Parallel Regular Expression Matching

Michael Whittaker (mjw297)

October 8, 2015

Regular Expressions

Regular expressions are a concise way to specify a set of strings to be matched. For example, given an alphabet $\Sigma = \{0, 1\}$, the regular expression $01(01)^*$ matches the bitstrings $\{01, 0101, 010101, \ldots\}$. Regular expressions are a common tool used to perform string manipulation, lex and parse code written in various programming languages, etc. They can also be formalized rigorously, as shown in Figure 2 [3], and studied in a principled manner. For example, NetKAT [1] is a network programming language and logic that is based on KAT, a formalism that underlies regular expressions.

$$
\begin{array}{l}
\langle r,s \rangle ::= \\
\quad | \quad \emptyset \\
\quad | \quad \epsilon \\
\quad | \quad \langle c \rangle \\
\quad | \quad \langle r \rangle + \langle s \rangle \\
\quad | \quad \langle r \rangle \cdot \langle s \rangle \\
\quad | \quad \langle r \rangle^*
\end{array}
\qquad
\begin{aligned}
\mathcal{L}[\![\emptyset]\!] &= \emptyset \\
\mathcal{L}[\![\epsilon]\!] &= \{\epsilon\} \\
\mathcal{L}[\![c]\!] &= \{c\} \\
\mathcal{L}[\![r + s]\!] &= \mathcal{L}[\![r]\!] + \mathcal{L}[\![s]\!] \\
\mathcal{L}[\![r \cdot s]\!] &= \{u \cdot v \mid u \in \mathcal{L}[\![R]\!], v \in \mathcal{L}[\![s]\!]\} \\
\mathcal{L}[\![r^*]\!] &= \{\epsilon\} \cup \mathcal{L}[\![r \cdot r^*]\!]
\end{aligned}
$$

Figure 2: Syntax and semantics of regular expressions.

Parallel Regular Expression Matching

A regular expression matcher is a program that determines whether a string a is accepted by a regular expression r (i.e. $a \in \mathcal{L}[\![r]\!]$). Given the popularity of regular expressions, a fast regular expression matcher is a powerful tool. One way to implement a fast regular expression matcher is with parallelization. Memeti and Pllana, for example, have developed PaREM [2]: a fast and parallel regular expression matching algorithm that achieves nearly linear speedup with respect to the number of threads. The algorithm involves partitioning the input string between a set of threads and later combining each threads partial results; a full explanation of the algorithm is left to the paper.

Project Proposal

I propose to either implement and optimize a custom parallel regular expression matcher, or analyze and optimize a set of existing parallel regular expression matchers. I will analyze the performance on a set of synthesized and real-world regular expressions.

References

[1] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. Netkat: Semantic foundations for networks. *ACM SIGPLAN Notices*, 49(1):113–126, 2014.

[2] Suejb Memeti and Sabri Pllana. Parem: A novel approach for parallel regular expression matching. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pages 690–697. IEEE, 2014.

[3] Scott Owens, John Reppy, and Aaron Turon. Regular-expression derivatives re-examined. *Journal of Functional Programming*, 19(02):173–190, 2009.