KMP PARALLEL ALGORITHM FOR PATTERN MATCHING

UBIT: rbammmidi
Professor: Dr. Russ Miller
Need of pattern matching

- Pattern matching is used to determine whether source files of high-level languages are syntactically correct.
- Many fingerprint recognition methods are in use to perform fingerprint matching out of which pattern matching approaches is widely used.
- Pattern matching enables users to find the locations of particular DNA subsequences in a database or DNA sequence.
- Searching for word in the large log files dump
- Validating the information received from the client before writing into DB.
Pattern Matching (Naive)

Text: A A A A A A A A A A A A A A A A A A B

Pattern: A A A B
Pattern Matching (Naive)

Text: A A A A A A A A A A A A A A B

Pattern: A A A B

$n = \text{Length(Text)}$

$m = \text{length(Pattern)}$

$T(n,m) = O(n*m)$

BruteForceStringMatch(S, P):

for $i = 0$ to $n-m$ do

$j = 0$

while $j < m$ and $S[i+j] == P[j]$ do

$j = j + 1$

if $j == m$ then

return $i$

return -1
Knuth-Morris-Pratt (KMP)

The Knuth-Morris-Pratt (KMP) algorithm is an algorithm that is used to search for a pattern in a given text in $O(m + n)$ time (where $m$ and $n$ are the lengths of pattern and text).
KMP Algorithm ~ 2 Step Process

Step 1: Pre-process the Pattern

  step 1.1: Build the LPS table from the pattern

Step 2: Iterate through the Text and Pattern and check for the existence of the pattern in the text
Components and Terminology of KMP Algorithm

In the KMP algorithm, we have two terms, proper prefix and suffix.

A proper prefix of the pattern will be a subset of the pattern using only the beginning portion (the first index), or the first few indices of the pattern except the last character.

**Pattern**: a b c d a b c

- a
- a b
- a b c
- a b c d
- a b c d a b c
Components and Terminology of KMP Algorithm

A proper suffix of any pattern would be a subset of the pattern with elements taken only from the right end of the pattern as in, any number of elements, starting from the last character. Taking the first character of the string is not allowed.

Pattern : a b c d a b c

- c
- b c
- a b c
- d a b c
Longest prefix that is also a suffix (LPS)

LPS[] represents longest prefix that is also a suffix till i

Takes O(m) time to generate the LPS array
KMP Pattern Matching


Pattern: A A B A A A A B A A B

0 1 0 1 2 2 2 3 4 5 3
KMP Pattern Matching


Pattern: A A B A A A A B A A

0 1 0 1 2 2 2 3 4 5 3
KMP Pattern Matching

Text: A A A B A A A C A B B A A A A B A A A B A A A B A A A A A A A A A A A A

Pattern: A B A A A B A A B 0 1 0 1 2 2 2 3 4 5 3
KMP Pattern Matching

Text: AAAABAAAACCABBBAAAABBBAAAABAA

Pattern: AABAAABABA
Parallel KMP Algorithm
Components and Terminology of Parallel KMP Algorithm

• KMP Table: \( \text{kmpTable}[i] \) stores the length of the longest pattern that matches with the text till index \( i \)
• Cumulative KMP Table: It holds the cumulative KMP table information from the processor 0 to processor \( i \)
• Non-cumulative KMP Table: It holds the non-cumulative kmp table information, which means it doesn’t contain the KMP information from processor 0 (partial kmp table)
KMP Table usage

Text: D A A B B C A D

kmpTable

1 2 3 4

PE 0
D A A B

Pattern: D A A B B C A D

PE 1
BCAD

Pattern: D A A B B C A D

0 0 0 0
Initial Attempt

Drawbacks:
Every $i^{th}$ processor has to wait until it receives the parallel KMP table from $[0, i - 1]$ processors.
Parallel KMP Steps

- Split the given text equally of size \( \frac{n}{k} \) each to all the processors — Broadcasting
- Each processor executes sequential KMP independently on the given text & pattern
- Every processor checks if the cumulative KMP table is available to receive from its predecessor
- If the cumulative KMP table is not available in the buffer, it receives the non-cumulative KMP table from its preceding processor.
- This process continues till it finds the pattern in the given text.
Broadcasting
Parallel KMP Visualization

Step 1: Process Text & Pattern
Step 2: Async send
Step 3: Async recv
Profiling a typical processor

Recv non-cumulative kmp table from its predecessor

Send non-cumulative kmp table to its successor

Send cumulative kmp table to its successor

Recv non-cumulative kmp table from its predecessor
Pattern existence in a single processor

Pattern: D A A B
Pattern existence in two processors

Pattern: D A A B B C A D
Pattern existence in three processors

Pattern: A C A A D A A B B C A D
Pattern existence in four processors

Pattern: C A C A A D A A B B C A D
Input Size Vs Time In Secs for Sequential KMP

<table>
<thead>
<tr>
<th>Input Size</th>
<th>Time In Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E3</td>
<td>1.047</td>
</tr>
<tr>
<td>1E4</td>
<td>1.259</td>
</tr>
<tr>
<td>1E5</td>
<td>3.581</td>
</tr>
<tr>
<td>1E6</td>
<td>27.454</td>
</tr>
<tr>
<td>1E12</td>
<td>48.4</td>
</tr>
</tbody>
</table>
## Processors Vs Run Time Text Size=1e12 & P=33 (slurm)

### 1 Processor per node

<table>
<thead>
<tr>
<th>Processors</th>
<th>Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>41.1</td>
</tr>
<tr>
<td>4</td>
<td>25.8</td>
</tr>
<tr>
<td>8</td>
<td>42.3</td>
</tr>
<tr>
<td>16</td>
<td>44.9</td>
</tr>
<tr>
<td>32</td>
<td>55.1</td>
</tr>
<tr>
<td>64</td>
<td>172.7</td>
</tr>
<tr>
<td>128</td>
<td>190.0</td>
</tr>
</tbody>
</table>

![Graph showing the relationship between processors and run time.](image-url)
### TCP/IP Vs IB|OPA Network Band Performance

<table>
<thead>
<tr>
<th>Processor</th>
<th>IB</th>
<th>TCP/IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>41.1</td>
<td>271.604</td>
</tr>
<tr>
<td>4</td>
<td>25.8</td>
<td>277.253</td>
</tr>
<tr>
<td>8</td>
<td>42.3</td>
<td>277.377</td>
</tr>
<tr>
<td>16</td>
<td>44.9</td>
<td>287.313</td>
</tr>
<tr>
<td>32</td>
<td>55.1</td>
<td>302.991</td>
</tr>
<tr>
<td>64</td>
<td>172.7</td>
<td>349.8</td>
</tr>
<tr>
<td>128</td>
<td>190</td>
<td>719.4</td>
</tr>
</tbody>
</table>
## Scale (Constant Data/PE)

<table>
<thead>
<tr>
<th>Processors</th>
<th>Time in secs</th>
<th>Data Per PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30.821</td>
<td>1e4</td>
</tr>
<tr>
<td>2</td>
<td>28.999</td>
<td>1e4</td>
</tr>
<tr>
<td>4</td>
<td>37.272</td>
<td>1e4</td>
</tr>
<tr>
<td>8</td>
<td>46.078</td>
<td>1e4</td>
</tr>
<tr>
<td>16</td>
<td>55.606</td>
<td>1e4</td>
</tr>
<tr>
<td>32</td>
<td>50.129</td>
<td>1e4</td>
</tr>
<tr>
<td>64</td>
<td>92.168</td>
<td>1e4</td>
</tr>
<tr>
<td>128</td>
<td>289.702</td>
<td>1e4</td>
</tr>
</tbody>
</table>

**TextSize ~ 1e4 & PatternSize ~ 100**
Speed Up Vs Processors

Input text size $1e12$, $T_{seq} = 48.478$ secs

$SpeedUp = \frac{T_{seq}}{T_{parallel}}$

1 Processor per node

<table>
<thead>
<tr>
<th>Processors</th>
<th>$T_{parallel}$</th>
<th>Speed Up</th>
<th>Data per PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$1e12$</td>
</tr>
<tr>
<td>2</td>
<td>41.1</td>
<td>1.17</td>
<td>5000000000000</td>
</tr>
<tr>
<td>4</td>
<td>25.8</td>
<td>1.87</td>
<td>2500000000000</td>
</tr>
<tr>
<td>8</td>
<td>42.3</td>
<td>1.14</td>
<td>1250000000000</td>
</tr>
<tr>
<td>16</td>
<td>44.9</td>
<td>1.07</td>
<td>625000000000</td>
</tr>
<tr>
<td>32</td>
<td>55.1</td>
<td>0.87</td>
<td>312500000000</td>
</tr>
<tr>
<td>64</td>
<td>172.7</td>
<td>0.28</td>
<td>1562500000000</td>
</tr>
<tr>
<td>128</td>
<td>190</td>
<td>0.25</td>
<td>78125000000</td>
</tr>
</tbody>
</table>
References

• https://ieeexplore.ieee.org/document/6618720
• https://cmps-people.ok.ubc.ca/ylucet/DS/KnuthMorrisPratt.html
• https://buffalo.app.box.com/s/nqj3neyt2w1dtdb3gix6zxq5gcc9x30n
• http://koreascience.or.kr/article/JAKO201814955686557.page
• https://ieeexplore.ieee.org/document/8599534
Thank You

Questions?