Procedia Computer Science

Volume 51, 2015, Pages 2754–2758

ICCS 2015 International Conference On Computational Science

# Efficient BSP/CGM algorithms for the maximum subsequence sum and related problems*

Anderson C. Lima[1], Rodrigo G. Branco[1], Edson N. Cáceres[1], Roussian R. A. Gaioso[2], Samuel Ferraz[1], Siang W. Song[3], and Wellington S. Martins[2]

[1] Federal University of Mato Grosso do Sul, Campo Grande, Mato Grosso do Sul, Brazil
[2] Federal University of Goiás, Goiânia, Goiás, Brazil
[3] University of São Paulo, São Paulo, São Paulo, Brazil

**Abstract**

Given a sequence of $n$ numbers, with at least one positive value, the maximum subsequence sum problem consists in finding the contiguous subsequence with the largest sum or score, among all derived subsequences of the original sequence. Several scientific applications have used algorithms that solve the maximum subsequence sum. Particularly in Computational Biology, these algorithms can help in the tasks of identification of transmembrane domains and in the search for $GC$-content regions, a required activity in the operation of pathogenicity islands location. The sequential algorithm that solves this problem has $O(n)$ time complexity. In this work we present BSP/CGM parallel algorithms to solve the maximum subsequence sum problem and three related problems: the maximum longest subsequence sum, the maximum shortest subsequence sum and the number of disjoints subsequences of maximum sum. To the best of our knowledge there are no parallel BSP/CGM algorithms for these related problems. Our algorithms use $p$ processors and require $O(n/p)$ parallel time with a constant number of communication rounds for the algorithm of the maximum subsequence sum and $O(\log p)$ communication rounds, with $O(n/p)$ local computation per round, for the algorithm of the related problems. We implemented the algorithms on a cluster of computers using MPI and on a machine with GPU using CUDA, both with good speed-ups.

*Keywords:* Parallel algorithms, multicore, GPU, maximum subsequence sum problem.

## 1 Introduction

The maximum subsequence sum problem consists in finding the contiguous subsequence with the largest sum or score, among all derived subsequences of an original sequence of $n$ numbers,

*Authors thank CNPq, CAPES and FAPEG for financial support and NVIDIA for equipment donations.

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015


doi:10.1016/j.procs.2015.05.421

with at least one positive number [2]. Solution of this problem arises in many areas of Science. One such area is Computational Biology, where many applications require the solution of the maximum subsequence sum problem. Among these, the identification of transmembrane domains in protein sequences is an important application and represents one of the important tasks to understand the structure of a protein or the membrane topology [7]. Another biological application is finding regions of DNA that are rich or poor in nucleotides $G$ and $C$ (CG-content). The GC-content is especially important in the operation of search for pathogenicity islands [5].

The best sequential algorithm for the maximum subsequence sum problem, due to J. Kadane, has $O(n)$ time complexity [2]. However, despite being a simple and fast algorithm, it returns little information about the maximum subsequence sum. The output returns only the value of the maximum subsequence sum.

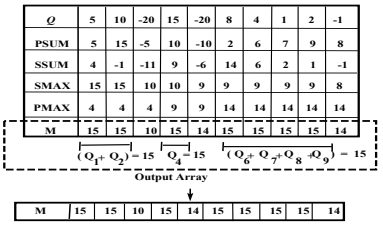


**Figure 1:** The value 15 is the highest and represents the value of maximum sum of **Q**.

Previous works have reported good parallel solutions for the maximum subsequence sum problem. In particular, for this work, we have used some ideas from Perumalla and Deo [6] parallel algorithm in our results. This algorithm generates a series of arrays, whose meanings and calculations can be checked in the original work [6]. Figure 1 illustrates an example of the output array for this algorithm.

In this work, we revisited the maximum subsequence sum problem and propose solutions to three related problems: the maximum longest subsequence sum, the maximum shortest subsequence sum and the number of disjoints subsequences of maximum sum. To the best of our knowledge there are no parallel BSP/CGM algorithms for these three problems. The basis of our solutions involves several variations of prefix sum in parallel [4].

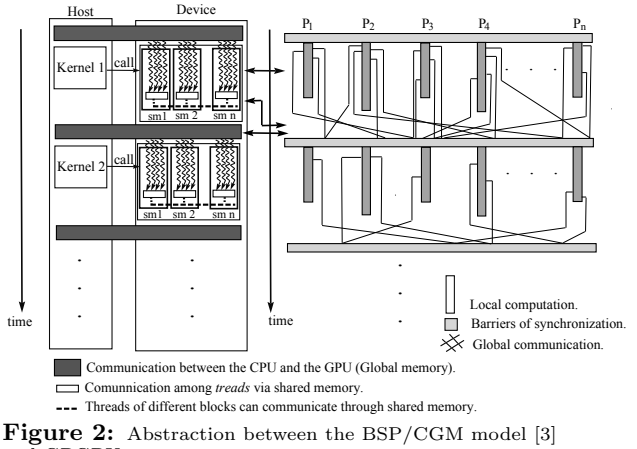


**Figure 2:** Abstraction between the BSP/CGM model [3] and GPGPU

Our algorithms use $p$ processors with $O(n/p)$ parallel time and a constant number of communication rounds. In order to show the efficiency not only in theory but also in practice, the algorithms were implemented on a cluster of computers using MPI and on a machine with GPU using CUDA, both with good results. Figure 2 illustrates our suggestion for working on a GPGPU (General Purpose Graphics Processing Unit) with CUDA using the BSP/CGM model [3] .

This paper is organized as follows: Section 2 presents the background and the related works to this study. Section 3 presents our proposed BSP/CGM algorithms. The implementations and results are presented in Section 4. Finally, Section 5 presents the conclusions and possible future work.

# 2 Background

Consider a sequence $Q_n = (q_1, q_2, \ldots, q_n)$ composed of $n$ integers, a subsequence is any contiguous segment $(q_i, \ldots, q_j)$ of $Q_n$ for $1 \leq i \leq j \leq n$. The **maximum subsequence sum** problem

is to determine, among all possible subsequences, the subsequence $M = (q_i, \ldots, q_j)$ that has the maximum sum $(\sum_{k=i}^{j} q_k)$ [2]. In the sequence represented by Figure 3 there are three disjoints subsequences of maximum sum. All have sum equal to 15, but with different sizes.

For simplicity, without loss of generality, assume the numbers to be integers. In a given sequence of integers, as the sequence $Q$ illustrated in Figure 3, there might be more than one maximum subsequence sum. In this context, at least three new problems arise: the maximum longest subsequence sum, the maximum shortest subsequence sum and the number of disjoints subsequences of maximum sum.

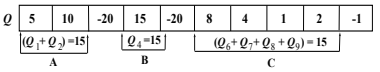


**Figure 3:** The three subsequences of maximum sum of the sequence $Q$.

# 3 Proposed Algorithms

Although there is a BSP/CGM parallel algorithm for the 1D maximum subsequence problem, we cannot use it to find, directly, the shortest/longest maximum subsequences. Using the ideas of the PRAM algorithm proposed by Perumalla and Deo [6], we devised a new BSP/CGM parallel algorithm for this problem (see Algorithm 1) such that, using the array output, we can compute the shortest/longest maximum subsequences and the total number of maximum subsequences. The algorithms in steps 2, 3 and 4 are variations of the prefix sum. [6].

**Algorithm 1** Maximum Subsequence Sum

**Input:** (1) A set of $P$ processors; (2) The number $i$ of each processor $p_i \in P$, where $1 \leq i \leq P$; (3) Sequence Q of $n$ integers.
**Output:** (1) Array $\mathbf{M}[1 \ldots n]$ of integers with all disjoints subsequences of maximum sum; (2) The maximum value in $\mathbf{M}$.

1: PSUM $\leftarrow$ Prefix Sum Algorithm($P$, Q) **in parallel**.
2: SSUM $\leftarrow$ Suffix Sum Algorithm($P$, Q) **in parallel**. {*Prefix sum operation applied on the inverse array Q.*}
3: SMAX $\leftarrow$ Suffix Maxima Algorithm($P$, PSUM) **in parallel**. {*Propagation of maximum values from the end to the beginning.*}
4: PMAX $\leftarrow$ Prefix Maxima Algorithm($P$, SSUM) **in parallel**. {*Propagation of maximum values from the beginning to the end.*}
5: Processor $p_1$ sends $n/p$ elements of each array **Q**, **PSUM**, **SSUM**, **SMAX** and **PMAX** to each processor $p_i \in P$.
6: Each processor $p_i$ obtains the local arrays
**LocalMS** $\leftarrow$ **PMAX**$(n/p)$-**SSUM**$(n/p)$+**Q**$(n/p)$
**LocalMP** $\leftarrow$ **SMAX**$(n/p)$-**PSUM**$(n/p)$+**Q**$(n/p)$.
**LocalM** $\leftarrow$ **LocalMS**$(n/p)$+**LocalMP**$(n/p)$-**Q**$(n/p)$.
7: Each processor $p_i$ sends the array **LocalM** to processor $p_1$, which computes the array $M$:
$M = [LocalM_{p1_1}, \ldots, LocalM_{p1_{n/p}}, \ldots, LocalM_{pp_1}, \ldots, LocalM_{pp_{n/p}}]$ .
8: Maximum Subsequence Sum $\leftarrow$ Maximum Reduction Algorithm ($P$, M) **in parallel** .

The second algorithm solves the related problems. For better understanding, we chose to present the steps of this algorithm through the Figure 4. The starting point is the output of the Algorithm 1. In particular, our algorithms use $p$ processors and require $O(n/p)$ parallel time with a constant number of communication rounds for the algorithm of the maximum subsequence sum and $O(\log p)$ communication rounds, with $O(n/p)$ local computation per round, for the algorithm of the related problems, due to internal list ranking algorithm.

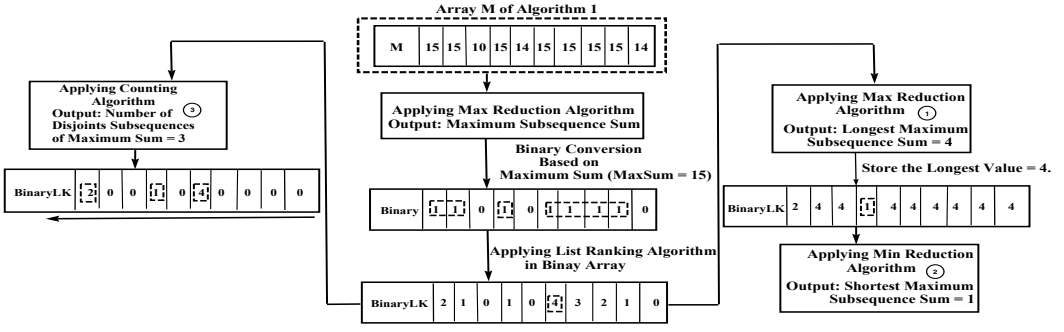


**Figure 4:** Problems that can be solved from the output array **M** of Algorithm 1

# 4 Results

Below we present the results achieved by the developed algorithms. The Figures 5-7 illustrated the cases of comparison between the versions of algorithms for the general problem of maximum subsequence sum. The Figure 8 illustrates the case of comparison for the related problems. The MPI version is based on the parallel BSP/CGM algorithm developed by Alves et al. [1] and involves data compression. The CUDA versions were built from Algorithm 1.

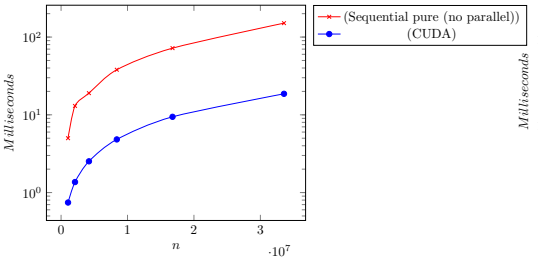


**Figure 5:** Sequential [2] × Parallel CUDA.

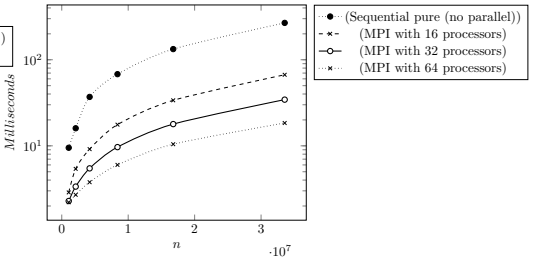


**Figure 6:** Sequential [2] × Parallel MPI (three executions: 16,32,64 (processors))

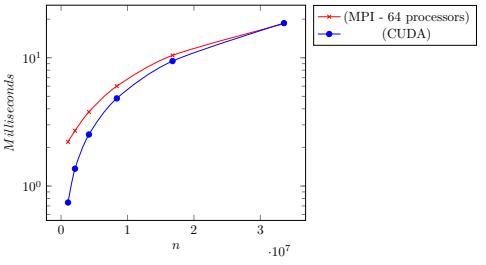


**Figure 7:** MPI × CUDA

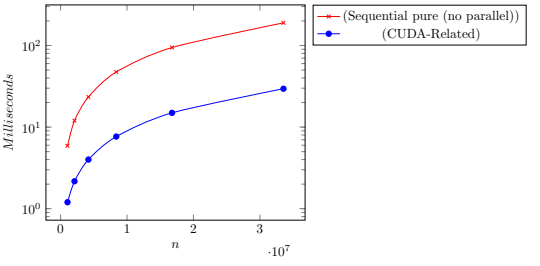


**Figure 8:** Sequential [2] × Related Problems. We modified the sequential algorithm to also solve the related problems.

| $n$ million(s) | Speedup - Figure 5 | Speedup - Figure 6 | Speedup - Figure 7 | Speedup - Figure 8 |
|---|---|---|---|---|
| 1.048.576 | 6.71709 | 4.29897 | 2.96873 | 4.91527 |
| 2.097.152 | 9.52437 | 5.92933 | 1.97700 | 5.53549 |
| 4.194.304 | 7.53388 | 9.78575 | 1.49924 | 5.84987 |
| 8.388.608 | 7.87658 | 11.32026 | 1.24511 | 6.18857 |
| 16.777.216 | 7.63946 | 12.73724 | 1.10791 | 6.32009 |
| 32.554.432 | 8.10558 | 14.54947 | 0.98876 | 6.41110 |

**Table 1: The values of speedup for the cases described in Figures 5 to 8**. Particularly, in the speedup of Figure 6, we use the best run time (64 processors). We have run the algorithms on two different computing systems. The **CUDA algorithms** were executed on a machine with 8 GB of RAM, Operating System Linux (Ubuntu 14:04), (R) Intel Core (TM) processor i5-2430M @ 2.40GHz CPU and an NVIDIA GeForce GTX 680 with 1536 processing cores. The **MPI algorithm** was executed on a cluster with 64 nodes, each node consisting of 4 processors Dual-Core AMD Opteron 2.2 GHz with 1024 KB of cache and 8 GB of memory. All nodes are interconnected via a Foundry SuperX switch using Gigabit ethernet. This cluster belongs to High Performance Computing Virtual Laboratory (HPCVL) of Carleton University. We expect that the triggered processes were equally distributed among processors of each node of the cluster. For equivalent comparisons with the CUDA/MPI algorithms, the **sequential algorithms** were run on the cluster environment and on the machine with CPU and GPU. Experiments with different sizes of sequences were performed. For each input sequence, the algorithms were run 10 times and the arithmetic mean of the running times was computed.

# 5 Conclusion and Future Work

In this work we propose efficient parallel solutions to the maximum subsequence sum and three related problems. The related problems are: the maximum longest subsequence sum, the

maximum shortest subsequence sum and the number of disjoints subsequences of maximum sum. To the best of our knowledge there are no parallel algorithms for these related problems. Our algorithms use $p$ processors and require $O(n/p)$ parallel time with a constant number of communication rounds for the algorithm of the maximum subsequence sum and $O(\log p)$ communication rounds, with $O(n/p)$ local computation per round, for the algorithm of the related problems. The good performance of the parallel algorithms is confirmed by experimental results. The CUDA version, for the Algorithm 1, presented a speedup of $\approx$ 9x as compared to the efficient $O(n)$ sequential algorithm [2]. On the other hand the CUDA version for the algorithm of related problems, presented a speedup of $\approx$ 6x as compared with the version of the sequential algorithm, which solves the same problems. When we compare the MPI version with the CUDA version, both versions present close running times. Importantly, it was from the CUDA version of Algorithm 1, that we propose an algorithm for the three related problems. We also conclude that the BSP/CGM model can be used efficiently to design parallel algorithms in GPGPU/CUDA environments. Besides all the details of this environments, we have mapped our algorithms into implementations with good speedups, where a single GPGPU got better performance than a cluster of workstations. Finally, we believe that the speedup of our CUDA algorithms can be further improved through the use of multiple GPUs.

### 5.1 Future Work

A possibility of future work consists in computing the $k$-th maximal longest or shortest subsequence sum. Other work is the implementation of the our BSP/CGM algorithm for the maximum subsequence sum using a multi GPGPU enviroment. We believe that the parallel time of this algorithm in an multi GPGPU can decrease considerably.

## References

[1] Carlos E. R. Alves, Edson N. Cáceres, and Siang W. Song. BSP/CGM algorithms for maximum subsequence and maximum subarray. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 3241 of *Lecture Notes in Computer Science*, pages 139–146. Springer Berlin Heidelberg, 2004.

[2] Jon Bentley. Programming pearls: Algorithm design techniques. *Commun. ACM*, 27(9):865–873, September 1984.

[3] Silvia M. Götz. *Communication-Efficient Parallel Algoritms for Minimum Spanning Tree Computation.* PhD thesis, University of Paderborn, May 1998.

[4] Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. *J. ACM*, 27(4):831–838, October 1980.

[5] Sandra L. Marcus, John H. Brumell, Cheryl G. Pfeifer, and B.Brett Finlay. Salmonella pathogenicity islands: big virulence in small packages. *Microbes and Infection*, 2(2):145–156, 2000.

[6] Kalyan Perumalla and Narsingh Deo. Parallel algorithms for maximum subsequence and maximum subarray. *Parallel Processing Letters*, 5:367–363, 1995.

[7] Walter L. Ruzzo and Martin Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 234–241, August 1999.