INTERNATIONAL JOURNAL OF
INFORMATION AND SYSTEMS SCIENCES
Volume 1, Number 1, Pages 1–18

EFFICIENT COARSE GRAINED DATA DISTRIBUTIONS AND STRING PATTERN MATCHING

LAURENCE BOXER AND RUSS MILLER

(Communicated by)

Abstract. Distributing a packet of data from a source processor to all processors in a parallel computer is a fundamental operation. The efficiency of this operation depends on the configuration of the processors. We show that the sets of architectures A allowing efficient distribution for coarse grained multicomputers include those satisfying the hypothesis that there exists a constant c_A such that every member of A has a spanning tree in which all vertices have degree less than c_A. We give a simple example of a set A that does not satisfy this hypothesis and whose members cannot distribute a packet of data efficiently. We discuss applications of this result to string pattern recognition problems.

Key Words. scalable parallel algorithm, coarse grained parallel computer, string matching, spanning tree

1. Introduction

An important problem in coarse grained parallel computation is that of efficiently copying a small- to moderate-sized packet of data from one processor to all others. In this paper, we obtain a solution to this problem that is efficient on many architectures, and we apply this solution to problems in string pattern matching.

Coarse grained models have been studied in many recent papers. The Coarse Grained Multicomputer (CGM) model was introduced in [7]. Computational geometry algorithms for the CGM have been presented in [1, 3, 4, 7, 10, 8]; also see [6]. Image processing and string pattern matching algorithms for the CGM were presented in [2]. Other computational models for coarse grained parallel computers include [17, 5, 13].

As is the case for fine-grained parallel computing, it is also true for coarse-grained parallel computing that architecture often makes an asymptotic difference in the running time of an algorithm. This is illustrated in [2], in which the analysis of some of the algorithms presented depends on the architecture of the CGM on which they are implemented; and it is true of the problems we study in this paper.

When we design a parallel algorithm for a problem with an efficient sequential solution, we try to achieve

$$T_{par}(n) \;=\; \Theta(T_{seq}(n)\,/\,q) \tag{1}$$

where n is the size of the problem (typically, the amount of data processed), q is the number of processors used by the parallel computer, $T_{seq}(n)$ is the running time of the best sequential algorithm to solve the problem we're studying, and $T_{par}(n)$ is the running time of our parallel algorithm (we note this goal is not always achievable).

2000 *Mathematics Subject Classification.* 68Q05, 68Q25, 68W10, 68W32.

The problem of distributing an array of data D from a source processor to all other processors arises in a variety of problems (see [2]). Suppose, for example, a solution of problem X on a $CGM(n,q)$ requires such a distribution operation, and X has a sequential solution with running time of $T_{seq}(n) = \Theta(n)$. To achieve the goal of equation (1) for problem X, it is therefore necessary that our problem of distributing D to all processors have a solution that runs in $O(n/q)$ time.

The remainder of the paper is organized as follows.

- In Section 2, we give background material. Section 2.1 discusses our model of computation, the coarse grained multicomputer. In Section 2.2, we state the problem that this paper is concerned with. In Section 2.3, we give an example that shows that our goal of efficient distribution of a packet of data (as defined in Section 2.2) is not achievable on all parallel architectures.
- In Section 3, we state a simple algorithm for the distribution of a packet of data on a coarse grained multicomputer, and show that for many architectures, this algorithm can be implemented efficiently. Indeed, we have a simple condition, Hypothesis 3.1, satisfied by many architectures for which efficient implementation is possible.
- In Section 4, we give applications of our algorithm for distributing a packet of data among processors to several problems in string pattern matching. We show that for architectures that satisfy Hypothesis 3.1, we can implement efficient parallel versions of several classical sequential algorithms in exact and approximate string pattern matching.
- In Section 5, we give some concluding remarks.

2. Preliminaries

2.1. Model of Computation. The *coarse grained multicomputer*, or $CGM(n,q)$, is described in [7]. A $CGM(n,q)$ has a set of q processors that operate on $\Theta(n)$ data items. Every processor has $\Omega(n/q)$ local memory cells, each of $\Theta(\log n)$ bits. "Coarse grained" means the size $\Omega(n/q)$ of each local memory is "considerably larger" than $\Theta(1)$; customarily, we interpret "considerably larger" to mean that $n/q \geq q$ (equivalently, $n \geq q^2$), as we do in the current paper. The processors may share memory or may be arranged in some interconnection network.

We assume that processors may communicate data without sorting. Any directly connected pair of processors may exchange a unit of data in $\Theta(1)$ time. Similarly, in a shared memory system, any pair of processors may exchange a unit of data in $\Theta(1)$ time.

We regard a $CGM(n,q)$ as a connected graph G in which the vertex set is the set of processors and the edge set is the set of communications links that join pairs of processors (if the $CGM(n,q)$ is a shared memory machine, we regard it as a complete graph, *i.e.*, every pair of processors is regarded as joined by an edge).

We design the algorithms of this paper to be *scalable, i.e.*, they are described so they may be implemented over the full range of processors, $1 \leq q \leq n^{1/2}$, in terms of which the coarse grained models are described.

2.2. The problem. The problem we study is the following. Let D be a packet of data (we may think of D as an array, $D = [d_1, \ldots, d_m]$) with $|D| = m = O(n/q)$, such that D is initially stored in one processor, P_0, of a $CGM(n,q)$. Devise an efficient algorithm to distribute a copy of D to each processor. It was shown in [2] that this is a useful operation for CGM algorithms for string pattern matching.

If we assume P_0 must sequentially communicate each member of D to at least one other processor, and all other processors must sequentially receive the members

of D, perhaps after some delay, then an algorithm to solve this problem will require $\Omega(m+\delta)$, where δ represents the maximum delay for a processor awaiting data, and is at least proportional to the communication diameter of the $CGM(n,q)$. The worst case architecture with respect to δ is a linear array, for which $\delta = \Theta(q)$.

Thus, our goal is to find an algorithm that can solve this problem in $O(m+q)$ time. Since $m = O(n/q)$ and $q \leq n/q$, the time required would then be $O(n/q)$.

2.3. Counterexample. It was incorrectly claimed in [2] that the goal stated in Section 2.2 can be met for any CGM architecture. A counterexample to this claim is found in the *star* architecture, in which one processor, say, P_0, is directly connected to each of the other processors $P_1, \ldots, P_{q-1}$; but if $i, j \in \{k\}_{k=1}^{q-1}$, $i \neq j$, then processors P_i and P_j are not directly connected. If the array D is initially in P_0, then P_0 must sequentially send copies of D to all of its $q-1$ neighboring processors in $\Theta(mq)$ time.

If $m = n/q$, the star architecture gives a running time of $\Theta(n)$. This is undesirable, as it implies that no problem requiring $\Theta(n/q)$ data to be distributed from one processor to all other processors can be solved in sublinear time on a star $CGM(n,q)$.

3. Solutions for various architectures

The problem statement in Section 2.2 is a generalization of the problem of broadcasting a unit of data from one processor to all other processors. It follows that for many parallel architectures, this problem may be solved by having data flow along the lines of a standard broadcast algorithm.

3.1. CR PRAM. For a parallel random access machine (PRAM) with the concurrent read (CR) property, we achieve our goal by using the following algorithm.

- The source processor writes D into the shared memory. This takes $\Theta(m)$ time.
- In parallel, every other processor reads D from shared memory. This takes $\Theta(m)$ time.

The algorithm takes $\Theta(m) = O(n/q)$ time.

The case of the PRAM with the exclusive read (ER) property is covered by the discussion in Section 3.2.

3.2. Other models. We will show that the following hypothesis is satisfied by many sets of architectures A whose members can distribute a packet of data efficiently (by *architecture* we mean a connected graph in which vertices represent processors and edges represent communications links).

The *degree* of a vertex v in a graph G is the number of edges of G incident on v.

Hypothesis 3.1. *There is a positive constant c_A such that for every $G \in A$ there is a known spanning tree T of G such that every vertex of T has degree less than c_A.* ■

Notice the set of inefficient star architectures discussed in the example of Section 2.3 fails to satisfy Hypothesis 3.1.

Actually, we can replace, in the algorithms of Section 4, the hypothesis of the spanning tree being known by the weaker hypothesis that such a tree can be determined in $O(n/q)$ time.

Theorem 3.2. *Let A be a set of parallel computers used as CGMs. Suppose a processor of a member of A can send a unit of data to only one neighboring processor at a time (*i.e.*, there is no bus and no CR shared memory). If A satisfies Hypothesis 3.1, then a member of A can distribute a copy of an array D such that $|D| = m = O(n/q)$ from any source processor to all other processors in $O(n/q)$ time.*

Proof: The following algorithm can be executed on any parallel architecture. We are particularly interested in its implementations on sets of architectures A that satisfy Hypothesis 3.1.

In parallel, all processors act as described below.
- The source processor P_k does the following.
  For $i = 1$ to m
    For each neighbor P_j to which P_k would send data in a broadcast operation
      send d_i to P_j
    End For each
  End For i
- All other processors P_a do the following.
  For $i = 1$ to m
    Receive d_i
    For each neighbor P_j to which P_a would send data in a broadcast operation
      send d_i to P_j
    End For each
  End For i

End Parallel

The running time of this algorithm is analyzed, assuming Hypothesis 3.1, as follows.

- Excluding waiting time, the actions of each processor P_a take time jointly proportional to m and to the number of neighboring processors with which P_a communicates in a broadcast operation.
- Processors other than the source processor must wait for the first data value d_1 to arrive. If no processor sends data to more than c_A neighboring processors, the waiting time in the worst case is proportional to the length of a longest path in the spanning tree, described in Hypothesis 3.1, between the source processor and other processors, and is therefore $O(q)$.

Thus, if Hypothesis 3.1 is satisfied, this algorithm requires $O(m + q)$ time. Since $m = O(n/q)$ and $q \leq n/q$, the running time is $O(n/q)$. ■

Thus, for the following architectures, each of which satisfies Hypothesis 3.1, we achieve the goal of an algorithm that distributes $m = O(n/p)$ data from one processor to all other processors of a $CGM(n, q)$ configured in the given architecture, in $O(n/q)$ time, as each of these has a broadcast algorithm that is implemented in a spanning tree in which each processor-vertex has a bounded number of neighbors (see [16] for broadcast algorithms for these architectures).

- ER PRAM. Note that an ER PRAM can execute the algorithm as it would be executed on a binary tree architecture.
- Linear array, mesh, binary tree, pyramid, mesh-of-trees.
- Hypercube. Although the "standard" broadcast algorithm for a hypercube (see [16]) has the source processor sending its data to each of its $\log_2 q$ neighboring processors, we can use an alternate broadcast algorithm for the

hypercube that makes use of a binary tree, rooted at the source processor, as a spanning tree.

4. Applications to string pattern matching

Above, we studied the problem of distributing a packet of data, of small to moderate size, from one processor to all other processors of a $CGM(n,q)$. We showed that

this operation can be performed efficiently on many, but not all, models of parallel computing. In particular, it can be performed efficiently on sets of architectures satisfying Hypothesis 3.1.

Theorem 3.2 corrects Theorem 4.2 of [2]. Therefore, the assertions of dependent claims, the Theorems and Corollary numbered 6.1 through 6.4 of [2], must be restricted to those architectures for which a $CGM(n,q)$ can distribute a packet D of data, such that

$|D| = m = O(n/q)$, from a source processor to all other processors, within the running times claimed for the respective algorithms. The arguments given for these assertions in [2] include steps for construction of a spanning tree; however, we can omit this step in the presence of Hypothesis 3.1 (this is useful, for example, in the case of hypercubes, as discussed above, since the spanning tree constructed in [2] need not satisfy Hypothesis 3.1). Correct versions, including proofs, of these assertions follow.

We say a *gathering* of a set S of data items, initially distributed among the processors of a parallel computer, to the processor PE_i, is an operation that places a copy of each member of S in PE_i. We have the following.

Theorem 4.1. [2] *Let S be a set of m data items such that $m = \Omega(q)$ and $m = O(n/q)$, and suppose S is distributed among the processors of a $CGM(n,q)$, G. Then a gathering of S to any processor of G can be performed in $\Theta(m)$ time, which is optimal in the worst case.* ■

We use the following.

Proposition 4.2. [2] *Let T be a text of n characters evenly distributed among the processors of a $CGM(n,q)$ G so that the portion of T in each processor is a substring of T. Let m be a positive integer such that $m = O(n/q)$. Then, in parallel, substrings of m characters apiece of T stored in PE_{i+1} can be sent from PE_{i+1} to PE_i, $i \in \{1,\ldots,q-1\}$, in the following time.*

- *In $\Theta(m)$ time, if for all i we have PE_i and PE_{i+1} adjacent.*
- *In $\Theta(T_{sort}(mq,q))$ time, in general.* ■

String pattern matching problems are at the core of searches performed by word processors and Web search engines, as well as the searches of molecular biologists for strands of DNA in a genome or for protein building blocks in complex proteins. In the following sections, we obtain efficient coarse grained parallel algorithms for several string pattern matching problems, using Theorem 3.2.

For simplicity, we will assume all string pattern matching problems discussed use an alphabet of fixed size. To simplify the exposition, we assume processor PE_i initially has the substring

$$Q_i = T[\frac{(i-1)n}{q}+1,\ldots,\frac{in}{q}]$$

of $T = T[1, \ldots, n]$, $i \in \{1, \ldots, q\}$.

The algorithm of the following Lemma is an efficient implementation of data movement operations we will use in several string pattern matching algorithms.

Lemma 4.3. *Let A be a set of connected graphs satisfying Hypothesis 3.1. Let T be a text of n characters evenly distributed among the processors of a $CGM(n, q)$ $G \in A$ so that the portion of T in each processor is a substring of T. Let P be a pattern of m characters, where $m = O(n/q)$. Then every processor can have a copy of P, and $i < q$ implies PE_i can receive from PE_{i+1} a substring of T that has $O(m)$ characters, in time satisfying the following.*

- *In $O(n/q)$ time, if for all i we have PE_i and PE_{i+1} adjacent.*
- *In general, in $O(n/q + T_{sort}(mq, q))$ time.*

Proof: We give the following algorithm.

(1) Gather P to one processor, say, PE_1. By Theorem 4.1, this takes $\Theta(m) = O(n/q)$ time.
(2) Broadcast the pattern P from PE_1 so that every processor has a copy of P. By Theorem 3.2, this takes $O(n/q)$ time.
(3) Use the algorithm of Proposition 4.2 to send the desired substring of $O(m)$ characters of T from PE_{i+1} to PE_i, $i \in \{1, \ldots, q-1\}$. (We observe that this might be motivated by the possibility of finding a match for P in a substring of T that is not entirely contained in one processor.)
    - If for all i we have PE_i and PE_{i+1} adjacent, this takes $\Theta(m) = O(n/q)$ time.
    - More generally, this takes $\Theta(T_{sort}(mq, q))$ time.

Clearly, the running time is as claimed. ■

Note the running time of this algorithm is $O(n/q)$ if for all i we have PE_i and PE_{i+1} adjacent, or if $T_{sort}(mq, q) = O(n/q)$.

4.1. Exact string pattern matching. The exact string pattern matching problem is as follows. Given a text string T and a pattern string P, find all substrings P' of T such that P' is an exact copy of P. There are linear-time sequential solutions to this problem [12].

The following corrects Theorem 6.1 of [2].

Theorem 4.4. *Let A be a set of connected graphs satisfying*

Hypothesis 3.1. Let T be a text of n characters evenly distributed among the processors of a $CGM(n, q)$ $G \in A$ so that the portion of T in each processor is a substring of T. Let P be a pattern of m characters, where $m = O(n/q)$.

Then every exact copy of P in T can be identified in a running time satisfying the following.

- *In optimal $O(n/q)$ time, if for all i we have PE_i and PE_{i+1} adjacent.*
- *In general, in $O(n/q + T_{sort}(mq, q))$ time.*

Proof: We give the following algorithm.

(1) Use the algorithm of Lemma 4.3 so that every processor has a copy of P and processor PE_i, $i \in \{1, \ldots, q-1\}$, gets the substring $R_i = T[\frac{in}{q} + 1, \ldots, \frac{in}{q} + m - 1]$ of T from PE_{i+1}.
    - If for all i we have PE_i and PE_{i+1} adjacent, this takes $O(n/q)$ time.
    - More generally, this takes $O(n/q + T_{sort}(mq, q))$ time.

At the end of this step, processor PE_i has the substring

$$S_i \;=\; Q_i \cup R_i \;=\; T[\frac{(i-1)n}{q}+1,\ldots,\frac{in}{q}+m-1],\ i\in\{1,\ldots,q-1\};$$

PE_q has $S_q \;=\; Q_q$. Note S_i is the smallest substring of T that must be examined to find all matches to P in T starting in Q_i.

(2) In parallel, each processor PE_i applies a linear-time sequential algorithm for exact string pattern matching to the text S_i and the pattern P. This takes $O(n/q)$ time.

We summarize the running time of our algorithm as follows.

- If for all i we have PE_i and PE_{i+1} adjacent, the algorithm runs in $O(n/q)$ time. This is optimal in the worst case, since the sequential version of the problem has a worst case optimal running time of $\Theta(n)$ [12].
- In general, the algorithm uses $O(n/q \;+\; T_{sort}(mq,q))$ time. ■

Thus, our algorithm for the exact string matching problem achieves the goal of equation (1) on sets of graphs that satisfy Hypothesis 3.1 if either we have PE_i and PE_{i+1} adjacent for all i, or if $T_{sort}(mq,q) \;=\; O(n/q)$.

4.2. Approximate matching with mismatches. We quote from [2]:

> Given integers k, m, n such that $0 \le k \le m \le n$, a pattern P of size m, and a text T of size n, it is often useful to allow k mismatches in the string matching problem. That is, we seek all substrings P' of T such that $|P'| = m$ and P' is a copy of P except for at most k mismatched characters. This is the *k-approximate matching problem*....
>
> The problem of finding approximate matches is closely related to the *match-count problem*, in which, for every substring T' of T such that $|T'| = m$, we find the number of characters of T' that match the corresponding character of P.

In this section, we obtain two efficient coarse grained parallel solutions to the k-approximate matching problem, based on two different sequential solutions.

A sequential algorithm for the match-count problem based on the Fast Fourier Transform (FFT) [9, 11, 12] runs in $T_{seq}(k,m,n) \;=\; O(n\log n)$ time. The following corrects Theorem 6.2 of [2].

Theorem 4.5. *Let A be a set of connected graphs satisfying Hypothesis 3.1. Suppose T is a text of n characters distributed evenly in a $CGM(n,q)$ $G \in A$, and P is a pattern of size $m = O(n/q)$ stored in G. The match-count problem can be solved in G as follows.*

- *In $O(\frac{n\log n}{q})$ time, if for all i we have PE_i and PE_{i+1} adjacent.*
- *In $O(\frac{n\log n}{q} \;+\; T_{sort}(mq,q))$ time, in general.*

Proof: We give the following algorithm.

(1) Use the algorithm of Lemma 4.3 so that every processor has a copy of P and processor PE_i, $i \in \{1,\ldots,q-1\}$, gets the substring $R_i = T[\frac{in}{q}+1,\ldots,\frac{in}{q}+m-1]$ of T from PE_{i+1}. This takes $O(n/q)$ time, if for all i we have PE_i and PE_{i+1} adjacent. In the general case, the time required is $\Theta(T_{sort}(mq,q))$.

(2) In parallel, each PE_i solves the match-count problem for P and the text

$$S_i \;=\; Q_i \cup R_i \;=\; T[\frac{(i-1)n}{q} + 1 \ldots \frac{in}{q} + m - 1]$$

(except for PE_q, which solves the match-count problem for P and the text Q_q), using the sequential algorithm discussed above. The time for this step is $O(\frac{n \log(n/q)}{q})$, which is $O(\frac{n \log n}{q})$, since $q = O(n^{1/2})$.

It follows that the running time of our algorithm is as claimed above. ■

Thus, our algorithm for the match-count problem achieves the goal of equation (1), relative to the sequential algorithm discussed above, on sets of graphs that satisfy Hypothesis 3.1 if either we have PE_i and PE_{i+1} adjacent for all i, or if $T_{sort}(mq, q) \;=\; O(\frac{n \log n}{q})$.

The following corrects Corollary 6.3 of [2].

Corollary 4.6. *Let A be a set of connected graphs satisfying Hypothesis 3.1.*

For integers k, m, n, pattern P, and text T as described above,

a $CGM(n, q)$ $G \in A$ can solve the k-approximate matching problem for any k satisfying $0 \leq k \leq m$ *as follows.*

- *In $O(\frac{n \log n}{q})$ time, if for all i we have PE_i and PE_{i+1} adjacent.*
- *In $O(\frac{n \log n}{q} \;+\; T_{sort}(mq, q))$ time, in general.*

Proof: We give the following algorithm.

(1) Solve the match-count problem, using the algorithm of Theorem 4.5.

(2) For each $matches[i]$, compute

$$mismatches[i] \;=\; m \;-\; matches[i].$$

This takes $\Theta(n/q)$ time. Note for each index i, the substring of T of length m starting at $T[i]$ is a k-approximate match for P if and only if $mismatches[i] \leq k$.

By Theorem 4.5, the running time of this algorithm is dominated by the first step, and the assertion follows. ■

Thus, our algorithm for the k-mismatches problem achieves the goal of equation (1), relative to the analogous sequential algorithm, on sets of graphs that satisfy Hypothesis 3.1 if either we have PE_i and PE_{i+1} adjacent for all i, or if $T_{sort}(mq, q) \;=\; O(\frac{n \log n}{q})$.

A different sequential algorithm for the k-approximate matching problem has running time $T_{seq}(k, m, n) \;=\; O(k(m \log m \;+\; n))$ [14]. We use this algorithm as the basis of a coarse grained solution, as follows.

Theorem 4.7. *Let A be a set of connected graphs satisfying Hypothesis 3.1. Suppose T is a text of n characters distributed evenly in a $CGM(n, q)$ $G \in A$, P is a pattern of size $m = O(n/q)$ stored in G, and $0 \leq k \leq m$. The k-approximate matching problem can be solved in G as follows.*

- *In $O(k(m \log m + n/q))$ time, if for all i we have PE_i and PE_{i+1} adjacent.*
- *In $O(k(m \log m \;+\; n/q) \;+\; T_{sort}(mq, q))$ time, in general.*

Proof: We give the following algorithm.

(1) Use the algorithm of Lemma 4.3 so that every processor has a copy of P and processor PE_i, $i \in \{1, \ldots, q-1\}$, gets the substring $R_i = T[\frac{in}{q} + 1, \ldots, \frac{in}{q} + m - 1]$ of T from PE_{i+1}. This takes $O(n/q)$ time, if for all i we have PE_i and PE_{i+1} adjacent. In the general case, the time required is $O(n/q + T_{sort}(mq, q))$.

(2) In parallel, each PE_i uses the algorithm of [14] to solve the k-approximate matching problem for P and the text

$$S_i \;=\; Q_i \cup R_i \;=\; T[\frac{(i-1)n}{q} + 1, \ldots, \frac{in}{q} + m - 1]$$

(except for PE_q, which solves the k-approximate matching problem for P and the text Q_q). The time for this step is $O(k(m \log m \,+\, n/q))$.

It follows that the running time of our algorithm is as claimed above. ■

Thus, our algorithm for the k-approximate matching problem achieves the goal of equation (1), relative to the algorithm of [14], on sets of graphs that satisfy Hypothesis 3.1 if $m \log m = O(n/q)$ and either we have PE_i and PE_{i+1} adjacent for all i, or if $T_{sort}(mq, q) \;=\; O(kn/q)$.

4.3. String matching with differences. We quote from [2]:

> More general that the approximate string matching problem is the problem
>
> of *string matching with k differences....* In this problem, a pattern P of length m, a text T of length n, and an integer $k \geq 0$ are input. Output
>
> consists of identification of all substrings P' of T such that P' matches P with at most k differences of the following kinds:
>
> - A character of P corresponds to a different character of $P' \subset T$. A difference of this kind is a *mismatch* between the corresponding characters.
> - A character of P corresponds to no character of P'. A difference of this kind is an *insertion.*
> - A character of P' corresponds to no character of P. A difference of this kind is a *deletion.*
>
> Since the *edit distance* between strings P and P' is the minimum number of character substitutions, insertions, or deletions
>
> necessary to transform P to P' [12], our problem is to find all substrings P' of T such that the edit distance between P and P' is at most k.

In the following, we use the sequential algorithm of [15] for string matching with k differences. The running time of this algorithm is $O(kn)$. The algorithm uses $\Theta(kn)$ memory for a certain matrix. Correspondingly, we use a $CGM(kn, q)$, rather than our usual $CGM(n, q)$, but we continue to use the restriction $1 \leq q \leq n/q$. The following both corrects and improves on the running time asserted in Theorem 6.4 of [2].

Theorem 4.8. *Let A be a set of connected graphs satisfying Hypothesis 3.1.*
Let T be a text of n characters on an alphabet of fixed size,

distributed evenly among the processors of a $CGM(kn, q)$ $G \in A$, $1 \leq q \leq n/q$. Let P be a pattern of m characters stored in G, where $m = O(n/q)$. Let k be an integer such that $0 \leq k \leq m$. Then the string matching with k differences problem can be solved in the following time.

- *$O(kn/q)$, if for all i we have PE_i and PE_{i+1} adjacent.*
- *$O(kn/q + T_{sort}(mq, q))$, generally.* ■

Proof: We give the following algorithm.

(1) Since $k \leq m$, we can use the algorithm of Lemma 4.3 so that every processor has a copy of P and processor PE_i, $i \in \{1, \ldots, q-1\}$, gets the substring $R'_i = T[\frac{in}{q} + 1, \ldots, \frac{in}{q} + m + k - 1]$ of T from PE_{i+1}. PE_i now has the smallest substring of T, namely

$$S_i = Q_i \cup R'_i = T[\frac{(i-1)n}{q} + 1, \ldots, \frac{in}{q} + m + k - 1] \text{ for } i < q,$$

$S_q = Q_q$, that must be examined to find all substrings of T starting in Q_i that can match P with at most k differences. By Proposition 4.2, this takes $O(n/q)$ time, if for all i we have PE_i and PE_{i+1} adjacent. In the general case, the time required is $O(n/q + T_{sort}(mq, q))$. Note S_i has $\Theta(n/q)$ characters, for all i.

(2) In parallel, each processor PE_i uses the algorithm of [15] to solve the problem for P and the text S_i. This takes $O(kn/q)$ time.

Clearly, the running time of the algorithm is as claimed above. ■

Thus, our algorithm for the k-differences problem achieves the goal of equation (1), relative to the algorithm of [15], on sets of graphs that satisfy Hypothesis 3.1 if either we have PE_i and PE_{i+1} adjacent for all i, or if $T_{sort}(mq, q) = O(kn/q)$.

5. Further remarks

We have studied the problem of efficient distribution of an array, initially stored in one processor of a coarse grained multicomputer, to all other processors. We have shown that the efficiency of any solution to this problem depends on the architecture of the computer. In particular, we showed that if A is a set of connected graphs and the array D satisfies $|D| = m = O(n/q)$, then this distribution can be done in $O(n/q)$ time for all $G \in A$ (with G regarded as a $CGM(n, q)$) if A satisfies Hypothesis 3.1, *i.e.*, there is a constant c_A such that each $G \in A$ has a spanning tree T_G such that every vertex (processor) v of T_G has degree less than c_A. We use this result to obtain efficient coarse grained solutions to several string pattern matching problems for sets of CGM's satisfying Hypothesis 3.1.

Acknowledgments

The suggestions of the anonymous referees are gratefully acknowledged.

References

[1] L. Boxer and R. Haralick, Even faster point set pattern matching in 3-d, SUNY at Buffalo Department of Computer Science and Engineering Technical Report 2001-01.

[2] L. Boxer and R. Miller, Coarse grained gather and scatter operations with applications, *J. Parallel and Distributed Computing* 64 (2004), 1297-1310.

[3] L. Boxer, R. Miller, and A. Rau-Chaplin, Scaleable parallel algorithms for lower envelopes with applications, *J. Parallel and Distributed Computing* 53 (1998), 91-118.

[4] L. Boxer, R. Miller, and A. Rau-Chaplin, Scalable parallel algorithms for geometric pattern recognition, *J. Parallel and Distributed Computing* 58 (1999), 466-486.
[5] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, LogP: Towards a realistic model of parallel computation, *Proc. 4th ACM SIGPLAN Sym. on Principles of Parallel Programming*, 1993.
[6] F. Dehne, *ed.*, Coarse grained parallel algorithms,
special edition of *Algorithmica*, vol. 24, no. 3/4, July/August, 1999.
[7] F. Dehne, A. Fabri, and A. Rau-Chaplin, Scalable parallel geometric algorithms for multicomputers, *Proc. 9th ACM Symp. on Computational Geometry*, (1993), 298-307.
[8] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. Khokhar, A randomized parallel 3D convex hull algorithm for coarse grained multicomputers, *Proc. 7th ACM Symp. on Parallel Algorithms and Architectures*, (1995), 27-33.
[9] J. Felsenstein, S. Sawyer, and R. Kochin, An efficient method for matching nucleic acid sequences, *Nucleic Acids Research* 10 (1982), 133-139.
[10] A. Ferreira, A. Rau-Chaplin, and S. Ubéda, Scalable 2d convex hull and triangulation algorithms for coarse grained multicomputers, *Proc. 7th IEEE Symp. on Parallel and Distributed Processing* (1995), 561-569.
[11] M. Fischer and M. Paterson, String-matching and other products, in *Complexity of Computation*, R.M. Karp, ed., *SIAM-AMS Proceedings* (1974), 113-125.
[12] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, Cambridge, 1997.
[13] S. Hambrusch and A. Khokhar, C3: an architecture-independent model for coarse-grained parallel machines, Purdue University Computer Sciences Technical Report CSD-TR-93-080 (1993).
[14] G.M. Landau and U. Vishkin, Efficient string matching with k mismatches, *Theoretical Computer Science* 43 (1986), 239-249.
[15] G.M. Landau and U. Vishkin, Fast parallel and serial approximate string matching, *Journal of Algorithms* 10 (1989), 157-169.
[16] R. Miller and L. Boxer, *Algorithms Sequential & Parallel: a Unified Approach*, Charles River Media, Hingham, MA, 2005.
[17] L.G. Valiant, A bridging model for parallel computation, *Communications of the ACM* 33 (1990), 103-111.

Department of Computer and Information Sciences, Niagara University, NY 14109, USA; and Department of Computer Science and Engineering, State University of New York at Buffalo.
E-mail: `boxer@niagara.edu`
URL: `http://purple.niagara.edu/boxer`

Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, New York 14260, USA
E-mail: `miller@buffalo.edu`
URL: `http://www.cse.buffalo.edu/faculty/miller/`