Indian Institute of Technology Indore

Report for Lab Assignment 1

Parallel Computing Lab (CS 359)

Author
Aniket Sangwan (cse180001005@iiti.ac.in)

April 13, 2021

Contents

Chapter 1

Report

1.1 Problem Statement

Write a parallel program to parse a string of symbols. The inputs are a context-free grammar G in Chomsky Normal Form and a string of symbols. In the end, the program should print yes if the string of symbols can be derived by the rules of the grammar and no otherwise. Write a sequential program (no OpenMP directives at all) as well. Compare the running time of the sequential program with the running time of the parallel program and compute the speedup for different grammars and different string lengths.

1.2 Approach

Cocke–Younger–Kasami(CYK) Algorithm is a dynamic programming based algorithm for parsing strings for a given context free grammar in Chomsky Normal Form. The algorithm is based on the principle that the solution to problem [i, j] can constructed from solution to subproblem [i, k] and solution to sub problem [k, j]. It uses the result of previously parsed substring. Thus, it is based on DP.

- We can parallelize the substrings of same length as they only depend on previously parsed substrings, which are always shorter.
- We can also parallelize every loop iterating through all the productions because they are independent of each other.

1.3 Input

- The first line of input contains the number of productions n in the given CNF grammar.
- The next n contains each production in a new line. For Example:

4

S -> AC | AB

C -> SB

A -> a

B -> b

aabb.

- The third line contains the string to be parsed.

1.4 Output

- The first line states whether the string can be parsed or not.
- The second line outputs the time taken by program.
- You can view the parsing table using print_table() function. (Uncomment the line 206 in the OPENMP code).

1.5 Complexity

- The complexity of the serial program is $\mathbf{O(n^3 * |G|)}$, where n is the length of the string to be parsed and $|G|$ is the length of the grammar.
- The complexity of the parallel program (OPENMP) is $\mathbf{O(n^3 * |G|/no\_threads)}$, where n is the length of the string to be parsed and $|G|$ is the length of the grammar.

1.6 Runtime Comparsion

To evaluate the program for different cases, I used **three different grammars** (available in the files g1.txt, g2.txt and g3.txt) and generated **three different test files of length 200, 800 and 1500**. I compared the runtime of each test case for serial and parallel programs and generated graphs to visualise the improvement.

1.6.1 Grammar 1 (g1.txt)

The grammar used is:

S->AC|AB

C->SB

A->a

B->b.

The images below show the input and output for above grammar using different number of threads.

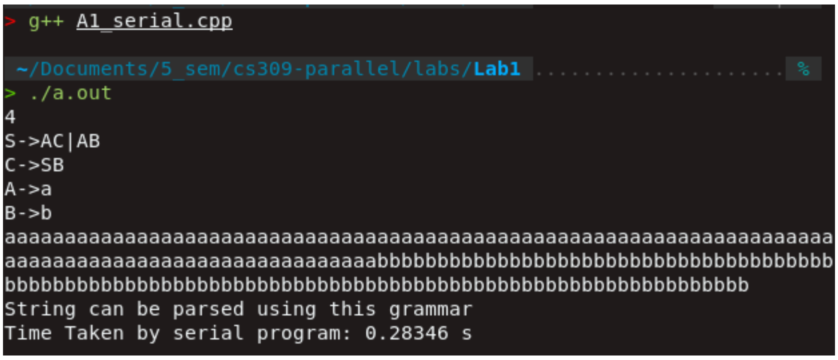


Figure 1.1: Serial Program

```
> g++ A1_openmp.cpp -fopenmp

 ~/Documents/5_sem/cs309-parallel/labs/Lab1 ..................... %
> ./a.out 2
4
S->AC|AB
C->SB
A->a
B->b
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
String can be parsed using this grammar
Time Taken by openmp program: 0.12829 s

 ~/Documents/5_sem/cs309-parallel/labs/Lab1 ..................... %
> ./a.out 3
4
S->AC|AB
C->SB
A->a
B->b
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
String can be parsed using this grammar
Time Taken by openmp program: 0.11161 s

 ~/Documents/5_sem/cs309-parallel/labs/Lab1 ..................... %
> ./a.out 4
4
S->AC|AB
C->SB
A->a
B->b
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
String can be parsed using this grammar
Time Taken by openmp program: 0.09739 s
```

Figure 1.2: OPENMP Program

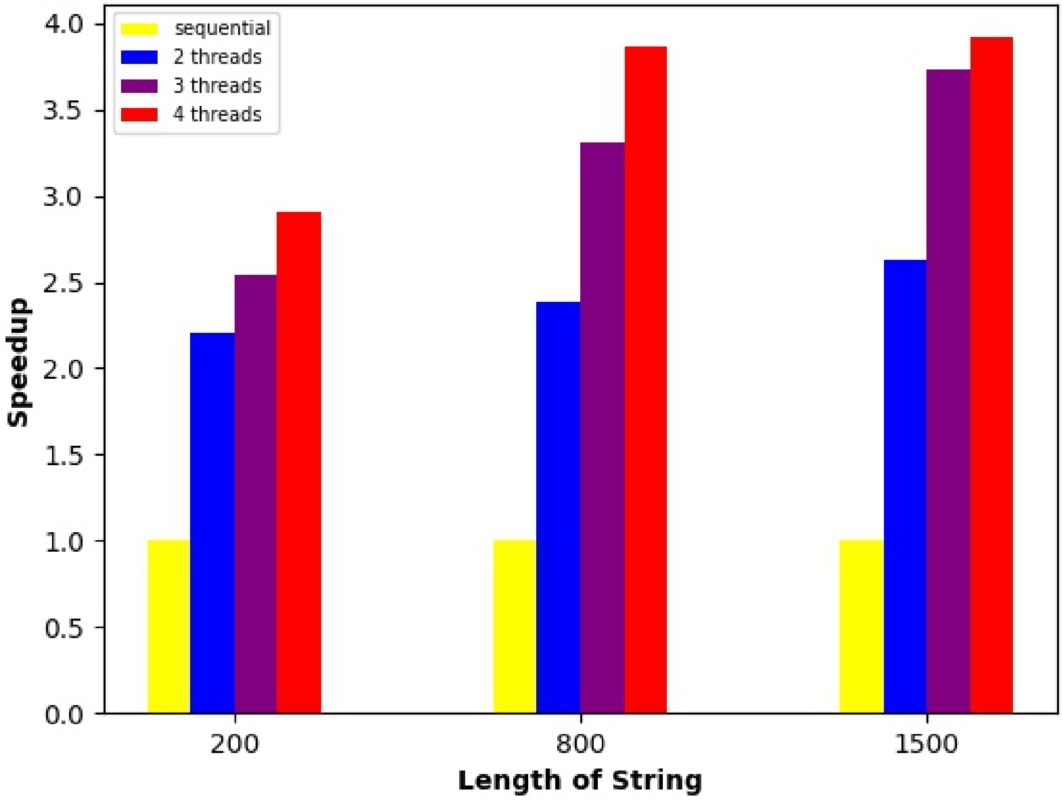


Figure 1.3: Speedup vs Length of string for Grammar 1

The below table displays the runtime comparison for different tests.

	Time (in sec)			
String Length	Sequential	2 Threads	3 Threads	4 Threads
200	0.28346	0.12829	0.11161	0.09739
800	10.97589	4.60321	3.32380	2.84105
1500	94.29656	35.81603	25.27589	24.07729

Table 1.1: Run-Time

1.6.2 Grammar 2 (g2.txt)

The grammar used is:

S -> BA | DC | c

A -> a

B -> AS

C -> b

D -> CS

The images below show the input and output for above grammar using different number of threads.

Figure 1.4: Serial Program

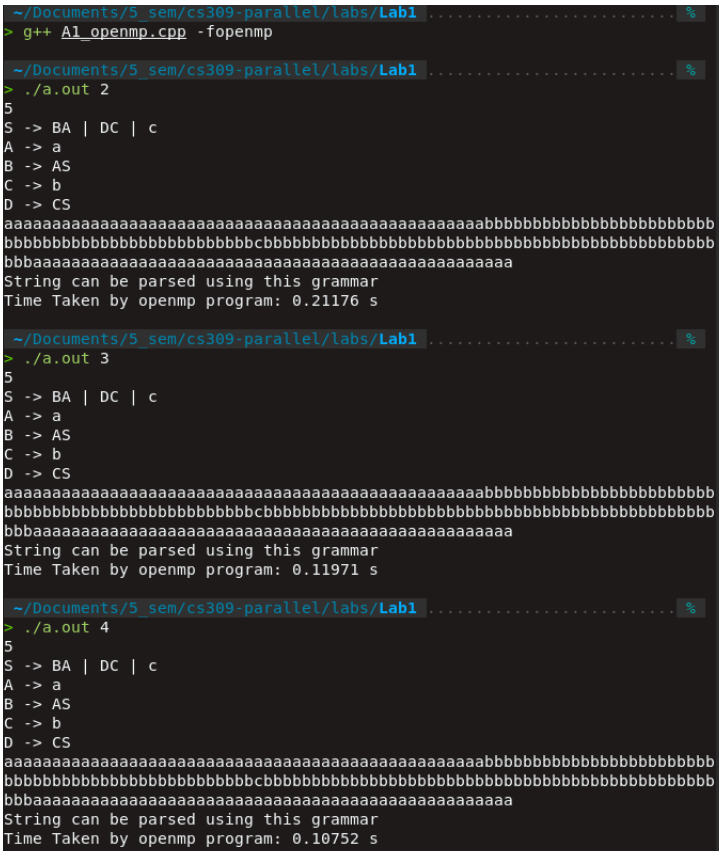


Figure 1.5: OPENMP Program

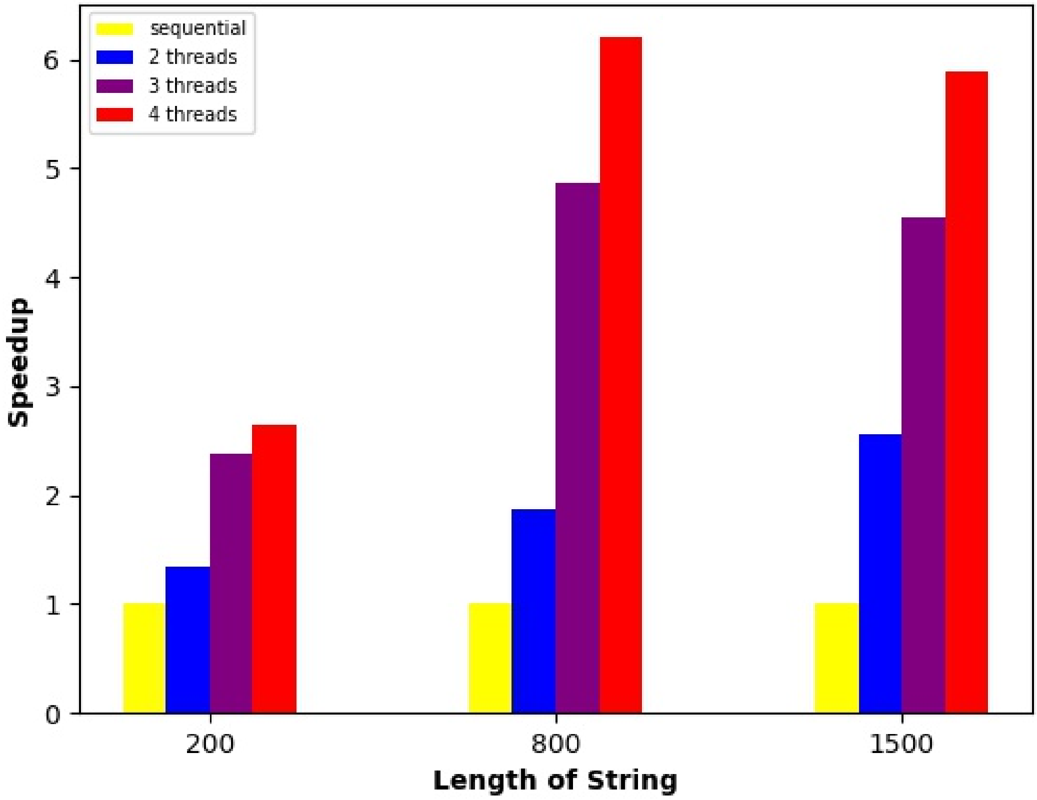


Figure 1.6: Speedup vs Length of string for Grammar 2

The below table displays the runtime comparison for different tests.

	Time (in sec)			
String Length	Sequential	2 Threads	3 Threads	4 Threads
200	0.28852	0.21176	0.11971	0.10752
800	14.18116	7.60084	2.90939	2.28607
1500	96.42192	37.58093	21.19945	16.35429

Table 1.2: Run-Time

1.6.3 Grammar 3 (g3.txt)

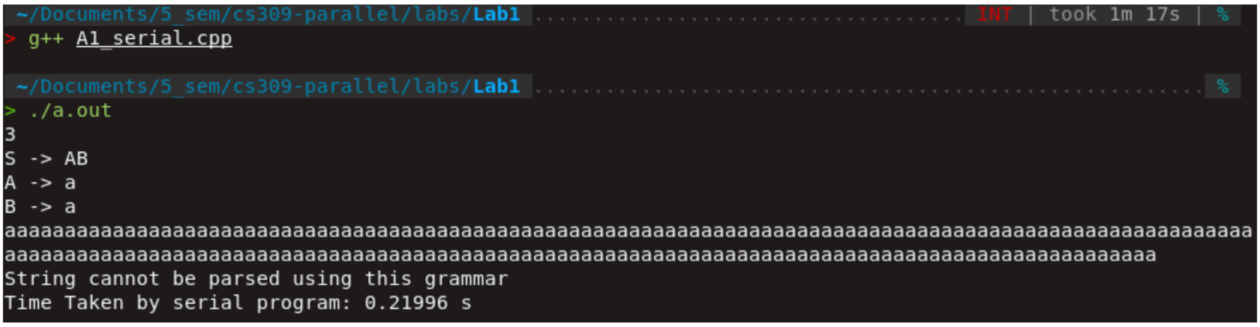
The grammar used is:

S -> AB

A -> a

B -> a

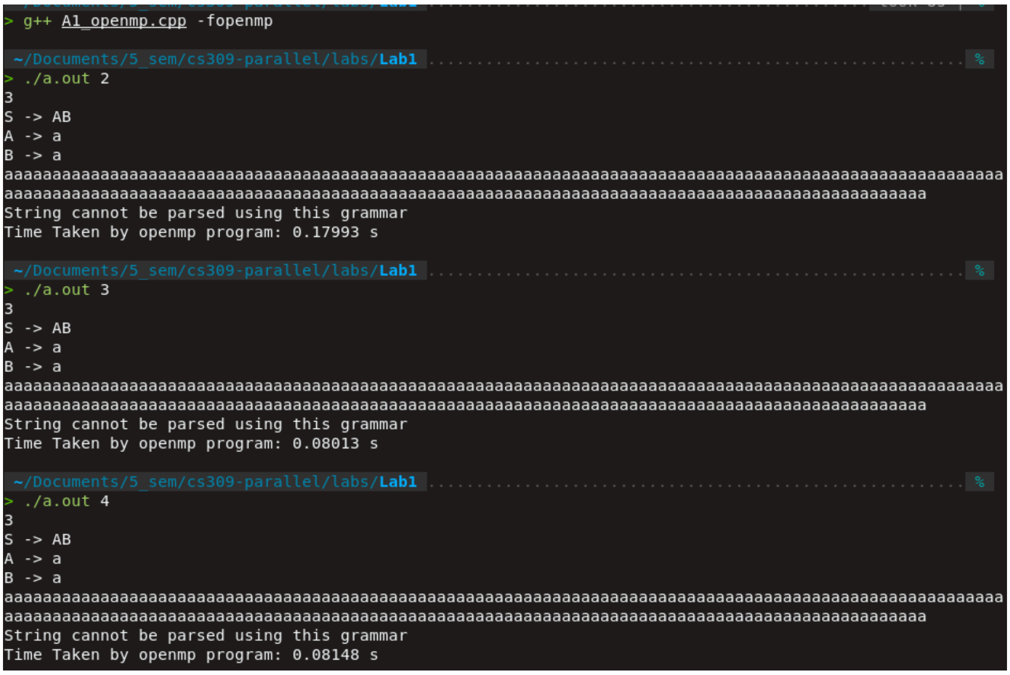
The images below show the input and output for above grammar using different number of threads.



```
~/Documents/5_sem/cs309-parallel/labs/Lab1 .................................. INT | took 1m 17s | %
> g++ A1_serial.cpp

~/Documents/5_sem/cs309-parallel/labs/Lab1 ................................................ %
> ./a.out
3
S -> AB
A -> a
B -> a
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
String cannot be parsed using this grammar
Time Taken by serial program: 0.21996 s
```

Figure 1.7: Serial Program



```
> g++ A1_openmp.cpp -fopenmp

~/Documents/5_sem/cs309-parallel/labs/Lab1 ................................................ %
> ./a.out 2
3
S -> AB
A -> a
B -> a
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
String cannot be parsed using this grammar
Time Taken by openmp program: 0.17993 s

~/Documents/5_sem/cs309-parallel/labs/Lab1 ................................................ %
> ./a.out 3
3
S -> AB
A -> a
B -> a
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
String cannot be parsed using this grammar
Time Taken by openmp program: 0.08013 s

~/Documents/5_sem/cs309-parallel/labs/Lab1 ................................................ %
> ./a.out 4
3
S -> AB
A -> a
B -> a
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
String cannot be parsed using this grammar
Time Taken by openmp program: 0.08148 s
```

Figure 1.8: OPENMP Program

String Length	Time (in sec)			
	Sequential	2 Threads	3 Threads	4 Threads
200	0.21996	0.17993	0.08013	0.08148
800	8.40129	7.86845	2.90648	2.45585
1500	100.83426	30.43219	20.60630	16.58329

Table 1.3: Run-Time

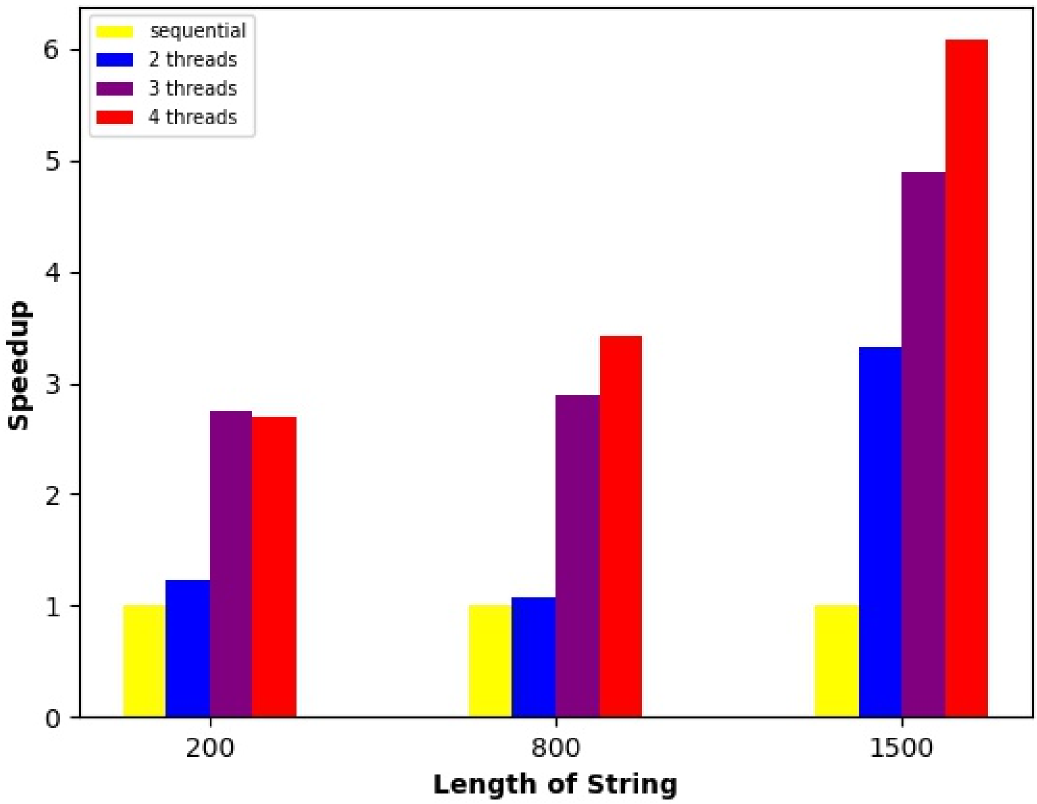


Figure 1.9: Speedup vs Length of string for Grammar 2

1.6.4 Conclusion

We conclude that the runtime reduces with a increase in the number of threads. The speedup also increases with increase in input size. Hence, this algorithm performs better for larger test cases.

1.6.5 Implementation

```
// This code is inspired from https://github.com/ahmadshafique/CYK-Parser/
    blob/master/CYK%20parser.cpp

#include<bits/stdc++.h>
#include <omp.h>

using namespace std;

#define pb push_back
#define endl "\n"

#define N 1600

string grammar[N][N]; // Stores the grammar provided by the user
string temp[N];
string parse_matrix[N][N];
int p,no_prod;     // no_prod stores the no of productions
int size;        // No of threads

void break_grammar(string a){ //Stores the RHS of grammar in the string
    array temp. Used to separate productions.

  int i;
  p=0;
  while(a.size()){

    i=a.find("|");

    int w=i;
    while(a[w-1]==' ')w--;  // Ignores space

    if(w>a.size()){

      temp[p++] = a;
      a="";
    }
    else{

      temp[p++] = a.substr(0,w);
      w=i;
      while(a[w+1]==' ')w++;  // Ignores space
      a=a.substr(w+1,a.size());
    }
  }
}

string join(string a,string b){ //Joins different non terminals to get new
    parsing matrix value

```

```
  int i;
  string curr=a;
  for(i=0;i<b.size();i++)
    if(curr.find(b[i]) > curr.size())
      curr+=b[i];
  return (curr);
}

string check_red(string p){ //Iterates through complete grammar to check if the string p can be reduced or not

  int j,k;
  string curr1="";
  #pragma omp parallel num_threads(size)
  {
    string curr="";

    #pragma omp for     // Parallelizes the next loop
    for(j=0;j<no_prod;j++){

      int k=1;

      while(grammar[j][k]!=""){

        if(grammar[j][k]==p){

          curr=join(curr,grammar[j][0]);
        }

        k++;
      }
    }
    #pragma omp critical  // To prevent concurrent writes
    curr1=join(curr1,curr);
  }
  return curr1;
}

string get_all_comb(string a, string b){ //Creates every possible combination of strings a and b

  int i,j;
  string temp1=a,re="";
  for(i=0;i<a.size();i++)
    for(j=0;j<b.size();j++){

      temp1="";
      temp1=temp1+a[i]+b[j];
      re=re+check_red(temp1);   //Checks if the current string can be obtained or not
    }
  return re;
}

void print_table(string str){ // Prints the parsing table
  for(int i=0;i<str.size();i++)
  {
    int k=0;
```

```
    int l=str.size()-i-1;
    for(int j=l;j<str.size();j++)
    {
      if(parse_matrix[k][j]!="")
      cout<<parse_matrix[k][j]<<"\t";
      else cout<<"phi\t";
      k++;
    }
    cout<<endl;
  }
}

int main(int argc, char **argv){

  int i,u,j,l,k;
  string a,str,curr,temp1,start;
  size = atoi(argv[1]);

  start="S";

  // cout<<"\nEnter the number of productions\n";
  // cout<<"Also enter each production in the format (S->AB|CD|a)\n";
  cin >> no_prod;
  cin.ignore();
  for(i=0;i<no_prod;i++){

    getline(cin,a);

    u=a.find("->");
    int w=u;

    while(a[w-1]==' ')w--; // Ignores space in grammar

    grammar[i][0] = a.substr(0,w);

    w=u+1;
    while(a[w+1]==' ')w++; // Ignores space in grammar

    a = a.substr(w+1, a.size());

    break_grammar(a);
    for(j=0;j<p;j++)
    {
      grammar[i][j+1]=temp[j];
    }
  }

  string st;

  // cout<<"\nEnter the string of symbols you need to check: ";
  getline(cin,str);

  double start_time=omp_get_wtime();

  for(i=0;i<str.size();i++){    //Assigns values to principal diagonal of matrix

    st = "";
```

```
      st+=str[i];

    #pragma omp parallel num_threads(size)
    {
      string curr="";

      #pragma omp for       // Parallelizes the next loop

      for(j=0;j<no_prod;j++){

        int k=1;
        while(grammar[j][k] != ""){

          if(grammar[j][k] == st){

            curr=join(curr,grammar[j][0]);
          }
          k++;
        }
      }
      #pragma omp critical  // To prevent concurrent writes
      parse_matrix[i][i]+=curr;
    }
  }

  for(k=1;k<str.size();k++){  //Assigns values to upper half of the matrix

    for(j=k;j<str.size();j++){

      #pragma omp parallel num_threads(size)
      {
        string curr="";

        #pragma omp for       // Parallelizes the next loop

        for(l=j-k;l<j;l++){

          string temp1 = get_all_comb(parse_matrix[j-k][l],parse_matrix[l
    +1][j]);
          curr = join(curr,temp1);
        }
        #pragma omp critical  // To prevent concurrent writes
        parse_matrix[j-k][j]+= curr;
      }
    }
  }

  // Prints the Parsing Table
  // print_table(str);

  int f=0;

  if(parse_matrix[0][str.size()-1].find(start)<=parse_matrix[0][str.size()
    -1].size()){  //Checks if last element of first row contains a Start
    variable
    cout<<"String can be parsed using this grammar\n";
  }else{
    cout<<"String cannot be parsed using this grammar\n";
```

```
    }


    double final_tot=omp_get_wtime()-start_time;

    cout<<fixed<<setprecision(5) <<"Time Taken by openmp program: "<<
      final_tot <<" s "<<endl;

    return 0;
}
```