Efficient Logic Circuit for Network Intrusion Detection

Huang-Chun Roan[1],Chien-Min Ou[2],Wen-Jyi Hwang[**1]and Chia-Tien Dan Lo[3]

[1] Graduate Institute of Computer Science and Information Engineering,
National Taiwan Normal University, Taipei, 117, Taiwan
[2] Department of Electronics Engineering,
Ching Yun University, Chungli, 320, Taiwan
[3] Department of Computer Science, University of Texas at San Antonio,
San Antonio, TX 78249, USA

Abstract. A novel architecture for a hardware-based network intrusion detection system (NIDS) is presented in this paper. The system adopts an FPGA-based signature match co-processor as a core for the NIDS. The signature matcher is based on an algorithm that employs simple shift registers, or-gates, and ROMs in which patterns are stored. As compared with related work, experimental results show that the proposed work achieves higher throughput and less hardware resource in the FPGA implementations of network intrusion detection.

1 Introduction

A network intrusion detection system (NIDS) monitors network traffic for suspicious data patterns and activities. It informs system administrators when malicious traffic is detected so that proper actions may be taken. Many NIDSs such as SNORT [1] prevent computer networks from attacks using pattern-matching rules. The computational complexity of NIDSs therefore may be high because of the requirement of the string matching during their detection processes.

The SNORT system running on general purpose processors may only achieve up to 60 Mbps [2] throughput because of the high computational complexity. Since these systems do not operate at line speed, some malicious traffic can be dropped and thus may not be detected. To accelerate the speed for intrusion detection, several FPGA-based approaches have been proposed [2–6]. Because the NIDS rules do not change frequently, the cost for FPGA implementations may not be high as compared with their software-based counterparts. Moreover, the hardware implementation can exploit parallelism for string matching so that the throughput of NIDSs can be increased.

One popular way for FPGA implementation is based on regular expressions [4, 7], which results in designs with low area cost and moderate throughput acceleration. In this approach, a regular expression is generated for every pattern.

** To whom all correspondence should be sent
This project is partially supported by the Center for Infrastructure Assurance and Security at UTSA and US Air Force under grant #26-0200-62

Each regular expression is then implemented by a nondeterministic finite automata (NFA) or deterministic finite automata (DFA). In the finite automata implementations, efficient exploitation of parallelism is difficult because the input stream is scanned one character at a time. Another alternative for FPGA implementation is to use the content addressable memory (CAM) [3, 6]. By the employment of multiple comparators in the CAM, the processing of multiple input characters per cycle is possible. This may effectively increase the throughput at the expense of higher area cost.

The objective of this paper is to present a novel FPGA implementation approach for NIDSs achieving both high throughput and low area cost. The proposed architecture is based on the shift-or algorithm for exact string matching [8]. The shift-or algorithm is an effective software approach for pattern matching because of its simplicity and flexibility. However, it may not perform well when the pattern size is larger than the computer word size, which is the case for many SNORT patterns. Accordingly, the software implementation of shift-or algorithm may not be suited for SNORT systems.

On the other hand, the hardware implementation of shift-or algorithm imposes no limitation on the pattern size. In our architecture, each SNORT pattern is only associated with a ROM and a shift register for pattern comparison, which are designed in accordance with the pattern size. Because of its simplicity, the architecture may operate at a higher clock rate as compared with other implementations. In addition, the number of logic elements (LEs) for the circuit implementation is reduced significantly when the ROM is realized by the embedded RAM blocks of the FPGA. The area cost therefore may be lower than the existing designs [3, 6]. Moreover, although the proposed architecture in its simplest form only processes one character at a time, the architecture can be extended to further enhance the throughput of the circuit. Multiple characters can be scanned and processed in one cycle at the expense of slight increase in area cost.

The proposed architecture has been prototyped and simulated by the Altera Stratix FPGA. Experimental results reveal that the circuit attains the throughput up to 5.14 Gbits/sec with area cost of 1.09 LE per character. The proposed architecture therefore is an effective solution to high throughput and low area cost NIDS hardware design.

2 Preliminaries

This section briefly describes the shift-or algorithm for exact string matching. Suppose we are searching for a *pattern* $P = p_1p_2...p_m$ inside a large *text* (or *source*) $T = t_1t_2...t_n$, where $n \gg m$. Every character of P and T belongs to the same alphabet $\Sigma = \{s_1, ..., s_{|\Sigma|}\}$.

Let R_j be a bit vector containing information about all matches of the prefixes of P that end at j. The vector contains $m+1$ elements $R_j[i], i = 0, ..., m$, where $R_j[i] = 0$ if the first i characters of the pattern P match exactly the last i characters up to j in the text (i.e., $p_1p_2...p_i = t_{j-i+1}t_{j-i+2}...t_j$). The transition

	s_k	a	b	c
S_k	$i=1$	0	1	1
	$i=2$	0	1	1
	$i=3$	1	0	1

(a)

j	0	1		2		3		4		5	
i	R_j	S_c	R_j	S_c	R_j	S_c	R_j	S_c	R_j	S_c	R_j
0	0		0		0		0		0		0
1	1	0	0	1	1	0	0	0	0	1	1
2	1	0	1	1	1	0	1	0	0	1	1
3	1	1	1	1	1	1	1	1	1	0	(0)

(b)

Fig. 1. An example of shift-or algorithm with pattern $P = aab$ and text $T = acaab$, (a) The bit vector S_k associated with each symbol $s_k \in \Sigma = \{a, b, c\}$ for the pattern P, (b) The bit vector R_j for the text T, where one occurrence of P is found (encircled).

from R_j to R_{j+1} is performed by the recurrence:

$$R_{j+1}[i] = \begin{cases} 0, \text{ if } R_j[i-1] = 0 \text{ and } p_i = t_{j+1}, \\ 1, \text{ otherwise,} \end{cases} \tag{1}$$

where the initial conditions for the recurrence are given by $R_0[i] = 1, i = 1, ..., m$, and $R_j[0] = 0, j = 0, ..., m$. The recurrence can be implemented by the simple shift and OR operations. To see this fact, we first associate each symbol $s_k \in \Sigma$ a bit vector S_k containing m elements, where the i-th element $S_k[i]$ is given by

$$S_k[i] = \begin{cases} 0, \text{ if } s_k = p_i, \\ 1, \text{ otherwise.} \end{cases} \tag{2}$$

Assume $t_{j+1} = s_c$. Based on eq.(2), the recurrence shown in eq.(1) can then be rewritten as

$$R_{j+1}[i] = R_j[i-1] \; OR \; S_c[i], i = 1, ..., m. \tag{3}$$

We can clearly see now the transition from R_j to R_{j+1} involves to no more than a shift of R_j and an OR operation with S_c, where $t_{j+1} = s_c$. Figure 1 shows an example of the exact string matching based on the shift-or algorithm, where $P = aab$ and $\Sigma = \{a, b, c\}$. The bit vector S_k associated with each $s_k \in \Sigma$, which is determined by eq.(2), is given in Figure 1.(a). In this example, $T = acaab$. Therefore, $s_c = a, c, a, a$ and b for $j = 1, 2, 3, 4$ and 5, respectively. The S_c associated with s_c for each j can be found from the table shown in Figure 1.(a). Given S_c and R_{j-1}, the R_j can be computed by eq.(3), as shown in Figure 1.(b). Note that, when $j = 5$, it can be found from Figure 1.(b) that $R_j[3] = 0$. Therefore, one occurrence of P is found when $j = 5$.

3 The Architecture

The proposed architecture for SNORT pattern matching is shown in Figure 2. The architecture contains M modules, where M is the number of SNORT rules

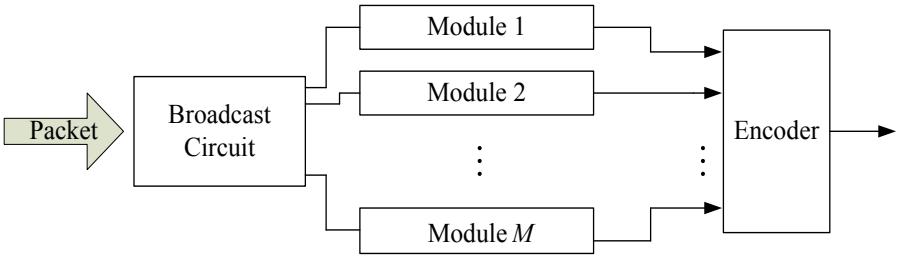

Fig. 2. The basic structure of the proposed circuit, where M is the number of rules implemented by the circuit.

for intrusion detection. The incoming source is first broadcasted to all the modules. Each module is responsible for the pattern matching of a single rule. The encoder in the architecture receives the intrusion alarms issued by the modules detecting matched strings, and transfers the alarms to the administrators for proper actions.

3.1 Basic module circuit

The module operates by scanning the source string one character at a time. Therefore, *after* the clock cycle j, the circuit completes the string matching process up to t_j. Moreover, the character t_{j+1} is the input character to the module during the clock cycle $(j+1)$. Assume $t_{j+1} = s_c$. The input character t_{j+1} is first delivered to the ROM for the retrieval of S_c to the OR gates. Each OR gate i has two inputs: one is from the i-th output bit of the ROM (i.e., $S_c[i]$), and the other is from the output of FF $(i-1)$, which contains $R_j[i-1]$ during the clock cycle $j+1$. From eq.(3), it follows that the OR gate i produces $R_{j+1}[i]$, which is then used as the input to the FF i. The $R_{j+1}[i]$ therefore will become the output of FF i during the clock $j+2$ for the subsequent operations.

Note that, during the clock cycle $j+1$, the m-th OR gate produces $R_{j+1}[m]$, which is identical to 0 when $p_1p_2...p_i = t_{j-i}t_{j-i+1}...t_{j+1}$. In this case, the module will issue an intrusion alarm to the encoder of the NIDS. Therefore, the output of the OR gate m is the check point of exact string matching with pattern size m.

For the FPGA devices with embedded memories, the ROM may be implemented solely by the memory bits. Hence, the LEs are required only for the implementation of the shift register. The circuit therefore may have low area cost (in terms of the number of LEs) for the FPGA implementation of SNORT rules.

To implement the ROM, we first note that each ASCII character in a SNORT rule contains 8 bits. Therefore, $|\Sigma| = 256$ and the ROM contains 256 entries for pattern matching. The ROM size can be reduced by observing the fact that some

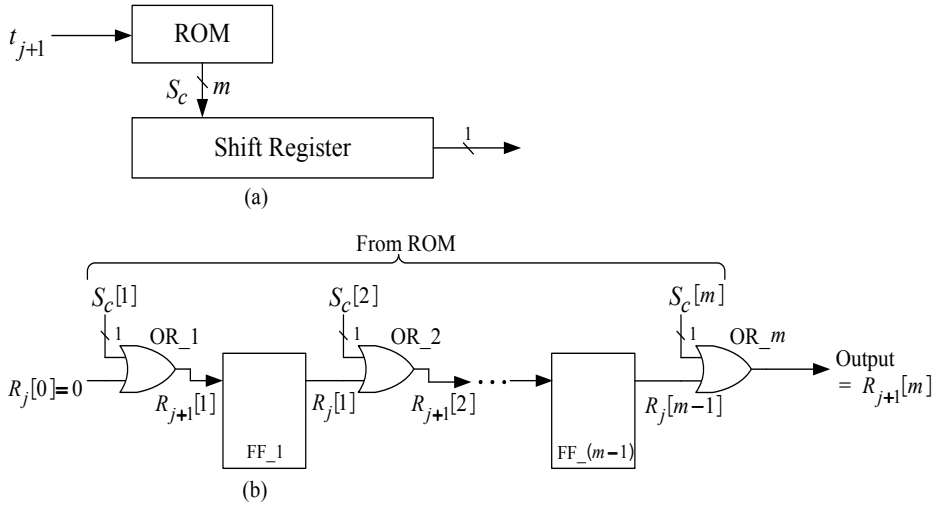

Fig. 3. The basic circuit of each module for exact pattern matching, (a) The block diagram of the circuit, (b) The shift register circuit during clock cycle $j + 1$.

symbols s_k in the alphabet Σ may not appear in the pattern P. Accordingly, they have the same bit vectors $S_k = \mathbf{1}$. These symbols then can share the same entry in the ROM for storage size reduction. One simple way to accomplish this is to augment a new symbol s_0 (with $S_0 = \mathbf{1}$) in the alphabet Σ. All the symbols s_k having $S_k = \mathbf{1}$ are then mapped to s_0 by a symbol encoder as shown in Figure 4. These symols then share the same entry associated with s_0 in the ROM.

Since the LEs are required for the implementation of the symbol encoders, the area cost may be high if each module has its own symbol encoder. We can lower the area cost by first dividing the SNORT rules into several groups, where the rules in each group use the same set of symbols. Therefore, all the rules in the same group can share the same symbol encoder, as shown in Figure 5. The overhead for the realization of symbol encoders then can be reduced.

3.2 High throughput module circuit

The basic module circuit shown in Figure 3 only process one character per cycle. The throughput of the NIDS can be improved further by processing q characters at a time. This can be accomplished by grouping q consecutive characters in the source into a single symbol. Without loss of generality, we consider $q = 2$. Let $\Omega = \{x_1, ..., x_{|\Omega|}\}$ be the alphabet for the new symbols, where $x_i = (y_1, y_2)$, and $y_1, y_2 \in \Sigma$.

Based on Ω, a pattern P can be rewritten as $P = u_1u_2...u_{\lceil m/2 \rceil}$, where $u_i = (p_{2i-1}, p_{2i})$. Note that $u_{\lceil m/2 \rceil} = (p_{m-1}, p_m)$ when m is even. However,

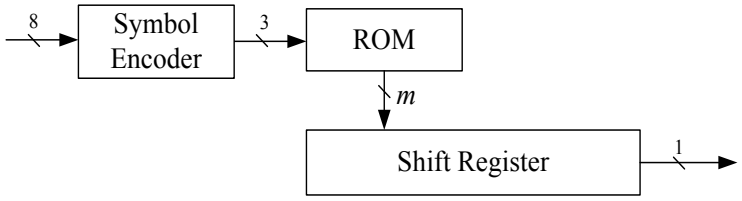

Fig. 4. The augment of a symbol encoder for reducing the ROM size. In this example, each input character is assumed to be an ASCII code (8 bits). We also assume the SNORT rule uses only 7 symbols in the alphabet. The output of the symbol encoder therefore is 3 bits.

when m is odd, $u_{\lceil m/2 \rceil} = (p_m, \phi)$, where ϕ denotes "don't care," and can be any character in Σ. We can then associate a bit vector X_k containing $\lceil m/2 \rceil$ elements for each symbol $x_k \in \Omega$, where the i-th element of X_k is given by

$$X_k[i] = \begin{cases} 0, \text{ if } x_k = u_i, \\ 1, \text{ otherwise.} \end{cases} \tag{4}$$

A ROM containing $X_1, ..., X_{|\Omega|}$ can then be constructed for shift-or operations. In this case, the ROM contain $|\Omega| = |\Sigma|^2$ entries, where each entry has $\lceil m/2 \rceil$ bits. It is therefore necessary to employ a larger ROM for a module with higher throughput. A symbol encoder similar to that shown in Figure 4 can be employed to reduce the ROM size. In this case we augment a new symbol x_0 (with $X_0 = \mathbf{1}$) in the alphabet Ω. All the symbols x_k having $X_k = \mathbf{1}$ are then mapped to x_0 by the symbol encoder.

Note that the string matching operations ending at j over the alphabet Ω are equivalent to the operations ending at either $2j$ or $2j + 1$ (but not both) over the alphabet Σ. It is necessary to perform the matching process ending at every location of the source over the alphabet Σ. Therefore, we employ two shift registers in the module as shown in Figure 6, where one is for even locations, and the other is for odd locations. Moreover, since each entry of the ROM contains only $\lceil m/2 \rceil$ bits, the shift registers with $\lceil m/2 \rceil - 1$ FFs and $\lceil m/2 \rceil$ OR gates are sufficient for the operations. When m is even, the total number of FFs and OR gates in the high throughput circuit is identical to those in the basic circuit presented in the previous subsection.

To perform the string matching operations ending at the *even* locations of the source over Σ, we convert the source T to the sequence $T_e = e_1 e_2 ...$ over alphabet Ω, where $e_j = (t_{2j-1}, t_{2j})$. During the clock cycle $j + 1$, symbol e_{j+1} is fetched to the ROM. This is equivalent to the scanning of two characters t_{2j+1} and t_{2j+2} simultaneously for shift-or operations.

The shift-or operations at the *odd* locations of the source can be performed in the similar manner, except that the source T is extracted as $T_o = o_1 o_2 ...$, where $o_j = (t_{2j}, t_{2j+1})$. During the clock cycle $j + 1$, we scan the symbol o_j. From

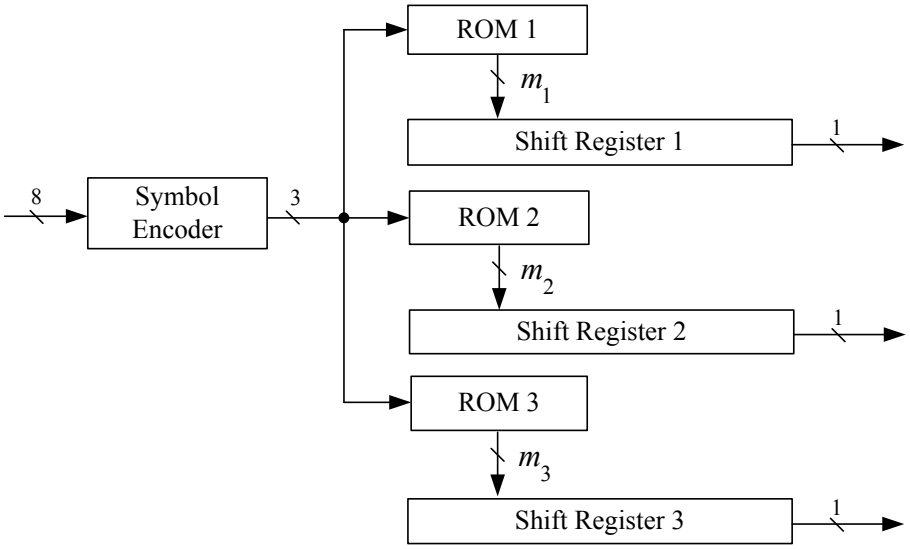

Fig. 5. The sharing of the same symbol encoder by three different SNORT rules. Each character is also assumed to be an ASCII code. All the SNORT rules use the same alphabet consisting of 7 symbols.

Figure 6, we observe that o_j can be obtained from e_j and e_{j+1} via delaying and broadcasting operations. Therefore, the shift-or operations at even and odd locations share the same input as shown in the figure.

4 Experimental Results and Comparisons

This section presents experimental results of the proposed architecture. Table 1 compares the throughput, the number of LEs per character, total number of memory bits and operating frequency of the proposed circuits with and without the symbol encoder. The throughput indicates the maximum number of bits per second the circuit can process. For the sake of simplicity, only the circuits processing one character at a time (i.e., $q = 1$) are considered in the table. Moreover, for the circuit with symbol encoders, the rules using the same set of symbols will share the same symbol encoder, as shown in Figure 5. We use the Altera Quartus II as the tool for circuit synthesization. The target FPGA device is Stratix EP1S80.

Table 2 shows the throughput, the number of LEs per character, the memory bits and operating frequency of our circuits with $q = 1$ and $q = 2$ realized by the FPGA device Stratix EP1S80. As shown in Table 2, because the circuit with $q = 2$ processes two characters for each clock cycle, it has higher throughput than that of the cuicuit with $q = 1$, which processes one character per cycle

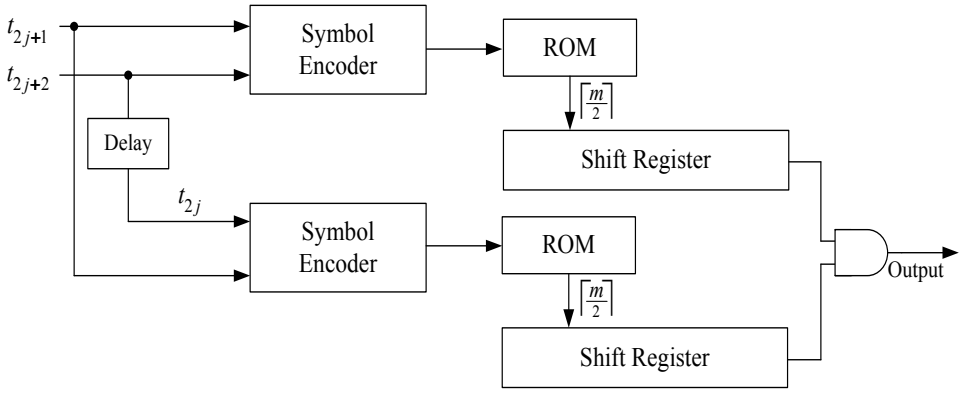

Fig. 6. The structure of a high throughput module circuit processing two characters at a time ($q = 2$).

Table 1. Comparisons of the proposed architecture with and without symbol encoder, where the number of characters available for pattern matching is 1568 characters.

Design	Throughput (Gb/s)	Logic cells /char	Memory bits	Operating Frequency (MHz)
Without symbol encoder	2.57	0.97	387,584	321.03
With symbol encoder	2.57	0.99	25,738	321.03

only. On the other hand, it can also be observed from Table 2 that the circuit with $q = 2$ has slightly higher number of LEs per character. This is because the circuit has more complex address encoders for reducing the storage size in ROM. In addition, for $q = 2$, the string matching operations for each rule requires two independent ROMs as shown in Figure 6. Therefore, the number of memory bits used by the circuit with $q = 2$ is higher than that of the circuit with $q = 1$.

Table 3 compares the FPGA implementations of the proposed architecture with those of the existing related works. Note that the exact comparisons of these circuits may be difficult because they use different distance measures, and are realized by different FPGA devices. However, it can still be observed from the table that our circuits have effective throughput-area performance as compared with existing work. This is because our design is based on the simple shift-or algorithm. The simplicity of circuit allows the string matching operations to be performed at high clock rate with small hardware area. In particular, when $q = 2$, our circuit attains the throughput of 5.14 Gbits/sec while requiring only

Table 2. Comparisons of the proposed architecture with $q = 1$ and $q = 2$, where the number of characters available for pattern matching is 1568 characters.

Design	Throughput (Gb/s)	Logic cells /char	Memory bits	Operating Frequency (MHz)
$q = 1$	2.57	0.99	25,738	321.03
$q = 2$	5.14	1.09	40,768	321.03

the area cost of 1.09 LEs per character. These facts demonstrate the effectiveness of our design.

Table 3. Comparisons of various string matching FPGA designs.

Design	Device	Throughput (Gb/s)	Chars	Logic cells /char
Proposed architecture ($q = 1$)	Altera Stratix EP1S40	2.25	5004	0.96
Proposed architecture ($q = 2$)	Altera Stratix EP1S40	5.14	1568	1.09
Gokhale et al. [3]	Xilinx VirtexE-1000	2.2	640	15.2
Hutchings et al. [4]	Xilinx Vertix-1000	0.248	8003	2.57
Singaraju et al. [5]	Xilinx Virtex2VP30-7	6.41	1021	2.2
Sourdis-Pnevmatikatos [6]	Xilinx Spartan33-5000	4.91	18000	3.69
Moscola et al. [7]	Xilinx VirtexE-2000	1.18	420	19.4

5 Conclusion

A novel FPGA implementation of network intrusion detection based on shift-or algorithm is presented in this paper. The proposed algorithm in the basic form processes one character at a time, and contains only a ROM and a simple shift register for each pattern matching. The throughput can be further enhanced by processing multiple characters in parallel. Both the basic form and two-character at a time of the proposed algorithm are implemented in our experiments. Comparisons with exsiting work reveal that our design is one of the cost-effective solutions to the FPGA implementations of the network intrusion detection.

References

1. SNORT official web site http://www.snort.org.
2. Ramirez, T., Lo, C.D.: Rule set decomposition for hardware network intrusion detection. in the 2004 International Computer Symposium (ICS 2004) (2004)
3. Gokhale, M., Dubois, D., Dubois, A., Boorman, M., Poole, S., Hogsett, V.: Granidt: towards gigabit rate network intrusion detection technology. Proceedings of the International Conference on Field Programmable Logic and Application (2002) 404–413
4. Hutchings, B.L., Franklin, R., Carver, D.: Assisting network intrusion detection with reconfigurable hardware. Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (2002) 111–120
5. Singaraju, J., Bu, L., Chandy, J.A.: A signature match processor architecture for network intrusion detection. Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (2005) 235–242
6. Sourdis, I., Pnevmatikatos, D.N.: Pre-decoded cams for efficient and high-speed nids pattern matching. Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (2004) 258–267
7. Moscola, J., Lockwood, J.W., Loui, R.P., Pachos, M.: Implementation of a content-scanning module for an internet firewall. Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (2003) 31–38
8. Baeza-Tates, R., Gonnet, G.: A new approach to text searching. Communications of the ACM **35** (1992) 74–82