者是gpu下的求和，后两者的具体实现如下：

第一种GPU求和（gpu\_sum）：用了32个blocks，每一个blocks有64个threads，输出是一个长度为block数量即32的数组。首先将原数组压缩成32\*64长度的数组，即每一个线程都计算。然后运用原子操作atomicAdd，复杂度是O(n)，每一个block内部求和，存在输出数组的对应位置，最后在cpu端对长度为32的输出数组求和，得到最终结果。

第二种GPU求和（SumArray）：用了32个blocks，每一个blocks有64个threads，输出是一个长度为block数量即32的数组。首先将原数组压缩成32\*64长度的数组，即每一个线程都计算。然后用reduce tree的方法，循环对折求和直到最终长度为1，即每一个block的和会存在第一位，这方法的复杂度是O（log n）。把存在输出数组的对应位置，最后在cpu端对长度为32的输出数组求和，得到最终结果。

实验结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 算法 | 原数组长度 | 使用的GPU | 运算时间（ms） |
| CPU | 64\*32 | RTX 2080 | 0.009 |
| GPU1 | 64\*32 | RTX 2080 | 0.013 |
| GPU2 | 64\*32 | RTX 2080 | 0.00013 |
| CPU | 64\*32\*10 | RTX 2080 | 0.0711 |
| GPU1 | 64\*32\*10 | RTX 2080 | 0.021 |
| GPU2 | 64\*32\*10 | RTX 2080 | 0.00026 |
| CPU | 64\*32\*100 | RTX 2080 | 0.695 |
| GPU1 | 64\*32\*100 | RTX 2080 | 0.054 |
| GPU2 | 64\*32\*100 | RTX 2080 | 0.00556 |
| CPU | 64\*32\*1000 | RTX 2080 | 5.901 |
| GPU1 | 64\*32\*1000 | RTX 2080 | 0.514 |
| GPU2 | 64\*32\*1000 | RTX 2080 | 0.0038 |
| CPU | 64\*32\*1000 | Tesla T4 | 5.01 |
| GPU1 | 64\*32\*1000 | Tesla T4 | 1.326 |
| GPU2 | 64\*32\*1000 | Tesla T4 | 0.019 |
| CPU | 64\*32\*100 | Tesla T4 | 0.400 |
| GPU1 | 64\*32\*100 | Tesla T4 | 0.805 |
| GPU2 | 64\*32\*100 | Tesla T4 | 0.00742 |
| CPU | 64\*32\*10 | Tesla T4 | 0.037 |
| GPU1 | 64\*32\*10 | Tesla T4 | 0.98 |
| GPU2 | 64\*32\*10 | Tesla T4 | 0.0157 |
| CPU | 64\*32 | Tesla T4 | 0.0144 |
| GPU1 | 64\*32 | Tesla T4 | 0.016 |
| GPU2 | 64\*32 | Tesla T4 | 0.000184 |

实验结论及分析：

1. 三个方法的比较上，使用shared memory加上reduce tree的方法在性能上要明显比另外两个方法要好，O（log n）复杂度的算法在计算大量数据时更有优势。
2. GPU1这个算法中，在原数组长度较小时，不需要进行压缩，即每个线程只进行一次atomic操作时，64个线程竞争一个写入地址，其效果不如CPU算法，这说明多个线程的竞争堵塞会导致运行速度的下降。只有当原数组长度较大时，能够体现并行的优势。
3. 在两块显卡的比较方面，reduce tree的算法在两块显卡上都有不错的表现，但atomic算法在RTX2080上的表现要比在Tesla T4上的表现要好不少。