情報処理学会論文誌 Vol.57 No.8 1664–1679 (Aug. 2016)

FPGA NIC向けノンパラメトリックオンライン外れ値検出機構

林 愛美[1,a)] 徳差 雄太[1,b)] 松谷 宏紀[1,2,3,c)]

受付日 2015年11月19日, 採録日 2016年5月17日

概要：センシング技術やInternet of Things（IoT）技術の発展にともない，生成されるセンサデータ量は増加し続け，組み込み分野においても大規模データを処理可能なセンサデータ収集システムの必要性が増している．このような膨大な量のデータの中から期待されるパターンと一致しないアイテムのみを効率的に検出するアルゴリズムとして，データセットモデルに依存しにくいノンパラメトリックな外れ値検出アルゴリズムの利用が期待されている．本論文では，ネットワークアプライアンスでの利用を想定し，10 Gbit Ethernetインタフェースを有するFPGAネットワークインタフェースカード（FPGA NIC）上に外れ値検出機構を実現するための手法を提案する．LOF（Local Outlier Factor）は精度の高い外れ値検出アルゴリズムであるが，その計算の複雑さやデータセットモデルの大きさから，FPGAにはオフロードされていない．本論文ではLOFアルゴリズムをFPGA NICのFPGA部に実装した．正常値を含むサンプルデータパケットはFPGA NICでフィルタされ，外れ値を含むデータのみソフトウェアで処理するシステムを提案する．ただし，LOFアルゴリズムをそのままFPGA NICにオフロードすると，計算やメモリのために必要な資源が膨大な量になってしまうため，データセットモデルの一部をキャッシュするシステムを提案する．100,000サンプルを含むデータセットモデルを用いた評価の結果，45%～90%のデータがLOFによる外れ値フィルタリングNICのキャッシュにヒットし，NICによる外れ値検出を実現できた．これは，外れ値検出をすべてソフトウェアで実行した場合に比べて1.82倍～10倍の外れ値検出処理スループット向上に相当する．

キーワード：FPGA，FPGA NIC，外れ値検出，LOF

A Nonparametric Online Outlier Detector for FPGA NICs

Ami Hayashi[1,a)] Yuta Tokusashi[1,b)] Hiroki Matsutani[1,2,3,c)]

Received: November 19, 2015, Accepted: May 17, 2016

Abstract: As the sensing technology and Internet of Things (IoT) technology advance, sensor data stream continuously grows in size, and thus a sensor data aggregation demands a high throughput even in embedded system domains. As outlier detection that filters non-essential data by comparing with expected patterns, especially a nonparametric outlier detection algorithm that does not depend on dataset model is an attractive choice for the sensor data aggregation. In this paper, we propose a nonparametric outlier detection mechanism using an FPGA network interface card (FPGA NIC) that equips four 10 Gbit Ethernet interfaces for network appliances. We employ LOF (Local Outlier Factor) algorithm for outlier detection as it is known as a high precision algorithm. However, FPGA-based design of LOF algorithm has not been reported due to the algorithm complexity and large data set required. In our design, LOF algorithm is implemented on the FPGA and non-essential data are filtered at the NIC, while the others are processed by a software. A naive offloading of LOF algorithm to FPGA devices may significantly increase hardware resources because of complexity of the computation and its large dataset model. Thus we propose to cache only a frequently-used portion of the dataset model in the FPGA NIC. The simulation results using a dataset model containing 100,000 sample data show that 45%–90% of input data are hit to the cache and filtered at the NIC. It corresponds to a 1.82x to 10x throughput improvement on the outlier filtering compared to that of a software-based execution.

Keywords: FPGA, FPGA NIC, outlier detection, LOF

© 2016 Information Processing Society of Japan

1. はじめに

現在，様々なものにセンサが組み込まれ，あらゆるものが情報化されている．このようなセンシング技術や Internet of Things（IoT）技術の発展にともない，生成されるセンサデータ量は増加し続け，組み込み分野においても膨大な量のセンサデータを処理可能なセンサデータ収集システムの必要性が増している．

膨大な量のセンサデータを効率的に蓄積するためには，データの選択（フィルタリング）もしくはデータの圧縮が一般的な解決策である．そこで，本論文では，センサデータ収集システムでの利用を想定し，受信したセンサデータをオンラインでフィルタリングし，さらなる解析もしくはストレージへの保存が必要なデータのみを選択的に取得するシステムを実現する．**図 1** に本研究が想定するシステムの概要を示す．センサノード（クライアント）からネットワークを通じてセンサデータ収集システム（サーバ）に入力されたセンサデータは，サーバ側の NIC（Network Interface Card）内でデータマイニングアルゴリズムによって検査され，必要なデータかどうかを判定される．必要でないと判定されたデータは NIC 内で破棄され，必要なデータのみがネットワークプロトコルスタックへと渡され，ストレージへの保存，もしくは，ソフトウェアによるさらなる解析が行われる．この場合，1 度破棄してしまったデータを復元することは困難であるため，データの選択は慎重でなければならない．そこで，本論文では，精度の高い外れ値検出アルゴリズムとして Local Outlier Factor（LOF）[3] を採用する．LOF はノンパラメトリックであらゆるデータセットに対して適用可能である反面，計算の複雑さや必要とするデータセットモデルの大きさから，NIC のハードウェアへのオフロードに関する既存研究はない．

ネットワークの帯域は増加し続けており，数十 Gbps，もしくはそれ以上の帯域も現在利用可能となってきた．この帯域の増加にともない，継続的に流れてくるデータをオンザフライ処理するストリーム処理の高性能化も重要となってきた．しかし，本論文で対象とする LOF を用いた外れ値検出は計算負荷が高くストリーム処理として利用することが難しいアルゴリズムである．たとえば，本論文での評価（4 章参照）では，ソフトウェアアプリケーションでは秒間 3,600 サンプル程度しか外れ値検出を行うことができなかった．これは，64 Byte 長のパケットを秒間 10,000,000 個以上受信可能な 10 Gbps のラインレートに比べると非常に低い性能であり，ストリーム処理のボトルネックとなってしまう．そのため，LOF アルゴリズムの性能向上を達成するためのシステムが必要であると考え，手段の 1 つとして LOF アルゴリズムの一部をホストアプリケーションから NIC 内にオフロードする方法を提案する．

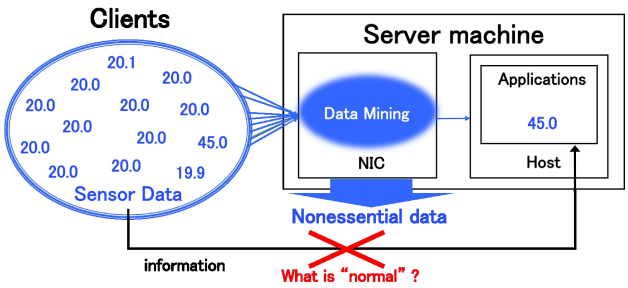

図 1 データマイニングを用いたフィルタリング NIC
Fig. 1 Filtering NIC using data mining.

受信したセンサデータに対して，NIC 内で外れ値検出を行い，外れ値以外のセンサデータをパケットのまま破棄する．これにより，アプリケーション層での外れ値検出処理だけでなく，ネットワークプロトコルスタックでの処理も削減することができ，ホストの負荷軽減に寄与し，ストリーム処理全体の性能向上を見込むことができる．

なお，アルゴリズムを NIC 内に移植することは可能であっても，シンプルにデータマイニングとフィルタリングを行うだけではシステムが成り立たない場合もある．具体的には，必要のないセンサデータを NIC 内で破棄することでホストは入力されてきたデータセット全体を把握することができなくなるが，このことがシステムに影響を与える場合である．この影響はデータマイニングアルゴリズムの種類や実装環境によって異なる．たとえば，データセットモデル（確率モデルや分類器など）の更新が逐次的に行われ，かつ，ホストでデータセットモデルの更新を行う場合，NIC 内で大部分のセンサデータを破棄されてしまうと，ホストはデータセットモデルを適切に更新できなくなってしまう．

本論文で対象とする LOF を用いた外れ値検出では，データセットモデルの更新を行わないため，この問題は起こらないが，文献 [5] で我々はこの問題が起こる場合に焦点を置いており，マハラノビス距離を用いた外れ値フィルタリング NIC の提案を通じて問題の解決方法と実装方法を提案している．

本論文の目的は，LOF を用いた外れ値検出機構を FPGA（Field Programmable Gate Arrays）ベースの NIC[*1] 上に実装し，NIC 内で外れ値検出およびセンサデータのフィルタリングを行うことである．また，LOF を用いた外れ値検

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University, Yokohama, Kanagawa 223–8522, Japan
[2] 科学技術新興機構さきがけ
PRESTO, Japan Science and Technology Agency
[3] 国立情報学研究所
National Institute of Informatics
a) hayashi@arc.ics.keio.ac.jp
b) tokusasi@arc.ics.keio.ac.jp
c) matutani@arc.ics.keio.ac.jp

[*1] 以降，FPGA NIC と略す．

© 2016 Information Processing Society of Japan

出機構を FPGA 上に構築する際に起こる問題点を指摘し，その解決策を提示する．本論文の構成は以下のとおりである．2 章で関連研究を紹介し，3 章で LOF を用いた外れ値フィルタリング NIC を提案する．4 章では提案手法を評価し，5 章で本論文をまとめる．

2. 関連研究

外れ値検出アルゴリズムのうち，NIC にオフロード可能と考えられるマハラノビス距離および LOF を用いた手法を紹介する．

2.1 マハラノビス距離を用いた外れ値検出

マハラノビス距離とは，統計や機械学習の分野で幅広く利用されている，データの特徴ごとの分散の相関を考慮した距離である．m 個のサンプルデータを含むデータセット $x_1, x_2, ...x_m$ に対する新たなサンプルデータ x のマハラノビス距離を用いた外れ値検出は以下の式で表される．

$$\mu = 1/m \sum_{i=1}^{m} x_i \tag{1}$$

$$\Sigma = 1/m \sum_{i=1}^{m} (x_i - \mu)(x_i - \mu)^T \tag{2}$$

$$\sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)} > \theta \tag{3}$$

式 (3) の左辺がマハラノビス距離である．マハラノビス距離は，特徴ごとの分散を相関付ける共分散行列（式 (2)）と，対象データの各特徴の値 x_i と対応する平均値（式 (1)）からの距離を用いて計算される．このマハラノビス距離を，ユーザによって設定された閾値 θ と比較し，閾値よりもマハラノビス距離が大きければ，分散を考慮したうえでデータ全体の平均から離れすぎていると判断され，外れ値と判定される．この計算に利用される m 個のサンプルデータは，最新 m 個のものに順次更新される．このような古いサンプルデータの情報を忘れて，新しいサンプルデータの情報のみを利用するアルゴリズムを忘却型アルゴリズムと呼ぶ．文献 [5] では，ホストが入力されてきたデータセット全体を把握することができなくなるという問題への対処に焦点を置いており，このマハラノビス距離を用いた外れ値フィルタリング NIC の提案を通じて問題の解決方法と実装方法を提案している．

このマハラノビス距離を用いた外れ値検出は単純で利用しやすい反面，古典的なパラメトリックアルゴリズムであるため，それ単体で利用するには力不足であることが多い．そこで，本論文では，外れ値検出の中でも，様々なデータセットに適用可能で精度も高いノンパラメトリックアルゴリズムである LOF を用いた外れ値検出アルゴリズムを利用した外れ値フィルタリング NIC を提案する．

2.2 Local Outlier Factor を用いた外れ値検出

2.2.1 Local Outlier Factor

LOF は Breunig らによって提案された密度をベースとしたノンパラメトリックな外れ値検出アルゴリズムである [3]．密度を基に外れ値かどうかを判定しているため，データセット内に確率モデルが異なる複数のクラスタが存在する場合にも，それぞれのモデルを特定することなく直接外れ値を検出することができる．以下にその基本的なアルゴリズム（オフライン実行）を簡単に説明する．データセット D に含まれるサンプルデータ p のうち，外れ値であるものを検出する．初めに，対象サンプル p の k-distance を $k\_distance(p)$ と定義する．$k\_distance(p)$ は，サンプル $o \in D$ と p の距離 $d(p,o)$ で表される．この $o \in D$ は以下の条件で与えられる．k はユーザによって与えられる正の整数パラメータとする．

少なくとも k 個のサンプル $o' \in D$ について，以下の関係が成り立つ．

$$d(p,o') \leq d(p,o) \tag{4}$$

また，$k-1$ 個以下のサンプル $o' \in D$ について，以下の関係が成り立つ．

$$d(p,o') < d(p,o) \tag{5}$$

次に，この $k\_distance(p)$ を用いて p の k-distance neighborhood を $N_{k\_distance(p)}(p)$ として以下の式で定義する．

$$N_{k\_distance(p)}(p) = \{q \in D | d(p,q) \leq k\_distance(p)\} \tag{6}$$

つまり，$N_{k\_distance(p)}(p)$ は p の近傍サンプルの集合であり，少なくとも k 個のサンプルが含まれている．また，p と $o \in D$ の reachability distance を $reach\_dist_k(p,o)$ として以下の式で定義する．

$$reach\_dist_k(p,o) = \max\{k\_distance(o), d(p,o)\} \tag{7}$$

p が $o \in D$ から離れている場合は，その距離 $d(p,o)$ がそのまま reachability distance となり，十分に近い場合は，o の k-distance（o と o の近傍サンプルから計算された $k\_distance(o)$）に統一される．このようにすることで，p と o が十分に近い場合の $reach\_dist_k(p,o)$ の変動を抑えることができる．この $N_{k\_distance(p)}(p)$ と $reach\_dist_k(p,o)$ から，p の local reachability density である $lrd_k(p)$ を以下の式で定義する．

$$lrd_k(p) = \frac{|N_{k\_distance(p)}(p)|}{\Sigma_{o \in N_{k\_distance(p)}(p)} reach\_dist_k(p,o)} \tag{8}$$

p の LOF は，式 (8) によって計算された $lrd_k(p)$ と，同様に式 (8) を利用し，$o \in D$ とその近傍サンプルから計算された o の local reachability density（$lrd_k(o)$ と表記）を

© 2016 Information Processing Society of Japan

用いた以下の式で求められる．

$$LOF_k(p) = \frac{\Sigma_{o \in N_{k\_distance(p)}(p)} \frac{lrd_k(o)}{lrd_k(p)}}{|N_{k\_distance(p)}(p)|} \quad (9)$$

この $LOF_k(p)$ が 1 から大きく離れていると，p は外れ値である可能性が高いとされているが，外れ値か否かの明確な基準は設けられていない．また，パラメータ k も推奨値は 10〜20 であることのみ示されている．

LOF を用いた外れ値検出は元々オフラインで実行するためのアルゴリズムであり，基本的にはデータセットに新たなサンプルデータが入力されることは考慮されておらず，用意されたデータセットに含まれるサンプルの中から外れ値であるものを検出することを目的としている．今回は，ネットワークを通じて新たに入力されるセンサデータをフィルタリングすることが目的であるため，あらかじめサーバ側に用意されたデータセットに対する入力データの LOF を計算することで外れ値かどうかを判定する．つまり，上記の手順におけるデータセット D はサーバに用意された外れ値検出の基準とするためのデータセットモデルとなり，対象サンプル p は入力センサデータとなる．

2.2.2 Local Outlier Factor の高速化

LOF アルゴリズムの FPGA へのオフロードに関する研究は知られていない．しかし，他のデバイスを用いた LOF アルゴリズムの高速化や，精度や計算効率の改善を目的とした改良アルゴリズムの提案はさかんに行われている．

Alshawbkeh らは，GPU（Graphic Processor Unit）を利用して LOF アルゴリズムの大幅な高速化を達成している [1]．GPU を用いると，LOF のような計算量が大きく並列性を含むアルゴリズムを効率的に実行することができる．文献 [1] ではマルチコア実行の 100 倍の速度向上を達成している．

Pokrajac らは，元より計算量が大きい LOF アルゴリズムをオンラインで効率的に実行するための改良アルゴリズムの提案を行っている [7]．本論文は，LOF の FPGA へのオフロードの第 1 段階として，オリジナルのアルゴリズムを直接的に拡張したオンラインアルゴリズムを対象とし，種々の改良版アルゴリズム [7] までは対象とはしない．

2.3 データマイニングアルゴリズムのフィルタリング NIC への適用

本論文の目的は，ネットワークから入力されるセンサデータを NIC 内で LOF を用いてフィルタリングし，ホストへの負荷を軽減することである．フィルタリング NIC に実装するデータマイニングアルゴリズムを変更することもできるが，FPGA 資源と NIC-ホスト間の通信量という 2 つの観点から，フィルタリング NIC に適したアルゴリズムの選択およびハードウェア設計をしなければならない．

図 2 に，この 2 つの観点によって，マハラノビス距離を用いた外れ値検出アルゴリズムと LOF を用いた外れ値検出アルゴリズムをプロットしたものを示す．それぞれのデータセットモデル（マハラノビス距離の共分散行列，LOF のデータセット）はホスト側で保持し，更新が必要なものについてはホストアプリケーションで更新を行うとする．

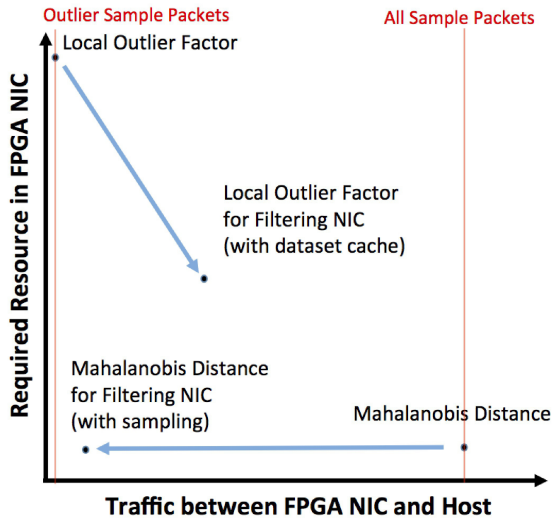

図 2 データマイニングアルゴリズムをフィルタリング NIC に適用する際に考慮すべき 2 つの点

Fig. 2 Two issues that should be considered when data mining algorithm is applied to filtering NIC.

マハラノビス距離を用いた外れ値検出は，2.1 節で示したアルゴリズムに沿って，最近のサンプルデータセットの平均と分散を利用した簡単な計算によって求められるため，比較的少ない FPGA 資源で実装できる．しかし，ホストで共分散行列の計算を行うため，純粋なアルゴリズムに基づいた場合，すべてのデータをホストアプリケーションへと渡さなければならない．そこで，文献 [5] では，平均値の計算を行うために一時的に NIC 内に保持されているサンプルデータセットを定期的に読み出し，それを基にホストアプリケーションで共分散行列を計算をしている．

LOF を用いた外れ値検出は，入力サンプルデータによるホスト上のデータセットへの影響（データセットモデルの更新）はないものの，そもそも外れ値検出を行うためにデータセットに含まれる全サンプルデータを必要とする．さらに，入力サンプルの近傍サンプルを決定するために，用意されたデータセットに含まれる全サンプル数の長さのソートを行う必要がある．Zuluaga ら [9] や小林ら [10] によって提案されている，ソート回路を効率的に構成するための手法を用いれば，ある程度の大きさのデータセット扱うことが可能になるが，4 章で示すように，現在，単一の FPGA では 3,000〜4,000 サンプル程度が限界だと考えられる．本論文では，ホストでデータセット全体を保持し，FPGA NIC 内ではデータセットの一部をキャッシュする

© 2016 Information Processing Society of Japan

ことによって，データセットを保持するために必要なメモリとソートに必要な資源の節約を試みる．

FPGA へのオフロードが困難なアルゴリズムは LOF アルゴリズムのほかにも様々なものがある．たとえば，ツリー型のアルゴリズムである Random Forest [2] は，再帰処理を行うため，容易にハードウェア化することができない．しかし，現在非常に注目されているアルゴリズムの 1 つであることに加え，工夫次第で FPGA による大幅な高速化を達成できるので，幅広く研究されている分野でもある．文献 [4] では，ツリーの深さを制限することのできる CRF（Compact Random Forest）と呼ばれるアルゴリズムを利用することによって，Random Forest による識別の FPGA へのオフローディングと高速化を達成している．

3. 外れ値検出機構を備えた FPGA NIC

この章では，本論文の提案である LOF を用いた外れ値フィルタリング NIC について設計の詳細を示す．

3.1 マハラノビス距離を用いた外れ値検出 NIC

LOF を用いた外れ値フィルタリング NIC の設計を述べる前に，文献 [5] で我々が提案したマハラノビス距離を用いた外れ値フィルタリング NIC の設計について簡単に説明をする．本論文で提案する設計は，このマハラノビス距離を用いた外れ値フィルタリング NIC の外れ値検出機構を LOF を用いた外れ値検出に置き換えたものである．

まず，マハラノビス距離を用いた外れ値フィルタリング NIC のシステムの全体像を**図 3** に示す．実装対象である NetFPGA-10G [8] の AXI（Advanced eXtensible Interface）幅の関係により，外れ値フィルタリングモジュールへのパケット入力は 256 bit ずつ行われる．入力されたパケットが特定のポートへ向けた UDP パケットである場合，そのパケットはサンプルデータを含むと判断される．1 つのサンプルデータパケットには 1 つのサンプルデータが含まれており，1 つのサンプルデータには 32 bit の整数値が特徴個だけ含まれている．ARP（Address Resolution Protocol）パケットなど，その他のパケットの場合は，通常どおりホストのネットワークプロトコルスタックへと渡される．サンプルデータパケットは FIFO バッファに一時的に保存され，中のサンプルデータの値のみが外れ値検出モジュールへと渡される．そして，サンプルデータが外れ値と判定された場合はサンプルデータパケットはホストへと渡される．一方で，外れ値ではないと判定された場合は NIC 内で破棄される．これらの処理はパイプライン化され，1 サンプルあたり 2 サイクルで処理可能である．NetFPGA-10G ボードの動作周波数は 160 MHz であるため，計算上は 1 秒間に 80,000,000 サンプルが処理できる．実機評価の結果，10GbE のラインレートの 95.8%の性能が確認できた [5]．

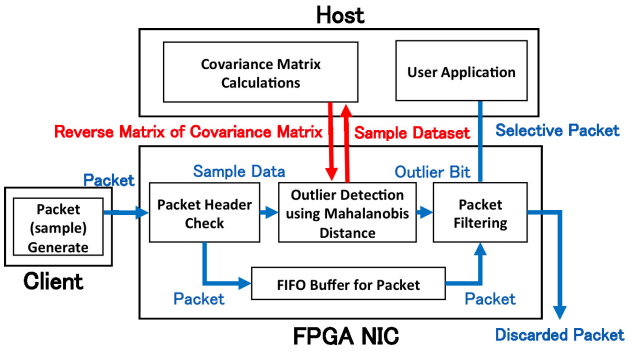

図 3 マハラノビス距離を用いた外れ値検出 NIC
Fig. 3 Outlier filtering NIC using Mahalanobis distance.

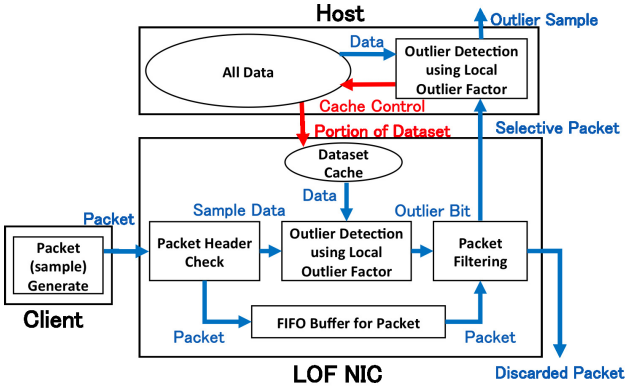

図 4 Local Outlier Factor を用いた外れ値検出 NIC
Fig. 4 Outlier filtering NIC using Local Outlier Factor.

2 章で述べたとおり，データセットモデル（共分散行列）の管理はホストアプリケーションで行っているが，外れ値以外のサンプルデータは NIC 内で破棄されてしまうため，ホストはモデルの更新を行うための情報を別の方法で得る必要がある．そこで，我々は，ホストアプリケーションが定期的に NIC 内に保持されているサンプルデータセットをサンプリングすることによって，データセットの更新を行う手法を提案した．文献 [5] で我々が行った評価では，1,024 個のサンプルデータを利用して共分散行列の計算を行うとき，100,000 個のサンプルデータが入力されるごとに共分散行列の計算を行う場合でも，本来のアルゴリズムに比べて精度の低下は 1%程度のみであった．つまり，1%の精度と引き換えに，すべてのサンプルをホストへ送る場合に比べて，NIC-ホスト間の通信量を約 99%削減できた．

3.2 Local Outlier Factor を用いた外れ値検出 NIC

図 4 は本論文で提案する LOF を用いた外れ値フィルタリング機構を備える FPGA NIC[*2]の概要である．青の矢印で表されている流れは，ネットワークを通じて受信したサンプルデータパケットに含まれるサンプルが外れ値であるかどうかを判定するための流れである．赤の矢印で表さ

[*2] 以降，LOF NIC と略す．

© 2016 Information Processing Society of Japan

れている流れは，LOF NIC によって外れ値と判定されホストへ渡されたサンプルが，ホストアプリケーションによって通常の値と判定された場合，LOF NIC 内にキャッシュされたデータセットの更新を行うための流れである．

3.3 LOF NIC におけるパケット選別

3.3.1 外れ値検出機構の概要

LOF NIC 内のパケット選別処理は図 4 の青の矢印で表されている．LOF NIC の処理の流れは，基本的には 3.1 節で述べたマハラノビス距離を用いた外れ値フィルタリング NIC と同様である．ただし，LOF NIC では，後述するデータセットキャッシュを利用した外れ値検出を行うため，適切なデータセットがキャッシュされていなかった場合，本来の LOF を用いた外れ値検出の結果よりも高く LOF が計算され，外れ値ではないサンプルデータを外れ値であると誤って判定する場合がある．このような場合，本当に外れ値なのか，外れ値と誤認されているのかを LOF NIC 内では判断できないため，そのサンプルデータはホストのアプリケーション層に渡され，ソフトウェアで実装された LOF アルゴリズムによって LOF の値が再計算される．

3.3.2 外れ値検出機構の詳細

LOF を用いた外れ値検出モジュールについて詳細を述べる．入力されたサンプルデータ p は，以下のステップで処理される．

(1) 一番近傍に存在すると推測されるデータセットを FPGA 内の Block RAM（BRAM）にキャッシュされたデータセット群の中から選択する（詳細は 3.4 節）．
(2) 選択したデータセットを基に，p の k-distance を計算する．つまり，データセットに含まれるすべてのサンプルと p との距離を計算し，その結果をソートすることによって k 番目に近いサンプルとの距離を p の k-distance とする（式 (4), (5)）．
(3) p の k-distance を用いて，p の k-distance neighborhood を特定する．つまり，$k+1$ 番目以降に近いサンプルと p との距離を p の k-distance と比較し，同じ距離ならば，そのサンプルも k-distance neighborhood に含める（式 (6)）．簡単のため，k-distance neighborhood の最大数をユーザパラメータ NEIGH_MAX として指定する．
(4) データセットに含まれるサンプルの k-distance と local reachability density（式 (8)）は，事前にホストで計算され，サンプルデータと一緒に NIC 内にキャッシュされている．これを用いて p の local reachability density（式 (8)）を計算する．
(5) ステップ (4) の結果を用いて，p の LOF（式 (9)）を計算する．
(6) ステップ (5) で得られた LOF の値を，ユーザから与えられた閾値と比較し，閾値を超えていた場合，入力サンプルデータを外れ値と判定する．結果は 1 bit の信号として，パケットフィルタリングモジュールへ送られる．

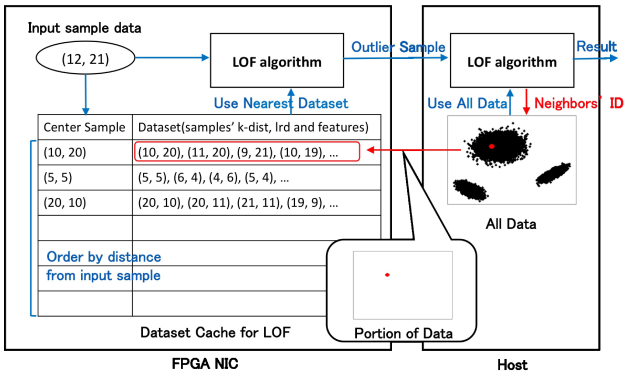

図 5 データセットキャッシュシステム
Fig. 5 Dataset cache system.

3.4 データセットのキャッシュ

3.4.1 キャッシュシステムの概要

図 5 に LOF NIC 内に用意されたデータセットのキャッシュシステムの概要を示す．このキャッシュシステムは FPGA NIC 内で入力サンプルの LOF を計算するためのものである．青の矢印は入力サンプルの LOF が計算される際の流れ，赤の矢印はキャッシュの更新が起こる際の流れである．

初めに，新たなサンプルデータが入力された際の流れについて説明をする．

(1) サンプルデータの LOF を計算するために，キャッシュ内に存在するデータセットの中から利用するものを 1 つ選択する．選択したデータセットをデータセット全体（式 (4)〜(9) における D）として，LOF の値を LOF NIC 内で計算する．
(2) LOF NIC 内での計算結果が外れ値だった場合，サンプルデータはホストの LOF アプリケーションに渡され，LOF の値が計算し直される．
(3) ホストアプリケーションによる計算結果が外れ値ではなかった場合，ホスト上にあるデータセット（図 5 の All Data）から一部のデータセット（赤い部分）を FPGA NIC へキャッシュする．

以上がキャッシュに関する簡単な流れである．以下に設計の詳細を示す．

3.4.2 キャッシュシステムの詳細

3.3.2 項のステップ 1 に示されるように，入力サンプルの LOF を計算するために，まずは利用するデータセットを決定する必要がある．本来のアルゴリズムならばこのデータセットとは用意されたサンプルデータ群全体を示すが，メモリ資源が限定された LOF NIC ではこれはホスト上にのみ存在している．そこで，本論文では，データセット全

© 2016 Information Processing Society of Japan

体から一部のサンプルデータ群を抜き出し，LOF NIC 上の FPGA の BRAM にキャッシュし，これをデータセットとして *LOF* を計算するようにした．キャッシュは複数のラインで構成されており，1 つのラインにデータセットが 1 つ保持される．ラインの数と大きさ（保持されるデータセット 1 つの大きさ）については，4 章で考察を行う．

LOF NIC に新たなサンプルが入力されると，キャッシュ内に存在するデータセットから 1 つが選択される．利用するデータセットは，キャッシュに格納されているデータセットの中で入力サンプルに対して一番近傍にあると推測されるものを利用する．これを実現するため，キャッシュインデックスの代わりに各データセットの代表サンプル（center sample）を用意し，その代表サンプルと入力サンプルの距離が一番近いデータセットを利用する．この代表サンプルはデータセットがキャッシュされる際に決定される．

LOF NIC 内で外れ値と判定されホストへと渡されたサンプルデータが，ホストのアプリケーションによって外れ値ではないと判定された場合，LOF NIC での計算結果が間違っていたことになる．この原因は LOF NIC 内にキャッシュされているデータセットが適当でないことによるものであるため，新たなデータセットを LOF NIC 内に保持するようキャッシュを更新する必要がある．このデータセットは，今回外れ値ではないと判定されたサンプルの近傍 n サンプルを集めたものである（n はキャッシュラインの大きさに依存する）．キャッシュするデータは，サンプルデータそのものの値だけでなく，そのサンプルデータの k-distance および local reachability density の値も含む．これらの値はあらかじめホストアプリケーションによって計算されているものとする．また，このとき最近傍に存在するサンプルデータを代表サンプルとする．

1 回の更新で n サンプル分のデータをホストから NIC へ送らなければならないため，キャッシュの更新にはある程度の時間がかかる．そのため，更新中にも次のサンプルデータがホストへ送られてくる可能性はあるが，そのサンプルが外れ値でない場合でも，そのサンプルによるキャッシュの更新は行わないものとする．キャッシュラインの置き換え方式については，FIFO（First In First Out）を採用している．

3.5 LOF を用いた外れ値検出モジュールの構成

この節では，データセットキャッシュを含めた LOF を用いた外れ値検出モジュール（図 4 における Outlier Detection using Local Outlier Factor の部分）に関して，具体的な回路の構成について説明する．以降の図には LOF を用いた外れ値検出を実現するために重要な要素のみが描かれており，clk や stop などの基本的な制御やその他（パイプライン化など）のための配線・レジスタは省略している．

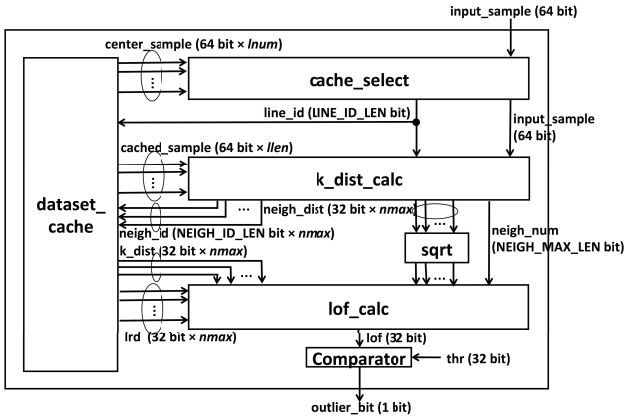

図 6 外れ値検出モジュールの全体図
Fig. 6 Overview of outlier filtering module.

図 6 に外れ値検出モジュールの全体図を示す．図中の *lnum* はキャッシュラインの数，*llen* は 1 ラインに含まれるサンプルの数，*nmax* は近傍サンプル数の最大数（3.3.2 項で示したユーザパラメータ NEIGH_MAX と同義）を表している．LINE_ID_LEN は使用するキャッシュラインの ID を表すために十分な幅であり，キャッシュラインの数で決定される．NEIGH_ID_LEN は近傍サンプルの ID を表すために十分な幅であり，1 ラインに含まれるサンプル数で決定される．なお，この節ではサンプルの特徴数は 2，1 個の特徴は 32 bit で表されるとする．このモジュールは，主に 3 つのモジュールとデータセットキャッシュから成り立っている．サンプルデータが入力されると，cache_select モジュールによって line_id が計算される．この line_id によって，利用されるデータセットが選択され，このデータセットに含まれるサンプルデータ群が k_dist_calc モジュールへと渡される．このモジュールでは近傍サンプルの計算が行われ，結果は近傍サンプルの ID（neigh_id）と近傍サンプルと入力サンプルの距離の 2 乗（neigh_dist），近傍サンプルの数（neigh_num）で出力される．neigh_id と cache_select モジュールから引き継いだ line_id によって，データセットキャッシュの中から近傍サンプルの k-distance（k_dist）と local reachability density（lrd）が選択され，それらと neigh_dist を sqrt 回路に通した近傍サンプルと入力サンプルの距離および neigh_num が lof_calc モジュールの入力となる．lof_calc モジュールでは，入力サンプルと近傍サンプルの reachability distance の計算，入力サンプルの local reachability density の計算，*LOF*（lof）の計算が行われ，lof が出力される．そして，この lof がユーザパラメータの thr と比較されることによって，入力サンプルが外れ値かどうか判定される．判定結果は 1 bit の信号で出力される．各モジュールで使用されている演算器群（掛け算器・割り算器・sqrt 回路）は面積やサイクル数を効率化するため，Xilinx ISE に付属の Core Generator によって生成されたものを使用している．また，大量の演算器によって LUT

© 2016 Information Processing Society of Japan

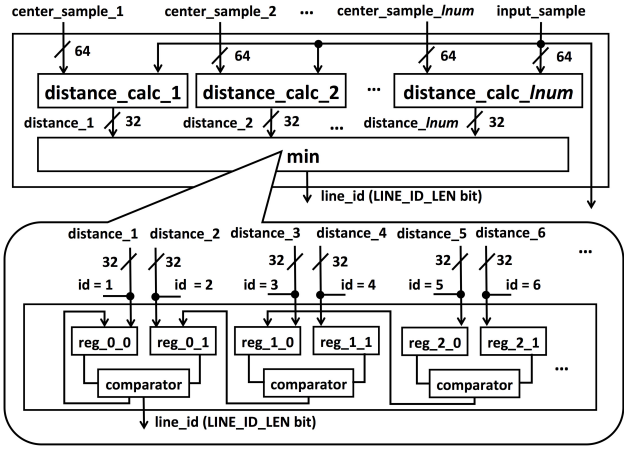

図 7 cache_select モジュール
Fig. 7 cache_select module.

の消費量が著しく増加し，資源不足に陥ることを防ぐため，掛け算器と割り算器は，LUT ではなく DSP による構成を選択した．以下，それぞれのモジュールとデータセットキャッシュの構成について詳細に説明していく．

3.5.1 cache_select モジュール

図 7 に cache_select モジュールの詳細を示す．このモジュールは，center sample（center_sample）群と入力サンプル（input_sample）を入力とし，利用するデータセットの ID を line_id として出力するモジュールである．サンプルが入力されると，distance_calc モジュール群によって，各 center_sample と input_sample の距離の 2 乗が計算される．この distance_calc 内には特徴数（以下，$feat$ と表記）個の掛け算器が使用されており，1 つの掛け算器は DSP スライスを 4 個使用しているため，合計で $4 \times feat$ 個の DSP スライスを使用している．そのため，cache_select モジュール全体では $4 \times feat \times lnum$ 個の DSP スライスを消費する．距離が一番近い center sample を発見するだけでよいので，sqrt 演算は行わずに掛け算のみを行い，距離の 2 乗のまま次の min モジュールへ送られる．

min モジュールは入力された値から最小値を発見し，その ID を出力するモジュールである．ID の付与は入力時にポートごとに行われる．各値はレジスタに保持され，隣り合ったレジスタの値を比較することよって，レジスタの次の値が決まる．たとえば，reg_0_0 と reg_0_1 の値を比較し，小さい方が次の reg_0_0 の値となる．このレジスタと比較演算器の組合せが $lnum/2$ 個並んでおり，1 サイクルに 1 回の比較が行われ，トーナメント方式で最小値が決まってゆき，$\log_2 lnum$ サイクルで最小値が求まる．最小値を求めるためには，値を 1 つずつ比較する方法もあるが，サイクル数が $lnum - 1$ サイクルかかってしまうため，今回は簡単に構成可能であり，サイクル数を効率化できるという理由で，マージソートをもとにしたトーナメント方式を採用した．

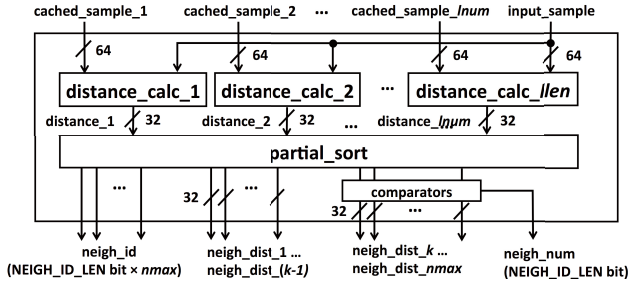

図 8 k_dist_calc モジュール
Fig. 8 k_dist_calc module.

3.5.2 k_dist_calc モジュール

図 8 に k_dist_calc モジュールの詳細を示す．このモジュールは，データセット内のサンプル群（cached_sample）の中から入力サンプル（input_sample）の近傍サンプルを発見し，近傍サンプルそれぞれの ID（neigh_id），近傍サンプルそれぞれと input_sample との距離の 2 乗（neigh_dist），近傍サンプルの数（neigh_num）を出力する．cache_celect モジュールと同様，まずは distance_calc モジュールによる距離の 2 乗の計算が行われる．このモジュールが消費する DSP スライスは $4 \times feat \times llen$ 個である．距離の 2 乗が計算されると，partial_sort モジュールによる選択的ソートが行われる．このモジュールは，入力された値の上位 $nmax$（近傍サンプルの数の最大数）個の ID と値を出力する．これが近傍 $nmax$ 個のサンプルの ID と入力サンプルとの距離の 2 乗となる．さらに，近傍サンプルの個数を知る必要があるため，k（近傍サンプルの最小数）番目に近いサンプルから順に，次に近いサンプルと入力サンプルとの距離の 2 乗を比較する．同じ距離に存在した場合はそのサンプルも近傍サンプルに含め，そのまた次に近いサンプルとの距離の 2 乗を比較する．距離に差があった場合は，そこで比較をやめて，近傍サンプルの個数を決定する．

以降では partial_sort モジュールについて説明する．本論文では簡単のため，広く知られているアルゴリズムを選択的ソート（上位 $nmax$ 個を決定するソート）回路に使用することにした．その中でも，比較的サイクル数の効率が良いマージソートをもとにした選択的ソートを partial_sort モジュールとして実装した．**図 9** に partial_sort モジュールの詳細を示す．図中の data は ID と値を含むため，幅は（NEIGH_ID_LEN+32）bit であるが，スペースの都合で省略している．また，このモジュールは破線で区切られた段階に分かれ，パイプライン化されている．入力された値にはポートごとに ID が与えられ，さらに隣の値と比較され，小さい順にレジスタに格納される．この 2 つの値が小さい順に格納されたレジスタに対してソートを行う．具体的には，レジスタのアドレスが示す値を次々と比較し，別のレジスタへとコピーしていく．最初のレジスタのアドレスは最小値を指しており，その値がコピーされた段階でアドレスがインクリメントされる．たとえば，図 9 の左上の部分

© 2016 Information Processing Society of Japan

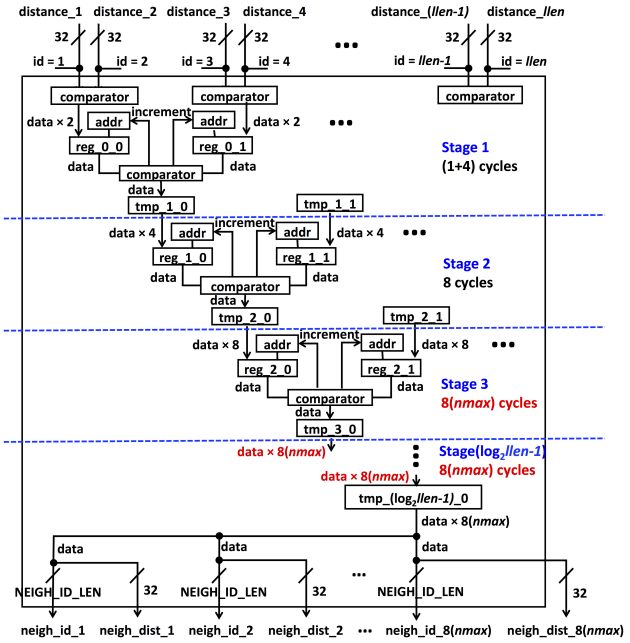

図 9 partial_sort モジュール
Fig. 9 partial_sort module.

で説明をすると，まず reg_0_0 の最小値と reg_0_1 の最小値が比較され，reg_0_0 の方が小さかった場合，その値が tmp_0_0 の先頭に書き込まれ，reg_0_0 のアドレスがインクリメントされる．そして，次のサイクルでは reg_0_0 の 2 番目に小さい値と reg_0_1 の最小値が比較される．4 サイクルかけて，tmp_0_0 に昇順に並べられた 4 つの値が揃ったら，その値を reg_1_0 にコピーし，次のステージで同様に 4 つの値でできた 2 つの昇順の数列から 8 つの値でできた 1 つの昇順の数列を作る．これを繰り返して，最終的な全体の数列を得る．ただし，最初のステージのみは，4 サイクルに加えて，隣り合う 2 つの入力サンプルの比較をしてレジスタに書き込む 1 サイクルの処理を含んでいるため，(4+1) サイクルとなっている．なお，これは選択的ソートであり，上位 $nmax$ 個の値以外をソートする必要はない．つまり，partial_sort モジュールでは，途中で $nmax$ 個よりも長い昇順の数列を作る機会があっても，上位 $nmax$ が確定した時点で処理を中断する．たとえば，図 9 は $nmax = 8$ の場合を表しているため，図の下部では 8 個の値からなる 2 個の数列（合計 16 個の値）から，上位 8 個の値のみを選択し，次のステージに渡している．こうすることで，通常のソート回路に比べてサイクル数や面積を節約している．このサイクル数が本論文で提案する LOF NIC の性能のボトルネックとなり，そのサイクル数は近傍サンプル数の最大数 $nmax$ となる．最終的に得られた上位 $nmax$ 個それぞれの ID と値が出力される．

3.5.3 lof_calc モジュール

図 10 に lof_calc モジュールの詳細を示す．このモジュールは，入力サンプルの近傍サンプルに関するデータ（入力サンプルと近傍サンプルの距離である neigh_dist，近傍サンプルの k-distance である k_dist，近傍サンプルの local reachability density である lrd，近傍サンプルの数である neigh_num）から，入力サンプルの LOF を計算・出力する．まず，neigh_dist と k_dist から入力サンプルと近傍サンプルの reachability distance（図中の reach_dist）を計算する．これは，2.2.1 項の式 (7) の計算にあたる．これ以降の計算は，式 (8) と式 (9) を合わせたものである．具体的には，reach_dist と lrd をそれぞれ合計し，その 2 つを掛け合わせたものを neigh_num の 2 乗で割った値が lof として出力される．このモジュールには 2 つの掛け算器と 1 つの割り算器が使用されている．掛け算器には 4 個の DSP スライス，割り算器には 14 個の DSP スライスが使用されているので，このモジュールは合計で 22 個の DSP スライスを消費している．

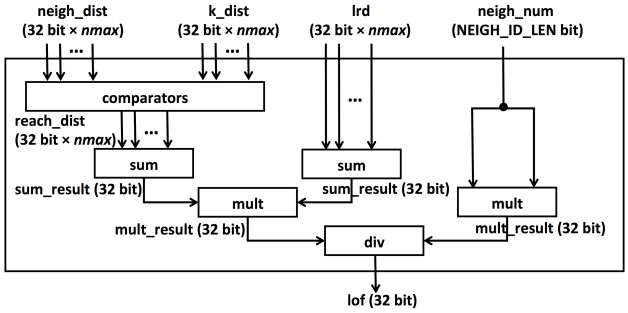

図 10 lof_calc モジュール
Fig. 10 lof_calc module.

3.5.4 データセットキャッシュ

データセットキャッシュには，3 つの役割がある．(1) center_sample を保持・出力する，(2) line_id から適切な cached_sample を出力する，(3) line_id と neigh_id から適切な k_dist と lrd を出力する，という 3 つの役割である．(1) に関しては，キャッシュの更新がない限り，常時同じ値を出力し続けるため，分散 RAM を用意してそこから直接出力に配線する．(2) と (3) に関してはデータセットキャッシュの本体であり，データ量も多いため BRAM で構築する．データセットキャッシュの更新はデータの使用に比べて低頻度であるため，説明を分かりやすくする目的で，以降の図および説明は read に関してのみ記述した．もちろん，キャッシュの更新を行うために，各キャッシュには write のための配線が存在するが，read とは独立している（read と write を同時に行う必要はない）．

まず (2) に関して，**図 11** に構成を示す．cached_sample として，line_id で示されたラインに含まれる全サンプルの特徴値（k_dist や lrd は含まない）を出力する必要がある．つまり，1 つのアドレスで出力データが指定可能であるので，理論上 1 つの BRAM インスタンスで構成可能だが，出力するデータ幅が大きい（たとえば，64 bit のサンプルが 128 個含まれていれば，1 kB の幅が必要）ため，実際には単一のインスタンスでは構成できない可能性が高い．

© 2016 Information Processing Society of Japan

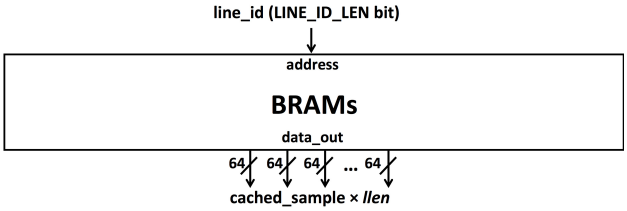

図 11 cached_sample のためのキャッシュ
Fig. 11 Cache for cached_sample.

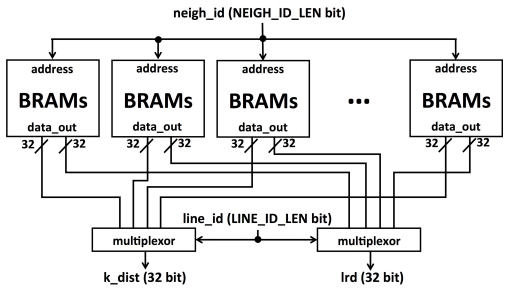

図 12 k_dist および lrd のためのキャッシュ
Fig. 12 Cache for k_dist and lrd.

次に (3) に関して，図 12 に構成を示す．近傍サンプルのデータを出力するためには，そのサンプルが含まれているラインを特定したうえで，その中から近傍サンプルを出力する必要がある．そのため，ラインごとに別の BRAM インスタンスで構成し，それぞれの出力のうち line_id で指定された BRAM から出力されたものをデータセットキャッシュの出力とする．また，近傍サンプルとして指定される ID は *nmax* 個あるため，*nmax* 回に分けて同じデータセットに対してアクセスをする必要がある．データセットを複製し，それぞれを別の BRAM インスタンスで保持することでアクセス回数を減らすことも可能だが，3.5.2 項で述べたとおり，本論文の実装ではサイクル数のボトルネックは選択的ソートにあるため，*nmax* サイクルかけて出力を行う．

4. 評価

4.1 キャッシュヒット率

ここでは，LOF NIC に入力されるサンプルデータ群の傾向と，キャッシュヒット率の関係について比較と考察を行う．ヒット率は「LOF NIC で外れ値ではないと判定されたサンプル数/(LOF NIC で外れ値ではないと判定されたサンプル数 +LOF NIC では外れ値と判定されたがホストの LOF アプリケーションによって外れ値ではないと判定されたサンプル数)」で表される（図 **13**）．

4.1.1 評価環境

R 言語によるヒット率のシミュレーションを行った．データセットモデル（ホストにあらかじめ用意された，LOF の計算のために利用されるデータセット）と入力サンプル群（ネットワークを通して受信し，外れ値検出をされるデータセット）は以下のものを使用した．

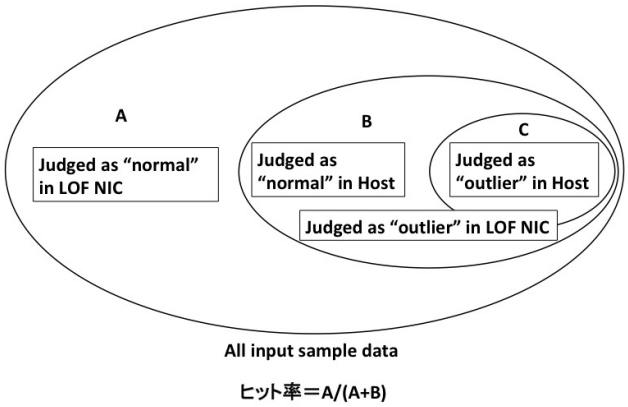

図 13 入力サンプルデータの処理結果とヒット率
Fig. 13 Relationship between processing result of input sample data and hit ratio.

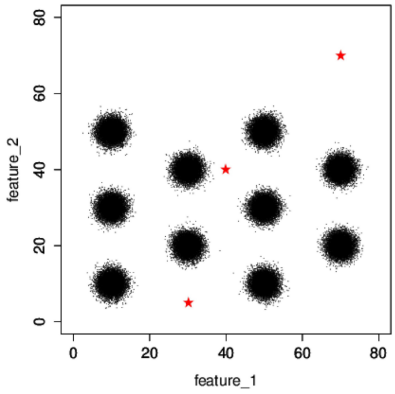

図 14 ホストのデータセットモデルと外れ値の分布
Fig. 14 Distributions of dataset model in host and outliers.

- 特徴数 2（32 bit floating point）
- データセットモデルは 10 個のガウス分布に基づくクラスタからなる．
- 各クラスタには 10,000 個のサンプルデータが含まれる（合計 100,000 サンプル）．
- 入力サンプル群はデータセットモデルのクラスタと同様のパラメータ（平均，分散）から生成したものに，3 つの外れ値を混入させたものを利用．

図 **14** に上述のデータセットモデルの分布を図示する．図中の X 軸と Y 軸はそれぞれの特徴量である（特徴数は 2 である）．図中の赤い星は意図的に混入された外れ値を表している．入力サンプル群として以下の 4 つの条件のものを用意した．

(1) 全クラスタのデータを均等に含むもの
(2) 特定の 1 クラスタに 90%のデータが集中したもの（図 **15**）
(3) 特定の 2 クラスタに 90%のデータが集中したもの（図 **16**）
(4) 特定の 3 クラスタに 90%のデータが集中したもの（図 **17**）

© 2016 Information Processing Society of Japan

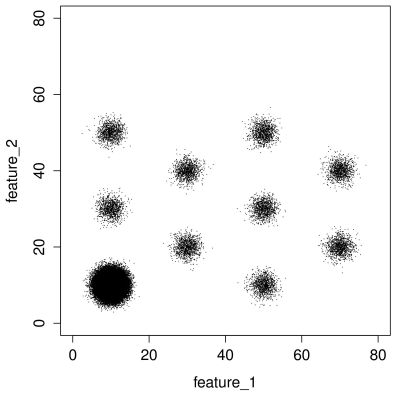

図 15 入力サンプル群が 1 クラスタに集中した際の分布
Fig. 15 Distribution of input sample data when concentrated in 1 cluster.

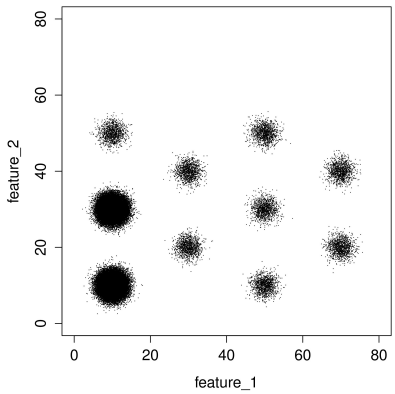

図 16 入力サンプル群が 2 クラスタに集中した際の分布
Fig. 16 Distribution of input sample data when concentrated in 2 clusters.

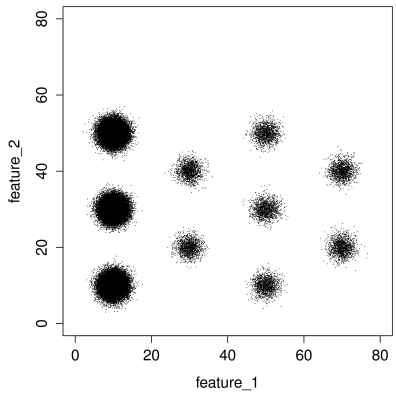

図 17 入力サンプル群が 3 クラスタに集中した際の分布
Fig. 17 Distribution of input sample data when concentrated in 3 clusters.

これらのデータセットを用いたうえで，LOF を用いた外れ値検出のパラメータは以下のように設定した．これらの設定は，図 14 に示す外れ値を正しく検出することができるように調整したものである．

- $k = 10$
- $thr = 20$（$thr < LOF$ ならば外れ値）

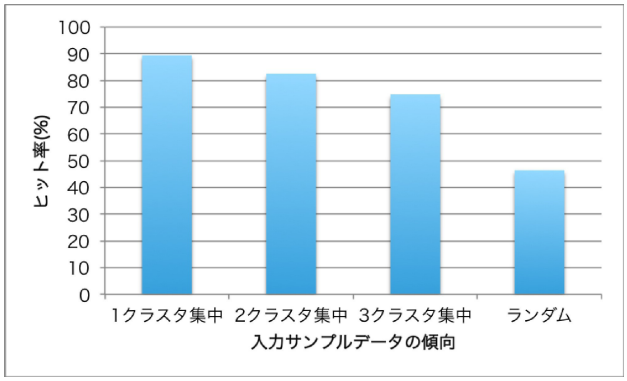

図 18 入力サンプルデータの傾向とヒット率
Fig. 18 Tendency of input sample data vs. hit ratio.

4.1.2 キャッシュヒット率

キャッシュのライン数を 128，1 ラインに 128 サンプルを含むデータセットを保持できるとき，入力されたサンプルデータが LOF NIC 内のキャッシュにヒットする確率（ヒット率）を調査した．シミュレーション結果を**図 18** に示す．

提案した設計では，LOF NIC 内のデータセットキャッシュがヒットしなかった場合，つまり，LOF NIC 内で入力サンプルデータが外れ値と検出された場合，そのデータはホストアプリケーションへと渡され，ホスト上に保持されている完全なデータセットを利用して LOF が計算される．つまり，ヒット率が高ければ高いほど，ホストアプリケーションによる LOF の計算負荷を軽減できる．たとえば，ヒット率が 90%ならば，ホストアプリケーションによる LOF の計算負荷を 90%削減できる．さらに，LOF NIC 内でパケットがフィルタリングされれば，その分だけサンプルデータを含むパケットをアプリケーション層に送るためのネットワークプロトコルスタックの処理も削減できる．

図 18 によると，少ないクラスタにサンプルデータが集中すればするほどヒット率は高くなっている．特に，1 クラスタに入力サンプルが集中しているときは，このシステムの性能が最大限に活かされ，ホストの CPU 負荷を 89.9%軽減できている．さらに，集中するクラスタの数が増えた場合，もしくは入力サンプルに局所性がない場合のヒット率の低下が比較的緩やかであることも，この LOF を用いた外れ値検出アルゴリズムがキャッシュシステムに適したものだということを示している．この評価で用意したキャッシュが保持できるサンプルの合計は，$128 \times 128 = 16{,}384$ サンプルであるため，ホストデータセット全体（100,000 サンプル）の 1/6 程度の情報しか保持することはできない．しかし，入力サンプルデータの本来の最近傍サンプルがキャッシュされていなかったとしても，キャッシュされているサンプルデータがある程度近い距離に存在し，その密度が近い値の場合，入力サンプルデータの LOF は外れ値ほど高いものにはならない．そのため，単純に対象デー

© 2016 Information Processing Society of Japan

タがキャッシュされているかいないかでヒット率が決まる通常のキャッシュシステムよりも，かなり高いヒット率を達成している．

また，この評価では与えた外れ値を正しく検出するために十分な値に *thr* を設定したが，*thr* をさらに上げれば，キャッシュされたデータセットが多少入力サンプルデータから離れていたとしても，明らかに外れていなければ *LOF* は *thr* より低い値となるため，LOF NIC 内で外れ値と判定される確率は上がり，ヒット率はさらに上昇する．ただし，外れ値でないパケットは NIC 内で破棄してしまうため，外れ値の可能性が少しでもあればホストで詳しく検査をした方がよいというアプリケーションも多いと考えられる．ここは精度と計算効率のトレードオフとなっている．

4.2 面積

提案システムの LOF を用いた外れ値検出モジュールついて，LUT（Look Up Table）と DSP（Digital Signal Processor）の使用率を評価する．

4.2.1 評価環境

論理合成の環境は以下のとおりである．

- Xilinx ISE Design Suite 13.4
- Virtex-7 XC7VX690T（NetFPGA SUME）

この評価は，NIC やフィルタリングに関するモジュールなどを含まず，LOF を用いた外れ値検出機構のみを対象としている．モジュール内に使用されている square root calculator，multiplexer および divider は Xilinx ISE に付属の Core Generator により生成した．また，4.1.2 項で考察したデータセットキャッシュのためのメモリ資源に関しては，すべて FPGA 内の BRAM で構成すると想定し，代わりに *LOF* の計算を行うために最低限の必要なデータ（キャッシュの center samples と 1 ライン分のデータ）のみを出力するモジュールをつなげた状態で評価をした．

4.2.2 面積

この LOF を用いた外れ値検出モジュールの面積には，

(1) 入力サンプルと距離が一番近い center sample を決定するための特殊な選択的ソートのための DSP Slice，レジスタおよび配線
(2) データセットの中から k から NEIGH_MAX 個の近傍サンプルを決定する選択的ソートのための DSP Slice，レジスタおよび配線

などが含まれている．

表 1 と**表 2** は，それぞれキャッシュのライン数と 1 ラインに含まれるサンプルデータ数を変えた際の，LUT および DSP の使用個数と Virtex-7 に対する使用率の変化について示したものである．

表 3 と**表 4** は，LOF を用いた外れ値検出モジュールを構成している主な 3 つのモジュール（図 6 における cache_select モジュール，k_dist_calc モジュール，lof_calc モジュール）と sqrt 回路について，それぞれの使用資源の個数と外れ値検出モジュール全体の使用資源（表 1，2）に対する割合を示したものである．表左の 128_128 および 64_64 は，キャッシュラインの数と 1 ラインに含まれるサンプル数を示している．

cache_select モジュールは (1) を含んでいるため，center sample の数（言い換えると，キャッシュラインの数）が増えると増加する．しかし，(1) は最小値を見つけるための特殊な選択的ソートであり，パイプラインの段数も k_dist_calc モジュールに比べて少ないため，必要とする資源は k_dist_calc モジュールと比較すると少ない．k_dist_calc モジュールは，(2) を含んでいるため，外れ値検出モジュールに使われる資源の多くがこのモジュールのために使用されており，必要とする資源は 1 ラインに含まれるサンプル数が増えると増加する．lof_calc モジュールは，データセットキャッシュの大きさには影響されない．このモジュールは，入力された対象サンプルと，前 2 つのモジュールによって決定された対象の近傍サンプルのデータのみを扱って *LOF* を計算するため，近傍サンプルの最大数（NEIGH_MAX）や近傍サンプルの最小数（k）などの他の

表 1 LUT 使用個数（使用率）
Table 1 Number of LUTs used (utilization).

		1 ラインに入るサンプル数	
		64	128
ライン数	64	69,638(16.1%)	116,723(26.9%)
	128	82,426(19.0%)	130,280(30.1%)

表 2 DSP 使用個数（使用率）
Table 2 Number of DSPs used (utilization).

		1 ラインに入るサンプル数	
		64	128
ライン数	64	1,046(29.1%)	1,588(44.1%)
	128	1,588(44.1%)	2,070(57.5%)

表 3 各モジュールの LUT 使用個数と外れ値検出モジュール全体に対する割合
Table 3 Number of LUTs used in each module and ratio for whole outlier detection module.

	cache_select	k_dist_calc	lof_calc	sqrt
128_128	17,276(13.3%)	84,621(65.0%)	2,782(2.1%)	7,120(5.5)
64_64	8,277(11.9%)	41,762(60.0%)	2,782(4.0%)	7,120(10.2%)

表 4 各モジュールの DSP 使用個数と外れ値検出モジュール全体に対する割合
Table 4 Number of DSPs used in each module and ratio for whole outlier detection module.

	cache_select	k_dist_calc	lof_calc	sqrt
128_128	1,024(49.5%)	1,024(49.5%)	22(1.1%)	0(0%)
64_64	512(48.9%)	512(48.9%)	22(2.1%)	0(0%)

© 2016 Information Processing Society of Japan

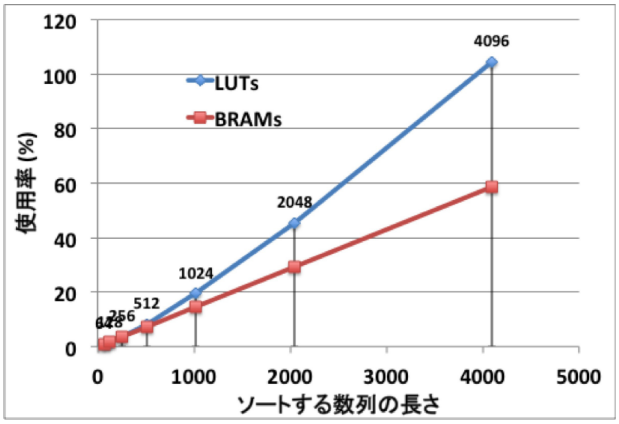

図 19 Sorting network IP core [6]
Fig. 19 Sorting network IP core [6].

パラメータに応じて必要資源が変化するが，データセットキャッシュの以外のパラメータが同一であれば，同一の回路となる．また，sqrt 回路も NEIGH_MAX にのみ影響される．具体的には，1 つの square root calculator は 455 個の LUT を消費するため，sqrt 回路全体で使用する LUT は 445× NEIGH_MAX 個と表すことができる．

2 章で示したとおり，LOF を用いた外れ値検出の FPGA へのオフロードが困難な大きな理由は 2 つ存在する．それは，(1) 巨大なデータセットを FPGA NIC 上で保持する必要があるということと，(2) データセットに含まれる全サンプルデータの数の長さのソートをオフローディングする必要があるということである．この 2 つの観点から，提案手法による面積の削減率について考察を行う．

(2) に関して，ソートのための回路はこの実装の一部に含まれている．そのうえで，4.1.2 項での評価によって，100,000 個のサンプルデータを含むデータセットモデルを扱う際に，ホストへの負荷を大幅に削減できる見込みが示された 128 × 128 のキャッシュサイズを持つ回路でも，LUT の使用率は 30.1%となっている．仮に，100,000 個のサンプルデータを FPGA 上でそのままソートしようとすると，膨大な資源が必要となってしまう．たとえば，**図 19** は Zuluaga らによって提供されている効率的なソーティングのための IP Core [6] の資源使用率を表しているものであるが，4,096 個のソートで LUT の使用率は 100%を超えている．100,000 個となれば，使用率が線形に増加していることを考えると，LUT の使用率は 2,000%を超える結果となってしまう．実際に LOF を用いた外れ値検出が使用するのは通常のソート回路ではなく，上位 k 個のサンプルを見つけるだけの回路で十分であるとはいえ，この結果から膨大な資源を必要とすることが容易に想像できる．本論文で提案したデータセットキャッシュは，ソートするサンプルの個数自体を削減することで，現在利用可能なデバイスに対しても実装可能な程度に資源使用率を削減している点がユニークである．

表 5 最大動作周波数
Table 5 Maximum operation frequency.

		1 ラインに入るサンプル数	
		64	128
ライン数	64	130.3 MHz	135.0 MHz
	128	133.9 MHz	132.0 MHz

(1) に関しては，この評価結果には含まれていない．そのため，キャッシュサイズによる必要メモリサイズの見積りについて考察をする．1 つのサンプルデータが含む情報は，データそのもの（32 bit×2）と *lrd*（式 (8)，32 bit）と *reach_dist*（式 (7)，32 bit）であるので，合計 16 Byte となる．そのため，128 × 128 のキャッシュを用意するためには，$16B \times 128 \times 128 = 262,144B$ のメモリが最低でも必要となる．このサイズは，4.1.2 項で示したように，ホストのデータセット全体の 1/6 程度の大きさであり，対象の Virtex-7 の BRAM（6,615 KB）上に十分確保可能な大きさである．

4.3 性能見積り

4.3.1 LOF NIC を用いた場合の性能向上率

本論文で提案する LOF NIC におけるサンプルデータの処理サイクル数は，k 個以上の近傍サンプル見つけるためのソート回路の実行サイクル数によって決まる．具体的には，NEIGH_MAX（3.2 節参照）cycles per Sample となる．そのため，キャッシュサイズやその他のパラメータにはサイクル数は影響されない．そこで，各キャッシュサイズにおける最大動作周波数から，スループットの見積りを計算する．評価環境は 4.2 節に準ずる．

表 5 に各キャッシュサイズの最大動作周波数を示す．本論文の評価では，$k = 10$ としているため，NEIGH_MAX を 16 と仮定すると，LOF NIC 単体のスループットの見積りは 8,000,000 Samples per sec 以上となる．

ただし，システム全体の性能は，ホストアプリケーションによる *LOF* の再計算によっても律速される．以下の環境で，ホストアプリケーションによる *LOF* の再計算処理の性能を測定した．

- R 言語，ライブラリ DMwR::lofactor を提案システムに合わせて直接的に拡張
- 2.5 GHz Intel Core i7
- 16 GB，1,600 MHz，DDR3
- OS X 10.9.5

表 6 にホストのデータセットモデルのサイズによる *LOF* の再計算処理の性能を示す．ホスト上ではデータセットに含まれるすべてのサンプルデータを対象にした *LOF* の計算が行われるため，データセットサイズが増えるごとにスループットは大幅に低下してしまう．

LOF NIC で外れ値と判定されたものは，このホストア

© 2016 Information Processing Society of Japan

表 6 ホストアプリケーションによる *LOF* の計算処理性能
Table 6 Throughput of outlier detection using LOF by host application.

データセットに含まれるサンプル数	スループット (Samples/sec)
1,000	3,585
10,000	528
100,000	49

プリケーションによる再計算を行わなければならない．つまり，ヒット率が 80%の場合，NIC 内で 100%のサンプルデータの *LOF* の計算が実行される間にホストでは 20%の *LOF* を計算しなければならない．

そのため，LOF NIC を用いて外れ値検出を行った場合のスループット性能 $T_{Proposal}$（単位は Samples/sec）は，以下の式で表すことができる．

$$T_{Proposal} = \min\{T_{NIC}, T_{Host}/(1 - P_{Hit})\} \quad (10)$$

ただし，ホストアプリケーションによる外れ値検出のスループットを T_{Host}，LOF NIC 単体による外れ値検出のスループットを T_{NIC}，そのヒット率を P_{Hit}（0〜1）とする．$\min\{A, B\}$ は変数 A，B のうち小さい値を表す．

また，ホストアプリケーションのみで外れ値検出を行った場合に対する，LOF NIC を用いて外れ値検出を行った場合の性能向上率（スループットの上昇率）は，以下の式で表すことができる．

$$T_{Proposal}/T_{Host} = \min\{T_{NIC}/T_{Host}, 1/(1 - P_{Hit})\} \quad (11)$$

本論文の評価では，T_{NIC} は 8,000,000，T_{Host} は 49〜3,585，P_{Hit} は 4.1 節より約 0.9〜0.45 である．このとき，T_{NIC}/T_{Host} は 163,265〜2,231，$1/(1-P_{Hit})$ は 10.0〜1.82 であるため，性能向上率は単純にヒット率から計算可能であり，10〜1.82 倍の性能向上を達成できたといえる．

4.3.2 データセットキャッシュ機構をソフトウェアのみで実現した場合の性能向上率

本論文では，ホスト上に存在する大量のデータの一部を NIC 内にキャッシュすることで，*LOF* の計算に必要なソートの長さを削減している．この「データの一部をキャッシュすることで，ソートにかかる計算コストを削減する」という提案は，ハードウェアによる実装を可能にするだけでなく，ソフトウェアのみで実装する際にも有効である．つまり，NIC 内では特別な処理を行わず，ホスト上にデータセットキャッシュを設け，到着したサンプルデータに対して，そのキャッシュのデータのみで *LOF* を計算し，外れ値と検出された場合のみ全サンプルが含まれるデータセットを用いて *LOF* を再計算するシステムを構築することも可能である．この項では，このデータセットキャッシュ機構をソフトウェアのみで実現したシステムを用いた場合の性能について評価し，本来のシステム（到着したサンプルデータの *LOF* を最初から全サンプルデータが含まれるデータセットを用いて計算するシステム）および LOF NIC を用いたシステムと比較を行う．

表 7 データセットキャッシュを用いたホストアプリケーションによる *LOF* の計算処理性能
Table 7 Throughput of outlier detection using LOF by host application with dataset cache.

		1 ラインに入るサンプル数	
		64	128
ライン数	64	5,090 [Samples/sec]	4,695 [Samples/sec]
	128	4,937 [Samples/sec]	4,558 [Samples/sec]

表 7 に，ホスト上のデータセットキャッシュを利用した，ホストアプリケーションによる *LOF* の計算処理の性能を示す．評価環境は 4.3.1 項に準ずる．表 7 に示した結果には，全サンプルが含まれたデータセットを用いた *LOF* の再計算は含まれていない．キャッシュされたデータセットの中から利用するデータセットを決定するための計算と，そのデータセットから入力サンプルの *LOF* を求めるための計算のみが含まれている．そのため，全サンプルが含まれたデータセットの大きさは性能に関係せず，キャッシュの大きさによって性能が影響される．また，LOF NIC を用いた場合と同様に，*LOF* の再計算はキャッシュを利用した *LOF* の計算とは独立に行うと想定する．

ソフトウェアアプリケーションのみで実現した場合のシステムも，LOF NIC を用いた場合と同様に，式 (10) と式 (11) で性能と性能向上率が求められる．具体的には，式 (10) と式 (11) の T_{NIC} をホスト上のデータセットキャッシュを利用した，ホストアプリケーションによる *LOF* の計算処理の性能に置き換えればよい．ただし，LOF NIC を用いた場合はホストアプリケーションによる *LOF* の再計算処理が性能のボトルネックとなっていたことに対して，ソフトウェアのみで実現した場合は条件によってボトルネックとなる処理が変化する．たとえば，データセットに 10,000 個のサンプルが含まれており，キャッシュラインの数が 128，1 ラインに含まれるサンプル数が 128 のとき，性能は式 (10) より $\min\{4558, 528/(1-P_{Hit})\}$ [Samples/sec] なので，P_{Hit} が 0.9 であれば性能は 4558 [Samples/sec] となりキャッシュを利用した *LOF* の計算そのものがボトルネックとなるが，P_{Hit} が 0.75 であれば性能は 2112 [Samples/sec] となり全サンプルを含むデータセットを利用した *LOF* の再計算がボトルネックとなる．

性能のボトルネックが全サンプルを含むデータセットを利用した *LOF* の再計算処理の場合，性能向上率は LOF NIC を用いた場合と同等になるが，ソフトウェアアプリケーションによるキャッシュを利用した *LOF* の計算がボトルネックとなる場合の性能向上率は LOF NIC よりも低くなる．LOF NIC を用いた場合よりも性能向上率が高くな

© 2016 Information Processing Society of Japan

ることはないが，ソフトウェアのみでデータセットキャッシュ機構を利用したLOFを用いた外れ値検出を行いたい場合は，式(11)に基づいて性能のボトルネックがどちらになるかを見積もることでLOF NICを用いた場合に比べて性能向上率が劣るかどうかを予測することができる．

4.4 LOF NICを利用することによる精度への影響

LOF NIC内に構築されている外れ値検出モジュールは，サンプルデータの一部を利用して*LOF*の計算を行うため，本来のLOFを用いた外れ値検出アルゴリズムを用いて計算したときとは異なる結果になる可能性がある．具体的には，false positive（本来のアルゴリズムでは外れ値でないと判定されるサンプルを，LOF NIC内で外れ値であると判定してしまう場合）と，false negative（本来のアルゴリズムでは外れ値であると判定されるサンプルを，LOF NIC内で外れ値でないと判定してしまう場合）が発生する可能性がある．false positiveが発生した場合，NIC内で外れ値と判定されたサンプルはホストアプリケーションへと渡され，本来のアルゴリズムに則って，つまり，全サンプルを含むデータセットを用いて*LOF*の再計算が行われる．そのため，false positiveはシステム全体で考えると発生しないと考えてよい．これに対して，false negativeが発生した場合，NIC内で外れ値でないと判定されたサンプルはそのままNIC内で破棄されてしまうため，*LOF*を再計算する機会は与えられない．そのため，false negativeが発生した場合のみ，本来のアルゴリズムで外れ値検出を行った場合とLOF NICを用いた提案システムの精度の差が発生する．

しかし，実際にfalse ngativeが起きる可能性は低いと予想される．本論文では，ヒット率の評価を行うため，図14に示される，10,000個のサンプルからなるクラスタが10個集合した100,000個のサンプルを含むデータセットに対して，各クラスタと同じモデルで発生させた100,000個のサンプルを入力し，キャッシュを用いた一部のデータセットで用いた計算で外れ値でないと正しく判定されるかどうかをヒット率として評価した（100,000個の入力サンプルが，ホストアプリケーションによって外れ値でないと判定されることはあらかじめ確認した）．この100,000個の入力サンプルとは別に，故意に生成した外れ値を3個（図14の星マーク）追加で入力し，各外れ値がホストアプリケーションによって外れ値と判定されることを確認したうえで，一部のデータセットを用いた計算でも外れ値と判定されることを確認した．つまり，我々の実験ではfalse negativeが観察されなかった．また，その発生条件は1) データセットの一部を用いた場合は，周囲のサンプルとの密度の差が小さく，*LOF*が低くなり，かつ2) データセット全体を用いた場合は，周囲のサンプルとの密度の差が大きく，*LOF*が高くなる場合である．false positiveは，これらの発生条件が逆なので，入力サンプルの周囲のデータがキャッシュされていない場合に容易に発生する．これに対して，faslenegativeに関しては，限定的な状況だと考えられる．本論文での評価ではfalse negativeが1度も観察されなかったこと，その発生条件が限定的なことから，発生確率は低いと予想されるが，数学的証明に関しては今後の課題とし，本論文では予想を行うにとどめる．

以上，false negativeが発生したときのみ精度が変化し，そのfalse negativeの発生確率は低いと予想されることから，LOF NICを用いた提案システムが精度に与える影響は低いと予想される．

5. 結論

本研究の目的は，外れ値検出精度は高いものの，これまでハードウェアへのオフロードが検討されてこなかったLOFアルゴリズムをFPGA NIC上に実現し，外れ値フィルタリングを行うことである．これによって，受信したセンサデータを効率的に選択・取得することができ，ネットワークプロトコルスタックやデータマイニングアプリケーションによるホストのCPU負荷を大幅に削減できる．しかし，LOFアルゴリズムはその計算の複雑さや必要とするデータセットモデルの量が膨大であることから，NICへのオフロードは挑戦的な課題であった．そこで，本論文ではデータセットモデルの一部をFPGA NIC内にキャッシュすることでこの問題の解決を試みた．

本LOF NICでは，入力されたサンプルデータがLOF NIC内で外れ値と判定された場合，そのデータはホストアプリケーションへと渡され，ホスト上に保持されている完全なデータセットを利用して*LOF*が再計算される．つまり，ヒット率（外れ値でないサンプルデータが，LOF NIC内で外れ値でないと正しく判定されフィルタリングされる確率）が高ければ高いほど，ホストアプリケーションによる*LOF*の計算負荷を軽減できる．

データキャッシュを有する提案LOF外れ値検出機構を論理合成し，面積や動作周波数について評価し，Xilinx Virtex-7 FPGA上に実現できることを示した．100,000サンプルを含むデータセットモデルを用いたシミュレーションの結果，45%〜90%のデータがLOFによる外れ値フィルタリングNICのキャッシュにヒットし，NICによる外れ値検出を実現できた．これは，外れ値検出をすべてソフトウェアで実行した場合に比べて1.82倍〜10倍の外れ値検出処理スループット向上に相当する．

謝辞 本研究の一部は，JST戦略的創造推進事業さきがけ「多様な構造型ストレージ技術を統合可能な再構成可能データベース技術」の補助による．

参考文献

[1] Alshawbkeh, M., Jang, B. and Kaeli, D.: Accelerating the Local Outlier Factor Algorithm on a GPU for

© 2016 Information Processing Society of Japan

Intrusion Detection Systems, *Proc. 3rd Workshop on General-Purpose Computation on Graphics Processing Units* (*GPGPU'10*), pp.104–110 (2010).
[2] Breiman, L.: Random Forests, Machine Learning 45, pp.5–32 (2001).
[3] Breunig, M.M., Kriegel, H.-P., Ng, R.T. and Sander, J.: LOF: Identifying Density-Based Local Outliers, *Proc. ACM SIGMOD International Conference on Management of Data*, pp.93–104 (2000).
[4] Essen, B.V., Macaraeg, C., Gokhale, M. and Prenger, R.: Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA?, *Proc. IEEE International Symposium on Field-Programmable Custom Computing Machines* (*FCCM'12*), pp.232–239 (2012).
[5] Hayashi, A., Tokusashi, Y. and Matsutani, H.: A Line Rate Outlier Filtering FPGA NIC using 10GbE Interface, *Proc. International Symposium Highly-Efficient Accelrators and Reconfigurable Technologies* (*HEART'15*) (2015).
[6] Zuluaga, M.: Sorting Network IP Generator, available from ⟨http://www.spiral.net/hardware/sort/sort.html⟩.
[7] Pokrajac, D., Lazarevic, A. and Latecki, L.J.: Incremental Local Outlier Detection for Data Streams, *Proc. IEEE Symposium on Computational Intelligence and Data Mining* (*CIDM'07*), pp.504–515 (2007).
[8] the NetFPGA team: The NetFPGA Project, available from ⟨http://netfpga.org/⟩.
[9] Zuluaga, M., Milder, P. and Püschel, M.: Computer Generation of Streaming Sorting Networks, *Proc. 49th Annual Design Automation Conference* (*DAC'12*), pp.1245–1253 (2012).
[10] 小林諒平，吉瀬謙二：FPGA ベースのソーティングアクセラレータの設計と実装，電子情報通信学会技術研究報告 CPSY2015-5，Vol.115, No.7, pp.25–30 (2015).

林 愛美 （学生会員）

2015 年慶應義塾大学理工学部情報工学科卒業．現在，同大学大学院理工学研究科開放環境科学専攻前期博士課程在籍．電子情報通信学会学生会員．

徳差 雄太

2014 年慶應義塾大学環境情報学部卒業．現在，同大学大学院理工学研究科修士課程在籍中．コンピュータアーキテクチャおよび FPGA システムに関する研究に従事．

松谷 宏紀 （正会員）

2004 年慶應義塾大学環境情報学部卒業．2008 年同大学大学院理工学研究科後期博士課程修了．博士（工学）．2009 年度より 2010 年度まで東京大学大学院情報理工学系研究科特別研究員，日本学術振興会特別研究員（SPD）．2011 年度より慶應義塾大学理工学部情報工学科専任講師．コンピュータアーキテクチャとインターコネクトに関する研究に従事．IEEE，電子情報通信学会各会員．

© 2016 Information Processing Society of Japan