MapReduce On GPUs

Final Project Report

Longhai Cui，Yuanxun Zhang

# Abstract

Make the program concise and easy to maintain since we eliminate the process of computing the total space and location to write in the device. In the map stage, it used exact the same logic to go through all the input data twice – one for determine location to write in device and the other to calculate the actual keyword indexes. We no longer need separated Map-Count /Map function, instead we only need one Atomic-Map function without making the user aware the changes in the system library. Thanks to this skim we can eliminate a lot of parameters such as the pointer to store the start-point and prefix sum space of inter-Mediate and final results, making the whole map computation time half. This is really a huge improvement especially for the applications like String Match and Word Count with big data size where no reduce stage is needed and map stage solely takes most of the entire computing time.

# Introduction

The MapReduce framework proposed by Google has emerged to a popular programming model for processing massive amount of data in parallel. Compared with CPUs, GPUs have an order of magnitude higher computation power and memory bandwidth. So, it is a good approach to implement the MapReduce Framework on GPUs.

Our project is based on Mars: A MapReduce Framework on Graphics Processors, which is the first attempt to implement MapReduce on GPUs. But Mars is implemented on old GPU version and old CUDA SDK, which is less powerful than current GPU version, and old CUDA SDK may not be compatible to current CUDA SDK version.

So, the goal of our project is to: 1) migrate the Mars to our current GPU version and CUDA SDK; 2) and make some improvements of Mars framework.

# Background of Mars

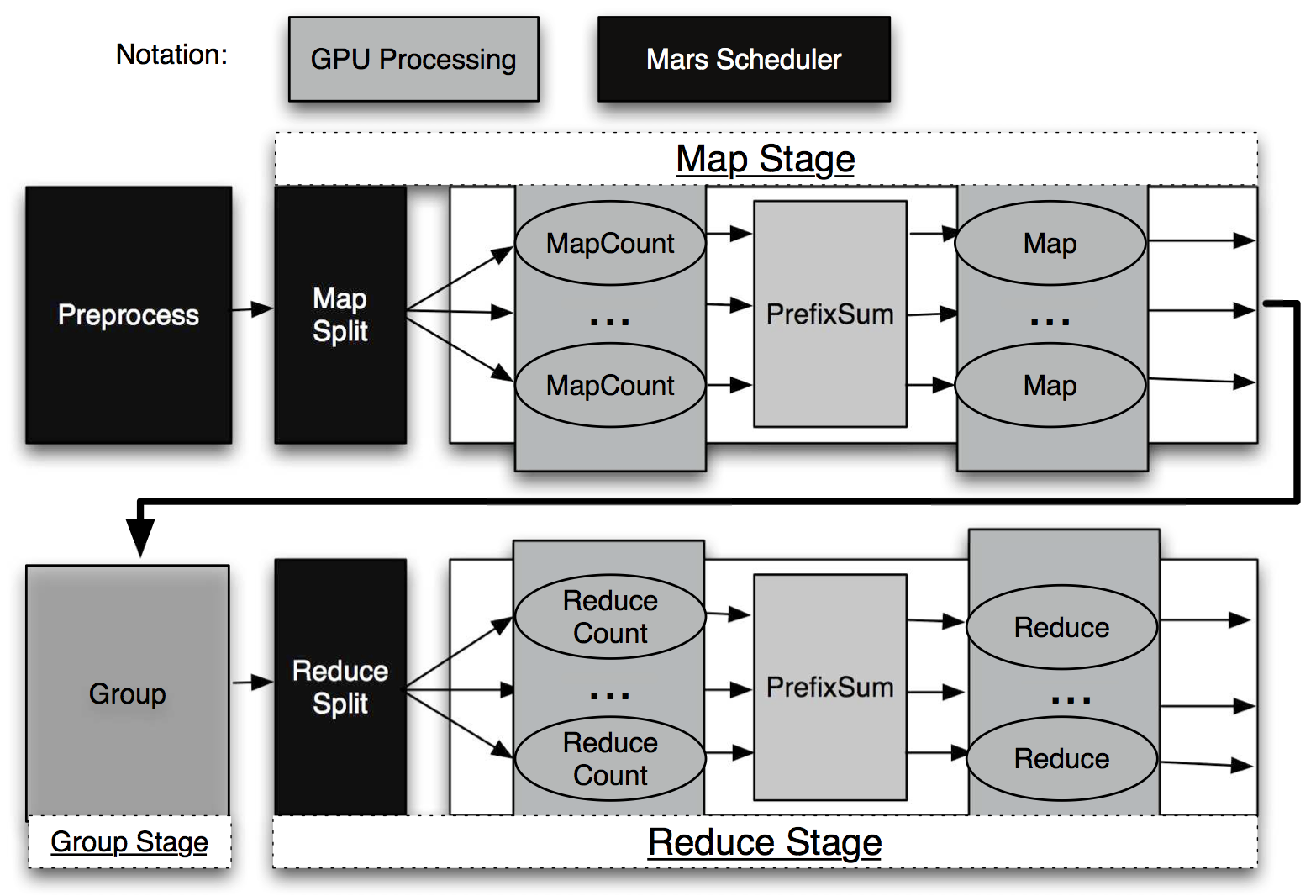
**1) APIs**

The goal of Mars’s APIs is to hide the programming complexity of GPU, and provide the APIs, which is similar to the current MapReduce framework. And the Mars provide two-types of APIs, the user-implement APIs, and the system-provided APIs.

|  |
| --- |
| **APIs of user-defined function** |
| void MAP\_COUNT(void \*key, void \*val, size\_t keySize, size\_t valSize)  *Counts the size of output results (in bytes) generated by the map function.* |
| void MAP(void \*key, void val, size\_t keySize, size\_t valSize)  *The map function. Each map task executes this function on the key/value pair.* |
| void REDUCE\_COUNT(void\* key, void\* vals, size\_t keySize, size\_t valCount)  *Counts the size of output results (in bytes) generated by the reduce function.* |
| void REDUCE(void\* key, void\* vals, size\_t keySize, size\_t valCount)  *The reduce function. Each reduce task executes this function on the key/value pairs with the same k* |
| **APIs of functions provided by runtime** |
| void EMIT\_INTERMEDIATE\_COUNT(size\_t keySize, size\_t valSize);  *Used in MAP\_COUNT to emit the key size and the value size of an intermediate result.* |
| void EMIT\_INTERMEDIATE(void\* key, void\* val, size\_t keySize, size\_t valSize);  *Used in MAP to emit an intermediate result.* |
| void EMIT\_COUNT(size\_t keySize, size\_t valSize);  *Used in RECORD\_COUNT to emit the key size and the value size of a final result.* |
| void EMIT(void \*key, void\* val, size\_t keySize, size\_t valSize);  *Used in REDUCE to emit a final result.* |
| Spec\_t\* GetDefaultConfiguration(char \*logFileName)  *Gets a default runtime configuration.* |
| void AddMapInputRecord(Configuration\_t\* conf, void\* key, void\* val, size\_t keySize, size\_t valSize); Adds a *key/value pair to the map input structure in the main memory.* |
| void MapReduce(Configuration\_t \* conf);  *Starts the main MapReduce procedure.* |
| void FinishMapReduce();  *Postprocessing such as logging when MapReduce is done* |

*Table 1: APIs of Mars*

**2) System workflow**



*Figure 1: workflow of Mars*

As the Figure 1 shown, there are two stages in Mars framework: Map stage and Reduce stage, which is similar to CPU-based MapReduce. (Note, the Mars Scheduler is run on the CPU processor). Before starting each stage, Mars initialize thread configuration on GPU. The *Map split* divides the input key/value pars into multiple chunks such that the number of chunks is equal to the number of threads.

In the Map stage, *MapCount* is used to emit the key size and value size according to the user defined MapCount function. The *Prefix sum* function is used to allocate the intermediate buffer according to the key sizes and value sizes, and determine the write position for each map. The *Map* finally generate emits the key/value pairs.

Before entering into the Reduce stage, the *Group Stage* is used to group and sort the same key emitted by Map Stage. Similar with Map stage, the Reduce stage will go through the Reduce Count, Prefix Sum, and Reduce operations to generate the final results.

**3) Two-step design and Lock-Free scheme**

Because, in size of output from map and reduce stages are unknown, and the old GPU version didn’t support atomic operation, so the write conflict may be occurred when multiple threads write results to the global or share memory. To address this problem, the Mars framework designs a Lock-Free scheme to solve the problem of written conflict and two-step design to solve the problem unknown the size of output.

* Two-step design

As the Figure 1 shown, the in the Map or Reduce stage, GPUs will execute the MapCount/ReduceCount and Map/Reduce two step operations to get final results. The MapCount/ReduceCount is used to calculate the size of key and value in order to get the size of memory allocated by GPUs, and then execute the Map operation to emit final result

* Lock-Free scheme

Based on the size of key and value generated by all MapCount tasks, the Prefix sum calculates the size of key and value allocated on device for the second Map step and determines the write position for each Map.

# Mars Framework Optimization

Based on previous analysis, we know the obvious drawback of Mars Framework: two step and lock-free to solve the problem we talked before. But it’s time consuming and needs more GPUs’ Memory and computation resources. And the main reason cause the Mars’s drawback is the GPUs limitation at that time. But currently, GPUs support the atomic operations, so we can use this new functionality to optimize the Mars framework.

In addition, Mars framework uses fixed number of threads according to size of input file without configuration. So, we attempt to use different number of threads to compare the results, and to the best number of threads for a certain application.

So, our optimizations are:

* Use one step design and atomic operation to solve the problem of written conflict.
* Increase the function to configure the number of threads.

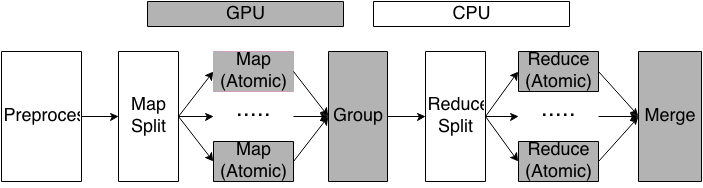
# Challenge

* Migration the Mars Framework to current CUDA version
* Deeply understand the Mars’s code
* How to implement a step design and atomic operations to solve the problem of written conflict.

# Implementation

**1) Workflow of our Mars**

As Figure 2 shown, our Mars framework is based one-step design comparing with original Mars framework. In Map or Reduce stage, the GPU only executes the Map or Reduce once. However, the original Mars will execute Map or Reduce twice. So, our framework can save lots of execution time and GPU resources.



*Figure 2: workflow of our Mars*

**2) APIs of our Mars**

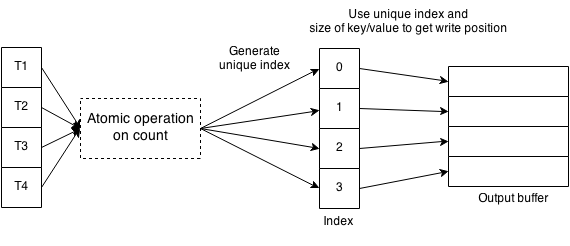
As the Table 2 shown, we decrease the MAP\_COUNT, REDUCE\_COUNT, EMIT\_INTERMEDIATE\_COUNT, and the EMIT\_COUNT in the original framework. And input parameters are same with original Mars. So, users can easily migrate their applications to our framework. Meanwhile, we framework is more easier for users to use, because the number of user-defined function is decreased.

|  |
| --- |
| **APIs of user-defined function** |
| void MAP\_ATOMIC(void \*key, void val, size\_t keySize, size\_t valSize)  *The map function. Each map task executes this function on the key/value pair.* |
| void REDUCE\_ATOMIC(void\* key, void\* vals, size\_t keySize, size\_t valCount)  *The reduce function. Each reduce task executes this function on the key/value pairs with the same k* |
| **APIs of functions provided by runtime** |
| void EMIT\_INTERMEDIATE\_ATOMIC(void\* key, void\* val, size\_t keySize, size\_t valSize);  *Used in MAP to emit an intermediate result.* |
| void EMIT\_ATOMIC(void \*key, void\* val, size\_t keySize, size\_t valSize);  *Used in REDUCE to emit a final result.* |
| Spec\_t\* GetDefaultConfiguration(char \*logFileName)  *Gets a default runtime configuration.* |
| void AddMapInputRecord(Configuration\_t\* conf, void\* key, void\* val, size\_t keySize, size\_t valSize); Adds a *key/value pair to the map input structure in the main memory.* |
| void MapReduce(Configuration\_t \* conf);  *Starts the main MapReduce procedure.* |
| void FinishMapReduce();  *Postprocessing such as logging when MapReduce is done* |

*Table 2: workflow of our Mars*

**3) Atomic operation in Map/Reduce operation.**

We use the atomic operation to solve the problem of written conflict. Also, because of this, our framework needs only on step in Map/Reduce.



*Figure 3: Atomic operations in Map*

As the figure shown, we use the atomic function in Map stage to generate the unique index, and according to the unique index and key/value to calculate the written position.

# Evaluation

Because our implementation only improves the execution efficiency in Map stage, so, we only compare the execution time in Map stage.

1) Comparing the Map execution time with different data size

*Figure 4: Evaluation with differ data size*

As figure 4 shown, our Mars framework is twice faster than original ones.

2) Comparing the Map execution time with different thread numbers

*Figure 5: Evaluation with differ thread number*

As figure 5 shown, our Mars framework is also faster than original ones. And as the number of thread increased, the execution time will decrease.

# Conclusion

Our Mars framework uses one step design and atomic operation to save the increase the execution efficiency and save GPUs resource. And our framework decrease the user APIs interface and don’t change the input parameters, so it’s easy for user to use and migrate application.