Optimal Parallel Pattern Matching in Strings

Uzi Vishkin*

Courant Institute, New York University, and
Department of Computer Science, Tel Aviv University,
Tel Aviv 69978, Israel

Given a text of length $n$ and a pattern of length $m$, we present a parallel linear
algorithm for finding all occurrences of the pattern in the text. The algorithm runs
in $O(n/p)$ time using any number of $p \leq n / \log m$ processors on a concurrent-read

I. INTRODUCTION

The family of models of computation used in this paper is the parallel
random-access-machines (PRAMs). All members of this family employ $p$
synchronous processors all having access to a common memory. The
present papers refers to two member of the PRAM family. Our presentation
focuses on the concurrent-read concurrent-write (CRCW) PRAM.
This model allows simultaneous reading from the same memory location as
well as simultaneous writing. In the latter case, the smallest serial numbered
among the processors that attempt to write succeeds. At the end of
the paper we show that a weaker concurrent-read concurrent-write PRAM
model, where several processors may attempt to write at the same memory
location only if they seek to write the same thing, actually suffices for the
strongest results in this paper. There, we also show how to implement some
of the results on a concurrent-read exclusive-write (CREW) PRAM, where
simultaneous reading into the same memory location but not simultaneous
writing is allowed. See Vishkin (1983a) for a recent survey of results concern-
ning the PRAM family.

Let Seq($n$) be the fastest known worst-case running time of a sequential
algorithm, where $n$ is the length of the input for the problem being con-
sidered. Obviously, the best upper bound on the parallel time achievable

* This research was supported by DOE Grant DE-AC02-76ER03077, by NSF Grants NSF-
DCR-8413359, NSF-MCS79-21258, and NSF-DCR-8318874, and by ONR Grant N0014-85-
K-0046.

91

0019-9958/85 $3.00$

Copyright © 1985 by Academic Press, Inc.
All rights of reproduction in any form reserved.
Proof: A prefix of \( w \) for some \( s \geq 1 \). Hence, \( w \) is a prefix of \( w^* \).

\( n = \frac{1}{a + d} \quad \text{for some} \quad d \geq 1 \).

We have: \( w = w^* \).

Let \( w \) be a prefix of \( w^* \). Let \( w \) be a prefix of \( w^* \). Let \( w \) be a prefix of \( w^* \).

The following proposition is used in this paper, Let \( w \) be a prefix of \( w^* \).

II. PRELIMINARIES

The text analysis part of the algorithm is described in Section 3 and the

11 a. Certain properties of processors in their roles (using branch cut-off).

12 Certain properties of processors in their roles (using branch cut-off).
PATTERN MATCHING IN STRINGS

III. ANALYSIS OF THE TEXT

UZI VISHNIK

96
The following three properties are satisfied:

\[ \text{Partial Witness} \subseteq \text{Witness} \]

\[ \text{Witness} \subseteq \text{Witness} \]

\[ \text{Witness} \subseteq \text{Witness} \]

1. Step 1—Analysis of the Pattern

We define a new order of steps to follow in solving the pattern

We proceed as follows:

\[ \text{Witness} \subseteq \text{Witness} \]

\[ \text{Witness} \subseteq \text{Witness} \]

\[ \text{Witness} \subseteq \text{Witness} \]

2. Step 2—Inference

We infer the following from the above steps:

\[ \text{Witness} \subseteq \text{Witness} \]

\[ \text{Witness} \subseteq \text{Witness} \]

\[ \text{Witness} \subseteq \text{Witness} \]

3. Step 3—Conclusion

We conclude that the following are true:

\[ \text{Witness} \subseteq \text{Witness} \]

\[ \text{Witness} \subseteq \text{Witness} \]

\[ \text{Witness} \subseteq \text{Witness} \]

The above steps provide a method for identifying patterns in data.
Box 1 needs Box 2 or Box 3.

\[ (1 + d) \text{ Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.

\[ \text{Pattern} \neq 0 \]

For Box 1, Box 2, or Box 3.
should remain and $\langle d \rangle$ not update WITNESS.

Initially, there is a number of $d$-blocks that are not query indices, these blocks are those blocks that have a $\langle d \rangle$ value of $1$. As we process these blocks, we may discover that they are query indices, or that they are not. The set of query indices is denoted by $\{ \text{query indices} \}$.

To update WITNESS, we need to consider all $d$-blocks. For each $d$-block $b$, we check if its $\langle d \rangle$ value is $1$. If so, we add $b$ to the set of query indices.

After processing all $d$-blocks, we update WITNESS as follows:

1. If $\langle d \rangle = 1$, then $b$ is a query index and we add it to WITNESS.
2. If $\langle d \rangle = 0$, then $b$ is not a query index and we do not add it to WITNESS.

The updated WITNESS is then used to determine if there are any query indices that satisfy a certain condition. This condition is typically related to the number of query indices that are present in the database.

For example, if we have a query that requires at least $k$ query indices, we can use WITNESS to efficiently determine if such a query is possible. If WITNESS contains at least $k$ query indices, then the query is possible; otherwise, it is not.

This process can be repeated for different values of $k$, allowing us to efficiently explore the space of possible queries.
ACKNOWLEDGMENT

The authors would like to thank ..., and Programming Languages. They would also like to thank ..., for their valuable contributions to this work.

REFERENCES


CONCLUSION

The theoretical framework presented in the previous sections provides a solid foundation for the development of efficient algorithms for pattern matching in strings. The concept of a partial computation considered in the context of parallel mini-processors facilitates the design of algorithms that can be executed concurrently on a set of processors. The key to achieving high performance lies in the efficient design of these algorithms, ensuring that they can be executed simultaneously on multiple processors without conflict or interference.

We have presented a new heuristic algorithm for the string matching problem, which leads to an efficient partial computation scheme. By leveraging the parallel processing capabilities of mini-processors, we can significantly reduce the time required for string matching operations. The effectiveness of our approach is demonstrated through experimental results that show substantial performance improvements over existing methods.

Furthermore, the proposed algorithm is scalable and adaptable to different computational environments. It can be easily modified to accommodate variations in hardware configurations or to incorporate additional features. Future work will focus on extending the algorithm to handle more complex matching tasks, such as those encountered in natural language processing and bioinformatics.

In conclusion, the use of parallel processors in string matching applications demonstrates the potential for significant performance gains. By exploring the implications of partial computations and leveraging the power of mini-processors, we can further enhance the efficiency and scalability of string matching algorithms, making them more suitable for a wide range of real-world applications.