

A Coarse-Grained Multicomputer Algorithm for the Longest Common Subsequence Problem

Thierry Garcia, Jean-Frédéric Myoupo 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
E-mail: {garcia,myoupo,seme}@laria.u-picardie.fr

Abstract

The paper presents a Coarse-Grained Multicomputer algorithm that solves the Longest Common Subsequence Problem. 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 Common Subsequence Problem (LCS for short) is a good illustration of dynamic programming and has interested many scientists. Finding an LCS can be interesting in many domains such as genetic engineering, data compression, editing error correction and syntactic pattern recognition and there is strong motivation for designing efficient algorithms [25].

Given two input strings, the LCS problem consists in finding a subsequence of both strings which is of maximal possible length, say L . Computing L only solves the problem of determining the length of an LCS (LLCS problem). Given a string A over an alphabet Σ (any finite set of symbols), a *subsequence* of A is any string C that can be obtained from A by deleting zero or some symbols (not necessarily consecutive). The *Longest Common Subsequence* (LCS) problem for two input strings $A = A_1A_2\cdots A_N$ and $B = B_1B_2\cdots B_M$ ($M \leq N$) consists in finding another string $C = C_1C_2\cdots C_L$ such that C is a subsequence of both A and B , and is a maximal possible length.

The LLCS and LCS problems had been extensively studied in the literature on sequential algorithms, among

others [1, 3, 10, 19, 21, 15, 16, 17, 22], but the inherent $O(NM)$ time complexity of the LCS problem can only be improved under restrictive assumptions. Several other sequential algorithms are recalled in [11]. In recent years, exploiting the parallelism of this problem has attracted many research interests, see, for example, [2, 20] on PRAM models. In [2] this problem was solved in time $O(\log M \log N)$ with $O(MN/\log M)$ processors in CREW PRAM model (with $M < N$). The CRCW bound is $O(\log N(\log \log M)^2)$ time with $O(MN/\log \log M)$ processors. In [20] the problem is solved in $O(\log^2 M + \log N)$ time with $MN/\log M$ processors on the CREW PRAM model.

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 [26] introducing the so-called Bulk Synchronous Parallel (BSP) machine, and was refined in different directions for example by Culler et al. [6], LogP, and Dehne et al. [7], CGM extensively studied in [4, 5, 8, 9, 12, 13, 18].

CGM seems to be the best suited 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. [14].

In this paper we describe a CGM (Coarse Grained Multicomputers) solution to the Longest Common Subsequence (LCS) problem. In this model, we use P processors for solving the LCS problem between two strings A and B with the same length N . In the case where the lengths of the two strings are different, we complete the smaller string with empty character.

The paper is organized as follows. In the Section 2, we present the LCS 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 The LCS problem

2.1 Basic definitions and notations

Definition 1 Let Σ be any finite set of symbols called an alphabet. A string over Σ is any finite sequence of elements from Σ . Σ^* is the set of all strings over Σ , including the empty one ϵ . Σ^+ denotes $\Sigma^* - \{\epsilon\}$. For any string A , $|A|$ denotes the length of A (the number of symbols in A). Note $|\epsilon| = 0$.

Definition 2 A string $A \in \Sigma^+$ is fully specified by writing $A = A[1] \dots A[n]$, where $A[i] \in \Sigma$ ($1 \leq i \leq n$). String $A' = A[i_1] \dots A[i_k]$ ($1 \leq k \leq n$), where $1 \leq i_1 < \dots < i_k \leq n$, is called a subsequence of A . ϵ is also a subsequence of A .

Definition 3 Let A be a string specified by $A = A[1] \dots A[n]$.

$A[i : j]$ represents a subsequence of A such that $A[i : j] = A[i] \dots A[j]$ ($1 \leq i \leq j \leq n$).

Definition 4 Let A and B be two strings over Σ . String U is a common subsequence of A and B if it is a subsequence of both A and B .

Definition 5 U is a longest common subsequence (LCS) of A and B if U is a common subsequence of A and B of maximal possible length.

We recall that $llcs(i, j)$ denotes the length of the Longest Common Subsequence of $A[1 : i]$ and $B[1 : j]$.

2.2 Dynamic programming approach

Proposition 1 For $1 \leq i \leq n$ and $1 \leq j \leq m$:

$$llcs(i, 0) = llcs(0, j) = llcs(0, 0) = 0$$

$$llcs(i, j) = \begin{cases} llcs(i-1, j-1) + 1, & \text{if } A[i]=B[j] \\ \max(llcs(i-1, j), llcs(i, j-1)), & \text{otherwise} \end{cases}$$

Proof: see [23]. □

Proposition 2 For $1 \leq j \leq m$:

$$LLCS(j) = \max \{ llcs(i, j) \mid \forall i: 1 \leq i \leq n \}$$

Proof: The row j of the table $llcs$ contains the lengths of all the longest common subsequences of $B[0 \dots j]$. Then, the largest of these values gives the correct value for $LLCS(j)$. This ends the proof. □

	b	c	a	c	b	d	a	b	c
	0	0	0	0	0	0	0	0	0
a	0	0	0	1	1	1	1	1	1
c	0	0	1	1	2	2	2	2	2
b	0	1	1	1	2	3	3	3	3
a	0	1	1	2	2	3	3	4	4
d	0	1	1	2	2	3	4	4	4
c	0	1	2	2	3	3	4	4	5
d	0	1	2	2	3	3	4	4	5
a	0	1	2	3	3	3	4	5	5
b	0	1	2	3	3	4	4	5	6

Table 1. Example for input string A=acbadcdab and B=bcacbdabc.

Proposition 3 For $1 \leq j \leq m$ and $1 \leq i \leq n$:

$$PRED(j) = \begin{cases} \min\{k \mid \exists k < i, l < j : LLC S[j] = llcs(k, l)\}, & \text{if } LLC S[j] \neq 0 \\ -1, & \text{otherwise} \end{cases}$$

$B[j]$	b	c	a	c	b	d	a	b	c
$LLCS[j]$	1	2	3	4	5	5	6	7	7
$PRED[j]$	-1	0	1	2	3	4	5	6	7

Table 2. Example for input string $A=acbadcdab$ and $B=bcacbdabc$.

$PRED[j]$ is the index of the character of B which precedes $B[j]$ in the longest common subsequence containing it.

Proof: Proposition 2 leads that there exists, $\forall j, \exists k$ and l such that:

$$LLCS[j] = llcs(k, l) + 1 \quad \text{if } LLCS[j] \neq 0 \quad (1)$$

Among all the k that verify (1) we keep the minimum. Then, this value k is the index of the character of B which precedes $B[j]$ in the longest common subsequence containing it. \square

Theorem 6 Let $B[k]$ be the last character of the longest common subsequence.

The LCS is made up by $B[PRED^{p-1}[k]] \dots B[PRED^2[k]]B[PRED[k]]B[k]$ (with p is the length of the LCS and $PRED^n[k] = \underbrace{PRED[PRED[\dots PRED[k]\dots]}_{n \text{ termes}}$).

Proof: We have seen in the Proposition 3 that $PRED[j]$ is the index of the character of B which precedes $B[j]$ in the longest common subsequence containing it.

Starting from $B[k]$, be the last character of the longest common subsequence (i.e. k is the minimum index such that $LLCS[k]=\max LLCS[j] - 1 \leq j \leq m$), the character of B which precedes $B[k]$ in the longest common subsequence is $B[PRED[k]]$. In applying this operation recursively, we have:

$LCS = B[PRED^{p-1}[k]] \dots B[PRED^2[k]]B[PRED[k]]B[k]$ with p is the length of the longest common subsequence and $PRED^n[k] = \underbrace{PRED[PRED[\dots PRED[k]\dots]}_{n \text{ termes}}$. \square

3 Coarse Grained Multicomputer Algorithm

Our CGM algorithm is based on three sequential algorithms. Each sequential algorithm represents a phase of the CGM algorithm.

3.1 Sequential algorithms

Algorithm 1 computes for each character of B the length of the longest common subsequence containing it and the

index of its predecessor in that LCS. For each j , the computation of $LLCS[j]$ (i.e. the length of the longest common subsequence containing $B[j]$), is based on the proposition 2. For that, we should use the proposition 1 that permit to calculate $llcs(i, j)$ (with $1 \leq i \leq n$). Thus, the variables $LL[j]$, $X[j]$, and $Y[j]$ are used to store $llcs(i-1, j)$, $llcs(i-1, j-1)$, and $llcs(i, j-1)$ respectively. The computation of the predecessor of $B[j]$ in the LCS containing $B[j]$ is directly issue from the proposition 3. The result of this operation is in $PRED[j]$.

Algorithm 1

```

for (i=1) to (i=N)
  for (j=1) to (j=N)
    TEMP=LL[j]
    if (Y[i]>LL[j]) then
      if ((B[j]=A[i]) and (LL[j]<1+X[i])) then
        LLCS[j]=1+X[i]
        PRED[j]=LLP[i]
      endif
      LL[j]=Y[i]
      LLP[i]=j
    else
      if ((B[j]=A[i]) and (LL[j]<1+X[i])) then
        LL[j]=1+X[i]
        LLCS[j]=LL[j]
        PRED[j]=LLP[i]
      else
        if ((LLCS[j]=LL[j]) and (LLCS[j]≠0)) then
          LLP[i]=j
        endif
      endif
    endif
    Y[i]=LL[j]
    X[i]=TEMP
  endfor
endfor

```

Algorithm 2 determines the length of the longest common subsequence of A and B stored in the variable Max . The variable ind contains the position in B of the last character of the LCS. It is easy to compute ind and Max since we have the length of the longest common subsequence ending in each character of B .

Algorithm 2

```

Max=LLCS[1]
for (i=2) to (i=n)
  if LLCS[i]>Max then
    Max=LLCS[i]
    ind=i
  endif
endfor

```

Algorithm 3 presents the extraction of a longest common subsequence of A and B from Theorem 6 and works as follows: The construction of the LCS begins by the element $B[ind]$ such that $LLCS[ind]$ is equal to Max . $PRED[ind]$ represents the position of the previous element of $B[ind]$ in the LCS. Then, we consider the previous element of $B[ind]$ and we decrease the variable Max . We repeat these operations while Max is not equal to 0.

```

Algorithm 3
do
  LCS[Max]=B[ind]
  ind=PRED[ind]
  Max=Max-1
while (Max≠0)
  
```

3.2 CGM algorithm

Algorithm LCS_CGM presents the CGM solution of the LCS problem. Each processor num ($1 \leq num \leq P$) have the num -th partition of $\frac{N}{P}$ elements of the input sequences A and B . The following CGM algorithm presents the program of each processor num .

That CGM algorithm uses sequential algorithm 1, 2 and 3 described before as local computation parts. Figures 1, 2, and 3 described communication rounds 1, 2, and 3 respectively used in the CGM algorithm.

```

Algorithm LCS_CGM
for (k=1) to (k=P)
  if k ≤ num
    Local computation using Algorithm 1
    Communication round 1
  endif
endfor
Local computation using Algorithm 2
Communication round 2
Local computation using Algorithm 3
Communication round 3
  
```

3.3 Complexity

The complexity with P processors is $O(\frac{N^2}{P})$ in time and $O(P)$ communication steps. We have a time complexity of $O(P \times (\frac{N}{P})^2)$ with $O(P)$ for Algorithm 1, $O(\frac{N}{P})$ with $O(1)$ communication steps for Algorithm 2, $O(\frac{N}{P})$ with $O(1)$ communication steps for Algorithm 3.

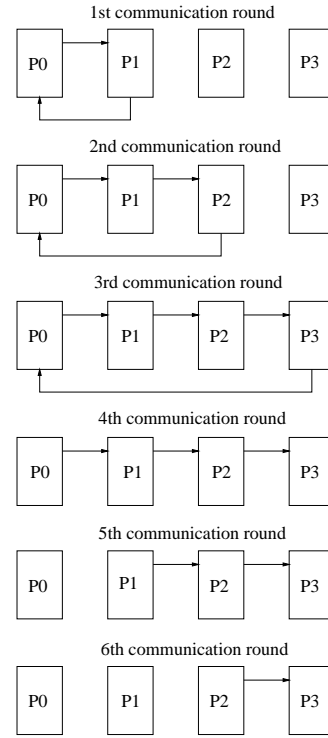


Figure 1. Communication round 1 (for 4 processors).

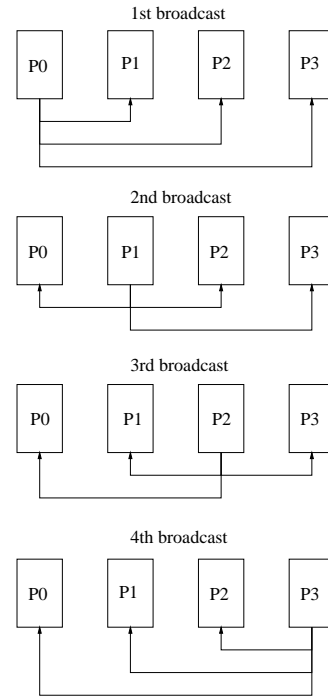


Figure 2. Communication round 2 (for 4 processors).

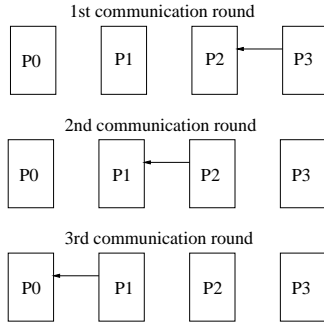


Figure 3. Communication round 3 (for 4 processors).

N	P=8	P=16	P=32
8192	16.98	14.39	14.00
16384	79.98	57.63	54.77
32768	359.76	290.64	223.70
65536	1541.41	1285.57	1023.71
131072	5520.55	4869.59	4228.59
262144	22072.40	20097.39	17871.40
524288	179902.82	85096.64	77384.11

Table 4. Total running times (in seconds) for each configuration of 8, 16 and 32 processors respectively.

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 3 and 4 present total running times (in seconds) for each configuration of 1, 2, 4, 8, 16 and 32 processors respectively. Figure 4 and Figure 5 presents total running times (in seconds) for different number of data items.

N	P=2	P=4	P=8	P=16	P=32
4096	0.01	0.03	0.26	0.39	1.18
8192	0.02	0.05	0.29	0.30	0.54
16384	0.03	0.09	0.17	0.34	1.06
32768	0.07	0.17	0.31	0.58	1.20
65536	0.16	0.33	0.57	1.05	2.07
131072	0.31	0.73	1.13	2.03	3.80
262144	0.61	1.44	2.52	4.01	7.24
524288	1.21	2.83	4.93	8.91	14.21

Table 5. Communication times (in seconds) for each configuration of 2, 4, 8, 16 and 32 processors respectively.

N	P=1	P=2	P=4
8192	47.17	35.42	24.80
16384	205.39	154.12	101.62
32768	1372.90	1029.67	429.70
65536	6351.09	4763.31	1986.05
131072	25662.83	19247.12	8022.57
262144	80840.70	60630.52	25268.50
524288	1279308.97	959481.72	399784.05

Table 3. Total running times (in seconds) for each configuration of 1, 2 and 4 processors respectively.

Table 5 presents communication times (in seconds) for each configuration of 2, 4, 8, 16 and 32 processors respectively. These communication times correspond to send and receive of $\frac{N}{P}$ characters by a processor to another with the use of the communication rounds described before. Here, we have $N=2^k$ and k is an integer such that $12 \leq k \leq 19$.

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.

Figure 6 shows that communication times (in seconds) increase when the number of processors used is greater. But figures 4 and 5 show 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.

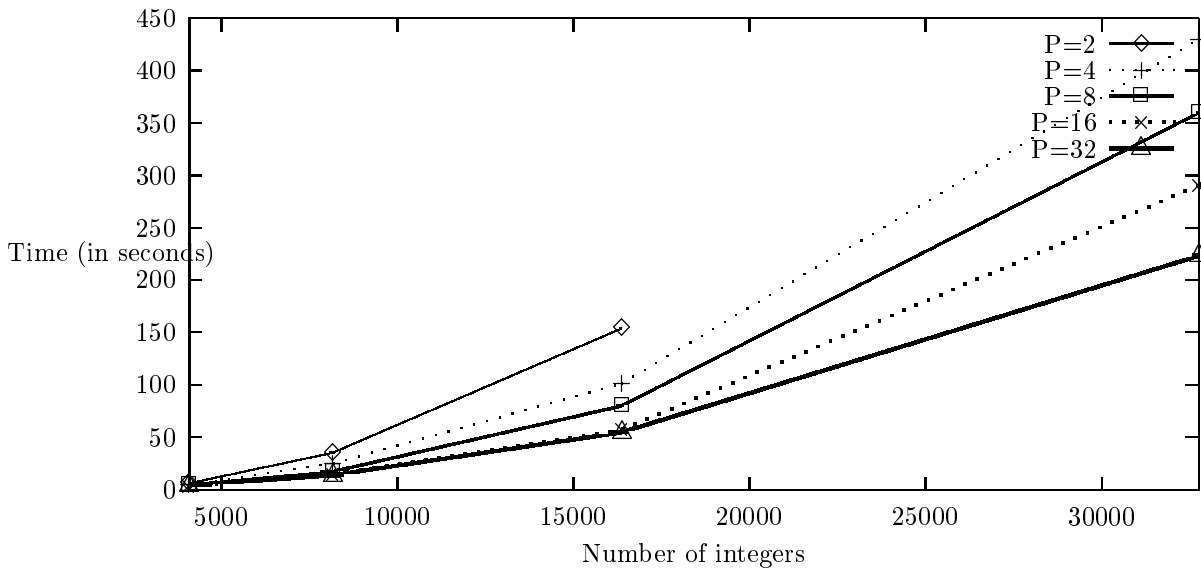


Figure 4. Total running times (in seconds) for different numbers of data items. The curves represent configurations of 2, 4, 8, 16 and 32 processors, respectively.

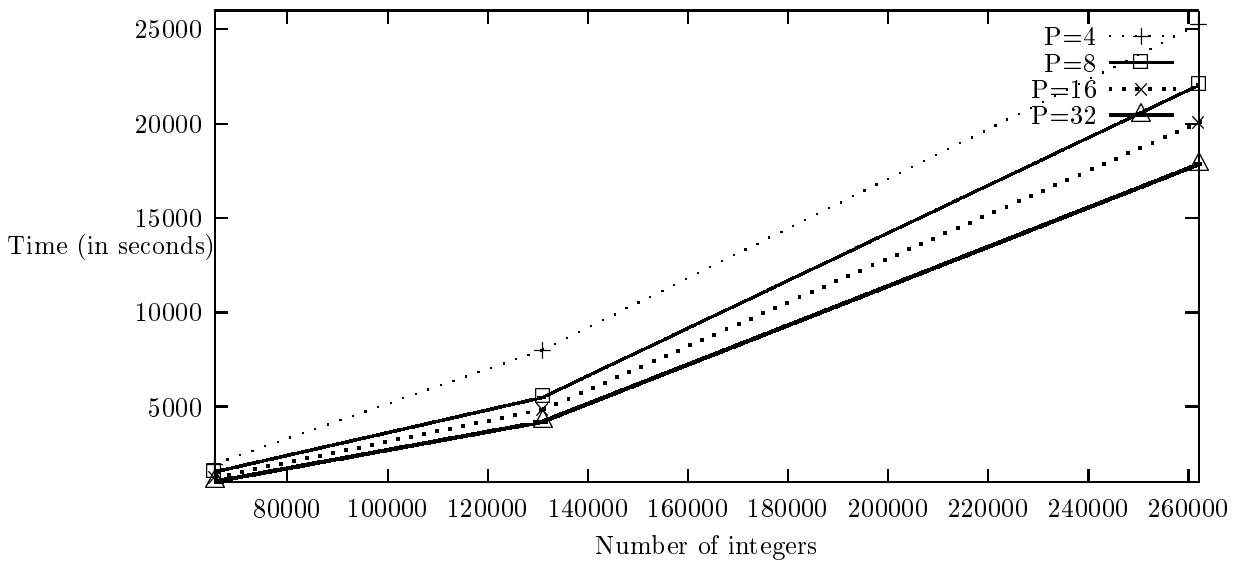


Figure 5. Total running times (in seconds) for different numbers of data items. The curves represent configurations of 4, 8, 16 and 32 processors, respectively.

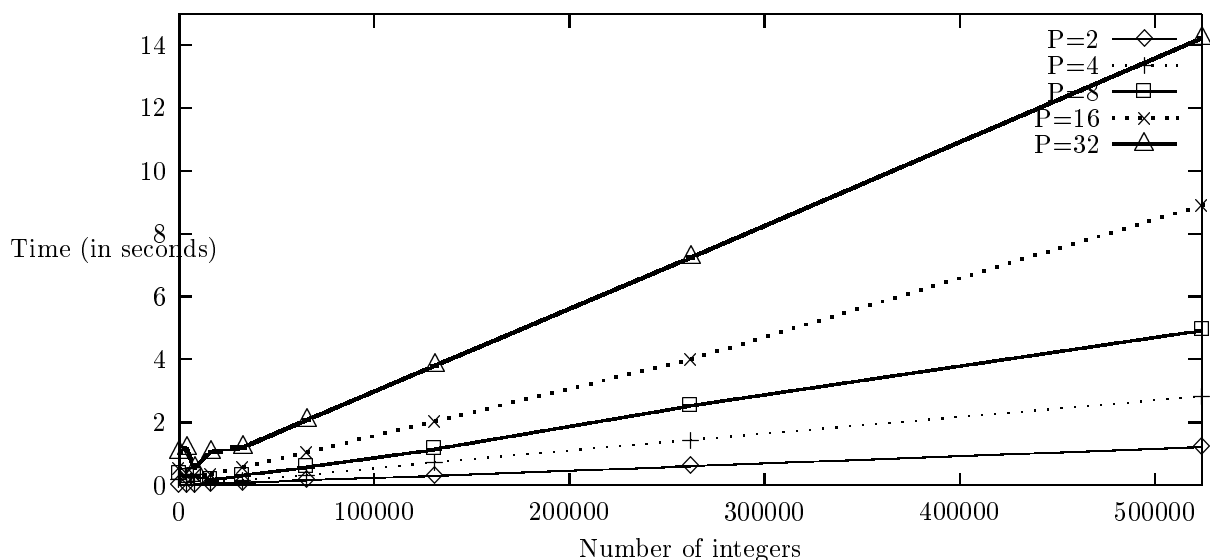


Figure 6. Communication times (in seconds) for different numbers of processors. The curves represent configuration of 256, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288 integers, respectively.

5 Concluding remarks

We presented a Coarse-Grained Multicomputer algorithm that solves the Longest Common Subsequence Problem. 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. The paper presents also experimental results showing that the CGM algorithm is very efficient, for great sequences A and B , when we increase the number of processors. We should now implement this algorithm on another platform in order to show its portability.

References

- [1] A. Aho, D. Hirschberg, and J.D. Ullman, Bounds on the Complexity of the Longest Common Subsequence Problem, *Journal of the ACM*. **23**,1 (1976),1–12.
- [2] A. Apostolico, M. Attalah, L. Larmore and S. Mcfaddin, Efficient Parallel Algorithms for String Editing and Related Problems, *SIAM Journal on Computing* (1990) 19:968–988.
- [3] A. Apostolico, S. Brown and C. Guerra, Fast linear-space computations of Longest Common Subsequences, *Theoretical Computer Science* **92** (1992),3–17.
- [4] 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)*, (1999) 125–129.
- [5] A. Chan and F. Dehne, A Note on Coarse Grained Parallel Integer Sorting, *Parallel Processing Letters*, (1999) 9(4):533–538.
- [6] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauer, 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* (1996) 1–12.
- [7] F. Dehne, A. Fabri and A. Rau-Chaplin, Scalable Parallel Computational Geometry for Coarse Grained Multicomputers, *International Journal on Computational Geometry* (1996) 6(3):379–400.
- [8] 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*, (1995) 27–33.
- [9] M. Diallo, A. Ferreira, A. Rau-Chaplin and S. Ubeda, Scalable 2D Convex Hull and Triangulation Algorithms for Coarse Grained Multicomput-

- ers, *Journal of Parallel and Distributed Computing*, (1999) 56(1):47-70.
- [10] M. Du and W. Hsu, New Algorithms for the Longest Common Subsequence Problem, *Journal of Computer and System Sciences* **29** (1984),133–152.
- [11] V. Dančík and M.S. Paterson, Longest Common Subsequences, In E.W. Mayr, P. Enjalbert and K.W. Wagner, editors, *Proc. 11th Annual Symp. on Theoretical Aspects of Computer Science, Caen, France*, number 775 in Lecture Notes in Computer Science, Berlin, Springer-Verlag, Academic Press. (1994),127–142.
- [12] A. Ferreira and N. Schabanel, A Randomized BSP/CGM Algorithm for the Maximal Independent Set Problem, *Parallel Processing Letters*, (1999) 9(3):411–422.
- [13] T. Garcia, J.F. Myoupo and D. Semé A Work-Optimal CGM Algorithm for the Longest Increasing Subsequence Problem, *International Conference n Parallel and Distributed Processing Techniques and Applications (PDPTA'01)*, (2001)
- [14] M. Goudreau, K. Lang, S. Rao, 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)* (1996) 1-12.
- [15] D.S. Hirschberg, Algorithms for the Longest Common Subsequence Problem, *Journal of the ACM* (1977) 24(4):664–675.
- [16] J.W. Hunt and T.G. Szymanski, A Fast Algorithm for Computing Longest Common Subsequences, *Communications of the ACM* (1977) 20:350–353.
- [17] Y. Kayambayashi, N. Nakatsu and S. Yajima, A Longest Algorithm Suitable for Similar Text String, *Acta Informatica* (1982) 18:171–179.
- [18] S.R. Kim and K. Park, Fully Scalable Fault-Tolerant Simulations for BSP and CGM, *Journal of Parallel and Distributed Computing*, (2000) 60:1531–1560.
- [19] T. Lecroq, G. Luce and J.F. Myoupo, A Faster Linear Systolic Algorithm for Recovering a Longest Common Subsequence, *Information Processing Letters* **61**,3 (1997),129–136.
- [20] M. Lu and H. Lin, Parallel Algorithms for the Longest Common Subsequence Problem, *IEEE Transactions on Computers* **5**,8 (1994),835–848.
- [21] G. Luce and J.F. Myoupo, Application-Specific Array Processors For The Longest Common Subsequence Problem of Three Sequences, *Parallel Algorithms and Applications* **13** (1998),27–52.
- [22] W.J. Masek and M.S. Paterson, A Faster Algorithm for Computing String Edit Distances, *Journal of Computer and System Sciences* (1980) 20:18–31.
- [23] Y. Robert and M. Tchuente, A Systolic Array for the Longest Common Subsequence Problem, *Information Processing Letters* **21** (1985),191–198.
- [24] D. Semé, Algorithmique Parallèle pour des Problèmes de Reconnaissance de Formes et de Motifs sur les Modèles Systolique et BSR, *Thèse de l'Université Picardie Jules Verne - Amiens* (1999)
- [25] M.S. Waterman, Sequence Alignments, In: *Mathematical Methods for DNA Sequence* **Ch 3** (1985),53–92.
- [26] L.G. Valiant, A Bridging Model for Parallel Computation, *Communications of the ACM* (1990) 33(8):103–111.