Introduction to Parallel and Distributed Programing (ECE 451/ECE 566)

**Final Project**

Creating an inverted index for a search engine using OpenMP

December 26, 2013.

Group Members:

Theophilus Mensah (ID# 129006665)

Zhan Chen (ID# 155003497)

Professors:

Manish Parachar

Ivan Rodero

**Project Description:**

Today, tons of data is created on the web every minute, and information retrieval is becoming a daily routine as people jump on Google and other search engines on the regular basis to find information. The speed to which users of such search engines expect their queries to be returned is of a major concern to search engines in order to remain relevant in the search engine business.

To increase the rate of query return, an index of the word in the documents can be created in advance where search engines can quickly search for words and their respective documents. It is with this idea that creating an inverted index for search engines becomes very handy.

Creating an inverted index for a search engine is very time consuming, and so we wanted to parallelize this process with OpenMP in order to gain some speedup. The parallelization of this process will be implemented through a map and reduce process.

**Design & Architecture Description:**

The implementation of the inverted index for a search engine was achieved by using a map reduce paradigm. The program was implemented to accept the names of all the documents to be used in the creation of the inverted index as command line arguments.

In the map phase of the implementation, each document is divided into different segments for all threads created to work on in parallel. The document name was used as the ID for the mapping of each word. Each thread moves on to map its section of the next document until all the documents are mapped. Each thread the stored its copy of the results in a global variable to used in the reduce phase.

In the reduce phase, on two threads were used for simplicity. The result of the map phase is divided equally between the two reducers. Each reducer combines all it’s load into a single array and merges each duplicated word. A single copy of a duplicated word in the same document is kept, and a duplicated word in another documents is merge with a single word field and multiple ID fields for the different documents respectively. After the combination and merging, the final output is sorted and written to an output file.

**Experimental results & Performance Analysis:**

**Conclusion:**

This project was long and tiring, but was very intuitive and useful. 20% of the time was spent researching on the project topic and design. The gain in performance of the outcome was definitely worth the effort.