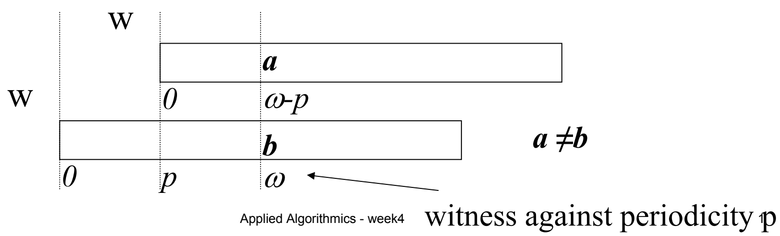
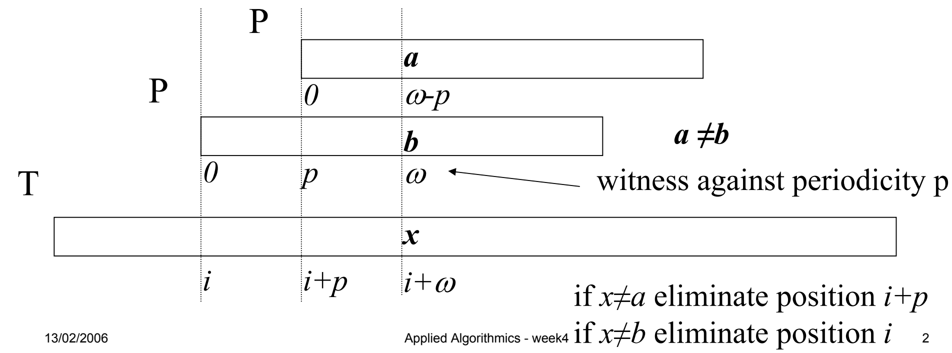
Non-periodicity and witnesses

- Periodicity - continued
  - If string $w=w[0..n-1]$ has periodicity p if $w[i]=w[i+p]$, for $i=0,\dots, n-p-1$
  - But what happens if w does not have a period p?
  - We say that there is a *witness* against periodicity p, and it is defined as an arbitrary position $p \le \omega \le n$, s.t., $w[\omega] \neq w[\omega-p]$



Dueling via witnesses

- If pattern P has no period p, then comparing any two positions i and $i+p$ in text T we can eliminate at least one of them as an occurrence of P



Pattern matching via duels

- Assume that pattern P is *non-periodic*, i.e., the shortest period of P is longer than $m/2$ (we will deal with periodic case later)
- Split the whole text T into consecutive segments S_i, for $i=0..2n/m$, of size $m/2$ each.
- In each segment S_i eliminate all (but at most one) occurrences using $< m/2$ duels
- Test the remaining $2n/m$ occurrences test naively
- The total cost of dueling and the final test is $O(n)$

Pattern matching via dules

- The search stage is preceded by pattern preprocessing when the witness is computed for every non-period in pattern P
- The witnesses can be computed on the basis of KMP failure function in linear $O(m)$ time
- ***Theorem:*** The pattern matching via duels is performed in time $O(n+m)$ and memory $O(m)$

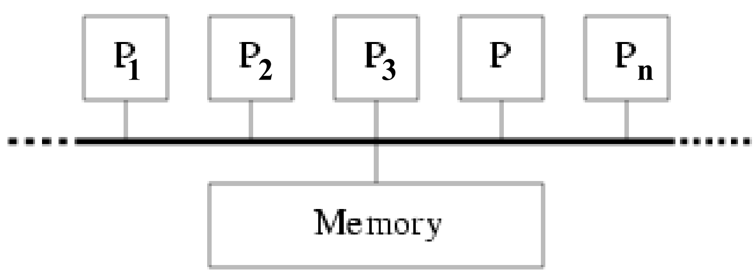
Pattern matching via duels

- One of the greatest advantages of pattern matching by duels is that the elimination of pattern occurrences can be done in many segments of the text parallel
- And indeed, the idea of duels (extended by deterministic sampling method) provides tools for optimal pattern matching in 2d-meshes, PRAMs, and hyper-cubes.

Parallel Random Access Machine

- *Parallel Random Access Machine* (PRAM) is a system of enumerated (uniform) processors that communicate with each other and perform various algorithmic tasks with a help of shared memory
- In simple words PRAM is a regular RAM (simplified model of standard PC) in which instead of one we have a number of processors with the same computational power.

Parallel Random Access Machine



- There are several sub-models of PRAM that differentiate according to the memory access protocols

PRAM sub-models

- *EREW* - exclusive read / exclusive write
  - Two (or more) processors can never read from or write to the same memory cell simultaneously
- *CREW* - concurrent read / exclusive write
  - Processors can read from though cannot write to the same memory cell simultaneously
- *CRCW* - concurrent read / concurrent write
  - Processors can both read from and write to the same memory cell simultaneously
  - It is often assumed that in case of concurrent write an arbitrary value is written into the memory cell

PRAM complexity measures

- The following measures are used:
  - *Time complexity*
  - *Space complexity*
  - *Work* which is defined as multiplication of the time and the number of processors used to solve the problem
- In case of PRAM we are mostly interested in the design of parallel algorithms whose time complexity is bounded by $O(\log^c n)$ and work is comparable with the time complexity of the most efficient sequential algorithms

Parallel pattern matching in time O(log m)

Reduce the number of occurrences to 2n/m

for $j=1$ **to** $\log m -1$ **do**

  for $i=1$ **to** $n/2^j$ **in-parallel do**

    Processor P_i reduces (using one duel) the number occurrences in segment $P[(i-1)2^j..i2^j-1]$ to one

Test naively all remaining pattern occurrences

for $i=1$ **to** n **in-parallel do**

  processors with index $i = k \pmod{m/2}$ test naively single remaining occurrence in segment $k\cdot m/2..(k+1)\cdot(m/2) -1$

Parallel pattern matching

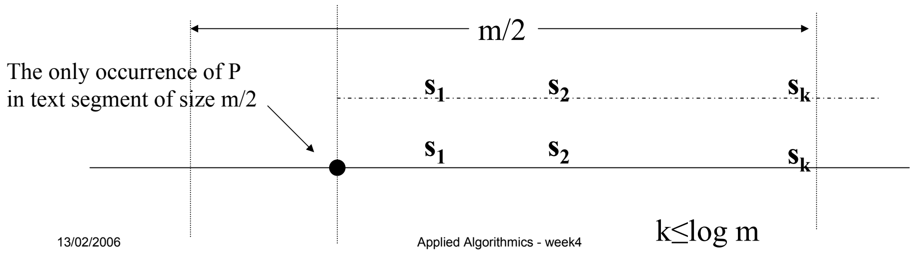
- The algorithm uses:
  - $O(\log m)$ rounds (time)
  - $O(n)$ processors and $O(n+m)$ memory
  - $O(n\cdot\log m)$ work (*which is non-linear! I.e, non-optimal*)
- The work can be reduced to $O(n)$
  - In each consecutive text segment of length $\log m$ use one processor to reduce the number of occurrences to one
  - Later apply non-optimal algorithm on at most $n/\log m$ remaining pattern occurrences

Pattern preprocessing

- One can prove that pattern preprocessing (computation of witnesses) in all PRAM sub-models can be done in optimal time $O(\log m)$ and work $O(m)$.
- The preprocessing is based on non-trivial *pseudo-period method.*
- ***Theorem:*** Parallel pattern matching can be done in optimal time $O(\log m)$ and work $O(n+m)$.

Parallel pattern matching in O(1) time

- In the most powerful PRAM sub-model CRCW the search stage can be performed in *O(1)* time
- The search is based on the notion of *deterministic sample,* which is a small collection of witnesses



Parallel pattern matching in O(1) time

- The search stage on CRCW PRAM can be done in *O(1)* time on *O(n·log m)* processors as follows
- 1) Each position in text *T* is tested for the occurrence of deterministic sample using *O(log m)* processors
- 2) In each segment of size *m/2* use *m/2* processors to mark the *first* and the *last* occurrence of the deterministic sample (all other occurrences can be ignored)
- 3) Test marked occurrences naively using *O(n)* processors
- All 3 steps can be performed in *O(1)* time

Parallel pattern matching in O(1) time

- The search stage in pattern matching based on deterministic sample can be tuned to work in optimal time *O(1)* and work *O(n)*.
- The search stage is preceded by pattern preprocessing in optimal time *O(loglog m)* and work *O(m)*.
- There exists *O(1)* time parallel pattern matching algorithm with *O(1)* expected preprocessing time

Pattern matching in small extra space

- Note that one can use the deterministic sample to implement sequential pattern matching that works in *O(n)* time and *O(log m)* extra space.
- In fact, one can reduce the extra space to *O(1)* and still keep the linear *O(n)* pattern search time.

Pattern matching with "don't care" symbols

- In some applications we are interested in finding patterns that contain special symbols that match any symbol in the alphabet
- We call these special character *"don't care"* symbols
- The methods designed for exact string matching do not work efficiently if the number of "don't care" symbols in the pattern is large
- Efficient solution to pattern matching with don't care symbols is based on fast computation of *convolutions*

Pattern matching with "don't care" symbols

- We assume that the strings are built over binary $\{0,1\}$ extended by a "don't care" symbol $*$
- For a larger alphabet A of cardinality n we can encode each symbol by exactly $\log n$ bits
- For any pattern $P \in \{0,1,*\}^*$ we define two strings:
  - P_0 which is P in which all occurrences of $*$s are replaced by 0s, and
  - P_1 which is P in which all occurrences of $*$s are replaced by 1s and then all bits are negated

Example of P, P_1 and P_2

- Let pattern $P=011*01011**1*0$, reversed P is
- $P_0=01100101100100$,
- Also after replacing $*$s with 1s we get a string
- 01110101111110, and negating all bits we have
- $P_1=10001010000001$.

Pattern matching with "don't care" symbols

- The search for pattern P in text T is replaced by:
  1) the search for P_0 in text T, and
  2) the search for P_1 in text T with all bits negated
- In both cases for each text position we count the number of recognized 1s (in case 2 the number of 1s corresponds to the number of 0s in P)
- If at any text position the number of matched 1s and 0s is the same as the number of 1s and 0s in P the pattern occurrence is found.

Pattern matching with "don't care" symbols

Let $P=011{*}01011{*}{*}1{*}0$, and

Text $T=0101010111100101110101110110101110$

Then $P_1=01100101100100$, $P_2=10001010000001$.

$P_1=01100101100100$

$T=0101010111100101110101110110101110$

$\sim T=1010101000011010001010001001010001$

$P_2=10001010000001$

All 1s are matched

The Fast Fourier Transform

- But how can we match efficiently all 1s in patterns P_1 and P_0?
- Rather surprisingly we can translate the matching of all 1s into the *multiplication* of *large integers* and *polynomials*
- *The Fast Fourier Transform* is a surprisingly efficient procedure for multiplying such objects

The Fast Fourier Transform

- A polynomial represented in a coefficient form is described by a coefficient vector $\boldsymbol{a}=[a_0, a_1, \ldots, a_{n-1}]$ as follows:

$$p(x)=\sum_{i=0}^{n-1} a_i x^i$$

- The degree of such a polynomial is the largest index of non-zero coefficient a_i
- A coefficient vector of length n can represent polynomials of degree $n-1$

Multiplication of Polynomials

- Fast multiplication of two polynomials $p(x)\cdot q(x)$ as defined in coefficient form, is seen as follows
- Consider $p(x)\cdot q(x)$, where:
  - $p(x)=\sum_{i=0}^{n-1} a_i\cdot x^i$ and $q(x)=\sum_{i=0}^{m-1} b_i\cdot x^i$
- Then $p(x)\cdot q(x)=$
  - $\sum_{i=0}^{n+m-2} c_i\cdot x^i$, where $c_i=\sum_{j=i-m+1}^{i} a_j\cdot b_{i-j}$, for $i=m-1,\ldots,n+m-2$
- The coefficients c_i for other values are based on shorter summations, and we have no interest in them

Convolutions and FFT

- The equation defines a vector $\boldsymbol{c} = [c_0, c_1, \ldots, c_{n-1}]$, which we is the *convolution* of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
- If we apply the definition of convolutions directly, then it will take us time $\Theta(nm)$ to multiply the two polynomials $p(x)$ and $q(x)$
- The *Fast Fourier Transform (FFT)* algorithm allows us to perform this multiplication in time $O(n \log m)$.

Convolutions and PM with don’t cares

- So what the convolutions have to do with pattern matching with “don’t care” symbols?
- In fact convolutions help a lot. One can interpret patterns P_1^R and P_0^R as well as text(s) T and $\sim T$ as binary coefficients of polynomials of degrees $m-1$ and $n-1$ respectively.
- And in particular we are interested in convolutions in polynomials $P_1^R(x) \cdot T(x)$ and $P_0^R(x) \cdot \sim T(x)$

Convolutions and PM with don’t cares

- Since convolutions of polynomials of degree n and m can be computed in time $O(n \cdot \log m)$ we can compute convolutions for $P_1^R(x) \cdot T(x)$ and $P_0^R(x) \cdot \sim T(x)$ in time $O(n \cdot \log m)$.
- The values of convolutions correspond to the number of matched 1s and 0s at consecutive positions in text T
- ***Theorem:*** The pattern matching with don’t care symbols can be solved in time $O(n \cdot \log m)$.

Convolutions and PM with don’t cares

- Convolutions forming a coefficient at terms with power $i = m-1, \ldots, n+m-2$ correspond to number of matched ones