Parallel algorithm for pattern matching problems under substring consistent equivalence relations

Davaajav Jargalsaikhan
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Diptarama Hendrian
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Ryo Yoshinaka
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Ayumi Shinohara
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Abstract
Given a text and a pattern over an alphabet, the pattern matching problem searches for all occurrences of the pattern in the text. An equivalence relation \( \approx \) is called a substring consistent equivalence relation (SCER), if for two strings \( X \) and \( Y \), \( X \approx Y \) implies \( |X| = |Y| \) and \( X[i:j] \approx Y[i:j] \) for all \( 1 \leq i \leq j \leq |X| \). In this paper, we propose an efficient parallel algorithm for pattern matching under any SCER using the “duel-and-sweep” paradigm. For a pattern of length \( m \) and a text of length \( n \), our algorithm runs in \( O(\xi_t m \log^2 m) \) time and \( O(\xi_w m \cdot n \log^2 m) \) work, with \( O(\tau_t n + \xi_t m \log^2 m) \) time and \( O(\tau_w n + \xi_w m \cdot m \log^2 m) \) work preprocessing on the Priority Concurrent Read Concurrent Write Parallel Random-Access Machines (P-CRCW PRAM), where \( \tau_t \), \( \tau_w \), \( \xi_t \), and \( \xi_w \) are parameters dependent on SCERs, which are often linearly bounded in \( n \) and \( m \), respectively.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases parallel algorithm, substring consistent equivalence relation, pattern matching

Introduction
The string matching problem is fundamental and widely studied in computer science. Given a text and a pattern, the string matching problem searches for all substrings of the text that match the pattern. Many matching functions that are used in different string matching problems, including exact [16], parameterized [4], order-preserving [15, 17] and cartesian-tree [19] matchings, fall under the class of substring consistent equivalence relations (SCERs) [18]. An equivalence relation on strings is an SCER, if two strings \( X \) and \( Y \) match under the equivalence relation, then they have equal length and \( X[i:j] \) matches \( Y[i:j] \), for all \( 1 \leq i \leq j < |X| \). Matsuoka et al. [18] generalized the KMP algorithm [16] for pattern matching problems under SCERs. They also investigated periodicity properties of strings under SCERs. Kikuchi et al. [14] proposed algorithms to compute the shortest and longest cover arrays for a given string under any SCER. Hendrian [9] generalized Aho-Corasick algorithm for the dictionary matching under SCERs.

Vishkin proposed two algorithms for exact pattern matching, pattern matching by dueling [20] and pattern matching by sampling [21]. Both algorithms match the pattern to a substring of the text from some positions which are determined by the property of the
2 Parallel algorithm for pattern matching problems under SCERs

Table 1 Summary of the encoding complexities for some SCER on P-CRCW PRAM.

| SCER Type       | $\tau_n$ | $\tau_m$ | $\xi_n$ | $\xi_m$ |
|-----------------|----------|----------|---------|---------|
| Exact           | $O(1)$   | $O(1)$   | $O(1)$  | $O(1)$  |
| Parameterized   | $O(\log n)$ | $O(n \log n)$ | $O(1)$  | $O(1)$  |
| Cartesian-tree  | $O(\log n)$ | $O(n \log n)$ | $O(1)$  | $O(1)$  |

pattern, instead of its prefix or suffix as in, for instance, the KMP algorithm \[10\]. These algorithms are developed for parallel processing.

The dueling technique by Vishkin \[20\] has been proved to be useful for various kinds of pattern matching. Amir et al. \[2\] proposed a duel-and-sweep algorithm for two-dimensional exact matching, which is named “consistency and verification”. Cole et al. \[8\] extended it to two-dimensional parameterized matching. In addition, Jargalsaikhan et al. \[11\] \[12\] proposed serial and parallel duel-and-sweep algorithms for order-preserving matching.

In this paper, we propose an efficient parallel algorithm based on the dueling technique for the pattern matching problem under SCERs. Our parallel algorithm is the first to solve the problem under an arbitrary SCER in parallel. While Vishkin’s dueling algorithm for exact matching depends on the preferable properties of periods of strings, many of those do not hold with SCERs. Therefore, our algorithm involves new ideas and appears quite different from the original for exact pattern matching. For a pattern of length $m$ and a text of length $n$, our algorithm runs in $O(\xi_m \log^2 m)$ time and $O(\xi_m \cdot n \log^2 m)$ work, with $O(\tau_n + \xi_m \log^2 m)$ time and $O(\tau_m + \xi_m \cdot m \log^2 m)$ work preprocessing on the Priority Concurrent Read Concurrent Write Parallel Random-Access Machines (P-CRCW PRAM) \[10\]. Here, $\tau_n$ and $\tau_m$ are time and work respectively, needed on P-CRCW PRAM to encode in parallel a string $X$ of length $n$ under the SCER in concern. Given the encoding of $X$, $\xi_n$ and $\xi_m$ are time and work respectively to re-encode an element w.r.t. some suffix of $X$ of length $m$. Table 1 shows the encoding time and work complexities for some SCERs.

This manuscript fixes minor errors and improves the algorithm efficiency in \[13\].

2 Preliminaries

We use $\Sigma$ to denote an alphabet of symbols and $\Sigma^*$ denotes the set of strings over the alphabet $\Sigma$. For a string $X \in \Sigma^*$, the length of $X$ is denoted by $|X|$. The empty string, denoted by $\epsilon$, is the string of length 0. For a string $X \in \Sigma^*$ of length $n$, $X[i]$ denotes the $i$-th symbol of $X$. $X[i:j] = X[i]X[i+1] \ldots X[j]$ denotes a substring of $X$ that begins at position $i$ and ends at position $j$ for $1 \leq i \leq j \leq n$. For $i > j$, $X[i:j]$ denotes the empty string.

Definition 1 (Substring consistent equivalence relation (SCER) \[18\]). An equivalence relation $\approx \subseteq \Sigma^* \times \Sigma^*$ is a substring consistent equivalence relation (SCER) if for two strings $X$ and $Y$, $X \approx Y$ implies $|X| = |Y|$ and $X[i:j] \approx Y[i:j]$ for all $1 \leq i \leq j \leq |X|$.

For instance, while the parameterized matching \[4\] and order-preserving matching \[17\] \[15\] are SCERs, the permutation matching \[6\] \[7\] and function matching \[11\] are not.

Hereafter we fix an arbitrary SCER $\approx$. We say that a position $i$ is the tight mismatch position if $X[1:i-1] \approx Y[1:i-1]$ and $X[1:i] \not\approx Y[1:i]$. For two strings $X$ and $Y$, let $LCP(X,Y)$ be the length $l$ of the longest prefixes of $X$ and $Y$ match. That is, $l$ is the greatest integer such that $X[1:l] \approx Y[1:l]$. Obviously, if $i$ is the tight mismatch position for $X \not\approx Y$, then $LCP(X,Y) = i - 1$. The converse holds if $i \leq \min\{|X|,|Y|\}$. Similarly, for
a string $X$ and an integer $0 \leq a < |X|$, we define $LCP_X(a) = LCP(X, X[a + 1 : |X|])$. In other words, $LCP_X(a)$ is the length of the longest common prefix, when $X$ is superimposed on itself with offset $a$. We say $X \approx$-matches $Y$ iff $X \approx Y$. Given a text $T$ of length $n$ and a pattern $P$ of length $m$, a position $i$ in $T$, $1 \leq i \leq n - m + 1$, is an $\approx$-occurrence of $P$ in $T$ iff $P \approx T[i : i + m - 1]$.

Definition 2 (\(\approx\)-pattern matching).

**Input:** A text $T \in \Sigma^*$ of length $n$ and a pattern $P \in \Sigma^*$ of length $m \leq n$.

**Output:** All $\approx$-occurrences of $P$ inside $T$.

In the remainder of this paper, we fix text $T$ to be of length $n$ and pattern $P$ to be of length $m$. We also assume that $n = 2m - 1$. Larger texts can be cut into overlapping pieces of length that are less than or equal to $(2m - 1)$ and processed independently. That is, we search for pattern occurrences in each substring $T[1:2m-1], T[m+1:3m-1], \ldots, T[\lfloor \frac{n+1}{m} \rfloor \cdot m + 1 : n]$, independently. For an integer $x$ with $1 \leq x \leq n - m + 1$, a candidate $T_x$ is the substring of $T$ starting from $x$ of length $m$, i.e., $T_x = T[x : x + m - 1]$.

For SCER matchings often it is convenient to encode the strings where $\approx$-equivalence is reduced to the identity. Amir and Kondratovsky \cite{AmirK08} showed that every SCER admits an encoding satisfying the following property\footnote{Lemma 12 in \cite{AmirK08} does not explicitly mention the third property, but their proof entails it.}.

Definition 3 (\(\approx\)-encoding). Let $\Sigma$ and $\Delta$ be alphabets. We say a function $f : \Sigma^* \rightarrow \Delta^*$ is an $\approx$-encoding if

1. for any string $X \in \Sigma^*$, $|X| = |f(X)|$,
2. $f(X[1 : i]) = f(X)[1 : i]$ for any $i \leq |X|$,
3. for two strings $X$ and $Y$ of equal length $k$, $f(X)[i] = f(Y)[i]$ implies $f(X[j + 1 : k])[i - j] = f(Y[j + 1 : k])[i - j]$ for any $j < i \leq k$, and
4. $f(X) = f(Y) \iff X \approx Y$.

Proposition 4. An equivalence relation $\approx$ is an SCER if and only if it admits an $\approx$-encoding.

Proof. It suffices to show the “if” direction. Suppose we have an $\approx$-encoding $f$. If $X \approx Y$, then $f(X) = f(Y)$ by (4) of Definition \cite{AmirK08}. In this case, we have $f(X[j : k])[i] = f(Y[j : k])[i]$ for any $1 \leq j \leq k \leq |X|$ and $1 \leq i \leq k - j + 1$ by $f(X)[i + j - 1] = f(Y)[i + j - 1]$, (3), and (2). Hence, $X[j : k] \approx Y[j : k]$ by (4). □

Standard encodings of SCERs often satisfy the above definition, such as the prev-encoding \cite{AmirK08} for parameterized matching and parent-distance encoding \cite{Sakai13} for cartesian-tree matching. However, the nearest neighbor encoding \cite{Sakai13} for order-preserving matching violates the third condition. Our algorithm for $\approx$-pattern matching proposed in this paper relies on the property of Definition \cite{AmirK08} and does not work with the nearest neighbor encoding. Nonetheless, duel-and-sweep algorithms for order-preserving matching based on the encoding are possible by further elaboration \cite{MeyersonNT00} \cite{MeyersonNT01}, but we will not discuss it in this paper.

Fixing an $\approx$-encoding $f$, we denote $f(X)$ by $\tilde{X}$ for simplicity. In addition, we denote the encoding of $X[x : |X|]$ as $\tilde{X}_x = f(X[x : |X|])$. Thus $\tilde{X}_x = \tilde{X}$. For a string $X$, we suppose that $\tilde{X}$ can be computed in $\tau_{|X|}^\approx$ time and $\tau_{|X|}^\approx$ work in parallel on P-CRCW PRAM. Moreover, we assume that given $\tilde{X}$, $x$, and $k$ such that $x + k - 1 \leq |X|$, to compute $\tilde{X}_x[k]$, i.e. re-encoding the element at position $k$ with respect to suffix $X[x : |X|]$, takes $\xi_k$ time.
and $\xi_k$ work on P-CRCW PRAM. Of course, one can obtain the value $\tilde{X}_a[k]$ by compute the whole $\tilde{X}_a[:k]$ in $\tau_k$ time and $\tau_k$ work, but re-encoding a single position is usually much cheaper. Those parameters are often reasonably small. See Table 1 and Appendix A for the
prev-encoding for parameterized matching and the parent-distance encoding for cartesian-tree matching.

Vishkin’s dueling technique essentially depends on the preferable properties of periods of
strings. Matsuoka et al. \cite{18} have discussed in detail how the classical notion of periods and
their properties can be generalized when considering SCER matching. Unfortunately,
one of the generalizations yield a straightforward adaptation of Vishkin’s algorithm for
SCER matching. Among those, the kind of periods involved in the duel-and-sweep algorithm
discussed in this paper is border-based period.

Definition 5 (Border-based period). Given a string $X$ of length $n$, positive integer $p < n$ is
called a border-based period of $X$ if $X[1:n - p] \approx X[p + 1:n]$.

Throughout the rest of the paper, we will refer to a border-based period as a period.

The family of models of computation used in this work is the priority concurrent-read
concurrent-write (P-CRCW) PRAM [10]. This model allows simultaneous reading from the
same memory location as well as simultaneous writing. In case of multiple writes to the same
memory cell, the P-CRCW PRAM grants access to the memory cell to the processor with
the smallest index.

3 Parallel algorithm for pattern matching under SCERs

We give an overview of the duel-and-sweep algorithm [2] [20]. The pattern is first preprocessed
to obtain a witness table, which is later used to prune candidates during the pattern searching.
As the name suggests, in the duel-and-sweep algorithm, the pattern searching is divided into
two stages: the dueling stage and the sweeping stage. The pattern searching algorithm prunes
candidates that cannot be pattern occurrences, first by performing “duels” between them,
and then by “sweeping” through the remaining candidates to obtain pattern occurrences.

First, we explain the idea of dueling. Suppose $P$ is superimposed on itself with an offset
$a < m$ and the two overlapped regions of $P$ do not match under $\approx$. Then it is impossible for
two candidates $T_x$ and $T_{x+a}$, with offset $a$ to match $P$ simultaneously (see Figure 1). The
dueling stage lets each pair of candidates with such offset $a$ “duel” and eliminates one based
on this observation, so that if candidate $T_x$ gets eliminated during the dueling stage, then
$T_x \not\approx P$. However, the opposite does not necessarily hold true: $T_x$ surviving the dueling stage
does not mean that $T_x \approx P$. On the other hand, it is guaranteed that if distinct candidates
$T_x$ and $T_{x+a}$ that survive the dueling stage overlap, then the suffixes of $T_x$ and $P$ of length
$m-a$ match if and only if so do the prefixes of $T_{x+a}$ and $P$ of the same length. The sweeping stage
takes advantage of this property when checking whether surviving candidates and the
pattern match, so that this stage can also be done quickly.

Prior to the dueling stage, the pattern is preprocessed to construct a witness table based
on which the dueling stage decides which pair of overlapping candidates should duel and
how they should duel. For each offset $0 \leq a < m$, when the overlapped regions obtained by
superimposing $P$ on itself with offset $a$ do not match, we need only one position $i$ to say
that the overlapping regions do not match. We say that $w$ is a witness for the offset $a$ if
$\tilde{P}_{n+1}[w] \neq \tilde{P}[w]$. We denote by $W_P(a)$ the set of all witnesses for offset $a$. We say a witness
$w$ for offset $a$ is tight if $w = \min W_P(a)$. Obviously, $W_P(0) = \emptyset$ if and only if $a = 0$ or $a$ is a
period of $P$. A witness table $W[0:m - 1]$ is an array such that $W[a] \in W_P(a)$ if $W_P(a) \neq \emptyset$. 

\[\frac{a}{w}\]
D. Jargalsaikhan et al.

Algorithm 1 Dueling with respect to $S$. There is one survivor assuming $x$ is not consistent with $y$.

Function $\text{Dueling}(S, x, y)$

1. $w \leftarrow W[y - x]$;
2. if $\tilde{S}[w] = \tilde{P}[w]$ then return $y$;
3. else return $x$;

Figure 1 If $T_x \approx P \approx T_{x+a}$, then the overlapped regions of $P$ superimposed on itself with offset $a$ should match, i.e., $P_{a+1}[1:m-a] \approx P[1:m-a]$. If the overlapped region does not match, there must be a witness $w$ such that $\tilde{P}_{a+1}[w] \neq \tilde{P}[w]$. Candidate positions $x$ and $x+a$ perform a duel using the witness $w$ based on Lemma 6.

When the overlap regions match for offset $a$, which implies that no witness exists for $a$, we express it as $W[a] = 0$.

More formally, in the dueling stage, we “duel” positions $x$ and $x+a$ such that $W_P(a) \neq \emptyset$ based on the following observation (see Figure 1).

Lemma 6. Suppose $w \in W_P(a)$. Then,
\begin{itemize}
  \item if $\tilde{S}_{x+a}[w] = \tilde{P}[w]$, then $T_x \neq P$,
  \item if $\tilde{S}_{x+a}[w] \neq \tilde{P}[w]$, then $T_{x+a} \neq P$.
\end{itemize}

Proof. If $\tilde{S}_{x+a}[w] \neq \tilde{P}[w]$, then by the fourth property of the $\approx$-encoding (Definition 3), $T_{x+a} \neq P$. If $T_{x+a}[w] = P[w] \neq \tilde{P}_{a+1}[w]$, then by the third property of the $\approx$-encoding, $T_x[w + a] \neq \tilde{P}[w + a]$, so $T_x \neq P$.

Based on this lemma, we can safely eliminate either candidate $T_x$ or $T_{x+a}$ without looking into other positions. This process is called dueling and described as Algorithm 1. Since it compares just a single position, it runs in $O(\xi^t_m)$ time and $O(\xi^w_m)$ work assuming that $S$, $\tilde{P}$, and $W$ have already been computed. On the other hand, if the offset $a$ has no witness, i.e. $P[1:m-a] \approx P[a + 1:m]$, no dueling is performed on them. We say that a position $x$ is consistent with $x+a$ if $W_P(a) = \emptyset$.

After the dueling stage, all surviving candidate positions are pairwise consistent. The dueling stage algorithm makes sure that no occurrence gets eliminated during the dueling stage. Taking advantage of the fact that surviving candidates from the dueling stage are pairwise consistent, the sweeping stage prunes them until all remaining candidates match the pattern. By ensuring pairwise consistency of the surviving candidates, the pattern searching algorithm reduces the number of comparisons at a position in the text during the sweeping stage.

Hereinafter, in our pseudo-codes we will use “←” to note assignment operation into a local variable of a processor or assignment operation into a global variable which is accessed by a single processor at a time. We will use “⇐” to note assignment operation into a global
Algorithm 2 Check in parallel whether $X$ and $Y$ match, given $\tilde{X}$ and $\tilde{Y}$. If they do not match, it returns the tight witness.

1. **Function GetTightMismatchPos($\tilde{X}, \tilde{Y}$)**
2. \[ w \leftarrow 0; \]
3. for each $i \in \{1, \ldots, |X|\}$ do in parallel
4. \[ \text{if } \tilde{X}[i] \neq \tilde{Y}[i] \text{ then } w \leftarrow i; \]
5. return $w$;

variable which is accessible from multiple processors simultaneously. In case of a write conflict, the processor with the smallest index succeeds in writing into the memory.

### 3.1 Pattern preprocessing

The goal of the preprocessing stage is to compute a witness table $W[0:m-1]$, where $W[a] = 0$ if $W_P(a) = \emptyset$, and $W[a] \in W_P(a)$ otherwise. Algorithm 2 computes the tight mismatch position for $X$ and $Y$, given $\tilde{X}$ and $\tilde{Y}$.

**Lemma 7.** For strings $X$ and $Y$ of equal length, given $\tilde{X}$ and $\tilde{Y}$, Algorithm 2 computes the tight mismatch position in $O(1)$ time and $O(|X|)$ work on the P-CRCW PRAM.

**Proof.** In Algorithm 2, for each element of $X$, we “attach” a processor to each position of $X$. If $\tilde{X}[i] \neq \tilde{Y}[i]$ for some $i$, the corresponding processor tries to update the shared variable $w$. Recall that in P-CRCW PRAM, the processor with the lowest index will succeed in writing into $w$. Thus, at the end of the algorithm $w$ contains the tight mismatch position. □

One can compute a witness table naively inputting $\tilde{P}[1:m-a]$ and $\tilde{P}_{a+1}[1:m-a]$ for all the offsets $a < m$ to Algorithm 2. However, this naive method costs as much as $\Omega(\xi_m \cdot m^2)$ work. We will present a more efficient algorithm in this subsection.

Our pattern preprocessing algorithm is described in Algorithm 3 and its outline is illustrated in Figure 2. Initially, all entries of the witness table are set to zero. Throughout preprocessing, each element of $W$ is updated at most once. Therefore, at any point of the execution of the preprocessing algorithm, if $W[i] \neq 0$, then it must hold $W[i] \in W_P(i)$. We say that position $i$ is finalized if $W[i] = 0$ implies $W_P(i) = \emptyset$ and $W[i] \neq 0$ implies $W[i] \in W_P(i)$. During the execution of Algorithm 3, the table is divided into two parts. The head is a prefix of a certain length and the tail is the rest suffix. Let us write the head and the tail at the round $k$ of the while loop by $\text{Head}_k$ and $\text{Tail}_k$, respectively. The variable tail in Algorithm 3 represents the starting position of the tail, or equivalently, the length of the head. Throughout the algorithm execution, the tail part is always finalized. On the other hand, the zero entries of the head are not necessarily reliable, such zero positions become fewer and fewer. Consider partitioning the head into blocks of size $2^k$. We will call each block a $2^k$-block, with the last $2^k$-block possibly being shorter than $2^k$. That is, the $2^k$-blocks are $W[i \cdot 2^k : (i+1) \cdot 2^k - 1]$ for $i = 0, \ldots, [h/2^k] - 1$ and $W[[h/2^k] \cdot 2^k : h - 1]$ where $h = |\text{Head}_k|$ is the size of the head. We say that $W[0:x]$ is $2^k$-sparse if every $2^k$-block of $W[0:x]$ contains exactly one zero entry possibly except that the last $2^k$-block has no zero entry. We will guarantee that $\text{Head}_k$ is $2^k$-sparse. Note that when the head is $2^k$-sparse, the unique zero position of the first $2^k$-block $W[0:2^k-1]$ is always 0 ($W[0] = 0$) and $W[1:2^k-1]$ contains no zeros.
Algorithm 3 Parallel algorithm for the pattern preprocessing.

Function PreprocessingParallel()

1. tail ← m, k ← 0; /* tail is the starting position of Tailk */
2. while \(2^k \leq \text{tail}\) do
3. \(p \leftarrow \text{GetZeros}(2^k, 2^{k+1} - 1, k)[0];\)
4. \(W[p] \leftarrow \text{GetTightMismatchPos}(\tilde{P}[1 : m - p], \tilde{P}_{p+1}[1 : m - p]);\)
5. if \(W[p] = 0\) then
6. \(\text{lcp} \leftarrow m - p;\)
7. else
8. \(\text{lcp} \leftarrow W[p] - 1;\)
9. \(\text{old_tail} \leftarrow \text{tail};\)
10. \(\text{tail} \leftarrow \min(\text{old_tail} - 2^k, m - \text{lcp});\)
11. \(\text{SatisfySparsity}(\text{tail} - 1, k);\)
12. \(\text{FinalizeTail}(\text{tail}, \text{old_tail}, p, k);\)
13. \(k \leftarrow k + 1;\)

Algorithm 3
\[\begin{array}{c}
2^k \\
\vdots \\
p_k \\
\hdots \\
2^{k+1}
\end{array}\]

Figure 2 Illustration of the preprocessing invariant. \(W\) is partitioned into head and tail. The head is \(2^k\)-sparse and the tail is finalized. The \(2^k\)-sparsity is achieved by duels. The tail grows by at least \(2^k\) at each round.

Initially, the entire table is the head and the size of the tail is zero: \(\text{Head}_0 = W\) and \(\text{Tail}_0 = \varepsilon\). The head is shrunk and the tail is extended by the following rule. Let the suspected period \(p_k\) at round \(k\) be the first zero position after the index 0, i.e., \(p_k\) is the unique position in the second \(2^k\)-block such that \(W[p_k] = 0\). Then, we let \(\text{Head}_{k+1} = W[0 : m - x - 1]\) and \(\text{Tail}_{k+1} = W[m - x : m - 1]\) for \(x = |\text{Tail}_{k+1}| = \max(|\text{Tail}_{k}| + 2^k, \text{LCP}_P(p_k))\). When \(|\text{Head}_k| < 2^k\), the \(2^k\)-sparsity means that all the positions in the witness table are finalized. So, Algorithm 3 exits the while loop and halts. The goal of this subsection is to show the following theorem.

Theorem 8. Given \(\tilde{P}\), the pattern preprocessing Algorithm 3 computes a witness table in \(O(\zeta_m n \cdot \log^2 m)\) time and \(O(\zeta_m n \cdot m \log^2 m)\) work on the P-CRCW PRAM.

Proof. By Lemmas 12 and 13.

In the remainder of this subsection, we explain how to maintain the \(2^k\)-sparsity of the head and finalize the tail. Before going into the detail, we prepare a technical function GetZeros(l, r, k) in Algorithm 4 which returns positions \(i \in \{l, \ldots, r\}\) such that \(W[i] = 0\) in an array, assuming that \(W[0 : r]\) satisfies the \(2^k\)-sparsity. Algorithm 4 runs in \(O(1)\) time and \(O(r)\) work on the P-CRCW PRAM.
Parallel algorithm for pattern matching problems under SCERs

To fulfill the condition, we control the length of the head part carefully. Concerning the

Comparison with Vishkin’s algorithm

The preprocessing algorithm for exact matching by Vishkin [20] also constructs a witness
table so that it satisfies the $2^k$-sparsity, incrementing $k$, where it has no head/tail separation.
Maintaining the $2^k$-sparsity for the whole table is possible due to the periodicity property
which holds for the exact identity but not for general SCERs. Let $p \leq \lfloor i/2 \rfloor$ be the shortest
period of $P[i]$ for some $i$. In exact matching $P[i] \neq P[i + j - 1]$ implies $P[i - p] \neq P[i + j - 1]$. Thus,
we can update $W[j + p]$ by using $W[j]$, i.e., we may let $W[j + p] = W[j] - p$. However,
this property does not hold on SCERs generally. Still, Vishkin’s technique for keeping the
$2^k$-sparsity can partially be applied to SCER cases under a certain condition (Lemma 9).

Head invariant

First we discuss how the algorithm makes $Head_k$ $2^k$-sparse. We maintain the head so that at
the beginning of round $k$ of Algorithm 3 it satisfies the following invariant properties.

- $Head_k$ is $2^k$-sparse.
- For all positions $i$ of $Head_k$,
  - $W[i] \neq 0$ implies $W[i] \in W_P(i)$,
  - $W[i] \leq |Tail_k| + 2^k$.

The head maintenance procedure $SatisfySparsity$ is described in Algorithm 5. Before
calling the function $SatisfySparsity$, Algorithm 3 finalizes the suspected period $p_k$, the first
position after 0 such that $W[p_k] = 0$. Due to the $2^k$-sparsity, $2^k \leq p_k < 2^{k+1}$. Algorithm 3
finds the suspected period $p_k$ at Line 4 and then finalizes the position $p_k$ at Line 5.
Let us explain how Algorithm 5 works. The task of SatisfySparsity(x,k) is to make $W[0 : x]$ satisfy the $2^{k+1}$-sparsity. In the case where the suspected period $p_k$ is the smallest period of $P$, i.e., $W_P(p_k) = \emptyset$, we have $\text{tail} = m - \text{LCP}_P(p_k) = p_k < 2^{k+1}$ when Algorithm 3 calls SatisfySparsity($\text{tail} - 1, k$). Then the array $A$ obtained at Line 2 is empty and SatisfySparsity($\text{tail} - 1, k$) does nothing. After finalizing Tail$_{k+1}$, which will be explained later, the algorithm will halt without going into the next loop, since $|\text{Head}_{k+1}| \leq m - \text{LCP}_P(p_k) = p_k < 2^{k+1}$. At that moment all positions of $W$ are finalized.

Hereafter we suppose that $p_k$ is not a period of $P$. When SatisfySparsity($\text{tail} - 1, k$) is called, the value of $W[p_k]$ is the tight witness and the first $2^{k+1}$-block contains no zeros except $W[0]$. At that moment, the other part of the head is $2^k$-sparse. To make it $2^{k+1}$-sparse, we perform duels between two zero positions $i$ and $j$ ($i < j$) within each of the $2^{k+1}$-blocks of the head except for the first one. The witness used for the duel between $i$ and $j$ is $W[a]$ for $a = j - i$, which is in the first $2^{k+1}$-block. The following two lemmas ensure that indeed such duels are possible. Suppose that the pattern is superimposed on itself with offsets $i$ and $j$. Lemma 9 below claims that if we already know $w \in W_P(a)$ and $j + w \leq m$, in other words, if the witness lies within the overlap region, then we can obtain a witness for one of the offsets $i$ and $j$ by dueling them using $w$, without looking into other positions. Lemma 10 ensures that indeed we have a witness $w = W[a]$ in our table such that $j + w \leq m$ holds, thanks to the invariant property.

**Lemma 9.** For two offsets $i$ and $j = i + a$ with $a > 0$, suppose $w \in W_P(a)$ and $j + w \leq m$. Then,
1. if the offset $j$ survives the duel, i.e., $\tilde{P}_{j+1}[w] = \tilde{P}[w]$, then $w + a \in W_P(i)$;
2. if the offset $i$ survives the duel, i.e., $\tilde{P}_{j+1}[w] \neq \tilde{P}[w]$, then $w \in W_P(j)$.

**Proof.** If $\tilde{P}_{j+1}[w] \neq \tilde{P}[w]$, then $w \in W_P(j)$ by definition. Suppose $\tilde{P}_{j+1}[w] = \tilde{P}[w]$. The fact $w \in W_P(a)$ means $\tilde{P}[w] \neq \tilde{P}_{a+1}[w]$ and thus $\tilde{P}_{j+1}[w] \neq \tilde{P}_{j+1}[w]$. By Property (3) of the $\approx$-encoding (Definition 4), we have $\tilde{P}_{j+1}[w + a] \neq \tilde{P}[w + a]$, which means $w + a \in W_P(i)$. □

**Lemma 10.** For round $k$, suppose the preprocessing invariant holds true and $W_P(p_k) \neq \emptyset$. Then, when SatisfySparsity is about to be called at Line 10 of Algorithm 3 for any two positions $i$ and $j$ of Head$_{k+1}$ such that $0 < j - i < 2^{k+1}$, it holds that $j + W[j - i] \leq m$.

**Proof.** Let $a = j - i$ and $w = W[a]$. Recall that $a$ belongs to the first $2^{k+1}$-block and $W[a]$ is updated only if $a = p_k$. Suppose $a \neq p_k$. At the beginning of round $k$, by the invariant property, we have $w \leq \text{Tail}_k + 2^k$. Since $j < |\text{Head}_{k+1}| = m - |\text{Tail}_k|$, $j + w \leq j + |\text{Tail}_k| + 2^k < m - |\text{Tail}_{k+1}| + |\text{Tail}_k| + 2^k$. Since $|\text{Tail}_{k+1}| - |\text{Tail}_k| \geq 2^k$, $m - |\text{Tail}_{k+1}| + |\text{Tail}_k| + 2^k < m$. Thus, $j + w \leq m$.

If $a = p_k$, $w = W[p_k]$ is the tight witness for offset $p_k$, i.e., $w = \text{LCP}_P(p_k) + 1$. Since $|\text{Tail}_{k+1}| \geq \text{LCP}_P(p_k)$, $j + w \leq j + |\text{Tail}_{k+1}| + 1$. Since $j < |\text{Head}_{k+1}|$, $j + |\text{Tail}_{k+1}| + 1 \leq |\text{Head}_{k+1}| + |\text{Tail}_{k+1}| \leq m$. We have proved that $j + w \leq m$.

Algorithm 5 updates the witness table in accordance with Lemma 9. In this way, the $2^k$-sparsity of the head and the correctness of (non-zero) witnesses in the head are maintained. The invariants $W[i] \leq |\text{Tail}_k| + 2^k$ and $|\text{Head}_k| + \text{LCP}_P(p_k) \geq m$ are used in the proof of Lemma 10. It remains to show the invariant.

**Lemma 11.** At the beginning of round $k$, for all $i \in \{0, \ldots, 2^k - 1\}$, it holds $W[i] \leq |\text{Tail}_k| + 1$ and for all $i \in \{2^k, \ldots, |\text{Head}_k| - 1\}$, it holds $W[i] \leq |\text{Tail}_k| + 2^k$. 
Parallel algorithm for pattern matching problems under SCERs

Proof. We show the lemma by induction on $k$. At the beginning of round 0, every element of $W$ is zero and $|\text{Tail}_0| = 0$, thus, the claim holds. We will show that the lemma holds for $k+1$ assuming that it is the case for $k$.

Suppose $i < 2^k + 1$ and $i \neq p_k$. Then $W[i]$ is not updated. By induction hypothesis, $W[i] \leq |\text{Tail}_k| + 2^k \leq |\text{Tail}_{k+1}|$ holds. Suppose $i = p_k$. If $W(p_k) = \emptyset$, the algorithm sets $W[p_k] = 0$ and thus the claim holds. If $W(p_k) \neq \emptyset$, the algorithm sets $W[p_k]$ to the tight witness $\text{LCP}_p(p_k) + 1$. Thus, $W[p_k] = \text{LCP}_p(p_k) + 1 \leq |\text{Tail}_{k+1}| + 1$.

Suppose $2^{k+1} - 2^k \leq i < 2^{k+1}$ and $i \neq p_k$. Then $W[i]$ is not updated. By induction hypothesis, $W[i] \leq |\text{Tail}_k| + 2^k \leq |\text{Tail}_{k+1}|$ holds. Suppose $i = p_k$. If $W(p_k) = \emptyset$, the algorithm sets $W[p_k] = 0$ and thus the claim holds. If $W(p_k) \neq \emptyset$, the algorithm sets $W[p_k]$ to the tight witness $\text{LCP}_p(p_k) + 1$. Thus, $W[p_k] = \text{LCP}_p(p_k) + 1 \leq |\text{Tail}_{k+1}| + 1$.

Lemma 12. In the round $k$ of the while loop, Algorithm 6 updates the witness table so that $\text{Head}_{k+1}$ is $2^{k+1}$-sparse in $O(\xi_m)$ time and $O(\xi_w \cdot m/2^k)$ work on P-CRCW PRAM.

Proof. Before the execution of Algorithm 6, since the preprocessing invariant is satisfied and $W[p_k]$ holds the tight witness for offset $p_k$, $W[0 : 2^k - 1]$ contains one zero. Now, let us consider a $2^k$-block $B$ of $\text{Head}_k$ that is not the first $2^k$-block. Since $\text{Head}_k$ satisfies the $2^k$-sparsity, there are at most two zero positions in $B$. Suppose that $B$ has two distinct zero positions $i$ and $i + a$. Since $a < 2^k$, $W[a] \neq 0$. By Lemma 10 for offsets $i$ and $i + a$, $i + a \leq m - W[a]$. Thus, by Lemma 9 at least one of $W[i]$ and $W[i + a]$ is updated as the result of the duel. Thus, after performing duels for all $2^k$-blocks of $\text{Head}_k$, $\text{Head}_k$ satisfies the $2^k$-sparsity.

Since each duel takes $O(\xi_m)$ time and $O(\xi_w)$ work and there are $O(m/2^k)$ duels in total, the overall time and work complexities are $O(\xi_m)$ and $O(\xi_w \cdot m/2^k)$, respectively.

Tail invariant

Next, we discuss how the algorithm finalizes $\text{Tail}_{k+1}$ in the round $k$. This procedure is described in Algorithm 6. For the sake of convenience, we denote by $\mathcal{T}_k$ the set of positions of $\text{Tail}_k$. Since $\text{Tail}_k$ has already been finalized, it is enough to update $W[i]$ for $i \in \mathcal{T}_{k+1} \setminus \mathcal{T}_k$.

We have two cases depending on how much the tail is extended.

The first case where $|\text{Tail}_{k+1}| = |\text{Tail}_k| + 2^k$ is handled naively. Since $\text{Head}_k$ satisfies the $2^k$-sparsity by the invariant, there are at most two zero positions in $\mathcal{T}_{k+1} \setminus \mathcal{T}_k$. Algorithm 6 naively uses Algorithm 2 to finalize those positions.

Now, we consider the case $|\text{Tail}_{k+1}| = \text{LCP}_p(p_k) > |\text{Tail}_k| + 2^k$. The following lemma holds concerning the periodicity under SCERs.

Lemma 13. Suppose that $a$ and $b$ are periods of $X$. If $a + b < |X|$, then $(b + a)$ is a period of $X$. If $a < b$, then $(b - a)$ is a period of $X[1 : |X| - a]$.

This lemma implies that if $p$ is a period of $X$, then so is $qp$ for every positive integer $q \leq (|X| - 1)/p$.

Proof. Let $n = |X|$. Since $a$ is a period of $X$, by the definition $X[1 : n - a] \equiv X[a + b + 1 : n]$. Similarly, since $b$ is a period of $X$, by the definition $X[1 : n - b] \equiv X[1 + b : n]$. Thus, $X[1 : n - a] \equiv X[1 + b : n - a]$. Since $X[1 + b : n - a] \equiv X[1 + b : n - a]$, it means that $(b + a)$ is a period of $X$.
Figure 3 Suppose that $a$ and $b$ are periods of $X$. If $a + b < |X|$, then $(b + a)$ is a period of $X$. If $a < b$, then $(b - a)$ is a period of $X[1 : |X| - a]$.

Figure 4 For offsets $a,b$ such that $m - LCP_P(p) < b < m$ and $a \equiv b \pmod{p}$, if $w \in W_P(b)$, then $(w + b - a) \in W_P(a)$.

Since $a$ and $b$ are periods of $X$, $X[1+b-a:n-a] \approx X[1+b:n]$ and $X[1:n-b] \approx X[1+b:n]$ hold. Thus, by the transitivity property, $(b - a)$ is a period of $X[1 : n - a]$.

We finalize the tail based on the following lemma.

**Lemma 14.** Suppose $m - LCP_P(p) < b < m$. If $w \in W_P(b)$, then $(w + b - a) \in W_P(a)$ for any offset $a$ such that $0 \leq a \leq b$ and $a \equiv b \pmod{p}$.

**Proof.** Figure 4 may help understanding the proof. Suppose $w \in W_P(b)$, i.e., $P_{b+1}[w] \neq P[w]$. Since $p$ is a period of $P[1 : LCP_P(p)]$ and $a \equiv b \pmod{p}$, by Lemma 13, $(b - a)$ is also a period of $P[1 : LCP_P(p)]$, i.e., $P[1 + b - a : LCP_P(p)] \approx P[1 : LCP_P(p) - (b - a)]$. Particularly for the position $w \leq m - b \leq m - a$, we have $P_{b-a+1}[w] = P[w]$. Then, $P_{b-a+1}[w] \neq P_{b+1}[w]$ by the assumption (Figure 4). By Property (3) of the $\approx$-encoding (Definition 3), $P_{b-a+w} \neq P_{b-a+1}[w-b-a+w]$. That is, $(w + b - a) \in W_P(a)$.

Let us partition $T_{k+1} \setminus T_k$ into $p_k$ subsets $S_0, \ldots, S_{p_k-1}$ where $S_s = \{i \in T_{k+1} \setminus T_k \mid i \equiv s \pmod{p_k}\}$, some of which can be empty. Lemma 14 implies that for each $s \in \{0, \ldots, p_k - 1\}$, there exists a boundary offset $b_s$ such that, for every $i \in S_s$, $W_P(i) = \emptyset$ iif $i > b_s$. Fortunately, for many $s$, one can find the boundary $b_s$ very easily, unless $S_s = \emptyset$. Let $q_s = \max S_s$ for non-empty $S_s$. Due to the $2^k$-sparsity and the fact $p_k < 2^{k+1}$, it holds $W[q_s] \neq 0$ for all but at most three $s$. If $W[q_s] \neq 0$, then $q_s$ is the boundary. By Lemma 14, $W[w + q_s - i] \in W_P(i)$ for all $i \in S_s$. Accordingly, Algorithm 6 updates those values $W[i]$ in parallel in Lines 8-11.

On the other hand, for $s$ such that $W[q_s] = 0$, Algorithm 7 uses binary search to find $b_s$ and a witness $w \in W_P(b_s)$ if it exists. Then, following Lemma 14, Algorithm 7 sets in parallel $W[i]$ to $w + (b_s - i)$ where $w \in W_P(b_s)$ for $i \in S_s$ such that $i \leq b_s$ (Line 10). If there is no boundary $b_s$ in $S_s$, then $W_P(i) = \emptyset$ for all $i \in S_s$. We do nothing in that case.

In Algorithm 7 the invariant is as follows. For $i \in S_s$, $W_P(i) = \emptyset$ if $i \leq l \cdot p_k + s$, and $W_P(i) = \emptyset$ if $i \geq r \cdot p_k + s$. Each condition check of the binary search (Line 5) takes $O(\xi_m)$ time and $O(\xi_m \cdot m)$ work. Thus, the overall complexity of Algorithm 7 is $O(\xi_m \cdot \log m)$ time and $O(\xi_m \cdot m \log m)$ work.
Algorithm 6 Finalize $Tail_{k+1}$.

1. Function FinalizeTail($tail, old\_tail, p, k$)
2.     if $old\_tail - tail = 2^k$ then
3.         $Z \leftarrow \text{GetZeros}(tail, old\_tail - 1, k)$; /* $|Z| \leq 2 */
4.         for $i = 0$ to $|Z| - 1$ do
5.             $z \leftarrow Z[i]$;
6.             if $z \neq -1$ then
7.                 $W[z] \leftarrow \text{GetTightMismatchPos}(\tilde{P}[1 : m - z], \tilde{P}_{z+1}[1 : m - z])$
8.         else
9.             for each $i \in \{tail, \ldots, old\_tail - 1\}$ do in parallel
10.                $q \leftarrow j$ where $j \in \{old\_tail - p, \ldots, old\_tail - 1\}$ and $j \equiv i \pmod{p}$;
11.                if $W[i] = 0$ and $W[q] \neq 0$ then $W[i] \leftarrow W[q] + q - i$;
12.                $Z \leftarrow \text{GetZeros}(old\_tail - p, old\_tail - 1, k)$; /* $|Z| \leq 3 */
13.         for $i = 0$ to $|Z| - 1$ do
14.             $z \leftarrow Z[i]$;
15.         if $z = -1$ then Finalize($tail, old\_tail, p, z \pmod{p}$)

Algorithm 7 Finalize $i \in T_{k+1} \setminus T_k$ s.t. $i \equiv s \pmod{p_k}$.

1. Function FinalizeTail($tail, old\_tail, p, s$)
2.     $l \leftarrow [(tail - s)/p] - 1$, $r \leftarrow [(old\_tail - s)/p] + 1$;
3.     while $r - l > 1$ do
4.         $i \leftarrow [(l + r)/2]$, $j \leftarrow i \cdot p + s$;
5.         if GetTightMismatchPos($\tilde{P}[1 : m - j], \tilde{P}_{j+1}[1 : m - j]$) = 0 then $r \leftarrow i$;
6.         else $l \leftarrow i$;
7.     $b_s \leftarrow l \cdot p + s$;
8.     $w \leftarrow \text{GetTightMismatchPos}(\tilde{P}[1 : m - b_s], \tilde{P}_{b_s+1}[1 : m - b_s])$;
9.     for each $i \in \{tail, \ldots, b_s\}$ do in parallel
10.         if $W[i] = 0$ and $i \equiv b_s \pmod{p}$ then $W[i] \leftarrow w + b_s - i$;

Lemma 15. In round $k$, Algorithm 6 finalizes $Tail_{k+1}$ in $O(\xi_m^k \log m)$ time and $O(\xi_m^k \cdot m \log m)$ work on P-CRCW PRAM.

Proof. First, if $|Tail_{k+1}| = |Tail_k| + 2^k$, Algorithm 6 finalizes all positions $i \in T_{k+1} \setminus T_k$ in $O(1)$ time and $O(m)$ work. Next, let us consider the case when $|Tail_{k+1}| = LCP_P(p_k)$. Algorithm 6 finalizes all positions $i \in S_l$ such that $W[q_i] \neq 0$ in $O(1)$ time and $O(m)$ work. Considering $i \in S_r$ such that $W[q_i] = 0$, since $Head_k$ is $2^k$-sparse and $2^k \leq p_k < 2^{k+1}$, there are at most three zero positions in the suffix of length $p_k$ of $Head_k$. Therefore, there are at most three $s$ where $W[q_s] = 0$. Algorithm 6 updates all positions of $S_l$ in parallel in $O(\xi_m^k \log m)$ time and $O(\xi_m^k \cdot m \log m)$ work. Thus, overall Algorithm 6 runs in $O(\xi_m^k \log m)$ time and $O(\xi_m^k \cdot m \log m)$ work.

3.2 Pattern searching

Now we suppose that a witness table of the pattern has been computed. Our pattern searching algorithm prunes candidates in two stages: dueling and sweeping stages. During
the dueling stage, candidate positions duel with each other, until the surviving candidate positions are pairwise consistent. During the sweeping stage, the surviving candidates from the dueling stage are further pruned so that only pattern occurrences survive. To keep track of the surviving candidates, we introduce a Boolean array $C[1 : m]$ and initialize every entry of $C$ to $True$. If a candidate $T_i$ gets eliminated, we set $C[i] = False$. The pattern searching algorithm updates $C$ in such a way that $C[i] = True$ iff $i$ is a pattern occurrence. Entries of $C$ are updated at most once during the dueling and sweeping stages.

**Comparison with Vishkin’s algorithm**

When considering exact matching, Vishkin [20] found that if the pattern is periodic, i.e., $P = Q^j Q'$ for some aperiodic string $Q$, a proper prefix $Q'$ of $Q$, and $j \geq 2$, the problem can be reduced to finding occurrences of $Q$ and $Q'$ in the text. Then a position $i$ is an occurrence of $P$ if and only if $i$ is a starting position of $j$ consecutive occurrences of $Q$ followed by an occurrence of $Q'$. His dueling stage keeps the table $C$ to be $2^k$-sparse in the sense that $C[i] = True$ for at most one position $i$ in every $2^k$-block, incrementing $k$ up to $\lfloor \log |Q|/2 \rfloor$. This can be done without violating the invariant, since the occurrences of an aperiodic string $Q$ are guaranteed to be sparse in the sense that the distance of two consecutive occurrences is bigger than $|Q|/2$. Then the sweeping stage naively checks whether those sparse surviving positions $i$ with $C[i] = True$ are real occurrences. Apparently, this idea does not work in SCER matching. If $P$ has a period $p$ under an SCER, it does not mean that $P$ is a repetition of $Q = P[1 : p]$ or that consecutive occurrences of $Q$ form an occurrence of $P$. Our dueling and sweeping algorithms presented here are quite different from Vishkin’s.

**Dueling stage**

The dueling stage is described in Algorithm 8. A set of positions is said to be **consistent** if all elements in the set are pairwise consistent. During the round $k$, the algorithm partitions the candidate positions into blocks of size $2^k$. Let $C_{k,j} \subseteq \{ (j-1)2^k + 1, \ldots, j \cdot 2^k \}$ be the set of candidate positions in the $j$-th $2^k$-block which have survived after the round $k$. The invariant of Algorithm 8 is as follows.

- At any point of execution of Algorithm 8 all pattern occurrences survive.
- For round $k$, each $C_{k,j}$ is consistent.

Set $C_{k,j}$ is obtained by “merging” $C_{k-1,2j-1}$ and $C_{k-1,2j}$. That is, $C_{k,j}$ shall be a consistent subset of $C_{k-1,2j-1} \cup C_{k-1,2j}$ which contains all the occurrence positions in $C_{k-1,2j-1} \cup C_{k-1,2j}$. After the dueling stage, $C_{\lfloor \log m \rfloor,1}$ is a consistent set including all the occurrence positions. We then let $C[i] = True$ iff $i \in C_{\lfloor \log m \rfloor,1}$. In our algorithm, each set $C_{k,j}$ is represented as an integer array, where elements are sorted in increasing order.

Let us consider merging two respectively consistent sets $A(= C_{k-1,2j-1})$ and $B(= C_{k-1,2j})$ where $A$ precedes $B$, i.e., $\max A < \min B$. Sets $A$ and $B$ should be merged in such a way that the resulting set is consistent and contains all occurrences in $A$ and $B$. That is, we must find a consistent set $C$ such that $\bar{A} \cup \bar{B} \subseteq C \subseteq A \cup B$ where $\bar{A} = \{ a \in A \mid T_a \approx P \}$ and $\bar{B} = \{ b \in B \mid T_b \approx P \}$ are the sets of occurrences in $A$ and $B$, respectively.

**Lemma 16.** Suppose that we are given two respectively consistent position sets $A$ and $B$ such that $A$ precedes $B$. If $a \in A$ and $b \in B$ are consistent, then $A_{\leq a} \cup B_{\geq b}$ is also consistent, where $A_{\leq a} = \{ i \in A \mid i \leq a \}$ and $B_{\geq b} = \{ j \in B \mid j \geq b \}$.

**Proof.** It is enough to show that if $i$ and $j$ are consistent and $j$ and $k$ are consistent, then $i$ and $k$ are consistent for $i < j < k$. This claim can be rephrased so that if $j - i$ and $k - j$ are periods, then so is $k - i$. This is an immediate corollary to Lemma 13. □
14 Parallel algorithm for pattern matching problems under SCERs

**Algorithm 8** Parallel algorithm for the dueling stage.

Function DuelingStageParallel()

for each $j \in \{1, \ldots, m\}$ do in parallel

$C_{0,j}[1] \leftarrow j$

$k \leftarrow 1$

while $k \leq \lceil \log m \rceil$ do

for each $j \in \{1, \ldots, \lceil m/2^k \rceil\}$ do in parallel

$A \leftarrow C_{k-1,j-1}$, $B \leftarrow C_{k-1,2j}$;

$\langle a, b \rangle \leftarrow \text{Merge}(A, B)$;

Let $C_{k,j}$ be array of length $(a + |B| - b + 1)$;

for each $i \in \{1, \ldots, a\}$ do in parallel

$C_{k,j}[i] \leftarrow A[i]$;

for each $i \in \{b, \ldots, |B|\}$ do in parallel

$C_{k,j}[a + i - b + 1] \leftarrow B[i]$;

$k \leftarrow k + 1$;

Initialize all elements of $C$ to False;

for each $i \in \{1, \ldots, \lceil \log m \rceil\}$ do in parallel

$C[|C|_{\lceil \log m \rceil - 1}][i] \leftarrow \text{True}$;

Therefore, it suffices to find $(a, b) \in A \times B$ such that $a \geq \max \hat{A}$, $b \leq \min \hat{B}$, and $a$ and $b$ are consistent. Then, $A_{\leq a} \cup B_{\geq b}$ has the desired property.

To find such a pair $(a, b)$, let us consider a grid $G$ of size $(|A| + 2) \times (|B| + 2)$. Figure 5 illustrates the grid, where indices of $A$ and $B$ are presented along the directions of rows and columns, respectively. For $1 \leq i \leq |A|$ and $1 \leq j \leq |B|$, $G[i][j]$ represents the result of the duel between $A[i]$ and $B[j]$ using the witness table $W$, which are the $i$-th and $j$-th smallest elements of $A$ and $B$, respectively. We define $G[i][j] = 0$ if $W[d] = 0$ for $d = B[j] - A[i]$. If $W[d] \neq 0$ and $A[i]$ wins the duel, then $G[i][j] = -1$. Otherwise, $B[j]$ wins the duel and $G[i][j] = 1$. For the sake of explanatory convenience, we pad grid $G$ with $-1$s along the leftmost column, with $1$s along the bottom row, and with $0$s along the upper row and rightmost column. Specifically, $G[i][0] = -1$ for $i \in \{0, \ldots, |A|\}$, $G[|A| + 1][j] = 1$ for $j \in \{0, \ldots, |B|\}$, $G[i][|B| + 1] = 0$ for $i \in \{1, \ldots, |A| + 1\}$, and $G[0][j] = 0$ for $j \in \{1, \ldots, |B| + 1\}$. We will not compute the whole $G$, but this concept helps understanding the behavior of our algorithm.

In terms of the grid representation, our goal is to find a coordinate $(i, j)$ such that $G[i][j] = 0$ and it is to the lower left of $(i, j)$ (brown dot in Figure 5) such that $\hat{A}[i'] \in \hat{A}$ and $\hat{B}[j'] \in \hat{B}$ does not have the desired property, where $A_{\leq a} \cup B_{\geq b}$ are assumed to be empty.

Lemma 15 implies that if $G[i'][j'] = 0$ then $G[i'][j'] = 0$ for any $i' \leq i$ and $j' \geq j$. Therefore, grid $G$ can be divided into two regions: the upper-right region that consists of only 0 and the rest that consists of a mixture of $-1$ and 1. The boundary line looks like a step function. The distributions of 1 and $-1$ in the non-zero region are not totally random. Since occurrences will never lose the duel, if $A[i] \in \hat{A}$, then row $i$ consists of non-positive elements only, and if $B[j] \in \hat{B}$, then column $j$ consists of non-negative elements only. Particularly, $G[1][j] = 0$. The following lemma strengthens this observation.

**Lemma 17.** If $A[i] \leq \max \hat{A}$, then row $i$ consists only of non-positive elements. Similarly,
Lemma 17. Our algorithm outputs $B$ if $\xi_j = 0$. We show that if $\xi_j = 0$, then $G[i][j] = 1$, $G[i][j-1] = 1$, and $G[i][j+1] = 1$, and $G[i+1][j'] = 1$ for some $j'$. If there are more than one such coordinate, the smallest $i$ will be chosen by the priority. The output coordinate is indicated by the green circle above. Then the obtained set consists of the elements represented by the two green lines.

![Figure 5](image-url) Padded grid $G$ given two consistent sets $A$ and $B$. The grid is separated into the zero region and the non-zero region by the dotted boundary line (Lemma 16). The coordinate $(i, j)$ is indicated by the brown dot. The red- and blue-shaded areas consist of $-1$ and $1$ only, respectively (Lemma 17). Our algorithm outputs $(i, j)$ such that $G[i][j] = 0$, $G[i][j-1] = 1$, and $G[i+1][j'] = 1$, and $G[i][j+1] = 1$ for some $j'$. If there are more than one such coordinate, the smallest $i$ will be chosen by the priority. The output coordinate is indicated by the green circle above. Then the obtained set consists of the elements represented by the two green lines.

If $B[j] \leq \min B$, then column $j$ consists only of non-negative elements.

Proof. We prove the first claim in the lemma. The second claim can be proven in the same way. We show that if $i \leq i$ and $G[i][j] \neq 0$, then $G[i][j] = 1$ for any $1 \leq j \leq |B|$. Let $a = A[i]$, $\hat{a} = \max \hat{A}$, $b = B[j]$, and $\hat{b} = \min \hat{B}$, and suppose the inconsistency of $a$ and $b$ is witnessed by $W[b - a] = w \neq 0$, i.e., $P[w] \neq P_{\hat{b} - a + 1}[w]$. Since $T_a \approx P$, $T[b : m + \hat{a} - 1] \approx P[b - \hat{a} + 1 : m]$, which implies $T_{\hat{b}}[w] = P_{\hat{b} - a + 1}[w]$. On the other hand, since $a$ and $\hat{a}$ are consistent, i.e., $P[1 : m - (\hat{a} - a)] \approx P[\hat{a} - a + 1 : m]$, we have $P[b - \hat{a} + 1 : m - (\hat{a} - a)] \approx P[b - a + 1 : m]$, which implies $P_{\hat{b} - a + 1}[w] = P_{\hat{b} - a + 1}[w]$. Therefore, $\hat{T}_{\hat{b}}[w] = P_{\hat{b} - a + 1}[w] = \hat{P}_{\hat{b} - a + 1}[w] \neq \hat{P}[w]$. Hence, $a$ wins the duel against $b$ and thus $G[i][j] = 0$.

Algorithm 9 firstly finds the unique column $j_i$ for each row $i$ such that $G[i][j_i] \neq 0$ and $G[i][j_i + 1] = 0$. Among those boundary coordinates, the algorithm finds a neighbour pair $(i, j_i)$ and $(i + 1, j_{i+1})$ such that $G[i][j_i] = 1$ and $G[i + 1][j_{i+1}] = 1$. Then, it outputs $(i, j_i + 1)$.

Lemma 18. Algorithm 9 finds a coordinate $(i_*, j_*)$ such that $i_* \geq \hat{i}$, $j_* \leq \hat{j}$, and $G[i_*][j_*] = 0$ in $O(\xi_m \log |B|)$ time with $O(\xi_m^2 |A| \log |B|)$ work.

Proof. In the while-loop, always $G[i][j'] \neq 0$, $G[i][j''] = 0$, and $j_0 < i_1$ hold. When exiting the first while-loop, $D[i] = j'$ such that $G[i][j'] \neq 0$ and $G[i][j' + 1] = 0$ hold for all $i$. Then, the algorithm finds $i$ such that $G[i][j] = -1$, $G[i][j + 1] = 0$, $G[i + 1][j'] = 1$, where $j = D[i]$ and $j' = D[i + 1]$. Since $G[i][j] = -1$, by Lemma 17, $j_i < \hat{j}$. Similarly, $G[i + 1][j'] = 1$ implies $i + 1 > i$. Thus, $i_* = \hat{i}$ and $j_* = j + 1 \leq \hat{j}$ satisfy the desired property by Lemma 16.

Since a duel takes $O(\xi_m^2)$ time and $O(\xi_m^2)$ work, we obtain the claimed complexity.

Lemma 19. Given a witness table, $\hat{P}$, and $\hat{T}$, Algorithm 9 performs the dueling stage in $O(\xi_m^2 \log^2 m)$ time and $O(\xi_m^2 m \log^2 m)$ work on P-CRCW-PRAM.
16 Parallel algorithm for pattern matching problems under SCERs

Algorithm 9 Merge two consistent sets \( A \) and \( B \)

Function Merge\( (A, B) \)

1. for each \( i \in \{0, \ldots, |A| + 1 \} \) do in parallel
2. \( j' \leftarrow 0, j'' = |B| + 1; \)
3. while \( j'' - j' > 1 \) do
4. \( j \leftarrow \lfloor (j' + j'')/2 \rfloor; \)
5. if \( G[i][j] = 0 \) then
6. \( j'' \leftarrow j; \)
7. else
8. \( j' \leftarrow j; \)
9. \( D[i] \leftarrow j' \);
10. for each \( i \in \{0, \ldots, |A| \} \) do in parallel
11. if \( G[i][D[i]] = -1 \) and \( G[i + 1][D[i + 1]] = 1 \) then
12. \( i_* \leftarrow i, j_* \leftarrow D[i] + 1; \)
13. return \((i_*, j_*);\)

Proof. Since the while-loop runs \( O(\log m) \) times and each loop takes \( O(\xi_m \log m) \) time by Lemma 18, the overall time complexity is \( O(\xi_m \log^2 m) \). Now, let us look at the work complexity. Concerning each round \( k \) of the while-loop of Algorithm 8, Merge\( (A, B) \) takes \( O(\xi_m \cdot 2^k \log m) \) work by Lemma 18, and thus it takes \( O(m/2^k) \cdot \xi_m \cdot 2^k \log m) \) work. Since \( k \in \{0, \ldots, |\log m|\} \), the overall work complexity is \( O(\xi_m \log^2 m) \).

Sweeping stage

The sweeping stage is described in Algorithm 10. The sweeping stage updates \( C \) until \( C[i] = True \) iff \( i \) is a pattern occurrence. All entries in \( C \) are updated at most once. Recall that all candidates that survived from the dueling stage are pairwise consistent. In addition to \( C \), we will create a new integer array \( R[1 : m] \). Throughout the sweeping stage, we have the following invariant properties:

- if \( C[x] = False \), then \( T_x \neq P \)
- if \( C[x] = True \), then \( LCP(T_x, P) \geq R[x] \)

The purpose of bookkeeping this information in \( R \) is to ensure that the sweeping stage algorithm uses \( O(n) \) processors in each round. We do not want to access the same position of the text for each candidate covering the position. For two consistent candidate positions \( x \) and \( x + a \) with \( a > 0 \), once we have calculated the value \( r = LCP(T_x, P) \), we know that \( LCP(T_{x+a}, P) \geq r - a \) for free, i.e., \( T_{x+a}[1 : r - a] = P[1 : r - a] \). Then it suffices to check \( T_{x+a}[r - a + 1 : m] = P[r - a + 1 : m] \). We keep the value \( r - a \) in \( R[x + a] \) for this trick, if \( r - a > 0 \). Throughout this section, we assume that a processor is attached to each position of \( C \) and \( T \).

For each stage \( k \), \( C \) is divided into \( 2^k \)-blocks. Unlike the preprocessing algorithm, \( k \) starts from \( |\log m| \) and decreases with each round until \( k = 0 \). Let us look at each round in more detail. For the \( b \)-th \( 2^k \)-block of \( C \), we pick as the “pivot” the smallest index \( x_{k,b} \) in the second half of the \( 2^k \)-block such that \( C[x_{k,b}] = True \). In Algorithm 10, we introduce array \( \tilde{P}_0[0 : |m/2^k|] \) where \( \tilde{P}_0[b] = x_{k,b} \). For each \( x_{k,b} \), the algorithm computes \( LCP(T_{x_{k,b}}, P) \) exactly and store the value in \( R[x_{k,b}] \) on Lines 11-13. Suppose that \( LCP(T_{x_{k,b}}, P) < m \), i.e.,
Algorithm 10. Parallel algorithm for the sweeping stage

1: Function SweepingStageParallel()
2: 
3: create $R[1 : m]$ and initialize elements of $R$ to 0;
4: $k ← \lceil \log m \rceil$;
5: while $k \geq 0$ do
6:   create Pivot[0 : \lfloor m/2^k \rfloor] and initialize its elements to $-1$;
7:   for each $i \in \{1, \ldots, m\}$ do in parallel
8:     if $C[i] = \text{True}$ and $(i \mod 2^k) > 2^{k-1}$ then
9:       Pivot[\lfloor i/2^k \rfloor] $\leftarrow i$;
10:   for each $b \in \{0, \ldots, \lfloor m/2^k \rfloor\}$ do in parallel
11:     $x ← \text{Pivot}[b]$;
12:     if $x \neq -1$ then
13:       $w ← \text{GetTightMismatchPos}(\tilde{P}[R[x] + 1 : m], \tilde{T}_x[R[x] + 1 : m])$;
14:       if $w = 0$ then $R[x] \leftarrow m$;
15:       else $R[x] \leftarrow R[x] + w - 1$;
16:   for each $i \in \{1, \ldots, m\}$ do in parallel
17:     if $i \leq x$ and $R[x] \leq m - (x - i) - 1$ then $C[i] \leftarrow \text{False}$;
18:     if $i > x$ and $C[i] = \text{True}$ then $R[i] \leftarrow R[x] - (i - x)$;
19:   $k ← k - 1$;

Let $T_{x_k,b} \neq P$ and $w = \text{LCP}(T_{x_k,b}, P) + 1$ is the tight mismatch position. Since all surviving candidate positions are pairwise consistent, if $T_{x_k,b} \neq P$, then any candidate $T_{x_k,b-a}$ that “covers” $w$ cannot match the pattern. Generally, we have the following.

Lemma 20. If two candidate positions $x$ and $(x-a)$ with $a > 0$ are consistent and $\text{LCP}(T_x, P) \leq m - a - 1$, then $(x-a)$ is not an occurrence.

Proof. Let $w = \text{LCP}(T_x, P) + 1$. Then $w \leq m - a$ and $T_x[1 : m - a] \neq P[1 : m - a]$. Since $x$ is consistent with $(x-a)$, $P[a+1 : m] \approx P[1 : m-a] \neq T_x[1 : m-a] \approx T_{x-a}[a+1 : m]$, which means that $(x-a)$ is not a pattern occurrence. ▶

Based on Lemma 20, Algorithm 10 updates $C[i]$ for indices $i$ in the first half of each $2^k$-block at Line 16. On the other hand, at Line 17, the algorithm updates the values of $R[i]$ for indices $i$ in the second half of the block if $C[i] = \text{True}$. Since the surviving candidates are pairwise consistent, for candidate positions $(x_k,b+a)$ such that $a > 0$, $T_{x_k,b+a}[1 : r] \approx P[1 : r]$ for $r = R[x_k,b] - a$. In this way, the algorithm maintains the invariant properties. When $k = 0$, all the $2^k$-blocks contain just one position $x$ and $R[x]$ is set to be exactly $\text{LCP}(T_x, P)$ by Lines 11–13 unless $C[x] = \text{False}$ at that time. Then, if $R[x] < m$, then $C[x]$ will be $\text{False}$ on Line 16. That is, when the algorithm halts, $C[x] = \text{True}$ iff $T_x \approx P$.

It remains to show the efficiency of the algorithm.

Lemma 21. The value of each element of $R$ is never decreased.

Proof. Suppose that $R[i]$ is updated at round $k$ and then later at round $k'$, where $k > k'$ holds. Let $x$ and $x'$ be the pivots of the $2^k$- and $2^{k'}$-blocks where $i$ belongs at round $k$ and $k'$, respectively. It must hold that $x \leq x' \leq i$. For $y = \text{LCP}(T_x, P)$ and $d = x'-x$, we have $T_x[1 : y] \approx P[1 : y]$ and thus $T_{x'}[1 : y - d] = T_x[1 + d : y] \approx P[1 + d : y]$. ▶
Parallel algorithm for pattern matching problems under SCERs

18

![Figure 6](image-url) Before round $k$, for two surviving candidates $T_i$ and $T_j$ such that $j - i \geq 2^k$, $i + m - 1 < j + R_k[j]$.

Since $x$ and $x'$ are consistent, $P[1 + d : y] \approx P[1 : y - d]$. Hence, $T_{x'}[1 : y - d] \approx P[1 : y - d]$, i.e., $LCP(T_{x'}, P) \geq y - d$. Therefore, $LCP(T_x, P) + x \leq LCP(T_{x'}, P) + x'$ and thus $LCP(T_x, P) - (i - x) \leq LCP(T_{x'}, P) - (i - x')$ holds. That is, at Line 17 the value of $R[i]$ cannot be decreased.

**Lemma 22.** After the round $k$, for two surviving candidate positions $i$ and $j$ with $i < j$ that do not belong to the same $2^{k-1}$-block of $C$, $i + m \leq j + R_k[j]$.

**Proof.** See Figure 6. Before round $\lceil \log m \rceil$, which can be seen as after round $\lceil \log m \rceil + 1$, since all candidate positions belong to the same $2^{\lceil \log m \rceil}$-block, the statement holds (base case). Assuming that the statement holds after the round $(k + 1)$, we prove that it also holds after the round $k$. Let $R_{k+1}$ and $R_k$ be the states of the array $R$ after the rounds $(k + 1)$ and $k$, respectively. First, let us consider the case when surviving candidate positions $i$ and $j$ do not belong to the same $2^k$-block of $C$. Obviously, $i$ and $j$ cannot belong to the same $2^k$-block.

By the induction hypothesis, $i + m \leq j + R_{k+1}[j]$. Since $R_k[j] \geq R_{k+1}[j]$ by Lemma 21, $i + m \leq j + R_k[j]$.

Now, let us consider the case when candidate positions $i$ and $j$ belong to the same $2^k$-block of $C$. During round $k$, for each $2^k$-block of $C$, Algorithm 10 chooses as surviving candidate position $x_{k,b}$ which is the smallest index in the second half of the $2^k$-block. Thus, two surviving candidates positions $i$ and $j$ of the $b$-th $2^k$-block belong to different $2^k$-blocks iff $i < x_{k,b} \leq j$. For $T_i$ to be a surviving candidate after round $k$, it must be the case that $m + i \leq LCP(T_{x_{k,b}}, P) + x_{k,b}$. For $T_j$, Algorithm 10 updates $R_k[j]$ to $LCP(T_{x_{k,b}}, P) - (j - x_{k,b})$. Substituting it into the previous inequality, we get $m + i \leq R_k[j] + (j - x_{k,b}) + x_{k,b} = R_k[j] + j$.

**Lemma 23.** Each round of the while loop of Algorithm 10 can be performed in $O(\xi_m)$ time with $O(n)$ processors.

**Proof.** See Figure 6. Obviously it runs in constant time except for the computation at Line 11 where each processor attached to position $i$ is used for re-encoding $\hat{T}[i]$ into $\hat{T}_x[i - x + 1]$ and comparing the value with $\hat{P}[i - x + 1]$ for some $x$. Indeed, there is at most one $b$ such that $x_{k,b} + R[x_{k,b}] \leq i < x_{k,b} + m$, since $x_{k,b-1} + m \leq x_{k,b} + R[x_{k,b}]$ for all $b \in \{1, \ldots, \lceil m/2^k \rceil \}$ by Lemma 22.

**Lemma 24.** Given $\hat{P}$ and $\hat{T}$, the sweeping stage algorithm finds all pattern occurrences in $O(\xi_m \log m)$ time and $O(\xi_m^m \cdot m \log m)$ work on the P-CRCW PRAM.
Figure 7 Illustrating the sweeping stage. The shaded regions of the text \( T \) are referenced during round \( k \). Those referenced regions do not overlap.

**Proof.** The outer loop of Algorithm 10 runs \( O(\log m) \) times. By Lemma 23, each loop runs in \( O(\xi^t_m) \) time and \( O(\xi^w_m \cdot m) \) processors. Thus, the total time is \( O(\xi^t_m \cdot \log m) \) and total work is \( O(\xi^w_m \cdot m \log m) \).

By Theorem 8 and Lemmas 19 and 24, we obtain the main theorem. Recall that when \( n \geq 2m \), \( T \) is cut into overlapping pieces of length \((2m - 1)\) and each piece is processed independently.

\textbf{Theorem 25.} Given a witness table, \( \tilde{P} \), and \( \tilde{T} \), the pattern searching solves the pattern searching problem under SCER in \( O(\xi^t_m \cdot \log^2 m) \) time and \( O(\xi^w_m \cdot n \log^2 m) \) work on the P-CRCW PRAM.

\section{Conclusion}

Dueling \cite{20} is a powerful technique, which enables us to perform pattern matching efficiently. In this paper, we have generalized the dueling technique for SCERs and have proposed a duel-and-sweep algorithm that solves the pattern matching problem for any SCER. Our algorithm is the first algorithm to solve any SCER pattern matching problem in parallel. Given a witness table, \( \tilde{P} \), and \( \tilde{T} \), we have shown that pattern searching under any SCER can be performed in \( O(\xi^t_m \log^2 m) \) time and \( O(\xi^w_m n \log^2 m) \) work on P-CRCW PRAM. Given \( \tilde{P} \), a witness table can be constructed in \( O(\xi^t_m \log^2 m) \) time and \( O(\xi^w_m \cdot m \log^2 m) \) work on P-CRCW PRAM. The third condition of \( \approx \)-encoding in Definition 3 ensures the generality of our duel-and-sweep algorithm for SCERs. However, some standard encoding method of an SCER, namely the nearest neighbor encoding for order-preserving matching, does not fulfill the third condition. We do not know if there is an alternative encoding for order-preserving matching that fulfills the condition and is computationally as cheap as the nearest neighbor encoding. Nevertheless, Jargalsaikhan et al. \cite{11, 12} succeeded in designing a parallel duel-and-sweep algorithm for order-preserving matching using the nearest neighbor encoding, which appears quite similar to the SCER algorithm proposed in this paper. In our future work, we would like to investigate the relation between the encoding function and the dueling technique and further generalize the definition of encoding so that it becomes more inclusive.
References

1. Amihood Amir, Yonatan Aumann, Moshe Lewenstein, and Ely Porat. Function matching. *SIAM Journal on Computing*, 35(5):1007–1022, 2006.
2. Amihood Amir, Gary Benson, and Martin Farach. An alphabet independent approach to two-dimensional pattern matching. *SIAM Journal on Computing*, 23(2):313–323, 1994.
3. Amihood Amir and Eitan Kondratovsky. Sufficient conditions for efficient indexing under different matchings. In *Proceedings of 30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
4. Brenda S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of computer and system sciences*, 52(1):28–42, 1996.
5. Omer Berkman, Baruch Schieber, and Uzi Vishkin. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *Journal of Algorithms*, 14(3):344–370, 1993.
6. Ayelet Butman, Revital Eres, and Gad M. Landau. Scaled and permuted string matching. *Information processing letters*, 92(6):293–297, 2004.
7. Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Searching for jumbled patterns in strings. In *Proceedings of the Prague Stringology Conference 2009*, pages 105–117, 2009.
8. Richard Cole, Carmit Hazay, Moshe Lewenstein, and Dekel Tsur. Two-dimensional parameterized matching. *ACM Transactions on Algorithms (TALG)*, 11(2):12, 2014.
9. Davaajav Jargalsaikhan. Generalized dictionary matching under substring consistent equivalence relations. In *Proceedings of the 14th International Workshop on Algorithms and Computation*, pages 120–132, 2020.
10. Joseph JáJá. *An introduction to parallel algorithms*, volume 17. Addison-Wesley Reading, 1992.
11. Davaajav Jargalsaikhan, Diptarama, Yohei Ueki, Ryo Yoshinaka, and Ayumi Shinohara. Duel and sweep algorithm for order-preserving pattern matching. In *Proceedings of the 44th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2018)*, pages 624–635, 2018.
12. Davaajav Jargalsaikhan, Diptarama Hendrian, Ryo Yoshinaka, and Ayumi Shinohara. Parallel duel-and-sweep algorithm for the order-preserving pattern matching. In *Proceedings of the 46th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2020)*, pages 211–222, 2020.
13. Davaajav Jargalsaikhan, Diptarama Hendrian, Ryo Yoshinaka, and Ayumi Shinohara. Parallel algorithm for pattern matching problems under substring consistent equivalence relations. In *33rd Annual Symposium on Combinatorial Pattern Matching, CPM 2022, June 27-29, 2022, Prague, Czech Republic*, volume 223 of *LIPIcs*, pages 28:1–28:21, 2022.
14. Natsumi Kikuchi, Diptarama Hendrian, Ryo Yoshinaka, and Ayumi Shinohara. Computing covers under substring consistent equivalence relations. In *Proceedings of the 27th International Symposium on String Processing and Information Retrieval*, pages 131–146, 2020.
15. Jinil Kim, Peter Eades, Rudolf Fleischer, Seok-Hee Hong, Costas S. Iliopoulos, Kunsoo Park, Simon J. Puglisi, and Takeshi Tokuyama. Order-preserving matching. *Theoretical Computer Science*, 525:68–79, 2014.
16. Donald E. Knuth, James H. Morris, Jr, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.
17. Marcin Kubica, Tomasz Kulczyński, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear time algorithm for consecutive permutation pattern matching. *Information Processing Letters*, 113(12):430–433, 2013.
18. Yoshiaki Matsuoka, Takahiro Aoki, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Generalized pattern matching and periodicity under substring consistent equivalence relations. *Theoretical Computer Science*, 656:225–233, 2016.
A Examples of encoding

Prev-encoding for parameterized matching

For a string $X$ of length $n$ over $\Sigma \cup \Pi$, where $\Pi$ is an alphabet of parameter symbols and $\Sigma$ is an alphabet of constant symbols, the prev-encoding \[^{[4]}\] for $X$, denoted by $\text{prev}_X$, is defined to be a string over $\Sigma \cup \mathbb{N}$ of length $n$ such that for each $1 \leq i \leq n$,

$$
\text{prev}_X[i] = \begin{cases} 
X[i] & \text{if } X[i] \in \Sigma, \\
0 & \text{if } X[i] \in \Pi \text{ and } X[i] \neq X[j] \text{ for } 1 \leq j < i, \\
i - k & \text{if } X[i] \in \Pi \text{ and } k = \max\{j \mid X[j] = X[i] \text{ and } 1 \leq j < i\}.
\end{cases}
$$

\[\Diamond\] Theorem 26. Given a string $X$ of length $n$, $\text{prev}_X$ can be computed in $O(\log n)$ time and $O(n \log n)$ work on P-CRCW PRAM. Moreover, given $\text{prev}_X$, $\text{prev}_{X[x:n]}[i]$ can be computed in $O(1)$ time and $O(1)$ work.

Proof. Without loss of generality, we assume that $\Pi$ forms a totally ordered domain. We will construct the following string $X'$ from $X$. We define a new symbol, say $\infty$, such that, for any element $\pi \in \Pi$, $\pi$ is less than $\infty$. For $1 \leq i \leq |X|$, $X'[i] = X[i]$ if $X[i] \in \Pi$ and $X'[i] = \infty$ if $X[i] \in \Sigma$. For $X'$, we construct $L_{max}X'$, which is defined as $L_{max}X'[i] = j$ if $X'[j] = \max_{k < i} \{X'[k] \mid X'[k] \leq X'[i]\}$. We use the rightmost (largest) $j$ if there exist more than one such $j < i$. If there is no such $j$, then we define $L_{max}X'[i] = 0$. Suppose that $X[i] \in \Pi$ for $1 \leq i \leq |X|$. After computing $L_{max}X'$, $\text{prev}_X[i] = i - L_{max}X'[i]$ if $X[i] = X[L_{max}X'[i]]$. If $L_{max}X'[i] = 0$ or $X[i] \neq X[L_{max}X'[i]]$, then $X[i]$ is the first occurrence of this letter. Thus, $\text{prev}_X$ can be computed from $L_{max}X'$ in $O(1)$ time and $O(n)$ work. Since $L_{max}X'$ can be computed in $O(\log n)$ time and $O(n \log n)$ work \[^{[12]}\], overall complexities are $O(\log n)$ time and $O(n \log n)$ work.

Given $\text{prev}_X$, $\text{prev}_{X[x:n]}[i]$ can be computed in the following manner in $O(1)$ time and $O(1)$ work.

$$
\text{prev}_{X[x:n]}[i] = \begin{cases} 
0 & \text{if } X[x + i - 1] \in \Pi \text{ and } \text{prev}_X[x + i - 1] \geq i, \\
\text{prev}_X[x + i - 1] & \text{otherwise}.
\end{cases}
$$

\[\Diamond\]

Parent-distance encoding for cartesian-tree matching

For a string $X$ over a totally ordered alphabet, its parent-distance encoding \[^{[19]}\] for cartesian-tree matching $PD_X$ is defined as follows.

$$
PD_X[i] = \begin{cases} 
0 & \text{if there is no } j < i \text{ such that } X[j] \leq X[i], \\
i - \max_{1 \leq j < i} \{j \mid X[j] \leq X[i]\} & \text{otherwise}.
\end{cases}
$$
Theorem 27. Given a string $X$ of length $n$, $PD_X$ can be computed in $O(\log n)$ time and $O(n \log n)$ work on P-CRCW PRAM. Moreover, given $PD_X$, $PD_{X[x:n][i]}$ can be computed in $O(1)$ time and $O(1)$ work.

Proof. For $1 \leq i \leq n$, $PD_X[i]$ is the nearest smaller value to the left of $X[i]$. Since the all-smaller-nearest-value problem can be solved in $O(\log n)$ time and $O(n \log n)$ work on P-CRCW PRAM by Berkman et al. [5], $PD_X$ can be computed in $O(\log n)$ time and $O(n \log n)$ work on P-CRCW PRAM.

Given $PD_X$, $PD_{X[x:n][i]}$ can be computed in the following manner in $O(1)$ time and $O(1)$ work.

$$PD_{X[x:n][i]} = \begin{cases} 0 & \text{if } PD_X[x + i - 1] \geq i, \\ PD_X[x + i - 1] & \text{otherwise.} \end{cases}$$