HeteroDoop : Automatically Exploiting CPUs and GPUs for Big Data MapReduce

Amit Sabne, Rudolf Eigenmann. School of ECE, Purdue University
{asabne, eigenman}@purdue.edu

PURDUE UNIVERSITY

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) etc.
- Frameworks for processing large, unstructured data sets : MapReduce, Storm, Dryad

Can the massive GPU parallelism be used?

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, uses 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. CPU/GPU combined 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** : Can we 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 based 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

Map

```
#pragma mapreduce mapper key(word) \\
value(one)  keylength(10)
while ((read = getline(&line, &nbytes,
stdin)) != -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);
}
```

Reduce

```
char prevWord[20], word[20];
int count = 0;
int val;
prevWord[0] = '\0';
#pragma mapreduce reducer key(prevWord) \\
value(count) key_in(word) value_in(val) \\
key_length(10) value_length(1)
while(scanf("%s", word) == 1) {
  if(strcmp(word, prevWord) == 0 ) {
    count += val;
  }
  else {
    if(prevWord[0] != '\0')
      printf("%s\t%d\n", prevWord, count);
    strcpy(prevWord, word);
    count = val;
  }
}
if(prevWord[0] != '\0')//handle last KV pair
 printf("%s\t%d\n", prevWord, count);
```

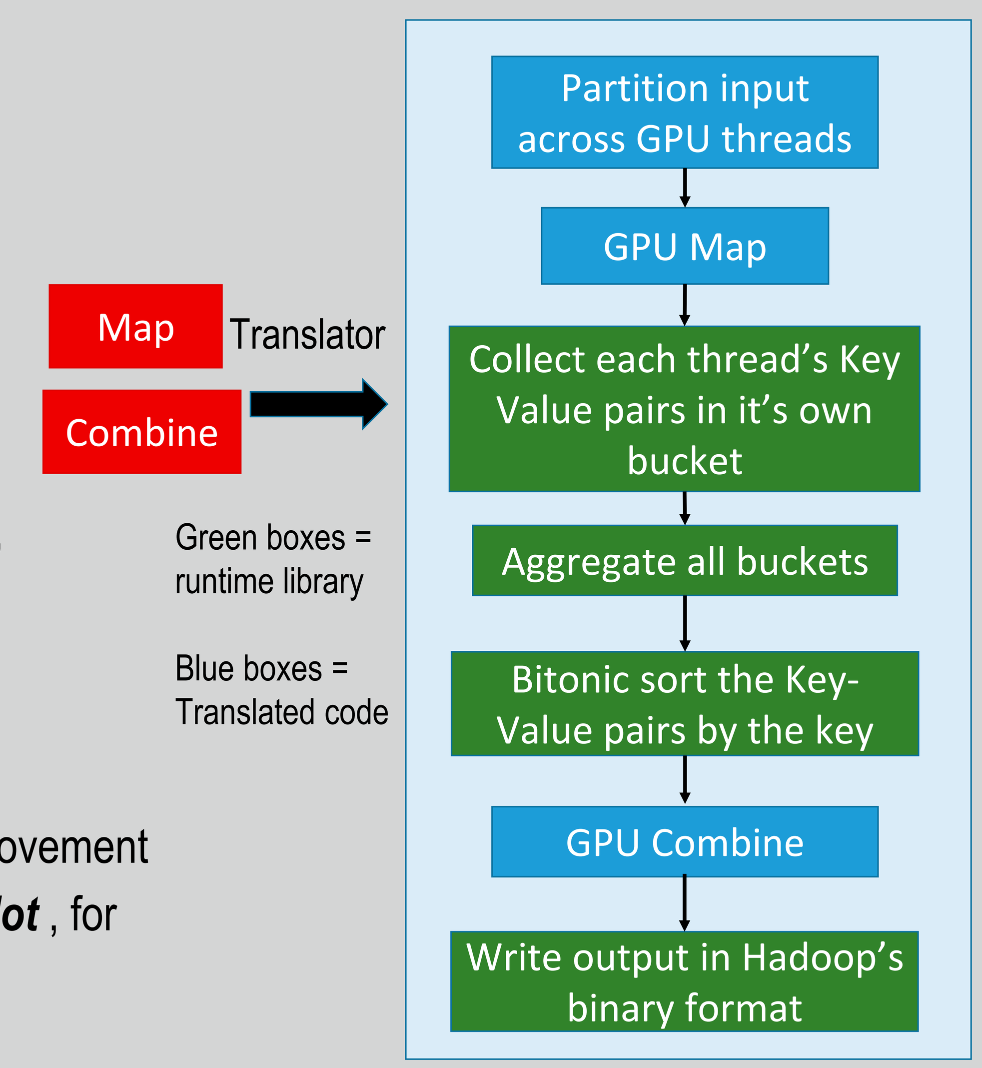
- Hadoop Streaming : Use language of your choice.
- Proposed directives : **Key observation – Map and Reduce codes contain the main code in a while loop**
- Proposed directives : In Hadoop streaming C code, on while loops
- Key and value lengths help optimize the space usage of the limited GPU device memories.

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

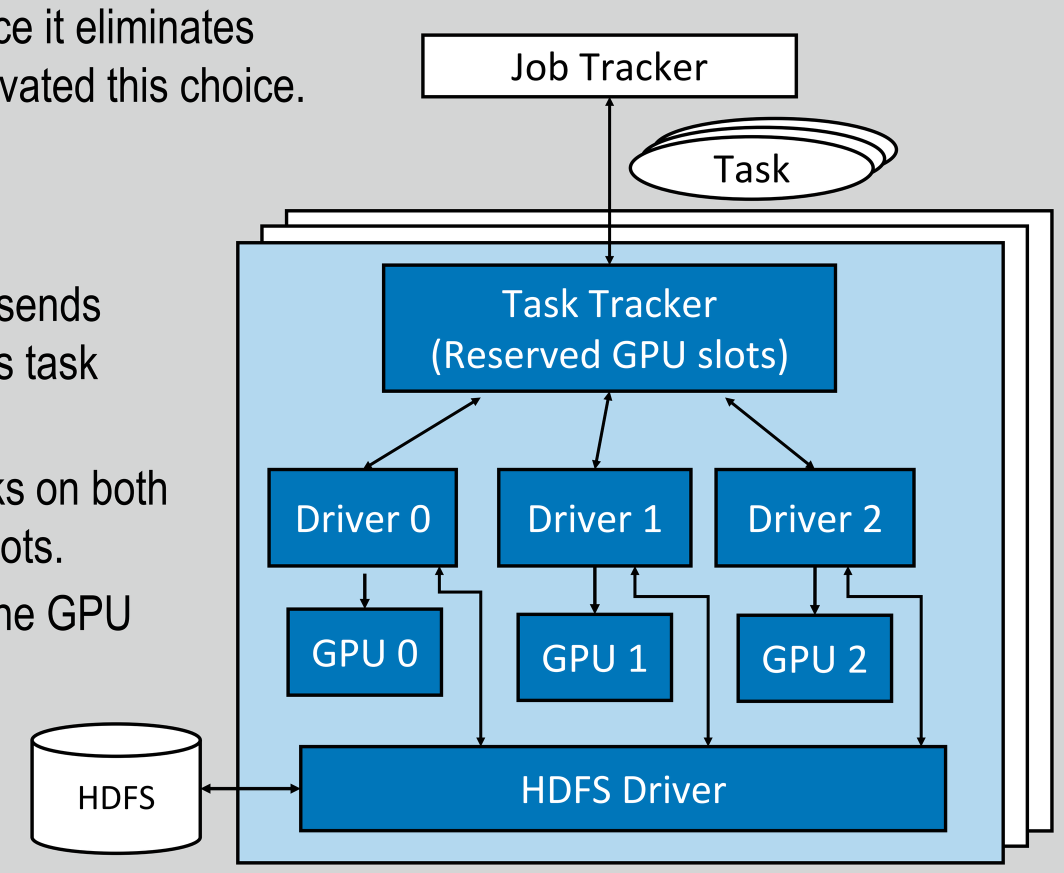
Optimizations :

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

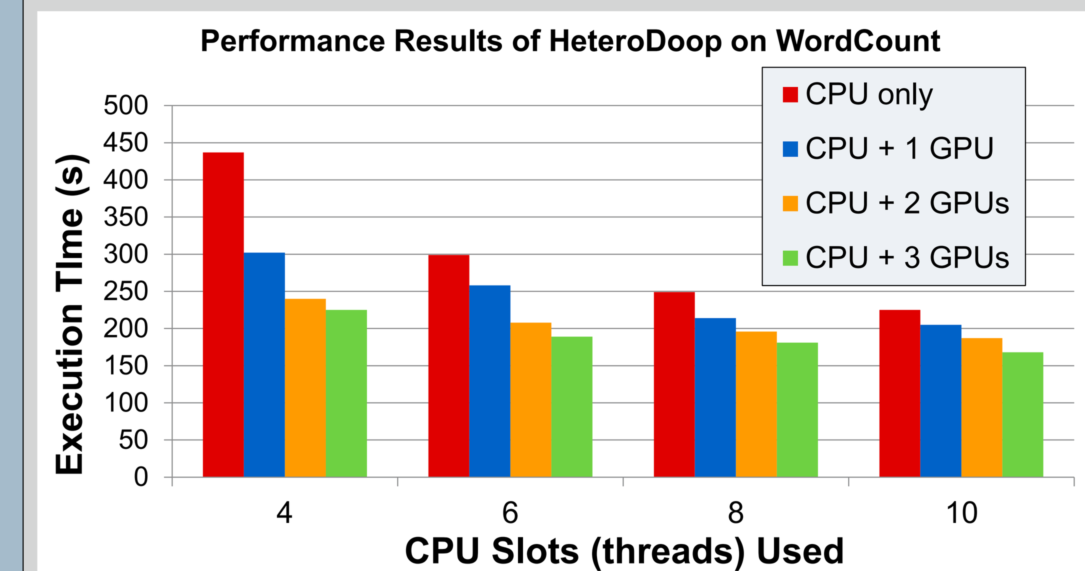


Hadoop Integration

- 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 CPU 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.

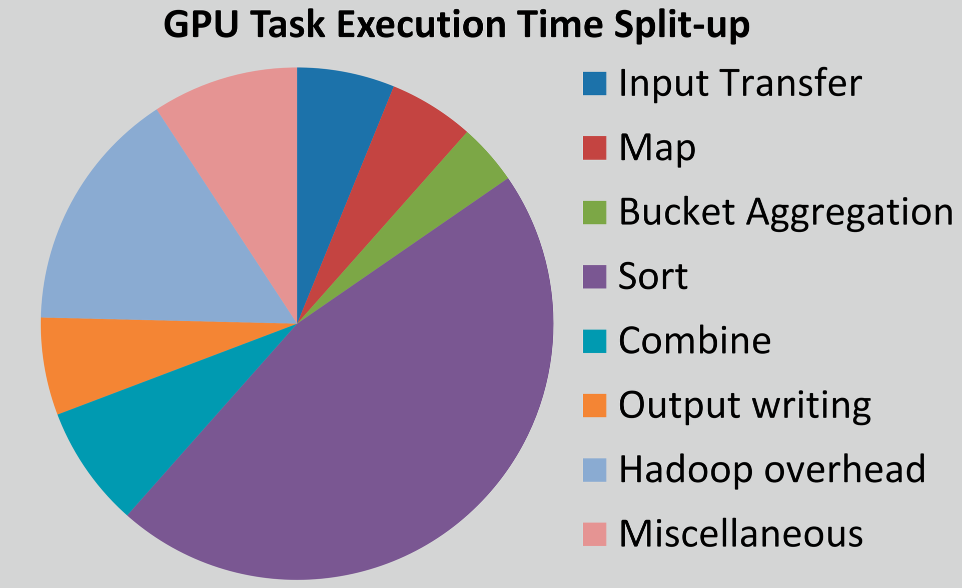


Preliminary Evaluation



Hadoop	Version 1.2.1
Input	Text files, 56 MB each, 57 GB
CPU	2 x Intel Xeon E5660, 12 cores overall
GPU	3 x Tesla M2090, 6 GB device memory
RAM	32 GB
No. of Nodes	16 slaves, 1 master
Network	QDR InfiniBand
HDFS	Block size : 64 MB, replication factor =1

GPU Task Execution Time Split-up



Discussion :
WordCount is one of the most basic MapReduce applications. The objective is to count the occurrence of every word in a big data set. As shown in the adjoining figure, WordCount spends most of the execution time in sorting, thereby limiting the achievable benefits. **Since Map is the embarrassingly parallel phase, applications with larger Map execution times would benefit furthermore from GPU usage**.

Next Steps

- Test more diverse MapReduce applications
- Enable offloading of global reducers on GPUs
- Explore different scheduling schemes for the heterogeneous environment

Acknowledgements

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