CSE 6230: High Performance Parallel Computing
Course Project Proposal for Fall 2013

Akash Gangil **902950416**

Lup Peng, Loke **903014478**

GPU Hash Sums

Overview
Data being sent over physical mediums are susceptible to interference from external sources. These interferences may change the value of the bits being sent and may render the entire data set to be useless at the receiver end. For example, a change in a single bit of a cryptographic ciphertext may cause a cascading effect over the rest of the message after that point.

The Cyclic Redundancy Check (CRC) is used to detect accidental changes to raw data being transferred over a medium/network. The sender will first perform a polynomial division on the data using a predefined polynomial to obtain the remainder, known as the checksum. This checksum is sent together with the data to the receiver. After receiving the data, the receiver performs a polynomial division on the data with the same predefined polynomial to obtain its own checksum. The receiver's checksum is then compared to the sender's checksum to verify that the data has been received with no errors. If the checksums do not match, it can be assumed that the data is corrupted and actions can be taken to rectify it.

Problem
With larger data sizes and faster network speeds, the calculation of the checksum becomes a bottleneck as the polynomial division is a non-trivial function. However, some CRC algorithms can be computed in parallel on different parts of the data, and then combined together at the end. A faster parallel CRC implementation will help overcome this potential bottleneck.

Thus the objective of this project is to exploit the parallelism potential in the CRC32-MPEG and CRC32C functions with CUDA on the Jinx Cluster in the the College of Computing at Georgia Tech.

Work Distribution
The work distribution of the project will be generally along the lines of:

Hash Sum	Allocation
CRC32C	Akash Gangil
CRC32-MPEG	Lup Peng, Loke

Project Flow

Preliminary Analysis:

During the initial phase, we will start by understanding the CRC32C and CRC32-MPEG hash sum functions. Their complexity will be analyzed and areas for potential parallelization will be identified

Phase 1: Determine Baseline CPU Performance

After analyzing the hash sums, we will develop a naive CPU implementation of the CRC32C and CRC32-MPEG hash sums using the Intel's PCLMULQDQ instruction.

Phase 2: Parallelization of CPU Implementation

We would be aiming to achieve a speedup on the previous CPU implementation by parallelizing it using techniques learnt in class.

Phase 3: Simple GPU Implementation

We would develop a naive GPU implementation which takes advantage of the large number of threads to compute the checksums parallely for different parts of sequence.

Phase 4: Optimizing GPU Implementation

Optimization of the naive GPU code will be done to develop a more scalable solution with better speedup times.

References:

- The CRC Pitstop – home of A Painless Guide to CRC Error Detection Algorithms, http://www.ross.net/crc/
- "Fast CRC Computationfor Generic Polynomials Using PCLMULQDQ", http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/fast-crc-computation-generic-polynomials-pclmulqdq-paper.html
- "Speeding up CRC32C computations with Intel CRC32 instruction"

  http://www.sciencedirect.com/science/article/pii/S002001901100319X#