ELECTRODYNAMIC TETHER SATELLITE DYNAMICS MODELLING

by

KAIWEN CHEN

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
in Aeronautical Engineering

May 2018, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

KAIWEN CHEN

ELECTRODYNAMIC TETHER SATELLITE DYNAMICS MODELLING

by

KAIWEN CHEN

This is to certify that I have examined the above M.Sc. thesis

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by

the thesis examination committee have been made.

Prof. Xun HUANG, Thesis Supervisor

Prof. Christopher Yu-Hang CHAO, Head of Department

Department of Mechanical and Aerospace Engineering

29 May 2018

ACKNOWLEDGMENTS

I would never have completed this work without the help from many people. First of all, I thank my advisor, Professor Xun Huang, for his kind mentoring, helpful advices, and encouragement. I have got generous material support, well-condition laboratory experiences and engineering-oriented practice.

I thank the members of my thesis committee, Professor Xin Zhang for his insightful comments on improving this work.

I thank my colleagues in HKUST – Huanxian Bu, Zheng Liu, Wei Feng, Jiafeng Wu, Jingwen Guo, Jiaqi Song and many others. We have finished several deadlines and projects as a team. In daily life, we have been good friends and enjoy many activities in school campus. Without them, my graduate study in HKUST would not be so colorful.

I give my special thanks to Dr.Pok Wang KWAN who spent nearly a whole year instructing me on how to do research. He gave the project a lot of remarkable ideas using his sufficient engineering background and rich industry experience. When everything goes more and more complicated, his clear mind and intuition always work quite well. I learned how to proceed an engineering research project and how to analyse problems logically and incrementally from his face to face teaching.

I thank my parents for their support and encouragement.

Hope this thesis will not be the end of my academia, I hope the first half sentence is not just a hope.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ELECTRODYNAMIC TETHER SATELLITE DYNAMICS MODELLING

by

KAIWEN CHEN

Department of Mechanical and Aerospace Engineering

The Hong Kong University of Science and Technology

ABSTRACT

Abstract

CHAPTER 1

INTRODUCTION

MapReduce is a successful paradigm [**?**], originally proposed by Google, for the ease of distributed data processing on a large number of machines. In such a system, users specify two functions: (1) a *map* function to process an input key/value pair, and to generate a set of intermediate key/value pairs; (2) a *reduce* function to merge all intermediate key/value pairs associated with the same key. The system will automatically distribute and execute tasks on multiple machines [**?**, **?**] or multiple CPUs in a single machine [**?**]. Thus, this paradigm reduces the programming complexity so that developers can easily exploit the parallelism in the underlying computing resources for complex tasks. Encouraged by the success of CPU-based MapReduce systems, in particular, Phoenix [**?**], we develop Mars, a MapReduce system accelerated with graphics processors, or GPUs.

GPUs can be regarded as massively parallel processors with an order of magnitude higher computation power (in terms of number of floating point operations per second) and memory bandwidth than CPUs [**?**]. Moreover, the computational performance of GPUs is improving at a rate higher than that of CPUs. However, it is a challenging task to program GPUs for general-purpose computing applications, including those that MapReduce users are familiar with. Specifically, GPUs are traditionally designed as special-purpose co-processors for dedicated graphics rendering. As such, GPU cores are SIMD (Single-Instruction-Multiple-Data), which discourages complex control flows. Furthermore, GPU cores are virtualized, and threads are managed by the hardware. Finally, GPUs manage their own on-board device memory and require programmers to explicitly transfer data between the GPU memory and the main memory. Additionally, the architectural details of GPUs vary by vendors as well as by product releases, and programmer's access to these details is limited. All these factors make desirable a GPGPU (General Purpose Computation on GPUs) framework on which users can develop correct and efficient GPU programs easily.

Recently, several GPGPU programming frameworks have been introduced, such as NVIDIA CUDA [**?**], and AMD Brook+ [**?**]. These frameworks significantly improve the programmability of GPUs; nevertheless, their interfaces are vendor-specific and their hardware abstractions may be unsuitable for complex applications such as those running on MapReduce. Therefore, we propose Mars, a MapReduce framework to ease the programming of such applications on the GPU. Furthermore, the MapReduce framework of Mars enables the integration of GPU-accelerated code to distributed environment, like Hadoop, with the least effort. Our Mars system can run on multi-core CPUs (MarsCPU), on CUDA-enabled NVIDIA GPUs (MarsCUDA) or Brook+-enabled AMD GPUs (MarsBrook), or on a combination of a multi-core CPU and a GPU on a single machine. We further integrate Mars into Hadoop [**?**], an open-source CPU-based MapReduce system on a network of machines, which results in MarsHadoop, where each machine can utilize its GPU with MarsCUDA or MarsBrook in addition to its CPU with the original Hadoop. No matter what GPU and/or CPU Mars runs on, the API (Application Programming Interface) to the user is the same and is similar to that of existing CPU-based MapReduce systems.

Easing up GPU programming for MapReduce applications is the main goal of our work. However, a higher-level abstraction for programming, specifically MapReduce, comes at a price of performance. In particular, we identify the following three technical challenges in implementing Mars on GPUs. First, since MapReduce divides up a task by data, load imbalance is an inherent problem in utilizing the massive thread parallelism on the GPU, especially because GPU threads are managed by the hardware. Second, GPUs lack efficient global synchronization mechanisms. Threads in Map or Reduce tasks are likely to have write conflicts on the output buffer. While atomic operations are enabled in recent GPUs, the overhead of atomic operations would harm the scalability of massive GPU threads [**?**]. We consider a lock free scheme to minimize the synchronization overhead among GPU threads. Third, MapReduce applications are in general data-intensive and their result sizes are data-dependent. These two characteristics pose the following requirements on programming the GPU: (a) sufficient thread parallelism to hide the high latency and to utilize the high bandwidth of the device memory; (b) pre-allocation of output buffers in the device memory for bulk DMA transfers, as GPU memory allocation is done through the CPU before the GPU program starts.

With these challenges in mind, we develop Mars for GPUs of two most common programming interfaces - CUDA and Brook+. We focus on MarsCUDA, than MarsBrook, because in our implementation and evaluation, CUDA was more flexible and had a higher performance than Brook+ for MapReduce applications.

In MarsCUDA, the massive thread parallelism on the GPU is well utilized as each thread is automatically assigned a key/value pair to work on. In *Map*, the system evenly distributes key/value pairs to each thread. In *Reduce*, we develop a simple but effective skew handling scheme to re-distribute data evenly across all reduce tasks. To avoid write conflicts between threads, we adopt a lock-free scheme that guarantees the correctness of parallel execution with little synchronization overhead.

We extend our general design of Mars from a GPU-only MapReduce framework to a MapReduce system with GPU acceleration enabled. With this extension, Mars components can work stand-alone on a single platform, e.g., MarsCUDA on a CUDA-enabled GPU or MarsCPU on a multi-core CPU, as well as to work together to utilize multiple processors, e.g., a CPU and a GPU on a single machine. We also integrate Mars into Hadoop to enable GPU-acceleration for individual machines in a distributed environment.

We evaluated the performance of Mars in comparison with its CPU-based counterparts and the native implementation without MapReduce. Our results demonstrate the effectiveness of our GPU-oriented optimization strategies. On average, our MarsCUDA is 22 times faster than the CPU-based MapReduce, Phoenix [**?**], and is less than 3 times slower than the hand-tuned native CUDA implementation. Additionally, the applications developed with Mars had a code size reduction up to seven times, compared with hand-tuned native CUDA code.

Organization: The remainder of the thesis is organized as follows. We give a brief overview of GPUs, and review prior work on GPGPU and MapReduce in Chapter **??**. We present the design and implementation details of Mars in Chapter **??** and Chapter **??** respectively. We present the extension to multiple machines in Chapter **??**. In Chapter **??**, we present our experimental results. Finally, we conclude in Chapter **??**.