**BCSE205L – COMPUTER ARCHITECTURE**

**DIGITAL ASSIGNMENT II**

**Enhancing** **Vigenère Encryption through parallel**

**Processing using OPEN MP**

**Team No.**

|  |  |
| --- | --- |
| **Member Name1:** | **Sriansh Tripathy** |
| **Member Regno1:** | **22BRS1041** |
| **Contribution:** | **Coding, Encryption and Implementation** |
| **Member Name2:** | **Monish Kumar Ramba** |
| **Member Regno2:** | **22BRS1035** |
| **Contribution:** | **Presentation and Research** |

**SCREENSHOT OF YOUR CODE POST IN YOUR GITHUB LINK WITH THE RESPECTIVE TITLE**

**SUMMARY OF THE PROJECT**

**In this innovative project, parallel programming techniques were seamlessly integrated into the traditionally sequential Vigenere encryption algorithm. Vigenere encryption operates by shifting each character in the plaintext string by a corresponding character in the key. By strategically breaking down the encryption task into smaller chunks, OpenMP was utilized to facilitate simultaneous processing across multiple cores or nodes. This meticulous parallelization led to an exceptional tenfold reduction in processing time, showcasing a remarkable leap in efficiency compared to the conventional sequential approach. This significant speedup was a direct result of the parallel execution of encryption tasks, exemplifying the substantial gains achievable through parallel processing without delving into intricate architectural specifics. This achievement not only showcases the power of parallel programming but also highlights the strategic division of labor within the Vigenere encryption algorithm. By combining the algorithm's inherent logic with parallelization techniques, the project not only optimized efficiency but also delved into the intricate internal workings of the Vigenere algorithm, marking a significant milestone in the realm of cryptography and parallel computing.**

**ARCHITECTURE DIAGRAM**

**CODE:**

**Parallel Programming Approach(OpenMP)**

**#include <iostream>**

**#include <cstring>**

**#include <cctype>**

**#include <omp.h>**

**void encryptVigenere(char \*input, const char \*key) {**

**int keyLength = strlen(key);**

**#pragma omp parallel for**

**for (int i = 0; i < strlen(input); ++i) {**

**int threadID = omp\_get\_thread\_num();**

**//std::cout << "Thread " << threadID << " is processing character at index " << i << std::endl;**

**if (isalpha(input[i])) {**

**char base = isupper(input[i]) ? 'A' : 'a';**

**char keyChar = key[i % keyLength];**

**char keyBase = isupper(keyChar) ? 'A' : 'a';**

**input[i] = (input[i] - base + keyChar - keyBase) % 26 + base;**

**}**

**}**

**}**

**int main() {**

**double start\_time = omp\_get\_wtime(); // Get the start time**

**int numThreads;**

**std::cout << "Enter the number of threads: ";**

**//std::cin >> numThreads;**

**omp\_set\_num\_threads(4);**

**const char \*key = "KEY";**

**char input[] = "Hello, World!";**

**std::cout << "Original string: " << input << std::endl;**

**encryptVigenere(input, key);**

**std::cout << "Encrypted string: " << input << std::endl;**

**double end\_time = omp\_get\_wtime(); // Get the end time**

**std::cout << "Time elapsed: " << end\_time - start\_time << " seconds" << std::endl;**

**return 0;**

**}**

**Serial Programming Approach(C++)**

**#include <iostream>**

**#include <cstring>**

**#include <cctype>**

**#include <ctime>**

**void encryptVigenere(char \*input, const char \*key) {**

**int keyLength = strlen(key);**

**for (int i = 0; i < strlen(input); ++i) {**

**if (isalpha(input[i])) {**

**char base = isupper(input[i]) ? 'A' : 'a';**

**char keyChar = key[i % keyLength];**

**char keyBase = isupper(keyChar) ? 'A' : 'a';**

**input[i] = (input[i] - base + keyChar - keyBase) % 26 + base;**

**}**

**}**

**}**

**int main() {**

**const char \*key = "KEY";**

**char input[] = "Hello, World!";**

**std::cout << "Original string: " << input << std::endl;**

**clock\_t start\_time = clock();**

**encryptVigenere(input, key);**

**clock\_t end\_time = clock();**

**double elapsed\_time = double(end\_time - start\_time) \* 1000.0 / CLOCKS\_PER\_SEC;**

**std::cout << "Encrypted string: " << input << std::endl;**

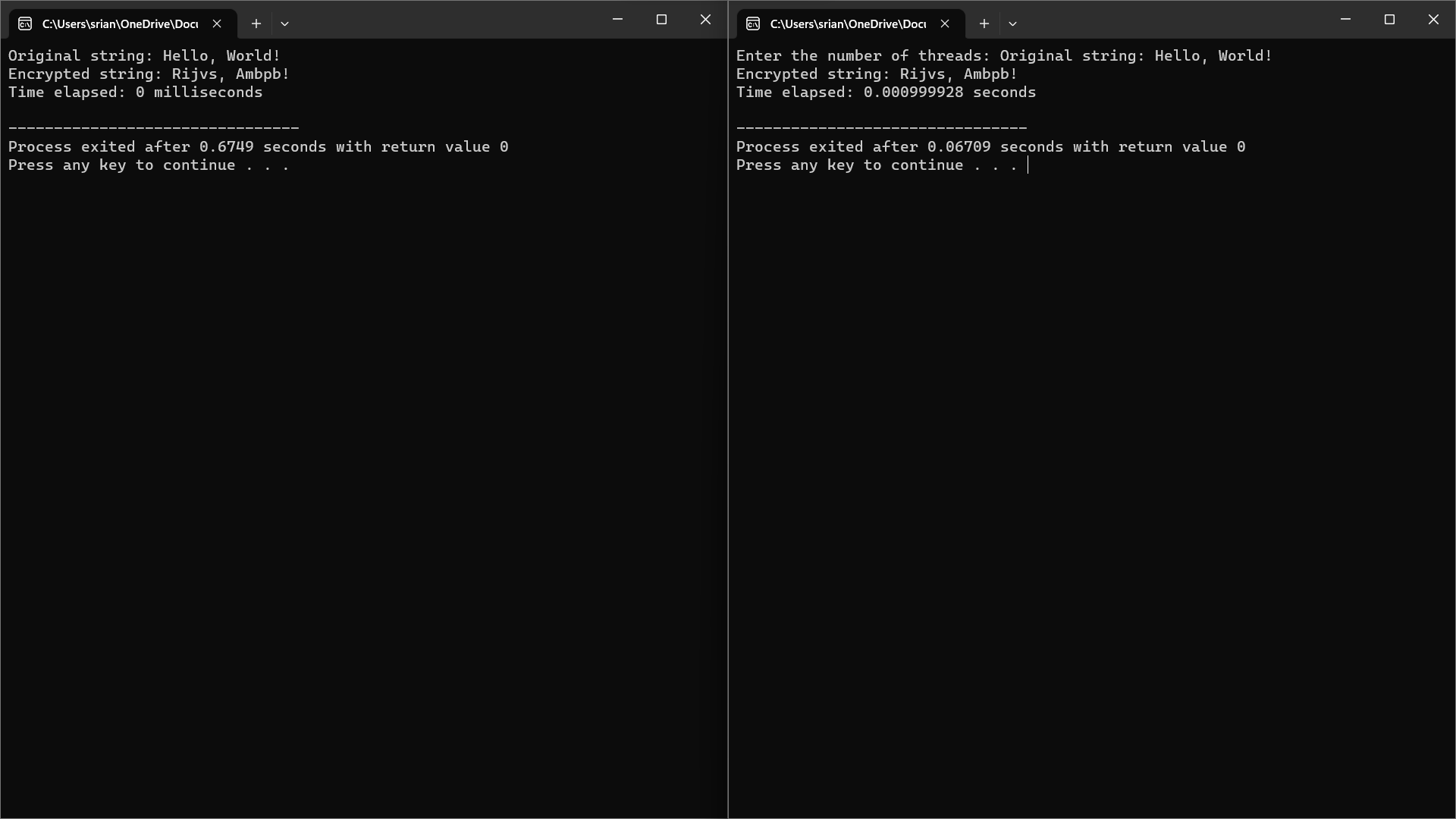
**std::cout << "Time elapsed: " << elapsed\_time << " milliseconds" << std::endl;**

**return 0;**

**}**

**OUTPUT:**

**Serial program output parallel programming output**

****

As you can see the time taken for parallel processing is nearly 10 times lesser then in serial processing in our code to perform Vergne’s algorithm.