On Compile Time Knuth-Morris-Pratt Precomputation

Justin Kourie\textsuperscript{1}  Bruce Watson\textsuperscript{2,1}  Loek Cleophas\textsuperscript{3,1}

\textsuperscript{1} FASTAR Research Group, Department of Computer Science, University of Pretoria, 0002 Pretoria, Republic of South Africa (justin@fastar.org)

\textsuperscript{2} FASTAR Research Group, Centre for Knowledge Dynamics and Decision-making, Stellenbosch University, Private Bag X1, 7602 Matieland, Republic of South Africa (bruce@fastar.org)

\textsuperscript{3} Software Engineering & Technology Group, Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands (loek@fastar.org)

Prague Stringology Conference 2011
Outline

1. The General Idea
1 The General Idea

2 Experiment
   - Benchmark Requirements
   - Implementation Experience

Justin Kourie, Bruce Watson, Loek Cleophas
On Compile Time Knuth-Morris-Pratt Precomputation
Taxonomological Optimization Targets

Taxonomological Optimization Targets

Taxonomological Optimization Targets

Taxonomological Optimization Targets[?]

Keywords Known at Compile Time
Perform precomputation at \textit{compile time} (metaprogramming)
Perform precomputation at *compile time* (metaprogramming)
Runtime search will receive performance boost
Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked
Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

- *When* will this be useful?
Keywords Known at Compile Time

- Perform precomputation at compile time (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

- When will this be useful?
- How can it be implemented?
Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

- *When* will this be useful?
- *How* can it be implemented?

Why KMP?
Perform precomputation at *compile time* (metaprogramming)
Runtime search will receive performance boost

**Questions Asked**
- *When* will this be useful?
- *How* can it be implemented?

**Why KMP?**
- Need a starting point to experiment with implementation techniques
Keywords Known at Compile Time

- Perform precomputation at compile time (metaprogramming)
- Runtime search will receive performance boost

Questions Asked
- When will this be useful?
- How can it be implemented?

Why KMP?
- Need a starting point to experiment with implementation techniques
- Start as simply as possible
Keywords Known at Compile Time

- Perform precomputation at *compile time* (metaprogramming)
- Runtime search will receive performance boost

Questions Asked

- *When* will this be useful?
- *How* can it be implemented?

Why KMP?

- Need a starting point to experiment with implementation techniques
- Start as *simply* as possible
- Primary aim was probatory research
Design Overview

- Compile time requirements
  - Generate a set of keywords ($K$)
    - For each keyword $k \in K$, precompute KMP fail index
- Run time requirements
  - Benchmark the following using some target text ($x$)
    - Traditional (non-optimised) KMP
    - Optimised KMP
      - Precomputation algorithm at runtime

Justin Kourie, Bruce Watson, Loek Cleophas

On Compile Time Knuth-Morris-Pratt Precomputation
Design Overview

Compile time requirements

- Generate a set of keywords \((K)\)
 Compile time requirements

- Generate a set of keywords ($K$)
- For each keyword $k \in K$, precompute KMP fail index
**Design Overview**

**Compile time requirements**
- Generate a set of keywords ($K$)
- For each keyword $k \in K$, precompute KMP fail index

**Run time requirements**
Design Overview

Compile time requirements
- Generate a set of keywords \((K)\)
- For each keyword \(k \in K\), precompute KMP fail index

Run time requirements
- Benchmark the following using some target text \((x)\)
  - Traditional (non optimised) KMP
  - Optimised KMP
  - Precomputation algorithm at runtime
A Pipelined Approach

NOTE: Complexity Analysis VS Complex Experiment

Decided to analyse non matching case only (i.e, optimized = \( O(n) \), traditional = \( O(n + m) \))

Generated \( K \) such that no \( k \in K \) present in \( x \)

Simple way to build large, consistent data to analyse

Allowed focus to be on implementation and analysis (not design paralysis)

Not ideal, can be improved

Justin Kourie, Bruce Watson, Loek Cleophas

On Compile Time Knuth-Morris-Pratt Precomputation
A Pipelined Approach

NOTE: Complexity Analysis VS Complex Experiment
A Pipelined Approach

NOTE: Complexity Analysis VS Complex Experiment

- Decided to analyse non matching case only (i.e, optimized = $O(n)$, traditional = $O(n + m)$)

Justin Kourie, Bruce Watson, Loek Cleophas
NOTE: Complexity Analysis VS Complex Experiment

- Decided to analyse non matching case only (i.e, optimized = $O(n)$, traditional = $O(n + m)$)
- Generated $K$ such that no $k \in K$ present in $x$
A Pipelined Approach

NOTE: Complexity Analysis VS Complex Experiment

- Decided to analyse non matching case only (i.e, optimized = $O(n)$, traditional = $O(n + m)$)
- Generated $K$ such that no $k \in K$ present in $x$
- *Simple* way to build large, consistent data to analyse
NOTE: Complexity Analysis VS Complex Experiment

- Decided to analyse non matching case only (i.e, optimized = $O(n)$, traditional = $O(n + m)$)
- Generated $K$ such that no $k \in K$ present in $x$
- *Simple* way to build large, consistent data to analyse
- Allowed focus to be on implementation and analysis (not design paralysis)
NOTE: Complexity Analysis VS Complex Experiment

- Decided to analyse non matching case only (i.e, optimized $= O(n)$, traditional $= O(n + m)$)
- Generated $K$ such that no $k \in K$ present in $x$
- *Simple* way to build large, consistent data to analyse
- Allowed focus to be on implementation and analysis (not design paralysis)
- Not ideal, can be improved
C++
Eventually Hit a Dead End

Severe Metaprogramming Constraints
Severe Metaprogramming Constraints

- Constrained string length
C++
Eventually Hit a Dead End

Severe Metaprogramming Constraints
- Constrained string length
- Very high computational overhead
Severe Metaprogramming Constraints

- Constrained string length
- Very high computational overhead
- 'Poor', 'wri', 'tabi', 'lity'

```cpp
typedef mpl::string<'hell','o wo','rld'> hello;
```
C++
Eventualy Hit a Dead End

Severe Metaprogramming Constraints

- Constrained string length
- Very high computational overhead
- 'Poor', 'wri', 'tabi', 'lity'
  ```cpp
typedef mpl::string<'hell', 'o wo', 'rld'> hello;
```
- Variadic compile time array initialisation
  ```cpp
  fail_idx[] = { precomp<k>::compute }
  // i.e., int foo = { 0, 0, 1, ... }
  ```
Designed for Metaprogramming

Addresses all problems encountered in C++ (length, computation, writability, array initialisation).
Addresses all problems encountered in C++ (length, computation, writability, array initialisation).

Why?

Designed for Metaprogramming
D

Designed for Metaprogramming

Addresses all problems encountered in C++ (length, computation, writability, array initialisation).

Why?

- Ground up design = more powerful metaprogramming constructs
Designed for Metaprogramming

Addresses all problems encountered in C++ (length, computation, writability, array initialisation).

Why?

- Ground up design = more powerful metaprogramming constructs
- Compile Time Function Evaluation
Hypotheses

In depth discussion of:

Justin Kourie, Bruce Watson, Loek Cleophas

On Compile Time Knuth-Morris-Pratt Precomputation
Hypotheses

In depth discussion of:

- Sanity checks
Hypotheses

In depth discussion of:

- Sanity checks
- Postulations about usefulness
Hypotheses

In depth discussion of:

- Sanity checks
- Postulations about usefulness
- No strict claims or generalisations
Hypotheses

In depth discussion of:

- Sanity checks
- Postulations about usefulness
- No strict claims or generalisations
- Interesting results to observe nonetheless
Hypotheses
The General Idea
Experiment
Analysis
Final Remarks

Theoretical Speculation
Results and Observations

C++ VS D
A Limited Comparison

C++ Benchmark |z| = 64

D Benchmark |z| = 64

Justin Kourie, Bruce Watson, Loek Cleophas

On Compile Time Knuth-Morris-Pratt Precomputation
Interpreting D Data
When can we justify Compile Time Optimizations?

Strong case for optimized search observed where $|k| \geq \approx \frac{|x|}{4}$
Optimized search gains neared redundancy where $|k| \leq \approx \frac{|x|}{50}$
Answers to the Original Questions
When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis).

Not strict generalisations nor surprising in themselves but...

How could this technique be implemented?

- Avoid C++ string metaprogramming pitfalls
- Choose a language designed for string metaprogramming
When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis)
When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis).
- Not strict generalisations nor surprising in themselves but...
When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis)
- Not strict generalisations nor surprising in themselves but...

...How could this technique be implemented?

- Avoid C++ string metaprogramming pitfalls
- Choose a language designed for string metaprogramming
- General idea of the benchmark's design
When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis).
- Not strict generalisations nor surprising in themselves but...

...How could this technique be implemented?

- Avoid C++ string metaprogramming pitfalls
When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis)
- Not strict generalisations nor surprising in themselves but...

...How could this technique be implemented?

- Avoid C++ string metaprogramming pitfalls
- Choose a language designed for string metaprogramming
### When could this technique be useful?

- Size of keyword is large relative to the size of text (e.g., realtime monitoring).
- Size of keyword is relatively small, gains tend towards redundancy (e.g., bulk analysis).
- Not strict generalisations nor surprising in themselves but...

### How could this technique be implemented?

- Avoid C++ string metaprogramming pitfalls
- Choose a language designed for string metaprogramming
- General idea of the benchmark’s design
Thanks and Acknowledgements
Thanks and Acknowledgements

...and of course questions! =/