جامعة الملك سعود
King Saud University

Parallel Processing

Project 2 Phase3

CYK Algorithm and Context-Free Grammar

Students:

Al-Anoud Al-Subaie

Raghad Al-Suhaibani

Ftoon Al-Rubayea

Table of contents:

Table of Figure :

1.Problem Definition :

P1 . Write a multicore program that uses CYK algorithm [13] to parse a string of symbols. The inputs are a context-free grammar Gin Chomsky Normal Form and a string of symbols. At 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 multicore-program and compute the speedup for different grammars and different string lengths .

2.Write a Multicore Program Using CYK Algorithm :

```
}
#pragma omp parallel for collapse(4) num_threads(4)
for (size_t j = 2; j <= word_len; j++) {
    for (size_t i = 1; i <= word_len - j + 1; i++) {
        for (size_t k = 1; k <= j - 1; k++) {
            for (size_t ruleIdx = 0; ruleIdx < pGrammar->grammar_size; ruleIdx++) {
                grammar_entry_t *pRule = pGrammar->grammar_entries + ruleIdx;
                if (pRule->single_char) {
                    continue;
                }

                alpha_t c1, c2;
                c1 = pRule->pTerminals[0];
                c2 = pRule->pTerminals[1];

                if (table_char_exists(pTable[TABLE_IDX(word_len, k, i)], c1) &&
                    table_char_exists(pTable[TABLE_IDX(word_len, j - k, i + k)], c2)) {
                    table_push_back(pTable, TABLE_IDX(word_len, j, i), pRule->non_terminal);
                }
            }
        }
    }
}
```

Figure 0.1

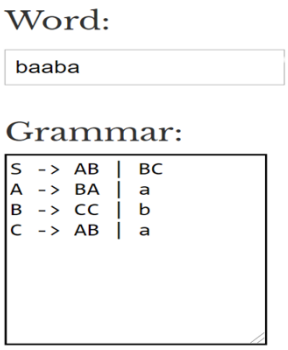
In **Figure** 0.1 we are add in method CYK algorithm “pragma” to parallels this code.

So we are add “#pragma omp for collaps(2)” to parallelizing loops . In this method we are parallelized for loop tow loops together .

3. Compare the sequential program with the multicore-program:

3.1 Compare the running time of the sequential program with the running time of the multicore-program and speed up :

3.1.1 Grammar 1 using word “baaba” long Length “YES”:




```
grammar_entry_t *pEntries = malloc(sizeof(gramm
int i = 0;
GRAMMAR_ENTRY(pEntries, i++, 'S', "AB");
GRAMMAR_ENTRY(pEntries, i++, 'S', "BC");
GRAMMAR_ENTRY(pEntries, i++, 'A', "BA");
GRAMMAR_ENTRY(pEntries, i++, 'A', "a");
GRAMMAR_ENTRY(pEntries, i++, 'B', "b");
GRAMMAR_ENTRY(pEntries, i++, 'B', "CC");
GRAMMAR_ENTRY(pEntries, i++, 'C', "AB");
GRAMMAR_ENTRY(pEntries, i++, 'C', "a");
```

Figure 0.2

Figure 0.3

We used this grammar for compare time and the word “baaba” and number of grammar 8 .

Result of sequential program :

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK-copy
Please enter the word: baaba
You entered "baaba" with a length of 5!
Word is valid: Yes

real    0m3.263s
user    0m0.000s
sys     0m0.015s
```

Figure 0.4

Result of multicore program and the threads number 4:

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK
Please enter the word: baaba
You entered "baaba" with a length of 5!
Word is valid: Yes

real    0m1.590s
user    0m0.000s
sys     0m0.046s
```

Figure 0.5

So Multicore will tack less time then Sequntial program .

3.1.2 Grammar 1 using word “ba” short Length “Yes” :

Result of sequential program :

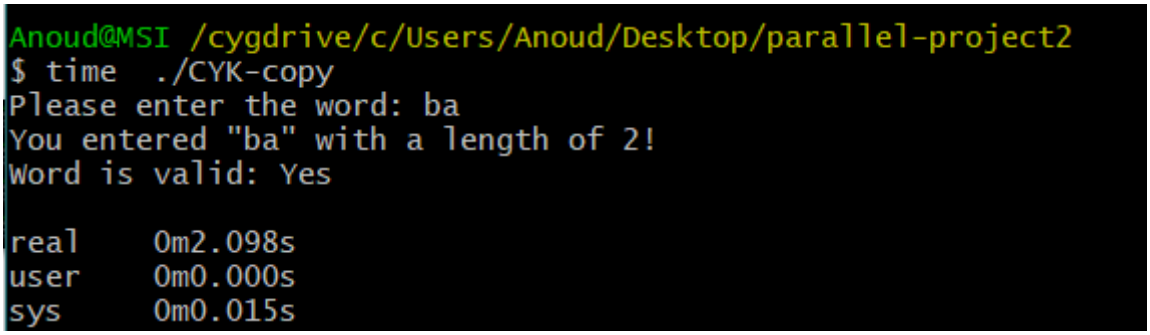


Figure 0.6

Result of multicore program and the thread number 4:

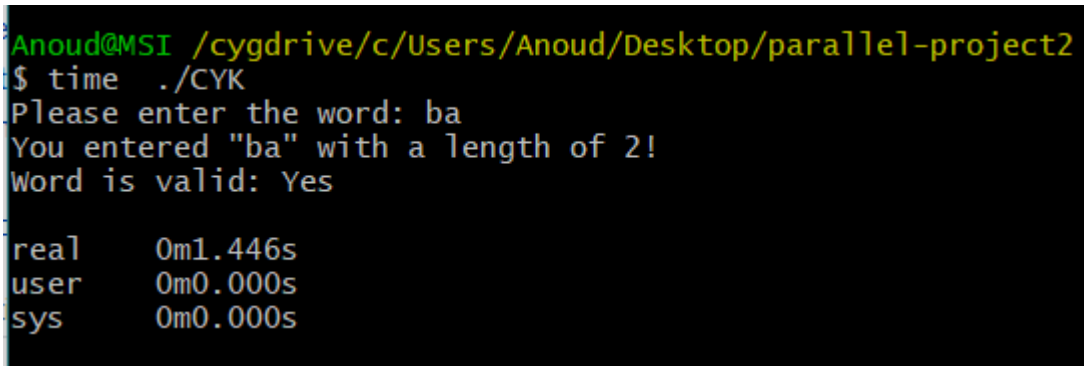


Figure 0.7

So Multicore will tack less time then Sequntial program .

3.1.3 Grammar 1 using word " aabba" Result "No" Long Length :

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK-copy
Please enter the word: aabba
You entered "aabba" with a length of 5!
Word is valid: No

real    0m2.481s
user    0m0.000s
sys     0m0.015s
```

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK
Please enter the word: aabba
You entered "aabba" with a length of 5!
Word is valid: No

real    0m1.588s
user    0m0.000s
sys     0m0.000s
```

The first result for sequential program and that give it us more time then multicore program and the threads number 4 and the string of symbols can not be derived by the rules of the grammar .

3.1.4 Grammar 1 using word " acc" Result "No" short Length :

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK-copy
Please enter the word: acc
You entered "acc" with a length of 3!
Word is valid: No

real    0m1.790s
user    0m0.000s
sys     0m0.030s
```

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK
Please enter the word: acc
You entered "acc" with a length of 3!
Word is valid: No

real    0m0.849s
user    0m0.000s
sys     0m0.016s
```

The first result for sequential program and that give it us more time then multicore program and the threads number 4 and the string of symbols can not be derived by the rules of the grammar .

3.2.1 Grammar 2 using word “aabbc” long Length “YES” .

```
const int grammar_size = 9;

grammar_entry_t *pEntries = malloc(sizeof(grammar_entry_t) * grammar_size
int i = 0;

GRAMMAR_ENTRY(pEntries, i++, 'S', "AB");
GRAMMAR_ENTRY(pEntries, i++, 'A', "CD");
GRAMMAR_ENTRY(pEntries, i++, 'A', "CF");
GRAMMAR_ENTRY(pEntries, i++, 'B', "c");
GRAMMAR_ENTRY(pEntries, i++, 'B', "EB");
GRAMMAR_ENTRY(pEntries, i++, 'C', "a");
GRAMMAR_ENTRY(pEntries, i++, 'D', "b");
GRAMMAR_ENTRY(pEntries, i++, 'E', "c");
GRAMMAR_ENTRY(pEntries, i++, 'F', "AD");
```

Figure 0.8

We will using this grammar for compare time and the word “aabbc” and number of grammar 9 and length 5 .

Result of sequential program :

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK-copy
Please enter the word: aabbc
You entered "aabbc" with a length of 5!
Word is valid: Yes

real    0m4.542s
user    0m0.000s
sys     0m0.030s
```

Figure 0.9

Result of multicore program and the thread number 4:

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK
Please enter the word: aabbc
You entered "aabbc" with a length of 5!
Word is valid: Yes

real    0m1.572s
user    0m0.015s
sys     0m0.015s
```

Figure 0.10

So Multicore will tack less time then Sequntial program .

3.2.2 Grammar 2 using word “abc” short Length “YES” .

Result of sequential program :

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK-copy
Please enter the word: abc
You entered "abc" with a length of 3!
Word is valid: Yes

real    0m2.699s
user    0m0.000s
sys     0m0.000s
```

Figure 0.11

Result of multicore program and the thread number 4:

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK
Please enter the word: abc
You entered "abc" with a length of 3!
Word is valid: Yes

real    0m1.925s
user    0m0.000s
sys     0m0.030s
```

Figure 0.12

So Multicore will tack less time then Sequntial program .

3.2.3 Grammar 2 using word “bbaba” Long Length “NO” .

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK-copy
Please enter the word: bbaba
You entered "bbaba" with a length of 5!
Word is valid: No

real    0m4.244s
user    0m0.000s
sys     0m0.000s

Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK
Please enter the word: bbaba
You entered "bbaba" with a length of 5!
Word is valid: No

real    0m2.338s
user    0m0.000s
sys     0m0.000s
```

The first result for sequential program and that give it us more time then multicore program and the threads number 4 and the string of symbols can not be derived by the rules of the grammar .

3.2.4 Grammar 2 using word “bba” Short Length “NO” .

```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK-copy
Please enter the word: bba
You entered "bba" with a length of 3!
Word is valid: No

real    0m2.268s
user    0m0.000s
sys     0m0.030s

Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ time  ./CYK
Please enter the word: bba
You entered "bba" with a length of 3!
Word is valid: No

real    0m1.492s
user    0m0.000s
sys     0m0.030s
```

The first result for sequential program and that give it us more time then multicore program and the threads number 4 . In the long length the time will tack more than the short time .

4.SpeedUp and Efficiency of the method:

We set the number of threads 4 to multicore

4.1 The Speedup and Efficiency of 3.1.1 Grammar 1

Speedup= $\frac{3.263}{1.590} = 2.052$

Efficiency = $\frac{3.263}{4 \times 1.590} = 0.513$

4.2 The Speedup and Efficiency of 3.1.2 Grammar 1

Speedup = $\frac{2.098}{1.446} = 1.450$

Efficiency = $\frac{2.098}{4 \times 1.446} = 0.362$

4.3 The Speedup and Efficiency of 3.1.3 Grammar 1

Speedup = $\frac{2.481}{1.588} = 1.5623$

Efficiency = $\frac{2.481}{4 \times 1.588} = 0.390$

4.4 The Speedup and Efficiency of 3.1.4 Grammar 1

Speedup = $\frac{1.790}{0.849} = 2.1083$

Efficiency = $\frac{1.790}{4 \times 0.849} = 0.527$

4.5 The Speedup and Efficiency of 3.2.1 Grammar 2

Speedup= $\frac{4.542}{1.572} = 2.8893$

Efficiency = $\frac{4.542}{4 \times 1.572} = 0.72232$

4.6 The Speedup and Efficiency of 3.2.2 Grammar 2

Speedup= $\frac{2.699}{1.925} = 1.402$

Efficiency = $\frac{2.699}{4 \times 1.925} = 1.40207$

4.7 The Speedup and Efficiency of 3.2.3 Grammar 2

Speedup= $\frac{4.244}{2.338} = 1.81522$

Efficiency = $\frac{4.244}{4 \times 2.338} = 0.4538$

4.8 The Speedup and Efficiency of 3.2.4 Grammar 2

Speedup= $\frac{2.268}{1.492} = 1.5201$

Efficiency = $\frac{2.268}{4 \times 1.492} = 0.3800$

5.Results:

When we have used short length and grammar number short, it will give it us short time and speed up and it will be faster.

The multicore program will be faster than sequential code in all different grammar and different length, So run time in the multicore program less than the sequential program.