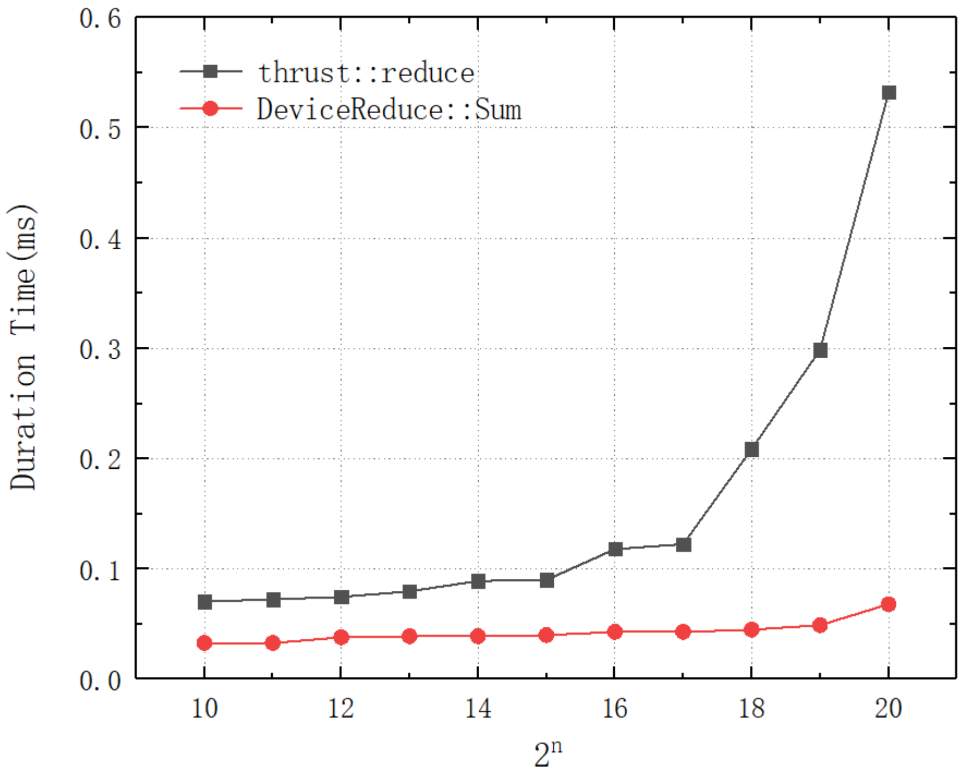
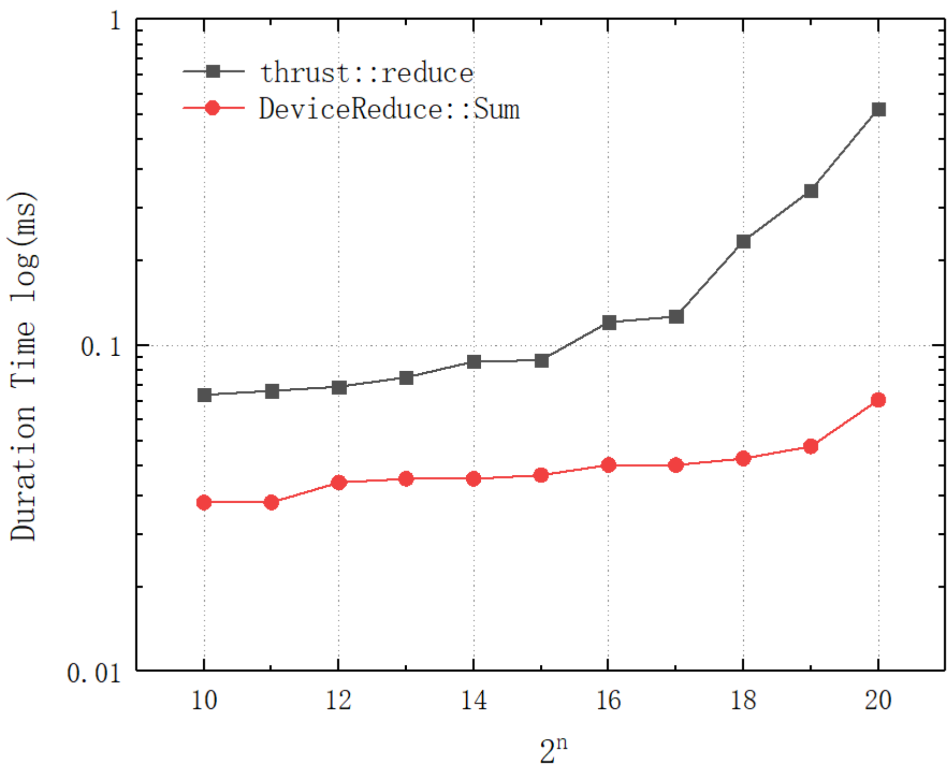
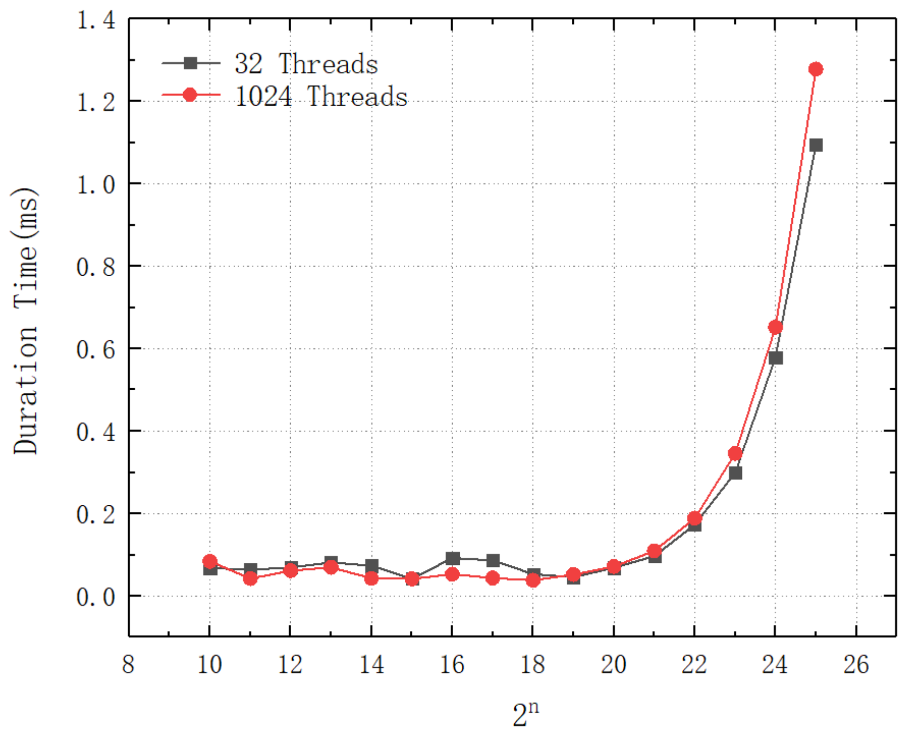
HW07 Task1 c


thrust::reduce
DeviceReduce::Sum
Duration Time(ms)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
10
12
14
16
18
20
2^n


thrust::reduce
DeviceReduce::Sum
Duration Time log(ms)
0.01
0.1
1
10
12
14
16
18
20
2^n



HW07 Task1 d

1. **DeviceReduce::Sum (CUB implementation):**
   DeviceReduce::Sum is part of the CUB library and uses highly optimized algorithms for performing reductions. For large matrices, CUB's performance is typically excellent because it can effectively leverage the parallelism of the GPU. As the matrix size increases, the performance of the CUB implementation may become more apparent because its algorithms and memory management are designed to optimize the processing of large-scale data.
2. **thrust::reduce (Thrust implementation):**
   thrust::reduce is part of the Thrust library and also employs highly optimized algorithms for reduction operations. For large matrices, Thrust generally provides excellent performance and shares similar advantages with CUB. Like CUB, as the input dataset size grows, the performance of the Thrust implementation becomes more evident.
3. **"First Add During Global Load" Parallel Reduction Method (Manually written CUDA implementation):**
   Manually written CUDA implementations often require more optimization work to ensure good performance on large matrices. Performance highly depends on the quality and level of optimization in the CUDA code. If the code is not efficient, it might be limited by the matrix size. For smaller datasets or cases where more control is needed, manually written CUDA implementations might not show better performance compared to CUB or Thrust.

In summary, for large matrices, both DeviceReduce::Sum and thrust::reduce library implementations typically perform well because they use highly optimized algorithms and can take full advantage of GPU parallelism. Manually written CUDA implementations may require more effort to achieve similar performance. However, for smaller datasets or cases where more customization and optimization are required, manually written CUDA implementations offer greater flexibility and room for optimization.