



5 Parallel String Matching

10 March 2021

Sebastian Wild


5 Parallel String Matching

- 5.1 Elementary Tricks
- 5.2 Periodicity
- 5.3 String Matching by Duels

Parallelizing string matching

- ▶ We have seen a plethora of string matching methods
- ▶ But all efficient methods seem inherently sequential
Indeed, they became efficient only after building on knowledge from previous steps!

Sounds like the *opposite* of parallel!



↪ This unit:

- ▶ How well can we parallelize string matching?
- ▶ What new ideas can help?

Here: string matching = find *all* occurrences of P in T (more natural problem for parallel)
always assume $m \leq n$

5.1 Elementary Tricks

Embarrassingly Parallel

- ▶ A problem is called “*embarrassingly parallel*” if it can immediately be split into *many, small subtasks* that can be solved completely *independently* of each other
- ▶ Typical example: sum of two large matrices (all entries independent)
- ↪ best case for parallel computation (simply assign each processor one subtask)
- ▶ Sorting is not embarrassingly parallel
 - ▶ no obvious way to define many *small* (=efficiently solvable) subproblems
 - ▶ but: some subtasks of our algorithms are, e. g., comparing all elements with pivot

Clicker Question



Is the string-matching problem “embarrassingly parallel”?

- A** Yes
- B** No
- C** Only for $n \gg m$
- D** Only for $n \approx m$

sli.do/comp526

Click on “Polls” tab

Elementary parallel string matching

Subproblems in string matching:

- ▶ string matching = check all guesses $i = 0, \dots, n - m - 1$
- ▶ checking one guess is a subtask!

Elementary parallel string matching

Subproblems in string matching:

- ▶ string matching = check all guesses $i = 0, \dots, n - m - 1$
- ▶ checking one guess is a subtask!

Approach 1:

- ▶ Check all guesses in parallel

↪ Time: $\Theta(m)$

↪ Work: $\Theta((n - m)m)$ ↪ not great ...

Elementary parallel string matching

Subproblems in string matching:

- ▶ string matching = check all guesses $i = 0, \dots, n - m - 1$
- ▶ checking one guess is a subtask!

Approach 1:

- ▶ Check all guesses in parallel
- ↪ Time: $\Theta(m)$
- ↪ Work: $\Theta((n - m)m)$ ↪ not great ...

Approach 2:

- ▶ Divide T into **overlapping** blocks of $2m$ characters:
 $T[0..2m), T[m..3m), T[2m..4m), T[3m..5m) \dots$
- ▶ Find matches inside blocks in parallel, using efficient sequential method
↪ $\Theta(2m + m) = \Theta(m)$ each
- ↪ Time: $\Theta(m)$ Work: $\Theta(\frac{n}{m} \cdot m) = \Theta(n)$

Clicker Question



Is the string-matching problem “embarrassingly parallel”?

- A** Yes
- B** No
- C** Only for $n \gg m$
- D** Only for $n \approx m$

sli.do/comp526

Click on “Polls” tab

Clicker Question



Is the string-matching problem “embarrassingly parallel”?

- A** ~~Yes~~
- B** ~~No~~
- C** Only for $n \gg m$ ✓
- D** ~~Only for $n \approx m$~~

sli.do/comp526

Click on “Polls” tab

Elementary parallel matching – Discussion

- 👍 very simple methods
- 👍 could even run distributed with access to part of T
- 👎 parallel speedup only for $m \ll n$

Goal:

- ▶ methods with better parallel time! \rightsquigarrow higher speedup
- \rightsquigarrow must genuinely parallelize the matching process! (and the preprocessing of the pattern)
- \rightsquigarrow need new ideas

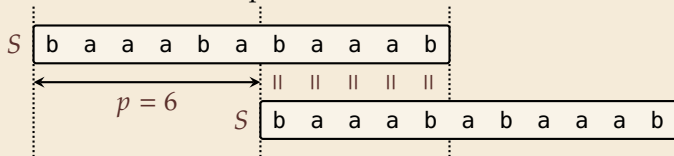
5.2 Periodicity

Periodicity of Strings

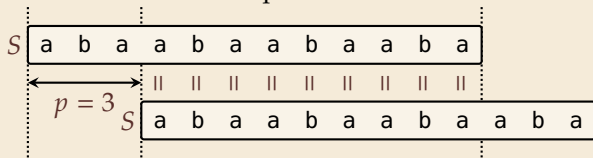
- ▶ $S = S[0..n - 1]$ has *period* p iff $\forall i \in [0..n - p) : S[i] = S[i + p]$
- ▶ $p = 0$ and any $p \geq n$ are trivial periods but these are not very interesting ...

Examples:

- ▶ $S = \text{baaababaaab}$ has period 6:



- ▶ $S = \text{abaabaabaaba}$ has period 3:



Periodicity and KMP

Lemma 5.1 (Periodicity = Longest Overlap)

$p \in [1..n]$ is the *shortest* period in $S = S[0..n - 1]$

iff $S[0..n - p]$ is the longest prefix that is also a suffix of $S[p..n)$.



Periodicity and KMP

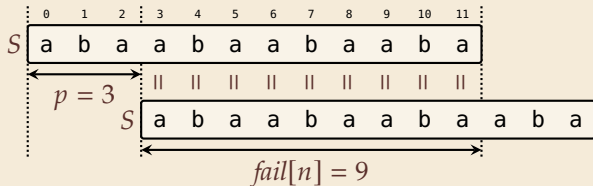
Lemma 5.1 (Periodicity = Longest Overlap)

$p \in [1..n]$ is the *shortest* period in $S = S[0..n-1]$

iff $S[0..n-p]$ is the longest prefix that is also a suffix of $S[p..n)$.



$S[0..n-1]$ has minimal period $p \iff fail[n] = n - p$



Periodicity Lemma

Lemma 5.2 (Periodicity Lemma)

If string $S = S[0..n - 1]$ has periods p and q with $p + q \leq n$, then it has also period $\text{gcd}(p, q)$.

 greatest common divisor

Proof: see tutorials; hint: recall Euclid's algorithm

Periodic strings

► What does the smallest period p tell us about a string $S[0..n-1]$?

► Two distinct regimes:

1. S is *periodic*: $p \leq \frac{n}{2}$

More precisely: S is totally determined by a string $F = F[0..p-1] = S[0..p-1]$
 S keeps repeating F until n characters are filled

↪ S is highly repetitive!

2. S is *aperiodic* (also *non-periodic*): $p > \frac{n}{2}$

S **cannot** be written as $S = F^k \cdot Y$ with $k \geq 2$ and Y a prefix of F

Clicker Question



Is $S = \text{aaaaaaaaaab}$ a periodic string?

A Yes

B No

sli.do/comp526

Click on "Polls" tab

Clicker Question



Is $S = \text{aaaaaaaaaab}$ a periodic string?

A ~~Yes~~

B No ✓

↪ “looking repetitive” is not enough for periodic!

sli.do/comp526

Click on “Polls” tab

5.3 String Matching by Duels

Periods and Matching

Witnesses for non-periodicity:

► Assume, $P[0..m - 1]$ does **not** have period p

$\rightsquigarrow \exists$ *witness against periodicity*: position $\omega \in [0..m - p) : P[\omega] \neq P[\omega + p]$

Periods and Matching

Witnesses for non-periodicity:

▶ Assume, $P[0..m-1]$ does **not** have period p

$\rightsquigarrow \exists$ *witness against periodicity*: position $\omega \in [0..m-p) : P[\omega] \neq P[\omega+p]$

Dueling via witnesses:

▶ If $P[0..m-1]$ does **not** have period p , then
at most one of positions i and $i+p$ can be (the starting position of) an occurrence of P .

Proof: Cannot have $T[(i+p)+\omega] = P[\omega] \neq P[\omega+p] = T[i+(\omega+p)]$.

Periods and Matching

Witnesses for non-periodicity:

- ▶ Assume, $P[0..m-1]$ does **not** have period p

$\rightsquigarrow \exists$ *witness against periodicity*: position $\omega \in [0..m-p) : P[\omega] \neq P[\omega+p]$

Dueling via witnesses:

- ▶ If $P[0..m-1]$ does **not** have period p , then
at most one of positions i and $i+p$ can be (the starting position of) an occurrence of P .

Proof: Cannot have $T[(i+p)+\omega] = P[\omega] \neq P[\omega+p] = T[i+(\omega+p)]$.

- ▶ **Duel** between guess i and $i+p$:
compare text character overlapped with witness ω



String Matching by Duels – Sequential

Assume that pattern P is *aperiodic*.

(can deal with periodic case separately; details omitted)

Algorithm:

1. Set $\mu := \lfloor \frac{m}{2} \rfloor$
2. Compute witnesses $\omega[1..\mu]$ against periodicity for all $p \leq \frac{m}{2}$.
3. For each block of μ consecutive indices $[0..\mu), [\mu..2\mu), [2\mu..3\mu), \dots$
run $\mu - 1$ duels to eliminate all but one guesses in the block
4. check remaining $\lceil \frac{n}{\mu} \rceil = O(n/m)$ guesses naively

String Matching by Duels – Sequential

Assume that pattern P is *aperiodic*.

(can deal with periodic case separately; details omitted)

Algorithm:

1. Set $\mu := \lfloor \frac{m}{2} \rfloor$
2. Compute witnesses $\omega[1..\mu]$ against periodicity for all $p \leq \frac{m}{2}$.
3. For each block of μ consecutive indices $[0..\mu), [\mu..2\mu), [2\mu..3\mu), \dots$ run $\mu - 1$ duels to eliminate all but one guesses in the block
4. check remaining $\lceil \frac{n}{\mu} \rceil = O(n/m)$ guesses naively

\rightsquigarrow another worst-case $O(n + m)$ string matching method!

Analysis:

1. $O(1)$
2. $O(m) \rightsquigarrow$ later
3. $O(\frac{n}{m})$ blocks
 $O(m)$ duels each
4. $O(\frac{n}{m})$,
 $\leq m$ cmps each

String Matching by Duels – Parallel

Assume that pattern P is *aperiodic*.

(can deal with periodic case separately; details omitted)

Algorithm:

1. Set $\mu := \lfloor \frac{m}{2} \rfloor$
2. Compute witnesses $\omega[1..\mu]$ against periodicity for all $p \leq \frac{m}{2}$.
3. For each block of μ consecutive indices $[0..\mu), [\mu..2\mu), [2\mu..3\mu), \dots$
run $\mu - 1$ duels to eliminate all but one guesses in the block
4. check remaining $\lceil \frac{n}{\mu} \rceil = O(n/m)$ guesses naively

String Matching by Duels – Parallel

Assume that pattern P is *aperiodic*.

(can deal with periodic case separately; details omitted)

Algorithm:

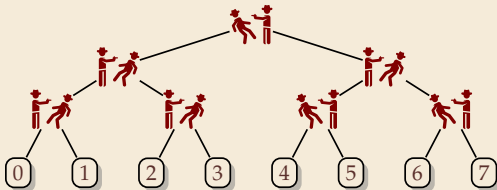
1. Set $\mu := \lfloor \frac{m}{2} \rfloor$
2. Compute witnesses $\omega[1..\mu]$ against periodicity for all $p \leq \frac{m}{2}$.
3. For each block of μ consecutive indices $[0..\mu), [\mu..2\mu), [2\mu..3\mu), \dots$ run $\mu - 1$ duels to eliminate all but one guesses in the block
4. check remaining $\lceil \frac{n}{\mu} \rceil = O(n/m)$ guesses naively

How to parallelize:

1. —
2. $O(\log^2(m)) \rightsquigarrow$ later
3. blocks in parallel (indep.), tournament of $\lceil \lg \mu \rceil$ rounds
4. check in parallel
collect result (like prefix sum)

Tournament of duels:

- ▶ each duel eliminates one guess
- \rightsquigarrow declare other guess *winner*
- ▶ conceptually like prefix sum!



String Matching by Duels – Parallel

Assume that pattern P is *aperiodic*.

(can deal with periodic case separately; details omitted)

Algorithm:

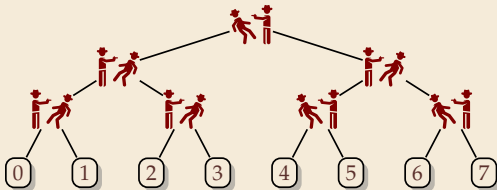
1. Set $\mu := \lfloor \frac{m}{2} \rfloor$
2. Compute witnesses $\omega[1..\mu]$ against periodicity for all $p \leq \frac{m}{2}$.
3. For each block of μ consecutive indices $[0..\mu), [\mu..2\mu), [2\mu..3\mu), \dots$ run $\mu - 1$ duels to eliminate all but one guesses in the block
4. check remaining $\lceil \frac{n}{\mu} \rceil = O(n/m)$ guesses naively

How to parallelize:

1. —
2. $O(\log^2(m)) \rightsquigarrow$ later
3. blocks in parallel (indep.), tournament of $\lceil \lg \mu \rceil$ rounds
4. check in parallel
collect result (like prefix sum)

Tournament of duels:

- ▶ each duel eliminates one guess
- \rightsquigarrow declare other guess *winner*
- ▶ conceptually like prefix sum!



\rightsquigarrow Matching part can be done in $O(\log m)$ parallel time and $O(n)$ work!

Computing witnesses

It remains to find the witnesses $\omega[1..\mu]$.

sequentially:

- ▶ an elementary procedure is similar in spirit to KMP failure array
- ▶ can be computed in $\Theta(m)$ time

parallel:

- ▶ much more complicated \rightsquigarrow beyond scope of the module
 - ▶ first $O(\log^2(m))$ time on CREW-RAM
 - ▶ later $O(\log m)$ time and $O(m)$ work using pseudoperiod method

Parallel Matching – State of the art

- ▶ $O(\log m)$ time & work-efficient parallel string matching
 - ▶ this is optimal for CREW-PRAM
- ▶ on CRCW-PRAM: matching part even in $O(1)$ time(!)
but preprocessing requires $\Theta(\log \log m)$ time