Journal of Discrete Algorithms 28 (2014) 61–72

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Fast and flexible packed string matching ☆

Simone Faro [a,*], M. Oğuzhan Külekci [b]

[a] Dipartimento di Matematica e Informatica, Università di Catania, Italy
[b] İstanbul Medipol University, Faculty of Engineering and Natural Sciences, Turkey

ARTICLE INFO

Article history:
Available online 24 July 2014

Keywords:
Exact string matching
Text algorithms
Experimental algorithms
Online searching
Information retrieval

ABSTRACT

Searching for all occurrences of a pattern in a text is a fundamental problem in computer science with applications in many other fields, like natural language processing, information retrieval and computational biology. In the last two decades a general trend has appeared trying to exploit the power of the word RAM model to speed-up the performances of classical string matching algorithms. In this model an algorithm operates on words of length w, grouping blocks of characters, and arithmetic and logic operations on the words take one unit of time.

In this paper we use specialized word-size packed string matching instructions, based on the Intel streaming SIMD extensions (SSE) technology, to design a very fast string matching algorithm. We evaluate our solution in terms of efficiency, stability and flexibility, where we propose to use the deviation in running time of an algorithm on distinct equal length patterns as a measure of stability.

From our experimental results it turns out that, despite their quadratic worst case time complexity, the new presented algorithm becomes the clear winner on the average in many cases, when compared against the most recent and effective algorithms known in literature.

1. Introduction

Given a text t of length n and a pattern p of length m over some alphabet Σ of size σ, the *exact string matching problem* consists in finding *all* occurrences of the pattern p in t. This problem has been extensively studied in computer science because of its direct application to many areas. Moreover, string matching algorithms are the basic components in many software applications and play an important role in theoretical computer science by providing challenging problems.

In a computational model where the matching algorithm is restricted to read all the characters of the text one by one the optimal complexity is $\mathcal{O}(n)$, and was achieved the first time by the well known Knuth–Morris–Pratt algorithm [26] (KMP). However, in many practical cases it is possible to avoid reading all the characters of the text achieving sub-linear performances on the average. The optimal average $\mathcal{O}(\frac{n \log_\sigma m}{m})$ time complexity [35] was reached for the first time by the Backward-DAWG-Matching algorithm [11] (BDM). However, all algorithms with a sub-linear average behavior may have to read all the text characters in the worst case. It is interesting to note that many of those algorithms have an even worse $\mathcal{O}(nm)$-time complexity in the worst-case [10,19,22].

☆ A preliminary version of the results presented in this paper has been previously published in [15].

* Corresponding author.
E-mail addresses: faro@dmi.unict.it (S. Faro), okulekci@medipol.edu.tr (M.O. Külekci).

http://dx.doi.org/10.1016/j.jda.2014.07.003
1570-8667/

In the last two decades a lot of work has been made in order to exploit the power of the word RAM model of computation to speed-up classical string matching algorithms. In this model, the computer operates on words of length w, thus blocks of characters are read and processed at once. This means that usual arithmetic and logic operations on the words all take one unit of time.

Most of the solutions which exploit the word RAM model are based on either the *bit-parallelism* technique or the *packed string matching* technique.

The *bit-parallelism* technique [1] takes advantage of the intrinsic parallelism of the bit operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to w. Bit-parallelism is particularly suitable for the efficient simulation of nondeterministic automaton. The Shift-Or [1] (SO) algorithm is the first of this genre, which simulates efficiently the nondeterministic version of the KMP automaton and runs in $\mathcal{O}(n\lceil\frac{m}{w}\rceil)$. It is still considered among the best practical algorithms in the case of very short patterns and small alphabets [22,19]. Later a very fast BDM-like algorithm (BNDM), based on the bit-parallel simulation of the nondeterministic suffix automaton, was presented in [31]. Some variants of the BNDM algorithm [16,18,12,32] are among the most practical efficient solutions in literature (see [22,19]). However, the bit-parallel encoding requires one bit per pattern symbol, for a total of $\lceil\frac{m}{w}\rceil$ computer words. Thus, as long as a pattern fits in a single computer word, bit-parallel algorithms are extremely fast, otherwise their performances degrade considerably as $\lceil\frac{m}{w}\rceil$ grows. Though there are a few techniques to maintain good performances in the case of long patterns [28,13,8,9], such limitation is intrinsic.

In the *packed string matching* technique multiple characters are packed into one larger word, so that the characters can be compared in bulk rather than individually. In this context, if the characters of a string are drawn from an alphabet of size σ, then $\lfloor\frac{w}{\log\sigma}\rfloor$ different characters fit in a single word, using $\lceil\log\sigma\rceil$ bits per characters. The packing factor is $\alpha = \lfloor\frac{w}{\log\sigma}\rfloor$.[1]

A first theoretical result in packed string matching was proposed by Fredriksson [23]. He presented a general scheme that can be applied to speed-up many pattern matching algorithms. His approach relies on the use of the *four Russian* technique (i.e. tabulation), achieving in favorable cases an $\mathcal{O}(n^{\varepsilon}m)$-space and $\mathcal{O}(\frac{n}{m\log\sigma}+n^{\varepsilon}m+occ)$-time complexity, where $\varepsilon > 0$ denotes an arbitrary small constant, and occ denotes the number of occurrences of p in t. Bille [5] presented an alternative solution with $\mathcal{O}(\frac{n}{\log_\sigma n}+m+occ)$-time and $\mathcal{O}(n^{\varepsilon}+m)$-space complexities by an efficient segmentation and coding of the KMP automaton. Belazzougui [2] proposed a packed string matching algorithm which works in $\mathcal{O}(\frac{n}{m}+\frac{n}{\alpha}+m+occ)$ time and $\mathcal{O}(m)$ space, reaching the optimal $\mathcal{O}(\frac{n}{\alpha}+occ)$-time bound for $\alpha \le m \le \frac{n}{\alpha}$. More recently, Belazzougui and Raffinot [3] introduced an average-optimal time string matching algorithm for packed strings, which achieves $\mathcal{O}(n/m)$ query time. However, none of these results leads to practical algorithms.

The first algorithm that achieves good practical and theoretical results was very recently proposed by Ben-Kiki et al. [4]. The algorithm is based on two specialized packed string instructions, the pcmpestrm and the pcmpestri instructions [29], and reaches the optimal $\mathcal{O}(\frac{n}{\alpha}+occ)$-time complexity requiring only $\mathcal{O}(1)$ extra space. Moreover the authors showed that their algorithm turns out to be among the fastest string matching solutions in the case of very short patterns. However, it has to be noticed that on the family of Intel Sandy Bridge processors, which we consider as the benchmark platform for the implementations throughout the study, pcmpestrm and pcmpestri have 2-cycle throughput and 7- and 8-cycle latency, respectively [29].

When the length of the searched pattern increases, another algorithm named Streaming SIMD Extensions Filter (SSEF), presented by Külekci in [27] (and extended to multiple pattern matching in [14]), exploits the advantages of the word-RAM model. Specifically it uses a filter method that inspects blocks of characters instead of reading them one by one. Despite its $\mathcal{O}(nm)$ worst case time complexity, the SSEF algorithm turns out to be among the fastest solutions when searching for long patterns [22,19].

Efficient solutions have been also designed for searching on packed DNA sequences [33,17]. However in this paper we do not take into account this type of solutions since they require a different type of data representation.

Streaming SIMD technology offers single-instructions to perform a variety of tests on packed strings. Unfortunately those instructions are heavier than other instructions provided in the same family as a consequence of their relatively high latencies. Hence, in this paper we focus on design of algorithms using instructions with low latency and high throughput, when compared with those used in [4].

Specifically we introduce a new practical and efficient algorithm for the exact packed string matching problem that turns out to be faster than the best algorithms known in literature in most practical cases [15].

The newly presented algorithm, named Exact Packed String Matching (EPSM), is based on four different search procedures used for, respectively, very short patterns ($0 < m < \frac{\alpha}{2}$), short patterns ($\frac{\alpha}{2} \le m < \alpha$), medium length patterns ($m \ge \alpha$) and long patterns ($m \ge w$). All search procedures have an $\mathcal{O}(nm)$ worst case time complexity. However, they have very good performances on average. In the case of very short patterns, i.e. when $m \le \frac{\alpha}{2}$, the first two search procedures achieve, respectively, an $\mathcal{O}(n+occ)$ and an optimal $\mathcal{O}(\frac{n}{\alpha}+occ)$-time complexity.

The paper is organized as follows. In Section 2, we introduce some notions and terminologies, while in Section 3 we describe the model of computations we assume for describing our solutions. We then present a new algorithm for the packed

[1] However, it is noteworthy that in practice supporting varying packing factors seems not very possible in todays SIMD technologies such as the Intel's SSE instruction set. The practical implementations assume the ASCII alphabet with size 8-bits per symbol and the packing factor used is 16 (32) symbols per block in 128-bits (256-bits) SIMD technologies.