Assignment 1

Parallel Computing(CS 309)

RUCHIR MEHTA
Roll No: 180001044

Date: 12-04-2021

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.

Code files for serial and parallel implementation: [Attached]

Main Code Snippet:

```
for(int j=0;j<n;j++){
    getVarTerm(R,s[j],table[n-1][j]);
}
for(int i=n-2;i>=0;i--){
    for(int j=0;j<=i;j++){
        pair<int,int> interval=getInterval(i,j,n);
        int diff=interval.second-interval.first;
        set<vector<string>> cross;
        rep(k,diff){
            pair<int,int> a= getTableIndex(interval.first,interval.first+k,n);
            pair<int,int> b= getTableIndex(interval.first+k+1,interval.second,n);

            for(auto s1:table[a.first][a.second]){
                for(auto s2:table[b.first][b.second]){
                    vector<string> temp={s1,s2};
                    cross.insert(temp);
                }
            }
        }
        for(auto x:cross)
            getVarVar(R,x,table[i][j]);
    }
}
```

Serial Execution

```
#pragma omp parallel for
for(int j=0;j<n;j++){
    getVarTerm(R,s[j],table[n-1][j]);
}
for(int i=n-2;i>=0;i--){
    #pragma omp parallel for
    for(int j=0;j<=i;j++){
        pair<int,int> interval=getInterval(i,j,n);
        int diff=interval.second-interval.first;
        set<vector<string>> cross;
        rep(k,diff){
            pair<int,int> a= getTableIndex(interval.first,interval.first+k,n);
            pair<int,int> b= getTableIndex(interval.first+k+1,interval.second,n);

            for(auto s1:table[a.first][a.second]){
                for(auto s2:table[b.first][b.second]){
                    vector<string> temp={s1,s2};
                    cross.insert(temp);
                }
            }
        }
        for(auto x:cross)
            getVarVar(R,x,table[i][j]);
    }
}
```

Parallel Execution

The approach used is Multithreading using OpenMP

Testing sample input

```
S A B C END
S
a b END
5
S AC
S AB
C SB
A a
B b
aaabbb
```

Grammar input

```
String to check membership is aaabbb
Serial Time taken: 0.000107558
Passed!!

Table Below:
{S, }
{}        {C, }
{}        {S, }        {}
{}        {}        {C, }        {}
{}        {}        {S, }        {}        {}
{A, }        {A, }        {A, }        {B, }        {B, }        {B, }
```

CYK Algorithm Serial implementation(Table Filling Method)

```
String to check membership is aaabbb
Time Taken by:
8 threads: 0.00576055 with speedup 0.0186715
Passed!!

Table Below:
{S, }
{}        {C, }
{}        {S, }        {}
{}        {}        {C, }        {}
{}        {}        {S, }        {}        {}
{A, }        {A, }        {A, }        {B, }        {B, }        {B, }
```

CYK Algorithm Parallel implementation(Table Filling Method)

Time Analysis(Time in seconds):
Test Case1 Input:

```
S A B C D END
S
c a b END
7
S BA
S DC
S c
A a
B AS
C b
D CS
aaaabaaaaaabbbbbbcbbbbbbaaaaaabaaaa
```

```
String to check membership is aaaabaaaaaabbbbbbcbbbbbbaaaaaabaaaa
Serial Time taken: 0.001298
Passed!!
```

```
String to check membership is aaaabaaaaaabbbbbbcbbbbbbaaaaaabaaaa
Time Taken by:
1 threads: 0.0014842 with speedup 0.0724688
2 threads: 0.000913367 with speedup 0.11776
3 threads: 0.000884614 with speedup 0.121587
4 threads: 0.00154057 with speedup 0.0698171
5 threads: 0.0010796 with speedup 0.0996276
6 threads: 0.00288955 with speedup 0.0372231
7 threads: 0.0207908 with speedup 0.00517336
8 threads: 0.0759907 with speedup 0.00141541
9 threads: 0.00217675 with speedup 0.0494122
10 threads: 0.00179357 with speedup 0.0599686
Passed!!
```

Test Case2 Input:

```
S A B C END
S
a b END
5
S AC
S AB
C SB
A a
B b
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
String to check membership is aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Serial Time taken: 43.8998
Passed!!
```

```
String to check membership is aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Time Taken by:
1 threads: 52.607 with speedup 0.834486
Passed!!
2 threads: 28.1827 with speedup 1.55769
Passed!!
3 threads: 21.9066 with speedup 2.00396
Passed!!
4 threads: 14.8871 with speedup 2.94885
Passed!!
5 threads: 16.0008 with speedup 2.74361
Passed!!
6 threads: 14.0441 with speedup 3.12586
Passed!!
7 threads: 12.4711 with speedup 3.52011
Passed!!
8 threads: 12.5931 with speedup 3.48601
Passed!!
9 threads: 16.1061 with speedup 2.72566
Passed!!
10 threads: 15.4243 with speedup 2.84615
Passed!!
```

Test Case3:

```
S A B C D END
S
c a b END
7
S BA
S DC
S c
A a
B AS
C b
D CS
baaaabaaaaaabbbbbbccbbbbbbaaaaaabaaaab
```

```
String to check membership is baaaabaaaaaabbbbbbccbbbbbbaaaaaabaaaab
Serial Time taken: 0.0010059

Fail!!
```

```
String to check membership is baaaabaaaaaabbbbbbccbbbbbbaaaaaabaaaab
Time Taken by:
1 threads: 0.00138012 with speedup 0.72885
2 threads: 0.000530508 with speedup 1.89611
3 threads: 0.000350497 with speedup 2.86992
4 threads: 0.000469518 with speedup 2.14241
5 threads: 0.00183105 with speedup 0.549357
6 threads: 0.000486349 with speedup 2.06827
7 threads: 0.00239362 with speedup 0.420243
8 threads: 0.00294568 with speedup 0.341484
9 threads: 0.00412284 with speedup 0.243982
10 threads: 0.00339304 with speedup 0.29646

Fail!!
```

Observations:

When the number of nodes and edges are very less then the speedup is less than one as the time gets wasted in thread creation. But as the problem size increases, we see a good amount of speedup in the parallel algorithm. For test case 1, the speedup is highest with 3 threads. For test case 2, the speedup is highest with 7 threads. For test case 3, the speedup is highest with 3 threads.