

A Coarse-Grained Multicomputer Algorithm for the Longest Repeated Suffix Ending at Each Point in a Word

Thierry Garcia and David Semé

LaRIA : Laboratoire de Recherche en Informatique d'Amiens
Université de Picardie Jules Verne,
CURI, 5, rue du Moulin Neuf, 80000 Amiens, France
{garcia,seme}@laria.u-picardie.fr

Abstract. The paper presents a Coarse-Grained Multicomputer algorithm that solves the problem of finding the longest repeated suffix ending at each point in a word. This algorithm can be implemented in the CGM with P processors in $O(\frac{N^2}{P})$ in time and $O(P)$ communication steps. It is the first CGM algorithm for this problem. We present also experimental results showing that the CGM algorithm is very efficient.

1 Introduction

The longest repeated suffix ending at each point in a word (LRSE for short) is a good illustration of dynamic programming. There have been a large number of studies on the problem of finding repetitions (or repeats) in a given word A of length N (see [14]).

Repetitions play a crucial role in many topics of combinatorics of word, formal language theory, term rewriting, data compression, musical analysis, and computational molecular biology. In this paper we are interested in finding the Longest Repeated Suffix Ending at each position in A . Lefebvre and Lecroq [13] used the factor oracle of A to get an on-line linear computation of a good approximation of these values. Lecroq, Myoupo and Semé [12] gave a sequential and parallel algorithms, which are the first ones exhibited to compute the exact values of the longest repeated suffix ending at each position in a word.

In recent years several efforts have been made to define models of parallel computation that are more realistic than the classical PRAM models. In contrast of the PRAM, these new models are coarse grained, i.e. they assume that the number of processors P and the size of the input N of an algorithm are orders of magnitudes apart, $P \ll N$. By the precedent assumption these models map much better on existing architectures where in general the number of processors is at most some thousands and the size of the data that are to be handled goes into millions and billions. This branch of research got its kick-off with Valiant [15] introducing the so-called Bulk Synchronous Parallel (BSP) machine, and was refined in different directions for example by Culler et al. [3], LogP, and Dehne et al. [5], CGM extensively studied in [1, 2, 4, 6–9, 11]. CGM seems to be the best

suitable for a design of algorithms that are not too dependent on an individual architecture.

We summarize the assumptions of this model :

- all algorithms perform in so-called supersteps, that consist of one phase of interprocessor communication and one phase of local computation,
- all processors have the same size $M=O(\frac{N}{P})$ of memory ($M > P$),
- the communication network between the processors can be arbitrary.

The goal when designing an algorithm in this model is to keep the individual workload, time for communication and idle time of each processor within $\frac{T}{s(P)}$, where T is the runtime of the best sequential algorithm on the same data and $s(P)$, the speedup, is a function that should be as close to P as possible. To be able to do so, it is considered as a good idea the fact of keeping the number of supersteps of such an algorithm as low as possible, preferably $O(M)$. As a legacy from the PRAM model it is usually assumed that the number of supersteps should be polylogarithmic in P , but there seems to be no real world rationale for that. In fact, algorithms that simply ensure a number of supersteps that are a function of P (and not of N) perform quite well in practice, see Goudreau et al. [10].

In this paper we describe a CGM (Coarse Grained Multicomputers) solution to the Longest Repeated Suffix Ending at Each Point in a Word (LRSE) problem. In this model, we use P processors for solving the LRSE problem for a word A of length N on an alphabet Σ .

The paper is organized as follows. In the Section 2, we present background needed for the LRSE problem. The CGM solution of the problem is described in Section 3. Section 4 presents some experimental results and the conclusion ends the paper.

2 Background

2.1 Basic definitions and notations

Definition 1. A string or word is a sequence of zero or more symbols from an alphabet Σ ; the string with zero symbols is denoted by ε .

Definition 2. The set of all strings over the alphabet Σ is denoted by Σ^* .

Definition 3. A string A of length N ($|A| = N$) is represented by $A[0] \dots A[N-1]$, where $A[i] \in \Sigma$ for $0 \leq i \leq N-1$.

Definition 4. A string W is a substring (or sub-word) of A if $A = UWV$ for $U, V \in \Sigma^*$; we equivalently say that the string W occurs at position $|U| + 1$ in the string A .

Definition 5. The position $|U| + 1$ is said to be the starting position of W in A and the position $|U| + |W|$ the ending position of W in A . We denote $W = A[|U| + 1 \dots |U| + |W|]$.

Definition 6. A string W is a prefix of A if $A = WU$ for $U \in \Sigma^*$. Similarly, W is a suffix of A if $A = UW$ for $U \in \Sigma^*$.

2.2 Dynamic programming approach

Let $A = A[0 \dots N - 1]$ be a word of length N on an alphabet Σ . We define a two-dimensional table T as follows:

$$T[i, j] = \max\{\ell \mid A[i - \ell + 1 \dots i] = A[j - \ell + 1 \dots j]\}$$

for $0 \leq j \leq N - 1$ and $0 \leq i < j$. That is, $T[i, j]$ is the length of the longest common suffix of $A[0 \dots i]$ and $A[0 \dots j]$. Figure 1 gives an example of table T .

j	0	1	2	3	4	5	6	7	8	9	10	11
i	a	b	b	c	a	b	c	d	a	b	c	d
-1	ε	0	0	0	0	0	0	0	0	0	0	0
0	a	0	0	0	1	0	0	0	1	0	0	0
1	b	1	0	0	2	0	0	0	2	0	0	0
2	b	0	0	1	0	0	0	1	0	0	0	0
3	c	0	0	2	0	0	0	2	0	0	0	0
4	a	0	0	0	1	0	0	0	0	0	0	0
5	b	0	0	0	2	0	0	0	0	0	0	0
6	c	0	0	0	3	0	0	0	0	0	0	0
7	d	0	0	0	0	4	0	0	0	0	0	0
8	a	0	0	0	0	0	0	0	0	0	0	0
9	b	0	0	0	0	0	0	0	0	0	0	0
10	d	0	0	0	0	0	0	0	0	0	0	0

Fig. 1. Values of the table T for $A = abbcabcdabcd$.

The following lemma shows how to compute the values of the table T .

Lemma 1. The values of the table T can be computed using the following recurrence formula:

$$T[-1, j] = 0 \text{ for } 0 \leq j \leq N - 1 \tag{1}$$

and

$$T[i, j] = \begin{cases} T[i - 1, j - 1] + 1 & \text{if } A[i] = A[j] \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

for $0 \leq j \leq N - 1$ and $0 \leq i < j$.

Proof. See [12].

$A[j]$	a	b	b	c	a	b	c	d	a	b	c	d
$S[j]$	0	0	1	0	1	2	2	0	1	2	3	4

Fig. 2. Values of the table S for the example of figure 1.

We now define a one-dimensional table S as follows:

$$S[j] = \max\{|W| \mid W \text{ is a suffix of } A[0 \dots j] \text{ and } W \text{ is a factor of } A[0 \dots j-1]\}$$

for $0 \leq j \leq N-1$. That is, $S[j]$ is the length of the longest suffix of $A[0 \dots j]$ that occurs at least twice in $A[0 \dots j]$. Figure 2 gives an example of table S .

The following lemma shows how to compute the values of the table S .

Lemma 2. *The values of the table S can be computed using table T by the following formula:*

$$S[j] = \max\{T[i, j] \mid 0 \leq i < j\} \quad (3)$$

for $0 \leq j \leq N-1$.

Proof. See [12].

So, we have an immediate algorithm to compute the values of the table S which is stated in the next corollary.

Corollary 1. *There exists an algorithm which computes the values of the tables T and S by dynamic programming. Its running time is $\Theta(N^2)$ and its space requirement is $O(N)$.*

Proof. See [12].

3 Coarse Grained Multicomputer Algorithm

Our CGM algorithm is based on the dynamic programming approach described before. The idea is to cut the table T in P columns of $\frac{N}{P}$ elements. Figure 3 gives an example of the cut-table T' .

Cut-table T' contains two kind of parts : triangular sub-tables (TT) and square sub-tables (ST). Lemma 1 can be applied on triangular sub-tables because of the comparison of a substring of A called $A1$ with itself. The following lemma shows how to compute the values of a square sub-table ST from the comparison between $A1$ and $A2$, two different substrings of A .

Lemma 3. *The values of the sub-table ST can be computed using the following recurrence formula:*

$$ST[-1, j] = 0 \text{ for } -1 \leq j \leq N-1 \quad (4)$$

$$ST[i, -1] = 0 \text{ for } 0 \leq i \leq N-1 \quad (5)$$

	j	0	1	2	3	4	5	6	7	8	9	10	11
i	a	b	b	c	a	b	c	d	a	b	c	d	
0	a	0	0	0	0	1	0	0	0	1	0	0	0
1	b	0	1	0	0	0	2	0	0	0	2	0	0
2	b	0	0	0	0	1	0	0	0	0	1	0	0
3	c	0	0	0	2	0	0	0	2	0	0	2	0
4	a					0	0	0	0	1	0	0	0
5	b					0	0	0	0	0	2	0	0
6	c					0	0	0	0	0	3	0	0
7	d					0	0	0	0	0	0	4	0
8	a									0	0	0	0
9	b									0	0	0	0
10	c									0	0	0	0
11	d											0	0

Fig. 3. Cut-table T' for $A = abbcabcdabcd$

and

$$ST[i, j] = \begin{cases} ST[i - 1, j - 1] + 1 & \text{if } A1[i] = A2[j] \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

for $0 \leq j \leq N - 1$ and $0 \leq i < N - 1$.

Proof. We prove the formula by recurrence on j . It is clearly true when $j = 0$ since the only value which is defined in the table ST with $j = 0$ is $ST[-1, 0]$ which is initialized to 0 by equation 4. Now let us assume that the property holds up to value $j - 1$ and we will show that it also holds for j .

For $0 \leq i < N - 1$, if $A1[i] \neq A2[j]$ then the longest common suffix of $A1[0 \dots i]$ and $A2[0 \dots j]$ is the empty word, which is computed by the second case of equation 6. If $A1[i] = A2[j]$ then the length of the longest common suffix of $A1[0 \dots i]$ and $A2[0 \dots j]$ is equal to the length of the longest common suffix of $A1[0 \dots i - 1]$ and $A2[0 \dots j - 1]$ plus 1, that is, by the recurrence hypothesis $ST[i - 1, j - 1] + 1$ which is computed by the first case of equation 6. This ends the proof.

Each triangular table must be initialized by

$$TT[-1, j] = L[j] \text{ for } 0 \leq j \leq N - 1 \quad (7)$$

and

each square table must be initialized by

$$ST[-1, j] = L[j] \text{ for } -1 \leq j \leq N - 1 \quad (8)$$

$$ST[i, -1] = C[i] \text{ for } 0 \leq i \leq N - 1 \quad (9)$$

where C and L are respectively the last column and the last row of the previous part table of the cut table.

The following section describes two sequential algorithm based on the different lemmas, which are used in our CGM algorithm.

3.1 Sequential algorithm

Algorithm 1 computes the triangular cut-table TT , column by column and construct the table S by using the lemmas 1 and 2. At the end, the algorithm keeps the last column in the table C .

```

Algorithm 1
for (i=2) to (i=N)
  for (j=1) to (j=i-1)
    TEMPC[j]=0
    if (A[i]=A[j]) then
      if (j=1) then TEMPC[j]=L[i-1]+1 else TEMPC[j]=C[j-1]+1 endif
      if (TEMPC[j]>S[i]) then S[i]=TEMPC[j] endif
    endif
  endfor
  for (j=1) to (j=i-1) C[j]=TEMPC[j] endfor
endfor

```

Algorithm 2 computes the square cut-table ST , column by column and construct the table S by using the lemmas 1 and 2. At the end, the algorithm keeps the last row in the table L and the last column in the table C . The first element of the table L is the last element of the previous column C .

```

Algorithm 2
TEMPL=L
L[0]=C[N]
for (i=1) to (i=N)
  for (j=1) to (j=N)
    TEMPC[j]=0
    if (A[i]=A[j]) then
      if (j=1) then TEMPC[j]=TEMPL[i-1]+1 else TEMPC[j]=C[j-1]+1 endif
    dif
      if (TEMPC[j]>S[i]) then S[i]=TEMPC[j] endif
    endif
  endfor
  L[i]=TEMPC[N]
  for (j=1) to (j=N) C[j]=TEMPC[j] endfor
endfor

```

3.2 CGM algorithm

Algorithm LRSE_CGM presents the CGM solution of the LRSE problem. Each processor num ($1 \leq num \leq P$) has the num -th partition of $\frac{N}{P}$ elements of the input sequence A . The following CGM algorithm presents the program of each processor num .

That CGM algorithm uses sequential Algorithms 1 and 2 described before as local computation parts. Two functions are used for communication rounds :

send(X,num) : a vector X is sent to the processor num

receive(Y,num) : a vector Y is received from the processor num

The next section explains the communication rounds.

```

Algorithm LRSE_CGM
for (k=1) to (k≤num)
  if (k = num) then
    Local computation using Algorithm 1
    if (num < P) then
      send (A,num+1)
      send (C,num+1)
    endif
  else
    receive (A,num-1)
    receive (C,num-1)
    Local computation using Algorithm 2
    if (num < P) then
      send (A,num+1)
      send (C,num+1)
    endif
  endif
endif
endfor

```

3.3 Communication rounds

The communication rounds are described in Figure 4. The complexity for the communication is $O(2P - 3)$.

Proof. The Figure 4 shows the processor 0 sending one message, the processor 1 sending two messages, ... the processor $P - 1$ sending P messages. So, there are $1 + 2 + \dots + P = \frac{(P-1)^2 + (P-1)}{2}$ messages sent. But, when the processor i has sent $i + 1$ messages, the processor $i + 1$ has sent $i - 2$ messages. So, the communication rounds number is equal to $\frac{(P-1)^2 + (P-1)}{2} - \frac{((P-1)-2)^2 + ((P-1)-2)}{2} = 2P - 3$. This ends the proof.

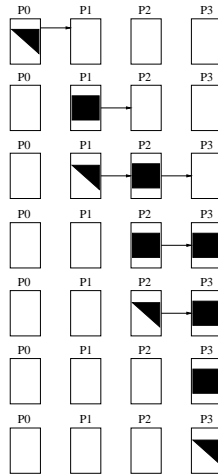


Fig. 4. Communication round (for 4 processors).

3.4 Complexity

The complexity for the communication rounds is $O(2P - 3)$. The complexity of the CGM algorithm is $O(2N^2 - \frac{N^2}{P})$.

Proof. The complexity of sequential algorithms is $O(\frac{N^2}{P})$. The complexity for the communication is $O(2P - 3)$. When the processor before the last has finished the execution of its algorithms and communications, the last processor must yet execute locally Algorithm 1 and Algorithm 2 without communication (Figure 4). So, the total complexity is $O(\frac{N^2}{P} \times ((2P - 3) + 2)) = 2N^2 - \frac{N^2}{P}$. This ends the proof.

4 Experimental Results

We have implemented these programs in C language using MPI communication library and tested them on a multiprocessor Celeron 466Mhz platform running LINUX. The communication between processors is performed through an Ethernet switch.

Table 1 and Table 2 present total running times and communication times (in seconds), respectively, for each configuration of 1, 2, 4, 8 and 16 processors. The communication times correspond to send and receive of $\frac{N}{P}$ characters by a processor to another using the communication rounds described before. Here, we have $N=2^k$ and k is an integer such that $12 \leq k \leq 20$.

Note that these communication times are much more lower than the total running times. This means that the computation times are much more greater than

communication times.

Table 2 shows that communication times increase when the number of processors used is greater. But Table 1 shows that total running times, for a fixed number of processors, decrease when the number of processors increase. This leads that there is more important to improve local computations than communication rounds.

N	P=1	P=2	P=4	P=8	P=16
4096	1.46	1.09	1.01	0.76	0.61
8192	5.83	4.37	4.02	2.71	1.78
16384	24.44	18.33	16.06	10.78	6.16
32768	117.56	88.17	69.06	42.98	24.49
65536	536.92	402.69	302.82	179.95	98.70
131072	2223.26	1667.45	1341.77	796.86	401.44
262144	8964.29	6723.22	5408.22	3398.74	1791.30
524288	35840.29	26880.22	21642.46	13694.10	7606.65
1048576	143493.65	107620.238	86981.87	54829.91	30717.08

Table 1. Total running times (in seconds) for each configuration of 1, 2, 4, 8 and 16 processors respectively.

N	P=2	P=4	P=8	P=16
4096	0.01	0.12	0.27	0.08
8192	0.03	0.13	0.18	0.23
16384	0.04	0.07	0.07	0.19
32768	0.04	0.07	0.31	0.13
65536	0.03	0.11	0.83	0.44
131072	0.05	0.21	3.45	4.01
262144	0.11	0.58	5.23	8.97
524288	0.46	1.05	11.18	10.23
1048576	0.42	2.35	22.09	17.71

Table 2. Communication times (in seconds) for each configuration of 2, 4, 8 and 16 processors respectively.

5 Concluding remarks

We presented a Coarse-Grained Multicomputer algorithm that solves the Longest Repeated Suffix Ending at Each Point in a Word Problem. This algorithm can be implemented in the CGM model with P processors in $O(\frac{N^2}{P})$ in time and $O(P)$ communication rounds. It is the first CGM algorithm for this problem.

The paper presents also experimental results showing that the CGM algorithm is very efficient, for great sequence A , when we increase the number of processors. We should now implement this algorithm on another platform in order to show its portability.

Our goal was to develop a CGM algorithm from a systolic solution presented in [12] in order to show that a linear systolic solution can be implemented in the CGM model with the same work. It will be interesting to develop another CGM algorithm for the same problem using the optimal sequential solution.

References

1. P. Bose, A. Chan, F. Dehne, and M. Latzel. Coarse grained parallel maximum matching in convex bipartite graph. *Proc. 13th International Parallel Processing Symposium (IPPS'99)*, pages 125–129, 1999.
2. A. Chan and F. Dehne. A note on coarse grained parallel integer sorting. *Parallel Processing Letters*, 9(4):533–538, 1999.
3. D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. Von Eicken. LogP: towards a realistic model of parallel computation. *4-th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming*, pages 1–12, 1996.
4. F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. Khokhar. A randomized parallel 3d convex hull algorithm for coarse grained multicomputers. *Proc. 7th ACM Symp. on Parallel Algorithms and Architectures*, pages 27–33, 1995.
5. F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. *International Journal on Computational Geometry*, 6(3):379–400, 1996.
6. M. Diallo, A. Ferreira, A. Rau-Chaplin, and S. Ubeda. Scalable 2d convex hull and triangulation algorithms for coarse grained multicomputers. *Journal of Parallel and Distributed Computing*, 56(1):47–70, 1999.
7. A. Ferreira and N. Schabanel. A randomized bsp/cgm algorithm for the maximal independent set problem. *Parallel Processing Letters*, 9(3):411–422, 1999.
8. T. Garcia, J.F. Myoupo, and D. Semé. A work-optimal cgm algorithm for the longest increasing subsequence problem. *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'01)*, 2001.
9. T. Garcia, J.F. Myoupo, and D. Semé. A coarse-grained multicomputer algorithm for the longest common subsequence problem. *11-th Euromicro Conference on Parallel Distributed and Network based Processing (PDP'03)*, 2003.
10. M. Goudreau, S. Rao K. Lang, T. Suel, and T. Tsantilas. Towards efficiency and portability: Programming with the bsp model. 8th Annual ACM Symp. on Parallel Algorithms and Architectures (SPAA'96), pages 1–12, 1996.
11. S.R. Kim and K. Park. Fully scalable fault-tolerant simulations for bsp and cgm. *Journal of Parallel and Distributed Computing*, 60:1531–1560, 2000.
12. T. Lecroq, J.-F. Myoupo, and D. Semé. Exact computations of the longest repeated suffix ending at each point in a word. In R. Gantenbein and S. Shin, editors, *Proceedings of the 17th International Conference on Computers and Their Applications*, pages 18–21, 2002.
13. A. Lefebvre and T. Lecroq. Computing repeated factors with a factor oracle. In L. Brankovic and J. Ryan, editors, *Proceedings of the 11th Australasian Workshop On Combinatorial Algorithms*, pages 145–158, 2000.
14. W. F. Smyth. Repetitive perhaps, but not boring. *Theor. Comput. Sci.*, 249(5):343–355, 2000.
15. L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.