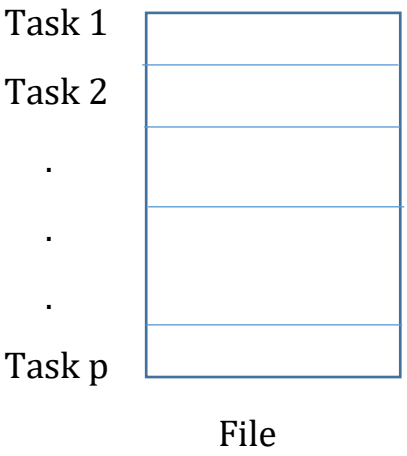
P1.

The task of this problem is to find the given set of words in the file. The query term could be a word or a combination of words to find in the file. For example,

The query “OR w1 w2” will find all the occurrences of w1 and w2 individually. The query “AND w1 w2” will find the occurrence of the words w1 and w2 occurring in a contiguous manner. The file path will also be given as a command line argument.

The key idea for parallelizing this task is in identifying that searching for a term is independent of the region. Therefore, we divide the file into regions and assign each region to a particular task. Each particular task will search the terms in its particular region and send its results back to the main process.



File

Let File size=N

Number of tasks = p

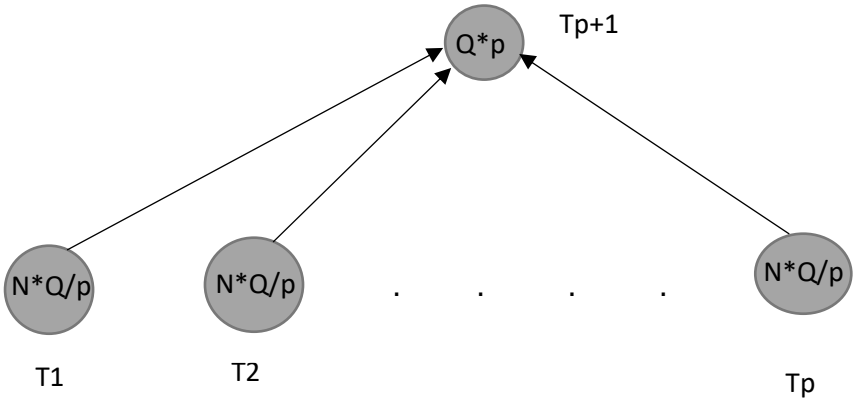
Size for each process = N/p

Size of query = Q

Work for each process= N*Q/p

The technique used here is Data decomposition on the input data, that is the input file is divided among the processes. Each Task will have the whole query item set for searching their respective regions.

Task interaction graph

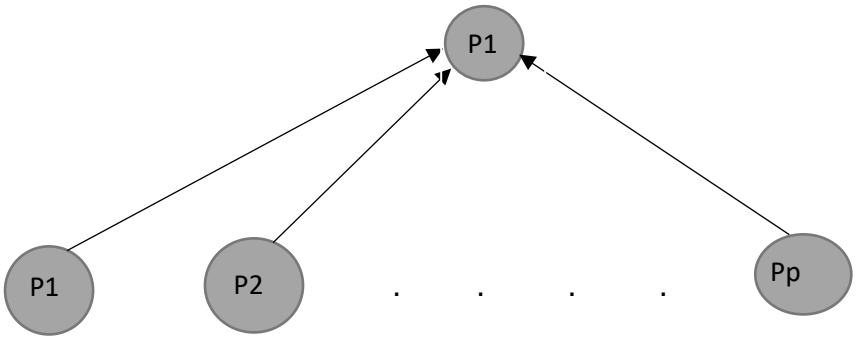


This Figure represents the task interaction graph.

Each process does N*Q/p amount of work. There is one task that combines results of all the processes to get the actual final result. It has to do Q amount of work per process hence it does Q*p amount of work.

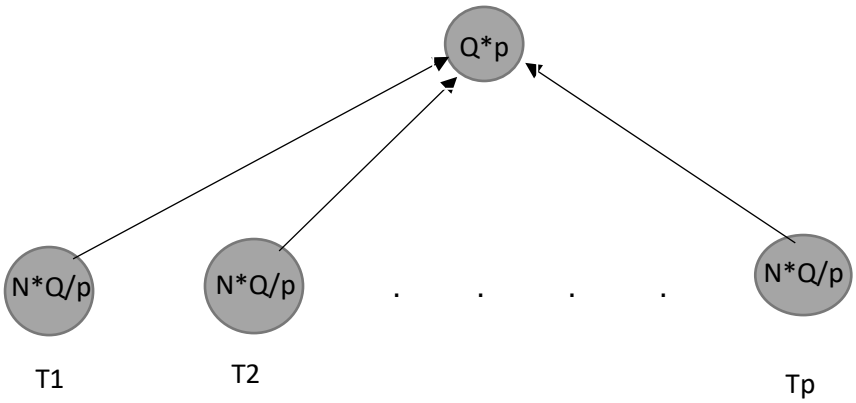
Process mapping

We used the method of mapping based on data partitioning. We follow the "owner computes" rule to operate on the assigned data for a particular process. We assign a process to each task accept the task of combining the results of all the processes. The first process can be reused for this.



Process mapping of the tasks

Each process has been assigned one tasks and each tasks had been assigned a region of the input data. Therefore, each process has its own region on which it computes and passes on the results. The process P1 collects the results from all the processes to get the final actual result.



This Figure represents the task interaction graph.

Critical path length

N*Q/p+Q*p

Eg. For a file of 20 bytes and 4 processes searching for 2 query terms

Length=18

Average degree of concurrency

Total work= N*Q+p*Q

Critical path length=N*Q/p+Q*p

Average degree of concurrency= (N+P)/(N/p +p)

For the above example =2.67

Implementation

The implementation details mimic the algorithm described above.

Initially each process calculates the size of the region that it is assigned and then changes the offset according to its starting point. This offset may fall in between a word so we reset the pointer by reading and discarding until we reach the starting of the next word. From here we keep on reading one word at a time until we either reach the end of file or exceed the data limit assigned to the process. As each word is read, it is compared to each query word and in case of the match, the relative positions are recorded. This is in the case of "OR" queries. In the case of "AND" queries a linked list is maintained of the size equal to the number of words in the query. After reading each word, the new word is added to the end of the list and the head is assigned to the second element. The whole list is compared with the set of words in the query continuously and if a match is found, the position of the first word is recorded.

This is done by all the processes and the results are sent to the first processes.

Speed up and Efficiency

We recorded times for searching a single word in a file of 688 bytes with different number of processes in different instances. The value is an average of 3 readings.

No. of processes	Time(microseconds)	Speedup	Efficiency
1	0.004331	-	
2	0.00279	1.9	95%
3	0.001468	2.95	98.3%
4	0.001109	3.90	97.6%
5	0.000913	4.74	94.8%
6	0.000831	5.21	86.8%

Time taken by a single process=(N/p)*Q

Time for communication =Q*(ts+tw*m)

here m is the size of the result vector for each query word

Speedup=(N*Q/(N/p*Q+Q*(ts+tw*m))

$$=\frac{N}{\frac{N}{p}+(ts+tw*m)}$$

Cost and Scalability

Cost=p*Tp

Tp is the execution time for a single process.

Therefore,

$$Tp=\left(\frac{N}{p}\right)*Q$$

Cost=N*Q

This algorithm is cost optimal

SCALABILITY

$$E=\frac{1}{1+\frac{To}{Ts}}$$

To is the cost of communication which in our case

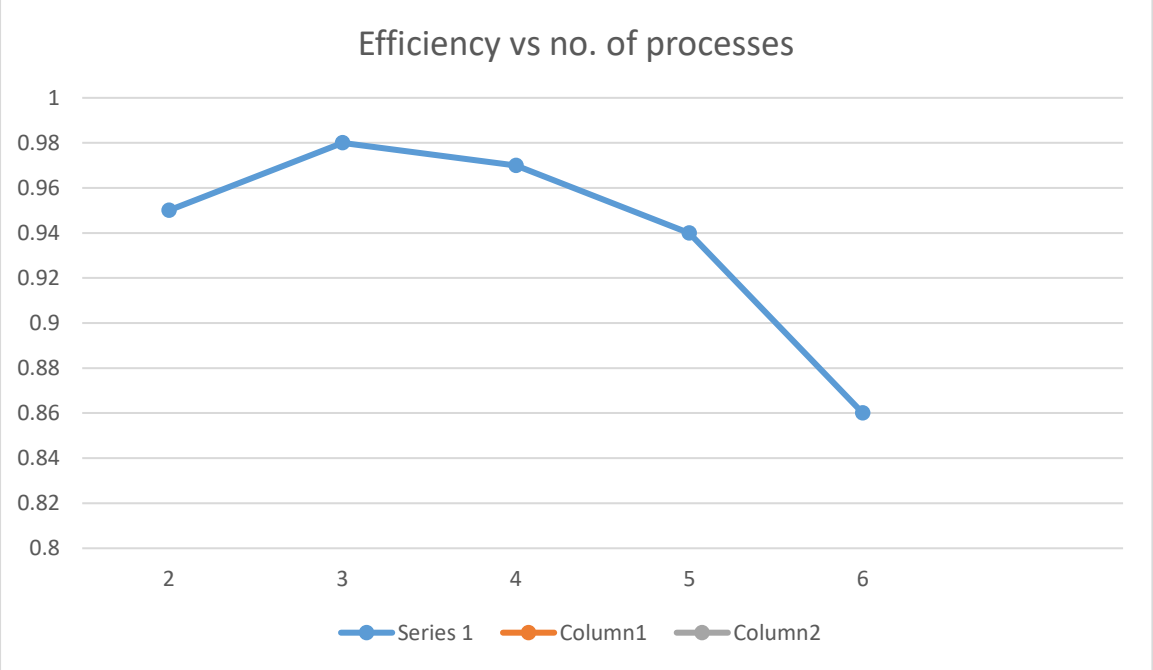
=Q(ts + m * tw)*p

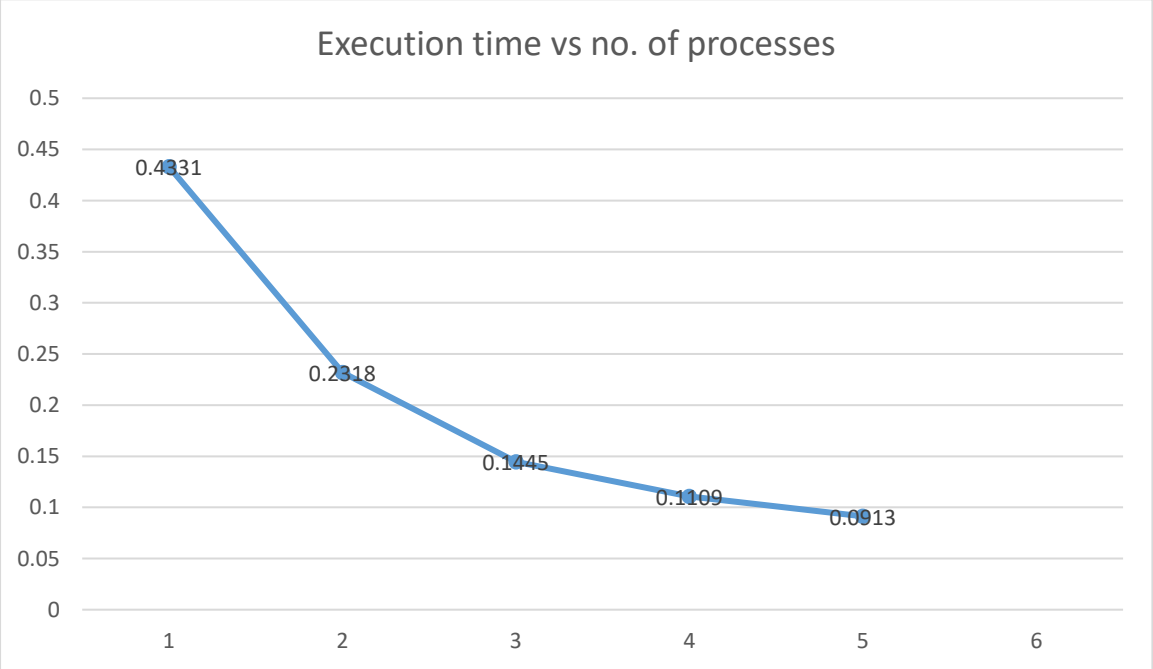
Ts =N*Q

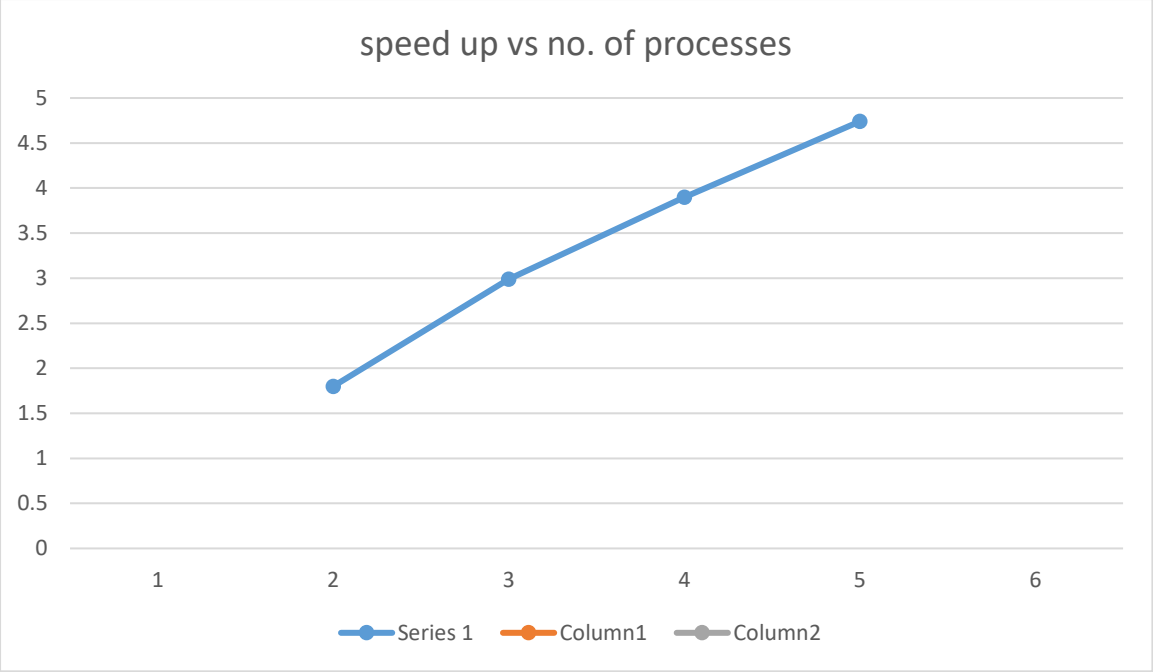
$$T0/Ts=\frac{(ts+m*tw)*p}{N}$$

This algorithm is scalable as To grows similarly as compared to Ts.

GRAPHS


Efficiency vs no. of processes
1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
2
3
4
5
6
Series 1
Column1
Column2


Execution time vs no. of processes
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0.4331
0.2318
0.1445
0.1109
0.0913
1
2
3
4
5
6


speed up vs no. of processes
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1
2
3
4
5
6
Series 1
Column1
Column2