A STUDY ON SERIAL AND PARALLEL WORD SEARCH PROCEDURES

[1]R.Ananda Devi, [2]R.Chithra Devi
[1]Assistant Professor, Department of Computer Science, Govindamal Aditanar College for Women, Tiruchendur
[2]Assistant Professor, Department of IT, Dr.Sivanthi Aditanar College of Engineering, Tiruchendur

ABSTRACT

Word search is used everywhere from local page search (Cntrl + F) to searching words on document viewer like "reader" in windows. In fact a whole branch called Information Retrieval was developed for this. This project was actually inspired by Information Retrieval. It has a lot of application in real word.As the name suggests "word search" is about searching words in documents parallely using OpenMP. This is not just a simple word search but it also ranks the documents based on the relevance of the documents with respect to the searched word. The documents are scanned word by word and a dictionary is maintained so that words can be searched. Since documents can have several thousands of terms scanning each term can take a lot of time and hence there is a lot of scope for parallelization.

Keywords: Open MP, Sequential, Parallel, C++, Thread

INTRODUCTION

This paper focuses on the principle of multithreaded systems. It uses multiple threads to read multiple files and perform the action. The action here being searching the word through the files, and doing so in very less time as compared to the sequential system. The parameters are similar to that of the sequential method, but the processors are used to do the sequential process on multiple files at the same time. It can be done through either sequential or multithreaded search works.Sequential method spokes about updates like Panda and Hummingbird and highlighted how important semantics in search is, in modern SEO strategies.Search strings are more conversational now, they are of the long-tail variety and are often, context rich. Traditionally, keyword research involved building a list or database of relevant keywords that we hoped to rank for. Often graded by difficulty score, click through rate and search volume, keyword research was about finding candidates in this list to go create content around and gather some organic traffic through exact matching.

A simple method of sequential method is used to search the file of the given file and is shown in fig. 1. The method is quite simple, input the file, read the file, input the word which needs to be searched, and search the file, and give output that it is found or not. Multithreaded method uses multithreaded library and declare multiple threads to handle each file [1, 3]. Functions are defined in the thread headerfile**.** Std::thread is the thread class that represents a single thread in C++. To start a thread we simply need to create a new thread object and pass the executing code to be called (i.e. a callable object) into the constructor of the object. Once the object is created a new thread is launched which will execute the code specified in callable.Word search process can be done by various parallel programming paradigms such as OpenMP, MPI, etc. [2, 4].It searches for words in multiple text files paralleland the concept is shown in fig. 2. This reduces the execution time of the code as compared to the sequential word search. The documents are scanned word by word and a dictionary is maintained so that words can be searched. Since documents can have several thousands of terms scanning each term can

take a lot of time and hence there is a lot of scope for parallelization. These approaches can be used in various applications such graph coloring, 8-directional array P system [5] and so on.

The purpose of this paper is to analysis which word search method is efficient to handle the multiple files. The conclusion will give us the sufficient reasons to choose the method. This could be helpful for future word searing engines

.

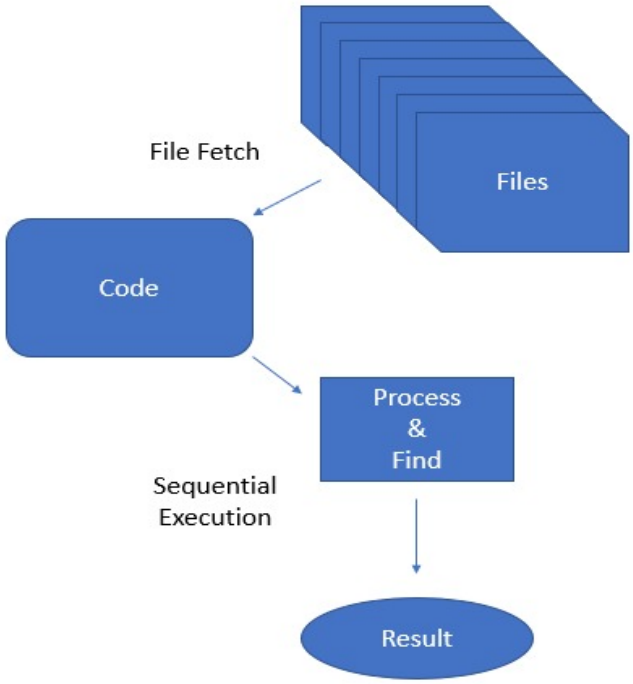

Fig. 1 Sequential model of word search

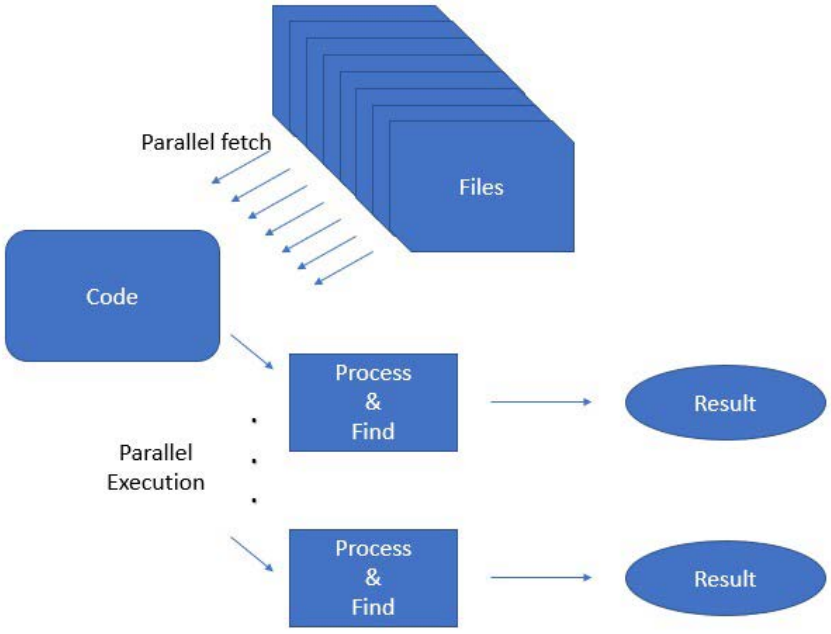

Fig. 2 Parallel model of word search

METHODOLOGY

The divide and conquer paradigm is used to find the optimal solution for the problem. Its basic idea is to decompose a given problem into two or more similar, but simpler, sub problems, to solve them in turn, and to compose their solutions to solve the given problem. Problems of sufficient simplicity are solved directly. For example, to sort a given list of *n* natural numbers, split it into two lists of about *n*/2 numbers each, sort each of them in turn, and interleave both results appropriately to obtain the sorted version of the given list. This approach is used in the merge sort algorithm.

An important application of divide and conquer is in optimization, where if the search space is reduced ("pruned") by a constant factor at each step, the overall algorithm has the same asymptotic complexity as the pruning step, with the constant depending on the pruning factor (by summing the geometric series); this is known as prune and search. For the implementation, the basic code is written in C++ which uses OpenMP to parallelize the code.The documents are located in a folder and file names are given as somecollection. The files are taken in a parallel way and all the data is collectively stored in avector. Then the vectors are sent to a search method that parallelize the search and concurrently browses through the workingdocuments. 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.

RESULT AND DISCUSSIONS

A sample of 65 files is used, each containing around 1500 words and compared the search time of both parallel and sequential codes.The code can search for statements as well as words.If the user gives a statement as input our code will break the statement into an array of words and search for each word in the files sequentially.For this we have created a method called substring which takes the starting index, ending index and the string as input parameters. At last it find the count of all the word in the files and sums it all.The output will be the count of occurrences of statement in the files. Statement entered “Dahl is a great writer”

Sequential code

Sequential time = 1.53 seconds

Parallel code

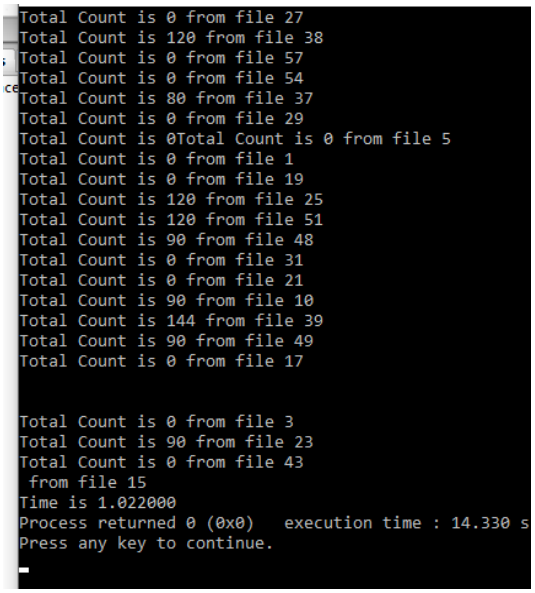

Fig. 3 Parallel search time of problem 1

Parallel search time of problem 1(when number of threads is equal to the number of files to be searched i.e 65)=1.022.

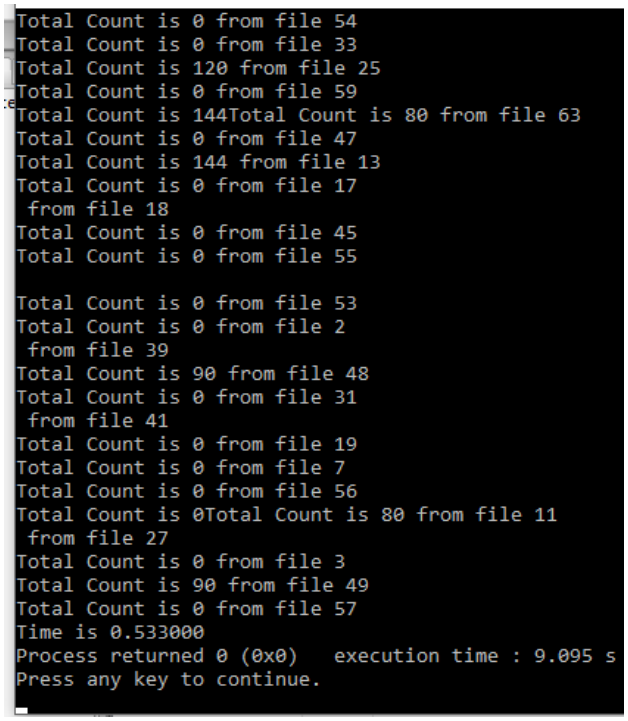

Fig. 4 Parallel search time of problem 2

Parallel search time of problem 2(when number of threads is equal to half the number of files to be searched i.e 32)=0.533 seconds

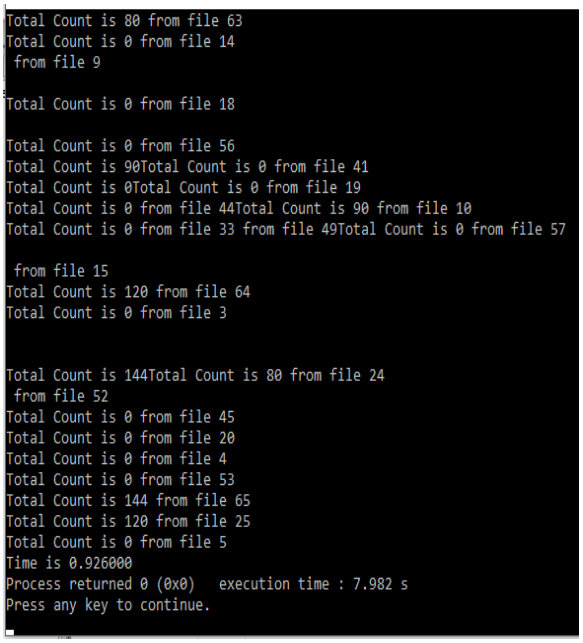

Fig. 5 Parallel search for problem 3

Parallel search timeof problem 3 (when number of threads is equal to one fourth the number of files to be searched I.e 15)=0.926000 seconds

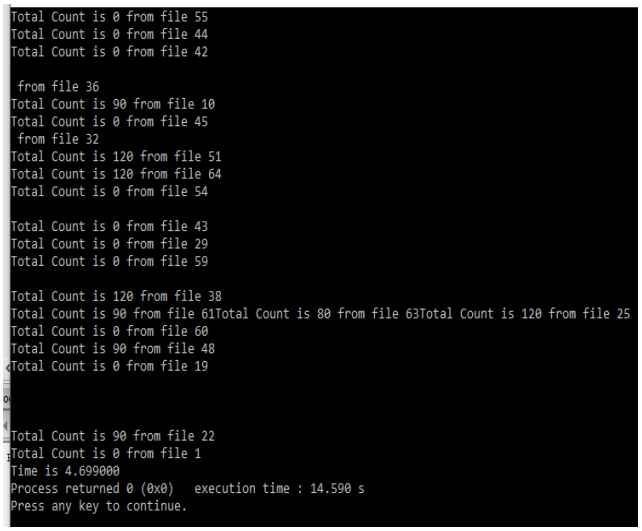

Fig. 6 Parallel search for problem 4

Parallel search time of problem 6 (when number of threads is far greater than number of files to be searched i.e 10000)=4.699000 seconds

It is understood, OpenMP follows for join concept.Thus additional overheads incur when we create threads for a program.Thus the number of threads chosen have to such that the parallel code gives the best possible result.When we used number of threads to be 65 we were getting access time which was less than the sequential time but due to the additional overhead of fork and join of threads the result was not the most optimum. When we took the number of threads to be too small, again we were getting the search time to be better than the sequential but not most optimum.When we took the number of threads to be 10000, the search time came out to be 4 seconds which was far worse than the sequential search time. Thus after detailed analysis of the code and taking different number of files and trying out with different number of threads we came to conclusion that for the search time to be least, the number of threads should be around half of the number of files used for less than 100 files to be read. Fig. 7 shows the result of parallel search where X-axis shows the number of threads and Y-axis shows the number search time in seconds.

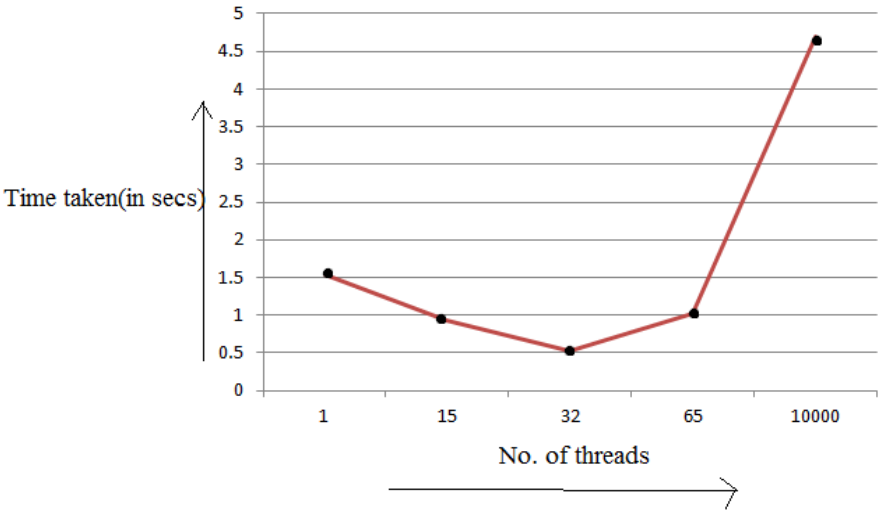

Fig. 7 Time taken for parallel search with respect to number of threads

CONCLUSION

The serial and parallel code was executed successfully and applying parallelism reduces running time significantly. Word search is used everywhere from local page search (Cntrl + F) to searching words on document viewer like "reader" in windows. Information retrieval is another application where word search concepts can be greatly used.

REFERENCES

[1] García-López, Félix, et al. "The parallel variable neighborhoodsearch for the p-median problem", Journal of Heuristics 8.3, 375-388, 2002

[2] Sub, Michael, and Claudia Leopold,"Implementing irregular parallelalgorithms with OpenMP", European Conference on ParallelProcessing, Springer, Berlin, Heidelberg, 2006.

[3]Drews, Frank, Jens Lichtenberg, and Lonnie Welch,"Scalableparallel word search in multicore/multiprocessor systems" TheJournal of Supercomputing, 51.1, 58-75, 2010.

[4]Rabenseifner, Rolf, Georg Hager, and Gabriele Jost, "Hybrid MPI/OpenMPparallel programming on clusters of multi-core SMP nodes", Parallel, Distributedand Network-based Processing, 2009 17th Euromicro International Conferenceon. IEEE, 2009.

[5]JN Rutanshu Jhaveri, Narayanan Prasanth,"Parallel and Serial Graph Coloring implementations with Tabu Search Method", International Journal of Recent Technology and Engineering, 8 (2), 1050-1056, 2019.

[6]W Sureshkumar, K Mahalingam, R Rama,"Pictures and chomsky languages in array P system", International Conference on Membrane Computing, Springer LNCS, 277-289, 2017.