Minimising processor communication in parallel approximate string matching

Christos Hadjinikolis and Costas S. Iliopoulos
Department of Informatics
King's College London
London, UK
{*christos.hadjinikolis, c.iliopoulos*}*@kcl.ac.uk*

Solon P. Pissis and Alexandros Stamatakis
Scientific Computing group
Heidelberg Institute for Theoretical Studies
Heidelberg, Germany
{*solon.pissis, alexandros.stamatakis*}*@h-its.org*

***Abstract*—In this report, we focus on the efficient parallelisation of approximate string-matching algorithms, which are based on dynamic programming, using the *message-passing programming* paradigm. We first present the decomposition and the mapping technique of the proposed parallel system. Then, we show a number of properties that follow from the application of this system, and use them to minimise the amount of point-to-point communication in each parallel step. We therefore provide a new cost-optimal parallel system, which can be directly and effectively applied to any approximate string-matching algorithm with analogous characteristics. We show that, from a practical point of view, the proposed system reduces the time for point-to-point communication up to five times compared to the naïve version of it, and that it can match, and also outperform, the classic parallel system introduced by Edmiston *et al.* in 1988. Finally, we provide a performance predictor that can be used to accurately and efficiently predict the performance of Edmiston's and our system for given input data.**

***Keywords*-algorithms on strings; parallel algorithms; message-passing programming paradigm; dynamic programming; wave-front parallelism**

INTRODUCTION

The problem of finding factors of a text similar to a given pattern has been intensively studied over the past decades, and it is a central problem for a wide range of applications, including signal processing [8], information retrieval [12], searching for similarities among biological sequences [11], file comparison [3], and spelling correction [9].

Approximate string matching, in general, consists in locating all occurrences of factors inside a text t that are similar to a pattern x. It consists in producing the positions of the factors of t that are at a distance of at most k from x, for a given natural number k. In particular, we consider the *edit distance* for measuring the approximation. The edit distance between two strings, not necessarily of the same length, is the minimum cost of a series of elementary edit operations to transform one string into the other.

A *parallel system* is the combination of an algorithm and the parallel architecture on which it is implemented and executed. There has been ample work in the literature for devising parallel systems, using different models of computation, parallel architectures, and programming paradigms, for the approximate string-matching problem (cf. [1], [4], [7], [10]). Here, we focus on the efficient parallelisation of approximate string-matching algorithms, which are based on dynamic programming, using the *message-passing programming* (MPP) paradigm.

As an example of an approximate string-matching algorithm, one may consider any algorithm using the edit distance, such as the classic algorithm for computing the edit distance [8], algorithm MAXSHIFT [5] for fixed-length approximate string matching, the well-established algorithm for solving the longest common subsequence problem [12], or the Smith-Waterman algorithm [13] for performing local sequence alignment. The common characteristic of these algorithms is the computation of a dynamic-programming matrix $\mathsf{D}[0\,..\,n, 0\,..\,m]$, where n is the length of the text, and $m \leq n$ is the length of the pattern. Another key property is that each cell $\mathsf{D}[i, j]$ of the matrix can be computed only using cells $\mathsf{D}[i-1, j]$, $\mathsf{D}[i-1, j-1]$, and $\mathsf{D}[i, j-1]$.

We first present the decomposition and the mapping technique of the cost-optimal parallel system, introduced in [6], which makes use of wave-front parallelism in the MPP paradigm. Then, we demonstrate a number of properties that follow from the application of this system, and use them to minimise the amount of point-to-point communication in each parallel step. We therefore provide a new cost-optimal parallel system that can be directly and effectively applied to any approximate

string-matching algorithm with analogous characteristics.

The rest of this report is structured as follows. In Section I, we describe the model of communication in the parallel computer. In Section II, we provide a detailed description of the proposed parallel system. In Section III, we present extensive experimental results, which demonstrate that the proposed system reduces the time for point-to-point communication up to five times compared to the one introduced in [6], and that it can match, and also outperform, the classic parallel system introduced by Edmiston *et al.* [1] in 1988. We also provide a performance predictor that can be used to accurately and efficiently predict the performance of these systems for given input data. We conclude in Section IV.

I. Model of communication

We make the following assumptions for the model of communication: (i) the parallel computer comprises a number of nodes and (ii) each node comprises one or several identical processors interconnected by a switched communication network. Although the latencies for intra- and inter-node communication can vary substantially in practice, we assume, as an abstraction, that the time taken to send a message of size s between any two nodes is independent of the distance between nodes, and can be modeled as $t_{\text{comm}} = t_s + st_w$, where t_s is the latency of the message, and t_w is the transfer time per data. The communication links between two nodes are bidirectional and single-ported, that is, a message can be simultaneously transferred in both directions via the link, and only one message can be sent, and one message can be received at the same time. We assume that the input of size $n + m$ is stored locally on each node. This can be achieved by using an initial, one-time-only, broadcast operation one-to-all in communication time $(t_s+(n+m)t_w)\log p$, which is asymptotically $\mathcal{O}(n \log p)$, where p is the total number of available processors.

II. The proposed parallel system

A. Decomposition technique

We make use of the *functional decomposition* technique, in which the initial focus is on the computation that is to be performed, rather than on the data manipulated by the computation. The main observation for introducing parallelism is that cell $\mathsf{D}[i, j]$ can be computed using cells $\mathsf{D}[i-1, j]$, $\mathsf{D}[i-1, j-1]$, and $\mathsf{D}[i, j-1]$. Based on this, if we partition the problem of computing matrix D into a set $\{\Delta_0, \Delta_1, \ldots, \Delta_{n+m}\}$ of anti-diagonal arrays—also known as a sequence of parallel *wavefronts* of a computation—as shown in Equation 1, the computation of each cell in each diagonal is independent from the other cells of the same diagonal, and can be executed concurrently. The size of Δ_ν, that is, the number of cells of Δ_ν, is denoted by δ_ν.

$$\Delta_\nu[x] = \begin{cases} \mathsf{D}[\nu - x, x] : 0 \le x \le \nu & \text{(a)} \\ \mathsf{D}[\nu - x, x] : 0 \le x < m + 1 & \text{(b)} \\ \mathsf{D}[n - x, \nu - n + x] : & \\ 0 \le x < n + m - \nu + 1 & \text{(c)} \end{cases} \tag{1}$$

where

(a) if $0 \le \nu < m$
(b) if $m \le \nu < n$
(c) if $n \le \nu < n + m + 1$

B. Mapping technique

Regarding the mapping of tasks to the available processors, we distribute the computation of the cells for each diagonal among the available processors. We assume, without loss of generality, that for each diagonal array Δ_ν, each processor r, for $0 \le r < p$, is assigned δ_ν/p contiguous cells of Δ_ν. The mapping is described by Equation 2.

$$\begin{array}{ll} f_\nu^r = r\lceil \delta_\nu/p \rceil & \\ l_\nu^r = f_\nu^r + \lceil \delta_\nu/p \rceil - 1 & \text{if } r < \delta_\nu \bmod p \\ & \\ f_\nu^r = r\lfloor \delta_\nu/p \rfloor + \delta_\nu \bmod p & \\ l_\nu^r = f_\nu^r + \lfloor \delta_\nu/p \rfloor - 1 & \text{otherwise} \end{array} \tag{2}$$

It relies on calculating both, the indices f_ν^r and l_ν^r of the first and the last cell of Δ_ν mapped to processor r, respectively. In other words, processor r is assigned the computation of $\Delta_\nu[f_\nu^r \,..\, l_\nu^r]$ (see Fig. 1 for $\nu = 13$ and $\nu = 23$). Trivially, the auxiliary space required is $\mathcal{O}(m/p)$, as each diagonal depends only on the two immediately preceding diagonals. Although there also exist cost-optimal parallel algorithms in terms of space [4], [10], these come with a runtime overhead, in practice, to find the partial balanced partition of the text and the pattern (see [10] for details).

Furthermore, it is necessary to introduce the notion of *neighbouring*. We say that two discrete processors r and $q > r$ are neighbouring if and only if $l_\nu^r = \Delta_\nu[x]$ and $f_\nu^q = \Delta_\nu[x+1]$, for some x, where $0 \le x < \delta_\nu - 1$.

C. Properties of the system

For each sequence $\nu - 2, \nu - 1, \nu$ of parallel steps, where $2 \le \nu < n + m + 1$, we distinguish among the following cases:

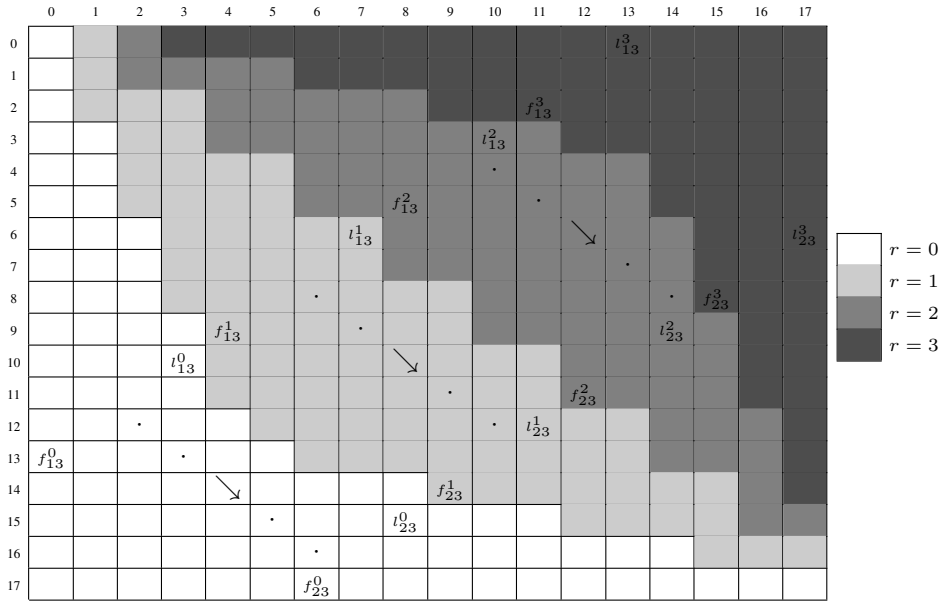


Figure 1. Mapping of the cells of matrix D to the available processors for $n = m = 17$ and $p = 4$

1) $\delta_{\nu-1} - \delta_{\nu-2} = 1$ and $\delta_\nu - \delta_{\nu-1} = 1$
2) $\delta_{\nu-1} - \delta_{\nu-2} = 1$ and $\delta_\nu - \delta_{\nu-1} = 0$
3) $\delta_{\nu-1} - \delta_{\nu-2} = 1$ and $\delta_\nu - \delta_{\nu-1} = -1$
4) $\delta_{\nu-1} - \delta_{\nu-2} = 0$ and $\delta_\nu - \delta_{\nu-1} = -1$
5) $\delta_{\nu-1} - \delta_{\nu-2} = 0$ and $\delta_\nu - \delta_{\nu-1} = 0$
6) $\delta_{\nu-1} - \delta_{\nu-2} = -1$ and $\delta_\nu - \delta_{\nu-1} = -1$

For each set $\{\mathsf{D}[i,j], \mathsf{D}[i-1,j], \mathsf{D}[i,j-1]\}$ of cells, where $1 \leq i \leq n$ and $1 \leq j \leq m$, we list the following immediate observations obtained from the problem decomposition (see Equation 1 in the Appendix):

Observation 1: If $\delta_\nu - \delta_{\nu-1} = 0$ and $\mathsf{D}[i,j] = \Delta_\nu[x]$, then $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x]$ and $\mathsf{D}[i,j-1] = \Delta_{\nu-1}[x-1]$, for some x, where $1 \leq x < \delta_\nu$.

Observation 2: If $\delta_\nu - \delta_{\nu-1} = -1$ and $\mathsf{D}[i,j] = \Delta_\nu[x]$, then $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x+1]$ and $\mathsf{D}[i,j-1] = \Delta_{\nu-1}[x]$, for some x, where $0 \leq x < \delta_\nu - 1$.

Observation 3: If $\delta_\nu - \delta_{\nu-1} = 1$ and $\mathsf{D}[i,j] = \Delta_\nu[x]$, then $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x]$ and $\mathsf{D}[i,j-1] = \Delta_{\nu-1}[x-1]$, for some x, where $1 \leq x < \delta_\nu$.

Lemma 1: If a processor r is assigned cell $\mathsf{D}[i,j]$ of Δ_ν, such that $\nu > r$, then r will also be assigned at least one of the cells $\mathsf{D}[i-1,j]$ and $\mathsf{D}[i,j-1]$ of $\Delta_{\nu-1}$.

Proof: Without loss of generality, assume that cell $\mathsf{D}[i,j] = \Delta_\nu[x]$ is mapped to processor r. Then, $f_\nu^r \leq x \leq l_\nu^r$ holds. We distinguish among the following cases:

1) $\delta_\nu - \delta_{\nu-1} = 0$: then, trivially, it holds $f_{\nu-1}^r = f_\nu^r \leq x \leq l_\nu^r = l_{\nu-1}^r$. From Observation 1, $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x]$, and, hence, $\mathsf{D}[i-1,j]$ is mapped to r.
2) $\delta_\nu - \delta_{\nu-1} = -1$: then, from Observation 2, $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x+1]$ and $\mathsf{D}[i,j-1] = \Delta_{\nu-1}[x]$. If the one extra cell of $\Delta_{\nu-1}$ ($\delta_\nu - \delta_{\nu-1} = -1$) is mapped to a processor $q > r$, then it holds $f_{\nu-1}^r = f_\nu^r$ and $l_{\nu-1}^r = l_\nu^r$. It follows that $f_{\nu-1}^r = f_\nu^r \leq x \leq l_{\nu-1}^r = l_\nu^r$, hence, $\Delta_{\nu-1}[x]$ is also mapped to r. If the one extra cell of $\Delta_{\nu-1}$ is mapped to a processor $q = r$, then it holds $f_{\nu-1}^r = f_\nu^r$ and $l_{\nu-1}^r = l_\nu^r + 1$. It follows that $f_{\nu-1}^r = f_\nu^r \leq x+1 \leq l_\nu^r + 1 = l_{\nu-1}^r$, hence, $\Delta_{\nu-1}[x+1]$ is mapped to r. If the one extra cell of $\Delta_{\nu-1}$ is mapped to a processor $q < r$, then it holds $f_{\nu-1}^r = f_\nu^r + 1$ and $l_{\nu-1}^r = l_\nu^r + 1$. It follows that $f_{\nu-1}^r = f_\nu^r + 1 \leq x+1 \leq l_\nu^r + 1 = l_{\nu-1}^r$, hence, $\Delta_{\nu-1}[x+1]$ is mapped to r.
3) $\delta_\nu - \delta_{\nu-1} = 1$: then, from Observation 3, $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x]$ and $\mathsf{D}[i,j-1] = \Delta_{\nu-1}[x-1]$. If the one extra cell of Δ_ν ($\delta_\nu - \delta_{\nu-1} = 1$) is mapped to a processor $q > r$, then it holds $f_{\nu-1}^r = f_\nu^r$ and $l_{\nu-1}^r = l_\nu^r$. It follows that $f_{\nu-1}^r = f_\nu^r \leq x \leq l_{\nu-1}^r = l_\nu^r$, hence, $\Delta_{\nu-1}[x]$ is also mapped to r. If the one extra cell of Δ_ν is mapped to a processor $q = r$, then it holds $f_{\nu-1}^r = f_\nu^r$ and $l_{\nu-1}^r = l_\nu^r - 1$. In case $f_\nu^r < x = l_\nu^r$, then it holds $f_{\nu-1}^r = f_\nu^r \leq x-1 = l_\nu^r - 1 = l_{\nu-1}^r$,

hence, $\Delta_{\nu-1}[x-1]$ is mapped to r. In case $f^r_\nu \leq x < l^r_\nu$, then it holds $f^r_{\nu-1} = f^r_\nu \leq x \leq l^r_\nu - 1 = l^r_{\nu-1}$, hence, $\Delta_{\nu-1}[x]$ is mapped to r. If the one extra cell of Δ_ν is mapped to a processor $q < r$, then it holds $f^r_{\nu-1} = f^r_\nu - 1$ and $l^r_{\nu-1} = l^r_\nu - 1$. It follows that $f^r_{\nu-1} = f^r_\nu - 1 \leq x-1 \leq l^r_\nu - 1 = l^r_{\nu-1}$, hence, $\Delta_{\nu-1}[x-1]$ is mapped to r.

■

Lemma 2: There exist no four cells $\mathsf{D}[i,j], \mathsf{D}[i-1,j], \mathsf{D}[i,j-1], \mathsf{D}[i-1,j-1]$, where $1 \leq i \leq n$ and $1 \leq j \leq m$, the calculation of the content of which is mapped to more than two discrete neighbouring processors.

Proof: Without loss of generality, assume that cell $\mathsf{D}[i,j] = \Delta_\nu[x]$ is mapped to processor r. Then it holds $f^r_\nu \leq x \leq l^r_\nu$. We distinguish among the following cases:

1) $\delta_{\nu-1} - \delta_{\nu-2} = 1$ and $\delta_\nu - \delta_{\nu-1} = 1$: From Observation 3, $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x]$, $\mathsf{D}[i,j-1] = \Delta_{\nu-1}[x-1]$, and $\mathsf{D}[i-1,j-1] = \Delta_{\nu-2}[x-1]$.

   (a) If the one extra cell of Δ_ν ($\delta_\nu - \delta_{\nu-1} = 1$) is mapped to a processor $q > r$, then $\Delta_{\nu-1}[x]$ is also mapped to r (see Lemma 1 Case 3). It follows that $\Delta_{\nu-1}[x-1]$ is allocated either to r or to $r-1$.

   If $\Delta_{\nu-1}[x-1]$ is mapped to r then it holds $f^r_{\nu-1} \leq x-1 < l^r_{\nu-1}$. If the one extra cell of $\Delta_{\nu-1}$ ($\delta_{\nu-1} - \delta_{\nu-2} = 1$) is mapped to a processor $q > r$ then it holds $f^r_{\nu-2} = f^r_{\nu-1}$ and $l^r_{\nu-2} = l^r_{\nu-1}$. It follows that $f^r_{\nu-2} = f^r_{\nu-1} \leq x-1 < l^r_{\nu-1} = l^r_{\nu-2}$, hence, $\Delta_{\nu-2}[x-1]$ is also mapped to r. If the one extra cell of $\Delta_{\nu-1}$ ($\delta_{\nu-1} - \delta_{\nu-2} = 1$) is mapped to a processor $q \leq r$ then it holds $f^r_{\nu-2} \leq f^r_{\nu-1}$ and $l^r_{\nu-2} = l^r_{\nu-1} - 1$. It follows that $f^r_{\nu-2} \leq f^r_{\nu-1} \leq x-1 < l^r_{\nu-1} - 1 = l^r_{\nu-2}$, hence, $\Delta_{\nu-2}[x-1]$ is also mapped to r.

   If $\Delta_{\nu-1}[x-1]$ is mapped to $r-1$ then it holds $f^{r-1}_{\nu-1} \leq x-1 = l^{r-1}_{\nu-1}$. If the one extra cell of $\Delta_{\nu-1}$ ($\delta_{\nu-1} - \delta_{\nu-2} = 1$) is mapped to a processor $q > r-1$ then it holds $f^{r-1}_{\nu-2} = f^{r-1}_{\nu-1}$ and $l^{r-1}_{\nu-2} = l^{r-1}_{\nu-1}$. It follows that $f^{r-1}_{\nu-2} = f^{r-1}_{\nu-1} \leq x-1 = l^{r-1}_{\nu-2} = l^{r-1}_{\nu-1}$, hence, $\Delta_{\nu-2}[x-1]$ is also mapped to $r-1$. If the one extra cell of $\Delta_{\nu-1}$ ($\delta_{\nu-1} - \delta_{\nu-2} = 1$) is mapped to a processor $q \leq r-1$ then it holds $f^r_{\nu-2} = l^{r-1}_{\nu-1}$. It follows that $f^r_{\nu-2} = x-1$, hence, $\Delta_{\nu-2}[x-1]$ is mapped to r.

   (b) If the one extra cell of Δ_ν ($\delta_\nu - \delta_{\nu-1} = 1$) is mapped to a processor $q \leq r$, then, we also have to examine the case when $\Delta_{\nu-1}[x-1]$ is also mapped to r, such that $f^r_{\nu-1} \leq x-1 = l^r_{\nu-1}$ (see Lemma 1 Case 3). It follows that $\Delta_{\nu-1}[x]$ is mapped to $r+1$. Similarly, as in the case when $\Delta_{\nu-1}[x]$ is mapped to r, and $\Delta_{\nu-1}[x-1]$ is mapped either to r or to $r-1$ (see Case 1(a)), we obtain that $\Delta_{\nu-2}[x-1]$ is mapped either to r or to $r+1$.

2) $\delta_{\nu-1} - \delta_{\nu-2} = 1$ and $\delta_\nu - \delta_{\nu-1} = 0$: From Observations 1 and 3, $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x]$, $\mathsf{D}[i,j-1] = \Delta_{\nu-1}[x-1]$, and $\mathsf{D}[i-1,j-1] = \Delta_{\nu-2}[x-1]$.

   Since $\delta_\nu = \delta_{\nu-1}$, $\Delta_{\nu-1}[x]$ is also mapped to r (see Lemma 1 Case 1). It follows that $\Delta_{\nu-1}[x-1]$ is mapped either to r or to $r-1$. If $\Delta_{\nu-1}[x-1]$ is mapped to r then $\Delta_{\nu-2}[x-1]$ is also mapped to r (see Case 1(a)). If $\Delta_{\nu-1}[x-1]$ is mapped to $r-1$, then $\Delta_{\nu-2}[x-1]$ is mapped either to r or to $r-1$ (see Case 1(a)).

3) $\delta_{\nu-1} - \delta_{\nu-2} = 1$ and $\delta_\nu - \delta_{\nu-1} = -1$: From Observation 2 and 3, $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x+1]$, $\mathsf{D}[i,j+1] = \Delta_{\nu-1}[x]$, and $\mathsf{D}[i-1,j-1] = \Delta_{\nu-2}[x]$.

   (a) If the one extra cell of $\Delta_{\nu-1}$ ($\delta_\nu - \delta_{\nu-1} = -1$) is mapped to a processor $q > r$, then $\Delta_{\nu-1}[x]$ is also mapped to r (see Lemma 4 Case 2). It follows that $\Delta_{\nu-1}[x+1]$ is mapped either to r or to $r+1$. $\Delta_{\nu-2}[x]$ is mapped to r since $\Delta_\nu[x]$ is mapped to r and $\delta_\nu = \delta_{\nu-2}$.

   (b) If the one extra cell of $\Delta_{\nu-1}$ ($\delta_\nu - \delta_{\nu-1} = -1$) is mapped to a processor $q \leq r$, then $\Delta_{\nu-1}[x+1]$ is also mapped to r (see Lemma 4 Case 2). It follows that $\Delta_{\nu-1}[x]$ is mapped either to r or to $r-1$. $\Delta_{\nu-2}[x]$ is mapped to r since $\Delta_\nu[x]$ is mapped to r and $\delta_\nu = \delta_{\nu-2}$.

4) $\delta_{\nu-1} - \delta_{\nu-2} = 0$ and $\delta_\nu - \delta_{\nu-1} = -1$: From Observation 1 and 2, $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x+1]$, $\mathsf{D}[i,j+1] = \Delta_{\nu-1}[x]$, and $\mathsf{D}[i-1,j-1] = \Delta_{\nu-2}[x]$.

   (a) If the one extra cell of $\Delta_{\nu-1}$ ($\delta_\nu - \delta_{\nu-1} = -1$) is mapped to a processor

$q > r$, then $\Delta_{\nu-1}[x]$ is also mapped to r (see Lemma 4 Case 2). It follows that $\Delta_{\nu-1}[x+1]$ is mapped either to r or to $r+1$. $\Delta_{\nu-2}[x]$ is mapped to r since $\Delta_{\nu-1}[x]$ is mapped to r and $\delta_{\nu-2} = \delta_{\nu-1}$.

(b) If the one extra cell of $\Delta_{\nu-1}$ ($\delta_\nu - \delta_{\nu-1} = -1$) is mapped to a processor $q \leq r$, then $\Delta_{\nu-1}[x+1]$ is also mapped to r (see Lemma 4 Case 2). It follows that $\Delta_{\nu-1}[x]$ is mapped either to r or to $r-1$. If $\Delta_{\nu-1}[x]$ is mapped to r, then $\Delta_{\nu-2}[x]$ is also mapped to r since $\delta_{\nu-1} = \delta_{\nu-2}$. If $\Delta_{\nu-1}[x]$ is mapped to $r-1$, then $\Delta_{\nu-2}[x]$ is also mapped to $r-1$ since $\delta_{\nu-1} = \delta_{\nu-2}$.

5) $\delta_{\nu-1} - \delta_{\nu-2} = 0$ and $\delta_\nu - \delta_{\nu-1} = 0$: From Observation 1, $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x]$, $\mathsf{D}[i,j+1] = \Delta_{\nu-1}[x-1]$, and $\mathsf{D}[i-1,j+1] = \Delta_{\nu-2}[x-1]$.

Since $\delta_{\nu-1} = \delta_\nu$, $\Delta_{\nu-1}[x]$ is also mapped to r (see Lemma 1 Case 1). It follows that $\Delta_{\nu-1}[x-1]$ is mapped either to r or to $r-1$. If $\Delta_{\nu-1}[x-1]$ is mapped to r, then $\Delta_{\nu-2}[x-1]$ is also mapped to r since $\delta_{\nu-2} = \delta_{\nu-1}$. If $\Delta_{\nu-1}[x-1]$ is mapped to $r-1$, then $\Delta_{\nu-2}[x-1]$ is also mapped to $r-1$ since $\delta_{\nu-2} = \delta_{\nu-1}$.

6) $\delta_{\nu-1} - \delta_{\nu-2} = -1$ and $\delta_\nu - \delta_{\nu-1} = -1$: From Observation 2, $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[x+1]$, $\mathsf{D}[i,j+1] = \Delta_{\nu-1}[x]$, and $\mathsf{D}[i-1,j-1] = \Delta_{\nu-2}[x+1]$.

(a) If the one extra cell of $\Delta_{\nu-1}$ ($\delta_\nu - \delta_{\nu-1} = -1$) is mapped to a processor $q > r$, then $\Delta_{\nu-1}[x]$ is also mapped to r (see Lemma 4 Case 2). It follows that $\Delta_{\nu-1}[x+1]$ is allocated either to r or to $r+1$.

If $\Delta_{\nu-1}[x+1]$ is mapped to r then it holds $f^r_{\nu-1} < x+1 \leq l^r_{\nu-1}$. If the one extra cell of $\Delta_{\nu-2}$ ($\delta_{\nu-1} - \delta_{\nu-2} = -1$) is mapped to a processor $q > r$ then it holds $f^r_{\nu-2} = f^r_{\nu-1}$ and $l^r_{\nu-2} = l^r_{\nu-1}$. It follows that $f^r_{\nu-2} = f^r_{\nu-1} < x+1 \leq l^r_{\nu-1} = l^r_{\nu-2}$, hence, $\Delta_{\nu-2}[x+1]$ is also mapped to r. If the one extra cell of $\Delta_{\nu-2}$ ($\delta_{\nu-1} - \delta_{\nu-2} = -1$) is mapped to a processor $q = r$ then it holds $f^r_{\nu-2} = f^r_{\nu-1}$ and $l^r_{\nu-2} = l^r_{\nu-1}+1$. It follows that $f^r_{\nu-2} = f^r_{\nu-1} < x+1 \leq l^r_{\nu-2} = l^r_\nu + 1$, hence, $\Delta_{\nu-2}[x+1]$ is also mapped to r. If the one extra cell of $\Delta_{\nu-2}$ ($\delta_{\nu-1} - \delta_{\nu-2} = -1$) is mapped to a processor $q < r$ then it holds $f^r_{\nu-2} = f^r_{\nu-1}+1$ and $l^r_{\nu-2} = l^r_{\nu-1}+1$. It follows that $f^r_{\nu-2} = f^r_{\nu-1} + 1 < x+1 \leq l^r_\nu + 1 = l^r_{\nu-2}$, hence, $\Delta_{\nu-2}[x+1]$ is also mapped to r.

If $\Delta_{\nu-1}[x+1]$ is mapped to $r+1$ then it holds $f^{r+1}_{\nu-1} = x+1 \leq l^{r+1}_{\nu-1}$. If the one extra cell of $\Delta_{\nu-2}$ ($\delta_{\nu-1} - \delta_{\nu-2} = -1$) is mapped to a processor $q \geq r+1$ then it holds $f^{r+1}_{\nu-2} = f^{r+1}_{\nu-1}$ and $l^{r+1}_{\nu-2} \geq l^{r+1}_{\nu-1}$. It follows that $f^{r+1}_{\nu-2} = f^{r+1}_{\nu-1} = x+1 \leq l^{r+1}_{\nu-1} \leq l^{r+1}_{\nu-2}$, hence, $\Delta_{\nu-2}[x+1]$ is also mapped to $r+1$. If the one extra cell of $\Delta_{\nu-2}$ ($\delta_{\nu-1} - \delta_{\nu-2} = -1$) is mapped to a processor $q < r+1$ then it holds $l^r_{\nu-2} = f^{r+1}_{\nu-1}$. It follows that $l^r_{\nu-2} = x+1$, hence, $\Delta_{\nu-2}[x+1]$ is mapped to r.

(b) If the one extra cell of $\Delta_{\nu-1}$ ($\delta_\nu - \delta_{\nu-1} = -1$) is mapped to a processor $q \leq r$, then $\Delta_{\nu-1}[x+1]$ is also mapped to r (see Lemma 4 Case 2). It follows that $\Delta_{\nu-1}[x]$ is mapped either to r or to $r-1$. Similarly as in the case when $\Delta_{\nu-1}[x]$ is mapped to r, and $\Delta_{\nu-1}[x+1]$ is mapped either to r or to $r+1$ (see Case 6(a)), we obtain that $\Delta_{\nu-2}[x+1]$ is mapped either to r or to $r-1$.

■

Lemma 3: For the computation of Δ_ν, each processor only needs to send the content of a number of cells of $\Delta_{\nu-1}$ to neighbouring processors.

Proof: Let us assume that for the computation of cell $\mathsf{D}[i,j] = \Delta_\nu[x]$ mapped to a processor r, processor $r-1$ (resp. $r+1$ by Lemma 2) needs to send the content of cell $\mathsf{D}[i-1,j-1]$ of $\Delta_{\nu-2}$ to r. This implies that, for the calculation of $\Delta_{\nu-1}$, it was not necessary for processor $r-1$ (resp. $r+1$) to send the calculated content of $\mathsf{D}[i-1,j-1]$ to r. This further implies that neither cell $\mathsf{D}[i,j-1]$ nor $\mathsf{D}[i-1,j]$ of $\Delta_{\nu-1}$ were mapped to r. Hence, since cell $\mathsf{D}[i-1,j-1]$ of $\Delta_{\nu-2}$ was mapped to $r-1$ (resp. $r+1$), then both of these cells must also have been mapped to $r-1$ (resp. $r+1$) by Lemma 2. But if those cells were mapped to $r-1$ (resp. $r+1$) for calculating $\Delta_{\nu-1}$, it is definitely the case that cell $\mathsf{D}[i,j]$ of Δ_ν will also be mapped to $r-1$ by Lemma 1. Thus, we obtain a contradiction, and the lemma holds. ■

Corollary 1: For the computation of Δ_ν, each processor only needs to receive the content of

a number of cells of $\Delta_{\nu-1}$ from neighbouring processors.

Proof: Direct consequence of Lemma 3. ■

Lemma 4: For the computation of Δ_ν, processor r needs to receive the contents of at most two cells, $\Delta_{\nu-1}[l_{\nu-1}^{r-1}]$ and $\Delta_{\nu-1}[f_{\nu-1}^{r+1}]$, from neighbouring processors $r-1$ and $r+1$, respectively.

Proof: Without loss of generality, assume that cell $\mathsf{D}[i,j]$ is mapped to processor r. From the proof of Lemma 2, we distinguish among the following cases (see Fig. 2 in this regard):

1) $\mathsf{D}[i-1,j]$, $\mathsf{D}[i,j-1]$, and $\mathsf{D}[i-1,j-1]$ are mapped to r.
2) $\mathsf{D}[i-1,j]$ is mapped to r, and $\mathsf{D}[i,j-1]$ and $\mathsf{D}[i-1,j-1]$ are mapped to $r-1$.
3) $\mathsf{D}[i-1,j]$ and $\mathsf{D}[i-1,j-1]$ are mapped to r, and $\mathsf{D}[i,j-1]$ is mapped to $r-1$.
4) $\mathsf{D}[i,j-1]$ is mapped to r, and $\mathsf{D}[i-1,j]$ and $\mathsf{D}[i-1,j-1]$ are mapped to $r+1$.
5) $\mathsf{D}[i,j-1]$ and $\mathsf{D}[i-1,j-1]$ are mapped to r, and $\mathsf{D}[i-1,j]$ is mapped to $r+1$.

We eliminate Case 1 since all cells are mapped to r. By Lemma 3 and Corollary 1, the problem is reduced to the following two cases:

- $\mathsf{D}[i-1,j]$ is mapped to r and $\mathsf{D}[i,j-1]$ is mapped to $r-1$. Clearly, $r-1$ needs to send $\mathsf{D}[i,j-1] = \Delta_{\nu-1}[l_{\nu-1}^{r-1}]$ to r for the computation of $\mathsf{D}[i,j]$.
- $\mathsf{D}[i,j-1]$ is mapped to r and $\mathsf{D}[i-1,j]$ is mapped to $r+1$. Clearly, $r+1$ needs to send $\mathsf{D}[i-1,j] = \Delta_{\nu-1}[f_{\nu-1}^{r+1}]$ to r for the computation of $\mathsf{D}[i,j]$.

■

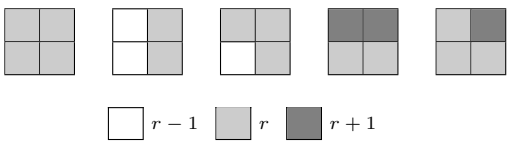


Figure 2. The five possible mappings of the set of cells $\{\mathsf{D}[i,j], \mathsf{D}[i-1,j], \mathsf{D}[i,j-1], \mathsf{D}[i-1,j-1]\}$ to processors $r-1$, r, and $r+1$, given that $\mathsf{D}[i,j]$ is mapped to r

Corollary 2: For the computation of Δ_ν, processor r needs to send the contents of at most two cells, $\Delta_{\nu-1}[l_{\nu-1}^{r-1}]$ and $\Delta_{\nu-1}[f_{\nu-1}^{r+1}]$, to neighbouring processors $r-1$ and $r+1$, respectively.

Proof: Direct consequence of Lemma 4.

■

D. Processor communication

Algorithm PP-COMM provides the point-to-point communication between r, $r-1$, and $r+1$ at step ν. It takes as input the size n of the text, the diagonal array Δ_ν, the indices f_ν^r and l_ν^r of the first and the last cell of Δ_ν mapped to processor r, respectively. It also needs as input the indices $f_{\nu+1}^r$ and $l_{\nu+1}^r$ of the first and the last cell of Δ_ν mapped to processor r, respectively.

By Observations 1–3, we obtain two cases: the case where $\nu < n$ (Algorithm PP-COMM line 1) by Observations 1 and 3 ($\delta_\nu - \delta_{\nu-1} \in \{0,1\}$), and the case where $\nu \geq n$ (Algorithm PP-COMM line 10) by Observation 2 ($\delta_\nu - \delta_{\nu-1} = -1$). By Corollary 2, for the computation of $\Delta_{\nu+1}$, processor r needs to send the contents of at most two cells, $\Delta_\nu[f_\nu^r]$ and $\Delta_\nu[l_\nu^r]$, to neighbouring processors $r-1$ (Algorithm PP-COMM lines 5 and 14) and $r+1$ (Algorithm PP-COMM lines 3 and 12), respectively. Hence, the data exchange between the processors in Algorithm PP-COMM involves a constant number of point-to-point message transfers at each step. Since the number of cells to be computed is $\Theta(nm)$, the number of parallel steps is $\Theta(n)$—exactly $n+m+1$ diagonals—and each processor is assigned the computation of $\mathcal{O}(m/p)$ cells—the size of each diagonal is $\mathcal{O}(m)$—at each parallel step, we obtain the following result.

Theorem 1: The proposed parallel system is cost-optimal.

ALGORITHM PP-COMM(r, n, ν, Δ_ν, f_ν^r, l_ν^r, $f_{\nu+1}^r$, $l_{\nu+1}^r$)

```
1:  if ν < n then
2:     if (f_{ν+1}^r ≤ l_ν^r + 1 ≤ l_{ν+1}^r) = false then
3:        r sends Δ_ν[l_ν^r] to r + 1
4:     if (f_{ν+1}^r ≤ f_ν^r ≤ l_{ν+1}^r) = false then
5:        r sends Δ_ν[f_ν^r] to r − 1
6:     if (f_ν^r ≤ f_{ν+1}^r − 1 ≤ l_ν^r) = false then
7:        r receives Δ_ν[f_ν^r] from r − 1
8:     if (f_ν^r ≤ l_{ν+1}^r ≤ l_ν^r) = false then
9:        r receives Δ_ν[l_ν^r] from r + 1
10: else
11:    if (f_{ν+1}^r ≤ l_ν^r ≤ l_{ν+1}^r) = false then
12:       r sends Δ_ν[l_ν^r] to r + 1
13:    if (f_{ν+1}^r ≤ f_ν^r − 1 ≤ l_{ν+1}^r) = false then
14:       r sends Δ_ν[f_ν^r] to r − 1
15:    if (f_ν^r ≤ l_{ν+1}^r + 1 ≤ l_ν^r) = false then
16:       r receives Δ_ν[l_ν^r] from r + 1
17:    if (f_ν^r ≤ f_{ν+1}^r ≤ l_ν^r) = false then
18:       r receives Δ_ν[f_ν^r] from r − 1
```

Furthermore, if $\mathsf{D}[i-1,j] = \Delta_\nu[l_\nu^r]$ is the last cell mapped to r in step ν, then, by checking if $\mathsf{D}[i,j]$ is mapped to r (Algorithm PP-COMM lines 2 and 11), we know whether r should send the

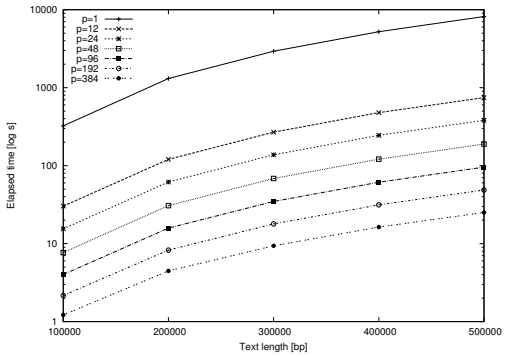
content of $\mathsf{D}[i-1,j]$ to $r+1$ or not. Analogously, if $\mathsf{D}[i,j-1] = \Delta_\nu[f^r_\nu]$ is the first cell mapped to r in step ν, then, by checking if $\mathsf{D}[i,j]$ is mapped to r (Algorithm PP-COMM lines 4 and 13), we know whether r should send the content of $\mathsf{D}[i,j-1]$ to $r-1$ or not. Hence we obtain the following result:

Theorem 2: The point-to-point communication in Algorithm PP-COMM is minimal for the proposed parallel system.

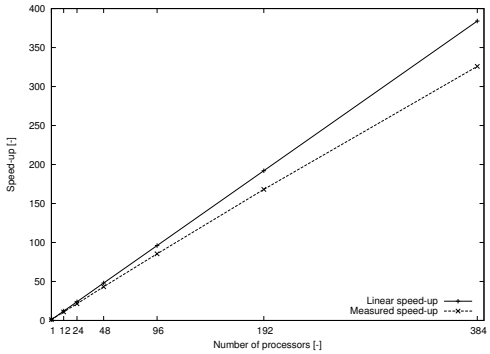
III. EXPERIMENTAL RESULTS

In order to evaluate the parallel efficiency of the proposed parallel system, we implemented algorithm MAXSHIFT in the C programming language, and parallelised it using the *Message Passing Interface* (MPI), which supports the MPP paradigm, implementing programme OPASM, which uses the proposed optimised parallel system. DNA sequences of the single chromosome of *Escherichia coli* str. K-12 substr. MG1655, obtained from GenBank [2], were used as input for the programme. Experimental results regarding the elapsed time of OPASM for different problem sizes are provided in Fig. 3a. The measured relative speed-up of OPASM is illustrated in Fig. 3b. The relative speed-up was calculated as the ratio of the runtime of the parallel algorithm on 1 processor to the runtime of the parallel algorithm on p processors—the runtime of the parallel algorithm on 1 processor was identical to the sequential one. The experiments highlight the good scalability of OPASM, even for small problem sizes. When increasing the problem size, OPASM achieves linear speed-ups and confirms our theoretical results.

In order to relate the amount of point-to-point communication of OPASM to other parallel systems, we also parallelised algorithm MAXSHIFT with MPI, implementing programme PASM, which uses the parallel system introduced in [6]. The only difference from OPASM is that processor communication in PASM involves point-to-point boundary cell swaps between all neighbouring processors at each parallel step. Two different versions were used for each of the two programmes: one with blocking send operations (MPI_Send); and one with blocking synchronous send operations (MPI_Ssend). We measured the elapsed point-to-point communication time of PASM and OPASM (Algorithm PP-COMM) for different problem sizes by commenting out the computation phase of both programmes. The main contribution of this report becomes apparent from Fig. 4: (i) the time for point-to-point communication of OPASM



(a) Elapsed time of OPASM

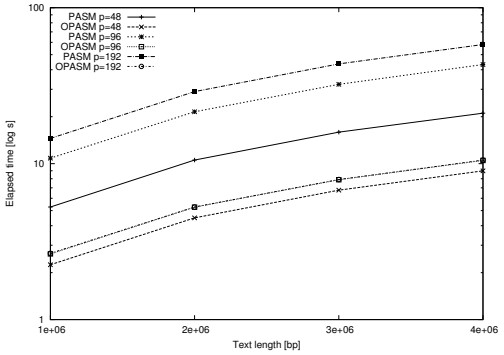


(b) Measured speed-up of OPASM

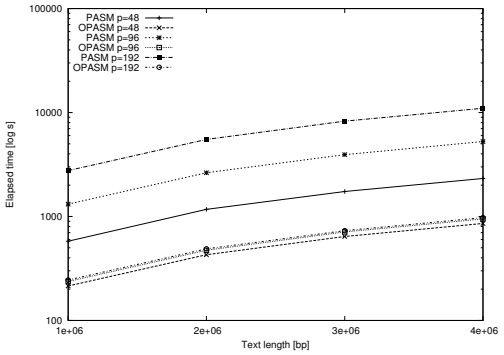
Figure 3. Elapsed time of OPASM for $n = m$, and measured speed-up of OPASM for $n = m = 500$K

is reduced up to five times with blocking send operations (Fig. 4a) and up to ten times with blocking synchronous send operations (Fig. 4b); (ii) at a certain point, the amount of point-to-point communication of OPASM becomes essentially constant (note that, the curves of OPASM on 96 and 192 processors coincide both, in Fig. 4a and in Fig. 4b).

In order to relate the overall performance of OPASM to other parallel systems, we parallelised algorithm MAXSHIFT with MPI, implementing programme EDM, which uses the classic parallel system introduced by Edmiston *et al.* Since latency of communication t_s is usually much worse than communication bandwidth t_w, the system by Edmiston *et al.* strives to minimise the number of point-to-point communications by sending the content of $\Theta(m/p)$ cells $\Theta(p)$ times—while the proposed parallel system sends the content of $\mathcal{O}(1)$ cells $\Theta(n)$ times. In the system proposed by Edmiston *et al.*, the dynamic-programming matrix is decomposed into a $p \times p$ matrix of submatrices. Each processor q, $0 \leq q < p$, is assigned a row of p submatrices, and processor q, $q > 0$, starts computing the first submatrix of its row



(a) Blocking send



(b) Blocking synchronous send

Figure 4. Elapsed time of the point-to-point communication of PASM and OPASM with blocking send operation and blocking synchronous send operation

only when processor $q-1$ has already computed the first submatrix of its row, resulting in a total of $2p-1$ parallel steps. When q, $q < p-1$, finishes the computation of a submatrix, it must send $\Theta(m/p)$ cells (the last row) of the submatrix that has been computed to processor $q+1$. Hence, in total, this system communicates only $\Theta(p)$ times. This approach exhibits two disadvantages: (i) the matrix size in either dimension must be a multiple of the number of processors; (ii) since processor q does not start computing until step q, this generates load imbalance, in particular for large submatrices.

We used the aforementioned dataset as input for EDM and OPASM. The elapsed times of EDM and OPASM are provided in Fig. 5. These experimental results confirm our theoretical findings: OPASM is more efficient than EDM for small processor counts and problems of large size. However, as we increase the number of processors, the size of submatrices computed by EDM decreases, and the two systems exhibit comparable performance.

Finally, we also implemented programme PRE, a performance predictor for EDM and OPASM. PRE

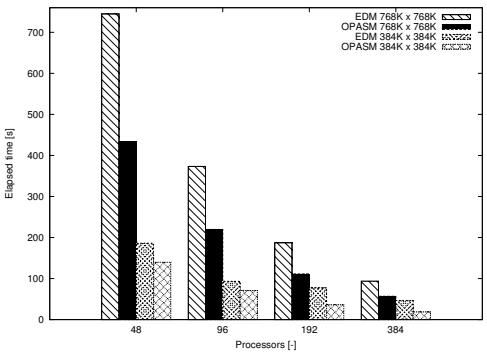


Figure 5. Elapsed time of EDM and OPASM for $n = m = 384$K and $n = m = 768$K

uses the theoretical results presented in [1] and in Section II to accurately and efficiently predict the performance of EDM and OPASM, respectively. The performance of EDM can be predicted by measuring the sum of the processor idle time, the sum of per submatrix calculation time, and the sum of communication times for $\Theta(m/p)$ cells. The performance of OPASM can be predicted by measuring the sum of calculation time per anti-diagonal, and the sum of communication costs times for exchanging $\mathcal{O}(1)$ (at most two) cells per anti-diagonal. Experimental results for the performance prediction of EDM and OPASM are provided in Fig. 6. The results demonstrate that the predictor is highly accurate. PRE only takes a few seconds to execute. It performs a number of iterations to measure the aforementioned average time costs. Users can chose to increase the number of iterations in the predictor to further increase its accuracy at the expense of longer predictor runtimes.

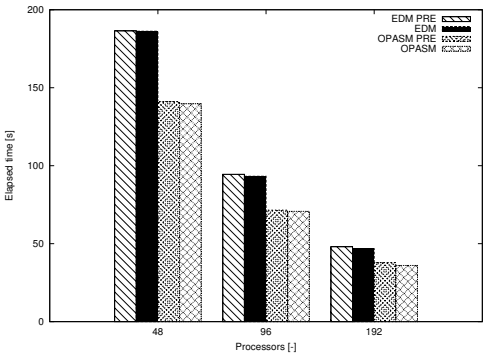


Figure 6. Performance prediction for $n = m = 384$K

The experiments were conducted on an Infiniband-connected cluster using 1 up to 384 2.4 GHz AMD 6136 processors under the Linux operating system. The implementations of `OPASM`, `EDM`, and `PRE` are available at a website (http://www.exelixis-lab.org/solon) under the terms of the GNU General Public License.

IV. Conclusion

In this report, we presented a new cost-optimal parallel system based on MPP paradigm for efficient parallelisation of approximate string-matching algorithms. From a practical point of view, our experimental results show that the proposed system reduces the time for point-to-point communication up to five times compared to the one introduced in [6], and that it can match, and also outperform, the classic parallel system introduced by Edmiston *et al.* [1]. Finally, we designed and implemented a performance predictor that can be used to accurately and efficiently predict the performance of these systems given a specific input dataset. We also tested other dynamic-programming kernels with the presented parallel systems obtaining similar results. Our immediate goal is to design, implement, and test a new hybrid system, between the one proposed here and the one proposed by Edmiston *et al.*, that will make use of the proposed system only for the load imbalanced parallel steps of the system proposed by Edmiston *et al.*

References

[1] E. Edmiston, N. Core, J. Saltz, and R. Smith. Parallel processing of biological sequence comparison algorithms. *International Journal of Parallel Programming*, 17:259–275, 1988.

[2] GenBank. http://www.ncbi.nlm.nih.gov/genbank/, June 2012.

[3] P. Heckel. A technique for isolating differences between files. *Communications of the ACM*, 21(4):264–268, 1978.

[4] X. Huang. A space-efficient parallel sequence comparison algorithm for a message-passing multiprocessor. *International Journal of Parallel Programming*, 18(3):223–239, 1990.

[5] C. Iliopoulos, L. Mouchard, and Y. Pinzon. The Max-Shift algorithm for approximate string matching. In G. Brodal, D. Frigioni, and A. Marchetti-Spaccamela, editors, *Proceedings of the fifth International Workshop on Algorithm Engineering (WAE 2001)*, volume 2141 of *Lecture Notes in Computer Science*, pages 13–25, Denmark, 2001. Springer.

[6] C. S. Iliopoulos, L. Mouchard, and S. P. Pissis. A parallel algorithm for the fixed-length approximate string matching problem for high throughput sequencing technologies. In B. Chapman, F. Desprez, G. R. Joubert, A. Lichnewsky, F. Peters, and T. Priol, editors, *Proceedings of the International Conference on Parallel Computing (PARCO 2009)*, volume 19 of *Advances in Parallel Computing*, pages 150–157, France, 2010. IOS Press.

[7] G. M. Landau and U. Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989.

[8] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, Soviet Physics Doklady, 1966.

[9] J. L. Peterson. Computer programs for detecting and correcting spelling errors. *Communications of the ACM*, 23(12):676–687, 1980.

[10] S. Rajko and S. Aluru. Space and time optimal parallel sequence alignments. *IEEE Transactions on Parallel and Distributed Systems*, 15:1070–1081, 2004.

[11] P. H. Sellers. On the theory and computation of evolutionary distances. *SIAM Journal on Applied Mathematics*, 26(4):787–793, 1974.

[12] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[13] M. S. Waterman and T. F. Smith. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.