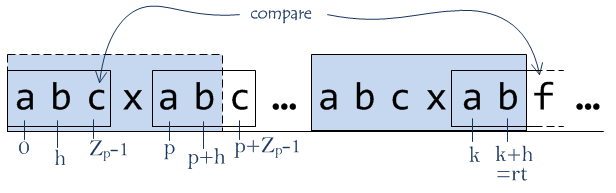
**Assignment 1**

**Parallel String Searching Algorithm on Shared Memory**

**Introduction**

For this assignment, I used Z-algorithms for string searching. Z-algorithms is a linear time pattern searching algorithm that has a fast enough searching speed that will not affect lots in a string searching system.



For this assignment, I used 3 books that I got from <https://www.gutenberg.org/ebooks/100> which is given in the Ed forum. The 3 books are **Dracula** (words: 164442), **Romeo and Juliet** (words: 28982), and **The Odyssey** (words: 132519). I get the word count of the books by using the inbuilt Linux function e.g., *wc -w Dracula.txt*. Besides I created a serial program to make the book into a list of words to use in the assignment.

**Serial Code**

The main function will loop through the books.

And for the read files function we will open the file, read it, and put it in an array. Then we will run the check unique function after we create the array.

Pseudocode

Main {

For (all books) {

run Read\_files function.

}

}

/Function/

Read\_files (file name, file length) {

For (file length) {

Write it into an array.

}

For(file length){

Change all the alphabet into lowercase.

Run check unique function if it is unique write to the unique array.

}

}

checkUnique (word, the unique array, length of the array) {

for (length of the array){

Combine the word with the word in the unique array.

Run z\_algo to see if it is inside the unique array already.

}

}

Z\_algo(combined string, z\_array, lengthof the string){

Initiate left and right pointer to zero and the first index of the array be the length of the string,

For (the length of the string) {

If (I > right pointer) {

Left = right =I;

While (check character) r++;

Update the z\_box;

Let the right pointer 1 index forward

}

Else{

Int k = i-l;

If (check k pointer in z\_box or not) {

Update z\_box

} else{

L=I;

While( check character) r++;

Update z\_box

Let the right pointer 1 index forward

}

}

}

}

So, from the pseudocode, we can separate the for loops into the main loop, read the file loop, check the unique loop, and pattern-matching loop.

**The main loop**: I­1 = 0, O1= readfile(filename (0), filelength (0))

I2=1, O2 =readfile(filename (1), filelength (1))

𝐼 1 ∩ 𝑂2 = ∅ (Anti dependency)

𝐼 2 ∩ 𝑂1 = ∅ (Flow dependency)

𝑂1 ∩ 𝑂2 = ∅ (Output dependency)

Therefore, it satisfies Bernstein’s condition.

**Read the file loop**: I1 = 0, O1 = W [0]=word0

I2 = 1, O2 = W [1] = word1

𝐼 1 ∩ 𝑂2 = ∅ (Anti dependency)

𝐼 2 ∩ 𝑂1 = ∅ (Flow dependency)

𝑂1 ∩ 𝑂2 = ∅ (Output dependency)

Therefore, it satisfies Bernstein’s condition.

Check unique loop: I1= 0, O1=check Unique(word0, the unique array, length of the array)

I2 = 1, O2=check Unique(word1, the unique array, length of the array)

𝑂1 ∩ 𝑂2 = uses same unique array

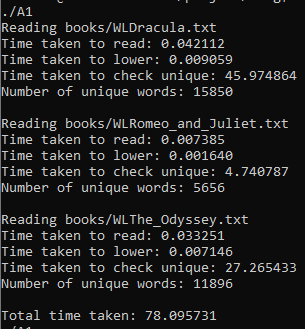
Therefore, it doesn’t satisfy Bernstein’s condition. It can be parallelized but will occur in race conditions.

**Pattern matching loop**: I1 = length of the array0, O1= return value

I2 = length of the array1, O2=return value

𝑂1 ∩ 𝑂2 = return value.

It doesn’t satisfy Bernstein’s condition.



Theoretical Speed Up:

1/ (Rs + Rp/p) = 1/ (all read time/total time + (all check unique/total time)/number of threads)

= 1/ (1.288\*10^-3 +0.9985/4)

=3.985

1/ (Rs + Rp/p) = 1/ (all read time/total time + (all check unique/total time)/number of threads)

= 1/ (4.7061\*10^-3 +0.9985/8)

=7.930

Word vs. Run time.

|  |  |  |
| --- | --- | --- |
|  | Total words | Run time |
| Dracula | 164442 | Approximate 46s |
| Romeo and Juliet | 28982 | Approximate 5s |
| The Odyssey | 132519 | Approximate 27s |

Based on the graph above we can see that the higher the word count the longer the time taken in serial code.

**Parallel code**

I selected to parallel the check unique loop to parallelize as it will have the most speedup from the serial code. I used 4 and 8 threads to parallelize the serial code and used reduction to get the unique word count. As we know this parallelized code will occur in race conditions while updating the unique word list since there will be different threads that find the same unique word that is not in the list and update at the same time so that there will be some words that consist in the list multiple times.

Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads (4) | Serial Time | Serial Word Count | Parallel Time | Parallel Word Count |
| *Dracula* | 45.974864 | 15850 | 7.411373 | 26509 |
| *Romeo & Juliet* | 4.740787 | 5656 | 0.832161 | 8945 |
| *The Odyssey* | 27.265433 | 11896 | 6.676119 | 20373 |
| *Total* | 77.981084 | 33402 | 13.722481 | 55827 |

Actual total speed up: 77.981084/13.722481= 5.682724866

Actual each book speed up: Dracula 45.974864/7.411373=6.20328568

Romeo & Juliet 4.740787/0.832161=5.696958882

The Odyssey 27.265433/6.676119=4.084024416

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads (8)** | **Serial Time** | **Serial Word Count** | **Parallel Time** | **Parallel Word Count** |
| *Dracula* | 45.974864 | 15850 | 3.274175 | 33758 |
| *Romeo & Juliet* | 4.740787 | 5656 | 0.323620 | 10941 |
| *The Odyssey* | 27.265433 | 11896 | 2.858759 | 26438 |
| *Total* | 77.981084 | 33402 | 6.456554 | 71131 |

Actual total speed up: 77.981084/6.456554=12.07781798

Actual each book speed up: Dracula 45.974864/3.274175= 14.0416636

Romeo & Juliet 4.740787/0.323620= 14.64923985

The Odyssey 27.265433/2.858759= 9.537506659

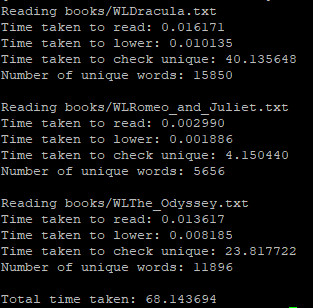
Based on these two tables above we can see that the more threads we used the more speed up we got but it sped up more than the theoretical speed up because we selected to parallel the check unique loop which will occur in race conditions so we can also observe that the word count from serial to parallel increased since the thread might upload the same unique word to the list at the same time. Furthermore, we can also observe from serial code the percentage of the unique word in the word list will also affect the word count after parallel. For, Dracula has 15850 unique words out of 164442. Percentage: 15850/164442=0.096 =9.6%, Romeo and Juliet 5656 unique words out of 28982. Percentage: 5656/28982=0.1859=19.52%, and The Odyssey 11896 unique words out of 132519. Percentage: 11896/132519=0.0852=8.97%

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads (4)** | **Percentage of Unique words** | **Speed up** | **False unique word count** | **Percentage of Wrong count** |
| *Dracula* | 9.6% | 6.20328568 | 10659 | 67.25% |
| *Romeo & Juliet* | 19.52% | 5.696958882 | 3289 | 58.15% |
| *The Odyssey* | 8.97% | 4.084024416 | 8477 | 71.26& |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads (8)** | **Percentage of Unique words** | **Speed up** | **False unique word** | **Percentage of Wrong count** |
| *Dracula* | 9.6% | 13.225 | 17908 | 112.98% |
| *Romeo & Juliet* | 19.52% | 14.736 | 5285 | 93.44% |
| *The Odyssey* | 8.97% | 9.7127 | 14542 | 122.24% |

Based on the table above, we can see that the percentage of unique words in the books will affect the speed up when we have more unique words in a book then the speed up will be faster. And we can also observe that when we increase the threads the false percentage of reading the same unique words will increase by approximately 40% for each book. Also, with a higher percentage of wrong word count, we will get a lower speed up. We can assume that the threads have been waiting to update the unique words while other threads are still updating the array.

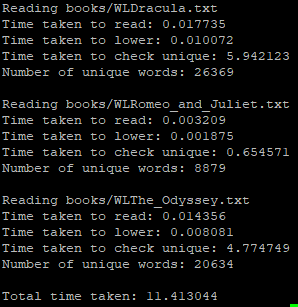
**CAAS**



Serial run in CAAS.

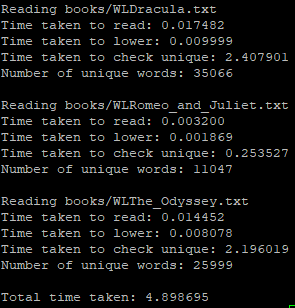
|  |  |
| --- | --- |
|  | Time taken |
| Dracula | 40.135648 |
| Romeo and Juliet | 4.150440 |
| The Odyssey | 23.817722 |
| Total time | 67.463308 |

We can compare the time on CAAS and our own local device.



Parallel in CAAS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Thread (4) | Serial time taken | Serial word count | Time taken | Parallel word count |
| Dracula | 40.135648 | 15850 | 5.942123 | 26369 |
| Romeo and Juliet | 4.150440 | 5656 | 0.654571 | 8879 |
| The Odyssey | 23.817722 | 11896 | 4.774749 | 20634 |
| Total time | 67.463308 | 33402 | 11.471443 | 55882 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Thread (8) | Serial time taken | Serial word count | Time taken | Parallel word count |
| Dracula | 40.135648 | 15850 | 2.407901 | 35066 |
| Romeo and Juliet | 4.150440 | 5656 | 0.253527 | 11047 |
| The Odyssey | 23.817722 | 11896 | 2.196019 | 25999 |
| Total time | 67.463308 | 33402 | 4.857447 | 72112 |

Based on the serial code and parallel code runs in a single computer and CAAS platform. We can observe that CAAS runs faster than a single device, which is normal as CAAS has a higher computing power compared to our own single computer.

Actual Speed up of CAAS

Threads 4

Actual total speed up: 67.463308/11.471443=5.88097836

Actual each book speed up: Dracula 40.135648/5.942123= 6.75442901

Romeo & Juliet 4.150440/0.654571= 6.34070254

The Odyssey 23.817722/4.774749= 4.98826682

Threads 8

Actual total speed up: 67.463308/4.857447=13.8886349146

Actual each book speed up: Dracula 40.135648/2.407901= 16.6683131906

Romeo & Juliet 4.150440/0.253527= 16.3708007431

The Odyssey 23.817722/2.196019= 10.8458633555

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Own Device (T4) | CAAS (T4) | Own Device (T8) | CAAS (T8) |
| Dracula | 6.20328568 | 6.75442901 | 13.225 | 16.6683131906 |
| Romeo and Juliet | 5.696958882 | 6.34070254 | 14.736 | 16.3708007431 |
| The Odyssey | 4.084024416 | 4.98826682 | 9.7127 | 10.8458633555 |
| Total time | 5.682724866 | 5.88097836 | 12.07781798 | 13.8886349146 |

From the above, we can observe that CAAS has a higher speedup compared to our own device. Since we choose to increase the speed up of the algorithms it means we sacrifice the accuracy of determining whether the word is the only word in the unique list or not which means that the unique word array will have multiple same words in it.