**Objective**

To provide a framework that can automatically run MapReduce jobs on a CPU + GPU cluster.

**Motivation**

- **Past decade**: CPU performance growth has stalled, while the GPU performance continues to grow.
- **Data explosion**: Internet, real-time devices (e.g., GPS).
- **Hadoop Streaming**: Use language of your choice.
- **Proposed directives**: In Hadoop streaming C code, on top of Hadoop streaming, that can be used for MapReduce jobs on a CPU + GPU cluster.

**Related Work**

- **MapReduce**: A distributed framework that splits computation into successive steps of two phases, Map and Reduce.
- **MARS**: MapReduce system for a single GPU, using a specialized programming model.
- **GPMR**: Distributed framework that employs GPUs in the network for MapReduce jobs. The input program is to be written in C++ and CUDA. GPU/GPU usage is not addressed.

**Challenges**

- **GPU Programming**: Traditional CPU programming approach doesn’t work on the GPU. GPUs require special, architecture-specific languages (e.g., CUDA, OpenCL).
- **Semantic Gap**: We can write a same set of Map and Reduce functions and let the underlying compiler and runtime system handle the execution on both CPUs and GPUs?
- **Scalability**: The framework must scale across multiple nodes, as well as across multiple GPUs in a node.

**Contributions**

- We propose a directive-based programming model, on top of Hadoop streaming, that can be used for programming MapReduce applications across CPUs and GPUs.
- We design and implement a source-to-source translation compiler that translates the program written with the above model into CUDA programs.
- We provide a runtime system for a multi-GPU execution. We integrate our translator and runtime system with the popular Hadoop framework.

**System Description**

**Example Map and Reduce Programs - WordCount**

```c
#include <stdio.h>

char prevWord[20], word[20];
int count = 0;
int val;
while (getWord(line, word) != -1) {
    char word[100];
    int one = 1;
    while (getWord(line, word) != -1) { // emit (word, 1) as KV pair
        printf("\%s\t\%d\n", word, one);
        prevWord[count] = word[0];
        int val = 1;
        printf("\%s\t\%d\n", word, one);
    }
}
```

**Compiler : Translation Scheme**

- Our source-to-source translation compiler translates the input Map and Combine (local reducer) programs into GPU equivalents. The translation scheme is developed with Cetus framework.
- The compiler generates CUDA codes with runtime library calls.
- The generated code is used to run a Map + Combine task on the GPU. In Hadoop terminology, one input split, or one input data file, is processed by one generated code instance.
- A runtime library is necessary to handle MapReduce specific functionalities e.g. intermediate KV pair sorting.

**Optimizations**

1. Bitonic sort implementation with indirection → reduction in data movement
2. Arithmetic optimisations using architecture specific warp-level function, **ballot**, for string comparisons
3. Bucket Aggregation → Using fast scanning methods for GPUs
4. Combine → Use only 1 thread in a warp for the combiner since it eliminates divergence. The relatively lesser parallelism in combiners motivated this choice.

**Hadoop Integration**

- Hadoop’s scheduler: Job Tracker daemon at the master node sends tasks to every slave. Each slave runs Task Tracker that governs task scheduling on the node’s GPU cores.
- We enhanced the Task Tracker design so that it schedules tasks on both CPUs and GPUs. Scheduling policy prefers GPUs over CPU slots.
- GPU drivers: Intermediate layer that communicates between the GPU and the Task Tracker.
- HDFS driver: Written in libHDFS, this driver enables the GPU to fetch the input files from HDFS.

**Performance Results of HeteroDoop on WordCount**

- **CPU Slots (threads) Used**
- **Performance Results**

**Acknowledgements**

This work was supported, in part, by the National Science Foundation under grants No. CNS-0720471, 0707931-CNS, 0833115-CCF and 0916817-CCF. We also thank NVIDIA Corporation for the hardware gift.