UNIVERSITÀ DEGLI STUDI DI MILANO

GPU Computing Project

Aligned and choalescent parallel sorting on Nvidia GPUs

Author:

Manuele Lucchi

08659A

Academic Year 2022/2023

1 Abstract

This documents describe the performances of an existing algorithm, the Warp Sort, with some additional optimizations on more recent GPUs, comparing it with a serial approach and focusing on the implementation.

Contents

2 Introduction

The article ”High Performance Comparison-Based Sorting Algorithm on Many-Core GPUs” introduces an enhanced version of the Bitonic Sort, another algorithm that allows for parallel sorting and is therefore well suited for GPUs.
The Bitonic Sort is based on the ability to easily sort Bitonic Sequences, partially ordinated sequences that can be split in two ordinated sequences with different direction. The new algorithm called Warp sort has been designed around the CUDA architecture, therefore using all the concepts of Nvidia GPU's to optimize the execution.

The original implementation (as well as this one) is composed by 4 steps and each step is meant to perform each operation ”by a warp”. A warp is a CUDA concept of a block of 32 threads that runs the same instruction at the same time and is the minimun execution unit of the GPU.
To obtain the best throughput, the access on the memory by the threads in a warp must be **aligned** and **choalescent**, meaning respectively that the first address of the transaction is an even multiple of the granularity of the used cache and that all the 32 threads of the warp access to a continuous memory block.
Therefore executing all the steps ”by a warp” allows for the previously detailed properties to be obtained automatically.

3 Algorithm

The algorithm follows the Bitonic Sort scheme, where the first part is about creating Bitonic Sequences and sort them using the Bitonic Merge.
But, since the algorithm should be able to sort sequences of arbitrary size and a warp has a fixed size, to maintain the parallelization meaningful there are some extra steps.

REQUISITI DI LUNGHEZZA

3.1 Step 1: Bitonic Sort

The first step splits the input sequence logically into chunks of 128 elements, each of one being sorted by its own warp using a standard Bitonic sort.
For each chunk, there are 8 phases each with k-1 stages, where k is the current phase. For each stage, each thread of the warp performs 2 comparison, one ascending and one descending, for a total of 4 elements per thread. Using the **max()** and **min()** CUDA instructions, the comparisons are done without any need for conditions, ence avoiding the **warp divergence**. // SHARED MEMORY

3.2 Step 2: Bitonic Merge

After the first step, the data is now a group of ordinated subsequences that needs to be merged and again, by a warp.
Given two ordinated sequences of length t and a block of shared memory of length t, for each warp the merge will follow this scheme:

1. The first half of each sequence will be copied as the first *t/2* and the last *t/2* elements of the shared memory block (the first one will be reversed to obtain a bitonic sequence). It's worth notice that the first half is reversed while reading it from the global memory to allow a parallel access on the shared memory.

2. At this point the data on the shared memory is sorted by a warp similarly to the last phase of step 1. The first half of the sorted sequence is then the new first quarter of the merged sequence.

3. The *(t/2)-1th* elements of the starting sequences are then compared to choose from which one the second half will be loaded (again, in reverse order), the smallest will be chosen and the merge sort will be performed, with again the first half now stored in the shared memory being the second quarter of the output sequence.

4. Finally the remaining half of the sequence net yet used will be merged with the half still in the shared memory, completing the output sequence.

3.3 Step 3: Split

3.4 Step 4: Final Merge

4 Implementation

5 Evaluation

6 Conclusion