Fast Regular Expression Matching with Bit-parallel Data Streams

Robert D. Cameron Kenneth S. Herdy Ben Hull
Thomas C. Shermer

School of Computing Science
Simon Fraser University

Abstract

A parallel regular expression matching pattern method is introduced and compared with existing approaches for efficient on-line search. The method is based on the concept of bit-parallel data streams, in which parallel streams of bits are formed such that each stream comprises bits in one-to-one correspondence with the character code units of a source data stream.

An implementation of the techniques in the form of a regular expression compiler is discussed. The compiler accepts a regular expression and forms unbounded bit-parallel data stream statements. Bit-parallel operations are then transformed into a low-level C-based implementation for compilation into native pattern matching application code. These low-level C-based implementations take advantage of the SIMD (single-instruction multiple-data) capabilities of commodity processors to yield a dramatic speed-up over traditional byte-at-a-time approaches. On processors supporting W-bit addition operations, the method processes W source characters in parallel and performs up to W finite state transitions per clock cycle. We further improve our methods through the introduce a new bit-parallel scanning primitive, *Match Star*. which performs a parallel Kleene closure operation over character classes and without backtracking.

We evaluate the performance of our method in comparison with several widely known *grep* implemenations, *Gnu grep*, *agrep*, *nr-grep*. Performance results are analyzed using the performance monitoring counters of commodity hardware. Overall, our results demonstrate dramatic speed-ups over other publically available software.

1 Introduction

Regular expresssion matching is an extensively studied problem with application to various problem domains, for example, text-processing and bioinformatics. A multitude of algorithms and software tools exist to address the specific demands of the various problem domains.

The pattern matching problem can be stated as follows. Given a text $T_{1..n}$ of n characters and a pattern P, find all the text positions of T that start an occurrence of P. Alternatively, one may want all the final positions of occur-

rences. Some applications require slightly different output such as the line that matches the pattern.

A pattern P can be a simple string, but it can also be a regular expression. A regular expression, is an expression that specifies a set of strings and is composed of (i) simple strings and (ii) the union, concatenation and Kleene closure of other regular expressions. To avoid parentheses it is assumed that the Kleene star has the highest priority, next concatenation and then alternation, however, most formalisms provides grouping operators to allow the definition of scope and operator precedence. Readers unfamiliar with the concept of regular expression matching are referred classical texts such as [2].

Regular expression matching is typically performed using a wide variety of publically available software tools designed for efficient on-line pattern matching. For example, UNIX grep, Gnu grep, agrep, cgrep, nr-grep, and Perl regular expressions [1]. Amongst these tools Gnu grep (egrep), agrep, and nr-grep are widely considered the fastest regular expression matching tools in practice [10].

Although traditional finite state machine methods used in the scanning and parsing of text streams is considered to be the hardest of the "13 dwarves" to parallelize [1], bit-parallel data stream technology shows considerable promise as a general approach to regular expression matching. In this approach, character streams are processed W positions at a time using the W-bit SIMD registers commonly found on commodity processors (e.g., 128-bit XMM registers on Intel/AMD chips). This is achieved by first slicing the byte streams into eight separate basis bitstreams, one for each bit position within the byte. These basis bitstreams are then combined with bitwise logic and shifting operations to compute further parallel bit streams of interest.

We further increase the performance of our approach with the introduction of a new parallel scanning primitive coined *Match Star*.

The remainder of this paper is organized as follows. Section 2 introduces the notations and basic concepts used throughout this paper. Section 3 presents background material on classical automata-based approaches to regular expression matching and describes the *grep* family utilities. Next, section 4 describes our parallel regular expression matching techniques. Section 5 presents our software toolchain for constructing pattern matching applications. Section 6 describes the evaluation framework and Section 7 presents a detailed performance analysis of our data parallel bitstream techniques in comparison to several software tools. Section 9 concludes the paper.

The fundamental contribution of this paper is fully general approach to regular expression matching using bit-parallel data stream operations. The algorithmic aspects of this paper build upon the fundamental concepts of our previous work [7, 5, 6, 9]. Specific contributions include:

- compilation of regular expressions into unbounded bit-parallel data stream equations;
- documentation of character classes compilation into bit-parallel character class data streams;
- Match Star parallel scanning primitive;
- efficient support for unicode characters.

2 Basic Concepts

In this section, we define the notation and basic concepts used throughout this paper.

2.1 Notation

We use the following notations. Let $P = p_1 p_2 \dots p_m$ be a pattern of length m and $T = t_1 t_2 \dots t_n$ be a text of length n both defined over a finite alphabet *sigma* of size *alpha*. The goal of simple pattern expression matching is to find all the text positions of T that follow an occurrence of P. P may be a simple pattern, extended pattern, or a regular expression.

We use C notations to represent bitwise operations $\neg$, $\vee$, $\wedge$, $\oplus$ represent bitwise NOT, OR, AND, XOR, respectively. Operators $\ll k$, and $\gg$ represent logical left shift, and logical right shift, respectively and are further modulated by the number of bit positions a given value shall be shifted by, for example "shift left by n". Vacant bit-positions are filled in with zeroes.

2.2 Regular Expressions

3 Background

3.1 Regular Expressions and Finite Automata

The origins of regular expression matching date back to automata theory and formal language theory developed by Kleene in the 1950s [8]. Thompson [13] is credited with the first construction to convert regular expressions to nondeterministic finite automata (NFA) for regular expression matching. Following Thompson's approach, a regular expression of length m is first converted to an NFA with O(m) nodes. It is possible to search a text of length n using the NFA directly in O(mn) worst case time. Often, a more efficient choice is to convert the NFA into a DFA. A DFA has only a single active state and allows to search the text at O(n) worst-case optimal. It is well known that the conversion of the NFA to the DFA may result in the state explosion problem. That is the resultant DFA may have $O(2^m)$ states.

Thompson's original work marked the beginning of a long line of regular expression pattern matching methods that process an input string, character-at-a-time, and that transition from state to state according to the current state and the next input character.

Whereas traditional automata techniques acheive O(n) worst-case optimal efficiency, simple string matching algorithms, such as the Boyer-Moore family of algorithms, skip input characters to achieve sublinear times in the average case [4].

Boyer-Moore methods, begin comparison from the end of the pattern instead of the beginning and precompute skip information to determine how far ahead a pattern search can skip in the input whenever a non-matching character is encountered. Generally, Boyer-Moore family algorithms improve faster as the pattern being searched for becomes longer. In many cases, the techniques used to skip characters in simple string matching approaches can be extended to regular expression matching. Widely known techniques used to facilitate character

skipping in regular expression matching include necessary strings and backward suffix matching inspired by the Backward Dawg Matching (BDM) algorithm [11].

3.2 Bit-parallel Simulation of Automata

bit-parallelism [3]

Shift-Or / Shift-And [15]

Bit-parallel suffix automata (Backward Non-Deterministic Dawg Matching (BNDM) [12] algorithm)

i.e. bit parallel simulation of automata

3.3 Software Tools

4 Bit-parallel Data Streams

The bit-parallel data streams use the wide SIMD registers commonly found on commodity processors to process byte positions at a time using bitwise logic, shifting and other operations.

A signficant advantage of the bit-parallel data stream methods over other pattern matching methods that rely on bit-parallel automata simulation is the potential to skip register width characters.

Skip SIMD register width.

4.1 Match Star

Match Star takes a marker bitstream and a character class bitstream as input. It returns all positions that can be reached by advancing the marker bitstream zero or more times through the character class bitstream.

Figure one illustrates the Match Star method. The second and third rows are the input bitstreams: the initial marker position bitstream and the character class bitstream derived from the source data.

In the first operation (T_0), marker positions that cannot be advanced are temporarily removed from consideration by masking off marker positions that aren't character class positions using bitwise logic. Next, the temporary marker bitstream is added to the character class bitstream. T_1 has 1s in three types of positions. There will be a 1 immediately following a block of character class positions that spanned one or more marker positions, at any character class positions that weren't affected by the addition (and are not part of the desired output), and at any marker position that wasn't the first in its block of character class positions. Any character class positions that have a 0 in T_1 were affected by the addition and are part of the desired output. These positions are obtained and the undesired 1 bits are removed by XORing with the character class stream. T_2 is now only missing marker positions that were removed in the first step as well as marker positions that were 1s in T_1. The output marker stream is obtained by ORing T_2 with the initial marker stream.

source data	`--142578---125-231-----127--5394---94761205-`
M_0	`.......1......1..1..1...1.............1..1..`
$D =$ `[0-9]`	`..111111...111.111.....111..1111...11111111.`
$T_0 = M_0 \wedge D$	`.......1.........1......1.............1..1..`
$T_1 = T_0 + D$	`.1.........1111.......1..1..1111..1...1...1.`
$T_2 = T_1 \oplus D$	`.1111111......1111....111.........1111.111..`
$M_1 = T_2 \mid M_0$	`.1111111......1111..1.111.........11111111..`

Figure 1: Match Star

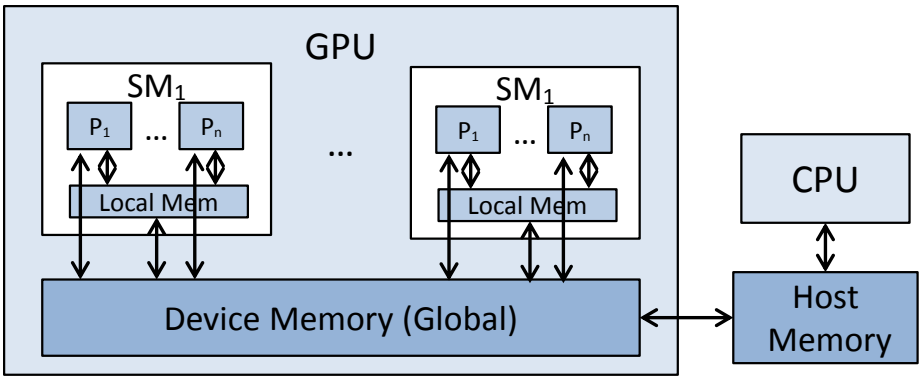

Figure 2: GPU Architecture

5 Compiler Technology

6 Methodology

We compare the performance of our bit-parallel data stream techniques against several fast regular expression matching implemenations, Gnu grep, agrep [14], nr-grep, and re2.

7 Experimental Results

8 GPU Implementation

AMD GPU architecture is used in this study, running OpenCL applications. OpenCL is a C-Like programming model that provides interface to execute the kernel on GPU. The kernel is a user-defined code section that, in this case, holds the core regular expression algorithm. The host reads the input data and transfers it to device memory.

9 Conclusion

References

[1] Abou-assaleh, T., and Ai, W. Survey of global regular expression print (grep) tools. Tech. rep., 2004.

[2] Aho, A. V., Sethi, R., and Ullman, J. D. *Compilers: principles, techniques, and tools*. Addison Wesley, 2007.

[3] Baeza-yates, R. A., Encalada, B., and Gonnet, G. H. A new approach to text searching.

[4] Boyer, R. S., and Moore, J. S. A fast string searching algorithm. *Communications of the ACM 20*, 10 (1977), 762–772.

[5] Cameron, R., Herdy, K., and Amiri, E. Parallel bit stream technology as a foundation for xml parsing performance. In *International Symposium on Processing XML Efficiently: Overcoming Limits on Space, Time, or Bandwidth* (2009), vol. 8.

[6] Cameron, R. D., Amiri, E., Herdy, K. S., Lin, D., Shermer, T. C., and Popowich, F. P. Parallel scanning with bitstream addition: An xml case study. In *Euro-Par 2011 Parallel Processing*. Springer, 2011, pp. 2–13.

[7] Cameron, R. D., Herdy, K. S., and Lin, D. High performance xml parsing using parallel bit stream technology. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds* (2008), ACM, p. 17.

[8] Kleene, S. C. Representation of events in nerve nets and finite automata.

[9] Lin, D., Medforth, N., Herdy, K. S., Shriraman, A., and Cameron, R. Parabix: Boosting the efficiency of text processing on commodity processors. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on* (2012), IEEE, pp. 1–12.

[10] Navarro, G. Nr-grep: A fast and flexible pattern matching tool. *Software Practice and Experience (SPE 31* (2000), 2001.

[11] Navarro, G. Pattern matching. *Journal of Applied Statistics 31* (2002).

[12] Navarro, G., and Raffinot, M. A bit-parallel approach to suffix automata: Fast extended string matching. In *Combinatorial Pattern Matching* (1998), Springer, pp. 14–33.

[13] Thompson, K. Programming techniques: Regular expression search algorithm. *Communications of the ACM 11*, 6 (1968), 419–422.

[14] Wu, S., and Manber, U. Agrep - a fast approximate pattern-matching tool. In *In Proc. of USENIX Technical Conference* (1992), pp. 153–162.

[15] Wu, S., and Manber, U. Fast text searching: allowing errors. *Communications of the ACM 35*, 10 (1992), 83–91.