CUDA Homework Assignment 4

Hao-Tse , Hsiao

B12902100

Abstract. In Homework Assignment 4, we compute the dot product of two real vectors using multiple GPUs with CUDA parallel reduction. In particular, we evaluate performance on 2 GPUs using random vectors of size 40,960,000, generated by the routine `RandomInit`. The experiment also investigates the optimal configuration of block and grid sizes to maximize execution speedup.

1 Task Description

We begin by initializing two vectors A and B of size N with random real values. The dot-product of two vectors, defined as $\mathrm{A} \cdot B = \sum_{i=0}^{N-1} A_i B_i$, is computed using both CPU and GPU implementations. We use parallel reduction on the two GPUs to efficiently accumulate the diagonal elements.

2 Task Result

After performing the vector computation on both the CPU and two GPUs, we compiled the results in the following table. CPU execution time is used as the baseline for calculating speedup.Each value in the table represents the execution speedup. The relative difference between GPU and CPU computed results is on the order of 10^{-7}, indicating a high degree of numerical accuracy.

Table 1. GPU Execution Speedup Relative to CPU (Higher is Better)

Threads per Block / Blocks per Grid	64	128	256
128	0.434464	0.428327	0.411511
256	0.450322	0.458863	0.412958
512	0.409025	**0.498623**	0.493033

Table 1 summarizes the execution speedup of various block and grid configurations on the GPU, normalized against CPU execution time. Each value represents the relative performance improvement. The best performance on GPU was observed with 512 threads per block and 128 blocks per grid. The relative error between GPU and CPU results was below 10^{-7}, indicating high numerical accuracy.

3 Discussion

This implementation distributes the workload evenly across two GPUs, with each assigned half of the input vectors and performing partial dot product computation using CUDA kernels.

In this experiment, although GPU parallelism is expected to accelerate computation, the CPU outperformed the GPU in total execution time. This is mainly because the CPU does not incur memory transfer and kernel launch overhead. On the other hand, the GPU experiences significant setup and data I/O time, which reduces its advantage for relatively small problem sizes. For large-scale vector operations, however, GPU acceleration is expected to yield substantial speedup.

Also, a block size that is too large may reduce parallelization efficiency, while a block size that is too small could create excessive thread overhead, reducing computational performance. These findings indicate that while GPU acceleration may not be effective for small vector sizes, it holds significant advantages for large-scale computations with sufficient parallelism.

4 Reference

1. Introduction to CUDA Parallel Programming — Homework Assignment 4