**File Management Application Using String Evaluation**

**By Matthew Staudigel**

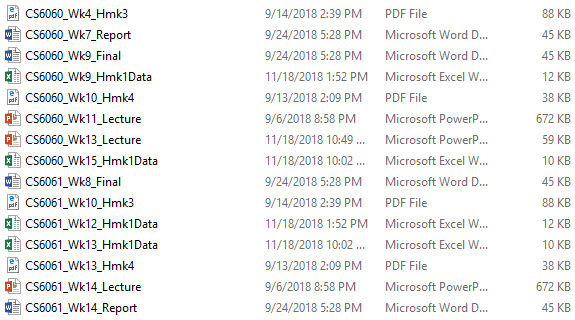
**University of Cincinnati**

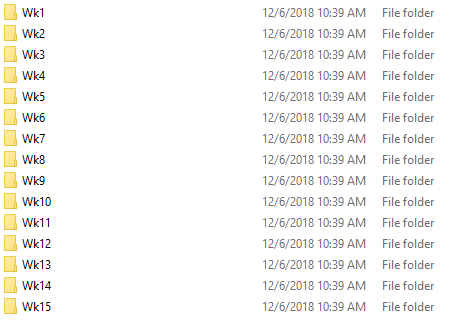
**Final Report**

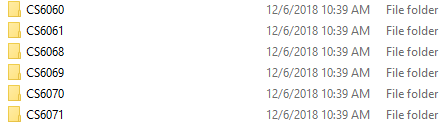
**Parallel Computing CS6068**

**Introduction**

In college, student often download and access hundreds, maybe even thousands of documents and files given to them by their professors for studying, learning, and other reasons for the class. Sometimes, it may be tedious organize the files into a well-formatted folder system in their hard drives in order to easily find the files they need, issues like files for each class, the week of the semester the file was given, or even the type of file (.pdf, .pptx, .docx, .xlsx, etc..). It may be easy to create a sequential program that will create a file system housing the documents in their corresponding classes, but what if a parallel program can do the same, and complete the task in less time?

The goal of this project is to create a file management and string evaluation program that would analyze a list of files gathered in a semester for several classes of a student. This program will analyze the characteristics of the file names found within a specified directory and determine the week that the file was supplied, the class that the file belongs to, and the type of file that the file is. The program will reorganize the files in the specified directory in a neat, well-formatted hierarchy of folders and classifications. However, the program will be completely run in parallel rather than sequential. The program will take into consideration the characteristics of the files, such as which class the document is from, what type of file it is, and what week the file was supplied in.



For this program, I want to use string evaluation of each of the file names of a specified directory. For a directory that contains thousands, and quite possibly even hundreds of thousands of files within it, a sequential program may take too long to determine the number of classes, weeks, and filetypes that exist within the directory. If a parallel program is developed to analyze a filename per thread, could the string analysis be sped up for the completion of the task? I believe that analyzing a filename string per thread of a GPU could speed up the file management task by a decent amount rather than analyzing each filename in sequential order.

**Design and Optimization Approach**

Beginning with the sequential version of the application, the program will have 3 stages. In the first stage, the targeted directory will be analyzed for the strings of all the files. The list of strings of file names will then be analyzed for the following characteristics: the class name, the week of the semester that the file was given, and the file type (the file extension) of the file. Each of these characteristics will then be used for the third stage. However, in order to focus on string evaluation, the files were given a generic file name format. The file will contain the class name at the beginning of the file name. Then, the week of the file that it is given by the professor will be identified by “Wk” and separated by the underscores seen below. Finally, the file extension will then be used to determine the file type of the file as well. These characteristics of the file will then be used during the final stage of the program.

*Example of a filename in the targeted directory:*

***CS6068****\_****Wk2****\_Quiz for week2****.pptx***

In the third and final stage, the organized hierarchy will be created. During this stage, the determined classes, weeks, and filetypes from the previous stage will be used to create the folder hierarchy. As the folder hierarchies are being created, the files that need to be stored in the file hierarchies are moved from the targeted directory to the new and final directory. Once this stage is complete, the time of completion is captured.

As you can see, there is much analysis on the list of filenames in the sequential program. For example, the list of filenames needs to be fully-traversed once, for the gathering of the filenames, again for the analysis of the filename characteristics, and once again for the moving of the files. This would yield the following complexity:

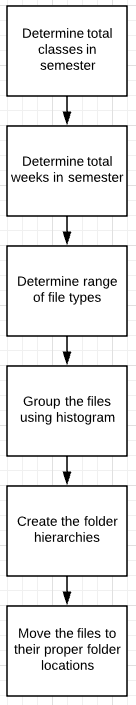
*Complexity of Sequential Program: n3*

The goal of the parallel version of this program is to minimize the complexity of the sequential program. The targeted step of the sequential program that the parallel program will be done concurrently is the second step, during the string analysis phase. Due to the concurrent step of the parallel program, the targeted complexity of the program in parallel would be the following:

*Targeted complexity of concurrent program: n2 + n*

Although not much of a speedup, it does appear that there will be a slight speed up during the parallel program’s execution rather than the sequential program’s execution because the string analysis will occur concurrently rather than sequentially.

The parallel program will operate in three stages as well, which can be seen in illustration below in the following flowchart.



First, the parallel program will obtain the strings of the filenames from the targeted directory sequentially using Python. Once the filenames are determined, the strings will then be sent to the GPU for string evaluation. Each ‘n’ thread will be given ‘n-1’ string filename. Depending on how many filenames there are (if n > 1024), one or more blocks will be assigned to the program.

Once the strings are sent to the device, each thread will analyze their assigned filename for the following file characteristics:

* Class name (ex. CS6068)
* Week of the semester (ex. Wk10)
* File type (.pdf, .xlsx, .docx, etc)

Once the individual characteristics are discovered, the results will be added to a histogram. The histogram will contain bins for all three characteristics. The results contained in the histogram will be return to the device, where the program will create the hierarchy based on the classes, weeks, and filetypes determined. The program will then sequentially move the files to their correct, corresponding directory location.

**Technical Challenges**

This section describes the technical challenges that were met during the development of the parallel version of the program.

* *Reading the string in the CUDA kernel after it was passed from the host*

During the development of the CUDA kernel of the program, there was a major issue attempting to read the character values of the string. In order to pass the list of strings from the host to the device, the list needs to be prepared using NumPy, a library used within Python. When the list is prepared for the GPU, it is converted into a NumPy array. A NumPy array can efficiently be passed to the GPU in comparison to a Python list.

However, when attempting to read the characters of the string in order to determine the file characteristics in the CUDA kernel, characters were being skipped over. At first, this appeared to be an issue related to the amount of memory allocated for the threads in terms of handling the size of the string. However, it resulted in how NumPy’s array structure is created. In order to read the string, every 4th character needed to be read versus characters iterating by every single character.

\_\_global\_\_ void DetermineCharacteristics(char\* File, char\* length, char\* className, char\* week)

{

int idx = threadIdx.x;

int stringCounter = 0;

char\* str = new char[\*length];

int i = 0;

bool isClass = true;

for(int j=0; j < (\*length)\*4; j = j+4) {

if (File[j] == '\_')

As you can see by the provided code above from the parallel version of the program, this is the CUDA kernel. In order to traverse the string correctly, the length of the string passed to the thread, then the kernel must loop every 4th character of the string in order to read the string correctly as it would be normally. This was a strange difficulty that was not foreseen in the design stage of the project.

* *Copying strings to the device from host takes a decent amount of time*.

Throughout development of the kernel itself, it was discovered that the action of copying strings to the GPU, or the device, can be a taxing demand in terms of time. If the strings that need to be copied to the device are quite long and extensive, the amount of time to copy them to the device will take longer compared to a smaller string size. This additional time it would take to move longer strings to the GPU will cut into the time it would take to complete the concurrent string analysis, essentially eliminating the computational time gained from parallelism.

For the sake of this project, however, the string lengths are kept quite small in order to focus the pickup of computational time due to concurrent programming.

**Application Performance Analysis and Project Results**

This section lays out the performance and project results of the file management and string evaluation program.

**System Requirements and Specifications Needed**

For this project, the following system components were used in order to execute both the sequential and parallel version of the program:

* Nvidia GPU, EVGA GeForce GTX 1060 Sc GPU
* Windows 10 Operating System
* Processor: Intel Core i7-7700K CPU, 4 Cores, 8 Logical Processors
* Total Physical Memory: 16.0 GB
* Total Virtual Memory: 26.8 GB
* GPU RAM: 8 GB
* CUDA-supported programming language: PyCuda, Python integrated with CUDA

**Project Testing, Results, and Analysis**

For testing of both the sequential and parallel versions of the programs, 3 different sizes of directories were created. These 3 different sizes would give a decent understanding as to whether the parallel version of the program can speed up the computation of string evaluation. The small directory contained 72 files, the medium directory contained 288 files, and the largest directory contained 576 files. Here are the results of the two versions of the program:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Small Directory - 72 Files** | **Medium Directory - 288 Files** | **Largest Directory - 576 Files** |
| **Sequential Version** | 0.196475 seconds | 0.8521421 seconds | 1.5502622 seconds |
| **Parallel Version** | 0.7709069 seconds | 1.015016 seconds | 2.052235 seconds |

Based on the results shown above, there does not appear to be a large amount of speedup yielded by the parallel version of the program. I believe there may be a major reason why there is no speedup with the parallel program.

* *The time it took to copy the strings to the device eliminated any speedup in computation.*

As mentioned during the Technical Challenges section, copying the strings from the host to the device seemed to eliminate any speedup in computation involving concurrent string evaluation. Regardless of the size of the strings themselves, the volume of strings seemed to cause the program to slow down in order to efficiently copy the memory to the device, or the GPU. During the sequential version of the program, there is no copying of the filename strings from one location to another, which allows the program to operate without the additional step of the parallel program. This was a design flaw that was not foreseen during the design stages of the project.

**Future Potential Improvements and Project Changes**

Throughout the development of the programs, it became apparent that the computational complexity of the design does not generate a strong computational speedup. String evaluation in a concurrent program, particularly with the design of the project, does not seem to be a candidate for a parallel version of the program. For this project to be a viable candidate for a concurrent version of the program, some key aspects of the project need to be altered.

Instead of evaluating the file name for file characteristics, a better file management program would be to analyze the contents of each individual file. File contents can be quite large in volume and size. Evaluating the contents of a file in a sequential program would take quite a long time to complete. However, if this version of the program was parallelized, allowing an individual thread to analyze the contents of a file in a targeted directory, the computation time would be cut down dramatically in comparison to the sequential version of the program.

**Division of Work**

The team for this project consisted of myself only. Here is a quick explanation of the work conducted over the past 4 weeks.

For the week of November 12th, several types of files needed to be created. These files consisted of actual files obtained from my semester this year. The files consisted of a large range of files, consisting of documents, spreadsheets, pdf files, and more. In addition to the creation of the files, the directories that would be used for testing needed to be created as well. For testing, a small directory, and medium directory and a large directory were created to show the differences in computational time of the sequential and parallel versions of the program. In addition to that, I obtained a decent understanding of directory creation and file management with Python. With the completion of both the directory creation and file management, along with the directories created, the sequential version of the program was finished. For the following week, the parallel program began development. However, as described in the Technical Challenges section, the issue with reading the strings within the CUDA kernel was not foreseen, causing a backup in development. Investigation of this issue carried over to the week of November 26th, when the solution to this issue was completed. It was also during this week that the final presentation and report was started.

**Bibliography**

1. S. Kouzinopoulos, Charalampos & Margaritis, Konstantinos G.. (2009). String Matching on a multicore GPU using CUDA. 14-18. 10.1109/PCI.2009.47.

**Code Appendix**

**Presentation Slides**



