Ex-4	Parallel File Access and parallel for, reduction ,critical
4/08/2025	

AIM:

To

1)Write a C++ program with OpenMP directives that performs parallel file access.

2)Implement the Caesar Cypher (both encryption and decryption) in C++ using OpenMP. Compute the strong scaling and weak scaling graphs.

3)Compute a frequency histogram of characters in a large file. Use reduction, critical, and compare the speedup between the two methods.

ALGORITHM:

1)

a)start
b)include necessary headers
c) read the files of different sizes using parallel for
d)stop

2)

a)start
b)include necessary headers
c)implement encrypt,decrypt,generatetext functions
d)get textsize and threadnum from user and encrypt decrypt and measure time
e)stop

3)

a)start
b)include necessary headers
c)implement functions for finding frequency using critical and reduction
d)call the functions and calculate time
e)stop

CODE:

1)

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <omp.h>


using namespace std;


std::size_t read_file(const std::string &filename) {
    std::ifstream file(filename, std::ios::binary | std::ios::ate);

    omp_set_num_threads(4);
    std::size_t size=file.tellg();
    file.seekg(0,std::ios::beg);

    std::vector<char> buffer(size);
    file.read(buffer.data(), size);

    std::cout << "Thread " << omp_get_thread_num()
              << " read " << size << " bytes from " << filename << std::endl;

    return size;
}

int main(){

std::vector<std::string> filenames = {
        "random_data_file_1024.bin",
        "random_data_file_1048576.bin",
        "random_data_file_1073741824.bin"
    };

    #pragma omp parallel for
    for (int i = 0; i < filenames.size(); ++i) {
        read_file(filenames[i]);
    }

    return 0;

}
```

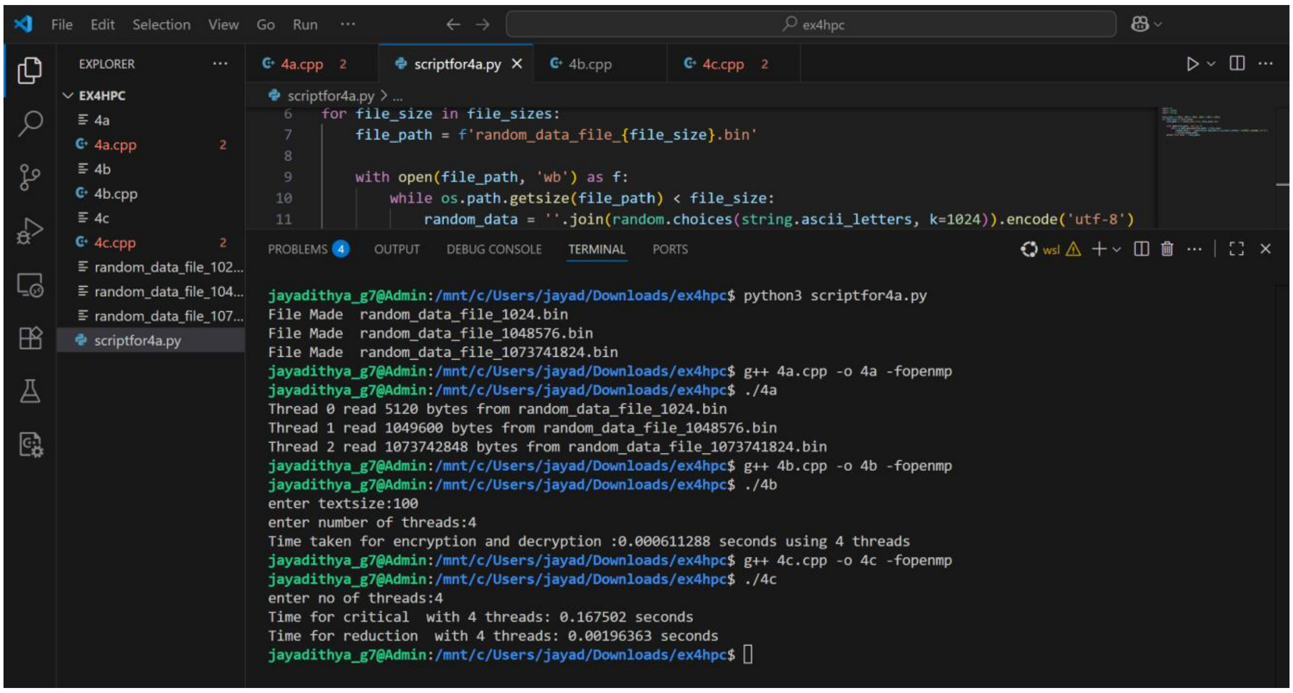
2)

```
#include <iostream>
#include <fstream>
#include <string>
#include <omp.h>
#include <chrono>

using namespace std;

void encrypt(string& text,int shift,int num_threads) {
    int length=text.length();

    #pragma omp parallel for num_threads(num_threads)
    for(int i=0;i<length;++i){
        if(isalpha(text[i])) {
            char base=isupper(text[i])?'A':'a';
            text[i]=(text[i]-base+shift)%26+base;
        }
    }
}

void decrypt(string& text,int shift,int num_threads) {
    encrypt(text,26-shift,num_threads);  // decryption is just reverse shift
}

string generateText(int size){
    string result;
    result.reserve(size);
    for (int i=0;i<size;++i) {
        result+='A'+(rand()%26);
    }
    return result;
}

int main(){


    int text_size;
    int shift=3;
    int threadnum;

    cout<<"enter textsize:";
    cin>>text_size;
    string text=generateText(text_size);

   cout<<"enter number of threads:";
   cin>>threadnum;

    auto start=std::chrono::high_resolution_clock::now();
    encrypt(text,shift,threadnum);

    decrypt(text,shift,threadnum);


    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> dur=end-start;

    std::cout<<"Time taken for encryption and decryption :"<<dur.count()<<" seconds using
"<<threadnum<<" threads\n";

    return 0;
}
```

3)

```
#include <iostream>
#include <fstream>
#include <vector>
#include <chrono>
#include <omp.h>


using namespace std;


string readFile(const string& filename) {
    ifstream file(filename,ios::binary);


    return string((istreambuf_iterator<char>(file)),
                    istreambuf_iterator<char>());
}

void frequencyCritical(const string& data,vector<int>& histogram,int num_threads) {
    #pragma omp parallel for num_threads(num_threads)
    for (int i=0;i<data.size();++i) {
        unsigned char ch=data[i];
        #pragma omp critical
        {
            histogram[ch]++;
        }
    }
}


void frequencyReduction(const string& data,vector<int>& histogram,int num_threads) {
    vector<vector<int>> local_hist(num_threads,vector<int>(256,0));

    #pragma omp parallel num_threads(num_threads)
    {
        int tid=omp_get_thread_num();
        #pragma omp for
        for (int i=0;i<data.size();++i) {
            unsigned char ch=data[i];
            local_hist[tid][ch]++;
        }
    }


    for(int t=0;t<num_threads;++t)
        for(int i=0;i<256;++i)
            histogram[i]+=local_hist[t][i];
}

int main() {




    int num_threads;
    cout<<"enter no of threads:";
    cin>>num_threads;

    string data=readFile("random_data_file_1048576.bin");
    std::vector<int> histogram1(256,0);
    std::vector<int> histogram2(256,0);

    auto start = std::chrono::high_resolution_clock::now();


        frequencyCritical(data,histogram1,num_threads);
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> time = end - start;

    std::cout<<"Time for critical  with "<<num_threads<<" threads: " << time.count()<<"
seconds\n";



start = std::chrono::high_resolution_clock::now();
        frequencyReduction(data,histogram2,num_threads);
    end = std::chrono::high_resolution_clock::now();
time = end - start;


    std::cout<<"Time for reduction  with "<<num_threads<<" threads: " << time.count()<<"
seconds\n";

    return 0;
}
```

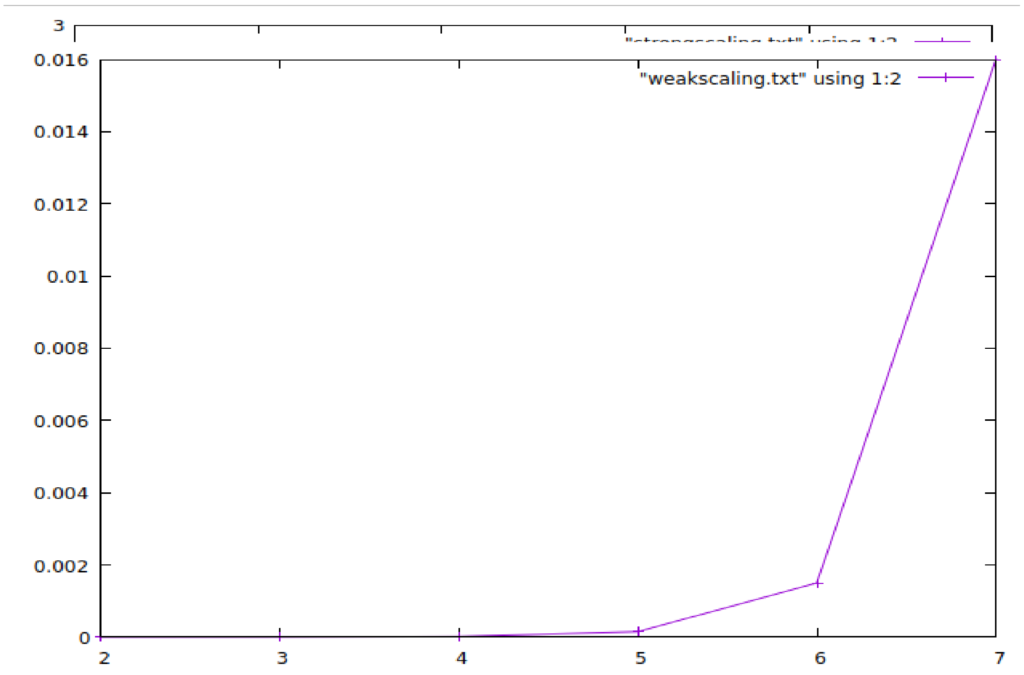
OUTPUT:



```python
for file_size in file_sizes:
    file_path = f'random_data_file_{file_size}.bin'

    with open(file_path, 'wb') as f:
        while os.path.getsize(file_path) < file_size:
            random_data = ''.join(random.choices(string.ascii_letters, k=1024)).encode('utf-8')
```

```
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex4hpc$ python3 scriptfor4a.py
File Made  random_data_file_1024.bin
File Made  random_data_file_1048576.bin
File Made  random_data_file_1073741824.bin
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex4hpc$ g++ 4a.cpp -o 4a -fopenmp
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex4hpc$ ./4a
Thread 0 read 5120 bytes from random_data_file_1024.bin
Thread 1 read 1049600 bytes from random_data_file_1048576.bin
Thread 2 read 1073742848 bytes from random_data_file_1073741824.bin
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex4hpc$ g++ 4b.cpp -o 4b -fopenmp
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex4hpc$ ./4b
enter textsize:100
enter number of threads:4
Time taken for encryption and decryption :0.000611288 seconds using 4 threads
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex4hpc$ g++ 4c.cpp -o 4c -fopenmp
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex4hpc$ ./4c
enter no of threads:4
Time for critical  with 4 threads: 0.167502 seconds
Time for reduction  with 4 threads: 0.00196363 seconds
jayadithya_g7@Admin:/mnt/c/Users/jayad/Downloads/ex4hpc$ 
```

STRONG SCALING WEAK SCALING GRAPH



RESULT:

Hence parallel file access, parallel for,reduction,critical are implemented and executed successfully and output has been verified.