Parallel Shellsort Algorithm for Many-Core GPUs with CUDA

Chun-Yuan Lin, Chang Gung University, Taiwan

Wei Sheng Lee, National Tsing Hua University, Taiwan

Chuan Yi Tang, Providence University, Taiwan

ABSTRACT

Sorting is a classic algorithmic problem and its importance has led to the design and implementation of various sorting algorithms on many-core graphics processing units (GPUs). CUDPP Radix sort is the most efficient sorting on GPUs and GPU Sample sort is the best comparison-based sorting. Although the implementations of these algorithms are efficient, they either need an extra space for the data rearrangement or the atomic operation for the acceleration. Sorting applications usually deal with a large amount of data, thus the memory utilization is an important consideration. Furthermore, these sorting algorithms on GPUs without the atomic operation support can result in the performance degradation or fail to work. In this paper, an efficient implementation of a parallel shellsort algorithm, CUDA shellsort, is proposed for many-core GPUs with CUDA. Experimental results show that, on average, the performance of CUDA shellsort is nearly twice faster than GPU quicksort and 37% faster than Thrust mergesort under uniform distribution. Moreover, its performance is the same as GPU sample sort up to 32 million data elements, but only needs a constant space usage. CUDA shellsort is also robust over various data distributions and could be suitable for other many-core architectures.

Keywords: *Atomic Operation, CUDA, Graphics Processing Units (GPUs), In-Place Algorithm, Shellsort*

INTRODUCTION

Sorting is one of the widely studied algorithmic problems in the computer science (Cormen, Leiserson, Rivest, & Stein 2001; Martin, 1971). It appears in a wide variety of applications and evolves with various computational architectures (Akl, 1985). Hence, the efficient implementations of sorting algorithms profoundly influence the performance of those applications. As the computer architecture evolved, there is a continuing requirement to design efficient sorting algorithms to use the underlying hardware's computing power. Current high-end graphics processing units (GPUs), contain up to hundreds cores per chip, are very popular in the high performance computing community. GPU is a massively multi-threaded processor and expects the thousands of concurrent threads to fully utilize its computing power. The ease of access GPUs by using Compute Unified Device Architecture (CUDA) (NVIDIA, 2009), as opposite to graphic APIs, has made the supercomputing available to the mass. The advantages of the

DOI: 10.4018/jghpc.2012040101

computing power and memory bandwidth for modern GPUs have made porting applications on it become a very important issue.

Several excellent results for sorting algorithms on GPUs are obtained, including the internal and external sorting algorithms (Govindaraju, Gray, Kumar, & Manocha, 2006; Leischner, Osipov, & Sanders, 2010; Satish, Harris, & Garland, 2009). Among the internal sorting algorithms, CUDPP radix sort (Satish, Harris, & Garland, 2009) is currently the fastest sorting on GPUs and GPU sample sort (Leischner, Osipov, & Sanders, 2010) is considerably the fastest comparison-based sorting on GPUs. Comparing to CUDPP radix sort, GPU sample sort is more flexible and suitable for sorting the keys with a large size and is more robust over various data distributions, although it is slower than CUDPP radix sort on 32-bit keys. Both of them are the art of implementations for sorting algorithms on GPUs. However, the implementations for most of sorting algorithms on GPUs, including above algorithms, need either an extra space or the atomic operation support by GPUs.

In this paper, we address the above problems and propose a comparison-based sorting, shellsort, on GPUs, namely CUDA shellsort, that releases those constraints. In CUDA shellsort, we exploited the inherently embarrassing parallelism among each pass of shellsort and used the per-thread registers to accelerate the insertion operation on each subsequence (block) within a shellsort pass. Until the parallelism of a shellsort pass was insufficient (ex. <2048 concurrent threads), we switched it to the bitonic merge sort to sort blocks concurrently (ex. 2048-element blocks). After all blocks had sorted by the bitonic merge sort, an odd-even bitonic merge was used to rearrange the out-of-order data elements between two adjacent blocks. We demonstrated how to impose a block-wise structure on the parallel shellsort algorithm, bitonic merge sort, and odd-even bitonic merge. We also used registers to maintain a heap structure to avoid the shared memory bank conflict and result the efficient memory bandwidth utilization while keeping high GPU cores occupancy. The implementation of CUDA shellsort extracts the computing power of GPU cores, shared memory, and register file. CUDA shellsort needs $O(\log n)$ (n is a ratio of a size of input sequence and a given threshold) passes of memory access and is nearly the same as $O(\log_k n)$ passes of the multi-way approach suggested by the literature (Leischner, Osipov, & Sanders, 2010). In the experimental tests, under the uniform distribution, CUDA shellsort is, on average, 37% faster than Thrust mergesort and two times faster than GPU quicksort. The performance of CUDA shellsort is nearly the same as that of GPU sample sort up to 32 million data elements, but only needs a constant space usage. These results showed that CUDA shellsort is one of the fastest comparison-based sorting on GPUs and is robust over various data distributions evaluated as by the literature (Helman, Bader, & JaJa, 1998).

The rest of this paper is organized as follows. The next section reviews some related work for sorting algorithms on GPUs. The shellsort algorithm is then briefly introduced. The following section provides an overview of CUDA shellsort algorithm, followed by detailed implementations. An analysis of time and space complexities of CUDA shellsort is provided. The last section presents the experimental results for CUDA shellsort under various data distributions.

RELATED WORK

Sorting algorithm is the most wildly studied subject in the computer science and there is too much work done in the sorting problems to review it here. Hence, we focus on the parallel sorting algorithms that exploit the modern GPU architectures but do not discuss the sorting algorithms implemented by using graphic API, such as GPGPU.

GPU sample sort (Leischner, Osipov, & Sanders, 2010) is currently the state of the art comparison-based sorting on modern GPUs. It firstly randomly selects M-1 splitters from N input data elements, then sorts the M-1 split-