Doc Search

By Ivan Carrillo

CS/EE147

Lab Section 021

Presentation Link: https://youtu.be/OP4RmDrzMxU

Overview

The purpose of this project is to create an information retrieval system where documents are ranked based on an input of words. A matrix is used to represent a collection of documents and their words. The columns of the matrix represent documents, and the rows are the words. The data that is stored is the number of times a word appears in a document.

For this project I let the program randomly generate a query and a matrix. A query is a vector of words exactly the same as the one used for the documents.

The first part of the project deals with calculating the document frequency. The document frequency is the number of documents that contain a specific word. This is necessary to use the BM25 formula. The second part deals with the calculation of each document's score.

GPU Utilization and Implementation

We read the input matrix twice. Once for calculating the document frequency of each word, and once for calculating the score of each document. The input matrix of # words by # docs is first read from global memory by the row.

In the first part, the gpu calculates the document frequency by checking for the total amount of non-zero elements for each word / row. The kernel uses atomic adds to a shared memory block which is written to global memory at the end. The gpu calculates the document frequency of one word at a time, distributing the threads to check whether a document contains a word or not. Two blocks are used in strides to calculate the sum of how many times a word appears similar to the histogram program.

The second part consists of calculating the scores of each document in parallel. Each thread in a block will be responsible for calculating the score of one document. A for loop is used to calculate the score of the document word by word. Because the documents are represented column wise, and we are calculating the score one word (or row) at a time, the loading of memory should be coalesced.

The project is split into two parts. The first part will be the kernel for calculating the document frequency. The second part will calculate the scores of each document.

Running Code

Simply run make to create the project and then run ./search with optional parameters:

```
./search
./ search <# rows and columns>
./search <# rows> <# columns>
```

Examples

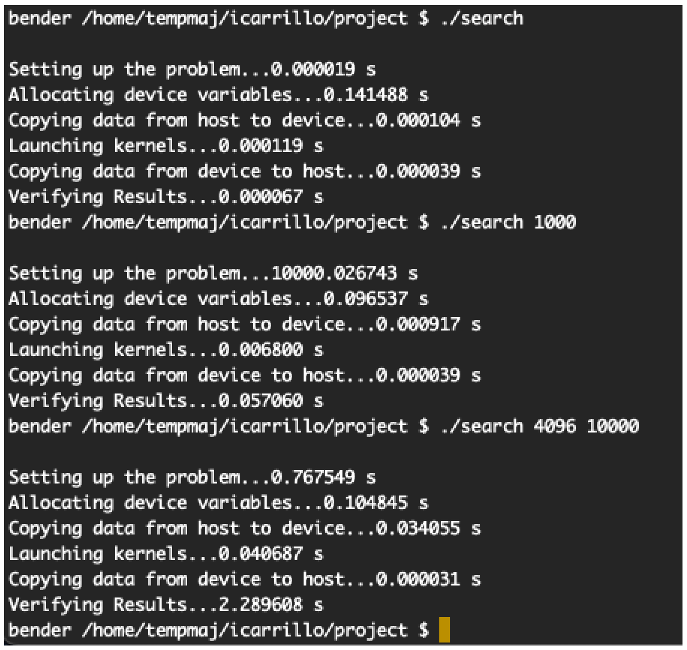
1. ./search
    a. Runs with default size

2. ./search 1000
    a. Here the program will make a matrix of 1000 x 1000
2. ./search 1000 x 2000
    a. Here the program will make a matrix of 1000 x 2000

Note: currently the max number of rows is 4096 but the number of columns can be larger

Results

The GPU performs a lot better than the CPU when using larger matrix sizes.



```
bender /home/tempmaj/icarrillo/project $ ./search

Setting up the problem...0.000019 s
Allocating device variables...0.141488 s
Copying data from host to device...0.000104 s
Launching kernels...0.000119 s
Copying data from device to host...0.000039 s
Verifying Results...0.000067 s
bender /home/tempmaj/icarrillo/project $ ./search 1000

Setting up the problem...10000.026743 s
Allocating device variables...0.096537 s
Copying data from host to device...0.000917 s
Launching kernels...0.006800 s
Copying data from device to host...0.000039 s
Verifying Results...0.057060 s
bender /home/tempmaj/icarrillo/project $ ./search 4096 10000

Setting up the problem...0.767549 s
Allocating device variables...0.104845 s
Copying data from host to device...0.034055 s
Launching kernels...0.040687 s
Copying data from device to host...0.000031 s
Verifying Results...2.289608 s
bender /home/tempmaj/icarrillo/project $
```

1000 x 1000 Matrix

- GPU: 0.006800s
- CPU: 0.057060s

4096 x 10,000 Matrix

- GPU: 0.104845s
- CPU: 2.289608s

Problems

- I was able to implement streams for calculating the document frequency of a word, but it would return an incorrect number of matrices larger than 2000.
- The matrix / the number of documents that can be processed is limited by the GPU memory. The program is not made to run in chunks which would be better for very large sets of data.
- The program is limited to 4096 words at a time.

Work

Ivan Carrillo - Everything