Distributed Computing Final Report

Program:

Course Code: CSE336

Course Name: Distributed Computing

Examination Committee

Dr. Safwat Hamad

Ain Shams University
Faculty of Engineering
International Credit Hours Engineering Programs (I-CHEP)
Spring Semester – 2020

Student Personal Information for Group Work

Student Names:	Student Codes:
Engy Samy Salah	16p3004
Mario Medhat	16p3017
Mayar Wessam Hassan	16p3008

Plagiarism Statement

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student Name: **Engy Samy**
Mayar Wessam
Mario Medhat

Date: **31/5/2020**

Submission Contents

01 Introduction

First topic

Distributed computing is a concept in which multi-computers share the same operating device components to increase the output, productivity, performance, and efficiency. The distributed computing is restricted to systems that, according to the meanings, shared components exchanged between computers within a specific geographic. Broader concepts require raising device operations, apps, and features of the system. It is a concept of computing which, in its broadest sense, refers to multiple computer systems working on a single problem. In distributed computation, a single problem is broken into several parts, separate computers solve each component. They will connect with each other if the computers are networked to solve the issue. The machines behave as a single body when performed properly. The primary aim of distributed computing is to optimize efficiency by cost-effectively, transparently, and efficiently linking users and Their resources. It also maintains fault tolerance and allows usability of resources in the event of failure of one of the components. The distributed computing infrastructure consists of a variety of client computers that have very lightweight virtual agents built on one or more dedicated distributed computing, control servers. The agents working on the client machines normally sense when the computer is idle and give the management system a message that the computer is not in operation and is ready for processing work. Distributed computing has two types which is grid computing and cloud computing. Grid computing consists of autonomous multiple computing structure that function as a grid as they consist of non-administrative resource nodes. The definition of cloud computing type is a paradigm change in computing, where computation is shifted away from personal computers or an actual applications server to data cloud of computers.

02 The Difference between Shared and Distributed Address Space

Second topic

Along time ago there were real cache-coherent multiprocessor network with mutual memory. The devices connected with each other via common bus and mutual primary memory. This implied that every link to main memory from either device should be of equivalent latency such kinds of structures are not manufactured anymore. There are also various point to point relations between processing elements and memory elements (this is the explanation for non-uniform memory access, or NUMA). However, the concept of the memory direct contact remains a valuable construct in programming. And in certain applications this is done by hardware and no specific instructions need to be implemented by the programmer. Open MP and Pthreads are the most two common programming strategies that use such abstractions.

Distributed memory has historically been synonymous with processors performing local memory computation and only use specific messages to share data with remote processors once. This adds to the programmer's workload, however, simplifies hardware execution, because the machine no longer has to sustain the impression that all information is currently shared. Traditionally, this kind of programming has been used for supercomputers which have hundreds or thousands of computing elements. MPI is one commonly used technique. There is usually a processor, a memory, and some sort of interconnection inside a distributed memory network that enables programs to communicate with each other on each device. In distributed shared memory each cluster node has access to a broad shred memory as well as a small non-shared private memory for each node.

In our problem we will use shared memory as we are going to use open MP and threads for parallelism.

03

Third topic

Applications in distributed computing:

Distributed computing has a lot of categories of application in our real life. It is used in science, life sciences, cryptography, internet, financial, mathematics, language, Art, puzzle, games, miscellaneous, distributed human project, collaborative knowledge bases, charity. So, in other situations, they are found in mobile networks such a telephone network so cable network, electronic network, digital sensor networks, and routing algorithms. It is also used in network applications such as global web and peer-to - peer networks, distributed caches such as burst buffers, distributed databases and distributed database management systems, Online gaming and virtual reality societies, network file systems, distributed information retrieval systems such as banking systems and airline reservation systems are massively multiplayer. Often used in real-time process control systems such as aircraft control systems, industrial control systems. Rendering distributed in computer graphics has also been used in scientific computing in parallel, including cluster computing , grid computing, cloud computing and numerous collaborative computing projects.

04 Specific Application

Fourth topic

We are going to use shared address as we will use the open MP program to represent multithreads and parallelism. Distributed computing is used bioinformatics in a lot of applications. Most computational power is needed in bioinformatics to perform many small to medium tasks in size, such as creating functional annotation for a complete genome, or to perform one to several large tasks such as molecular representation. Estimating the suitability of the need for distributed computing often relies on two factors: how easily can be split into smaller subtasks, and how much data must be transmitted before or after transmission. Key features for a full distributed computing system are: the ability to provide an summary of available resources such as machines and software along with their status, an interface for task management and results retrieval, careful handling of specific cases such as single machine failures, simple incorporation with existing applications such as proven sequence analysis tool. I n bioinformatics the easiest and most widely utilized method of distributed computing is basic batch processing and load balancing. This process entails shell commands being submitted to a central server, which then determines when and when to execute them depending on variables such as availability and server load. The person who sent it will be alerted via email when a task is through.

05

Fifth topic

Literature review

This project is all about how to make use of the distributed computing in translating the information encoded in the genetic material of the DNA sequences into protein and amino acids. Most of the living organisms have similar structure and their DNA sequences can be translated to nearly the same groups of amino acids, these groups of amino acids are all grouped in a table that have 64 entries.

A single amino acid in the protein chain is consisted of three nucleotide codon. Each combination of three codons has a specific name of amino acid, all stored in a table or a scheme.

Below is the scheme that shows the translation of each three amino acids(codon) into a specific protein. Here the 1st letters abbreviation is used.

Scheme= {

'ATA': 'I', 'ATC': 'I', 'ATT': 'I', 'ATG': 'M' , 'ACA': 'T' , 'ACC': 'T' , 'ACG': 'T' , 'ACT': 'T', 'AAC': 'N', 'AAT': 'N', 'AAA': 'K', 'AAG': 'K' , 'AGC': 'S', 'AGT': 'S', 'AGA': 'R', 'AGG': 'R', 'CTA': 'L', 'CTC': 'L', 'CTG': 'L', 'CTT': 'L', 'CCA': 'P', 'CCC': 'P', 'CCG': 'P', 'CCT': 'P', 'CAC': 'H', 'CAT': 'H', 'CAA': 'Q', 'CAG': 'Q', 'CGA': 'R', 'CGC': 'R', 'CGG': 'R', 'CGT': 'R', 'GTA': 'V', 'GTC': 'V', 'GTG': 'V', 'GTT': 'V', 'GCA': 'A', 'GCC': 'A', 'GCG': 'A', 'GCT': 'A', 'GAC': 'D', 'GAT': 'D', 'GAA': 'E', 'GAG': 'E', 'GGA': 'G', 'GGC': 'G', 'GGG': 'G', 'GGT': 'G', 'TCA': 'S', 'TCC': 'S', 'TCG': 'S', 'TCT': 'S', 'TTC': 'F', 'TTT': 'F', 'TTA': 'L', 'TTG': 'L', 'TAC': 'Y', 'TAT': 'Y', 'TAA': '_', 'TAG': '_', 'TGC': 'C', 'TGT': 'C', 'TGA': '_', 'TGG': 'W', }

In this project we take a Human DNA from overlapping chromosome 19-specific cosmids R29515 and R28253 as in input and translate this DNA to amino acids. Also calculates the frequency of each codon.

06

Sixth topic

Sequential Part

Pseudo Code

Its complexity: O (size of dna/3)

Serial code ()

Initialize protein and amino and codo and s to empty string Initialize count and en and ex and percent to int zero Input dna and codon and trancodon Set en to (size of dna - size of dna%3)-1 For i=0 to end step 3 For j=I to i+3 Set s to dna[j] Append s to codo For x=0 to 64 If codo is equal to codon[x] then Set amino to trancodon[x] Append amino to protein Set codo to empty string Print protein	Initialize array of integers counter For k=0 to 27 Set counter[k] to zero Initialize lengthP to length of protein For i=0 to lengthP If protein[i] equal to I then Counter[0]=couter[0]+1 If protein[i] equal to M then Counter[1]=counter[1]+1 **Continue the if conditions for all possible protein letters and increment the corresponding counter array** For i=0 to index set percent to counter[i] divided by length *100 print counter[i] and percent

07 Parallel Algorithms and Their Complexities

Seventh topic

Parallel code:

Its complexity: O (n/p)

Parallel code ()

```
Initialize protein and amino and codo and
s to empty string
Initialize count and end and ex and
percent to zero
Input dna and codon and trancodon
Set end to (size of dna - size of dna%3 )-1

#pragma omp parallel for schedule
(dynamic) firstprivate (s, dna, codo,
amino) shared(en, protein)
For i=0 to end step 3
  #pragma omp parallel for
      For j=I to i+3
          Set s to dna[j]
          Append s to codo
  #pragma omp parallel for
      For x=0 to 64
          If codo is equal to codon[x]
          then
          Set amino to trancodon[x]
      #pragma omp critical
          Append amino to protein
  Set codo to empty string
Print protein

Initialize array of integers counter
#pragma omp parallel for
shared(counter,k)
      For k=0 to 21
          Set counter[k] to zero
```

```
Initialize lengthP to length of protein
#pragma omp parallel for
      For i=0 to lengthP
          If protein[i] equal to I
          then
          #pragma omp critical
           Counter[0]=counter[0]+1

          If protein[i] equal to M then
          #pragma omp critical
          Counter[1]=counter[1]+1

     Continue the if conditions for all
possible protein letters and increment
     the corresponding counter array

For i=0 to index
     set percent to counter[i] divided by
length *100
      print counter[i] and percent
```

Section Code:

its complexity : O(n/p)

Code ()

Initialize protein and amino and codo and s to empty string
Initialize count and end and ex and percent to zero
Input dna and codon and trancodon
Set end to (size of dna - size of dna%3)-1
#pragma omp parallel firstprivate(codo,amino,s,en)
#pragma omp sections
#pragma omp section
For i=0 to floor((en / 16)) step 3
For j=I to i+3
Set s to dna[j]
Append s to codo
For x=0 to 64
If codo is equal to codon[x] then
Set amino to trancodon[x]
Append amino to protein
Set codo to empty string
#pragma omp section
For i=floor((en / 16))+1 to floor((2*en / 16)) step 3
For j=I to i+3
Set s to dna[j]
Append s to codo
For x=0 to 64
If codo is equal to codon[x] then
Set amino to trancodon[x]
Append amino to protein
Set codo to empty string
 	
Continue the for loops till reaching the 16th loop and append the amino to the rest of the protein
 	
Print protein
	Initialize array of integers counter
	#pragma omp parallel for shared (counter, k)
	For k=0 to 21
	Set counter[k] to zero
	Initialize lengthP to length of protein
	#pragma omp parallel
	#pragma omp sections
	#pragma omp section
	For i=0 to (lengthP / 16)
	If protein[i] equal to I then
	#pragma omp critical
	Counter[0]=counter[0]+1
	If protein[i] equal to M then
	#pragma omp critical
	Counter[1]=counter[1]+1
	Continue the if conditions for all possible protein letters and increment the corresponding counter array.
	Then make another 15 sections another 15 for loops to count the codons in each of the 16 proteins defined above
	#pragma omp section
	For i= (lengthP / 16) to =(2*lengthP / 16)
	.
	.
	#pragma omp section
	For i= (15*lengthP / 16) to lengthP
	
	For i=0 to index
	set percent to counter[i] divided by length *100
	print counter[i] and percent

08

Eighth topic

Performance Evaluation

Parallel For:

Number of threads 2:

Speedup: (0.844682/0.651658)=1.2962

Efficiency :S/2=1.2962/2=0.6481

Iso-efficiency:

E/((1-E) P^(E/(1-E))) *log(P) where p is number of threads=0.1547

Number of threads 4:

Speedup: (0.844682/0.495283)=1.7

Efficiency :S/4=1.7/4=0.426

Iso-efficiency:

E/((1-E) P^(E/(1-E))) *log(P) where p is number of threads=0.1596

Number of threads 8:

Speedup: (0.844682/0.449801)=1.8779

Efficiency :S/8=1.8779/8=0.23473

Iso-efficiency:

E/((1-E) P^(E/(1-E))) *log(P) where p is number of threads=0.14637996

Number of threads 16:

Speedup: (0.844682/0.383329)=2.2

Efficiency :S/2=2.2/16=0.1377

Iso-efficiency:

E/((1-E) P^(E/(1-E))) *log(P) where p is number of threads=0.12349

Function Sections:

Number of threads 2:

Speedup: (0.844682/0.589559)=1.4327

Efficiency :S/2=0.71636

Iso-efficiency:

E/((1-E) P^(E/(1-E))) *log(P) where p is number of threads=0.132

Number of threads 4:

Speedup: (0.844682/0.469172)=1.8

Efficiency :S/4=0.45

Iso-efficiency:

E/((1-E) P^(E/(1-E))) *log(P) where p is number of threads=0.1584

Number of threads 8:

Speedup: (0.844682/0.405438)=2.08338

Efficiency :S/8=0.26

Iso-efficiency:

E/((1-E) P^(E/(1-E))) *log(P) where p is number of threads=0.1528

Number of threads 16:

Speedup: (0.844682/0.384509)=2.1967

Efficiency :S/2=0.13729

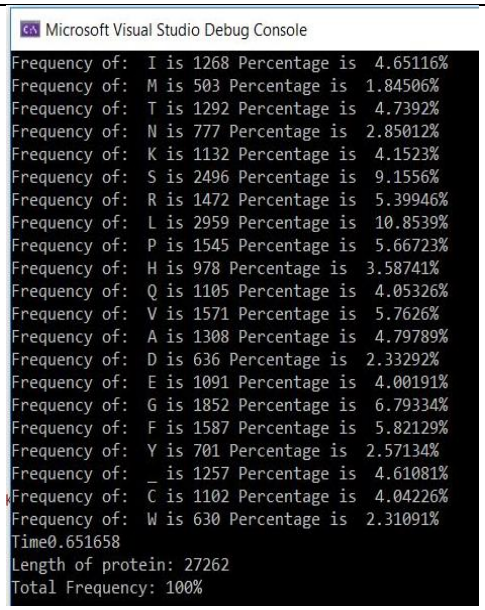
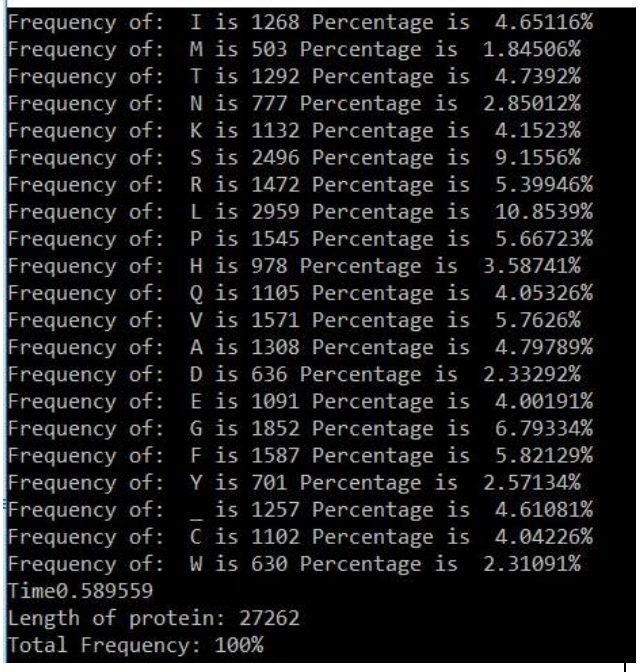
Iso-efficiency:

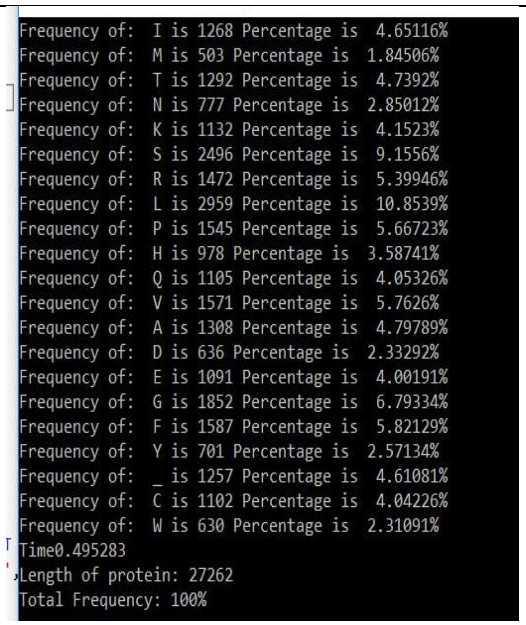
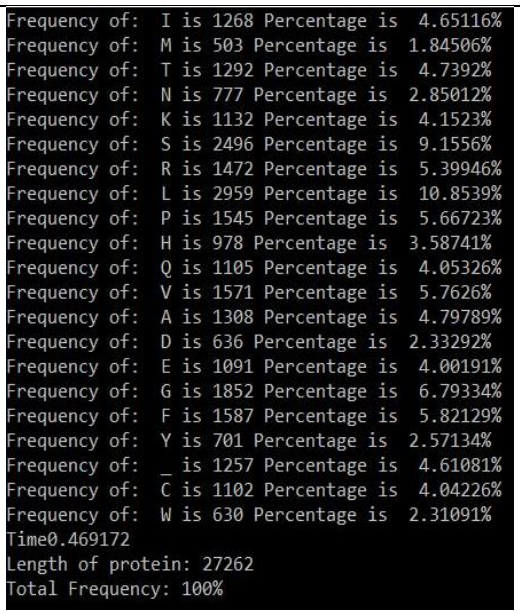
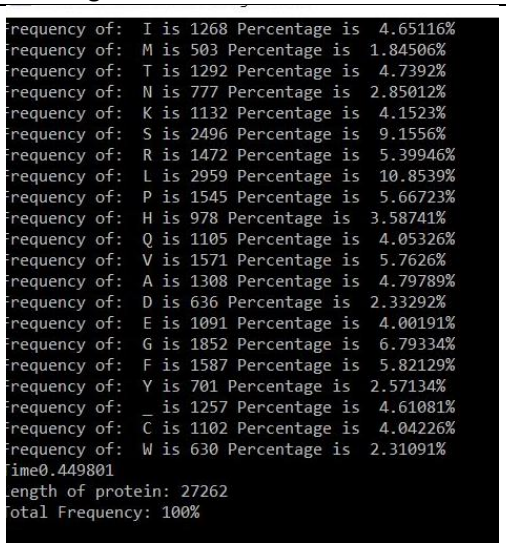
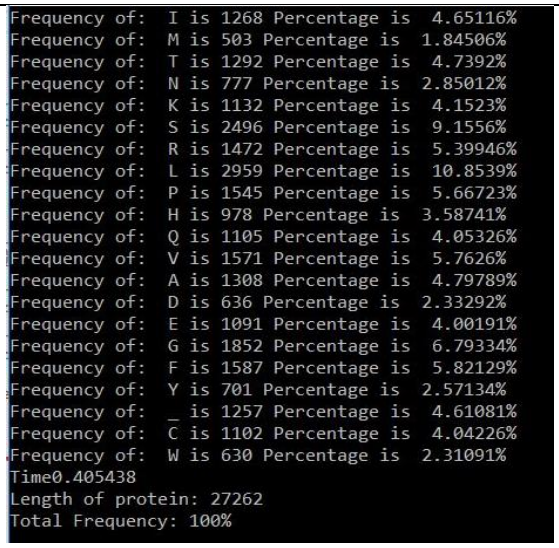
E/((1-E) P^(E/(1-E))) *log(P) where p is number of threads=0.12326

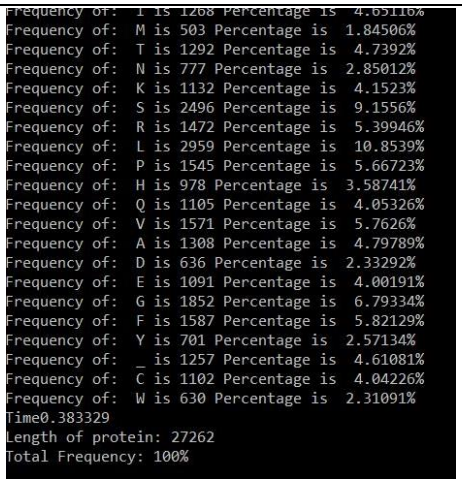
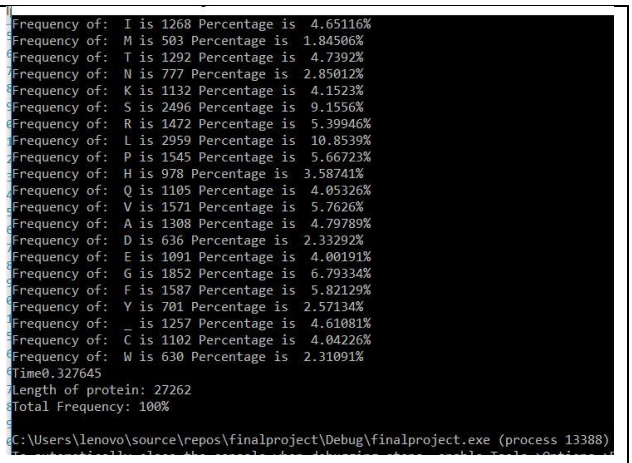
09

Ninth topic

Used H/W Specs, results and discussions

N.T	Parallel For	Parallel Section
2	 The time using two thread in parallel for is 0.651658 which is less than serial but greater with low percent comparing with parallel sections in which for loop is divided among 2 threads only	 The time using two thread in parallel sections is 0.589559 which is less than serial and less than parallel for with 2 threads as each thread take a sections and after they finish they take anther section

4	 The time of parallel for with 4 threads is 0.495283 which is much less than 2 threads as the for loop is divides among 4 threads	 The time of parallel sections with 4 threads is 0.469172 which is much less than 2 and less than parallel for with 4 threads
8	 The time of parallel for with 8 threads is 0.4498 which is less but near not parallel for with 4 threads	 The time in parallel sections with 8 is 0.405438 which less than parallel for w determine that the parallel sections is better in time than parallel for.

16	 This is the best time appear in parallel for which is 0.3833 as the for	 This is the best time appear in the parallel sections (0.32) which is the best time among all of the previous ones.

10

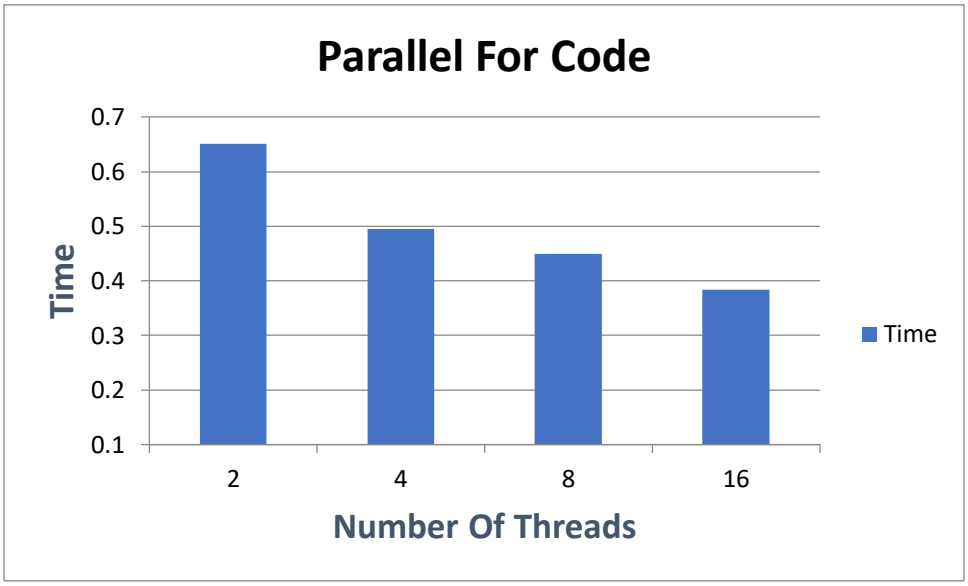
Tenth topic

Graph for each parallel solution

Graph for comparing the time resulting from different number of threads:

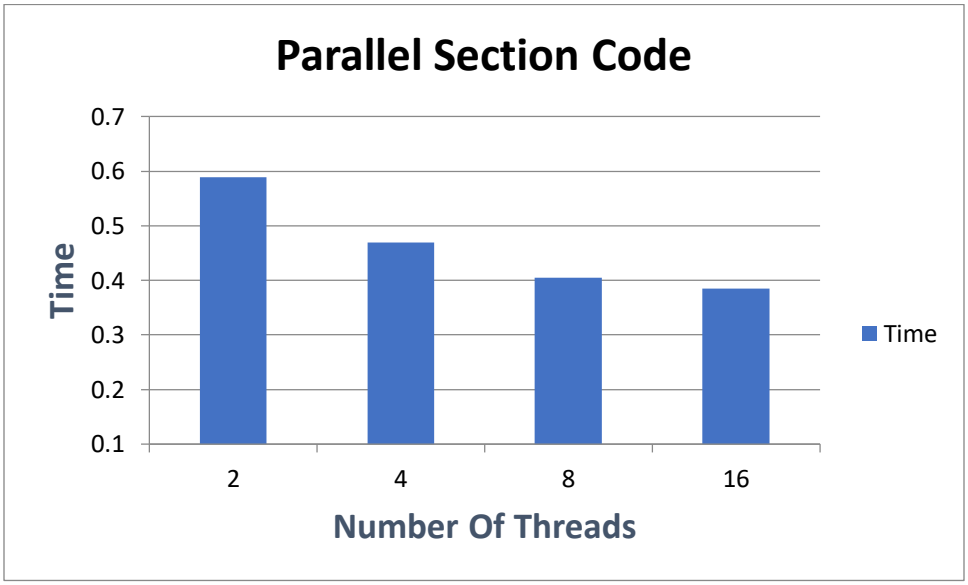
- **Parallel For:**

Number Of Threads (X-axis)	Time (Y-axis)
2	0.6516
4	0.4952
8	0.4498
16	0.3833



- **Sections:**

Number Of Threads (X-axis)	Time (Y-axis)
2	0.589559
4	0.469172
8	0.405438
16	0.384509



11

Eleventh topic

Conclusion

Finally, we have observed and concluded various things concerning how to use distributed computing on our project (Translating the DNA sequence). We used a certain scheme to follow in translating the DNA into their corresponding codons amino acids. This can be done either by making the code run sequentially one line after the other, or make its for loops run parallel ,or divide the code into sections.

We did implement the three ways and we observed the results. They all of course have the same translated protein and the same frequencies of each codon, but they have different time in doing so.

The sequential code has larger time than that of the parallel for and the section ones. This is because the parallel ones use threads so the work is divided among them. But the parallel sections have less time than the parallel for code, yet the difference is not that big.

In our opinion using the parallel ones is more efficient than using a sequential code as it saves so much time using a number of threads.

12 Twelfth

Appendix

```
#include <iostream>
#include <string>
#include <fstream>
#include <algorithm>
#include <string.h>
#include <omp.h>
#include <cmath>
#define N 16
using namespace std;

string codon[] = {
        "ata", "atc", "att", "atg",
        "aca", "acc", "acg", "act",
        "aac", "aat", "aaa", "aag",
        "agc", "agt", "aga", "agg",
        "cta", "ctc", "ctg", "ctt",
        "cca", "ccc", "ccg", "cct",
        "cac", "cat", "caa", "cag",
        "cga", "cgc", "cgg", "cgt",
        "gta", "gtc", "gtg", "gtt",
        "gca", "gcc", "gcg", "gct",
        "gac", "gat", "gaa", "gag",
        "gga", "ggc", "ggg", "ggt",
        "tca", "tcc", "tcg", "tct",
        "ttc", "ttt", "tta", "ttg",
        "tac", "tat", "taa", "tag",
        "tgc", "tgt", "tga", "tgg" };

char trancodon[] = {
'I','I','I','M','T','T','T','T','N','N','K','K','S','S','R'
,'R','L','L','L','L','P','P','P','P','H','H','Q','Q','R','
R','R','R','V','V','V','V','A','A','A','A','D','D',
'E','E','G','G','G','G','S','S','S','S','F','F','L','L','
Y','Y','_','_','_','C','C','_','W' };

string dna = "";
```

```
void read_seq()
{
    fstream newfile;

    char tpp[3000];
    newfile.open("InputSeq.txt", ios::in);
    if (newfile.is_open())
    {
        string tp = " ";
        while (getline(newfile, tp))
        {

            tp.erase(remove(tp.begin(), tp.end(), ' '), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '1'), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '2'), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '3'), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '4'), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '5'), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '6'), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '7'), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '8'), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '9'), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '0'), tp.end());
            tp.erase(remove(tp.begin(), tp.end(), '/'), tp.end());

            dna.append(tp);
        }
        newfile.close();
    }
    cout << dna << "\n";

}
```

```
void serialcode()
{
   string protein = "";
   string codo = "";
   string s = "";
   string amino = "";
   char I = 'I', M = 'M', T = 'T', n = 'N', K = 'K', S =
'S', R = 'R', L = 'L', P = 'P', H = 'H', Q = 'Q', V = 'V',
A = 'A', D = 'D', E = 'E', G = 'G', F = 'F', Y = 'Y',
stop = '_', C = 'C', W = 'W';
char letters[21] = { 'I', 'M','T','N', 'K','S', 'R', 'L', 'P',
'H', 'Q', 'V', 'A', 'D','E','G', 'F','Y', '_','C', 'W' };
   int count = 0,ex=0,en=0,id=0;
   int i, j, x, k, lengthP, doneboo = 0, index = 0;
   float percent = 0.0, sum = 0.0;
   double start, end;
   en = ((dna.size()) - (dna.size()) % 3) - 1;

   omp_set_num_threads(N);
   start = omp_get_wtime();

   for (int i = 0; i <= en; i += 3)
   {
      for (int j = i; j < (i + 3); j++)
      {
         s = dna[j];
         codo.append(s);
      }
      for (int x = 0; x < 64; x++)
      {
         if (codo == codon[x])
         {
            amino = trancodon[x];
            protein.append(amino);
         }
      }
      codo = "";
   }

   cout << protein << "\n";
   cout << protein.length() << "\n";
   int counter[21];
   lengthP = protein.length();

   for (k = 0; k < 21; k++)
   {
      counter[k] = 0;
   }
   for (int i = 0; i < lengthP; i++)
   {
      if (protein[i] == I)
      {
         counter[0] = counter[0] + 1;
      }
      if (protein[i] == M)
      {
         counter[1] = counter[1] + 1;
      }
      if (protein[i] == T)
      {
         counter[2] = counter[2] + 1;
      }
      if (protein[i] == n)
      {
         counter[3] = counter[3] + 1;
      }
      if (protein[i] == K)
      {
         counter[4] = counter[4] + 1;
      }
      if (protein[i] == S)
      {
         counter[5] = counter[5] + 1;
      }
      if (protein[i] == R)
      {
         counter[6] = counter[6] + 1;
      }
      if (protein[i] == L)
      {
         counter[7] = counter[7] + 1;
      }
      if (protein[i] == P)
      {
         counter[8] = counter[8] + 1;
      }
      if (protein[i] == H)
      {
         counter[9] = counter[9] + 1;
      }
      if (protein[i] == Q)
      {
         counter[10] = counter[10] + 1;
      }
      if (protein[i] == V)
      {
         counter[11] = counter[11] + 1;
      }
```

```
    if (protein[i] == A)
    {
       counter[12] = counter[12] + 1;
    }
    if (protein[i] == D)
    {
       counter[13] = counter[13] + 1;
    }
    if (protein[i] == E)
    {
       counter[14] = counter[14] + 1;
    }
    if (protein[i] == G)
    {
       counter[15] = counter[15] + 1;
    }
    if (protein[i] == F)
    {
       counter[16] = counter[16] + 1;
    }
    if (protein[i] == Y)
    {
       counter[17] = counter[17] + 1;
    }
    if (protein[i] == stop)
    {
       counter[18] = counter[18] + 1;
    }
    if (protein[i] == C)
    {
       counter[19] = counter[19] + 1;
    }
    if (protein[i] == W)
    {
       counter[20] = counter[20] + 1;
    }
}
```

```
end = omp_get_wtime();
   for (int i = 0; i < 21; i++)
   {
      //percent=0.0;
      percent = (float)(((float)counter[i] /
(float)lengthP) * 100);
      sum = sum + percent;
      cout << "Frequency of:  " << letters[i] << " is
" << counter[i] << " Percentage is  " << percent <<
"%" << "\n";
   }

   cout << "Time" << end - start << "\n";
   cout << "Length of protein: " << lengthP <<
"\n";
   cout << "Total Frequency: " << sum << "%" <<
"\n";
}
```

```
void paralelcode()
{
   string protein = "",codo="", s = "", amino = "";
   char I = 'I', M = 'M', T = 'T', n = 'N', K = 'K', S =
'S', R = 'R', L = 'L', P = 'P', H = 'H', Q = 'Q', V = 'V',
A = 'A', D = 'D', E = 'E', G = 'G', F = 'F', Y = 'Y',
stop = '_', C = 'C', W = 'W';
char letters[21] = { 'I', 'M','T','N', 'K','S', 'R', 'L', 'P',
'H', 'Q', 'V', 'A', 'D','E','G', 'F','Y', '_','C', 'W' };
   int count = 0,ex=0,en=0,id=0;
   int i, j, x,k,lengthP,doneboo=0,index=0;
   float percent = 0.0,sum=0.0;
   double start, end;

  en = ((dna.size()) - (dna.size()) % 3) - 1;
  omp_set_num_threads(N);
  start = omp_get_wtime();

#pragma omp parallel for schedule(dynamic)
firstprivate(s,dna,codo,amino) shared(en,protein)
   for (int i = 0; i <= en; i += 3)
   {
#pragma omp parallel for
      for (int j = i; j < (i + 3); j++)
      {
         s = dna[j];
         codo.append(s);
      }
      #pragma omp parallel for
      for (int x = 0; x < 64; x++)
      {
         if (codo == codon[x])
         {
            amino = trancodon[x];
#pragma omp critical
            protein.append(amino);
         }
      }
      codo = "";
   }
   cout << protein << "\n";
   cout << protein.length() << "\n";

   int counter[21];
   lengthP = protein.length();
#pragma omp parallel for shared(counter,k)
   for (k = 0; k < 21; k++)
   {
      counter[k] = 0;
   }
```

```
#pragma omp parallel for
   for (int i = 0; i < lengthP; i++)
   {
      if (protein[i] == I)
      {
#pragma omp critical
         counter[0]=counter[0]+1;
      }
      if (protein[i] == M)
      {
#pragma omp critical
         counter[1] = counter[1] + 1;
      }
      if (protein[i] == T)
      {
#pragma omp critical
         counter[2] = counter[2] + 1;
      }
      if (protein[i] == n)
      {
#pragma omp critical
         counter[3] = counter[3] + 1;
      }
      if (protein[i] == K)
      {
#pragma omp critical
         counter[4] = counter[4] + 1;
      }
      if (protein[i] == S)
      {
#pragma omp critical
         counter[5] = counter[5] + 1;
      }
      if (protein[i] == R)
      {
#pragma omp critical
         counter[6] = counter[6] + 1;
      }
      if (protein[i] == L)
      {
#pragma omp critical
         counter[7] = counter[7] + 1;
      }
      if (protein[i] == P)
      {
#pragma omp critical
         counter[8] = counter[8] + 1;
      }
```

```
if (protein[i] == H)
      {
#pragma omp critical
         counter[9] = counter[9] + 1;
      }
      if (protein[i] == Q)
      {
#pragma omp critical
         counter[10] = counter[10] + 1;
      }
      if (protein[i] == V)
      {
#pragma omp critical
         counter[11] = counter[11] + 1;
      }
      if (protein[i] == A)
      {
#pragma omp critical
         counter[12] = counter[12] + 1;
      }
      if (protein[i] == D)
      {
#pragma omp critical
         counter[13] = counter[13] + 1;
      }
      if (protein[i] == E)
      {
#pragma omp critical
         counter[14] = counter[14] + 1;
      }
      if (protein[i] == G)
      {
#pragma omp critical
         counter[15] = counter[15] + 1;
      }
      if (protein[i] == F)
      {
#pragma omp critical
         counter[16] = counter[16] + 1;
      }
      if (protein[i] == Y)
      {
#pragma omp critical
         counter[17] = counter[17] + 1;
      }
```

```
    if (protein[i] == stop)
      {
#pragma omp critical
         counter[18] = counter[18] + 1;
      }
      if (protein[i] == C)
      {
#pragma omp critical
         counter[19] = counter[19] + 1;
      }
      if (protein[i] == W)
      {
#pragma omp critical
         counter[20] = counter[20] + 1;
      }
   }

   end = omp_get_wtime();

   for (int i = 0; i < 21; i++)
   {
      percent = (float)(((float)counter[i] /
                  (float)lengthP) * 100);
      sum = sum + percent;
      cout << "Frequency of:  " << letters[i] << " is
" << counter[i] << " Percentage is  " << percent <<
"%" << "\n";
   }

  cout << "Time" << end - start << "\n";
  cout << "Length of protein: " << lengthP << "\n";
  cout << "Total Frequency: " << sum << "%" <<
"\n";

}
```

```
void prallelSectionCode()
{
   string protein1 = "", protein2 = "", protein3 = "",
protein4 = "", protein5 = "", protein6 = "", protein7
= "", protein8 = "", protein9 = "", protein10 = "",
protein11 = "", protein12 = "", protein13 = "",
protein14 = "", protein15 = "", protein16 = "";
   string codo = "", s = "", amino = "";
   char I = 'I', M = 'M', T = 'T', n = 'N', K = 'K', S =
'S', R = 'R', L = 'L', P = 'P', H = 'H', Q = 'Q', V = 'V',
A = 'A', D = 'D', E = 'E', G = 'G', F = 'F', Y = 'Y',
stop = '_', C = 'C', W = 'W';
   char letters[21] = { 'I', 'M','T','N', 'K','S', 'R', 'L',
'P', 'H', 'Q', 'V', 'A', 'D','E','G', 'F','Y', '_','C', 'W' };
   int count = 0, ex=0, en=0, id=0;
   int i, j, x, k, lengthP, doneboo = 0, index = 0;
   float percent = 0.0, sum = 0.0;
   double start, end;
   en = ((dna.size()) - (dna.size()) % 3) - 1;

   omp_set_num_threads(N);
   start = omp_get_wtime();
#pragma omp parallel firstprivate(codo,amino,s,en)
   {
#pragma omp sections
      {
#pragma omp section
         {
            for (int i = 0; i < floor((en / 16)); i += 3)
            {
               for (int j = i; j < (i + 3); j++)
               {

                  s = dna[j];
                  codo.append(s);
               }

               for (int x = 0; x < 64; x++)
               {
                  if (codo == codon[x])
                  {
                     amino = trancodon[x];
                     protein1.append(amino);
                  }
               }
               codo = "";
            }
         }
#pragma omp section
         {
            for (int i= floor((en / 16))+1;i<
floor(((2*en) / 16));  i += 3)
            {
               for (int j = i; j < (i + 3); j++)
               {

                  s = dna[j];
                  codo.append(s);
               }

               for (int x = 0; x < 64; x++)
               {
                  if (codo == codon[x])
                  {
                     amino = trancodon[x];
                     protein2.append(amino);
                  }
               }
               codo = "";
            }
         }
#pragma omp section
         {
            for (int i = floor(((2*en) / 16))+1; i <
floor(((3 * en) / 16)); i += 3)
            {
               for (int j = i; j < (i + 3); j++)
               {

                  s = dna[j];
                  codo.append(s);
               }

               for (int x = 0; x < 64; x++)
               {
                  if (codo == codon[x])
                  {
                     amino = trancodon[x];
                     protein3.append(amino);
                  }
               }
               codo = "";
            }
         }
```

```
#pragma omp section
        {
           for (int i = floor(((3 * en) / 16))+2; i <
floor(((4 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein4.append(amino);
                 }
              }
              codo = "";
           }
        }
#pragma omp section
        {
           for (int i = floor(((4 * en) / 16))+2; i <
floor(((5 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein5.append(amino);
                 }
              }
              codo = "";
           }
        }
```

```
#pragma omp section
        {
           for (int i = floor(((5* en) / 16)); i <
floor(((6 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein6.append(amino);
                 }
              }
              codo = "";
           }
        }
#pragma omp section
        {
           for (int i = floor(((6 * en) / 16)); i <
floor(((7 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein7.append(amino);
                 }
              }
              codo = "";
           }
        }
```

```
#pragma omp section
        {
           for (int i = floor(((7 * en) / 16))+1; i <
floor(((8 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein8.append(amino);
                 }
              }
              codo = "";
           }
        }
#pragma omp section
        {
           for (int i = floor(((8 * en) / 16))+1; i <
floor(((9 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein9.append(amino);
                 }
              }
              codo = "";
           }
        }
```

```
#pragma omp section
        {
           for (int i = floor(((9 * en) / 16))+1; i <
floor(((10 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein10.append(amino);
                 }
              }
              codo = "";
           }
        }
#pragma omp section
        {
           for (int i = floor(((10 * en) / 16))+2; i <
floor(((11 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein11.append(amino);
                 }
              }
              codo = "";
           }
        }
```

```
#pragma omp section
        {
           for (int i = floor(((11 * en) / 16))+2; i <
floor(((12 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein12.append(amino);
                 }
              }
              codo = "";
           }
        }
#pragma omp section
        {
           for (int i = floor(((12 * en) / 16)); i <
floor(((13 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein13.append(amino);
                 }
              }
              codo = "";
           }
        }
```

```
#pragma omp section
        {
           for (int i = floor(((13 * en) / 16)); i <
floor(((14 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein14.append(amino);
                 }
              }
              codo = "";
           }
        }
#pragma omp section
        {
           for (int i = floor(((14 * en) / 16))+1; i <
floor(((15 * en) / 16)); i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {

                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein15.append(amino);
                 }
              }
              codo = "";
           }
        }
```

```
#pragma omp section
        {
           for (int i = floor(((15 * en) / 16))+1; i < (16 * en) / 16; i += 3)
           {
              for (int j = i; j < (i + 3); j++)
              {
                 s = dna[j];
                 codo.append(s);
              }

              for (int x = 0; x < 64; x++)
              {
                 if (codo == codon[x])
                 {
                    amino = trancodon[x];
                    protein16.append(amino);
                 }
              }
              codo = "";
           }
        }
     }
  }

  protein1.append(protein2);
  protein1.append(protein3);
  protein1.append(protein4);
  protein1.append(protein5);
  protein1.append(protein6);
  protein1.append(protein7);
  protein1.append(protein8);
  protein1.append(protein9);
  protein1.append(protein10);
  protein1.append(protein11);
  protein1.append(protein12);
  protein1.append(protein13);
  protein1.append(protein14);
  protein1.append(protein15);
  protein1.append(protein16);

  cout << protein16 << "\n";
  cout << protein1.length() << "\n";

  int counter[21];
  lengthP = protein1.length();
  for (k = 0; k < 21; k++)
  {
     counter[k] = 0;
  }
```

```
#pragma omp parallel
   {
#pragma omp sections
      {
#pragma omp section
         {
            for (int i = 0; i < floor(lengthP / 16); i++)
            {
               if (protein1[i] == I)
               {
#pragma omp critical
                  counter[0] = counter[0] + 1;
               }
               if (protein1[i] == M)
               {
#pragma omp critical
                  counter[1] = counter[1] + 1;
               }
               if (protein1[i] == T)
               {
#pragma omp critical
                  counter[2] = counter[2] + 1;
               }
               if (protein1[i] == n)
               {
#pragma omp critical
                  counter[3] = counter[3] + 1;
               }
               if (protein1[i] == K)
               {
#pragma omp critical
                  counter[4] = counter[4] + 1;
               }
               if (protein1[i] == S)
               {
#pragma omp critical
                  counter[5] = counter[5] + 1;
               }
               if (protein1[i] == R)
               {
#pragma omp critical
                  counter[6] = counter[6] + 1;
               }
               if (protein1[i] == L)
               {
#pragma omp critical
                  counter[7] = counter[7] + 1;
               }
```

```
            if (protein1[i] == P)
               {
#pragma omp critical
                  counter[8] = counter[8] + 1;
               }
               if (protein1[i] == H)
               {
#pragma omp critical
                  counter[9] = counter[9] + 1;
               }
               if (protein1[i] == Q)
               {
#pragma omp critical
                  counter[10] = counter[10] + 1;
               }
               if (protein1[i] == V)
               {
#pragma omp critical
                  counter[11] = counter[11] + 1;
               }
               if (protein1[i] == A)
               {
#pragma omp critical
                  counter[12] = counter[12] + 1;
               }
               if (protein1[i] == D)
               {
#pragma omp critical
                  counter[13] = counter[13] + 1;
               }
               if (protein1[i] == E)
               {
#pragma omp critical
                  counter[14] = counter[14] + 1;
               }
               if (protein1[i] == G)
               {
#pragma omp critical
                  counter[15] = counter[15] + 1;
               }
               if (protein1[i] == F)
               {
#pragma omp critical
                  counter[16] = counter[16] + 1;
               }
```

```
            if (protein1[i] == Y)
            {
#pragma omp critical
                counter[17] = counter[17] + 1;
            }

            if (protein1[i] == stop)
            {
#pragma omp critical
                counter[18] = counter[18] + 1;
            }
            if (protein1[i] == C)
            {
#pragma omp critical
                counter[19] = counter[19] + 1;
            }
            if (protein1[i] == W)
            {
#pragma omp critical
                counter[20] = counter[20] + 1;
            }
        }
    }

#pragma omp section
    {
        for (int i = floor(lengthP / 16); i < floor((2 *
lengthP) / 16); i++)
        {
            if (protein1[i] == I)
            {
        #pragma omp critical
                counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
        #pragma omp critical
                counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
        #pragma omp critical
                counter[2] = counter[2] + 1;
            }
            if (protein1[i] == n)
            {
        #pragma omp critical
                counter[3] = counter[3] + 1;
            }
            if (protein1[i] == K)
            {
        #pragma omp critical
                counter[4] = counter[4] + 1;
            }
            if (protein1[i] == S)
            {
        #pragma omp critical
                counter[5] = counter[5] + 1;
            }
            if (protein1[i] == R)
            {
        #pragma omp critical
                counter[6] = counter[6] + 1;
            }
            if (protein1[i] == L)
            {
#pragma omp critical
                counter[7] = counter[7] + 1;
            }
            if (protein1[i] == P)
            {
#pragma omp critical
                counter[8] = counter[8] + 1;
            }
            if (protein1[i] == H)
            {
#pragma omp critical
                counter[9] = counter[9] + 1;
            }
            if (protein1[i] == Q)
            {
#pragma omp critical
                counter[10] = counter[10] + 1;
            }
            if (protein1[i] == V)
            {
#pragma omp critical
                counter[11] = counter[11] + 1;
            }
            if (protein1[i] == A)
            {
#pragma omp critical
                counter[12] = counter[12] + 1;
            }
            if (protein1[i] == D)
            {
#pragma omp critical
                counter[13] = counter[13] + 1;
            }
```

```
            if (protein1[i] == E)
            {
#pragma omp critical
                counter[14] = counter[14] + 1;
            }
            if (protein1[i] == G)
            {
#pragma omp critical
                counter[15] = counter[15] + 1;
            }
            if (protein1[i] == F)
            {
#pragma omp critical
                counter[16] = counter[16] + 1;
            }
            if (protein1[i] == Y)
            {
#pragma omp critical
                counter[17] = counter[17] + 1;
            }
            if (protein1[i] == stop)
            {
#pragma omp critical
                counter[18] = counter[18] + 1;
            }
            if (protein1[i] == C)
            {
#pragma omp critical
                counter[19] = counter[19] + 1;
            }
            if (protein1[i] == W)
            {
#pragma omp critical
                counter[20] = counter[20] + 1;
            }
        }
    }
```

```
#pragma omp section
    {
        for (int i = floor((2 * lengthP) / 16); i <
floor((3 * lengthP) / 16); i++)
        {

            if (protein1[i] == I)
            {
#pragma omp critical
                counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
#pragma omp critical
                counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
#pragma omp critical
                counter[2] = counter[2] + 1;
            }
            if (protein1[i] == n)
            {
#pragma omp critical
                counter[3] = counter[3] + 1;
            }
            if (protein1[i] == K)
            {
#pragma omp critical
                counter[4] = counter[4] + 1;
            }
            if (protein1[i] == S)
            {
#pragma omp critical
                counter[5] = counter[5] + 1;
            }
            if (protein1[i] == R)
            {
#pragma omp critical
                counter[6] = counter[6] + 1;
            }
            if (protein1[i] == L)
            {
#pragma omp critical
                counter[7] = counter[7] + 1;
            }
```

```
            if (protein1[i] == P)
            {
#pragma omp critical
                counter[8] = counter[8] + 1;
            }
            if (protein1[i] == H)
            {
#pragma omp critical
                counter[9] = counter[9] + 1;
            }
            if (protein1[i] == Q)
            {
#pragma omp critical
                counter[10] = counter[10] + 1;
            }
            if (protein1[i] == V)
            {
#pragma omp critical
                counter[11] = counter[11] + 1;
            }
            if (protein1[i] == A)
            {
#pragma omp critical
                counter[12] = counter[12] + 1;
            }
            if (protein1[i] == D)
            {
#pragma omp critical
                counter[13] = counter[13] + 1;
            }
            if (protein1[i] == E)
            {
#pragma omp critical
                counter[14] = counter[14] + 1;
            }
            if (protein1[i] == G)
            {
#pragma omp critical
                counter[15] = counter[15] + 1;
            }
            if (protein1[i] == F)
            {
#pragma omp critical
                counter[16] = counter[16] + 1;
            }
```

```
            if (protein1[i] == Y)
            {
#pragma omp critical
                counter[17] = counter[17] + 1;
            }
            if (protein1[i] == stop)
            {
#pragma omp critical
                counter[18] = counter[18] + 1;
            }
            if (protein1[i] == C)
            {
#pragma omp critical
                counter[19] = counter[19] + 1;
            }
            if (protein1[i] == W)
            {
#pragma omp critical
                counter[20] = counter[20] + 1;
            }
        }
    }
```

```
#pragma omp section
        {
           for (int i = ((3 * lengthP) / 16) ; i <
floor((4 * lengthP) / 16); i++)
           {

              if (protein1[i] == I)
              {
#pragma omp critical
                 counter[0] = counter[0] + 1;
              }
              if (protein1[i] == M)
              {
#pragma omp critical
                 counter[1] = counter[1] + 1;
              }
              if (protein1[i] == T)
              {
#pragma omp critical
                 counter[2] = counter[2] + 1;
              }
              if (protein1[i] == n)
              {
#pragma omp critical
                 counter[3] = counter[3] + 1;
              }
              if (protein1[i] == K)
              {
#pragma omp critical
                 counter[4] = counter[4] + 1;
              }
              if (protein1[i] == S)
              {
#pragma omp critical
                 counter[5] = counter[5] + 1;
              }
              if (protein1[i] == R)
              {
#pragma omp critical
                 counter[6] = counter[6] + 1;
              }
              if (protein1[i] == L)
              {
#pragma omp critical
                 counter[7] = counter[7] + 1;
              }
```

```
              if (protein1[i] == P)
              {
#pragma omp critical
                 counter[8] = counter[8] + 1;
              }
              if (protein1[i] == H)
              {
#pragma omp critical
                 counter[9] = counter[9] + 1;
              }
              if (protein1[i] == Q)
              {
#pragma omp critical
                 counter[10] = counter[10] + 1;
              }
              if (protein1[i] == V)
              {
#pragma omp critical
                 counter[11] = counter[11] + 1;
              }
              if (protein1[i] == A)
              {
#pragma omp critical
                 counter[12] = counter[12] + 1;
              }
              if (protein1[i] == D)
              {
#pragma omp critical
                 counter[13] = counter[13] + 1;
              }
              if (protein1[i] == E)
              {
#pragma omp critical
                 counter[14] = counter[14] + 1;
              }
              if (protein1[i] == G)
              {
#pragma omp critical
                 counter[15] = counter[15] + 1;
              }
              if (protein1[i] == F)
              {
#pragma omp critical
                 counter[16] = counter[16] + 1;
              }
              if (protein1[i] == Y)
              {
#pragma omp critical
                 counter[17] = counter[17] + 1;
              }
```

```
                if (protein1[i] == stop)
                {
#pragma omp critical
                    counter[18] = counter[18] + 1;
                }
                if (protein1[i] == C)
                {
#pragma omp critical
                    counter[19] = counter[19] + 1;
                }
                if (protein1[i] == W)
                {
#pragma omp critical
                    counter[20] = counter[20] + 1;
                }
            }
        }


#pragma omp section
        {
            for (int i = ((4 * lengthP) / 16); i <
floor((5 * lengthP) / 16); i++)
            {
                if (protein1[i] == I)
                {
#pragma omp critical
                    counter[0] = counter[0] + 1;
                }
                if (protein1[i] == M)
                {
#pragma omp critical
                    counter[1] = counter[1] + 1;
                }
                if (protein1[i] == T)
                {
#pragma omp critical
                    counter[2] = counter[2] + 1;
                }
                if (protein1[i] == n)
                {
#pragma omp critical
                    counter[3] = counter[3] + 1;
                }
```

```
                if (protein1[i] == K)
                {
#pragma omp critical
                    counter[4] = counter[4] + 1;
                }
                if (protein1[i] == S)
                {
#pragma omp critical
                    counter[5] = counter[5] + 1;
                }
                if (protein1[i] == R)
                {
#pragma omp critical
                    counter[6] = counter[6] + 1;
                }
                if (protein1[i] == L)
                {
#pragma omp critical
                    counter[7] = counter[7] + 1;
                }
                if (protein1[i] == P)
                {
#pragma omp critical
                    counter[8] = counter[8] + 1;
                }
                if (protein1[i] == H)
                {
#pragma omp critical
                    counter[9] = counter[9] + 1;
                }
                if (protein1[i] == Q)
                {
#pragma omp critical
                    counter[10] = counter[10] + 1;
                }
                if (protein1[i] == V)
                {
#pragma omp critical
                    counter[11] = counter[11] + 1;
                }
                if (protein1[i] == A)
                {
#pragma omp critical
                    counter[12] = counter[12] + 1;
                }
```

```
                if (protein1[i] == D)
                {
#pragma omp critical
                    counter[13] = counter[13] + 1;
                }
                if (protein1[i] == E)
                {
#pragma omp critical
                    counter[14] = counter[14] + 1;
                }
                if (protein1[i] == G)
                {
#pragma omp critical
                    counter[15] = counter[15] + 1;
                }
                if (protein1[i] == F)
                {
#pragma omp critical
                    counter[16] = counter[16] + 1;
                }
                if (protein1[i] == Y)
                {
#pragma omp critical
                    counter[17] = counter[17] + 1;
                }
                if (protein1[i] == stop)
                {
#pragma omp critical
                    counter[18] = counter[18] + 1;
                }
                if (protein1[i] == C)
                {
#pragma omp critical
                    counter[19] = counter[19] + 1;
                }
                if (protein1[i] == W)
                {
#pragma omp critical
                    counter[20] = counter[20] + 1;
                }
            }
        }
```

```
#pragma omp section
        {
            for (int i = ((5 * lengthP) / 16); i <
floor((6 * lengthP) / 16); i++)
            {

                if (protein1[i] == I)
                {
#pragma omp critical
                    counter[0] = counter[0] + 1;
                }
                if (protein1[i] == M)
                {
#pragma omp critical
                    counter[1] = counter[1] + 1;
                }
                if (protein1[i] == T)
                {
#pragma omp critical
                    counter[2] = counter[2] + 1;
                }
                if (protein1[i] == n)
                {
#pragma omp critical
                    counter[3] = counter[3] + 1;
                }
                if (protein1[i] == K)
                {
#pragma omp critical
                    counter[4] = counter[4] + 1;
                }
                if (protein1[i] == S)
                {
#pragma omp critical
                    counter[5] = counter[5] + 1;
                }
                if (protein1[i] == R)
                {
#pragma omp critical
                    counter[6] = counter[6] + 1;
                }
                if (protein1[i] == L)
                {
#pragma omp critical
                    counter[7] = counter[7] + 1;
                }
                if (protein1[i] == P)
                {
#pragma omp critical
                    counter[8] = counter[8] + 1;
                }
```

```
            if (protein1[i] == H)
            {
#pragma omp critical
                counter[9] = counter[9] + 1;
            }
            if (protein1[i] == Q)
            {
#pragma omp critical
                counter[10] = counter[10] + 1;
            }
            if (protein1[i] == V)
            {
#pragma omp critical
                counter[11] = counter[11] + 1;
            }
            if (protein1[i] == A)
            {
#pragma omp critical
                counter[12] = counter[12] + 1;
            }
            if (protein1[i] == D)
            {
#pragma omp critical
                counter[13] = counter[13] + 1;
            }
            if (protein1[i] == E)
            {
#pragma omp critical
                counter[14] = counter[14] + 1;
            }
            if (protein1[i] == G)
            {
#pragma omp critical
                counter[15] = counter[15] + 1;
            }
            if (protein1[i] == F)
            {
#pragma omp critical
                counter[16] = counter[16] + 1;
            }
            if (protein1[i] == Y)
            {
#pragma omp critical
                counter[17] = counter[17] + 1;
            }
```

```
            if (protein1[i] == stop)
            {
#pragma omp critical
                counter[18] = counter[18] + 1;
            }
            if (protein1[i] == C)
            {
#pragma omp critical
                counter[19] = counter[19] + 1;
            }
            if (protein1[i] == W)
            {
#pragma omp critical
                counter[20] = counter[20] + 1;
            }
        }
    }


#pragma omp section
    {
        for (int i = ((6 * lengthP) / 16); i <
floor((7 * lengthP) / 16); i++)
        {

            if (protein1[i] == I)
            {
#pragma omp critical
                counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
#pragma omp critical
                counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
#pragma omp critical
                counter[2] = counter[2] + 1;
            }
            if (protein1[i] == n)
            {
#pragma omp critical
                counter[3] = counter[3] + 1;
            }
```

```
            if (protein1[i] == K)
            {
#pragma omp critical
                counter[4] = counter[4] + 1;
            }
            if (protein1[i] == S)
            {
#pragma omp critical
                counter[5] = counter[5] + 1;
            }
            if (protein1[i] == R)
            {
#pragma omp critical
                counter[6] = counter[6] + 1;
            }
            if (protein1[i] == L)
            {
#pragma omp critical
                counter[7] = counter[7] + 1;
            }
            if (protein1[i] == P)
            {
#pragma omp critical
                counter[8] = counter[8] + 1;
            }
            if (protein1[i] == H)
            {
#pragma omp critical
                counter[9] = counter[9] + 1;
            }
            if (protein1[i] == Q)
            {
#pragma omp critical
                counter[10] = counter[10] + 1;
            }
            if (protein1[i] == V)
            {
#pragma omp critical
                counter[11] = counter[11] + 1;
            }
            if (protein1[i] == A)
            {
#pragma omp critical
                counter[12] = counter[12] + 1;
            }
            if (protein1[i] == D)
            {
#pragma omp critical
                counter[13] = counter[13] + 1;
            }
            if (protein1[i] == E)
            {
#pragma omp critical
                counter[14] = counter[14] + 1;
            }
            if (protein1[i] == G)
            {
#pragma omp critical
                counter[15] = counter[15] + 1;
            }
            if (protein1[i] == F)
            {
#pragma omp critical
                counter[16] = counter[16] + 1;
            }
            if (protein1[i] == Y)
            {
#pragma omp critical
                counter[17] = counter[17] + 1;
            }
            if (protein1[i] == stop)
            {
#pragma omp critical
                counter[18] = counter[18] + 1;
            }
            if (protein1[i] == C)
            {
#pragma omp critical
                counter[19] = counter[19] + 1;
            }
            if (protein1[i] == W)
            {
#pragma omp critical
                counter[20] = counter[20] + 1;
            }
        }

    }
```

```
#pragma omp section
        {
            for (int i = ((7 * lengthP) / 16); i <
floor((8 * lengthP) / 16); i++)
            {

                if (protein1[i] == I)
                {
#pragma omp critical
                    counter[0] = counter[0] + 1;
                }
                if (protein1[i] == M)
                {
#pragma omp critical
                    counter[1] = counter[1] + 1;
                }
                if (protein1[i] == T)
                {
#pragma omp critical
                    counter[2] = counter[2] + 1;
                }
                if (protein1[i] == n)
                {
#pragma omp critical
                    counter[3] = counter[3] + 1;
                }
                if (protein1[i] == K)
                {
#pragma omp critical
                    counter[4] = counter[4] + 1;
                }
                if (protein1[i] == S)
                {
#pragma omp critical
                    counter[5] = counter[5] + 1;
                }
                if (protein1[i] == R)
                {
#pragma omp critical
                    counter[6] = counter[6] + 1;
                }
                if (protein1[i] == L)
                {
#pragma omp critical
                    counter[7] = counter[7] + 1;
                }
                if (protein1[i] == P)
                {
#pragma omp critical
                    counter[8] = counter[8] + 1;
                }
                if (protein1[i] == H)
                {
#pragma omp critical
                    counter[9] = counter[9] + 1;
                }
                if (protein1[i] == Q)
                {
#pragma omp critical
                    counter[10] = counter[10] + 1;
                }
                if (protein1[i] == V)
                {
#pragma omp critical
                    counter[11] = counter[11] + 1;
                }
                if (protein1[i] == A)
                {
#pragma omp critical
                    counter[12] = counter[12] + 1;
                }
                if (protein1[i] == D)
                {
#pragma omp critical
                    counter[13] = counter[13] + 1;
                }
                if (protein1[i] == E)
                {
#pragma omp critical
                    counter[14] = counter[14] + 1;
                }
                if (protein1[i] == G)
                {
#pragma omp critical
                    counter[15] = counter[15] + 1;
                }
                if (protein1[i] == F)
                {
#pragma omp critical
                    counter[16] = counter[16] + 1;
                }
```

```
                if (protein1[i] == Y)
                {
#pragma omp critical
                    counter[17] = counter[17] + 1;
                }
                if (protein1[i] == stop)
                {
#pragma omp critical
                    counter[18] = counter[18] + 1;
                }
                if (protein1[i] == C)
                {
#pragma omp critical
                    counter[19] = counter[19] + 1;
                }
                if (protein1[i] == W)
                {
#pragma omp critical
                    counter[20] = counter[20] + 1;
                }
            }
        }


#pragma omp section
        {
        for (int i = ((8 * lengthP) / 16); i < floor((9 *
lengthP) / 16); i++)
        {

            if (protein1[i] == I)
            {
#pragma omp critical
                counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
#pragma omp critical
                counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
#pragma omp critical
                counter[2] = counter[2] + 1;
            }
            if (protein1[i] == n)
            {
#pragma omp critical
                counter[3] = counter[3] + 1;
            }
            if (protein1[i] == K)
            {
#pragma omp critical
                counter[4] = counter[4] + 1;
            }
            if (protein1[i] == S)
            {
#pragma omp critical
                counter[5] = counter[5] + 1;
            }
            if (protein1[i] == R)
            {
#pragma omp critical
                counter[6] = counter[6] + 1;
            }
            if (protein1[i] == L)
            {
#pragma omp critical
                counter[7] = counter[7] + 1;
            }
            if (protein1[i] == P)
            {
#pragma omp critical
                counter[8] = counter[8] + 1;
            }
            if (protein1[i] == H)
            {
#pragma omp critical
                counter[9] = counter[9] + 1;
            }
            if (protein1[i] == Q)
            {
#pragma omp critical
                counter[10] = counter[10] + 1;
            }
            if (protein1[i] == V)
            {
#pragma omp critical
                counter[11] = counter[11] + 1;
            }
            if (protein1[i] == A)
            {
#pragma omp critical
                counter[12] = counter[12] + 1;
            }
            if (protein1[i] == D)
            {
#pragma omp critical
                counter[13] = counter[13] + 1;
            }
```

```
        if (protein1[i] == E)
        {
#pragma omp critical
            counter[14] = counter[14] + 1;
        }
        if (protein1[i] == G)
        {
#pragma omp critical
            counter[15] = counter[15] + 1;
        }
        if (protein1[i] == F)
        {
#pragma omp critical
            counter[16] = counter[16] + 1;
        }
        if (protein1[i] == Y)
        {
#pragma omp critical
            counter[17] = counter[17] + 1;
        }
        if (protein1[i] == stop)
        {
#pragma omp critical
            counter[18] = counter[18] + 1;
        }
        if (protein1[i] == C)
        {
#pragma omp critical
            counter[19] = counter[19] + 1;
        }
        if (protein1[i] == W)
        {
#pragma omp critical
            counter[20] = counter[20] + 1;
        }
      }
      }
```

```
#pragma omp section
        {
        for (int i = ((9 * lengthP) / 16); i < floor((10
* lengthP) / 16); i++)
        {

            if (protein1[i] == I)
            {
#pragma omp critical
                counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
#pragma omp critical
                counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
#pragma omp critical
                counter[2] = counter[2] + 1;
            }
            if (protein1[i] == n)
            {
#pragma omp critical
                counter[3] = counter[3] + 1;
            }
            if (protein1[i] == K)
            {
#pragma omp critical
                counter[4] = counter[4] + 1;
            }
            if (protein1[i] == S)
            {
#pragma omp critical
                counter[5] = counter[5] + 1;
            }
            if (protein1[i] == R)
            {
#pragma omp critical
                counter[6] = counter[6] + 1;
            }
            if (protein1[i] == L)
            {
#pragma omp critical
                counter[7] = counter[7] + 1;
            }
```

```
if (protein1[i] == P)
            {
#pragma omp critical
                counter[8] = counter[8] + 1;
            }
            if (protein1[i] == H)
            {
#pragma omp critical
                counter[9] = counter[9] + 1;
            }
            if (protein1[i] == Q)
            {
#pragma omp critical
                counter[10] = counter[10] + 1;
            }
            if (protein1[i] == V)
            {
#pragma omp critical
                counter[11] = counter[11] + 1;
            }
            if (protein1[i] == A)
            {
#pragma omp critical
                counter[12] = counter[12] + 1;
            }
            if (protein1[i] == D)
            {
#pragma omp critical
                counter[13] = counter[13] + 1;
            }
            if (protein1[i] == E)
            {
#pragma omp critical
                counter[14] = counter[14] + 1;
            }
            if (protein1[i] == G)
            {
#pragma omp critical
                counter[15] = counter[15] + 1;
            }
            if (protein1[i] == F)
            {
#pragma omp critical
                counter[16] = counter[16] + 1;
            }
```

```
    if (protein1[i] == Y)
            {
#pragma omp critical
               counter[17] = counter[17] + 1;
            }
            if (protein1[i] == stop)
            {
#pragma omp critical
               counter[18] = counter[18] + 1;
            }
            if (protein1[i] == C)
            {
#pragma omp critical
               counter[19] = counter[19] + 1;
            }
            if (protein1[i] == W)
            {
#pragma omp critical
               counter[20] = counter[20] + 1;
            }
        }
        }
#pragma omp section
        {
        for (int i = ((10 * lengthP) / 16); i < floor((11
* lengthP) / 16); i++)
        {

            if (protein1[i] == I)
            {
#pragma omp critical
               counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
#pragma omp critical
               counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
#pragma omp critical
               counter[2] = counter[2] + 1;
            }
            if (protein1[i] == n)
            {
#pragma omp critical
               counter[3] = counter[3] + 1;
            }
            if (protein1[i] == K)
            {
#pragma omp critical
               counter[4] = counter[4] + 1;
            }
            if (protein1[i] == S)
            {
#pragma omp critical
               counter[5] = counter[5] + 1;
            }
            if (protein1[i] == R)
            {
#pragma omp critical
               counter[6] = counter[6] + 1;
            }
            if (protein1[i] == L)
            {
#pragma omp critical
               counter[7] = counter[7] + 1;
            }
            if (protein1[i] == P)
            {
#pragma omp critical
               counter[8] = counter[8] + 1;
            }
            if (protein1[i] == H)
            {
#pragma omp critical
               counter[9] = counter[9] + 1;
            }
            if (protein1[i] == Q)
            {
#pragma omp critical
               counter[10] = counter[10] + 1;
            }
            if (protein1[i] == V)
            {
#pragma omp critical
               counter[11] = counter[11] + 1;
            }
            if (protein1[i] == A)
            {
#pragma omp critical
               counter[12] = counter[12] + 1;
            }
```

```
            if (protein1[i] == D)
            {
#pragma omp critical
                counter[13] = counter[13] + 1;
            }
            if (protein1[i] == E)
            {
#pragma omp critical
                counter[14] = counter[14] + 1;
            }
            if (protein1[i] == G)
            {
#pragma omp critical
                counter[15] = counter[15] + 1;
            }
            if (protein1[i] == F)
            {
#pragma omp critical
                counter[16] = counter[16] + 1;
            }
            if (protein1[i] == Y)
            {
#pragma omp critical
                counter[17] = counter[17] + 1;
            }
            if (protein1[i] == stop)
            {
#pragma omp critical
                counter[18] = counter[18] + 1;
            }
            if (protein1[i] == C)
            {
#pragma omp critical
                counter[19] = counter[19] + 1;
            }
            if (protein1[i] == W)
            {
#pragma omp critical
                counter[20] = counter[20] + 1;
            }
        }
        }
```

```
#pragma omp section
        {
        for (int i = ((11 * lengthP) / 16); i < floor((12
* lengthP) / 16); i++)
        {

            if (protein1[i] == I)
            {
#pragma omp critical
                counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
#pragma omp critical
                counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
#pragma omp critical
                counter[2] = counter[2] + 1;
            }
            if (protein1[i] == n)
            {
#pragma omp critical
                counter[3] = counter[3] + 1;
            }
            if (protein1[i] == K)
            {
#pragma omp critical
                counter[4] = counter[4] + 1;
            }
            if (protein1[i] == S)
            {
#pragma omp critical
                counter[5] = counter[5] + 1;
            }
            if (protein1[i] == R)
            {
#pragma omp critical
                counter[6] = counter[6] + 1;
            }
            if (protein1[i] == L)
            {
#pragma omp critical
                counter[7] = counter[7] + 1;
            }
```

```
        if (protein1[i] == P)
         {
#pragma omp critical
            counter[8] = counter[8] + 1;
         }
         if (protein1[i] == H)
         {
#pragma omp critical
            counter[9] = counter[9] + 1;
         }
         if (protein1[i] == Q)
         {
#pragma omp critical
            counter[10] = counter[10] + 1;
         }
         if (protein1[i] == V)
         {
#pragma omp critical
            counter[11] = counter[11] + 1;
         }
         if (protein1[i] == A)
         {
#pragma omp critical
            counter[12] = counter[12] + 1;
         }
         if (protein1[i] == D)
         {
#pragma omp critical
            counter[13] = counter[13] + 1;
         }
         if (protein1[i] == E)
         {
#pragma omp critical
            counter[14] = counter[14] + 1;
         }
         if (protein1[i] == G)
         {
#pragma omp critical
            counter[15] = counter[15] + 1;
         }
         if (protein1[i] == F)
         {
#pragma omp critical
            counter[16] = counter[16] + 1;
         }
```

```
         if (protein1[i] == Y)
         {
#pragma omp critical
            counter[17] = counter[17] + 1;
         }
         if (protein1[i] == stop)
         {
#pragma omp critical
            counter[18] = counter[18] + 1;
         }
         if (protein1[i] == C)
         {
#pragma omp critical
            counter[19] = counter[19] + 1;
         }
         if (protein1[i] == W)
         {
#pragma omp critical
            counter[20] = counter[20] + 1;
         }
      }
      }
```

```
#pragma omp section
        {
        for (int i = ((12 * lengthP) / 16); i < floor((13
* lengthP) / 16); i++)
        {

            if (protein1[i] == I)
            {
#pragma omp critical
                counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
#pragma omp critical
                counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
#pragma omp critical
                counter[2] = counter[2] + 1;
            }
            if (protein1[i] == n)
            {
#pragma omp critical
                counter[3] = counter[3] + 1;
            }
            if (protein1[i] == K)
            {
#pragma omp critical
                counter[4] = counter[4] + 1;
            }
            if (protein1[i] == S)
            {
#pragma omp critical
                counter[5] = counter[5] + 1;
            }
            if (protein1[i] == R)
            {
#pragma omp critical
                counter[6] = counter[6] + 1;
            }
            if (protein1[i] == L)
            {
#pragma omp critical
                counter[7] = counter[7] + 1;
            }
            if (protein1[i] == P)
            {
#pragma omp critical
                counter[8] = counter[8] + 1;
            }
            if (protein1[i] == H)
            {
#pragma omp critical
                counter[9] = counter[9] + 1;
            }
            if (protein1[i] == Q)
            {
#pragma omp critical
                counter[10] = counter[10] + 1;
            }
            if (protein1[i] == V)
            {
#pragma omp critical
                counter[11] = counter[11] + 1;
            }
            if (protein1[i] == A)
            {
#pragma omp critical
                counter[12] = counter[12] + 1;
            }
            if (protein1[i] == D)
            {
#pragma omp critical
                counter[13] = counter[13] + 1;
            }
            if (protein1[i] == E)
            {
#pragma omp critical
                counter[14] = counter[14] + 1;
            }
            if (protein1[i] == G)
            {
#pragma omp critical
                counter[15] = counter[15] + 1;
            }
            if (protein1[i] == F)
            {
#pragma omp critical
                counter[16] = counter[16] + 1;
            }
```

```
            if (protein1[i] == Y)
            {
#pragma omp critical
                counter[17] = counter[17] + 1;
            }
            if (protein1[i] == stop)
            {
#pragma omp critical
                counter[18] = counter[18] + 1;
            }
            if (protein1[i] == C)
            {
#pragma omp critical
                counter[19] = counter[19] + 1;
            }
            if (protein1[i] == W)
            {
#pragma omp critical
                counter[20] = counter[20] + 1;
            }
        }
        }


#pragma omp section
        {
        for (int i = ((13 * lengthP) / 16); i < floor((14
* lengthP) / 16); i++)
        {

            if (protein1[i] == I)
            {
#pragma omp critical
                counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
#pragma omp critical
                counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
#pragma omp critical
                counter[2] = counter[2] + 1;
            }
        if (protein1[i] == n)
            {
#pragma omp critical
                counter[3] = counter[3] + 1;
            }
            if (protein1[i] == K)
            {
#pragma omp critical
                counter[4] = counter[4] + 1;
            }
            if (protein1[i] == S)
            {
#pragma omp critical
                counter[5] = counter[5] + 1;
            }
            if (protein1[i] == R)
            {
#pragma omp critical
                counter[6] = counter[6] + 1;
            }
            if (protein1[i] == L)
            {
#pragma omp critical
                counter[7] = counter[7] + 1;
            }
            if (protein1[i] == P)
            {
#pragma omp critical
                counter[8] = counter[8] + 1;
            }
            if (protein1[i] == H)
            {
#pragma omp critical
                counter[9] = counter[9] + 1;
            }
            if (protein1[i] == Q)
            {
#pragma omp critical
                counter[10] = counter[10] + 1;
            }
            if (protein1[i] == V)
            {
#pragma omp critical
                counter[11] = counter[11] + 1;
            }
```

```
if (protein1[i] == A)
            {
#pragma omp critical
                counter[12] = counter[12] + 1;
            }
            if (protein1[i] == D)
            {
#pragma omp critical
                counter[13] = counter[13] + 1;
            }
            if (protein1[i] == E)
            {
#pragma omp critical
                counter[14] = counter[14] + 1;
            }
            if (protein1[i] == G)
            {
#pragma omp critical
                counter[15] = counter[15] + 1;
            }
            if (protein1[i] == F)
            {
#pragma omp critical
                counter[16] = counter[16] + 1;
            }
            if (protein1[i] == Y)
            {
#pragma omp critical
                counter[17] = counter[17] + 1;
            }
            if (protein1[i] == stop)
            {
#pragma omp critical
                counter[18] = counter[18] + 1;
            }
            if (protein1[i] == C)
            {
#pragma omp critical
                counter[19] = counter[19] + 1;
            }
            if (protein1[i] == W)
            {
#pragma omp critical
                counter[20] = counter[20] + 1;
            }
        }
         }
```

```
#pragma omp section
        {
        for (int i = ((14 * lengthP) / 16); i < floor((15
* lengthP) / 16); i++)
        {

            if (protein1[i] == I)
            {
#pragma omp critical
                counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
#pragma omp critical
                counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
#pragma omp critical
                counter[2] = counter[2] + 1;
            }
            if (protein1[i] == n)
            {
#pragma omp critical
                counter[3] = counter[3] + 1;
            }
            if (protein1[i] == K)
            {
#pragma omp critical
                counter[4] = counter[4] + 1;
            }
            if (protein1[i] == S)
            {
#pragma omp critical
                counter[5] = counter[5] + 1;
            }
            if (protein1[i] == R)
            {
#pragma omp critical
                counter[6] = counter[6] + 1;
            }
            if (protein1[i] == L)
            {
#pragma omp critical
                counter[7] = counter[7] + 1;
            }
            if (protein1[i] == P)
            {
#pragma omp critical
                counter[8] = counter[8] + 1;
            }
```

```
            if (protein1[i] == H)
            {
#pragma omp critical
                counter[9] = counter[9] + 1;
            }
            if (protein1[i] == Q)
            {
#pragma omp critical
                counter[10] = counter[10] + 1;
            }
            if (protein1[i] == V)
            {
#pragma omp critical
                counter[11] = counter[11] + 1;
            }
            if (protein1[i] == A)
            {
#pragma omp critical
                counter[12] = counter[12] + 1;
            }
            if (protein1[i] == D)
            {
#pragma omp critical
                counter[13] = counter[13] + 1;
            }
            if (protein1[i] == E)
            {
#pragma omp critical
                counter[14] = counter[14] + 1;
            }
            if (protein1[i] == G)
            {
#pragma omp critical
                counter[15] = counter[15] + 1;
            }
            if (protein1[i] == F)
            {
#pragma omp critical
                counter[16] = counter[16] + 1;
            }
            if (protein1[i] == Y)
            {
#pragma omp critical
                counter[17] = counter[17] + 1;
            }
```

```
            if (protein1[i] == stop)
            {
#pragma omp critical
                counter[18] = counter[18] + 1;
            }
            if (protein1[i] == C)
            {
#pragma omp critical
                counter[19] = counter[19] + 1;
            }
            if (protein1[i] == W)
            {
#pragma omp critical
                counter[20] = counter[20] + 1;
            }
        }
         }


#pragma omp section
        {
        for (int i = ((15 * lengthP) / 16); i < floor((16
* lengthP) / 16); i++)
        {

            if (protein1[i] == I)
            {
#pragma omp critical
                counter[0] = counter[0] + 1;
            }
            if (protein1[i] == M)
            {
#pragma omp critical
                counter[1] = counter[1] + 1;
            }
            if (protein1[i] == T)
            {
#pragma omp critical
                counter[2] = counter[2] + 1;
            }
            if (protein1[i] == n)
            {
#pragma omp critical
                counter[3] = counter[3] + 1;
            }
```

```
        if (protein1[i] == K)
        {
#pragma omp critical
            counter[4] = counter[4] + 1;
        }
        if (protein1[i] == S)
        {
#pragma omp critical
            counter[5] = counter[5] + 1;
        }
        if (protein1[i] == R)
        {
#pragma omp critical
            counter[6] = counter[6] + 1;
        }
        if (protein1[i] == L)
        {
#pragma omp critical
            counter[7] = counter[7] + 1;
        }
        if (protein1[i] == P)
        {
#pragma omp critical
            counter[8] = counter[8] + 1;
        }
        if (protein1[i] == H)
        {
#pragma omp critical
            counter[9] = counter[9] + 1;
        }
        if (protein1[i] == Q)
        {
#pragma omp critical
            counter[10] = counter[10] + 1;
        }
        if (protein1[i] == V)
        {
#pragma omp critical
            counter[11] = counter[11] + 1;
        }
        if (protein1[i] == A)
        {
#pragma omp critical
            counter[12] = counter[12] + 1;
        }
```

```
        if (protein1[i] == D)
        {
#pragma omp critical
            counter[13] = counter[13] + 1;
        }
        if (protein1[i] == E)
        {
#pragma omp critical
            counter[14] = counter[14] + 1;
        }
        if (protein1[i] == G)
        {
#pragma omp critical
            counter[15] = counter[15] + 1;
        }
        if (protein1[i] == F)
        {
#pragma omp critical
            counter[16] = counter[16] + 1;
        }
        if (protein1[i] == Y)
        {
#pragma omp critical
            counter[17] = counter[17] + 1;
        }
        if (protein1[i] == stop)
        {
#pragma omp critical
            counter[18] = counter[18] + 1;
        }
        if (protein1[i] == C)
        {
#pragma omp critical
            counter[19] = counter[19] + 1;
        }
        if (protein1[i] == W)
        {
#pragma omp critical
            counter[20] = counter[20] + 1;
        }
      }
      }
    }
  }
```

```
end = omp_get_wtime();
    for (int i = 0; i < 21; i++)
    {
        percent = (float)(((float)counter[i] / (float)lengthP) * 100);
        sum = sum + percent;
        cout << "Frequency of:  " << letters[i] << " is " << counter[i] << " Percentage is  " << percent <<
"%" << "\n";
    }

    cout << "Time" << end - start << "\n";
    cout << "Length of protein: " << lengthP << "\n";
    cout << "Total Frequency: " << sum << "%" << "\n";
}
int main()
{
    cout << "The dna seq" << "\n";
    read_seq();
    cout << "The amino acid" << "\n";
   serialcode();
   //paralelcode();
  // prallelSectionCode();
}
```