Parallel Data Mining
Using Multi-Core Computing

New Mexico

Supercomputing Challenge

Final Report

April 4, 2012

Team 63

Los Alamos High School

Team Members

Samuel Wang

Daniel Wang

Xiaoyu Deng

Ben Liu

Teachers

Lee Goodwin

Wyatt Dumas

Project Mentors

HB Chen

HsingHui Liu

Hailin Deng

Table of Contents

Executive Summary

Efficient parallel algorithms and implementation techniques are the key to meeting the scale and performance requirements entailed in such scientific data analyses. For the large volumes of datasets in every area, we have come to rely more and more on search engines. Multi-core parallel processing is the only effective way to speed up a search engine.

In recent times, CPU clock speeds have stagnated and manufacturers have shifted their focus to increasing core counts. This is problematic for programmers because the standard single-threaded code will not automatic run faster as a result of those extra cores. To speed up the search result, we are using multi-threading Java program to trigger the multi-core parallel processing.

Using the following scenario as a comparative example, it is easy to see why parallel processing is becoming the preferred supercomputing method. If you were preparing ice cream sundaes for yourself and nine friends, you would need ten bowls, ten scoops of ice cream, ten types of chocolate syrup, and ten cherries. Working alone, you would take ten bowls from the cupboard and line them up on the counter. Then, you would place one scoop of ice cream in each bowl, apply syrup on each scoop, and place a cherry on top of each dessert. This method of preparing sundaes would be comparable to vector processing. To get the job done more quickly, you could have some friends help you in a parallel processing method. If two people prepared the sundaes, the process would be twice as fast; with five it would be five times as fast; and so on.

On the other hand, assume that five people will not fit in your small kitchen, so therefore it would be easier to use vector processing and prepare all ten sundaes yourself. This same analogy holds true with supercomputing. Some researchers prefer vector computing because their calculations cannot be readily distributed among the many processors on parallel supercomputers. But, if a researcher needs a supercomputer that calculates trillions of operations per second, parallel processors are preferred—even though programming for the parallel supercomputer is usually more complex.

We will be able to achieve our goal by reason of Amdahl’s Law. It states that the more threads or processors you have to accomplish a set of tasks, the faster the tasks will be completed. This is because tasks can be split among the different processors. Increased multi-threading of the program is an extremely important factor that must be continually exploited by us, as the program contains the ability to split tasks down into a number of threads, all of which are parallelized with each other to maximize speed and efficiency of a task.

In our program, we use data parallelism strategy for partitioning work among threads. We partition the data into small chunks, execute those chunks in parallel via multithreading, and assemble the results as they become available. In our result, we can clearly see the program that divides the data and run on the threading, significantly reduced the search time before it hits the bottleneck due to I/O performance limitation, requirement of greater dataset, and switch context involvement.

1. Introduction

The overall purpose of our project is to extract knowledge from a data set in a human-understandable structure and to discover new patterns from large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics and database systems. We will create a search engine that is capable of utilizing multi-thread parallel processing to locate and read out specified files within large databases. With our completed program, we will be able to speed up search time on multi-platformed computers for a desired file.

The manual extraction of patterns from data has occurred for centuries. As data sets have grown in size and complexity, direct hands-on data analysis has increasingly been augmented with indirect, automatic data processing. This has been aided by other discoveries in computer science such as neural networks, cluster analysis, genetic algorithms (1950s), decision trees (1960s), and support vector machines (1990s). Data mining is the process of applying these methods to data, with the intention of uncovering hidden patterns in large data sets. It bridges the gap from applied statistics and artificial intelligence (which usually provide the mathematical background) to database management by exploiting the way data is stored and indexed in databases to execute the actual learning and discovering algorithms more efficiently, allowing such methods to be applied to larger data sets. The need for speeding up data mining is a natural consequence of the huge size of real-world databases and data warehouses. Real-world database systems are large with respect to at least three dimensions.

Besides the speed of search, the accuracy is also a problem that needs to be addressed. A good example would be Internet searching. As the Internet expanded to Terabyte or even Petabyte sizes, search engines began to list more and more results that may not necessarily be relevant to the desired results. An example of a search engine is Google, which finds information based on relevance; however, the relevancy is drawn from how many times users access a particular search result after typing in a query. Another example would be Windows search. If one were to search: “cat,” the computer search might give results that are not related to a “cat.” The engine could turn up with words like “catch,” “category,” and “catalogue,” in other words, not what we were looking for.

The rest of this report is structured as follows. Section 2 presents an overview of the application domain, relationship between parallel computing and data mining, the use of parallel computing on supercomputer, and the method we used for developing this search engine. In this section, we also developed several testing methods and discussed its results. Section 4 is conclusion and future work and section 5 recommendation. In the end of this report we enclosed acknowledgements and references.

2. Descriptions

Many items would benefit from the search method that we are attempting to create. These include hospital patient databases, scientific databases, and website databases. Popular search engines, such as Google and Yahoo, are not “accurate” since they give results based on popularity instead of relevance. These engines regulate huge databases, meaning that each search is very inaccurate. These databases require the most efficient search possible. This is what we wanted to achieve with our program.

2.1 Our Application Domains

a. **Analytics** – Our search engine can be used to solve problems in business and industry analytics. For example*:*

Portfolio analysis:

Banks and lending agencies have collections of accounts of varying value and risk. Their accounts may differ by the social status (wealthy, middle-class, poor, etc.) of the holder, the geographical location, its net value, and many other factors. The lender must balance the return on the loan with the risk of default for each loan. The question is then how to evaluate the portfolio as a whole.

The least risk loan may be to the very wealthy, but there are a very limited number of wealthy people. On the other hand there are many poor that can be lent to, but at greater risk. Some balance must be struck that maximizes return and minimizes risk. The analytics solution may combine time series analysis, with many other issues in order to make decisions on when to lend money to these different borrower segments, or decisions on the interest rate charged to members of a portfolio segment to cover any losses among members in that segment.

Bio-informatics – Our search engine could also help to reducing the complexity of data in large bio-informatics database and of discovering meaningful and useful patterns and relationships in data. For example:

Analysis of mutation in cancer

In cancer, the genomes of affected cells are rearranged in complex or even unpredictable ways. Massive sequencing efforts are used to identify previously unknown point mutations in a variety of genes in cancer.

b. **Business intelligence** – Our search engine could also be used in business intelligence (BI) area. This search engine is able to identifying, extracting, and analyzing business data, such as sales revenue by products and/or departments, or by associated costs and incomes. For example:

Hotels marketing:

A hotel franchise uses BI analytical applications to compile statistics on average occupancy and average room rate to determine revenue generated per room. It also gathers statistics on market share and data from customer surveys from each hotel to determine its competitive position in various markets. Such trends can be analyzed year by year, month by month and day by day, giving the corporation a picture of how each individual hotel is faring.

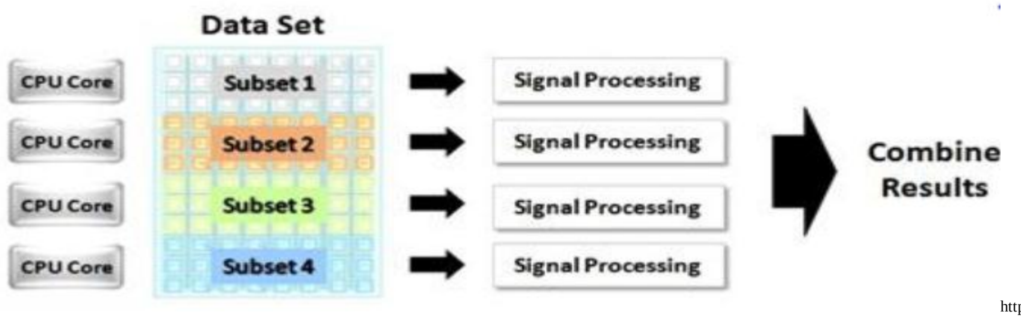
c. **Others** –Along with the aforementioned areas, parallel processing can also be applied to areas like data analytics, data warehouses, decision support systems, drug discoveries, predictive analytics, and web mining.

2.2 Parallelism and Data Mining

Parallel computing can use two parallelisms: Data parallelism and Control Parallelism. Our project uses Data Parallelism**.**

Data parallelism refers to the execution of the same operation or instruction on multiple large data subsets at the same time. This is in contrast to control parallelism (a.k.a. operation parallelism or task parallelism), which refers to the concurrent execution of multiple operations or instructions. The control parallelism is when data is split into the amount of processors the computer has, and then uses the processors to execute the tasks. This process speeds up processing by simply accomplishing two or more programs at a time.

Our search engine is designed to split up into subsets of data and each processor does a different subset, then combining the results to get the final product. This increases processing speed by decreasing the time taken to execute each task.



http://zone.ni.com/c
ms/images/devzone/tut/SmallDataParallelism.JPG

Figure 1: Data Parallelism illustration

Since our target interest is solving large dataset, we have several reasons to choose Data Parallelism over Control Parallelism:

1. Data parallelism is more automated. The control flow of a data parallel program is essentially the same as the control flow of a serial program in the sense that the data is parallel. Hence, a lot of previously written serial code can be reused for data parallel processing. This simplifies programming and leads to a significantly smaller development time than the one associated with control parallel programming.
2. Data parallelism has better scalability for large databases than control parallelism. In most database applications, including data mining, the amount of data can increase arbitrarily fast, while the number of lines of code typically increases at a much slower rate. To put it in simple terms, the more data is available, the more opportunity to exploit data parallelism. In principle we can add to the system a number of processing elements proportionally to the amount of data increase, to keep the response time nearly constant (i.e., linear scale-up).
3. Data parallelism addresses the problem of very large data bases, whereas control parallelism address the problem of very large search spaces (e.g., very many candidate rules). Note that data and control parallelism are not mutually exclusive. If a large enough number of processors are available, both types of parallelism can be exploited at the same time, which can greatly speed up the execution of data mining algorithms.

2. 3 Supercomputer vs. Desktop in Parallel Processing

The idea of our search engine is focusing on accurate search result and speed. To speed up a search time is related to a computer’s hardware configuration and its parallel processing capability.

Supercomputers, the world's largest and fastest computers, are primarily used for complex scientific calculations. The parts of a supercomputer are comparable to those of a desktop computer: they both contain hard drives, memory, and processors.

The supercomputer's large number of processors, enormous disk storage, and substantial memory greatly increase the power and speed of the machine. Although desktop computers can perform millions of floating-point operations per second (megaflops), supercomputers can perform at speeds of billions of operations per second (gigaflops) and trillions of operations per second (teraflops). Applications that use parallel processing are able to solve computational problems by simultaneously using multiple processors.

2.4 Method

We are using a runnable Java class with different levels of threading that can run parallel threading. While running the program, we will also monitor CPU and RAM activities in the Task Manager to ensure that our search engine program exploits the use of threading and that it runs on multiple CPU's. We have also used several graphics in this report to show the real time behavior or the computer when running the program.

A. **Java Programming**

We use Java programming to code the parallel search engine, because Java threads are run by OS threads, so multiple threads run on different cores by default. Also a Java Thread object wraps around an actual thread of execution. It effectively defines how the task is to be executed

B. **Multi-core CPU** We are using Intel quad-core processor and using Intel Hyperthreading Tool to double up the amount of cores for our system. Since the parallel processing is the simultaneous use of more than one CPU or processor core to carry out a program or multiple computational threads. The total of eight cores of our testing system will able to speeding up computing because it simultaneously uses more than one CPU/processor core to execute a program or multiple computational threads. It also drastically reduces the amount of time to compute a problem or carrying out larger problem in the same time.

Our program is using multiple computational threads to trigger parallel processing to makes programs run faster because there are more engines (CPUs or Cores) running it. Most computers have just one CPU, but some models have several, and multi-core processor chips are becoming the trend. The IBM Sequoia supercomputer will be constructed using thousands of CPUs connected together.

C. **Designing Parallelism into the Data Integration Models**

Parallel processing is the ability to break large data integration processes and/or data into smaller pieces that are run in parallel, thereby reducing overall runtime, as demonstrated in Figure 2.

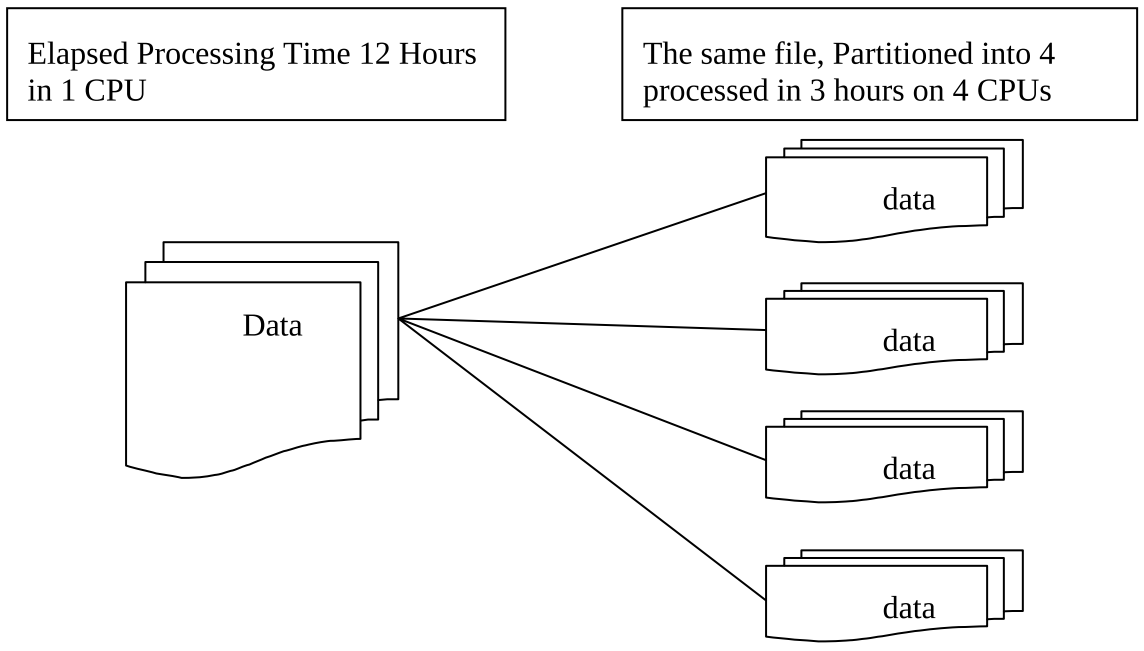

Elapsed Processing Time 12 Hours in 1 CPU
The same file, Partitioned into 4 processed in 3 hours on 4 CPUs
Data
data
data
data
data

Figure 2: Partitioning a Dataset

d. **Multi-Threading**

Our program is using multi-threading to trigger the parallel processing. In computer science, a thread of execution is the smallest unit of processing that can be scheduled by an operating system. The implementation of threads and processes differs from one operating system to another, but in most cases, a thread is contained inside a process. Multiple threads can exist within the same process and share assets such as memory, while different processes do not share these resources.

On a multiprocessor or our multi-core testing system, the threads or tasks will actually run at the same time, with each processor or core running a particular thread or task.

Example of Multi-threading

The graph below shows the advantage of multithreading. From the graph, Application #1, Application #2, and Java Virtual Machine are Tasks. The multithread is enabled under Java

Virtual Machine and it shares the memory. It is just like you can have a word processor that prints a document using a background thread, but at the same time another thread is running that accepts user input, so that you can type up a new document.

If we were dealing with an application that uses only one thread, then the application would only be able to do one thing at a time – so printing and responding to user input at the same time would not be possible in a single threaded application.

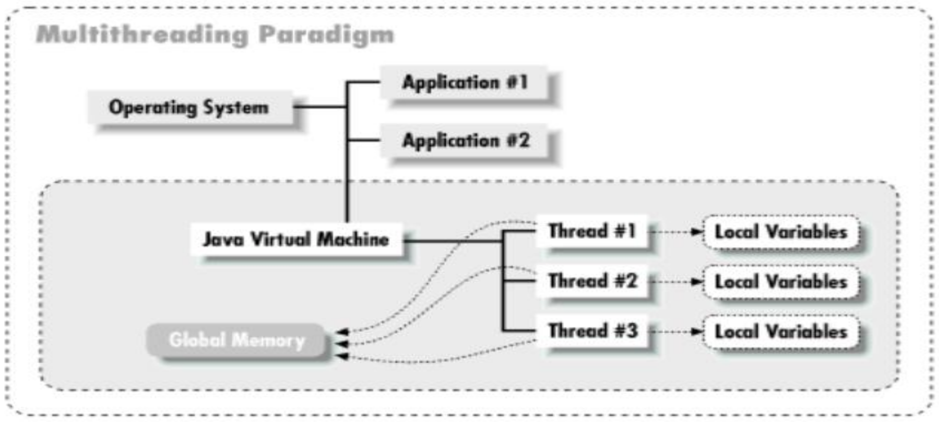


Figure 3: Multithreading Paradigm

e. Our Tools and System Environments

Tool:

Task Manager - We used to monitor the CPU and the memory

Eclipse IDE for Java Developers - We used to write Java and run its program

Intel Hyperthreading Tool - “doubles” amount of cores

Environment:

Processor - Intel Xeon quadcore @ 2.27GHz

Advantages: Speed and high number of cores (better for multitasking)

Disadvantages: None

Operating System - Windows 7 Ultimate x64

Memory - 6GB DDR3

Advantages: High bandwidth (DDR3), high amount of RAM

Disadvantages: None

2.5 Research and Code

a. *Depth-first search*

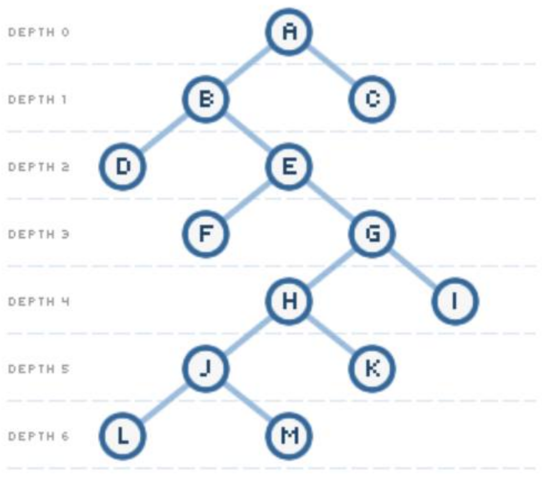


Figure 4: DFS tree

- Depth first search works by taking a node, checking its neighbors, expanding the first node it finds among the neighbors, checking if that expanded node is our destination, and if not, continue exploring more nodes.

<u>DFS Algorithm</u>

```
dfs (File A)
{
   list neighbors of A to an array
    for each neighbors of A
        if neighbor is directory
           dfs(B);                    //recursivly call function dfs

                  else
           add neighbor to a linked list
 }
```

<u>Coding :</u>

This coding represents the *search directory* recursive function.

see appendix A

b. *Using several Java class to search specific flies - FileReader, BufferReader, Tokenizer*

- ***FileReader* -** The FileReader class makes it possible to read the contents of a file as a stream of characters.
- ***BufferReader* -** Read text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.
- ***String Tokenizer*** - The string tokenizer class allows an application to break a string into tokens.

<u>Word search Algorithm:</u>

```
searchword (File A, searchname)
    {
            read a content of a file
            read text from a character (line)
            while there has a line
            {
                    break line to token
                    while there has a token
                    {
                            if token and searchword are same
                            {
                                    print line;
                                    print path;
                            }
                    }}}
```

<u>Our Coding:</u>

This coding represents the *search word* function that searches for key words in the text files.

see Appendix B

c. Runnable and Thread

Runnable class - The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

thread class - A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Four thread Algorithm -

```
create thread1 with Runnable interface
    define a method call run
    for each i from 0 -> ¼ size
        get the file and call function searchword
create thread2 with Runnable interface
    define a method call run
    for each i from half of the 1/4 -> ½ size
        get the file and call function searchword
create thread3 with Runnable interface
    define a method call run
    for each i from half of the ½ -> ¾ size
        get the file and call function searchword
create thread4 with Runnable interface
    define a method call run
    for each i from half of the 3/4 -> size
        get the file and call function searchword
thread1 start
thread2 start
thread3 start
thread4 start
```

<u>Our Coding</u>

This coding represents the *Java Runnable interface* and the *Java Run class.*

see Appendix C

d. Flow Chart:

- Chart1 - basic idea:

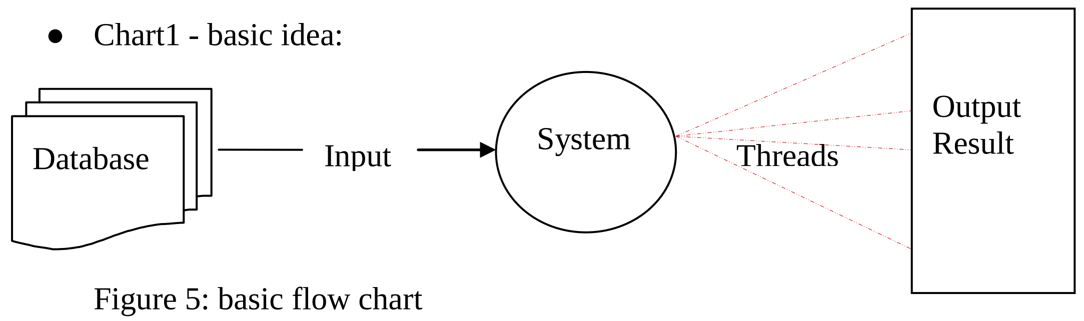


Figure 5: basic flow chart

- Chart2 - full flow:

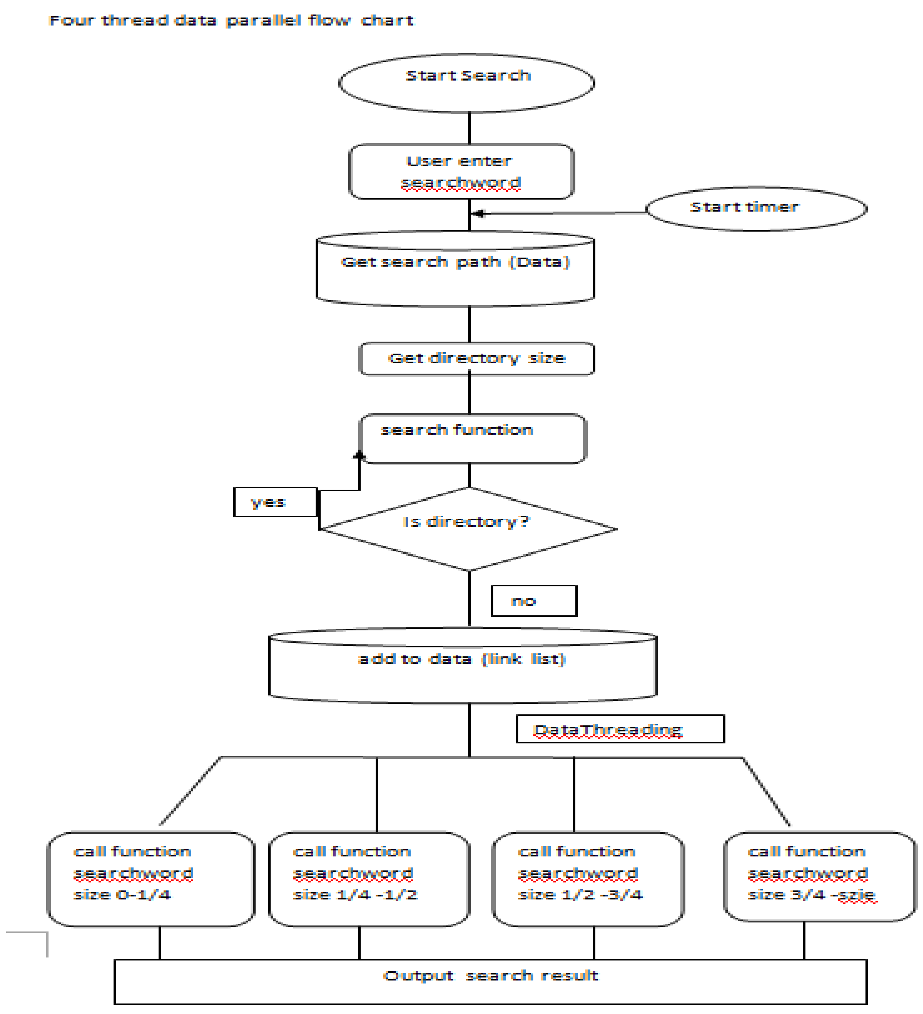

Four thread data parallel flow chart
Start Search
User enter searchword
Start timer
Get search path (Data)
Get directory size
search function
yes
Is directory?
no
add to data (link list)
DataThreading
call function searchword size 0-1/4
call function searchword size 1/4 -1/2
call function searchword size 1/2 -3/4
call function searchword size 3/4 -szie
Output search result

Figure6: Full flow chart

2.6 Data Set

The data set is that data that we created for testing with differing sizes and each file given a name.

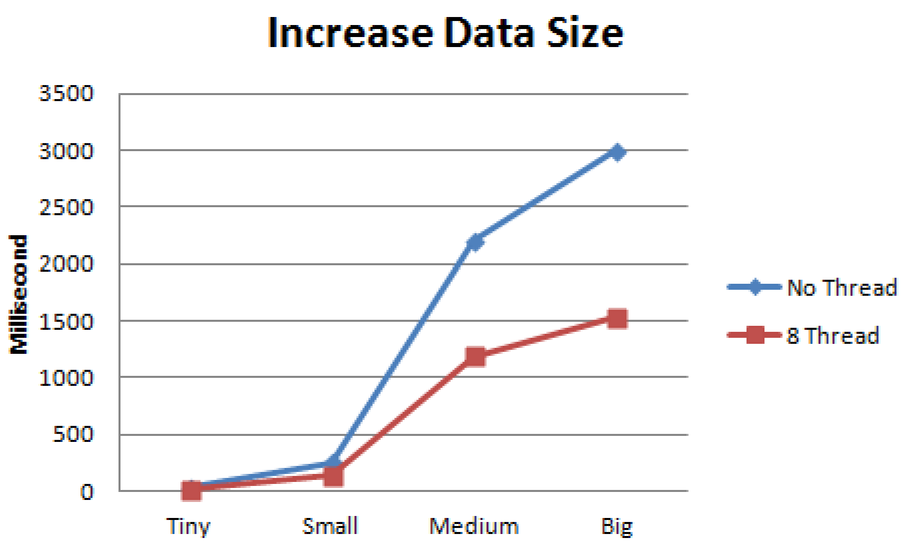
see appendix D

3. Results and Discussion

Test 1: The purpose of these initial tests were to simply verify whether or not larger datasets would require more time to run than smaller ones regardless of the thread count.

	No Thread
Tiny	32.0
Small	250.0
Medium	2201.0
Big	2997.0

	8 Thread
Tiny	15.0
Small	142.0
Medium	1191.0
Big	1532.0



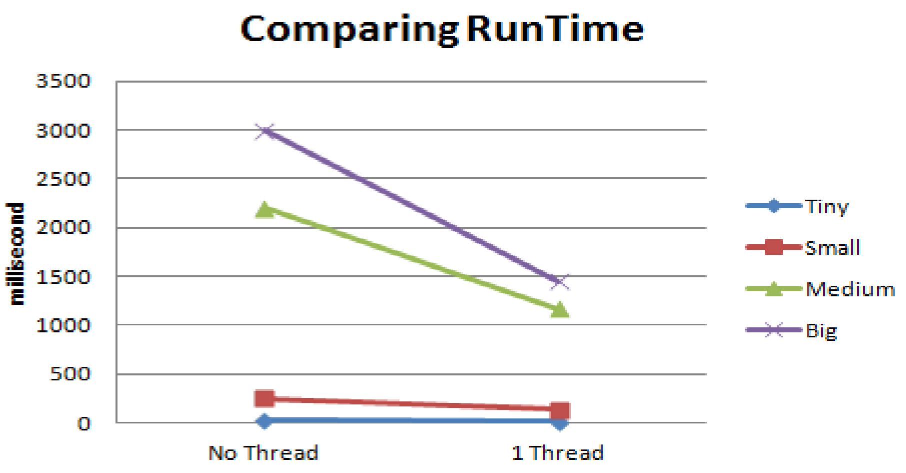
Graph7: Increase file size to compare no thread and 8 threads

From Graph7, we can see that with either no-threads or 8-threads, the run time still increases as we increase the file size.

Test2: Test 4 datasets with no thread and 1 Thread

	No Thread	1 Thread	Time Reduced
Tiny	32	15	53.10%
Small	250	141	43.60%
Medium	2201	1173	46.70%
Big	2997	1450	51.60%

Graph 2: Comparing Run Time with/without thread

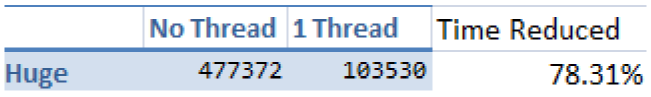


Graph 8: Comparing Run Time

In these tests, we simply attempted to verify Amdahl's Law and see if when you added more threads to a search, whether or not the search time would decrease. After we tested and recorded tests with both no-threads and 1-thread, we then graphed the information.

From the graph 8 displayed above, you can tell that increasing the thread count by even 1 thread has sped up our searches by as much as 53%. This is more evident the larger your file size is.

Test3: Huge file (50GB) on 0-1 Thread

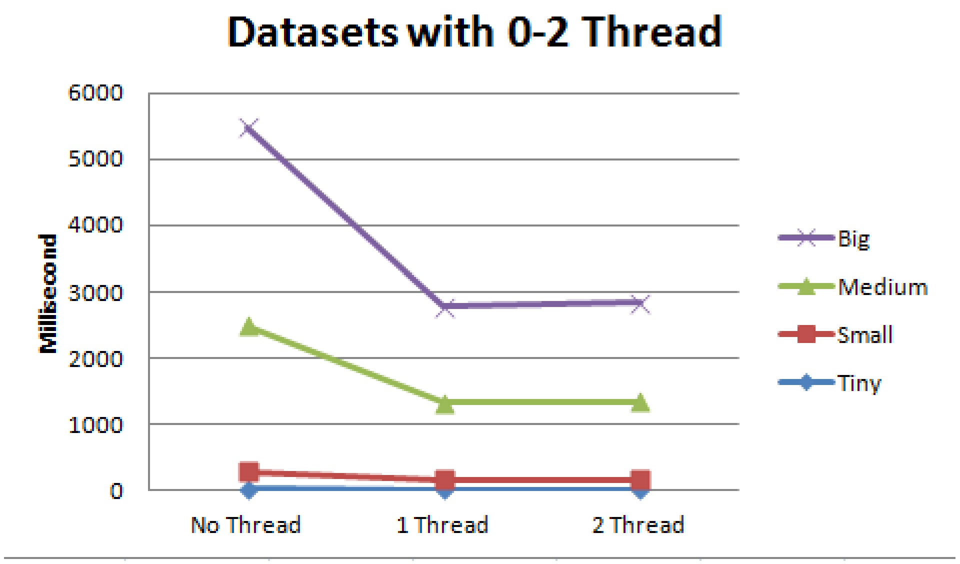


	No Thread	1 Thread	Time Reduced
Huge	477372	103530	78.31%

From this huge dataset, we found that the time had reduced by 78.31%, this showed us that the threading had greater effects on larger datasets, the larger the dataset becomes, the more time will be reduced.

Test4: Increase threading

	No Thread	1 Thread	2 Thread
Tiny	32	15	16
Small	250	141	142
Medium	2201	1173	1187
Big	2997	1450	1499

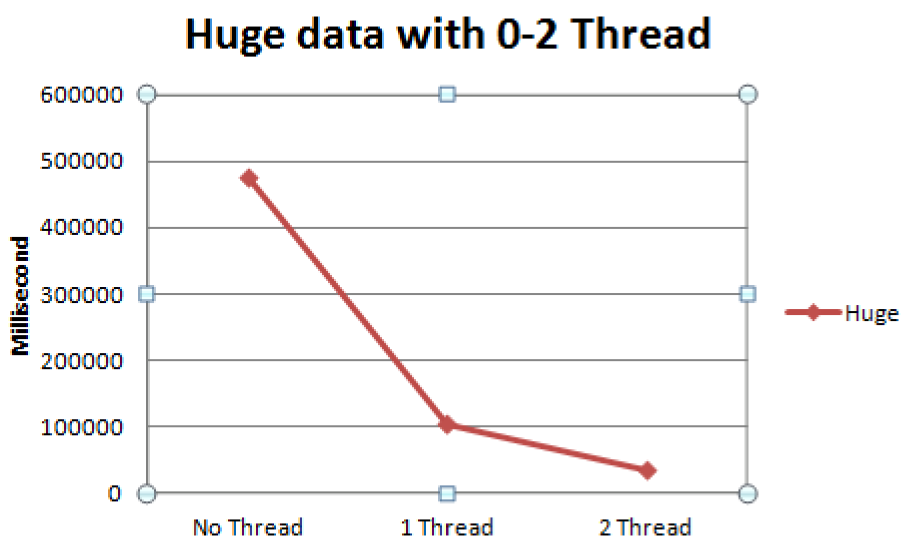


Graph9: Datasets with 0-2 Threading

From Graph9, we can see that increasing the thread count from 1 to 2 did not decrease the run time, but it increased. Since we have test in Test 4 to support that larger datasets would have more reduced run time, we would like to see if these 4 sets of data hits the bottleneck of the context switch caused by its limited size.

Test5: Increase threading on huge (50 GB) dataset

	No Thread	1 Thread	2 Thread
Huge	477372	103530	34690



Graph10: Huge dataset with 0-2 Threading

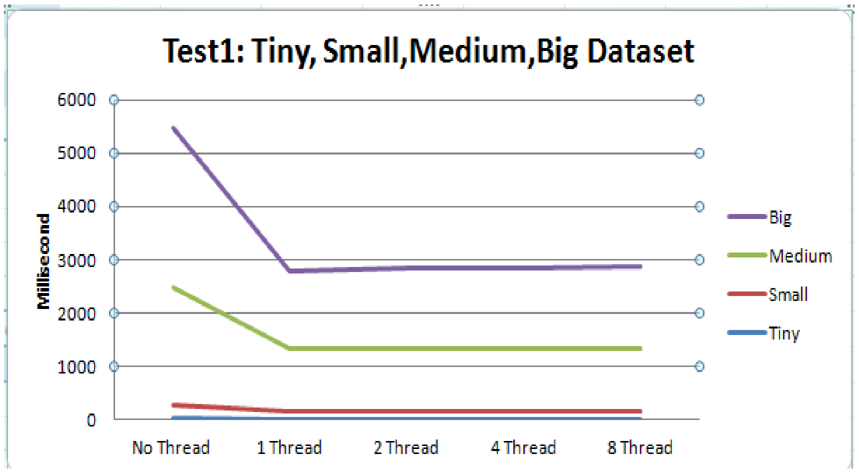
From our Huge dataset testing from 0-2 threads, when we tried using a thread count of 2, we found that the run time still decreased unlike the results from the smaller datasets.

Thread Count	Time Taken (Milliseconds)
No Thread	477372
1 Thread	103530
2 Thread	34690
Reduced percentage (0-1)	78.31%
Reduced percentage (0-2)	92.73%
Reduced percentage (1-2)	66.49%

The data chart above shows that in tests of the huge dataset (50GB), from 0 - 2 threads, the run time was reduced by up to 92.73%. This proves our assumption that the 4 smaller datasets (Tiny, Smail, Medium, Big) are too small to run up to 2-threads, as it hits the bottleneck of switch context and begins to increase run time. The chart also show us that with a large enough dataset, enabling 2-threads will continue decreasing the run time by 66.49%.

Test 6: Test 4 datasets with 0-8 Threads

Data Size	No Thread	1 Thread	2 Thread	4 Thread	8 Thread
Tiny	32.0	15.0	16.0	15.0	15.0
Small	250.0	141.0	142.0	140.0	142.0
Medium	2201.0	1173.0	1187.0	1170.0	1191.0
Big	2997.0	1450.0	1499.0	1529.0	1532.0

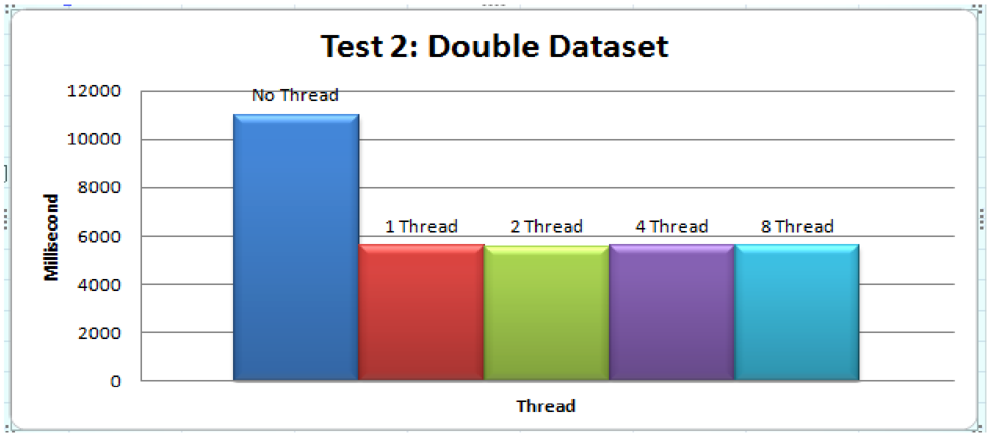


Graph 11: Comparing Run Time up to 8 threads

As can be seen from the graph 11, in every different dataset from Tiny - Big and in every other case, parallel searches took about half the time taken for 0-threads to search. As we observed in earlier testing, the results are what we expected. To add more threads program does not decrease run time, but actually increased it as caused by hitting the limit of the best performance of this size of data.

Test 7: Increased testing data-set to double the size and add all 4 data-sets and test 0-8 Threads

	No Thread	1 Thread	2 Thread	4 Thread	8 Thread
Dataset	10995.0	5569.0	5555.0	5600.0	5602.0



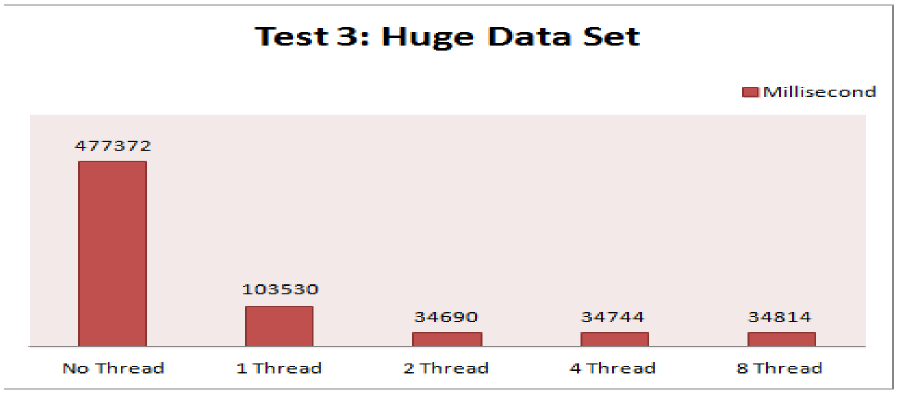
Graph 12: Double dataset size

Our maximum search speed is roughly 5555.0 milliseconds and can be seen by our graph and results for this test. Within this data-set, from No-threads - 1 thread, the search speed was halved. By, 2 threads, the search had only sped up by a minuscule amount. After 2 threads, the search then began slowing down by about 2 - 5 milliseconds. This maximum search speed can be attributed to the fact as such a minuscule data-set size can only be separated into 2 threads at max, therefore creating more threads only slows down the search.

From this result, we understand, if is small size dataset, increased the threading in programming will in fact decrease the performance.

Test 8: Test a Huge data set (50 GB) with 0-8 Thread

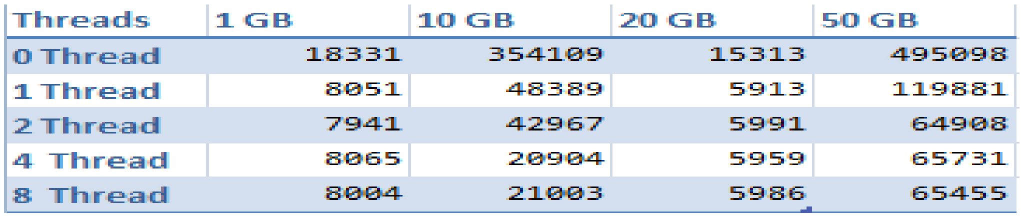
	No Thread	1 Thread	2 Thread	4 Thread	8 Thread
Huge	477372.0	103530.0	34690.0	34744.0	34814.0



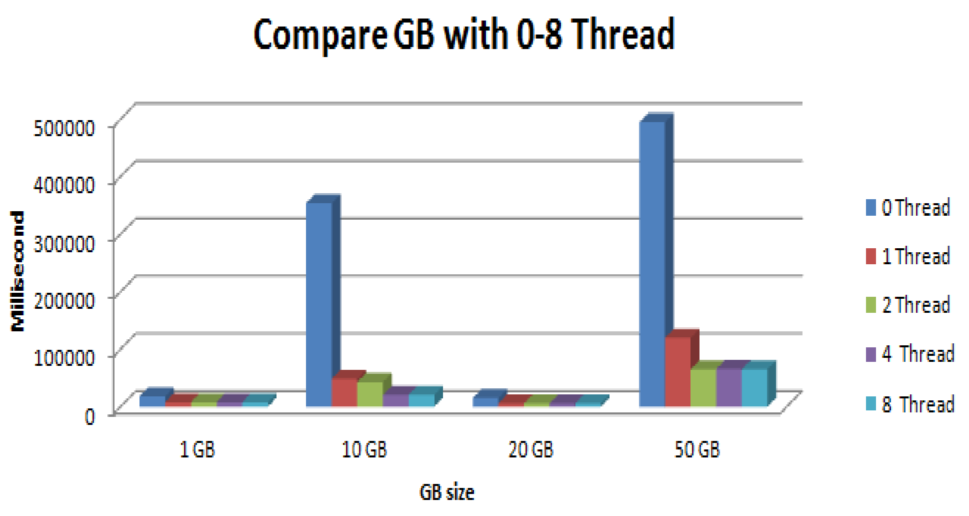
Graph 13: Huge Dataset with 0-8 Threads

When testing the Huge data set, the no thread search took 477 seconds or about 7 minutes and 57 seconds to search the entire set. Adding another thread drastically decreased this, to 103.53 seconds or about 1 minute and 44 seconds, almost a 7 minute improvement over the previous search. Adding yet another thread in the 2 thread search further improved on that time, decreasing the search time to almost 30 seconds for the entire search, shaving off just over a minute of search time. Unfortunately, adding additional threads after that only added time to the process without noticeable or satisfactory improvement, which means that the additional time is actually wasted CPU cycles.

Test 9: Compare different GB Data sets with 0-8 Thread



Threads	1 GB	10 GB	20 GB	50 GB
0 Thread	18331	354109	15313	495098
1 Thread	8051	48389	5913	119881
2 Thread	7941	42967	5991	64908
4 Thread	8065	20904	5959	65731
8 Thread	8004	21003	5986	65455



Graph 14: GB data with 0-8 Threads

The 10 gigabyte data set was the most of the text files so it used threading most effectively. The 20 GB dataset is most directory data, the decrease of searching time was expected.

Result: Within our fourth test, we tried using different amounts of threads for different sized files. Although they all sped up as expected, it seems that we achieved the same issue that we encountered in test 2. Past 2 threads, the search seems to choke up as if it couldn't speed up the search anymore, or it didn't have enough information in a file up to 50GB to split up amongst the 8 threads. This caused the graph of the each of the file sizes to look extraordinarily similar.

Test 10: Accuracy on searching

- Test: cat

1561694|Big Cat Slough|Gut|WI|55|Grant|043|431145N|0902705W|43.1958197|-90.4515141|||||204|669|Muscoda|08/29/1980|

In directory : C:\Users\temp\Documents\AllStates_20120204\WI_Features_20120204.txt

1580615|Tiger Cat Dam|Dam|WI|55|Sawyer|113|460147N|0911435W|46.0296783|-91.2429519|||||415|1362|Spider Lake|08/29/1980|

In directory : C:\Users\temp\Documents\AllStates_20120204\WI_Features_20120204.txt

1580616|Tiger Cat
Flowage|Reservoir|WI|55|Sawyer|113|460203N|0911511W|46.0341376|-
91.2530223|||||412|1352|Seeley|08/29/1980|07/13/2011

In directory : C:\Users\temp\Documents\AllStates_20120204\WI_Features_20120204.txt

Accurately found

- Test : 1990

(NECTAs) in Alphabetical Order and Numerical and Percent Change: 1990 and

In directory : C:\Users\temp\Desktop\data\New Haven, Connecticut - Wikipedia, the free encyclopedia.txt

1990

In directory : C:\Users\temp\Desktop\data\New Jersey - Wikipedia, the free encyclopedia.txt

between 1989 and 1990 and verification is needed to confirm North Korean claims

In directory : C:\Users\temp\Desktop\data\North Korea and weapons of mass destruction - Wikipedia, the free encyclopedia.txt

Accurately found

- Test: real-world

Enter text to search in system:

real-world

It took 70043.0 millisecond

Do you like to have another search? (yes/no) yes

Enter text to search in system:

This result shows the accuracy of our search engine because there has no "real-world" in our database

Accurately found

- Test: N=4

For example if N=4 :

In directory : C:\Users\data\upc(medium) - Copy\berkeley_upc-2.14.6\upc-tests\benchmarks\gwu_bench\nqueens\Description.txt

For example if N=4 :

In directory : C:\Users\data\upc(medium) - Copy\berkeley_upc-2.14.7\upc-tests\benchmarks\gwu_bench\nqueens\Description.txt

Accurately found

- Test: you and me

Enter text to search in system:

you and me

It took 56643.0 millisecond

Do you like to have another search? (yes/no)

no

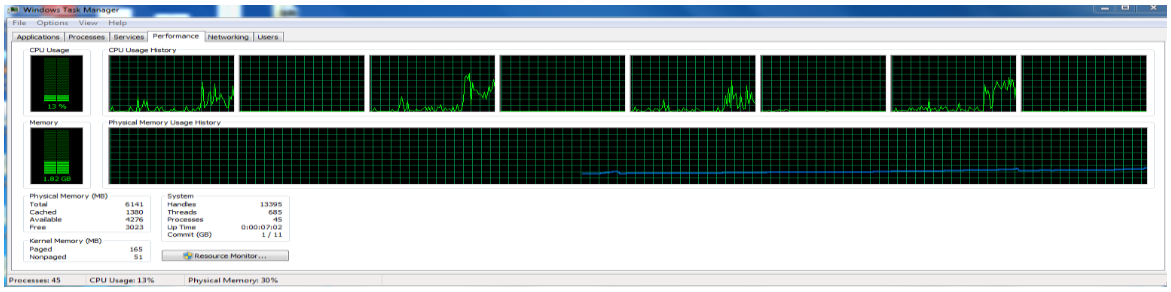
Good Bye

This actually is a correct result because our search engine isn't designed this way. We will improve our program in the future as deemed necessary

Accurately found

Test 11 : CPU monitor(Data set is set to 40 GB)

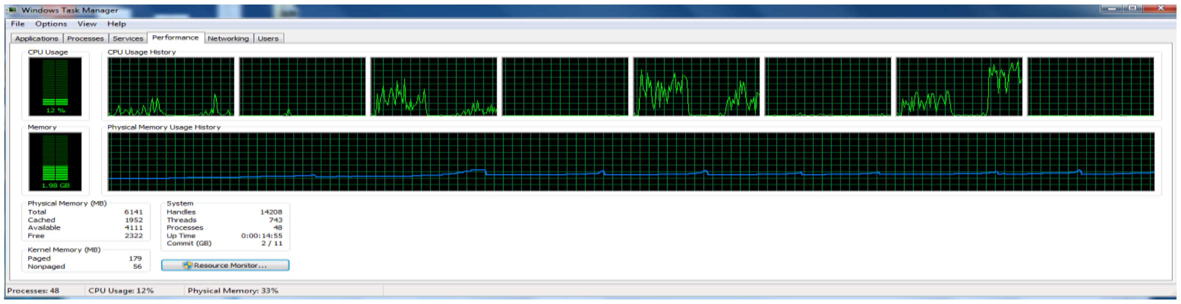
<u>1 Thread</u>



```
Physical Memory (MB)            System
Total          6141             Handles          13395
Cached         1380             Threads            685
Available      4276             Processes           45
Free           3023             Up Time     0:00:07:02
                                Commit (GB)     1 / 11
Kernel Memory (MB)
Paged           165
Nonpaged         51

Processes: 45    CPU Usage: 13%    Physical Memory: 30%
```

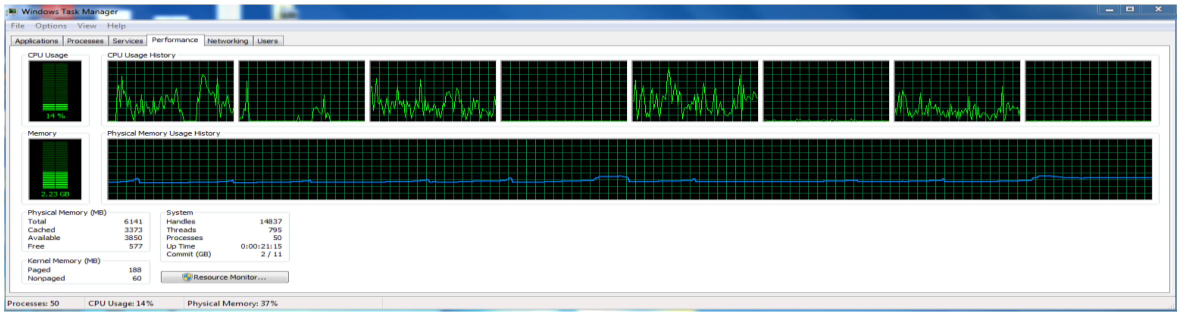
<u>2 Thread</u>



```
Physical Memory (MB)            System
Total          6141             Handles          14208
Cached         1952             Threads            743
Available      4111             Processes           48
Free           2322             Up Time     0:00:14:55
                                Commit (GB)     2 / 11
Kernel Memory (MB)
Paged           179
Nonpaged         56

Processes: 48    CPU Usage: 12%    Physical Memory: 33%
```

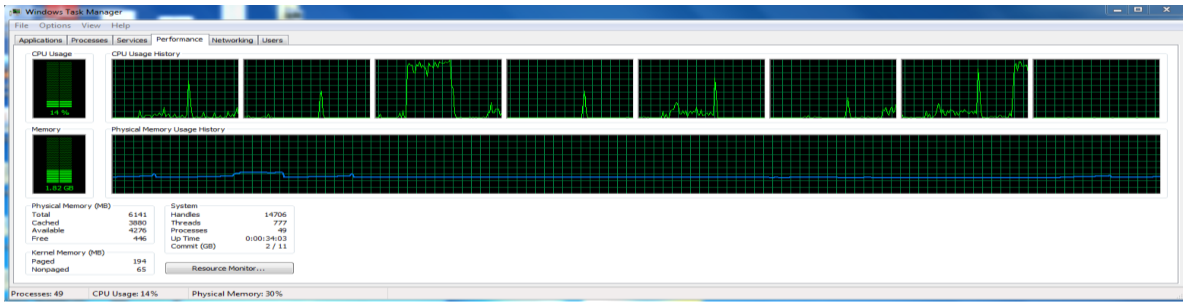
<u>4 Thread</u>



```
Physical Memory (MB)            System
Total          6141             Handles          14837
Cached         3373             Threads            795
Available      3850             Processes           50
Free            577             Up Time     0:00:21:15
                                Commit (GB)     2 / 11
Kernel Memory (MB)
Paged           188
Nonpaged         60

Processes: 50    CPU Usage: 14%    Physical Memory: 37%
```

<u>8 Thread</u>



```
Physical Memory (MB)            System
Total          6141             Handles          14706
Cached         3880             Threads            777
Available      4276             Processes           49
Free            446             Up Time     0:00:34:03
                                Commit (GB)     2 / 11
Kernel Memory (MB)
Paged           194
Nonpaged         65

Processes: 49    CPU Usage: 14%    Physical Memory: 30%
```

We found that the more threads used, the more processors are used. We also established that some of the processors worked harder than the others, namely cores 1, 3, 5, and 7. As shown, the CPU usage stayed about the same in all trials, and there was no relationship between threads and RAM usage. In terms of activated cores, we observed that 4-5 cores were actively used for 0-4 threads, but then jumped to 7 when running 8 threads.

4. Conclusions and Future Work

Conclusion

The search engine is the keystone in the bridge of data-mining. Our report is about the results of our search engine program. Throughout this project, we as a team have attempted to achieve true multi-thread parallel processing using Java programming.

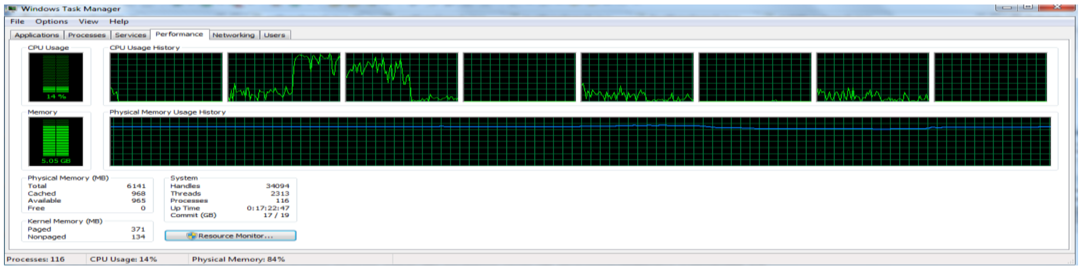
As the world’s amount of information is ever rapidly expanding, computer users must look towards new and more innovative ways to search for information. Multithreading is the answer to this problem. Within the course of this project, we found in the first few tests that multithreading exponentially decreases the amount of time taken to search a database of files. Within the next few tests that we did, we acquired results that said that with a database of about 200MB, we could not increase the amount of threads used to search past 2, otherwise, the search time would slightly increase. This problem allowed us to infer that smaller databases with small amount of information can only handle just a few threads before extending the search time (e.g. medium sized database – couldn’t get past 2 threads). After we encountered this problem, we decided to use a database with over 50GB of information. It was then able to handle more threads and show that with huge amounts of information, threads can limitlessly decrease search time. Overall throughout this project, we as a team have found out that in order to solve the problem of the world’s ever expanded plethora of information, we must look towards multithreading as a viable answer.

Testing Observations:

Observation 1: When testing 0 Thread, the time shows up after I/O result output. When testing 1-8 Thread, the time always shows up before the I/O result output.

- This can be understood as the I/O takes system resources, and when running a no-thread search, the computer can do the I/O result before the program. However, when running a threaded program, the computer gives the program results before the I/O results.

Observation 2: We found that our memory usage was very high. An example would be when our system, while running our program, used 5.6GB of its 6GB of memory.



```
Physical Memory (MB)            System
Total          6141             Handles          34094
Cached          968             Threads           2313
Available       965             Processes          116
Free              0             Up Time     0:17:22:47
                                Commit (GB)    17 / 19
Kernel Memory (MB)
Paged           371
Nonpaged        134

Processes: 116    CPU Usage: 14%    Physical Memory: 84%
```

- The RAM is always high because the program stores tokens and text lines in the main memory, which leads to high memory usage.

Observation 3: When the search dataset contains most of directory and few text file, the time will all decreased by all of 0-8 Threads testing.

- Example testing could be observed from Test4.

Observation4: With all of the different thread testing among the small dataset, the time has significantly reduced between no thread and 1 thread. The response times was not seems to be reduced after 2 thread and after.

- This happened because, in the case of concurrency for processor and memory, the response time is increased by the time spent in context switching among the threads. In the Test3 with huge dataset, we found the response time from 0 to 1 and to 2 threads, the run time had greatly reduced. And then, the time are almost the same from2, 4, and 8 threads . This also can be understood that the time spent in context switching among the threads is involved.

Future Work:

- We are looking to improve our search times and we also would like to request a real-world huge database and going forward into analyzing data mining in the foreseeing future.

5. Recommendations

For this project, we managed to complete a project that is of sufficient quality as to be a functional and complete product. In the future, we would like to enhance some interesting areas such as using key & value searching, semantic searching, and handling map-reduce type of data set.

We Accomplished:

This year, we have come to a long way in learning how to program in Java, parallel coding, and algorithms, system hardware CPU and memory analyzing. This year we accomplished the threading in searching text, which searched line by line for specific word. From this project we were able to get a taste for Java threading programming, and we explored the threading in software and hardware.

Acknowledgements

Thanks to:

The supercomputing staff, especially Mr. David Kratzer, for arranging the visit to Sandia labs and all of the other supercomputing activities.

Dr. HB Chen for supplying us with advice and the necessary knowledge to get as far as we did.

LANL editor Sue King to edit our report for compliance to a standard report form.

Dr. Guanhua Yan for advising us I/O performance, Java threading context switch concept.

Mr. Leroy Goodwin for supplying us with constructive criticism, hosting supercomputing meetings during Wednesdays, and for supporting us along the way.

Mr. Wyatt Dumas for teaching some of us our fundamental computer and programming skills as well as reviewing our project several times

Dr. Hailin Deng for reviewing and giving advice on our report.

References

"Analytics" *Wikipedia*. Wikimedia Foundation, 27 January 2012 Web OVERALLL <http://en.wikipedia.org/wiki/Analytics>

Charrie, "Benefit of Multithreading Programming", Web <http://charriecharry.blogspot.com/2009/07/benefits-of-multi-threading-programming.html>

Hussein A. Abbass, Ruhul A. Sarker, Charles Sinclair Newton, "Data mining: a heuristic approach", Web

<http://books.google.com/books?id=vmV2B_bzyD8C&pg=PA266&lpg=PA266&dq=Data+parallelism+refers+to+the+execution+of+the+same+operation+or+instruction&source=bl&ots=EB8JjCLaj0&sig=SuIxZdSI6qYmuC1_kzHYyvgLSbA&hl=en&sa=X&ei=V7BwT6n0DYjUiAL689ynBQ&ved=0CD0Q6AEwBA#v=onepage&q=Data%20parallelism%20refers%20to%20the%20execution%20of%20the%20same%20operation%20or%20instruction&f=false>

Kai Hwang, " Advanced Parallel Processing with Supercomputer Architectures ", PROCEEDINGS OF THE IEEE, 1987, VOL. 75, NO. IO, p1348-1379.

Karin, Sid; Bruch, Kimberly Mann. "Supercomputers." Computer Sciences. 2002. *Encyclopedia.com.* 15 Mar. 2012 <http://www.encyclopedia.com>.

"IBM Sequoia supercomputer to be the fastest ever" 07 Feb. 2012 , Web

<http://robotzeitgeist.com/tag/supercomputer>

"Multithreading." Wikipedia. *Wikimedia Foundation, 24 Jan. 2012. Web. <http://en.wikipedia.org/wiki/Multithreading>.*

Neeman, Henry. "Multithreading and Multiprocessing." *Petascale : Multithreading and Multiprocessing*. Web. <http://www.shodor.org/petascale/materials/UPModules/sipeMultithreadingMultiprocessModule2/>.

"Thread (computing)." *Wikipedia*. Wikimedia Foundation, 03 Sept. 2012. Web. <http://en.wikipedia.org/wiki/Thread_(computer_science)>.

Appendix A : Search file recursively

```
private static void search(File dir, String newfilename) throws IOException {
    File[] files = dir.listFiles(); //list file in folder dir to array list files
    if (files != null)
    {
    for (int k =0 ; k< files.length; k++) //loop through array
            {
            if (files[k].isDirectory()) // test if the file is a directory
                    search (files [k], newfilename); // if yes,recursively call function search
            }
    else if (files[k]!= null)// if it's a file
    {
            if ((files [k].getName()).endsWith(".txt"))
                    // verify if the file has .txt extension, if is txt
            data.add(files [k]); //add the file to linked list "data"
    }}}
```

Appendix B : search word in individual file

```
        private static synchronized void searchword (File pfile, String name) throws
FileNotFoundException
        {
        try {
                        FileReader file = new FileReader (pfile); //read a content of a file
                        BufferedReader br = new BufferedReader(file); //read text from a character (line)
                String line; //local variable
                while ((line = br.readLine())!= null) //while there still has line in file,
                                                        //read and assign to variable" line"
                {
                        StringTokenizer st = new StringTokenizer(line); //break line to token
                                                                //and assign to "st"
                        String word; //local variable
                        while(st.hasMoreTokens()) //while is not the end of line
                                {
                                word = st.nextToken(); //assign the token to variable "word"
                                if ((word.toLowerCase()).compareTo(name)== 0)
                                {                                       //compare with lowercase
                                        System.out.println(line); //print out the line that has the word in
                                it
                                        System.out.println("In directory : " + pfile.getCanonicalPath());
                                                                        //print out path
                                }
                                }
                }
                file.close(); //close filereader file
                br.close(); //close bufferreader file
        }catch (IOException e) {}
        }
```

Appendix C : Java threading

```
        Thread thread1 = new Thread (new Runnable(){ //implement Java Runnable interface to create
threading
```

```
public void run() {                          //define run method
            for (int i = 0; i< size/4 ; i++) {  //loop through1/4 size, if size is 10, it will be 0-2
            try{
            searchword (data.get(i), newfilename); //call function to search word infile
            }catch (FileNotFoundException e){}
      } // for loop
      } //run class
}); //Runnable interface

Thread thread2 =new Thread (new Runnable(){
      public void run() {
            for (int j = (int) Math.ceil(size/4) ; j < size/2 ; j++){ //loop3-4 if size is 10
                  try{
                  searchword (data.get(j), newfilename);
                  }catch (FileNotFoundException e){}
            }}
});
Thread thread3 = new Thread (new Runnable(){
public void run() {
            for (int k = (int) Math.ceil(size/2) ; k <(size *3/4) ; k++){//loop from5-7 if size is
10
                  try{
                  searchword (data.get(k), newfilename);
                  }catch (FileNotFoundException e){}
            }
      }
});

Thread thread4 = new Thread (new Runnable(){
      public void run() {
            for (int l = (int) Math.ceil((size *3/4)) ; l < size ; l++){//loop8-9 if siz eis 10
                  try{
                  searchword (data.get(l), newfilename);
                  }catch (FileNotFoundException e){}
```

```
                    }}
});
thread1.start();
thread2.start();
thread3.start();
thread4.start();
```

Appendix D : Testing Dataset

Tiny- 4.56 MB (4,785,608 bytes) - 280 Files, 40 Folders

Small-16.4 MB (17,211,216 bytes) - 1,560 Files, 232 Folders

Medium- 201 MB (210,880,952 bytes) - 16,976 Files, 1,336 Folders

Big-683 MB (716,988,016 bytes) - 18,704 Files, 3,248 Folders

Double - double the 4 dataset and adding all 4 dataset

Huge- 52.0 GB (55,924,727,808 bytes) - actually search from root