A Work-Optimal CGM Algorithm for the Longest Increasing 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

Abstract *This paper presents a work-optimal CGM algorithm that solves the Longest Increasing Subsequence Problem. It can be implemented in the CGM with P processors in O($\frac{N^2}{P}$) time and O(P) communication steps. It is the first CGM algorithm for this problem and it is work-optimal since the sequential algorithm has a complexity of O(N^2).*

Keywords: Parallel Algorithms, Coarse Grained Multicomputers, Longest Increasing Subsequence.

1 Introduction

The Longest Increasing Subsequence Problem (LIS for short) is a good illustration of dynamic programming and has interested many scientists [2, 8, 10, 15, 18, 19, 20]. In July 1978, E.W. Dijkstra at Marktoberdof's school, asked to his students to find the length of the longest increasing subsequence in a sequence of integers. Even though this problem seems relatively straightforward, few of them had been able to solve it. Today, this exercise remains a useful didactic example for topic of sequential programming methodology. In particular it shows how to strengthen an induction hypothesis in a very explicit way [12, 13, 17]. Finding LIS of two or more strings improves and speeds the compression in case of data transmission, for instance in image processing or in medical data processing.
Finding the length is performed in time O($N \log N$) (where N is the length of the input sequence) sequentially [17]. On the other hand we are interested in actually finding the longest increasing upsequence. For the problem, there exists a sequential algorithm running in time O(N^2) [14].

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, $P << N$. 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 [21] introducing the so-called Bulk Synchronous Parallel (BSP) machine, and was refined in different directions for example by Culler et al. [4], LogP, and Dehne et al. [5], CGM extensively studied in [1, 3, 6, 7, 9, 16]. 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. [11].

In this paper we present a work-optimal CGM algorithm that solves the Longest Increasing Subsequence Problem. This algorithm can be implemented in the CGM with P processors in $O(\frac{N^2}{P})$ time and $O(P)$ communication steps. It is the first CGM algorithm for this problem and it is work optimal since the sequential algorithm have a complexity of $O(N^2)$.

The paper is organized as follows. In section 2 we present the Longest Increasing Subsequence problem and some sequential algorithms. Section 3 presents the CGM solution of the LIS problem. The conclusion ends the paper.

2 The Longest Increasing Subsequence problem

2.1 Statement of the problem

Definition 1 *Given a sequence A of N distinct integers, a subsequence of A is a sequence L which can be obtained from A in deleting zero or some integers (not necessarily consecutive).*

Definition 2 *A sequence is increasing if each integer of this sequence is larger than the previous integer. Given a sequence $A = \{x_1, x_2, \ldots, x_N\}$ of N distinct integers, we define an increasing subsequence or upsequence of length l as a upsequence of A : $\{x_{i_1}, x_{i_2}, \ldots, x_{i_l}\}$ with $\forall\ j,k : 1 \leq j < k \leq l => i_j < i_k$ and $x_{i_j} < x_{i_k}$.*

Definition 3 *A longest or maximal increasing subsequence is one of maximal length. Note that a maximal upsequence in not necessarily unique.*

2.2 Sequential algorithms for the LIS problem

Definition 4 *A decreasing subsequence of A is a subsequence of A where the numbers are nonincreasing from left to right.*

Definition 5 *A cover of A is a set of decreasing subsequences of A that contains all the numbers of A.*

Definition 6 *The size of the cover is the number of decreasing subsequences in it, and a smallest cover is a cover with minimum size among all covers.*

Lemma 7 *If I is an increasing subsequence of A with length equal to the size of a cover of A, call it C, then I is a longest increasing subsequence of A and C is a smallest cover of A.*

Proof 1 *see [14].* □

We now summarize a sequential algorithm for the LIS due to [14], which is the basis of our CGM algorithm.

Let A be a set of N integers. We want to construct a decreasing cover of A. The idea is as follows: starting from the left of A, examine each successive number in A and place it at the end of the first (left-most) decreasing subsequence that it can extend. If there are no decreasing subsequences it can extend, then start a new decreasing subsequence to the right of all existing decreasing subsequences.
This algorithm produces a cover of A which is

called the greedy cover in [14]. After the greedy cover is found, a LIS of A can be found easily as it is described in [14].
At the end of the algorithm described in [14], I contains an LIS of A. The greedy cover of A is found in time $O(N^2)$ and the LIS found in time $O(N)$ given the greedy cover.

Theorem 8 *The i-th subsequence of the greedy cover contains all element of A which have the i-th position in the longest increasing subsequence containing it.*

Proof 2
Let GC_i be the i-th subsequence of the greedy cover.
Call x_k the last element of GC_k

In regard of the greedy cover's construction, we have : $x_1 < x_2 < \ldots < x_{i-1} < x_i$.
Then, x_i is the i-th element of the longest increasing subsequence containing it.
As each element of GC_i is greater than the last element of GC_i, x_i (i.e. the construction of the greedy cover), then all element of GC_i have the i-th position in the longest increasing subsequence containing it.

□

Find the position of an element in the longest increasing subsequence containing it can be done by the following proposition.

Proposition 1
$\forall i$, $Major[i]$=1
$\forall i, j : j < i$, $Major[i]$=max($Major[j] + 1$) if $A[j] < A[i]$.

Proof 3
$Major[i_k] = Major[i_{k-1}] + 1$ with $A[i_{k-1}] < A[i_k]$ and $\exists i_k, i_{k-1} : i_{k-1} < i_k$
$Major[i_{k-1}] = Major[i_{k-2}] + 1$ with $A[i_{k-2}] < A[i_{k-1}]$ and $\exists i_{k-1}, i_{k-2} : i_{k-2} < i_{k-1}$
$\ldots \qquad \ldots$
$\ldots \qquad \ldots$
$\ldots \qquad \ldots$
$Major[i_2] = Major[i_1] + 1$ with $A[i_1] < A[i_2]$ and $\exists i_2, i_1 : i_1 < i_2$

Then,

$Major[i_k] = k$ with $A[i_1] < A[i_1] < \ldots < A[i_{k-2}] < A[i_{k-1}] < A[i_k]$ and $i_1 < i_2 < \ldots < i_{k-2} < i_{k-1} < i_k$
We assume that k is maximum since we use the proposition 1.
Thus, the position of the element $A[i_k]$, in the longest increasing subsequence containing it, is k.

□

The following algorithm is based on the proposition 1. This sequential algorithm finds the position of each element $A[i]$ ($\forall$ i) in the longest increasing subsequence containing $A[i]$.

Sequential Algorithm 1

```
(1)  for (i=0) to (i=N-1)
        Major[i]=1
     endfor

(2)  for (i=1) to (i=N-1)
        for (j=0) to (j=i)
           if(A[j]<A[i] and Major[i]<Major[j]+1)
           Major[i]=Major[j]+1
           endif
        endfor
     endfor
```

Complexity: *It is obvious that the complexity in time of this sequential algorithm 1 is $O(N^2)$.*

The following sequential algorithm computes the Longest Increasing Subsequence from the result of the sequential algorithm 1 which computes for each element $A[i]$ ($\forall i$:$0 \leq i < N$) of A its position (i.e. $Major[i]$) in the Longest Increasing Subsequence containing it (i.e. $A[i]$). Then, we should find the maximum, called Max, between all the $Major[i]$ ($\forall i$:$0 \leq i < N$). We call Ind the minimum index such that $Max = Major[Ind]$. These operations are realized by the first part of the sequential algorithm 2.

Starting from $Major[Ind]$, which represents both the length of the LIS and the position of the element $A[Ind]$ in the LIS, we can easily extract an LIS in reverse order from the input sequence A. This operation is carried out by the second part of the sequential algorithm 2.

Sequential Algorithm 2

```
(1) Max=0
    Ind=0
    for (i=0) to (i=N-1)
      if (Major[i]>=Max)
        Max=Major[i]
        Ind=i
      endif
    endfor

(2) j=0
    for (i=N-1) to (i=0)
      if (Major[i]=Max')
        RLIS[j]=A[i]
        Max=Max-1
        j=j+1
      endif
    endfor
```

Complexity: *It is obvious that this sequential algorithm 2 has a complexity of O(N).*

Remark The array RLIS contains the Longest Increasing Subsequence in the reverse order. It is straightforward to construct the LIS from the RLIS in linear time.

In partitioning the sequence A, we can define another sequential algorithm based on the previous ones (sequential algorithms 1 and 2).

Sequential Algorithm 3

```
(1) for (num=0) to (num=P-1)
      for (i=0) to (i=$\frac{N}{P}$-1)
        Major[$\frac{num*N}{P}$+i]=1
      endfor
    endfor

(2) for (ii=0) to (ii=P-2)
      for (num=0) to (num=P-1)
        if (num=ii) then
          for (i=1)to (i=$\frac{N}{P}$-1)
            for (j=0) to (j=i)
              if (A[$\frac{num*N}{P}$+j]<A[$\frac{num*N}{P}$+i] and
Major[$\frac{num*N}{P}$+i]<Major[$\frac{num*N}{P}$+j]+1)
                Major[$\frac{num*N}{P}$+i]=Major[$\frac{num*N}{P}$+j]+1
              endif
            endfor
          endfor
        else
          if (num>ii)
            for (i=0) to (i=$\frac{N}{P}$-1)
              for (j=0) to (j=$\frac{N}{P}$-1)
                if (A[$\frac{ii*N}{P}$+j]<A[$\frac{num*N}{P}$+i] and
Major[$\frac{num*N}{P}$+i]<Major[$\frac{ii*N}{P}$+j]+1)
                  Major[$\frac{num*N}{P}$+i]=Major[$\frac{ii*N}{P}$+j]+1
                endif
              endfor
            endfor
          endif
        endif
      endfor
    endfor

(3) Max=0
    for (num=0) to (num=P-1)
      for (i=0) to (i=$\frac{N}{P}$-1)
        if (Major[$\frac{num*N}{P}$+i]>=Max)
          Max=Major[$\frac{num*N}{P}$+i]
        endif
      endfor
    endfor

(4) j=0
      for (num=P-1) to (num=0)
      for (i=$\frac{N}{P}$-1) to (i=0)
        if (Major[$\frac{num*N}{P}$+i]=Max')
          RLIS[j]=A[$\frac{num*N}{P}$+i]
          Max=Max-1
          j=j+1
        endif
      endfor
    endfor
```

Complexity: *The complexity of this sequential algorithm 3 is also $O(N^2)$. Indeed, we have $O(P \times \frac{N}{P})$ for the part (1), $O(P \times P \times (\frac{N}{P})^2)$ for the part (2), $O(P \times \frac{N}{P})$ for the part (3) and $O(P \times \frac{N}{P})$ for the part (4).*

3 The CGM solution for the LIS problem

The CGM algorithm presented in this section is directly issue to the sequential algorithm 3. Each processor *num* ($0 \leq num < P$) have 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*. Note that the k-th part of the program below corresponds to the k-th part of the sequential algorithm 3.

CGM Algorithm 1

```
(1)  for (i=0) to (i=$\frac{N}{P}$-1)
       Major[$\frac{num*N}{P}$+i]=1
     endfor

(2)  for (ii=0) to (ii=P-2)
       if (num=ii) then
         for (i=1)to (i=$\frac{N}{P}$-1)
           for (j=0) to (j=i)
           if (A[j]<A[i] and Major[i]<Major[j]+1)
               Major[i]=Major[j]+1
             endif
           endfor
         endfor
         Send(num,A,ALL_OTHERS)
         Send(num,Major,ALL_OTHERS)
       else
       Receive(num',A')
       Receive(num',Major')
         if (num>num')
           for (i=0) to (i=$\frac{N}{P}$-1)
             for (j=0) to (j=$\frac{N}{P}$-1)
               if (A'[j]<A[i] and Major[i]<Major'[j]+1)
               Major[i]=Major'[j]+1
               endif
             endfor
           endfor
         endif
       endif
     endfor

(3)  Max=0
     for (i=0) to (i=$\frac{N}{P}$-1)
       if (Major[i]>=Max)
         Max=Major[i]
       endif
     endfor
     Send(num,Max,ALL)
     for (i=0) to (i=P-1)
       Receive(num',Max')
       Max_proc[num']=Max'
     endfor
     Max'=Max_proc[0]
     for(i=1) to (i=P-1)
       if (Max'<Max_proc[i])
         Max'=Max_proc[i]
       endif
     endfor

(4)  if (num=P-1)
       j=0 for (i=$\frac{N}{P}$-1) to (i=0)
         if (Major[i]=Max')
           RLIS[j]=A[i]
           Max'=Max'-1
           j=j+1
         endif
       endfor
       Send(num,Max',num-1)
     else
       Receive(num',Max')
       j=0 for (i=$\frac{N}{P}$-1) to (i=0)
         if (Major[i]=Max')
           RLIS[j]=A[i]
           Max'=Max'-1
           j=j+1
         endif
       endfor
       Send(num,Max',num-1)
     endif
```

Remark Note that our approach uses two functions called **Send** and **Receive** which are defined as:
– **Send**(num,Max,**ALL**) where the values *num* (the processor's number) and Max are sent to all the processors,
– **Send**(num,A,**ALL_OTHERS**) where the values *num* and A are sent to all processors except the processor *num*,
– **Receive**(num',A') where the values num' and A' are received from the processor num'.
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 $O(\frac{N}{P})$ for the part (1), $O(P \times (\frac{N}{P})^2)$ with $O(P)$ communication steps for the part (2), $O(\frac{N}{P})$ with $O(1)$ communication steps, and $O(P \times \frac{N}{P})$ with $O(P)$ communication steps for the part (4). Then, this algorithm is work optimal, $O(N^2)$, since the complexity of the sequential algorithm is $O(N^2)$.*

4 Concluding remarks

We have described a work-optimal CGM algorithm that solves the Longest Increasing Subsequence Problem. This algorithm can be implemented in the CGM with P processors in $O(\frac{N^2}{P})$ time and $O(P)$ communication steps. It is the first CGM algorithm for this problem and it is work optimal since the sequential algorithm have a complexity of $O(N^2)$.

It will be interesting to reduce the number of communication steps: is there another approach yielding optimal communication rounds i.e. $\log P$? It seems to be a difficult problem since the LIS is based on a strong recursivity. Moreover, the complexity in time depends on the communication steps. As we have a work-efficient algorithm, reducing the communication steps yields the reduction of the complexity in time and then it will be necessary to have more processors in order to get work-efficiency.
The next step of this work consists of implementing our algorithm on many cluster of stations in order to study all its aspects.

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)*, (1999) 125–129.

[2] C. Cerin, C. Dufourd and J. F. Myoupo, An Efficient Parallel Solution for the Longest Increasing Subsequence Problem, *Fith International Conference on Computing and Information (ICCI'93)Sudbury, Ontario, IEEE Press*, (1993) 220-224.

[3] A. Chan and F. Dehne, A Note on Coarse Grained Parallel Integer Sorting, *Parallel Processing Letters*, (1999) 9(4):533–538.

[4] 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* (1996) 1–12.

[5] 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.

[6] 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.

[7] 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*, (1999) 56(1):47-70.

[8] P. Erdos and A. Szekers, A combinatorial problem in geometry, *Compositio Mathematica* (1935) 2:463–470.

[9] A. Ferreira and N. Schabanel, A Randomized BSP/CGM Algorithm for the Maximal Independant Set Problem, *Parallel Processing Letters*, (1999) 9(3):411–422.

[10] M.L. Fredman, On computing the length of the longest increasing subsequence, *Discrete Mathematic* **vol. 11** (1975) 29–35.

[11] 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.

[12] A. Gram, Raisonner pour programmer, *Dunod*, Paris, (1988) 97–160.

[13] D. Gries, The science of programming, *Springler Verlag* (1981) (fifth printing 1989).

[14] D. Gusfield, Algorithms on Strings, Trees, and Sequences : Computer Science and Computational Biology, *Cambridge University Press* (1997)

[15] G. Jacobson and K. P. Vo, Heaviest Increasing/Common Subsequence Prob-

lems, *In LNCS 644, Third Annual Symposium on Combinatorial Pattern Matching*, (1992).

[16] 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.

[17] U. Manber, Introduction to algorithms, a creative approach, *Adisson-Wesley* (1989).

[18] J. Misra, A technique of algorithm construction on sequence, *IEEE Trans. Software Engineering* **vol. SE-4 no. 1** (Janv. 1978) 65–69.

[19] D. Semé, Optimal Parallel Solutions for Longest Increasing Subsequence and Longest Increasing Chain Problems Using BSR Model, *Submitted* (2001).

[20] T.G. Szymanski, A Special Case of the Max Common Subsequences Problem, *Dep. Elec. Eng. Princeton University, Princeton N.J. Tech. Rep.* (1975).

[21] L.G. Valiant, A Bridging Model for Parallel Computation, *Communications of the ACM* (1990) 33(8):103–111.