Finding and Sorting Documents based on Parallel Word Search

DHANANJAY SUNIL MENON	:	18BCE2330
GOKUL RAJ	:	18BCE2308
HRITHIK AHUJA	:	18BCE2154
ROHAN ALLEN	:	18BCI0247

Project Goals

- Finding documents having the word searched.
- Constructing a table based on occurrences of the word
- Sorting documents based on table
- Comparative study of parallel and linear search

ABSTRACT

Abstract 1

- Imagine you wanted to search for a word within a document, and you had to go through an entire file of documents. This would be time consuming manually and serially.
- In this project, parallelism can be used to find the document with the highest occurrences of the word. Also a table will contain the number of times a word has occurred in each document.

Abstact 2

- Multi-threaded systems will be used for search purposes, and this would be compared to the sequential search of words (in a separate program).
- The comparison would be made based on time taken for different circumstances for both serial and parallel searches.

LITERATURE SURVEY

- OPEN MP
- LINEAR SEARCH
- PARALLEL SEARCH
- THREADS VS PROCESSORS

OpenMP

OpenMP consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior.

When we use OpenMP, we use directives to *tell* the compiler details of how our code should be run in parallel.

Linear Search

- Linear search is one of the simplest algorithms to implement. It has the worst case complexity of **O(n)**, The algorithm has to scan through the entire list to find the element if the required element isn’t in the list or is present right at the end.

Parallel Search

- To parallelize the for loop, the openMP directive is: #pragma omp parallel for. This directive tells the compiler to parallelize the for loop.

Threads vs Processors 1

- Threads share all memory in the program, but there are multiple threads of execution.
- In processes, all memory is local to each processor except memory that is explicitly shared.

Threads vs Processors 2

- threads are virtual, and help to generate parallelism in a program.
- processors/cores are physical components.

Previous Works

- *Word Searches and Computational Thinking*
- *Solve word search puzzles and learn about computational thinking and search algorithms*. Puzzles are a good way of developing computational thinking. Word searches involve pattern matching. To solve them quickly an algorithm helps – essentially an adapted linear search for individual letters by scanning the grid, followed by searching round that point for the second letter, then continuing the search in that direction.
- As an alternative it can be used as a practical use of linear search

Previous Works

- Key phrases, key terms, key segments or just keywords are the terminology which is used for defining the terms that represent the most relevant information contained in the document. Although the terminology is different, function is the same: characterization of the topic discussed in a document. Keyword extraction task is important problem in Text Mining, Information Retrieval and Natural Language Processing.

SCOPE

- The scope of this project is that the word search engines in the world are efficient enough for the current level of processing speed. But, as the technology evolves, the speed has to be greater than the present. The present method is sequential method. The metrics of sequential method are much inferior compared to that of the multithreaded search. The scope as of now is the files saved in the memory. Later developments can be made to make it suitable .

Algorithm:

- First words are searched in all the documents and text files in a sequential manner.
- The number of occurrences of each word in document is stored first. Only then do we go and search the next document.
- We use linear search to perform this sequential searching of documents

Algorithm:

- Now words are searched in all the documents and text files in a parallelized manner, i.e. all the documents are searched for that particular word concurrently.
- The number of occurrences of each word in all the documents as and when the have completed searching them.
- We use OpenMP library in C programming and : #pragma omp parallel for parallelizing the linear search.

Algorithm:

- The documents are located in a folder and file names are given as some collection
- The files are taken in a parallel way and all the data is collectively stored in a vector
- After this the vectors are sent to a search method that parallelizes the search and concurrently browses through the working documents

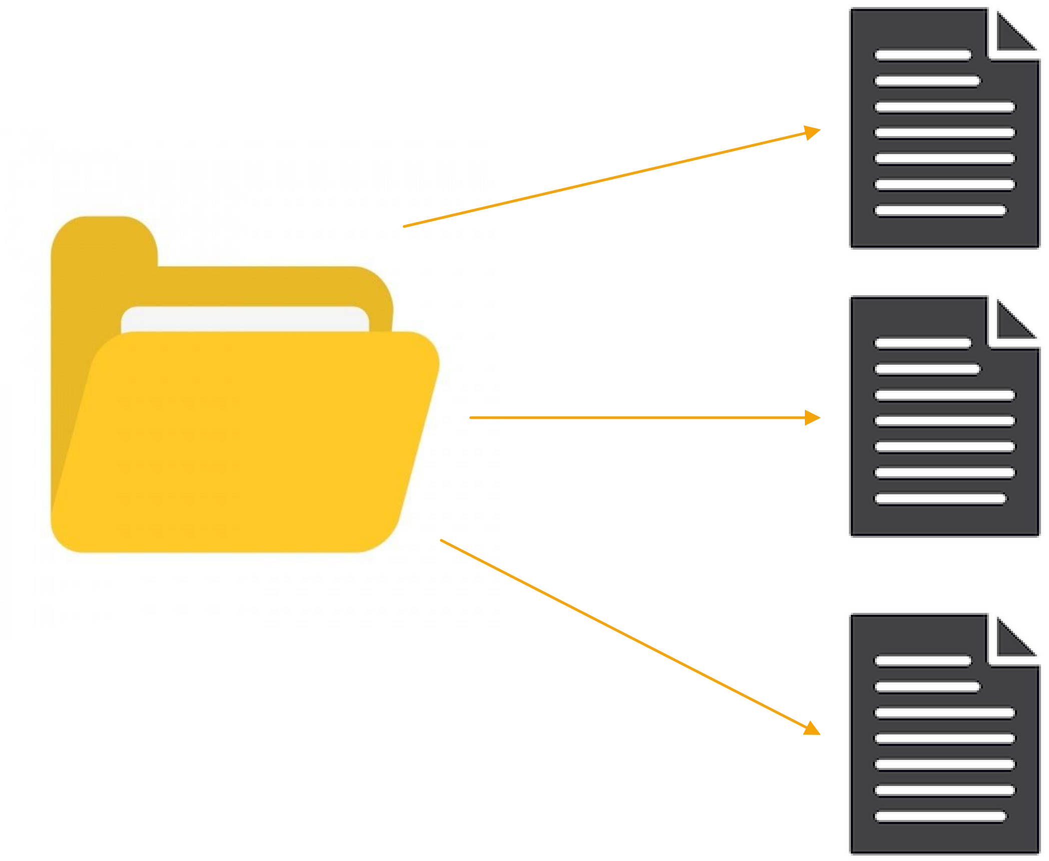
Algorithm:

- After this, the documents are ranked based on highest word searches.
- A mapping is done and the text files are displayed accordingly based on those with highest occurrence.
- Parallelizing the sequential search will expedite the word search process as there are thousands of words in each file.

Parallel Implementation

- Each Document was taken in parallel
- Each line within a document was taken in parallel

Document Structure

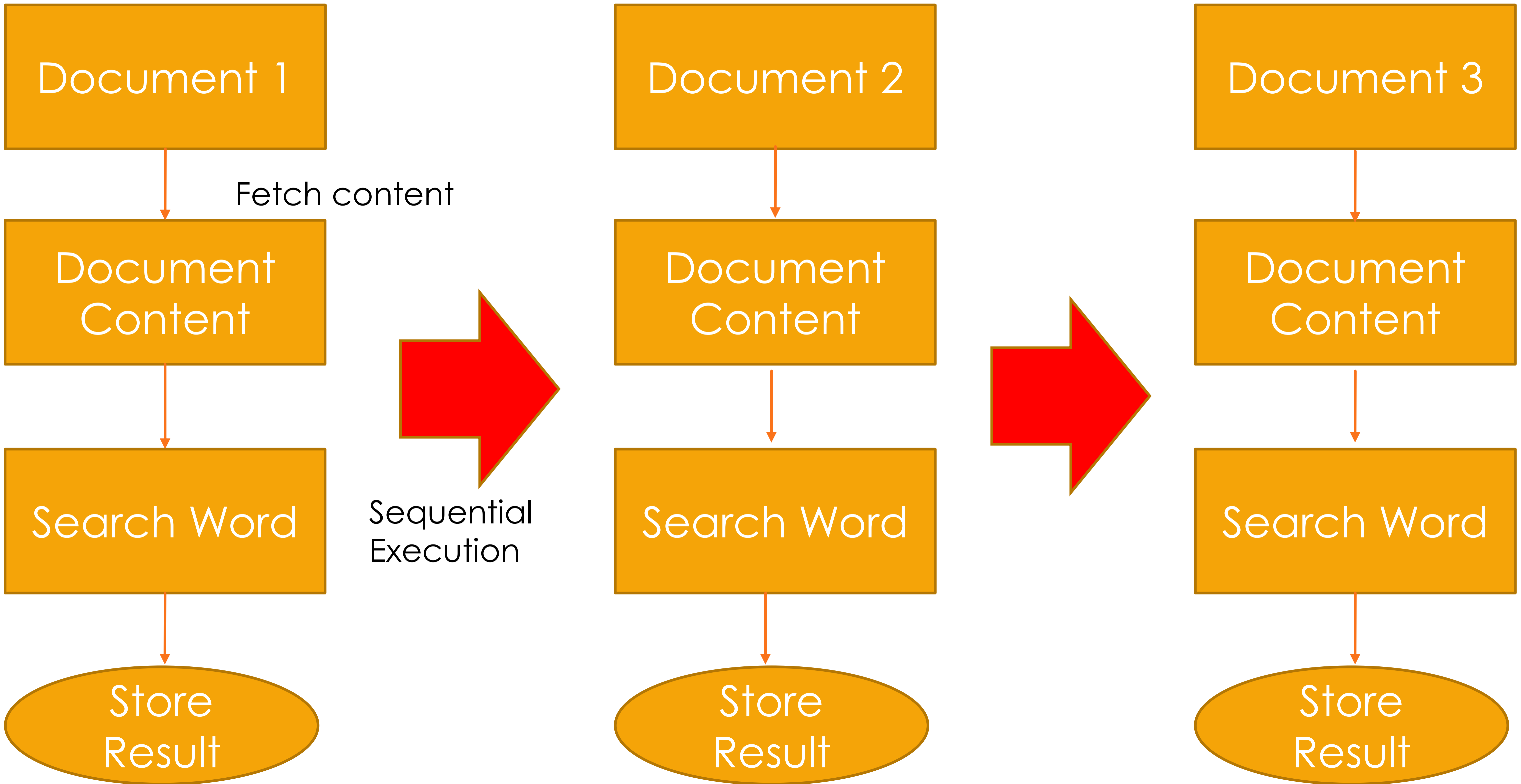


TEXT

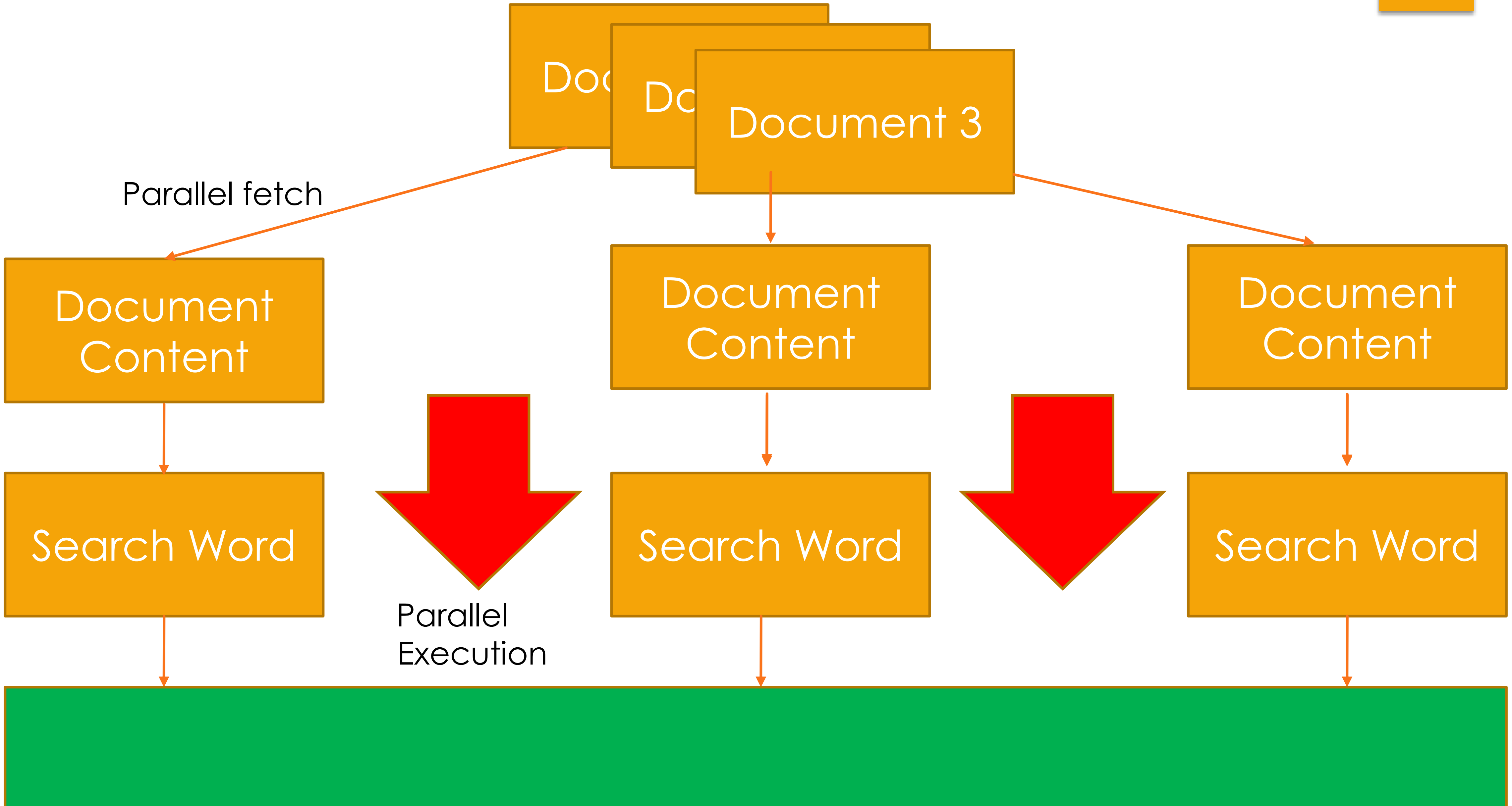
TEXT

TEXT

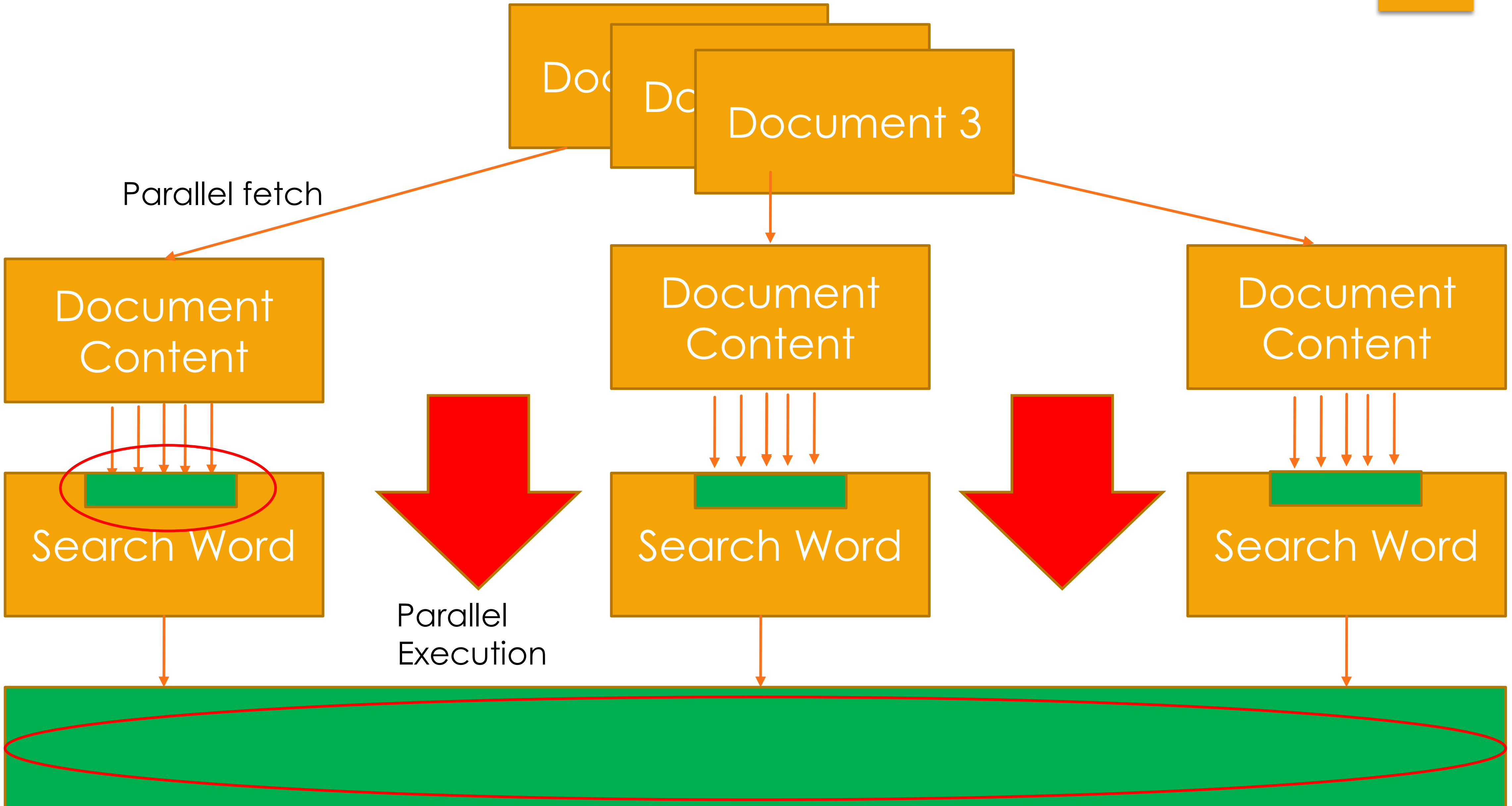
Sequential Execution



Parallal Execution

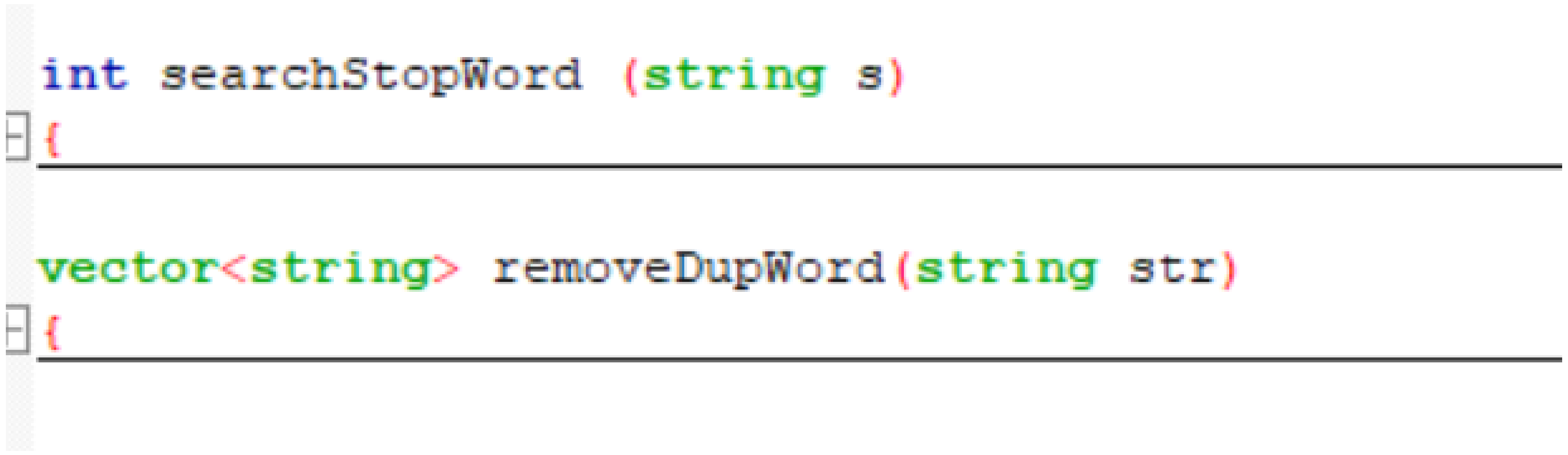


Parallal Execution 2



REMOVAL OF STOP WORDS

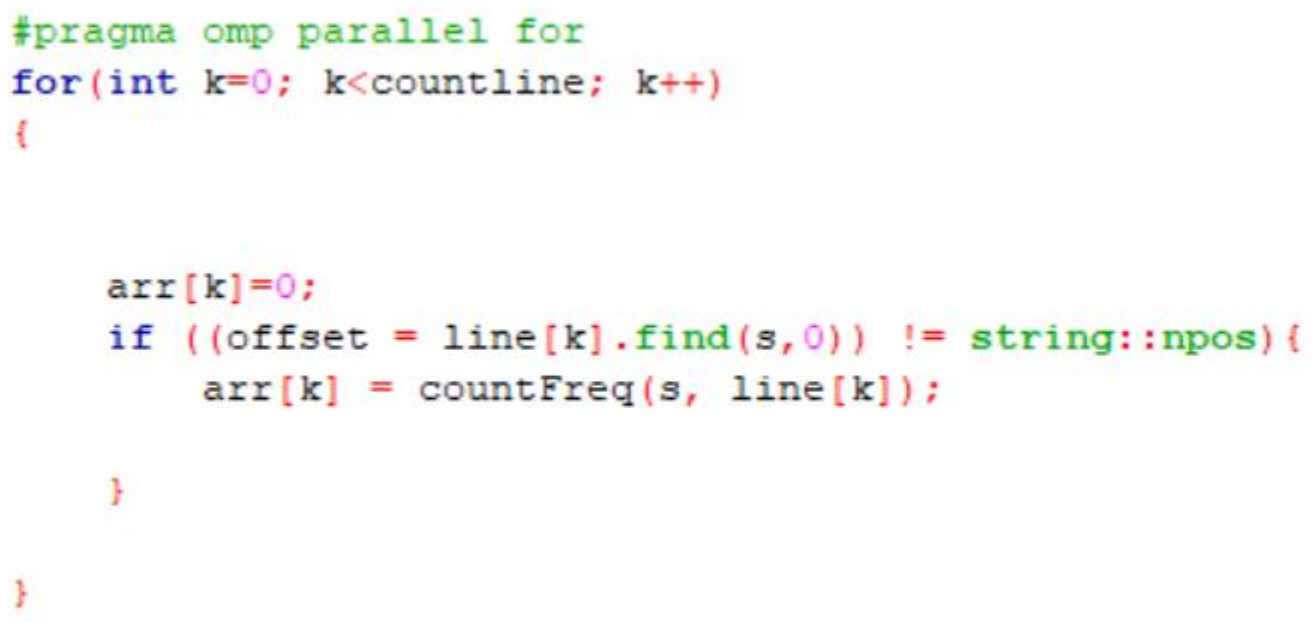
- THE TWO FUNCTIONS



```
int searchStopWord (string s)
{

vector<string> removeDupWord(string str)
{
```

Parallel Search

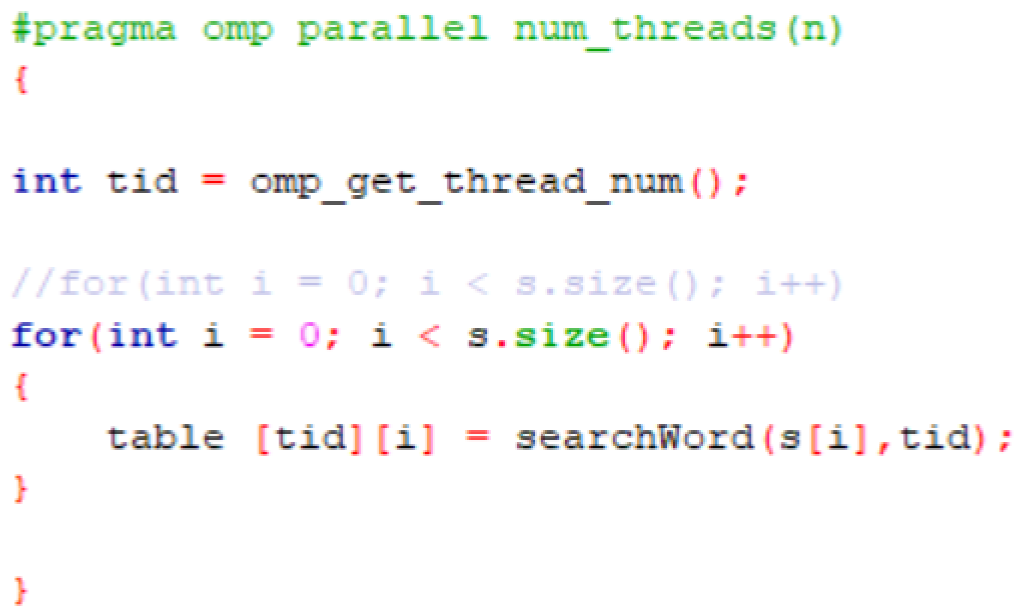
- INSIDE SEARCH WORD



```
#pragma omp parallel for
for(int k=0; k<countline; k++)
{


    arr[k]=0;
    if ((offset = line[k].find(s,0)) != string::npos){
        arr[k] = countFreq(s, line[k]);

    }

}
```

Parallel Search

- INSIDE INT MAIN



```
#pragma omp parallel num_threads(n)
{

int tid = omp_get_thread_num();

//for(int i = 0; i < s.size(); i++)
for(int i = 0; i < s.size(); i++)
{
    table [tid][i] = searchWord(s[i],tid);
}

}
```

COMPARISON

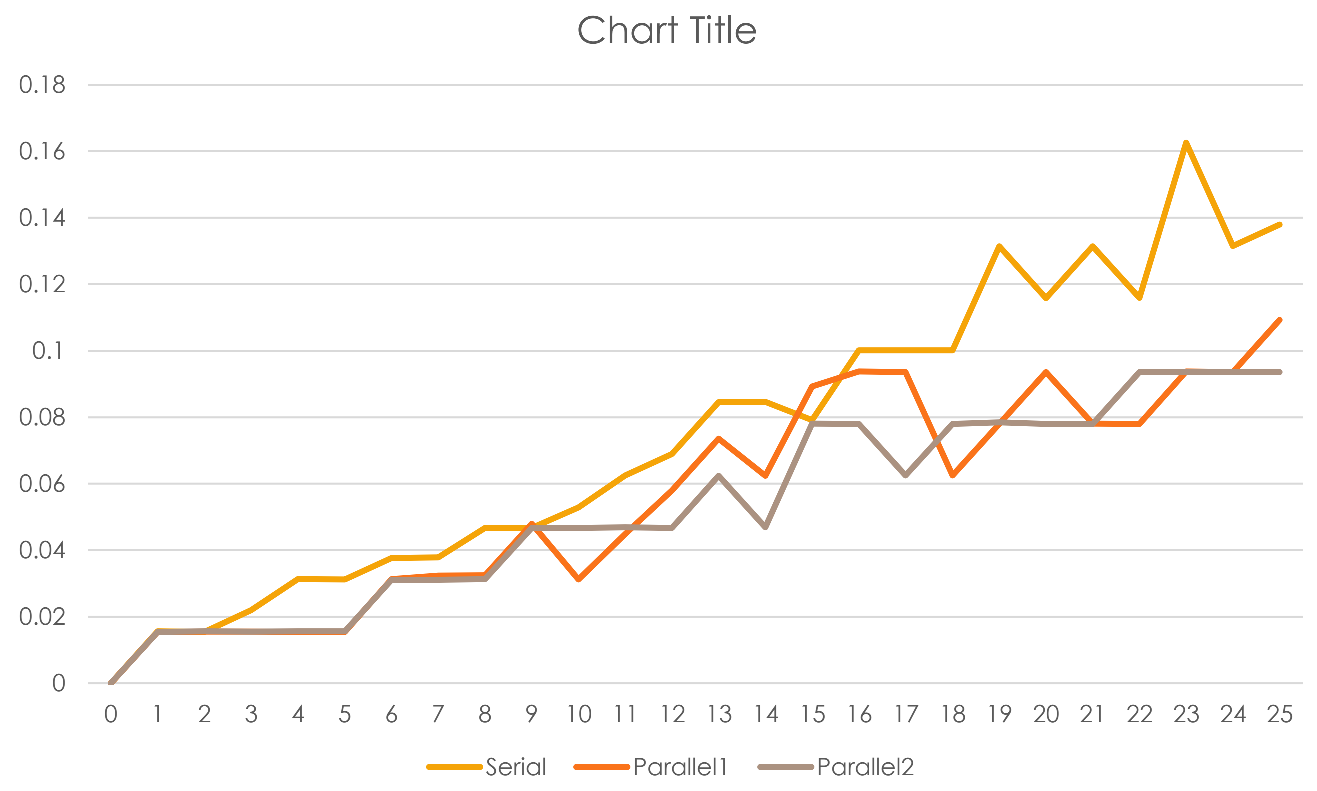


Chart Title
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Serial
Parallel1
Parallel2

SPEED UP

- SPEED UP 1 – 1.366340
- SPEED UP 2 – 1.413178

COMPARISON

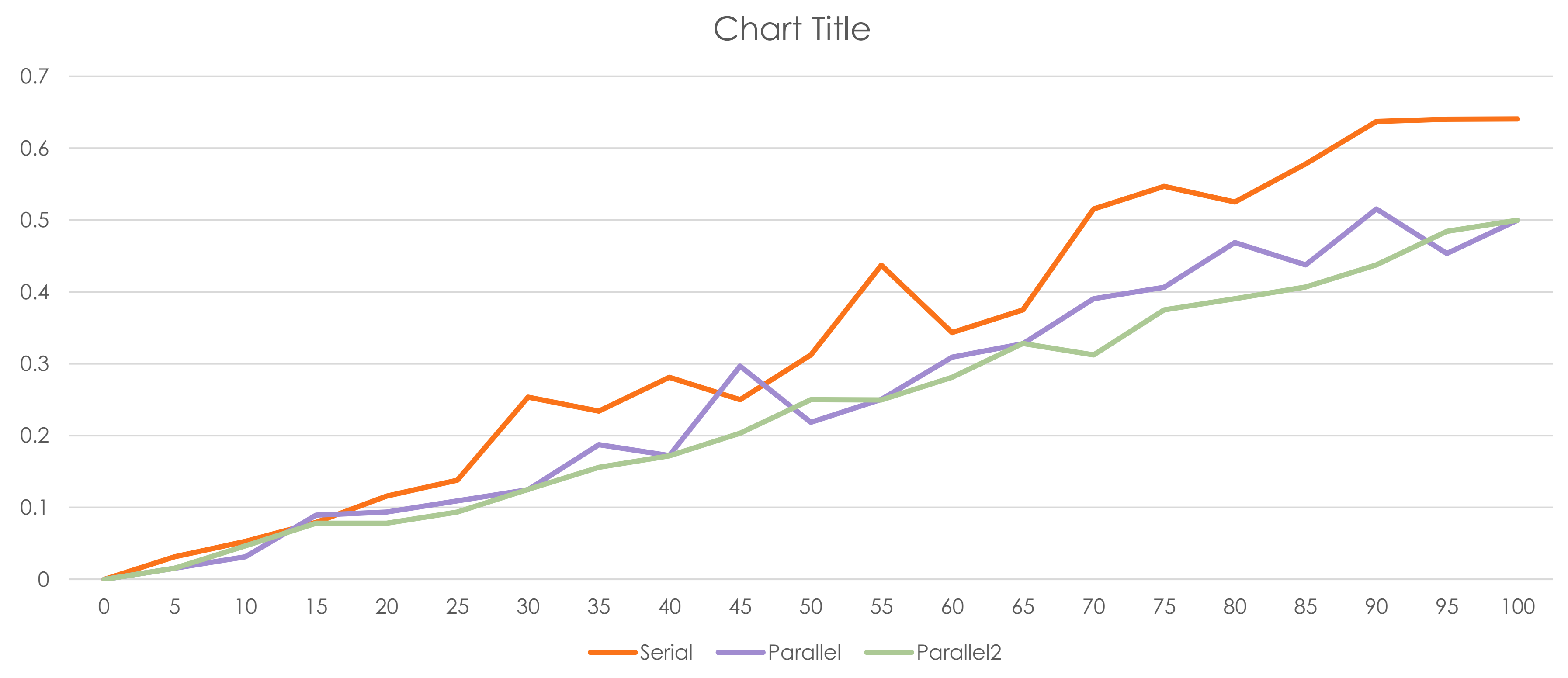


Chart Title
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
Serial
Parallel
Parallel2

SPEED UP

- SPEED UP 1 – 1.365930
- SPEED UP 2 – 1.43990