|  |  |
| --- | --- |
| 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;  
  
}

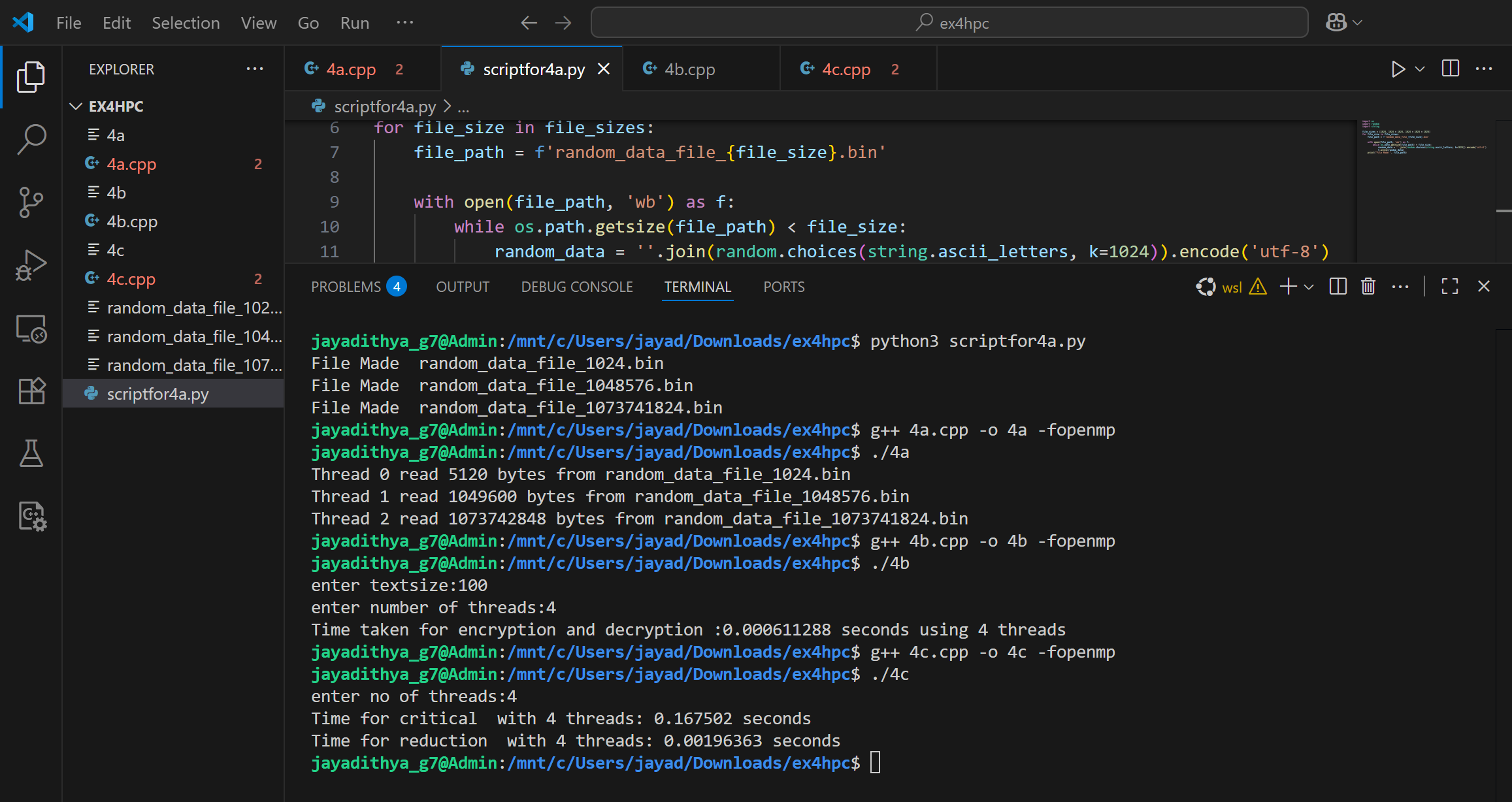
2)

#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;  
}

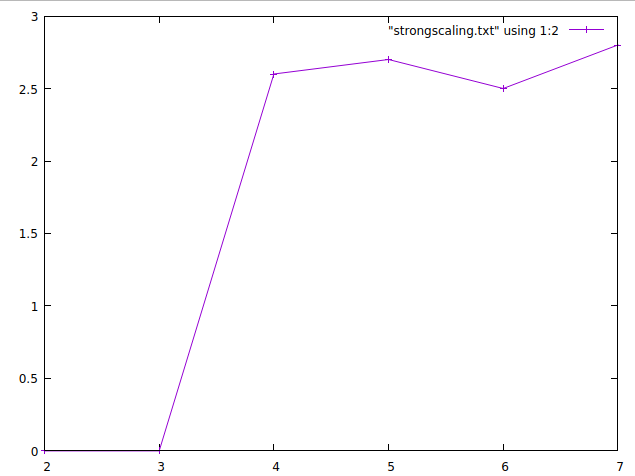
3)

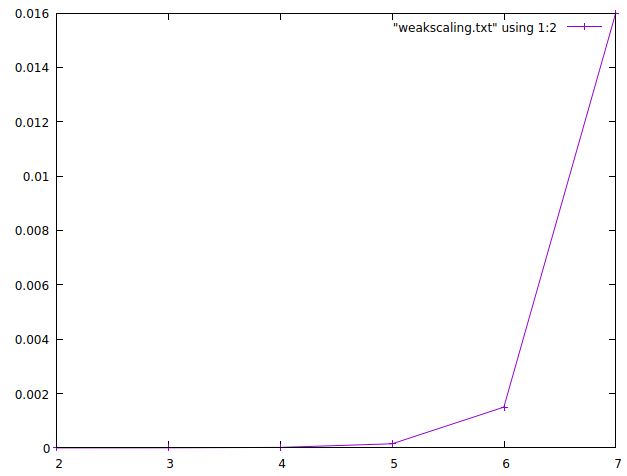
#include <iostream>  
#include <fstream>  
#include <vector>  
#include <chrono>  
#include <omp.h>  
  
  
using namespace std;  
  
  
string readFile(const string& filename) {  
 ifstream file(filename,ios::binary);  
   
   
 return string((istreambuf\_iterator<char>(file)),  
 istreambuf\_iterator<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:



STRONG SCALING WEAK SCALING GRAPH





RESULT:

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