社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

4IGU エミュレータと MPU を用いたウイルス検出エンジンについて

中原　啓貴† 　笹尾　勤† 　松浦　宗寛†

† 九州工業大学 情報工学部 〒 820–8502 福岡県飯塚市大字川津 680–4

あらまし　本論文では, 2 段階マッチングに基づく ClamAV のハードウェアアクセラレータを提案する. 第一段階では, ウイルスパターンの一部をハードウェアを用いて検出する. 第二段階では, ウイルスパターン全部をソフトウェアを用いて検出する. 厳密マッチングは 4 個の Index Generation Unit (IGU) を模擬する 4IGU エミュレータで行う. 正規表現マッチングは FPGA の組込みプロセッサで行う. 提案手法は小規模な FPGA と DDR2-SDRAM で実現するため, 安価である. また, 大規模な FPGA や TCAM を用いる手法と比較して消費電力が低い. ClamAV のパターン 1,290,808 個を Xilinx 社の FPGA と 3 個の DDR2-SDRAM を用いて実装し, 他の手法と比較を行った結果, スループットで 1.45 倍, LC 利用率で 16.3 倍, 組込みメモリ利用率で 49.6 倍優れていた.

A Virus Scanning Engine Using a 4IGU Emulator and an MPU

Hiroki NAKAHARA†, Tsutomu SASAO†, and Munehiro MATSUURA†

† Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan

Abstract　This paper shows a virus scanning system using two-stage matching. In the first stage, a hardware filter quickly detects a part of the virus pattern, while in the second stage, the MPU detects the full length of the virus pattern. To make a compact hardware filter, a finite-input memory machine (FIMM) is introduced. To further enhance the throughput, a hardware filter based on *s*-FIMM is used. To reduce the memory size, the *s*-FIMM is realized by four index generation unit emulator (4IGU emulator). The proposed system uses three DDR2-SDRAMs and a small FPGA. Thus the power consumption is lower than the TCAM-based method. Also, the system consists of inexpensive devices. The system loaded 1,290,617 ClamAV virus patterns. As for the area-throughput ratio, our method outperforms existing FPGA implementations.

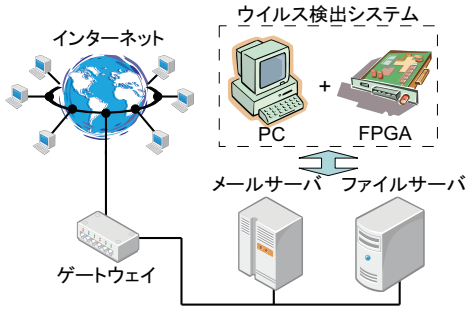

図 1　メールサーバ・ファイルサーバのウイルス検出.

1. はじめに

1.1 ウイルス検出機器

コンピュータウイルス, ワーム, スパイウェア, スパムメール等悪意を持ったソフトウェア (malicious software) を**マルウェア (Malware)** という. マルウェアに感染することにより, ボットウイルス, バックドア, キーロガーが仕掛けられ, ID やパスワードの搾取, 情報の盗難, 不正な遠隔操作が行われており社会問題となっている. ファイルサーバやメールサーバ向けオープンソースのウイルス検出ソフトである ClamAV は高々数 10 Mbps の性能であり [15], 数 Gbps でデータを送受信しているこれらのサーバのウイルス検出が追いつかない. 本論文では, ファイルサーバやメールサーバ向けウイルス検出システムについて述べる (図 1). ウイルス検出システムには以下の項目が要求される.

a) 高スループット
ファイルサーバやメールサーバの転送速度よりも高速であることが求められる. 本論文では FTTH (Fiber To The Home) の公称値 (1 Gbps) を超える処理速度を実現する.

b) 低消費電力・安価
スーパーコンピュータ, TCAM, ハイエンド FPGA を用いた手法が提案されている. スーパーコンピュータは高価であり, 消費電力が大きい. TCAM も消費電力が大きい. ハイエンド FPGA はデバイスが高価である. 本論文では安価な DDR2-SDRAM と小規模 FPGA を用いて低消費電力かつ安価なウイルス検出エンジンを実現する.

c) ウイルスパターンを更新可能
ウイルスパターンは短くて 1 時間程度 [7] で更新されるため, 書換え可能なハードウェアが必要である. 提案手法は FPGA と外付けメモリを書き換えることで, ウイルスパターンを更新可能である.

ClamAV (ver0.96.5) では高速かつ省メモリを達成するため, **2 段階マッチング**でウイルス検出を行う. 第一段階では, シグネチャの先頭 3 文字のみマッチングを行い, ウイルスの可能性のある部分を検出する. 第二段階では, 第一段階でマッチした部分に関してパターンと完全に一致するか調べる. 本論文も 2 段階マッチングに基づくハードウェアを提案する.

1.2 関連研究

種々の 2 段階マッチングに関する手法が提案されており, TCAM と汎用 MPU を用いる手法 [23]; ビット分割した Aho-Corasic DFA [20] とパターンマッチング専用 MPU を用いる手法 [2]; 並列ふるい法と汎用 MPU を用いる手法 [12] が挙げられる. ハッシュ法に基づくパターンマッチング法として

表 1　ClamAV のパターン (2010 年 12 月 1 日時点).

パターンの種類	個数	検出手法
MD5 Checksum (ブラックリスト) (ホワイトリスト)	760,804 723	厳密マッチング
Basic Pattern	94,227	正規表現マッチング
Google Browsing Database	434,863	厳密マッチング
Basic Pattern の組合せ	85	論理演算
圧縮ファイル (zip,rar) 解析	106	バイナリ解析
合計	1,290,808	

Cuckoo Hashing を用いた手法 [21], Bloom フィルタを用いた手法 (PERG-Rx) [6] が提案されている．ウイルスパターンは 100 万個を超えるので, 外付けメモリに格納するのが一般的であるが, メモリアクセスがボトルネックとなる．複数文字 (バイト) を同時に処理し, 性能を向上する手法が提案されており, 複数文字で遷移する DFA [1], 1.5 文字〜3 文字を同時に処理する手法 (Variable Stride 法) [13] が提案されている．

1.3　本論文の貢献点

(1) 低消費電力かつ安価なシステム
本論文では, 2 段階マッチングに基づくウイルス検出ハードウェアを FPGA と DDR2-SDRAM で実現する．提案手法は FPGA のリソースをほとんど使用しないので, 安価な FPGA で実現可能である．また, DDR2-SDRAM は SRAM や TCAM と比較して安価である．また, 大規模な FPGA や TCAM と比較して低消費電力である．
(2) 100 万個以上の ClamAV のパターンを実現
筆者らの知る限りでは, 100 万個以上の ClamAV のパターンを格納したハードウェア実装は本論文が初めてである．

本論文の構成は以下の通り．第 2 章では 2 段階マッチングに基づくウイルス検出を説明し, 第 3 章ではインデックス生成回路を用いたパターンの一部の実現法について述べ, 第 4 章ではウイルス検出エンジンを実装した結果を述べ, 第 5 章で本論文のまとめを行う．

2.　2 段階マッチングに基づくウイルス検出

2.1　ウイルス検出問題

検出対象の実行プログラムやデータを**テキスト**という．テキスト内に埋込まれたマルウェアを検出することを**ウイルス検出**という．マルウェアは特定のバイトコードで記述されており, **パターン**と呼ぶ．ウイルス検出問題は, 可変長のテキストの中から特定のパターンを探し出す**文字列照合問題 (パターンマッチング問題)** に帰着できる．

2.2　ClamAV のウイルスパターン

2010 年 12 月時点で, ClamAV (バージョン 0.96.5) [4] のパターンは 1,290,808 個である．表 1 にパターンの種類, 個数, 及び検出手法を示す．**MD5 Checksum** に関しては, マルウェアの MD5 値 (128 ビット) をパターンとし, テキストの MD5 値と**厳密マッチング**を行う．ブラックリストと一致すれば, マルウェアとみなす．正常なファイルが他のパターン検出法でマルウェアと誤検出される場合があるため, ホワイトリストも記憶しておく．**Basic Pattern** に関しては, 正規表現で記述されたパターンを検出する**正規表現マッチング**を行う．**Google Safe Browsing Database** に関しては, Google が提供している Safe Browsing API [5] を用いて送信先・送信元アドレスから MD5 値を生成し, 既知の危険な URL の MD5 値と厳密マッチングを行う．**Basic Pattern の組合せ**に関しては, Basic Pattern で検出したパターンの論理演算を行い, 真ならばマルウェアとみなす．論理演算は論理和, 論理積, 否定などが指定できる．**圧縮ファイル解析**に関しては, 圧縮ファイルのサイズ, ヘッダ解析等を行う．しかし, ClamAV は今後この解析をサポートしないことを宣言しているため, 本論文では圧縮ファイル解析を行わない．

図 2 にウイルス検出システムを示す．Safe Browsing API の処理は送信先・送信元アドレスのみに対して行われるため, 軽い処理である．また, Basic Pattern の組合せも軽い処理である．従って, これらの処理はソフトウェアで行う．一方, MD5 を計算するハードウェアは市販の IP コア [3] を用いる．本論文では, MD5 checksum, Basic Pattern, 及び Google Browsing Database のパターンを検出する**ウイルス検出エンジン**のみ実

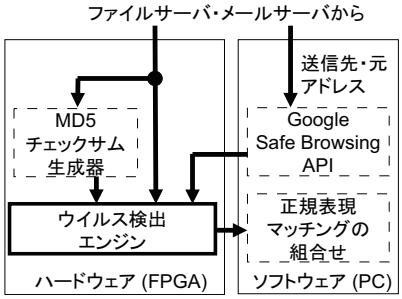

図 2　ウイルス検出システム.

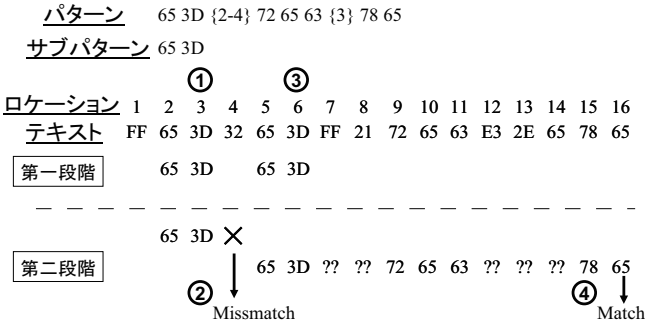

図 3　2 段階マッチングの例.

現する．正規表現マッチングは厳密マッチングの一般形であるため, 正規表現マッチングの実現を考える．

2.3　ClamAV のパターンの正規表現

パターンは文字と特殊な文字である**メタ文字**から成る**正規表現**で記述される．1 文字 (8 ビット) は 2 桁の 16 進数で表現される．メタ文字を含まない(注1)パターンの一部の文字列を**サブパターン**といい, サブパターンの一部を**フラグメント**, パターン中の文字の個数を**パターンの長さ**, サブパターンの出現位置を**ロケーション**, そして, サブパターン間のロケーションの差の絶対値を**距離**と定義する．本論文ではパターン数を k で表す．表 2 に ClamAV の正規表現のメタ文字を示す．
[例 2.1]　表 3 に ClamAV のパターンの例を示す．W32.Gop では, “736D74702E79656168” と “2D20474554204F49” がサブパターンである．■

表 2　ClamAV の正規表現のメタ文字

表記	意味	例
??	任意の 1 文字 (バイト)	
*	0 文字以上の任意の文字	AA*BB={AABB,AA??BB,AA????BB,···}
(AA|BB)	文字の集合	(AA|BB)={AA,BB}
{n-m}	n 文字以上 m 文字以下	AA{1-2}BB={AA??BB,AA????BB}

表 3　パターンの例

ウイルス名	パターン
Trojan.DelY-3	64656C74726565{-1}2F(59|79)20633A5C2A2E2A
Trojan.MkDir.B	406D64202572616E646F6D25????676F746F2048
W32.Gop	736D74702E79656168*2D20474554204F49
Worm.Bagle-67	68404840486886D5B0090EB01EbEB0A5BA9ED46

2.4　2 段階マッチングを用いたウイルス検出エンジン

ClamAV のパターンは, サブパターンと距離を表すメタ文字で構成される．本論文では, 2 段階マッチングを用いてパターンの検出を行う．第一段階で厳密マッチングを行ってパターンの一部であるフラグメントを検出する．フラグメントはメタ文字を持たないため, 単純なハードウェアで実現できる．フラグメントが検出された場合, パターンが存在する可能性があるので, 第二段階で正規表現マッチングを行って, パターンを厳密にチェックする．正規表現マッチングは複雑であるため, 正規表現ライブラリ (ソフトウェア) で実現する．

第二段階でのマッチングは第一段階でフラグメントが検出された場合にのみ行う．つまり, 第一段階は第二段階の検索箇所

(注1)：ただし, メタ文字 “??” は許容する.

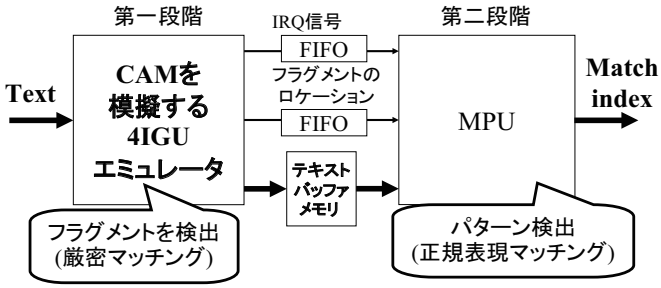

図 4　2 段階マッチングを用いたウイルス検出エンジン.

表 4　インデックス生成関数の例.

x_1	x_2	x_3	x_4	x_5	x_6	f
0	0	0	0	1	0	1
0	1	0	0	1	0	2
0	0	1	0	1	0	3
0	0	1	1	1	0	4
0	0	0	0	0	1	5
1	1	1	0	1	1	6
0	1	0	1	1	1	7

をフィルタリングしているといえる. 第二段階で行う正規表現マッチングは, 複雑な処理を行うため低速であるが, フィルタリングが十分であると仮定すれば検索対象を削減できるため, 第二段階での処理の遅さを隠蔽できる. この仮定が成立するならば, システムの処理速度は第一段階の処理速度に依存する. 本論文では, 第二段階の処理がボトルネックにならないように第一段階のフィルタを設計する.

[例 2.2]　図 3 に 2 段階マッチングの例を示す. ロケーション 3 でサブパターンを検出する (図 3(1)). 正規表現マッチングを行う (図 3(2)). ロケーション 6 でサブパターンを検出する (図 3(3)). 正規表現マッチングを行い, パターンを検出する (図 3(4)). ■

図 4 に 2 段階マッチングを用いたウイルス検出エンジンを示す. フラグメント検出は CAM を模擬する **4IGU エミュレータ** (第 3 章で述べる) で行い, パターンの検出は正規表現ライブラリ (Perl Compatible Regular Expression: PCRE) [14] を用いた組込みプロセッサ (MPU) で行う. フラグメントが検出された場合, **IRQ 信号**とフラグメントのロケーションが MPU に送られる. MPU の処理中に別のフラグメントが検出される場合が考えられるので, IRQ 信号とロケーションを保持する FIFO を用意する. また, テキストを保持する**テキストバッファメモリ**も用意する.

3. 4IGU エミュレータを用いたフラグメントの検出

3.1 インデックス生成関数

[定義 3.1]　k 個の異なる登録ベクトルに対して 1 から k までの固有のインデックスを対応させた表を, **インデックス表**という [19].

[定義 3.2]　$B = \{0,1\}$ とする. 関数 $f(\vec{X}) : B^n \to \{0,1,\ldots,k\}$ において k 個の異なる登録ベクトル $\vec{a}_i \in B^n$ $(i = 1,2,\ldots,k)$ に対して, $f(\vec{a}_i) = i$ $(i = 1,2,\ldots,k)$ が成立し, それ以外の $(2^n - k)$ 個の入力ベクトルに対しては, $f = 0$ が成立するとき, $f(\vec{X})$ を**重み k のインデックス生成関数**という. インデックス生成関数は, k 個の異なる 2 値ベクトルに対して, 1 から k までのアドレス (インデックス) を生成する.

[例 3.3]　図 4 にインデックス生成関数の例を示す. ■

登録ベクトルは長さ m のフラグメントに対応する. インデックス表は CAM [8] で直接実現できるが, CAM は消費電力が大きく高価である. また, FPGA 上に CAM 機能を実現するには大量の論理素子が必要であり, 消費電力も高い. 本論文ではメモリを用いてインデックス生成関数を実現する.

3.2 有限入力メモリ機械

厳密マッチングは図 5 に示す**有限入力メモリ機械 (FIMM: Finite Input Memory Machine)** で実現できる [10]. 長さ m の登録ベクトルを k 個格納する FIMM を実現する回路を図 5 に示す. メモリは各状態に対する出力関数を実現する. FIMM は連接閉包 "*" を受理できないため受理能力が制限されているが, 単純な回路で実現できる.

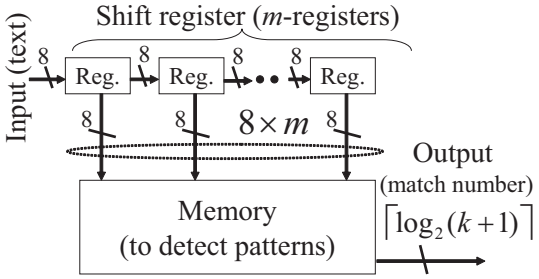

図 5　FIMM.

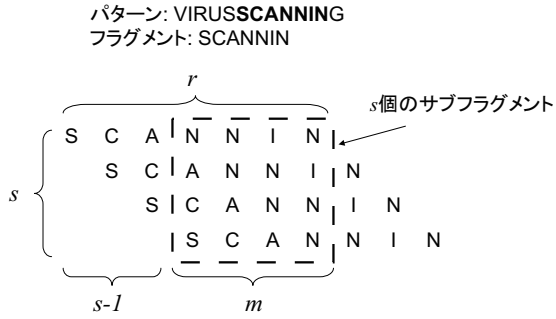

図 6　見逃しがない s-FIMM.

3.3 s-FIMM

図 5 に示した FIMM は, 1 クロック毎にテキストを 1 文字シフトしながらマッチングを行う. s 個の FIMM を用いるとスループットを s 倍にできる. FIMM のメモリは FPGA の組込みメモリでは容量不足であるので, 外付けの大容量メモリが必要である. s 個の FIMM に対して s 個の外付けメモリが必要となるが, FPGA の高速メモリ用ピン数には制限があるため, s を増加させるのは困難である. 本論文では, メモリ量を増加させてスループットを向上する.

[定義 3.3]　1 クロック毎に s 文字シフトする FIMM を **s-FIMM** という. s-FIMM には, フラグメントの一部である**サブフラグメント**を格納する.

s-FIMM は s 文字シフトしたフラグメントを並列に処理するため, 1-FIMM と比較してスループットを s 倍向上できる. このとき, 元のフラグメントとフラグメントを 1 文字から $s-1$ 文字までシフトした s 通りの出現が考えられるため, 1-FIMM に格納するフラグメントのみでは見逃しが発生してしまう. 見逃しを回避するために 1 文字から $s-1$ 文字までフラグメントをシフトし, それぞれから m 文字のサブフラグメントを抽出し s-FIMM のメモリに格納する.

[例 3.4]　パターンを "VIRUSSCANNING" とする. 図 6 に 4-FIMM の例を示す. フラグメントの出現は 4 通り考えられる. 4 つのサブフラグメント ("NNIN","ANNI","CANN","SCAN") を格納することで, 見逃しがない 4-FIMM を実現できる. ■

[例 3.5]　図 7 にサブフラグメント長 m を 4 とした場合の, 1-FIMM, 2-FIMM, 4-FIMM を示す. s を 2,4 と増やすと, 格納サブフラグメント数が $2k$, $4k$ と増えるが, スループットが 2 倍,4 倍と向上する. ■

3.4 サブフラグメント長 m の決定

例 3.5 に示すように, s-FIMM は長さ r の k 個のフラグメントに対して長さ m の ks 個のサブフラグメントを格納する. このとき, 関係

$$s = r + 1 - m \tag{1}$$

が成立する. s-FIMM では, 1-FIMM と比較して格納サブフラグメント数が s 倍に増加し, フラグメント長が $s-1$ 文字伸びる. つまり, s-FIMM はメモリ量を増やすことで最大 $r+1-m$ 倍スループットを向上できる (注2). ClamAV では, MD5 チェックサムパターン (16 文字, 128 ビット) が大部分を占めるため, r の最大値は 16 である. よって, s は m によって決まるため, 適切な m を決めなければならない. m が大きい場合, サブフラグメントのマッチ確率が減少し, MPU への IRQ 信号の発生は稀

(注2)：実装結果より, メモリアクセスがボトルネックとなるため, s を増加させてもこの仮定は成立する.

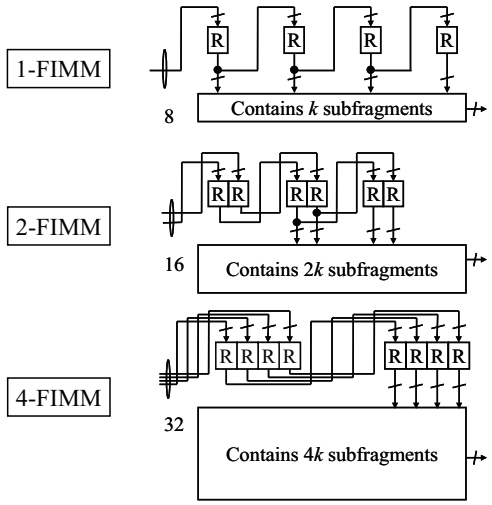

図 7　s-FIMM の例 ($m = 4$ の場合,).

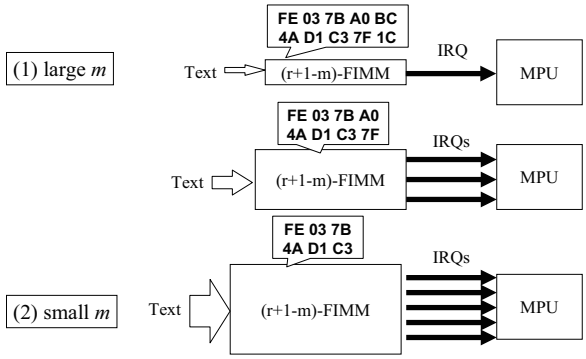

図 8　サブフラグメント長 m と IRQ 信号発生間隔の関係 (r は一定).

有である. しかし, 式 (1) より s が小さくなるため, スループットも低下してしまう (図 8(1)). 一方, m が小さい場合, 式 (1) から s が大きくなり, スループットが向上する. しかし, サブフラグメントのマッチ確率が増加し, MPU への IRQ 信号が頻繁に発生する (図 8(2)). IRQ 信号の発生間隔が MPU の処理時間を超える場合, s-FIMM の処理を停止しなければならず, システムのスループットが低下してしまう. 従って, MPU を停止させないサブフラグメント長 m の最小値を求める問題を解かなければならない.

[問題 3.1]　s-FIMM に格納するサブフラグメント長を m, 第二段階の MPU の平均処理時間を T_{MPU}, m に対するサブフラグメントの平均検出確率を $P(m)$, s-FIMM のサブフラグメント 1 個当りの検出時間を T_{s-FIMM} とする. 条件 $\frac{T_{s-FIMM}}{P(m)} \ll T_{MPU}$ を満たす最小の m を求めよ.
m は第 5 章で実験的に求める.

3.5　インデックス生成回路 (IGU)

サブフラグメント長を m, サブフラグメント数を sk とする. s-FIMM の出力関数を実現するために必要なメモリ量 (注3) は

$$M_{FIMM} = 2^{8m}\lceil log_2(sk+1)\rceil \qquad (2)$$

となり m が大きい場合, 実現できない. 本論文ではインデックス生成回路を用いて s-FIMM の出力関数を実現する.

$X_1 = (x_1, x_2, \ldots, x_p)$ を $Y_1 = (y_1, y_2, \ldots, y_p)$ に置き換えた関数を $\hat{f}(Y_1, X_2)$ とする. ただし, $y_i = x_i \oplus x_j$, $x_i \in \{X_1\}$, $x_j \in \{X_2\}$, $p \geq \lceil \log_2(sk+1)\rceil$ である.

[例 3.6]　例 3.3 に示したインデックス生成関数の**分解表**を表 5 に示す. 列ラベルは $X_1 = (x_1, x_2, x_3)$ を表し, 行ラベルは $X_2 = (x_4, x_5, x_6)$ を表す. 表の値は関数値を表す. $Y_1 = (x_1 \oplus x_6, x_2 \oplus x_5, x_3 \oplus x_4)$ と変数変換を行った場合の $\hat{f}(Y_1, X_2)$ の分解表を表 6 に示す. 列ラベルは Y_1 を示し, 行ラベルは X_2 を示す. f の分解表では非零要素を 2 つ以上持つ列が 3 つであるのに対し, $\hat{f}$ では非零要素を 2 つ以上持つ列が 1 つに減少している. ■

(注3)：状態遷移を記憶するシフトレジスタのビット数は出力関数を記憶するメモリのビット数よりも遥かに小さいので無視できる. よって本論文では**メモリ量**とは FIMM の出力関数を記憶するメモリのビット数を表すものとする.

表 5　$f(X_1, X_2)$ の分解表.

	0	0	0	0	1	1	1	1	x_3
	0	0	1	1	0	0	1	1	x_2
	0	1	0	1	0	1	0	1	x_1
000	0	0	0	0	0	0	0	0	
001	0	0	0	0	0	0	0	0	
010	1	0	2	0	3	0	0	0	
011	0	0	0	0	4	0	0	0	
100	5	0	0	0	0	0	0	0	
101	0	0	0	0	0	0	0	0	
110	0	0	0	0	0	0	0	6	
111	0	0	7	0	0	0	0	0	
x_6, x_5, x_4									

表 6　$\hat{f}(Y_1, X_2)$ の分解表.

	0	0	0	0	1	1	1	1	y_3
	0	0	1	1	0	0	1	1	y_2
	0	1	0	1	0	1	0	1	y_1
000	0	0	0	0	0	0	0	0	
001	0	0	0	0	0	0	0	0	
010	2	0	1	0	0	0	3	0	
011	0	0	4	0	0	0	0	0	
100	0	5	0	0	0	0	0	0	
101	0	0	0	0	0	0	0	0	
110	0	0	0	0	6	0	0	0	
111	0	0	0	0	0	7	0	0	
x_6, x_5, x_4									

表 7　$\hat{f}_1(Y_1, X_2)$ の分解表.

	0	0	0	0	1	1	1	1	y_3
	0	0	1	1	0	0	1	1	y_2
	0	1	0	1	0	1	0	1	y_1
000	0	0	0	0	0	0	0	0	
001	0	0	0	0	0	0	0	0	
010	2	0	1	0	0	0	3	0	
011	0	0	0	0	0	0	0	0	
100	0	5	0	0	0	0	0	0	
101	0	0	0	0	0	0	0	0	
110	0	0	0	0	6	0	0	0	
111	0	0	0	0	0	7	0	0	
x_6, x_5, x_4									

表 8　$\hat{f}_1(Y_1)$ の主メモリ.

y_3	0	0	0	0	1	1	1	1
y_2	0	0	1	1	0	0	1	1
y_1	0	1	0	1	0	1	0	1
$\hat{f}_1$	2	5	1	0	6	7	3	0

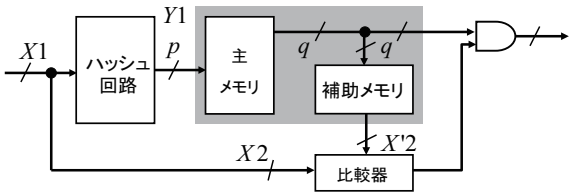

図 9　インデックス生成回路 (IGU).

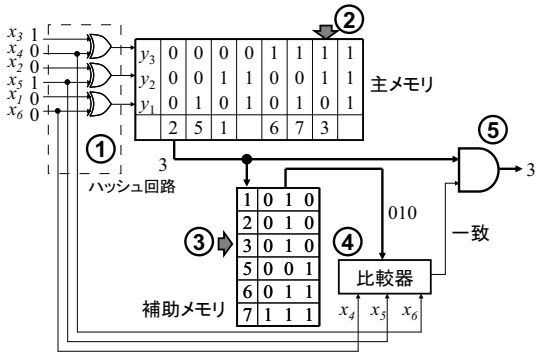

図 10　IGU の例.

[例 3.7]　図 10 に IGU を用いたベクトル検出の例を示す. まず, ハッシュ回路で変数を選択する (図 10 (1)). 次に, 主メモ

表 9　4IGU に登録されるサブフラグメント数の推定値と実験値.

	推定値			実験値		
	p	格納数	残り	p	格納数	残り
IGU_1	21	963,815	326,802	21	953,221	337,396
IGU_2	21	302,611	24,191	21	311,943	25,453
IGU_3	21	24,052	139	21	25,276	177
IGU_4	15	139	0	11	177	0

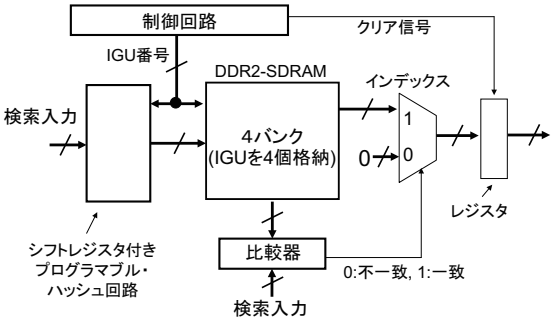

図 11　4IGU エミュレータ.

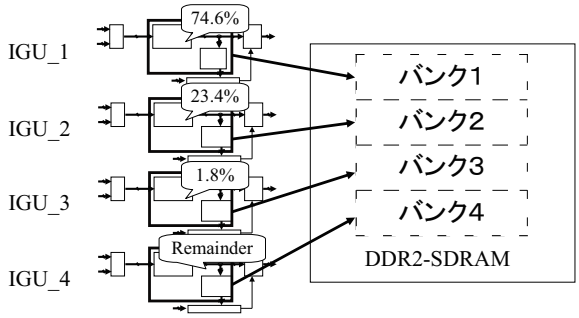

図 12　4IGU の主メモリを DDR2SDRAM へ格納.

リからインデックスを読み出し (図 10 (2)), 補助メモリから対応する変数を読み出す (図 10 (3)). そして, 比較器で入力と一致するか比較を行い (図 10 (4)), 一致しているので, インデックスを出力する (図 10 (5)). ■

3.6　IGU で実現可能な登録ベクトルの割合

主メモリの入力数 p に対する登録ベクトルの格納率が知られている.

[定理 3.1]　[16] 重み k のインデックス生成関数において, IGU の主メモリに格納される登録ベクトルの割合は $\delta \simeq \frac{1-e^{-\xi}}{\xi}$ である. ただし, 主メモリの入力数を p とすると $k \leqq 2^p$, $\xi = \frac{k}{2^p}$ であり, インデックス生成関数の分解表において, 非零要素は一様に分布しているものと仮定する.

[例 3.8]　$\frac{k}{2^p} = \frac{1}{1}$ とすると $\delta = 1 - e^{-1} \simeq 0.632$ である. このとき, 主メモリは登録ベクトルの 63.2%を格納できる. ただし, 登録ベクトルを一様に分布させるためにハッシュ回路が必要である. ■

主メモリの入力数を増やしてアドレス空間を十分に広くした場合, 登録ベクトルをほぼ全て格納可能であることが経験的に知られている.

[推論 3.1]　[17] 重み k のインデックス生成関数を実現するために必要な主メモリの入力数 p は高々$p = 2\lceil log_2(k+1) \rceil - 1$ である. ただし, インデックス生成関数の分解表において, 非零要素は一様に分布しているものとする.

3.7　インデックス生成回路を 4 台用いた実現 [16]

定理 3.1 と推論 3.1 を用いると, 入力数 p の主メモリに格納可能な登録ベクトル数 k を推定可能である. 4 個の IGU (**4IGU**) を用いて登録ベクトルを全て格納する手法が提案されている.

[例 3.9]　p を主メモリの入力数とし, $k = 1,290,617$ 個のサブフラグメントを 4IGU に格納し, 推定値との比較を行った結果を表 9 に示す. 表 9 の推定値は, IGU_1, IGU_2, IGU_3 に関しては定理 3.1 を, IGU_4 に関しては推論 3.1 から求めた. 表 9 から, 推定値は実験値とほぼ一致している. サブフラグメント数 sk が与えられれば, 全てのサブパターンを格納するのに必要な主メモリの入力数 p を推定できる. ■

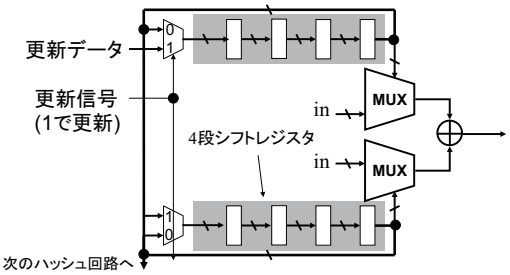

図 13　シフトレジスタ付きプログラマブル・ハッシュ回路.

3.8　4IGU エミュレータ

IGU を 4 個用いることで, フラグメントを全て格納できるが, 4 個の外付けメモリが必要となる. FPGA に取り付けることができるメモリの個数には制約があるため(注4), 1 つのメモリを繰り返しアクセスすることで 4 個の IGU をエミュレーションする **4IGU エミュレータ**を提案する. 図 11 に 4IGU エミュレータを示す. 4IGU エミュレータは DDR2-SDRAM の各バンクに 4 個の IGU のメモリ部を格納する (図 12). 図 13 に示す**シフトレジスタ付きプログラマブル・ハッシュ回路**は, 各 IGU のハッシュ回路を模擬する. シフトレジスタで選択変数を切り替える. また, 選択変数を更新できる. 制御回路で 4IGU エミュレータを制御する. データシートより, DDR2-SDRAM (266MHz 動作, DDR2-533, CL=4, 16 ビット ×8 バースト) は 4 バンクを切り替えることで, 4 クロック毎にプリチャージ付きデータ読出しができる [11]. 従って, 16 クロック毎に 4 個の IGU を模擬できる. s-FIMM は $8s$ ビットデータをシフトするので, スループットは以下の式となる.

$$\frac{0.266 \times 8s}{16} = 0.133s[\text{Gbps}] \tag{3}$$

ClamAV (version.0.96.5) では, 大部分のウイルスパターンは MD5 checksum の 16 文字 (128 ビット) で構成されるため(注5), $r \leqq 16$ を考える. また, 1-FIMM に基づく 2 段階マッチング回路の実装結果から, サブフラグメント数が $k = 500,000$ 程度の場合, $m = 4$ を選べばよいことが知られている [12]. 本論文では, 100 万個以上格納するので $m = 5$, $m = 6$ を検討する. 式 (1) より, $m = 4$ のとき s の最大値は 13, $m = 5$ のとき s の最大値は 12, そして $m = 6$ のとき s の最大値は 11 である. フラグメント数 $k = 1,290,617$ とし, $m = 4, 5, 6$ のときの s-FIMM のメモリ量を図 14 に, スループットを図 15 に示す. メモリ量は, 16 ビット ×4 バースト (64 ビット) を用いるので, $m = 4$ (32 ビット) から $m = 6$ (48 ビット) まで変化させても s のみに依存する. スループットは式 (3) から求めた.

[例 3.10]　フラグメント数 $k = 1,290,617$, $s = 1$ のとき, 4IGU エミュレータの必要メモリ量を求める. メモリ量は, 16 ビット ×4 バースト (64 ビット) を用いるので, $m = 4$ (32 ビット) から $m = 6$ (48 ビット) まで変化させても s のみに依存する. 定理 3.1 より, 主メモリの入力数は $p = 21$ である. 4 個の IGU を格納するので $2^{21} \times 64\ (bits) = \frac{2^{21} \times 64 \times 4}{8 \times 2^{20}} = 64$ (MBytes) となる. ■

4.　実 験 結 果

4.1　適切なサブフラグメント長 m の決定

問題 3.1 から, 4IGU エミュレータで検出するサブフラグメント長を m を実験的に求めた. サブフラグメントの平均マッチ確率 $P(m)$ に関して, サブフラグメント長 m を検出する s-FIMM を模擬する 4IGU エミュレータのサイクルベース・シミュレータを C 言語で設計し, 2,963 個の cygwin の実行ファイルのスキャンを行った結果を用いた. 4IGU エミュレータは 266 MHz で動作する DDR2-SDRAM を用いて 16 クロックでサブフラグメントをチェックするので, $T_{s-FIMM} = \frac{16}{0.266}$ とした. T_{MPU} に関して, Xilinx 社の組込み MPU である MicroBlaze [22] を

(注4)：FPGA の高速メモリインタフェースのピン配置と本数には制約がある. ピン数の多い FPGA は高価である.

(注5)：Basic pattern では 16 文字を超えるので 16 文字のフラグメントを取り出す. 検出確率を下げるため, なるべく他のフラグメントと重複しないようにする.

表 10　他の手法との比較.

	#Pattern	#Char	#LC	Mem [Bytes]	Th [Gbps]	#LC/ #Char	Mem/ #Char	Off-Chip Devices
USC RegExpController (2006) [2]	1,316	16,715	41,787	768819.2	1.40	2.4999	45.9957	SDRAM
Cuckoo Hashing (2007) [21]	4,748	68,266	2,982	142848.0	2.20	0.0436	2.0925	Unknown
PERG-Rx (2009) [6]	85,625	8,645,488	42,809	387072.0	1.30	0.0049	0.0447	SSRAM
Proposed Method	1,290,617	42,461,299	13,857	39116.8	3.19	0.0003	0.0009	DDR2-SDRAM × 3

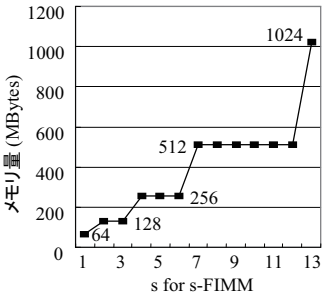

図 14　s-FIMM のメモリ量.

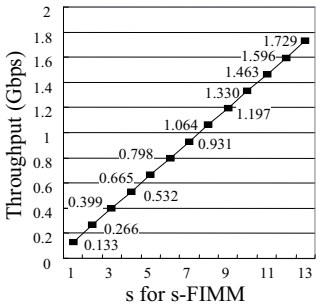

図 15　s-FIMM のスループット.

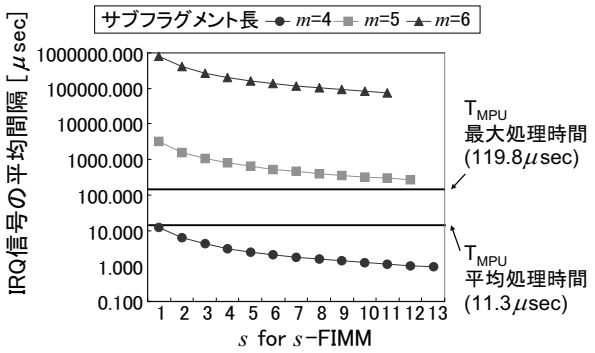

図 16　IRQ 信号の平均発生間隔と MPU の処理時間.

120 MHz で動作させて処理速度の最大値と平均値を用いた. なお, ClamAV のパターンから PCRE ライブラリを用いて MicroBlaze の実行コードに変換し, DDR2-SDRAM (266MHz) に格納している. 図 16 に s-FIMM とサブフラグメント長 m に対する IRQ の平均発生間隔 $\frac{T_{s-FIMM}}{P(m)}$ と MPU の処理時間 T_{MPU} の最大値と平均値を示す. 図 16 より, m=5 の場合, MPU の平均処理時間は IRQ の平均発生間隔よりも短いため, MPU は停止しないと考えられる. このとき, s の最大値は 12 である. 従って図 14 より必要なメモリ量は 128MBytes, 図 15 よりスループットは 1.596 Gbps となる.

4.2 実装結果

提案ウイルス検出エンジンを Inrevium 社 PCI Express 評価ボード (FPGA: Xilinx 社 Virtex5 VLX50T-GB-R) に実装した. 評価ボードは 266 MHz で動作する 512 MBytes の DDR2-SDRAM を 2 個, 512 MBytes の SO-DIMM モジュールを備えているので, 2 個の DDR2-SDRAM を 4IGU エミュレータに, 残り 1 個の DDR2-SDRAM を MicroBlaze に用いた. 合成ツールは Xilinx 社 ISE Design Suite 11.1 を用いた. 実装結果より, 4IGU エミュレータは 6,279 個の LC と 1 個の BRAM を消費し, 最大動作周波数は 312.9 MHz; MicroBlaze は 1,263 個の LC を消費し, 最大動作周波数は 131.2 MHz; 3 個の DDR2-SDRAM コントローラは, 合計で 6,324 個の LC と 9 個の BRAM を使用し, 最大動作周波数は 266.0 MHz; そして, テキストバッファメモリ用に 10 個の BRAM を消費した. 従って, 提案ウイルス検出エンジンは 13,857 個の LC と 20 個の BRAM を消費した. 提案エンジンは 4IGU エミュレータの DDR2-SDRAM コントローラの処理がボトルネックとなるので, 最大動作周波数は 266.0 MHz である. 実装では 2 個の DDR2-SDRAM を用いて 4IGU エミュレータのスループットを 2 倍にした. よって, 提案エンジンのスループットは 3.192 Gbps である.

提案エンジンと他の手法との比較を表 10 に示す. 表 10 より, 提案エンジンはスループットを 1.45 倍, LC 利用率で 16.3 倍, そして組込みメモリ利用率で 49.6 倍改善できた.

5. まとめとコメント

本論文では, 2 段階マッチングに基づくウイルス検出エンジンを提案した. 第一段階では, 4 個の IGU を模擬する 4IGU エミュレータを用いてパターンの一部を検出する. 第二段階では, 第一段階で検出されたパターンの一部がパターンと完全に一致するか FPGA の組込みプロセッサを用いて検出する. ClamAV のパターン 1,290,808 個を Xilinx 社の FPGA と 3 個の DDR2-SDRAM を用いて実装し, 他の手法と比較を行った結果, スループットで 1.45 倍, LC 利用率で 16.3 倍, 組込みメモリ利用率で 49.6 倍優れていた.

実装システムは, 大量のサブパターンが頻繁にマッチするように意図したテキストを投げる攻撃 (パフォーマンス攻撃) に弱い. Kumar らは二段階マッチングのパフォーマンス攻撃に耐性を持たせる手法を提案している [9]. 第一段階部と第二段階部の間にカウンタをとりつけ, 閾値を超えればパフォーマンス攻撃とみなす. Kumar の手法は我々の手法に容易に適用できる.

6. 謝　辞

本研究は, 一部, 日本学術振興会・科学研究費補助金, および, 文部科学省・知的クラスター創成事業 (第二期) の補助金による.

文　献

[1] M. Alicherry, M. Muthuprasanna, and V. Kumar, "High speed pattern matching for network IDS/IPS," *ICNP'06*, 2006, pp.187-196.
[2] Z. K. Baker, H. Jung, and V. K. Prasanna, "Regular expression software deceleration for intrusion detection systems," *FPL'06*, 2006, pp. 28-30.
[3] CAST inc., "MD5 IP Core," http://www.cast-inc.com/ip-cores/encryption/md5/
[4] ClamAV, http://www.clamav.net/
[5] Google, "Google Safe Browsing API," http://code.google.com/intl/ja/apis/safebrowsing/
[6] J. T. L. Ho and G. G. F. Lemieux, "PERG-Rx: A hardware pattern-matching engine supporting limited regular expressions," *FPGA 2009*.
[7] Kaspersky, http://www.kaspersky.com/
[8] T. Kohonen, *Content-Addressable Memories*, Springer Series in Information Sciences, Vol. 1, Springer Berlin Heidelberg, 1987.
[9] S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese, "Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia," *3rd ANCS'07*, 2007, pp. 155-164.
[10] Z. Kohavi, *Switching and Finite Automata Theory, McGraw-Hill Inc.*, 1979.
[11] Data Sheet: DDR2-SDRAM, http://www.micron.com/
[12] H. Nakahara, T. Sasao, M. Matsuura, Y. Kawamura, "The parallel sieve method for a virus scanning engine," *DSD'09*, 2009, pp.809-816.
[13] H. Nan, S. Haoyu, T. V. Lakshman, "Variable-stride multi-pattern matching for scalable deep packet inspection," *INFOCOM'09*, 2009, pp.415-423.
[14] PCRE: http://www.pcre.org/
[15] H. C. Roan, W. J. Hawang, and C. T. Dan Lo., "Shift-or circuit for efficient network intrusion detection pattern matching," *FPL'06*, 2006, pp.785-790.
[16] T. Sasao, M. Matsuura and H. Nakahara, "A realization of index generation functions using modules of uniform sizes," *IWLS'10*, June 18-20, 2010, pp.201-208.
[17] T. Sasao, "On the number of variables to represent sparse logic functions," *ICCAD'08*, Nov.10-13, 2008, pp. 45-51.
[18] T. Sasao and M. Matsuura, "An implementation of an address generator using hash memories," *DSD'07*, Aug. 27 - 31, 2007, pp.69-76.
[19] T. Sasao, "Design methods for multiple-valued input address generators," *ISMVL'06*(invited paper), May 17-20, 2006, pp.102-109.
[20] L. Tan, and T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," *ISCA'05*, 2005, pp.112-122.
[21] T. N. Thinh, S. Kittitornkun, and S. Tomiyama, "Applying cuckoo hashing for FPGA-based pattern matching in NIDS/NIPS," *ICFPT'07*, 2007, pp.121-128.
[22] Xilinx inc, "MicroBlaze", http://www.xilinx.com/
[23] F. Yu, R. H. Katz, and T. V. Lakshman, "Gigabit rate packet pattern matching using TCAM," *ICNP'04*, 2004, pp.174-183.