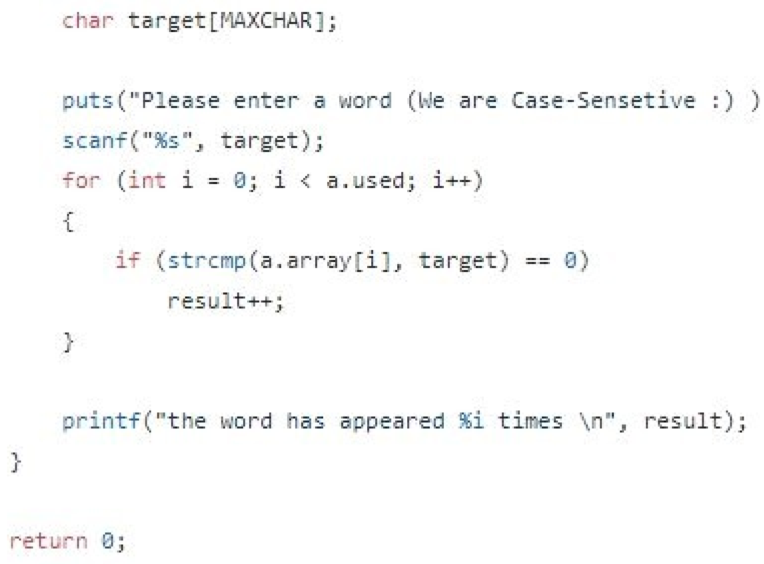
Report

Objective of program

Program objective is to scan for target input and count the number of times this input have appeared in a text file finally print the count.

Sequential Version

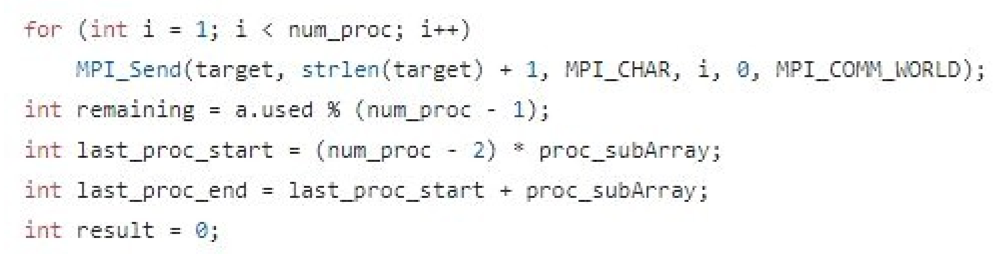
Program counts through a single for loop that iterates through the whole file and compare each word with the target word, finally incrementing the counter (result) if the word is the same as the target word and printing the result.



```
    char target[MAXCHAR];

    puts("Please enter a word (We are Case-Sensetive :) ) : ");
    scanf("%s", target);
    for (int i = 0; i < a.used; i++)
    {
        if (strcmp(a.array[i], target) == 0)
            result++;
    }

    printf("the word has appeared %i times \n", result);
}

return 0;
```

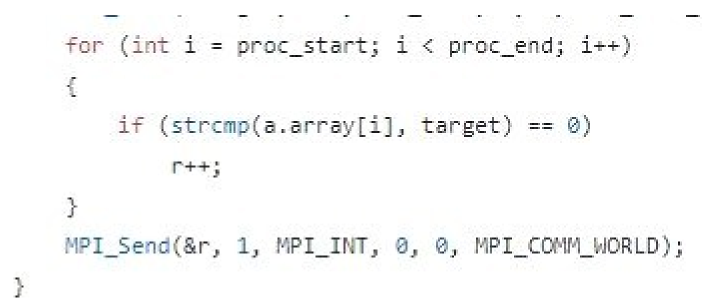
Send and Receive Version

Program creates 4 processes or nodes 0 is mater node and 3 slave nodes, this helps in achieving High Performance Computing through parallelism. Execution is done through Message Passing Interface after intializing the processes , process of rank zero scans for input of the target (word to search for) , then it sends the target to the three nodes through MPI_Send predefined function this is done inside this loop and if there is a remainder of the array after splitting node 0 searches through the remainder of the array for thetarget



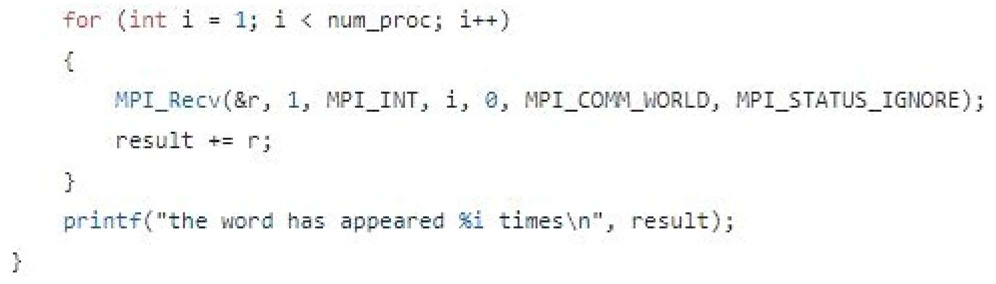
```
for (int i = 1; i < num_proc; i++)
    MPI_Send(target, strlen(target) + 1, MPI_CHAR, i, 0, MPI_COMM_WORLD);
int remaining = a.used % (num_proc - 1);
int last_proc_start = (num_proc - 2) * proc_subArray;
int last_proc_end = last_proc_start + proc_subArray;
int result = 0;
```

Each node received the message from node 0 with the target word through MPI_Receive and loops through a **subarray** instead of the whole array, start and the end of the subarray depends on the rank of the process.Then counting the number ofappearances of target inside subarray and incrementing the counter.The counter of each node is send to node 0 to find the whole sum up all the counter which is done through MPI_Send of integervalue



```
    for (int i = proc_start; i < proc_end; i++)
    {
        if (strcmp(a.array[i], target) == 0)
            r++;
    }
    MPI_Send(&r, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
}
```

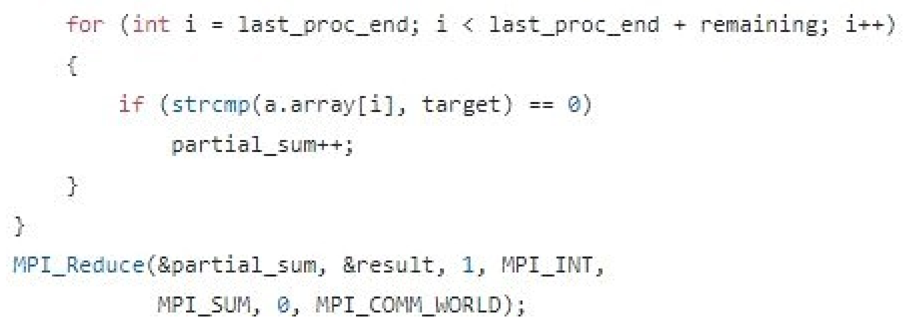
and on the other side node 0 receives the counter from each node and add it to the whole counter



```
    for (int i = 1; i < num_proc; i++)
    {
        MPI_Recv(&r, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        result += r;
    }
    printf("the word has appeared %i times\n", result);
}
```

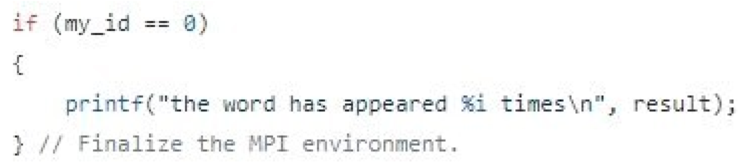
Broadcast and Reduce Version

Program uses splits the array into subarrays according to processes and each process gets a subarray to search for target.Node 0 scans for input stringafterthen broadcast string target from node 0 to all other processes through**one call**of MPI_Bcast. Then The routine MPI_Reduce is called by each proccess this combines data from all processes (by adding the sounters in this case), and returning the result to a single (process node 0).



```
        for (int i = last_proc_end; i < last_proc_end + remaining; i++)
        {
            if (strcmp(a.array[i], target) == 0)
                partial_sum++;
        }
    }
    MPI_Reduce(&partial_sum, &result, 1, MPI_INT,
               MPI_SUM, 0, MPI_COMM_WORLD);
```

Instead of summing the counters of all the counters from all processes reduce sums all the partial_sum and storing the total inside result. Finally ,node 0 prints out the result.



```
if (my_id == 0)
{
    printf("the word has appeared %i times\n", result);
} // Finalize the MPI environment.
```

Scatter and Gather Version

In this version scatter was used to distribute the elements of the array of strings that was extracted from the file. Performance is expected to improve as now each process has exactly its own array to search. There is no fight between processes on the same memory location and each one doesn't have to read the file. However, memory consumed seems to be larger. Gather was not used as it does not serve the purpose of this project and was going to be useless and redundant in this case.

<u>Analysis</u>

Parallelizable code

Total number of lines in source code = 100

number of lines parallelizable =5

then on source code level = 5%

Loop iterates through the length of the array , the array size is on average 1000 then on the machine code level   the program does the parallelizable 5 lines 5000 times so it is almost 100%parallelizable

on machine code level= 100%