Parallel Knuth Morris Pratt String Matching
CSE 633- Fall 2011
Outline

- String Matching
  - Knuth-Morris-Pratt Algorithm
  - How to improve with parallelization
- Experimental Set up
  - Data Description
  - Hardware Specifications
- Results
- Analysis
- Future Work
Goal: Find occurrences of a pattern $P$ of length $m$ in a string $S$ of length $n$ ($m < n$)

Applications:
- Text Processing
- DNA & Protein sequence matching
- Anti-Virus & Intrusion Detection
- Database Query
Knuth-Morris-Pratt

The prefix function $\pi[q]$

- Encapsulates how the pattern matches against shifts of itself
- When there is a mismatch, the prefix function tells you how far to shift the pattern

Mismatch: $P[q+1] \neq S[i] \rightarrow q = \pi[q]$

<table>
<thead>
<tr>
<th>$q$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P[q]$</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>$\Pi[q]$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
In Parallel

- Open MP on Shared Memory Machine
- Matching multiple patterns on same text
- If \( p = \) number of patterns, \( c = \) number of cores, \( S = \) input string
- Find match faster by:
  - Distribute \( p/c \) patterns to each core
  - Distribute \( S \) to each core
  - Projected speed up approximately \( c \) times
Master
Reads S from text
And sends to each thread

Run KMPMatch on S with $P_{T_H_id}$
Record Results in Struct

Run KMPMatch on S with $P_{T_H_id}$
Record Results in Struct

Run KMPMatch on S with $P_{T_H_id}$
Record Results in Struct

Master can analyze results from Structs
Experimental Set up - Data

**Input Text**
- 100,000 lines each 256 characters
- Random 1 and 0

**Pattern**
- Each permutation in order (ie binary counting)
- 4 ‘bit’ through 13 ‘bit’
  - 1, 2, and 3 ‘bit’ seemed too trivial
- Initial trials were fixed on the 6 ‘bit’ pattern
- Time is average over 3 trials
Experimental Set up - Hardware

- 12 Core Machine
  - 2.4 GHz
  - 48GB Memory

- 32 Core Machine
  - 2.13GHz – INTEL
  - 2.2GHz – AMD
  - 256GB Memory

- Initial tests varied threads from 1-64
Initial Results

- **RAM Time 32 Cores**: 16.176 seconds
- **Best Time 32 Cores**: 13 threads – 3.575 seconds
  - Deviation from 8-21 threads is 0.1575 seconds
- **RAM Time 12 Cores**: 14.626 seconds
- **Best Time 12 Cores**: 12 threads – 2.384 seconds
Overall Results for 6 Character Pattern
1 through 64 threads

![Graph showing the time in seconds vs. number of threads for 32 Core and 12 Core systems. The graph illustrates that the 32 Core system consistently has a lower time across all thread numbers compared to the 12 Core system.](image)
Speedup

Speedup On 32 Core Node

Best Speedup 4.52x

Speedup on 12 Core Node

Best Speedup 6.13x
How problem Size effects time

![Graph showing the relationship between number of patterns and time in seconds for different RAM and thread configurations.](image-url)
Summary of Results

- Parallel good until number of threads exceeds number of cores
- 14 ‘bit’, 16k patterns would not run on 12 core machine
- Max speedup on 12 core machine 6.13x not 12x
- Max speedup on 32 core machine even worse 4.52x not 32x!
Future Work

- Only compute prefix function one time for a pattern
- Write results to a file rather than into a struct
  - Memory issues with size of struct per pattern limited the length of pattern
  - Writing to a file solves this issue
- Examine load balancing directives