Lowest Common Subsequence - Serial and Parallel Implementation

José Antunes,[1] César Alves,[1] and Mauro Machado[1]

[1]*Departamento de Física, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal*

The Longest Common Subsequence (LCS) was implemented in a serial and parallel fashion. Our serial implementation creates a matrix that is run line by line. In the parallel version each anti-diagonal was calculated in sequence with each value being calculated in parallel.

I. INTRODUCTION

With this work we wish to study the difference in execution time between the serial and parallel implementation of the Longest Common Subsequence (LCS) algorithm. This is a relevant algorithm in the bioinformatics field where a comparison between two DNA sequences is needed.

Problems appear when sequences become very large and the serial implementation of the code starts to have a very long execution time.

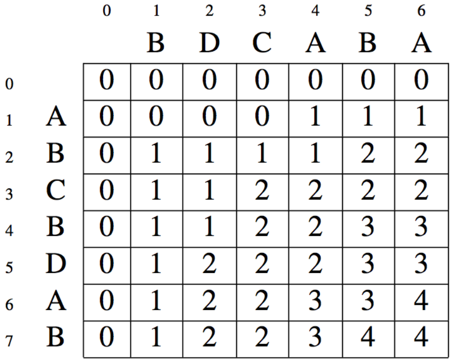
Processors have evolved more in the direction of having multiple cores over faster speeds and, as such, the trivial answer to these longer execution times is to have a parallel approach to the problem.

The LCS algorithm takes two sequences, X and Y, of different lengths and will use a matrix to compare them. The matrix will be filled with integer values depending on how each element and the previous ones compare to each other.

The basic structure of the algorithm is the following:

$$c(i,j) = \begin{cases} 0 & \text{if } i,j = 0 \\ c(i-1,j-1)+1 & \text{if } i,j > 0 \; x_i = y_j \\ max(c(i-1,j); c(i,j-1)) & \text{if } i,j > 0 \; x_i \neq y_j \end{cases}$$

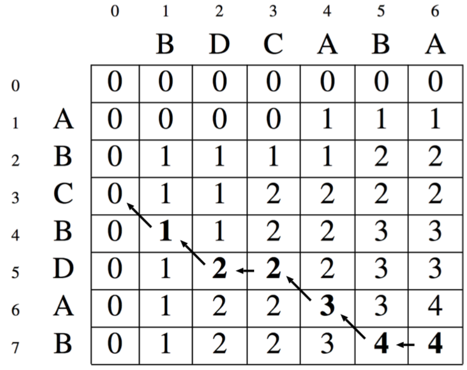
and this creates a matrix which will give the longest subsequence.



		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Figura 1: Filled matrix after LCS algorithm. [1].

With this matrix it is only needed that we start from the highest number and walk to the left. We go to the upper diagonal if there is a mismatch between the numbers and left only if they match.



		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	**1**	1	2	2	3	3
5	D	0	1	**2**	**2**	2	3	3
6	A	0	1	2	2	**3**	3	4
7	B	0	1	2	2	3	**4**	**4**

Figura 2: Representation of the LCS walk. [1].

We can achieve the LCS by selecting the letters corresponding to the indexes that make our walk move in the upper diagonal direction.

II. METHODS

A. Serial Implementation

The implemented serial version was straightforward to implemente. The matrix was calculated element by element, line by line, taking advantage of the cache's Locality Principal, both spacial and temporal.

Algorithm 1 Serial LCS implementation

```
for $i = 0$ to $N$ do
    for $j = 0$ to $M$ do
        Compute $C(i,j)$
    end for
end for
```

Albeit being the easiest solution to implement, this is not the best method when it comes to transforming the code into an effective paralleled version.

B. Parallel Implementation

The thought process for any parallelization revolves around computing as many simultaneous values as possible. The rule for being able to make these computations is that they don't depend on each other.

In the LCS algorithm, the dependencies for each cell are well defined in the algorithm.

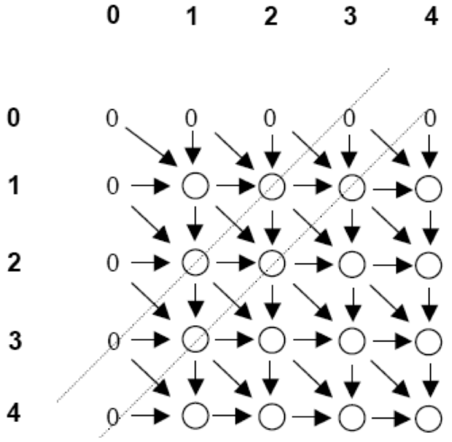


Figura 3: Data dependencies of each $C(i, j)$. [1].

It is trivial to see that each cell $C(i, j)$ depends on either $C(i, j-1)$ and $C(i-1, j)$ or $C(i-1, j-1)$. Furthermore, it is easy to say that each element in an anti-diagonal can be computed in parallel, since there are no dependencies.

As such the implemented algorithm has the form:

Algorithm 2 Parallel LCS implementation

```
for $anti - diagonal = 0$ to $N + M$ do
  for $i = 0$ to $N$ do
    j = anti-diagonal - i
    if i¿0 —— j¿0 then
      Compute $C(i, j)$
    end if
  end for
end for
```

This implementation raises concerns over the cache usage, which is not being used as efficiently as it could be.

III. RESULTS

METER AQUI AS TABELAS COM OS VALORES

Data	Serial	Parallel
ex10.15.in	5	6
ex150.200.in	5	6
ex3k.8k.in	5	6
ex18k.17k.in	5	6
ex48k.30k.in	8	9

IV. DISCUSSION

DISCUTIR AQUI AS TABELAS

V. CONCLUSIONS

CONCLUIR CENAS

VI. AKNOWLEDGEMENTS

We would like thank Professor José Monteiro and Professor José Costa, from IST-UL, for their time and help, for teaching us and shedding light over our doubts.

[1] K. Cantrell, "A Study of the Plasma Tweeter", B.Sc. Thesis, Ball State University (2011)

[2] M. Hopkins and T. Houlhan, "The Plasma Speaker: Construction and Characterization of both Full-bridge and Single-ended driving circuits", Project Report, University of Illinois at Urbana-Champaign (2012)

[3] D. Severinsen and G. Sen Gupta, "Design and Evaluation of Electronic Circuit for Plasma Speaker", Proceedings of the World Congress on Engineering 2013 Vol II (2013)

[4] L. Wayne Sieck, John T. Herron, and David S. Green, Plasma Chem., Plasma P., Vol. 20, No. 2, 2000

[5] John T. Herron and David S. Green, Plasma Chem., Plasma P., Vol. 21, No. 3, 2001

[6] K.H. Becker, U. Kogelschatz, K.H. Schoenbach, R.J. Barker, "Non-Equilibrium Air Plasmas at Atmospheric Pressure", p. 130, Institute of Physics Publishing, Bristol, UK (2005)

[7] Wolfram Research, Inc., Mathematica, Version 9.0, Champaign, IL (2012).