DESIGN DOCUMENT

PARALLEL FILE SEARCH

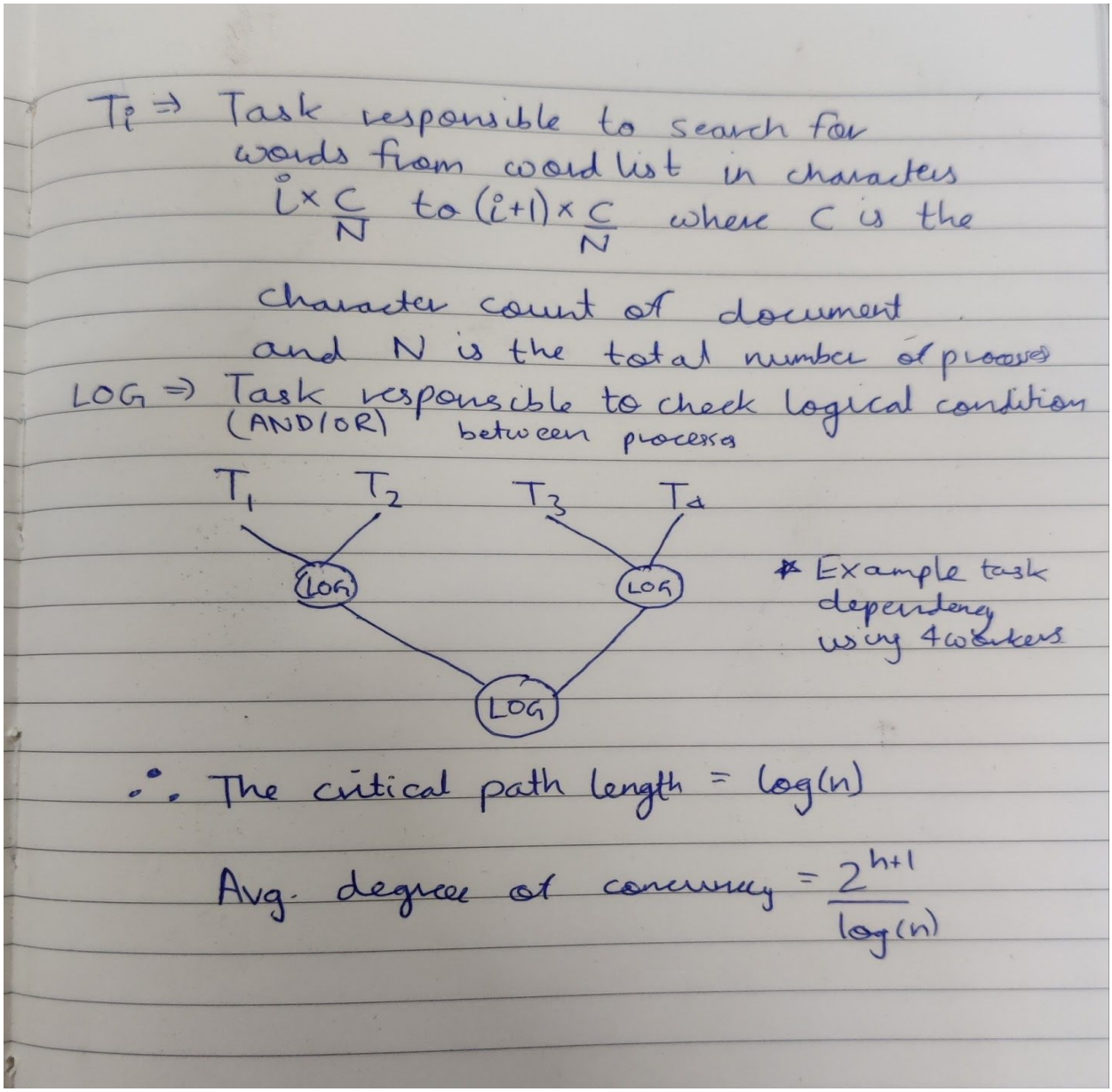
Ameya Bhalerao 2017A3PS0263P

Abhinav Tuli 2017A7PS0048P

The problem statement requires the file to be searched in parallel for the given set of words with AND/OR logical operators.

a) Each task consists of finding occurrences of words from the given word list on a subset of the lines of the file. If the total number of characters are C, and we have N workers, each worker works on C/N characters i.e. Ti involves finding words in characters i*C/N to (i+1)*C/N.

To handle cases where the division of tasks in this manner results in words being cut midway, we consider the entire line till the end as the task.



The tasks communicate in a reductive manner (as in broadcast/reduction in MPI) and the logical AND/OR operator is applied at the end to obtain final results.

Refer the following diagram for task dependency and interaction graphs, along with the calculation for critical path and average degree of concurrency.

b)

Primary approach

i) If the document is so large that reading C/N characters by each process exceeds memory

The processors can do the job in a cyclic manner, thus balancing the load among the N processors. The C characters of the document can be divided into XN parts where X>1. The processor *with ith rank handles all characters of x**i*C/N to (x*i+1)*C/N for x ranging from 1 to X. These characters will be handled by each processor sequentially.

ii) If the document fits in the memory

Divide the C characters into N parts. The processor *with ith rank handles* i*C/N to (i+1)*C/N.

Alternate approach

Let each process traverse the entire document and look for only W/N words where W is word list and N is number of processes. This approach was discarded as it involves a lot of redundancy and doesn't utilize parallelism fully.

We have used only the primary approach along with the ii) assumption.

The logical operations are performed by the process in a reductive manner (the same as in broadcast/reduction in MPI)

c) Parallel algorithm

The following describes the parallel algorithm designed:

1. Each process calculates the total size of the file and divides it by the number of processes.
2. This number is used to calculate the section of the file to be searched by the current process.
3. Each process searches for the given words in its own section.
4. Each process has a local buffer to store line number and line address (which will be used to access the line to print it later in constant time complexity O(1)).
5. Whenever a word is found, the local line number, the local line start address (local to each process) is stored.
6. After each process is done with searching for words, MPI_Scan is used to get partial sums, in order to convert local line numbers to global values.
7. MPI_Reduce (with MPI_MIN) is used on word line and address numbers so that the first occurrence of each word is found.
8. After MPI_Reduce, Process 0 checks if the logical condition is satisfied by iterating over the word list once.
9. If logical conditions are satisfied, Process 0 prints the line number, line (using the line start address) and the position of the word.

Usage:

mpicc search_parallel.c

mpiexec -n 4 search_parallel.exe “filename” “AND/OR” “word1” “word2” ...

(d)

The code has been tested against multiple files. The graph consists of three tests carried out on files of n characters with n = 10e6, n = 10e9 and n = 10e12.

The number of processes used are varied from 1 to 10.

N = 10e6 character file

# of processes	Parallel Runtime	Speedup	Efficiency	Cost
1	2.616352	1	100	2.616352
2	1.483084	1.76289	88.1445	2.966168
3	1.052564	2.48569	82.85634	3.157692
4	0.819215	3.12978	78.2445	3.27686
5	0.835952	3.19373	63.8746	4.17976
6	0.774918	3.37629	56.2715	4.649508
7	0.759595	3.4544	49.34857	5.317165
8	0.701231	3.73108	46.6385	5.609848
9	0.675128	3.87534	43.05933	6.076152
10	0.612444	4.27198	42.7198	6.12444

N = 10e9 character file

# of processes	Parallel Runtime	Speedup	Efficiency	Cost
1	6.525111	1	100	6.525111
2	4.013085	1.62595	81.2975	8.02617
3	3.030089	2.15343	71.781	9.090267
4	2.708773	2.40888	60.222	10.83509
5	2.196015	2.97134	59.4268	10.98007
6	1.992373	3.27504	54.584	11.95424

7	1.806517	3.61198	51.59971	12.64562
8	1.784332	3.65689	45.71112	14.27466
9	1.724576	3.7836	42.04	15.52118
10	1.684052	3.87464	38.7464	16.84052

N = 10e12 character file

# of processes	Parallel Runtime	Speedup	Efficiency	Cost
1	13.62537	1	100	13.62537
2	8.00301	1.70253	85.1265	16.00602
3	5.7826	2.35627	78.54234	17.3478
4	4.89394	2.78413	69.60326	19.57576
5	4.71848	2.88766	57.7532	23.5924
6	3.92033	3.47556	57.926	23.52198
7	3.6056	3.77894	53.98486	25.2392
8	3.45241	3.94662	49.33275	27.61928
9	3.39606	4.01211	44.579	30.56454
10	3.30353	4.12448	41.2448	33.0353

The system is scalable.

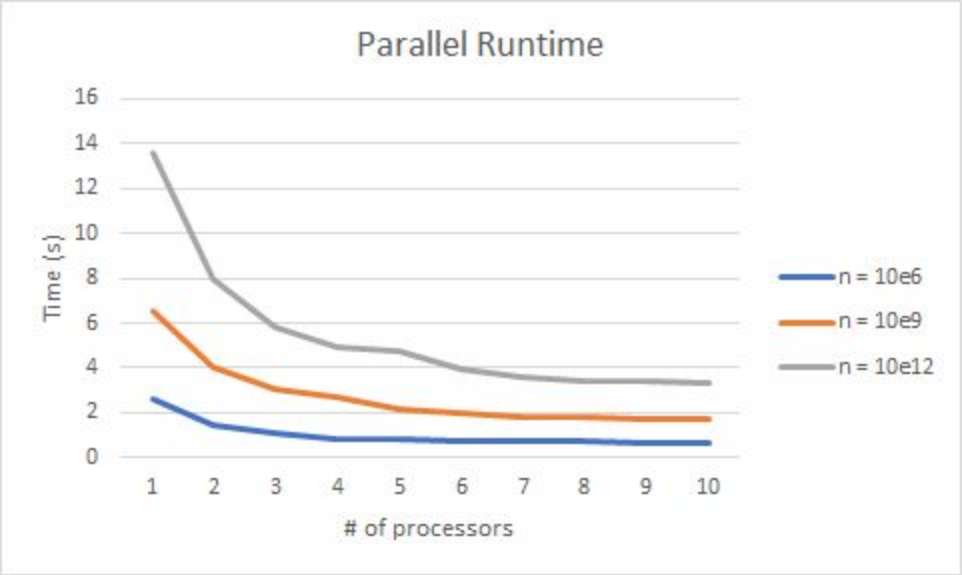
The system divides the file into different processes and each process works in a specified region to find a word.

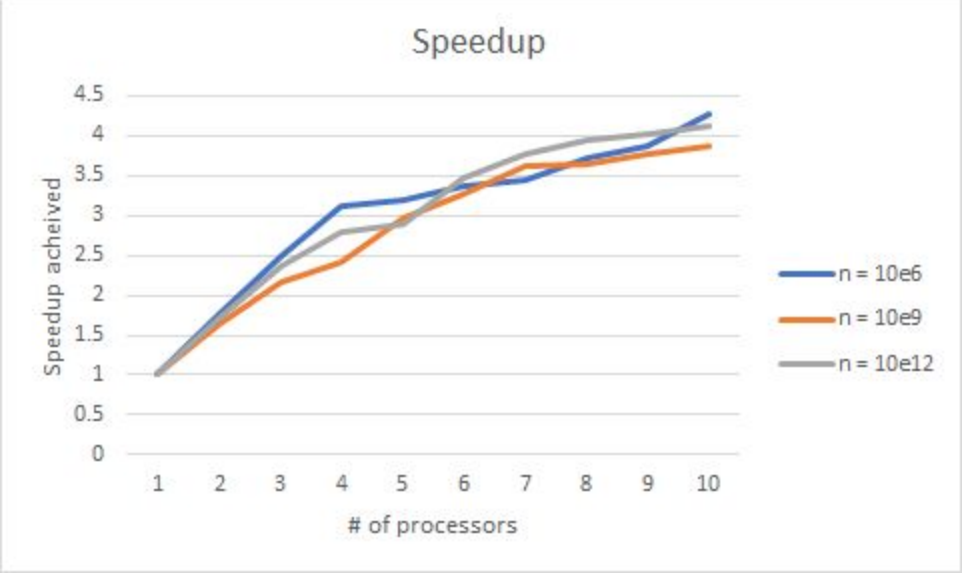
Thus the complexity of the algorithm ~ O(N/P), p = number of processes and n = size of file.

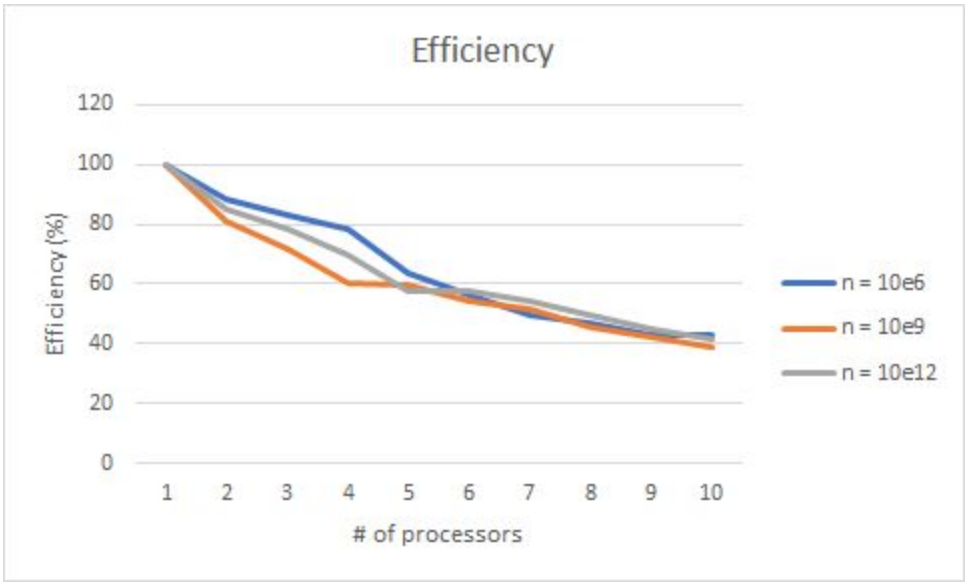
Thus even if the input data is increased drastically, the efficiency of the algorithm will be constant if processors are increased.

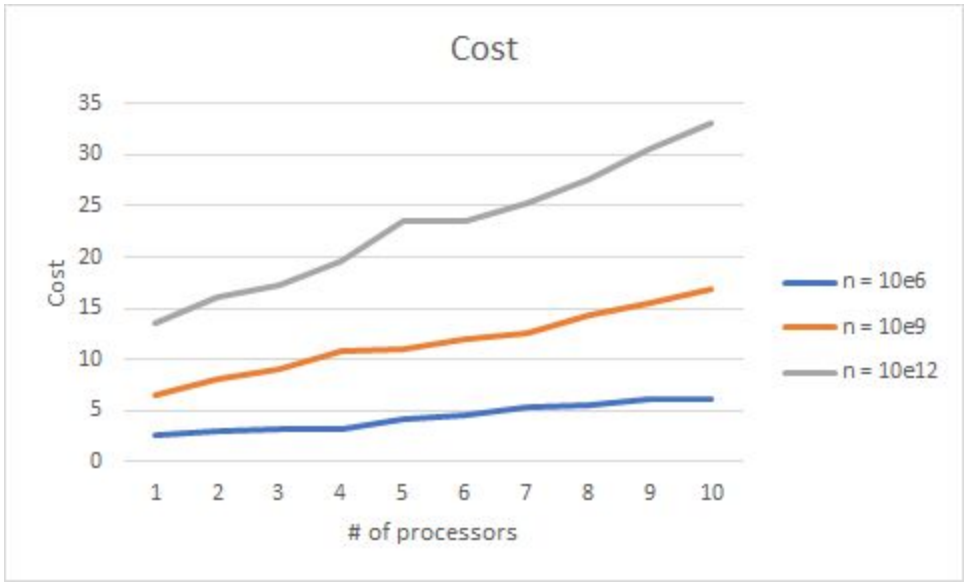
(e)

The code has been tested against multiple files. The graph consists of three tests carried out on files of n characters with n = 10e6, n = 10e9 and n = 10e12.
The number of processes used are varied from 1 to 10.






Efficiency
120
100
80
60
40
20
0
Efficiency (%)
1
2
3
4
5
6
7
8
9
10
# of processors
n = 10e6
n = 10e9
n = 10e12


Cost
35
30
25
20
15
10
5
0
Cost
1
2
3
4
5
6
7
8
9
10
# of processors
n = 10e6
n = 10e9
n = 10e12