一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CPSY2018-111,DC2018-93（2019-03）

FPGA NICを用いた多次元変化点検出の高性能化

岩田　拓真†　　松谷　宏紀†

† 慶應義塾大学大学院 理工学研究科　〒 223-8522 神奈川県横浜市港北区日吉 3-14-1
E-mail: †{iwata,matutani}@arc.ics.keio.ac.jp

あらまし　IoT 機器の広まりなどによって、より多くの通信データを効率よく処理することが求められるようになっている。変化点検出は時系列データから通常とは異なる変動パターンを検出する異常検知の一種であり、通信セキュリティや金融等の分野で用いられている。本研究ではオンライン動作可能な変化点検出アルゴリズムである Changefinder を NIC として機能する FPGA にオフロードし、より高い性能をだすことを目指す。前研究では一次元のデータのみに対応した変化点検出を実装したが、一次元のデータしか対応できないことで、用途が非常に限られてしまうことを解決するため Changefinder コアとパケット処理を多次元のデータへも対応できるように拡張を行った。評価では次元数を変えながら実装した多次元対応 Changefinderk コアの面積とスループットを調べ、実装可能な次元数を調べた。

キーワード　FPGA、変化点検出、NIC

Accelerating Multidimensional Change-Point Detection using FPGA NIC

Takuma IWATA† and Hiroki MATSUTANI†

† Graduate School of Science and Technology, Keio University 3-14-1, Hiyoshi, Yokohama, JAPAN 223-8522
E-mail: †{iwata,matutani}@arc.ics.keio.ac.jp

1. はじめに

近年、データ利活用の広まりやインターネットに接続する IoT (Internet of Things) の製品により、処理しなければならないデータ量が増大し続けている。このようなデータ量の増加に伴い、膨大なデータから通常と異なる部分を検出する異常検知が注目されている。

異常検出の一種である変化点検出は、時系列データに対してデータの大小ではなく、急激な時系列パターンの変わり目である変化点を検出する手法である。変化点検出の対象アプリケーションとしては、ウィルスによるトラフィック量の急増を検知するセキュリティ分野や金融、トレンド解析などがある。パターンの変わり目は確率分布の変化として捉えることができるので、変化点の候補点の前後で別々の確率分布モデルを作ることで変化点検出することができる。

そのような変化点検出アルゴリズムの一つに Change Finder [1] がある。Change Finder アルゴリズムは変化点検出と外れ値検出を同時に行うアルゴリズムである。オンラインで変化点検出が可能なアルゴリズムであり、AR モデル (Auto Regression model) を時系列モデルとして使いつつ、モデルの学習に忘却効果を取り入れることにより、非定常なモデルの学習を実現している。

Change Finder アルゴリズムはオンライン上で動作させることの出来る実用的なアルゴリズムであるが、ネットワークの異常検知等ではより高い帯域が求められることが考えられるため、FPGA(Field-Programmable Gate Array) による高性能化が行われている [2]。

文献 [2] では一次元の浮動小数点数のデータのみに対する変化点検出を行っていた。しかし実際の検出対象は多くの要素からなることもあり、一次元のデータしか扱えないことでアプリケーションが大きく制限されてしまうことが考えられる。

そこで本論文では多次元に対応した ChangeFinder コアを高位合成で実装し、NIC (Network Interface Card) 内ハードウェアへのオフロードを行った。具体的には一次元データに対応した ChangeFinder NIC を元に多次元への対応をするため学習モジュールの変更とパケット処理部の変更を行った。評価では現状の実装での面積評価とスループット評価を行い、対応可能な次元数についての検討をする。

本論文の構成は以下の通りである。2. 章で関連研究を紹介する。3. 章で多次元対応 Changefinder コアの設計について述べる。4. 章で本論文の実装を示す。5. 章では、提案手法のシミュレーションによる性能を示し、6. 章で本論文をまとめ、今後の課題について述べる。

2. 関 連 研 究

2.1 変化点検出

通常のパターンから外れたものを検出する異常検出はセキュリティや金融、工場整備等の様々な分野で広く用いられている。異常検知はその検出対象から外れ値検出、変化点検出、異常行動検出にわけることができる。

本論文ではこのうち変化点検出を主に扱う。変化点検出とは時系列データに対する異常検知であり、変動パターンの変化するものであり、統計的には変化点検出はその点の前後における確率分布の変化を検知することだといえる。統計的手法をとるものの例として以下に時系列モデルに AR(Auto Regression) モデルを用い、データ内に最大一個の変化点が存在すると仮定した場合の変化点検出アルゴリズムを示す。

$x_1^n = x_1, ..., x_n$ を時系列データとする。これを時間 t において $x_1^t = x_1, ..., x_t$ and $x_{t+1}^n = x_{t+1}, ..., x_n$ に分け、それぞれ x_1^t と x_{t+1}^n と表すことにする。

x_t のモデルとして k 次の AR モデルを考えた場合、その確率密度関数 $p(x_t|x_{t-k}^{t-1})$ は以下に示すようになる。

$$p(x_t|x_{t-k}^{t-1}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{(x_t-\omega_t)^T\Sigma^{-1}(x_t-\omega_t)}{2}\right], \quad (1)$$

ここで d と Σ はデータの次元数と共分散行列をそれぞれ表し、ω_t はモデルから推定された x_t の値を表す。ω_t は AR モデルに従って以下のように求められる。

$$\omega_t = \sum_{i=1}^{k} a_i(x_{t-i} - \mu) + \mu, \quad (2)$$

ここで $a_1, ..., a_k$ と μ は AR モデルのパラメータである。データを t で二つに分けた場合とそうでない場合でそれぞれ式 2 を用いて最尤推定を行うことでこれらのモデルのパラメータを求め、モデルに対する当てはめ誤差 I を求める。

$$I(x_1^n) = \sum_{t=1}^{n} ||x_t - \hat{\omega_t}||^2 \quad (3)$$

$$I(x_1^t) + I(x_{t+1}^n) = \sum_{t=1}^{t} ||x_t - \hat{\omega_t}||^2 + \sum_{t=t+1}^{n} ||x_t - \hat{\omega_t}||^2 \quad (4)$$

$I(x_1^t) + I(x_{t+1}^n)$ が最小となる時間 t において、二つの誤差の差 $I(x_1^n) - I(x_1^t) + I(x_{t+1}^n)$ が閾値より小さかった場合は分割してモデルを構築した方がいいと判断し、t を変化点とする。

この手法は未知の変化も検出できるが以下のような欠点がある。

- 定常モデルしか対応できない
- 計算量が $O(n^2)$ と大きい
- 変化点の個数は既知でなければならない

本論文ではオンラインで高速に動作可能な変化点検出を目指すため、アルゴリズムは非定常なモデルに対応でき、計算量が少ないことが必要である。そのため本論文では上記の AR モデルを使った手法を拡張した ChangeFinder [3] をアルゴリズムとして採用し、実装した。この ChangeFinder について次節で説明する。

2.2 ChangeFinder Algorithm

ChangeFinder は時系列データの学習モデルとして SDAR (Sequentially Discounting Auto-Regression model learning) を採用したものであり、オンラインで外れ値検出と変化点検出を同時に行えるアルゴリズムである [3]。SDAR の詳細については次節で述べる。

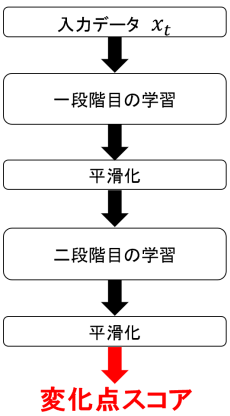

図 1 ChangeFinder のフローチャート

図 1 に ChangeFinder における処理の流れを示す。ChangeFinder は二段階の学習を行い、一回目の出力を外れ値スコア、二回目の出力を変化点スコアとすることで同時に二種類の異常を検知することが可能になっている。

各処理の詳細を以下で説明する。

Step 1 (データ入力) 時刻 t において新たなデータ x_t を受け取る。

Step 2 (一段階目の学習) それぞれの t に対して SDAR モデルを構築し、確率密度関数 $p_t(x) : t = 1, 2, ...$ を得る。$p_t(x) : t = 1, 2, ...$ の対数損失をとって外れ値スコアとする。このスコアが高いほど t が外れ値である可能性が高いことを示す。

$$Score(x_t) = -\log p_{t-1}(x_t) \quad (5)$$

Step 3 (一段階目の平滑化) それぞれの t に対して、外れ値スコアの移動平均 $y_t : t = 0, 1, 2...$ をとる。

$$y_t = \frac{1}{T}\sum_{i=t-T+1}^{t} Score(x_i), \quad (6)$$

ここで T はウィンドウ幅であり、ユーザが決定する。

Steps 4 & 5 (二段階目の学習と平滑化) それぞれの t に対して $y_t : t = 0, 1, 2, ...$ の SDAR モデルを構築し、新たな確率密度関数 $q_t(x) : t = 1, 2, ...$ を得る。一段階目と同様に対数損失の移動平均をとり、変化点スコア $z_t : t = 0, 1, 2, ...$ を得る。このスコアが高いほど t が変化点である可能性が高いことを示す。

$$z_t = \frac{1}{T}\sum_{i=t-T+1}^{t} (-\ln q_{t-1}(y_t)) \quad (7)$$

図 2 に示すように外れ値スコアの高い点には外れ値と変化点が含まれ、変化点スコアの高い点には変化点のみが含まれる。二つの結果を用いることで外れ値と変化点をそれぞれ検出可能である。

2.2.1 SDAR Model

SDAR は AR モデルに忘却を導入することにより、オンライン動作可能にした学習モデルである。ChangeFinder では確率密度関数 $p_t(x)$ と $q_t(x)$ を決定するために必要な AR モデルの係数 a_t と共分散行列 Σ を求めるのに SDAR モデルを使用している。

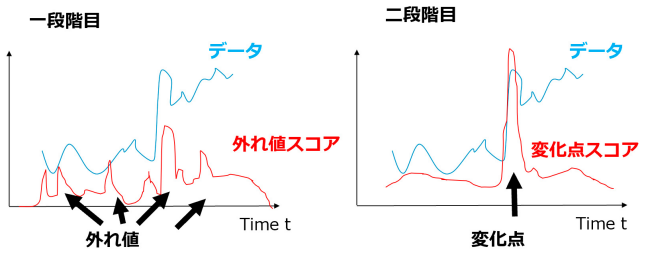

図 2　ChangeFinder の二段階学習

SDAR モデルでは以下の処理を行うことで入力ごとにパラメータを更新する。

$$\hat{\mu} := (1-r)\hat{\mu} + r \quad x_t \tag{8}$$

$$C_j := (1-r)C_j + r(x_t - \hat{\mu})(x_{t-j} - \hat{\mu})^T \tag{9}$$

$$\hat{a_i} := \sum_{i=1}^{k} \hat{a_i}(x_{t-i} - \hat{\mu}) + \hat{\mu} \tag{10}$$

$$\hat{\Sigma} := (1-r)\hat{\Sigma} + r(x_t - \hat{x}_t)(x_t - \hat{x}_t)^T \tag{11}$$

ここで r は忘却率を表し、小さい r の方が過去のデータの影響が大きくなる。

まず式 8 に示すようにそれぞれの t に対して忘却率 r で重みづけした平均 $\hat{\mu}$ を入力データ x_t を使用して更新する。次に式 9 によって自己共分散関数 $C_j : j = 1, ..., k$ を更新し、AR モデルの係数 $a_1, ..., a_k$ の推測値 $\hat{a_1}, ..., \hat{a_k}$ を以下の式を満たすように求める。

$$\sum_{i=1}^{k} A_i C_{j-i} = C_j \tag{12}$$

この式はユーレウォーカーの方程式と呼ばれる。この式の解法についてはのちに述べる。

求めた $\hat{a_1}, ..., \hat{a_k}$ を用いて共分散行列 Σ の推測値 $\hat{\Sigma}$ を式 11 で求めることができる。以上により $\hat{a_1}, ..., \hat{a_k}$ と $\hat{\Sigma}$ が求められるので確率密度関数が入力ごとに導ける。このように SDAR モデルは AR モデルに忘却率を導入することで $O(n)$ に計算コストを削減しているためオンライン動作に向いている。

2.3　FPGA NIC

FPGA をアクセラレータとして用いる場合、PCIe に接続し、CPU を介して処理をオフロードするのが一般的である。

しかし文献 [4]、文献 [5] 等でネットワークに直接 FPGA を接続し、ネットワークからのデータにホストから FPGA にオフロードした処理を行うことでホストを介さずに処理を行う手法が示された。ホストを介さないことでプロトコルスタック等のホストとの通信処理を回避することができるため高速な処理が可能になる。

このようなネットワークに直接接続した FPGA については既に多くの研究がなされている。文献 [6] ではマイクロバッチ方式のストリーム処理フレームワークである Spark Streaming の処理を FPGA NIC にオフロードすることでスループットを向上している。文献 [7] ではキャッシュを適用するブロックチェーンの検索において、FPGA NIC を用いた高速化を提案している。

本論文では外れ値検出と変化点検出を同時に扱える Changefinder アルゴリズムを FPGA NIC にオフロードする。ここでは同様に FPGA で変化点検出と外れ値検出を高性能化した研究について述べる。

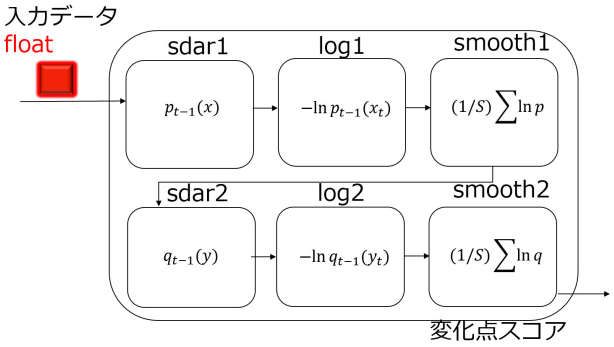

図 3　ChangeFinder コアのパイプライン

変化点検出を FPGA で高性能化したものの例として文献 [8] がある。この論文ではネットワークの攻撃を検知するために NPCUSUM (Non-Parametric Cumulative SUM) という以下に示す単純な変化点検出アルゴリズムを FPGA NIC 上に実装し、高速化している。

$$S_0 = 0 \tag{13}$$

$$S_n = max\{0, S_{n-1} + X_n - \hat{\mu} - \epsilon\hat{\theta}\}, \tag{14}$$

ここで X_n は入力データを表し、$\hat{\mu}$ は X_n の攻撃前の推定値、$\hat{\theta}$ は攻撃後の推定値、ϵ はチューニングパラメータを表す。S_n が劇的に変化した時に攻撃が検知される。これは非常に簡単な実装で実現可能であるが、$\hat{\mu}$ と $\hat{\theta}$ は事前に決めておく必要があり、対応可能なアプリケーションには制限がある。外れ値検出を FPGA で実装したものとしては文献 [9] がある。この論文ではマハラノビス距離に基づいた外れ値検出を FPGA NIC に実装することで性能を向上させている。これらの前例の研究は対象のデータは一次元であり、外れ値検出か変化点検出のどちらかしか行っていない。本論文ではオンラインで学習可能な多次元変化点検出と外れ値検出を行える FPGA NIC を実装する。

3.　設　　計

3.1　一次元変化点検出に対応した FPGA NIC

文献 [2] において一次元の浮動小数点数データのみに対応した Changedfinder NIC を提案、実装している。図 3 に示すように Changefinder コアは 6 つのモジュールのパイプラインからなる。入力は一次元の 32bit の浮動小数点数であり、各モジュールの詳細は下に示す通りである。

- *sdar1*: 入力 x_t に対する確率密度関数 $p_t(x)$ を算出する。
- *log1*: $p_t(x)$ の対数損失を算出する。
- *smooth1*: $p_t(x)$ の対数損失の移動平均 y_t を算出する。
- *sdar2, log2, smooth2*: *sdar1, log1, smooth1* と同様の処理をする。

各モジュールはその動作に必要なデータがそろってから動作する。この Changefinder コアは 125MHz で動作し、8 サイクルごとに入力データを処理できる。この作成した Changefinder コアを NetFPGA チーム [10] が提供する Reference NIC デザインに組み込んだ。図 4 にその様子を示す。実装対象の FPGA ボードである NetFPGA SUME はネットワークに接続する 10Gbit Ethernet の入出力インターフェースを 4 つと、ホストへの通信のための PCI-Express Gen3 x8 のインターフェー

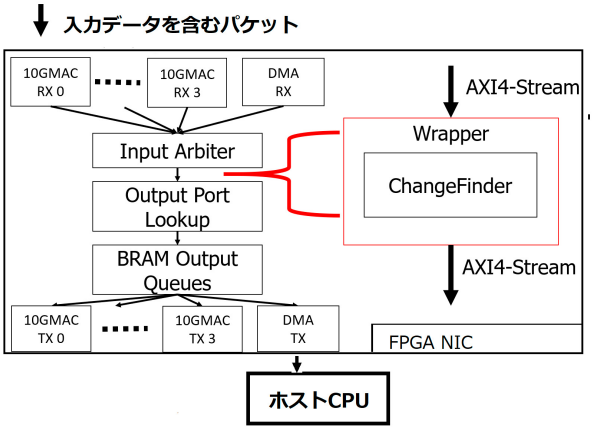

図 4　ChageFinder コアの NIC への組み込み

スを持っている。Reference NIC デザインは Input Arbiter モジュールでこれらのインターフェースからパケットを受け取り、Output Port Lookup モジュールでパケットの宛先を決定し、Output Queue で送信することで NIC としての機能を FPGA に実現している。

この実装では Changefinder コアを Input Arbiter モジュールと Output Port Lookup モジュールの間に実装することで NIC への組み込みを行っている。

評価では実機測定でソフトウェアでの処理に比べて 16.8 倍のスループットを得ることができた。レイテンシは 3 μ 秒に抑えることができた。

3.2　多次元変化点検出に対応した FPGA NIC

一次元のデータは 3.1 節で示したように変化点検出 NIC のように実装、評価されているが、現実の検出対象は多くの要素を持っており、一次元で表現できないものも多い。そのため本論文では多次元に対応した変化点検出 NIC を提案する。

Changefinder のアルゴリズム自体は多次元に対応しているものの、AR モデルの係数を求めるために解かなければならないユーレウォーカーの方程式の解法が一次元の場合と異なる。

式 12 で示したユーレウォーカーの方程式は以下のように変形できる。

$$\begin{pmatrix} c_0 & c_1 & \cdots & c_{k-1} \\ c_1 & c_0 & \cdots & c_{k-2} \\ \vdots & & \ddots & \vdots \\ c_{k-1} & c_{k-2} & \cdots & c_0 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{pmatrix} \quad (15)$$

この式の左辺の行列はテプリッツ行列と呼ばれ、このような線形方程式はアルゴリズム 1 で示す Levinson-Durbin のアルゴリズムで高速に解くことができることが知られている。このアルゴリズムにより、自己共分散関数から AR モデルの係数が逐次的に求めることができ、各データに対する計算量が抑えられるため一次元の Changefinder コアは 8 サイクルごとの処理が可能であった。しかしながらこのアルゴリズムの導出において AR モデルの係数 a_i がスカラであることを利用しているため、a_i が行列である多次元の場合はこのアルゴリズムは使用できない。そのため多次元の場合は自己共分散関数からテプリッツ行列を作り、その逆行列を求めて線形方程式を解くことになる。

Algorithm 1 Levinson-Durbin Algorithm

AR モデルの次数 k
次数が t の時の AR モデルの係数 $a_i^t (i = 1, 2, ..., k)$
自己共分散関数 $c_i (i = 1, 2, ..., k)$
$a_0^{(0)} \Leftarrow 1$
$a_1^{(0)} \Leftarrow 0$
$\omega_0 \Leftarrow c_1$
$u_0 \Leftarrow c_0$
$m \Leftarrow 1$
while $m \neq k$ **do**
  $k_m \Leftarrow \omega_{m-1}/u_{m-1}$
  $u_m \Leftarrow u_{m-1}/(1 - k_m^2)$
  for $i = 1$ to m **do**
    $a_i^{(m)} \Leftarrow a_i^{(m-1)} - k_m a_{m-i}^{(m-1)}$
  end for
  $\omega_m \Leftarrow \sum_{i=0}^{m} a_i^{(m)} c_{m+1-i}$
m $\Leftarrow m + 1$
end while

確率密度関数を求めた後は一次元の数値を扱うことになるため、変更が必要なのは sdar1 モジュールとデータの受け渡し部のみである。また次元数に応じて逆行列計算部の面積が増大するため、対応可能な次元には限界が生じるはずである。現状の実装における次元数の限界については評価において議論する。

4.　多次元変化点検出の実装

3.1 節を元にした一次元のデータ対応の変化点検出を元に多次元に対応した SDAR 学習モジュールとパケットから対応するデータを取り出すためのラッパーを実装した。

4.1　SDAR モジュールの実装

図 5 に共分散行列 σ と入力 x の推定値 $\hat{x}$ を出力する多次元対応の sdar モジュールのブロック図を示す。入力は忘却率である r とデータ本体である x_t の二つで、x_t は一次元あたり 32bit の浮動小数点数である。今回の実装では次元数 n は高位合成時に設定した定数であるとした。大きく分けると sdar モジュールは 6 つのパイプライン化されたサブモジュールでできている。

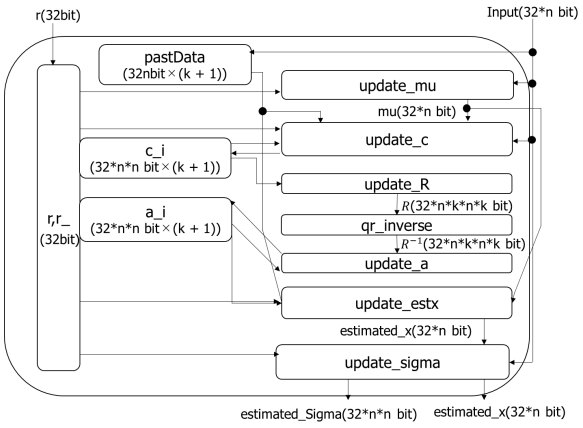

図 5　多次元対応 SDAR モジュールのブロック図

x_t は pastData と図に示される BRAM に保存される。k は AR モデルの次元数であり、AR モデルにおいて何個前までの入力を参照するかを示す値である。

x_t と r、$(1-r)$ を用いて *update_mu* モジュールの処理を行う。*update_mu* では式 8 に従って μ の値を更新する。μ の値は *update_c* モジュールと *update_estx* モジュールで使用される。*update_c* モジュールでは式 9 に従って自己共分散関数 C_i の値を更新する。

update_R モジュールではユーレウォーカーの方程式で用いるテプリッツ行列 R を C_i を使って求める。

qr_inverse モジュールでは Vivado HLS に用意された逆行列計算コアを用いて R の逆行列を求める。

update_a モジュールでは式 12 に基づいて R の逆行列を用いて AR モデルの係数 a_i を求める。*update_R*、*qr_inverse*、*update_a* の三つのモジュールが一次元の場合はアルゴリズム 1 で行っていた部分にあたる。

update_estx モジュールでは式 10 に基づいて a_i と pastData、μ を用いて x_t の推測値 $\hat{x}_t$ を求める。最後に *update_sigma* モジュールでは式 11 に基づいて $\hat{x}_t$ と x_t を用いて $\hat{\Sigma}$ を算出する。

4.2 FPGA NIC への組み込み

3.1 節での一次元変化点検出 NIC と同様に NIC としての機能は NetFPGA チームの Reference NIC を利用し、その中に高位合成で作成した Changefinder コアを組み込んだ。組み込む際にデータの受け渡しと動作周波数の調整のために実装したラッパーを図 6 に示す。Reference NIC における AXI-Stream のデータ幅からパケットは 256bit ごとのフリットにわけられて処理される。また Reference NIC の動作周波数は 160MHz だが、今回実装した多次元対応の Changefinder コアは 80MHz で動作するため、図中の CLK_gen においてクロックを作り、入出力のための非同期 FIFO と Changefinder コアに送っている。

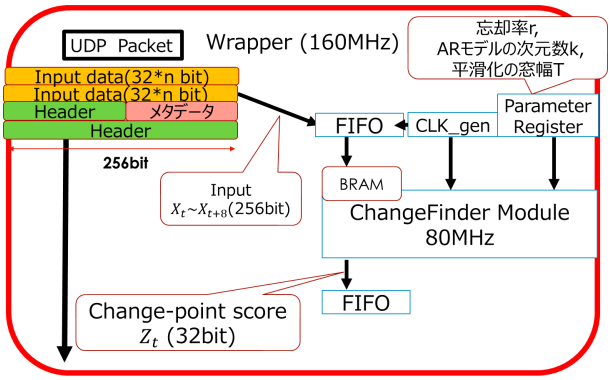

図 6　多次元対応 Changefinder NIC のラッパー

パケット処理の流れは以下の通りである。まず最初と次のフリットではヘッダーが含まれるので、ここで変化点検出の対象のパケットかどうかを判断する。対象でない場合は何の処理もせずにそのまま素通りさせる。処理対象であった場合は 2 フリット目から忘却率、AR モデルの次元数、平滑化の際の窓幅等のパラメータを取り出す。3 フリット目以降にデータ本体が含まれる。

データ本体は次元数分の 32bit 浮動小数点数で構成されるが、256bit ごとに処理されるので 8 個ずつ非同期 FIFO に入れられる。必要分のデータがたまったら Changefinder コアでの処理を開始する。データは 256bit ずつ入力され、Changefinder コア内で BRAM に配置される。

出力の変化点スコアは一次元の 32bit の浮動小数点数である。この値は非同期 FIFO に出力され、ホストが AXI を通して読み込めるレジスタに書き込まれる。

5. 評　　価

本論文の評価ではまず次元数に応じた面積評価を行うことで次元数の面積的な限界を調べる。次に次元数の増加がスループットにもたらす影響を評価する。

5.1 評 価 環 境

評価環境を表 1 に示す。

表 1　評 価 環 境

OS	CentOS release 6.9
CPU	Intel Xeon E5-1620v2 (4C) × 2 @3.70GHz
メモリ	128GB

また論理合成の環境は以下の通りである。

- Xilinx Vivado Design Suite 2016.4
- Xilinx Virtex-7 VX690T (NetFPGA-SUME)

5.2 AR モデルの次元数に関する予備評価

本論文の実装ではモデル学習の際に逆行列計算する行列の大きさはデータの次元数と AR モデルの次元数 k によって決まる。そのため対応可能なデータの次元数を議論する前に k がどの程度大きくなければならないか簡易な評価を行った。

今回は入力次元数は 2 として以下の 2 つのケースについて疑似データを作り、k=2,4 の時でそれぞれソフトウェアシミュレーションを行った。また忘却率は 0.008、平滑化窓幅は 8 とした。

- 共に平均が 1 で分散が 0.05 の正規分布で、平均 2 と 1.5 にそれぞれ変化する。
- 共に分散は 0.1 の正規分布で、一次元目は平均が 0.1 ずつ増加する。その後増減の割合が-0.1,0.05 に変化する。二次元目は平均が 0.02 ずつ増加し、増減の割合が 0.1,-0.1 に変化する。

図 7 と図 8 に結果を示す。結論として今回の疑似データでは $k=2$ でも十分変化点の検知は可能であり、それほど k が大きい必要はないと考えられる。実データによる定性的評価は今後の課題とする。

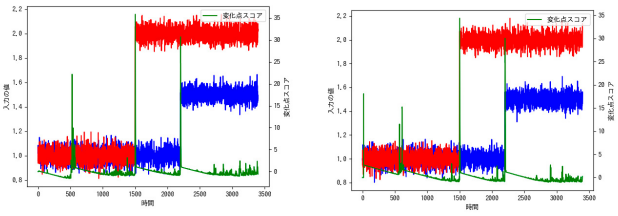

図 7　平均の変化の検出

5.3 多次元 Changefinder コアの面積評価

データの次元数 n と AR モデルの次元数 k を変化させながら、多次元 Changefinder コア単体を面積を評価した。一次元のものは文献 [2] の実装に従った。その結果を表 2、3、4 に示す。Changefinder コア単体でみると k=2 の時は 6 次元、k=3 の時は 4 次元、k=4 の時は 2 次元まで今回の実装対象の FPGA に実装可能であるといえる。

5.4 多次元 ChangefinderNIC の面積評価

作成した Changefinder コアを FPGA NIC に組み込んだ時の

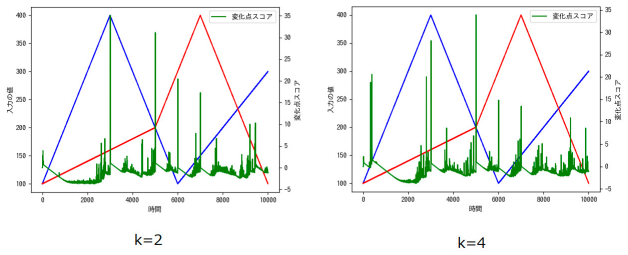

図 8　直線的の変化の検出

表 2　Changefinder コアの面積評価 k=2

	LUT 使用率	FF 使用率	DSP 使用率	BRAM 使用率
1 次元	10%	11%	11%	0%
2 次元	10%	4%	9%	0.1%
3 次元	18%	8%	17%	0.2%
4 次元	29%	13%	30%	0.2%
5 次元	51%	27%	68%	0.3%
6 次元	84%	44%	98%	0.3%

表 3　Changefinder コアの面積評価 k=3

	LUT 使用率	FF 使用率	DSP 使用率	BRAM 使用率
1 次元	10%	11%	11%	0%
2 次元	10%	4%	9%	0.1%
3 次元	44%	22%	49%	0.2%
4 次元	78%	41%	90%	0.2%

表 4　Changefinder コアの面積評価 k=4

	LUT 使用率	FF 使用率	DSP 使用率	BRAM 使用率
1 次元	12%	5%	13%	0%
2 次元	27%	12%	28%	0.4%
3 次元	133%	74	180%	1.2%

面積評価を行った。今回は k=2 で次元数 5 の場合について行った。結果を表 5 に示す。次元数 6 の場合を実装対象の FPGA NIC に組み込むことは面積の制限でできなかった。資源には多

表 5　Changefinder NIC の面積評価 (k=2、次元数 5)

LUT 使用率	FF 使用率	DSP 使用率	BRAM 使用率
63%	35%	68%	16%

少の余裕があるので表 2、3、4 において $k = 2$ で次元数が 5 より少ないものに関しても実装可能であると考えられる。

5.5　スループット評価

次元数を増やすことによるスループットの低下についてデータの次元数 n と AR モデルの次元数 k を変化させながらシミュレーション評価した。各値は入力のインターバルから算出した。結果を図 9 に示す。

面積の時と同様に AR モデルの次元数 k が大きく、入力次元数が大きいほどスループットは低くなった。$k = 2$、次元数が 2 の時は 0.82Gbps、次元数 5 の時は 79Mbps のスループットとなった。

6.　まとめと今後の課題

本論文では、オンライン動作可能な変化点検出アルゴリズムである Changefinder を NIC として機能する FPGA にオフロードした研究を踏まえ、多次元データに対応することを提案した。面積評価により入力データが 5 次元程度であれば実用的な範囲で実装可能であることがわかった。今後の課題としては

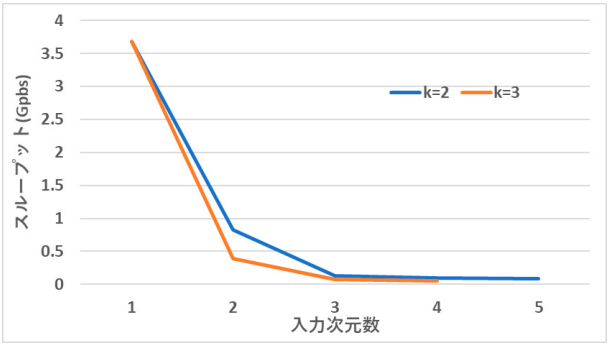

図 9　スループット評価

固定小数点化や実装の改善による対応可能な次元数の向上とともに、より大きな次元数に対応するための近似的手法の検討が必要であると考えられる。

謝辞　本研究の一部は、JSPS 科研費 JP16H02793 および JST CREST JPMJCR1785 の助成による。

文　献

[1] Kenji Yamanishi, Anomaly Detection with Data Mining, Kyouritsu, 2010.

[2] T. Iwata, K. Nakamura, Y. Tokusashi, and H. Matsutani, “Accelerating Online Change-Point Detection Algorithm using 10 GbE FPGA NIC,” Proceedings of the International European Conference on Parallel and Distributed Computing (Euro-Par’18) Workshops, pp.506–517, Aug. 2018.

[3] K. Yamanishi and J. Takeuchi, “A Unifying Framework for Detecting Outliers and Change Points from Non-Stationary Time Series Data,” Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD’02), pp.676–681, July 2002.

[4] A.M. Caulfield, E.S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, “A cloud-scale acceleration architecture,” 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp.1–13, Oct. 2016.

[5] J. Weerasinghe, R. Polig, F. Abel, and C. Hagleitner, “Network-Attached FPGAs for Data Center Applications,” Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT), pp.36–43, Dec. 2016.

[6] K. Nakamura, A. Hayashi, and H. Matsutai, “An FPGA-Based Low-Latency Network Processing for Spark Streaming,” Proceedings of the IEEE International Conference on Big Data (BigData’16) Workshops, pp.2410–2415, Dec. 2016.

[7] Y. Sakakibara, K. Nakamura, and H. Matsutani, “An FPGA NIC Based Hardware Caching for Blockchain,” Proceedings of the 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART’17), pp.11–16, June 2017.

[8] P. Benacek, R.B. Blazek, T. Cejka, and H. Kubatova, “Change-Point Detection Method on 100 Gb/s Ethernet Interface,” Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS’14), pp.245–246, June 2014.

[9] A. Hayashi, Y. Tokusashi, and H. Matsutani, “A Line Rate Outlier Filtering FPGA NIC using 10GbE Interface,” ACM SIGARCH Computer Architecture News, vol.43, no.4, pp.22–27, Sept. 2015.

[10] “The NetFPGA Project”. `http://netfpga.org/`.