CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Missouri University of Science and Technology (Missouri S&T): Scholars' Mine

Missouri University of Science and Technology
Scholars' Mine

Computer Science Faculty Research & Creative Works

Computer Science

01 Jan 1997

RMESH Algorithms for Parallel String Matching

Hsi-Chieh Lee

Fikret Erçal
Missouri University of Science and Technology, ercal@mst.edu

Follow this and additional works at: https://scholarsmine.mst.edu/comsci_facwork

Part of the Computer Sciences Commons

Recommended Citation

H. Lee and F. Erçal, "RMESH Algorithms for Parallel String Matching," *Proceedings of the 3rd International Symposium on Parallel Architectures, Algorithms, and Networks, 1997*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1997.
The definitive version is available at https://doi.org/10.1109/ISPAN.1997.645099

This Article - Conference proceedings is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Computer Science Faculty Research & Creative Works by an authorized administrator of Scholars' Mine. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact scholarsmine@mst.edu.

RMESH Algorithms For Parallel String Matching

Hsi-Chieh Lee†, Fikret Ercal‡

Abstract— **String matching problem received much attention over the years due to its importance in various applications such as text/file comparison, DNA sequencing, search engines, and spelling correction. Especially with the introduction of search engines dealing with tremendous amount of textual information presented on the world wide web and the research on DNA sequencing, this problem deserves special attention and any algorithmic or hardware improvements to speed up the process will benefit these important applications.**

In this paper, we present three algorithms for string matching on reconfigurable mesh architectures. Given a text T of length n and a pattern P of length m, the first algorithm finds the exact matching between T and P in O(1) time on a 2-dimensional RMESH of size $(n-m+1)\times m$. The second algorithm finds the approximate matching between T and P in O(k) time on a 2D RMESH, where k is the maximum edit distance between T and P. The third algorithm allows only the replacement operation in the calculation of the edit distance and finds an approximate matching between T and P in constant-time on a 3D RMESH.

Keywords— **String Matching, Approximate String Matching, Reconfigurable Mesh Architecture, Parallel Algorithms, RMESH**

I. Introduction

The exact string matching is the problem of detecting the occurrence of a particular substring P, called a pattern, in another string T, called the text. It most commonly arises in text processing and pattern recognition. The well-known Knuth-Morris-Pratt algorithm [1] has a time complexity of $O(n+m)$. Boyer and Moore [2] developed an algorithm that exhibits better performance in practice when the pattern length is relatively long [3].

The *k-differences approximate string matching* problem is to find the occurrences of substrings on a text string T whose edit distance from a pattern string P is less than k. The *edit distance* between strings S_1 and S_2, $d(S_1, S_2)$, is defined as the minimum number of edit steps in converting S_1 to S_2 using the following three kinds of edit operations:

- Delete an element from S_1.
- Insert an element to S_1.
- Replace an element of S_1 with another element.

The definition of edit distance can also be generalized by assigning different costs for different operations or different alphabets.

The k-differences approximate string matching problem has received much attention over the last few decades due to its importance on various areas, such as, text/file comparison, molecular biology, error/spelling correction and many others. A dynamic programming algorithm for computing the edit distance between two strings was first developed by Needleman and Wunsch [4] and variations were then independently developed by many researchers. Sankoff and Kruskal's book [5] provides references to related work and describes the variations of the dynamic programming algorithms. The sequential dynamic programming algorithm requires time and space complexity of O(mn) where m is the length of the pattern and n is the length of the text. A sequential algorithm of complexity O(nk) that derived from Ukkonen's work [6] was developed by Landau and Vishkin [7]. They also designed an O(log(m)+k) parallel algorithm using n processors. However, Galil and Park [8] pointed out that their algorithm uses suffix trees which required $O(\log n)$ time for construction on n processors [9]. To achieve speedup without first constructing the suffix trees, Jiang and Wright [10] designed an O(k) parallel algorithm using the priority CRCW-PRAM parallel model. Their algorithm was based on a variation of Ukkonen's algorithm [6], [7].

In this paper, we first describe the reconfigurable mesh architecture (RMESH) model used for solving various string matching problems (Section I-A). Then, section II-A presents an O(1) time algorithm for exact string matching between a text T and a pattern P using a 2-dimensional RMESH architecture. The algorithm to find the approximate matching between strings T and P is presented in Section II-B. This algorithm is an efficient adaptation of Ukkonen's algorithm [6] to reconfigurable mesh architectures and it has a time complexity of O(k) on a 2-D RMESH, where k is the maximum edit distance between T and P. A third algorithm is presented in Section II-C which allows only the replacement operation in the computation of the edit distance. This algorithm runs in O(1) time on a 3-dimensional RMESH architecture.

A. RMESH Model

There are various but similar reconfigurable mesh (RMESH) architectures proposed in the literature. First two algorithms proposed in this paper use a 2D RMESH model [11] while the third algorithm requires a 3D RMESH. Some important features of this model are as follows:

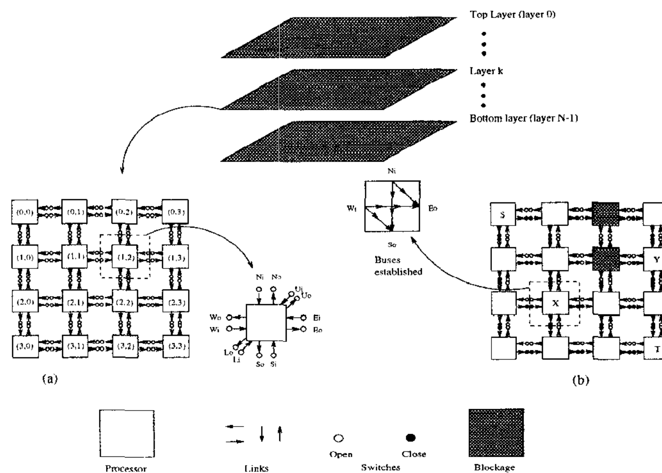


Fig. 1. A 3D RMESH with 4 x 4 x N processors

1. A 3D RMESH is an N×N×N array of PEs connected in a standard mesh topology. The index of a PE is a 3-tuple (i, j, k), where $0 \leq i \leq N-1$ is the row index,

†Hsi-Chieh Lee is with the Information Management Department, Yuan-Ze University, 135 Far Eastern Rd., Chung-Li, Taiwan 326 Republic of China. E-mail: imhlee@saturn.yzu.edu.tw .

‡Fikret Ercal is with the Computer Science Department, University of Missour-Rolla, Rolla, MO 65401, USA. E-mail: ercal@cs.umr.edu

1087-4089/97 $10.00 © 1997 IEEE

$0 \leq j \leq N-1$ is the column index, and $0 \leq k \leq N-1$ is the layer index.

2. Each of the PEs has six sets of ports that is connected to its six nearest neighbors in the plane as shown in Figure 1. $\{E_i, S_i, W_i, N_i, U_i, L_i\}$ stands for the set of input ports from the east, south, west, north, upper, and lower neighbors, respectively. $\{E_o, S_o, W_o, N_o, U_o, L_o\}$ is the list of output ports to the east, south, west, north, upper, and lower neighbors, respectively. Within a PE, any combination of the input ports can be connected to any combination of the output ports. This feature allows multiple busses to pass through a particular PE.
3. Output ports cannot read the bus and input ports cannot write onto the bus. Data between PEs can move only from output ports into input ports.
4. A link is established if and only if the input (output) port and the corresponding output (input) port are both closed at the same time.
5. Only one processor in a subnet can write data on the bus at any given time. In unit time, data put onto a directional bus can be read by every PE who is reachable from the source PE by following the directional links that define the subnet.

The list of ports connected to the same bus within a PE will be indicated inside { } parenthesis. For example, when we say, "a processor sets its $\{X_i, Y_i, Z_o\}$ switches", it means that the input ports, X_i and Y_i, and the output port, Z_o are enabled and connected to the same bus within a PE. Figure 1(b) illustrates a case where processors set their $\{N_i, W_i, S_o, E_o\}$ switches.

II. String Matching Algorithms on an RMESH

A. Exact String Matching

Given a text T $(t_0, t_1, ..., t_{n-1})$ of length n and a pattern P $(p_0, p_1, ..., p_{m-1})$ of length m, we present a constant time algorithm to find all the occurrences of P in T on the RMESH architecture described in section I-A. Without loss of generality, let us assume that elements of pattern P initially reside on processors at row 0, one element per PE, i.e., p_0 on $PE_{0,0}$, p_1 on $PE_{0,1}$, etc. Similarly, text T initially resides at column 0 (see Figure 2(a)).

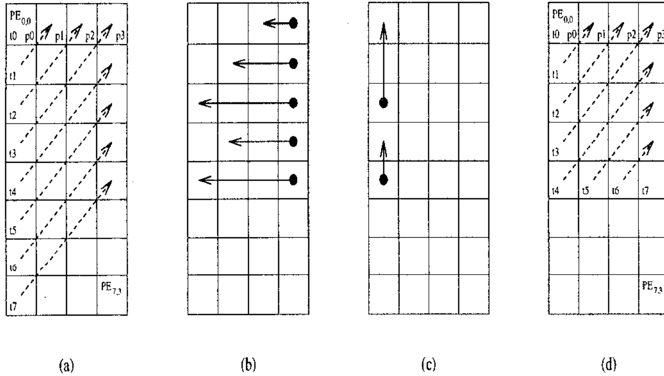


Fig. 2. Illustration of Exact String Matching Algorithm

Algorithm II.1 (Exact String Matching Algorithm)
{* Initially, all input/output ports are disabled *}

- Step 1: Form n diagonal buses as shown in Figure 2(a).
- Step 2: $PE_{i,0}$, $\forall i, 0 \leq i \leq n-1$, broadcasts t_i on its corresponding diagonal bus.
- Step 3: Broadcast p_j's along the column buses. At this point, $PE_{i,j}$ holds both p_j and t_{i+j}, $\forall i, j, 0 \leq i \leq n-m$ and $0 \leq j \leq m-1$.
- Step 4: Processors at each of the first $(n-m+1)$ rows form special broadcast buses to determine whether there is a perfect match or not along their particular rows, as follows:
  - If $p_j = t_{i+j}$, enable switches $\{E_i, W_o\}$. Otherwise, leave all the switches disabled.
- Step 5: $PE_{i,m-1}$, $\forall i, 0 \leq i \leq n-m$ sends a special signal '*' towards left and $PE_{i,0}$ reads the bus. Those PEs which receive the special signal '*' indicate a match starting at the string position that they hold.
- Step 6: If we want to detect the existence of at least one exact match, the algorithm continues with the following steps:
  - Form a special column bus on column 0 according to the following rules: If $PE_{i,0}$ receives the special signal '*' in Step 5, it enables only switch $\{N_o\}$ ($\{S_i\}$ remains disabled) and it is marked as "selected". All others enable both switches $\{N_o, S_i\}$ (Figure 2(c)).
  - Any selected $PE_{i,0}$, $\forall i, 0 \leq i \leq n-m$ sends its id on the bus. If $PE_{0,0}$ receives a value q, then there is an exact match starting at t_q, otherwise, there is no exact match of P anywhere in T.

The number of processors could be reduced from O(mn) to O(m(n-m+1)) by changing the initial distribution of the original string as shown in Figure 2(d)).

B. Approximate String Matching I

The *k-differences approximate string matching* problem is to find the occurrences of substrings of a text string T whose edit distance from a pattern string P is less than k.

Given a text T $(t_1, t_2, ..., t_n)$ of length n and a pattern P $(p_1, p_2, ..., p_m)$ of length m, the well-known dynamic programming algorithm that constructs a D-matrix to find the approximate matching between T and P with at most k differences is given below. This algorithm has a sequential time complexity of O(mn):

```
    Initialization:
        D_{0,j} = 0, ∀j where 0 ≤ j ≤ n
        D_{i,0} = i, ∀i where 0 ≤ i ≤ m
  for i=1 to m do
        for j = 1 to n do
        if(p_i == t_j) D_{i,j} = min(D_{i-1,j} + 1, D_{i,j-1} + 1, D_{i-1,j-1})
        else D_{i,j} = min(D_{i-1,j}+1, D_{i,j-1}+1, D_{i-1,j-1}+1)
```

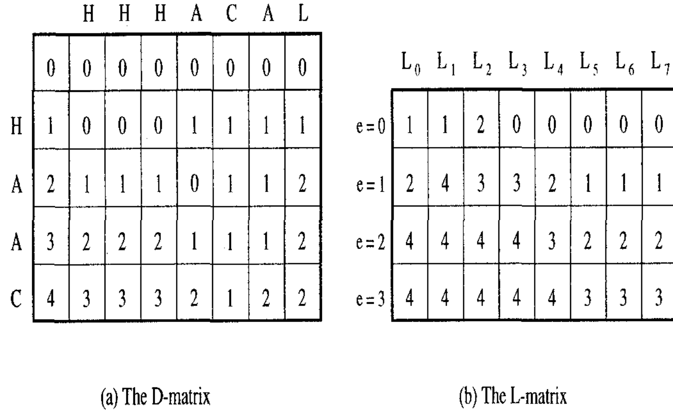
Figure 3(a) shows the corresponding D matrix for string T = "HHHACAL" and P = "HAAC".

An alternative dynamic programming algorithm achieves the same results by constructing an L-matrix. The advantage of this algorithm is that it is suitable for parallel implementation. It is originally due to Ukkonen [6]. Based on the same algorithm, Landau and Vishkin [7] presented an $O((logm)+k)$ time algorithm on the CRCW-PRAM model and Jiang and Wright[10] developed several algorithms to solve the approximate string matching problem in O(k) time on CRCW-PRAM model. Code for this algorithm is given below.

```
    Initialization:
        L_{d,-1} = -1, ∀d where 0 ≤ d ≤ n
        L_{d,|d|-1} = |d| - 1, L_{d,|d|-2} = |d| - 2, ∀d where -(k+1) ≤ d ≤ -1
        L_{n+1,e} = -1, ∀e where -1 ≤ e ≤ k
  for e=0 to k do
        for d = -e to n do
            row = max(L_{d-1,e-1}, L_{d,e-1}+1, L_{d+1,e-1}+1)
            row = min(row, m)
```

while $row < m$ and $p_{row+1} == t_{row+1+d}$
$row = row + 1$
$L_{d,e} = row$
if $L_{d,e} == m$
print "There is an approximate matching ending at t_{d+m}."

The resulting L-matrix for the previous example is given in Figure 3(b).



		H	H	H	A	C	A	L
	0	0	0	0	0	0	0	0
H	1	0	0	0	1	1	1	1
A	2	1	1	1	0	1	1	2
A	3	2	2	2	1	1	1	2
C	4	3	3	3	2	1	2	2

(a) The D-matrix

	L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7
e=0	1	1	2	0	0	0	0	0
e=1	2	4	3	3	2	1	1	1
e=2	4	4	4	4	3	2	2	2
e=3	4	4	4	4	4	3	3	3

(b) The L-matrix

Fig. 3. The D-matrix and L-matrix for approximate string matching

In this section, we present an algorithm which is an efficient adaptation of the above algorithm to the RMESH model. Our algorithm runs in O(k) time where k is the maximum number of differences allowed between the two strings. Note that this algorithm can also be used for exact string matching. It is considered as a special case when the difference k equals 0. In such a case, the time complexity of the algorithm is reduced to O(1) which is the same as Algorithm II.1.

Without loss of generality, let's first assume that $PE_{i,j}$ contains p_i, t_j and a variable $MATCH_{i,j}$ which equals 1 if $p_i = t_j$ and 0 otherwise. Note that switches will be enabled explicitly, i.e., every step in the algorithms given below will assume that all input/output ports are disabled initially.

Algorithm II.2 (Approximate String Matching Algorithm I)

{* Initially, all input/output ports are disabled. When not specified explicitly, to form special buses means to establish diagonal buses as shown in Figure 4(a). *}

- Step 1: [Initialization] $PE_{0,j}$ holds all the $L_{j,*}$ values initially:
  $L_{j,-1} = -1, \forall j$ where $0 \leq j \leq n$
  $L_{j,|j|-1} = |j| - 1$, $L_{j,|j|-2} = |j| - 2, \forall j$ where $-(k+1) \leq j \leq -1$
  $L_{n+1,e} = -1, \forall e$ where $-1 \leq e \leq k$
- Repeat Steps 2-6, for $e = 0, 1, ..., k$
- Step 2: [Neighbor communications] $PE_{0,j}$ reads $L_{j-1,e-1}$ and $L_{j+1,e-1}$ from its east and west neighbors, for $0 \leq j \leq n$.
- Step 3: [Broadcast the initial $L_{j,e}$ values] $PE_{0,j}$, $\forall j$ where $0 \leq j \leq n$, computes $L_{j,e} = max((L_{j-1,e-1}), (L_{j,e-1} + 1), (L_{j+1,e-1} + 1))$ and broadcasts the result on its corresponding subnet (diagonal bus, see Figure 4(a)).
- Step 4: [Each PE checks its status and set switches] Form special buses according to the following rules (See Figure 4):
  - $PE_{i,j}$ enables its southeast port, for $0 \leq i < L_{j,e}$ and $0 \leq j \leq n$.
  - $PE_{i,j}$ enables its southeast port if $MATCH_{i+1,j+1} = 1$ and $L_{j,e} \leq i \leq m$ (neighbor communication is required to get the $MATCH_{i+1,j+1}$ value).

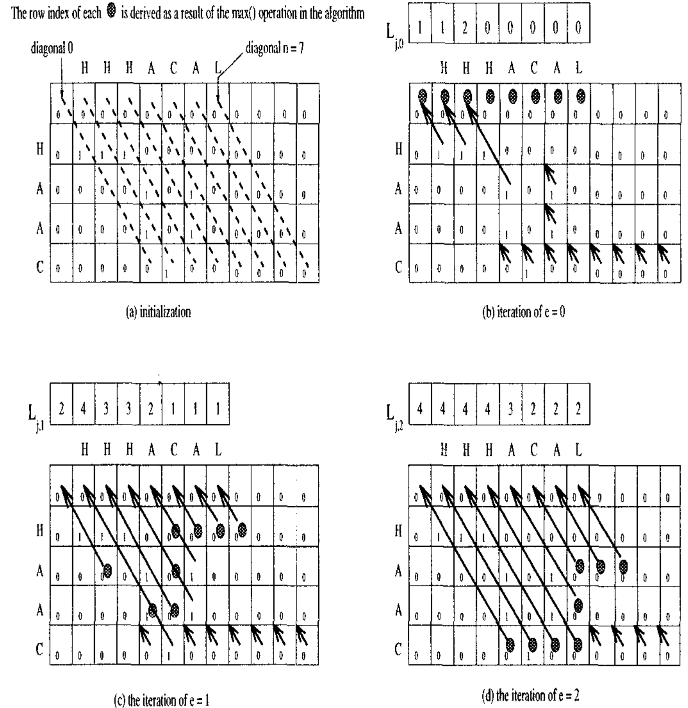


Fig. 4. Illustration of Approximate String Matching Algorithm I

- Step 5: [Each PE checks its status and decides if it is allowed to broadcast] Any $PE_{i,j}$ that satisfies at least one of the following conditions sends its row index i along the corresponding subnet and $PE_{0,j}$ records the received value as $L_{j,e}$. (for each $PE_{i,j}$, neighbor communication is required to get the $MATCH_{i+1,j+1}$ value)
  1. $(i = L_{j,e})$ and $(MATCH_{i,j} = 0)$ and $(MATCH_{i+1,j+1} = 0)$.
  2. $(i > L_{j,e})$ and $(MATCH_{i,j} = 1)$ and $(MATCH_{i+1,j+1} = 0)$.
  3. $(i = m)$.
- Step 6: [PEs on row 0 check if there is a match] $PE_{0,j}$ checks the $L_{j,e}$ value. If $L_{j,e} = m$, quit and report that an approximate string is found with a difference of e.

This algorithm uses an RMESH of size $(m+n+1)(m+1)$ where m and n are the lengths of the pattern P and the text T respectively. Since repeat loop executes at most k times, it achieves a time complexity of $O(k)$ where k is the maximum difference allowed between P and T.

C. Approximate String Matching II

In some applications, we are interested in detecting the occurrences of a particular substring P in another string T where the number of mismatches between the two strings is at most k (k $\leq$ m). This is a special case of the *k-differences approximate string matching* problem described above. Here, only the replacement operation is considered in calculating the edit distance. If we assume that $(n - m + 1) \geq m \geq 16$, then this algorithm uses a 3-dimensional RMESH of size $(n - m + 1) \times m \times (n - m + 1)$, where m and n are the lengths of the pattern (P) and the text (T) respectively.

Algorithm II.3 (Approximate String Matching Algorithm II)

{* Assume that $(n - m + 1) \geq m \geq 16$ and initially, all input/output ports are disabled *}

- Steps 1-3: Steps 1-3 of Algorithm II.1 (*Exact String Matching Algorithm*)
- Step 4: Each $P_{i,j}$ executes:
  if $p_j = t_{i+j}$ then c=0 else c=1
- Step 5:[Count the number of 1's on each row] It is proved in [12] that, counting the number of 1's in a 0/1 sequence of length m can be performed in O(1) time on $m \times \log^2 m$ reconfigurable mesh (Lemma 2). Since it is assumed that $(n-m+1) \geq m \geq 16$ and $m \geq \log^2 m$ for $m \geq 16$, processors utilize the third dimension of the RMESH to form (n-m+1) 2D horizontal planes of size $m \times (n - m + 1)$ and each 2D plane counts the number of 1's on its corresponding row in O(1) time (see Figure 5(b)). The results are stored in the first column of the front plane as shown in Figure 5(c).
- Step 6:[Find the minimum count, mc] Form a 2D mesh of size $(n - m + 1) \times (n - m + 1)$ to find the minimum of the (n-m+1) counts each representing the number of mismatches between P and T at different starting points (see Figure 5(c). Call this number mc and store it along with the processor id in $PE_{0,0}$. A minimum-finding-algorithm with O(1) complexity can be used for this purpose (e.g. the algorithm in section III of [13]).
- Step 7: If the minimum count, $mc \leq k$, then $PE_{0,0}$ reports success: "there is a match between P and T with at most k mismatches". Otherwise, report that the search is not successful.

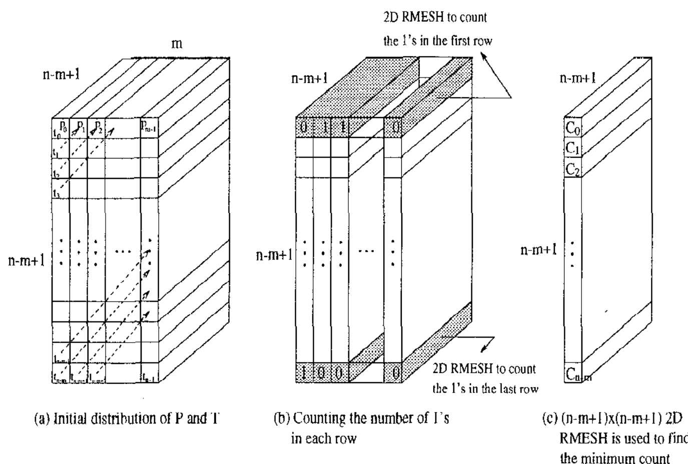


Fig. 5. 3D RMESH used for solving the constant-time approximate string matching algorithm

Steps 1-3 are same as those in Algorithm II.1 and can be executed in $O(1)$ time. Since the counting operation in Step 5 and the minimum operation in Step 6 can both be performed in $O(1)$ time [12], [13], the overall time complexity of this algorithm is $O(1)$. Here, note that, in step 6, if we were to use an O(1) sorting algorithm instead of a minimum-finding-algorithm, we would have found all the occurrences of P in T where the number of mismatches between the two strings is at most k ($k \leq m$). There are a number of papers which present algorithms to sort N numbers in O(1) time using an N^2 RMESH [14], [15], [16].

III. Conclusions

In recent years, the researchers have shown interest in designing efficient algorithms for RMESH architectures because they offer the needed efficiency and flexibility in interprocessor communications by allowing the network topology to change dynamically as required by the algorithm. String matching has been a common problem in many applications, such as, searching, DNA sequencing, spell checking, and file comparison. This paper presents three time-efficient algorithms for string matching on an RMESH. The first algorithm finds the exact matching between a text T of length n and a pattern P of length m in $O(1)$ time. The second algorithm finds the approximate matching between T and P in $O(k)$ time, where k is the maximum edit distance between T and P. Both algorithms require a 2-D RMESH. The third algorithm finds an approximate match between T and P with an edit distance of at most k where only the replacement operation is considered in the computation of the edit distance. This algorithm runs in O(1) time on a 3-D RMESH. As a final note, we state that the RMESH model described in section I-A is directional [11] and provides more power than needed. A simpler model would be sufficient to run the proposed algorithms without increasing the reported time complexities.

References

[1] D. E. Knuth, J. H. J. Morris, and V. R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, vol. 6(2), pp. 323–350, June 1977.

[2] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20(10), pp. 762–772, October 1977.

[3] S. Baase, *Computer Algorithms; Introduction to Design and Analysis*. Addison-Wesley, 2nd ed., 1988.

[4] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequences of two proteins," *Journal of Mol. Bio.*, vol. 48, pp. 444–453, 1970.

[5] D. Sankoff and J. B. K. (Eds.), *Time Warp, String Edits, and Macro molecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.

[6] E. Ukkonen, "On approximate string matching," in *Proceedings Int. Conf. Found. Comput. Theory, Lecture Notes in Computer Science*, vol. 158, pp. 487–495, Springer-Verlag, Berlin/New York, 1983.

[7] G. M. Landau and U. Vishkin, "Fast parallel and serial approximate string matching," *Journal of Algorithms*, vol. 10, pp. 157–169, 1989.

[8] Z. Galil and K. Park, "An improved algorithm for approximate string matching," *SIAM J. Comput.*, vol. 19(6), pp. 989–999, 1990.

[9] A. Apostolico, C. Iliopoulos, G. M. Landau, B. Schieber, and U. Vishkin, "Parallel construction of a suffix tree with applications," *Algorithmica*, vol. 3, pp. 347–365, 1988.

[10] Y. Jiang and A. H. Wright, "O(k) parallel algorithms for approximate string matching," *Neural, Parallel and Scientific Computations*, vol. 1, pp. 443–452, 1993.

[11] F. Ercal and H. C. Lee, "Constant-time reconfigurable mesh algorithms for maze routing," *Technical Report CSC 96-05, University of Missouri-Rolla (also submitted to JPDC)*, 1996.

[12] H. Park, H. J. Kim, and V. K. Prasanna, "An O(1) time optimal algorithm for multiplying matrices on reconfigurable mesh," *Information Processing Letters*, vol. 47, pp. 109–113, 1993.

[13] R. Miller, V. K. Prasanna-Kumar, D. I. Reisis, and Q. F. Stout, "Parallel computations on reconfigurable meshes," *IEEE Trans. Computers*, vol. 42(6), pp. 678–692, June 1993.

[14] M. Nigam and S. Sahni, "Sorting n numbers on $n \times n$ reconfigurable meshes with buses," in *Technical Report 92-5*, University of Florida, 1992.

[15] J. W. Jang and V. K. Prasanna, "An optimal sorting algorithm on reconfigurable mesh," in *Proceedings. Sixth International Parallel Processing Symposium*, pp. 130–7, 1992.

[16] S. Olariu, J. L. Schwing, and J. Zhang, "Integer sorting in o(1) time on an $n \times n$ reconfigurable mesh," in *Eleventh Annual International Phoenix Conference on Computers and Communications*, pp. 480–4, 1992.