ウィンドウ結合演算子のFPGAによる実現

三好　健文[†a)]　寺田　祐太[††b)]　川島　英之[†††c)]　吉永　努[†d)]

Realizing Window Join Operator by Using FPGA

Takefumi MIYOSHI[†a)], Yuta TERADA[††b)], Hideyuki KAWASHIMA[†††c)], and Tsutomu YOSHINAGA[†d)]

あらまし　本論文では，ストリームデータのウィンドウ結合をFPGA上に実現する手法とアーキテクチャを提案する．提案アーキテクチャは二つのストリームデータの処理を並列に行い，かつそれをパイプライン処理することで高性能化を実現する．提案アーキテクチャは，2^{16}個のタプルに関するウィンドウ結合を実現し，1 ms周期で発生するストリームデータを処理できることを示す．高性能化に起因して，出力結果タプルが欠落する問題が生じることを述べ，提案するアドミッション制御機構によりその問題を防ぐことを示す．

キーワード　FPGA，ストリームデータ処理，ウィンドウ結合，アクセラレータ

1. まえがき

不正アクセス検知やウイルス検知等のトラフィックにおける異常検知を行うために，IPパケットストリームへの関係処理の適用がストリームデータ処理というフレームワークで研究されている[1]～[7]．ストリームデータ処理では，連続的に到達するデータに対して取り零しなくクエリ処理を実行することが求められる．ルータにおけるIPパケットのようにデータ到達頻度が高ければ，求められる計算能力も高くなる．

ストリームデータ処理における最重要研究課題の一つに，高性能化が挙げられる．その実現のために，FPGAを用いてストリームデータ処理を高性能化する研究[8]～[11]が近年行われている．このうち，Streams on Wires [8]はFPGA上に所望のストリームデータ処理グラフを構築するためのビルディングブロックであるAlgebraと呼ばれる演算要素を定義している．Algebraには，選択，射影，集約，和，グルーピングが含まれる．

Streams on Wiresが提供しない関係演算子もある．その中に，ストリームデータ処理における最重要演算子の一つであるウィンドウ結合[12]がある．ウィンドウ結合とは，ユーザの条件に応じて二つのストリームデータを結合する演算である．ストリームデータ処理の主目的は異常検知であるが，ウィンドウ結合は異常検知のために重要な役割を果たす．ウィンドウ結合を用いた異常検知として，DoS(Denial of Service)検知が挙げられる．例えば，筑波大と電通大のトラヒックに対する同一ポートへの同時大量アクセスを，代表的なストリーム問合せ言語CQL(Continuous Query Language) [3]で記述すると，図1のようになる．この問合せでは，tsukuba[ROWS 100], uec[ROWS 100]がそれぞれ筑波大と電通大の直近100個のパケットストリームデータを表す．そして両ストリームデータを

```
SELECT uec.dstport, COUNT(*)
FROM tsukuba[ROWS 100], uec[ROWS 100]
WHERE tsukuba.dstport = uec.dstport
GROUP BY uec.dstport;
```

図1　DoS検知用問合せ
Fig. 1　A Query for DoS.

† 電気通信大学大学院情報システム学研究科，調布市
Graduate School of Electro-Communications,The University of Electro-Communications, 1–5–1 Chofugaoka, Chofu-shi, 182–8585 Japan
†† 電気通信大学電気通信学部，調布市
Faculty of Electro-Communications,The University of Electro-Communications, 1–5–1 Chofugaoka, Chofu-shi, 182–8585 Japan
††† 筑波大学大学院システム情報工学研究科，つくば市
Graduate School of Systems and Information Engineering, University of Tsukuba, 1–1–1 Tennodai, Tsukuba-shi, 305–8577 Japan
a) E-mail: miyoshi@is.uec.ac.jp
b) E-mail: terada@comp.is.uec.ac.jp
c) E-mail: kawasima@cs.tsukuba.ac.jp
d) E-mail: yosinaga@is.uec.ac.jp

ウィンドウ結合し，ポート番号ごとにカウントした結果を返す．

ストリームデータは無限に到着するため，ストリームデータ同士を結合することはできない[3]．そのため，論理的には無限長のストリームデータを有限長のリレーションに変換し，リレーションに対して結合演算を施す．ウィンドウ結合には，リレーションのサイズを指定する方式として，データ個数による方式 (Tuple based Window) と，時間による方式 (Time based Window) がある．本論文ではデータ個数を指定する方式 (Tuple based Window) のみを取り上げて議論する(注1)．

ウィンドウ結合を FPGA 上に実現するためには二つの課題がある．第一の課題は，ウィンドウ結合において並列性を見出すことである．FPGA 上で効率良くウィンドウ結合を実行するためには，その並列処理技術を開発すべきである．しかし，そのような技術は我々が知る限り存在しない．

第二の課題は出力タプルによる外部インタフェース通信容量の圧迫である．ウィンドウ結合は，結合選択率によって入力データ量よりも多くのデータを出力する場合がある．出力されるデータが多すぎると，FPGA から外部へのインタフェースへの通信容量が不足することがある．このとき，出力されるデータがインタフェースへ送り込まれる前に，新たな結果データにより上書きされてしまい，欠落してしまう可能性がある．

本論文では，これらの課題を解決する手法と，その手法を実現するアーキテクチャを提案する．また，提案アーキテクチャを FPGA 上で実現し，その性能を実験的に評価する．我々が知る限り，本論文は FPGA によるウィンドウ結合の実現に関する初めての論文である．

本論文の構成は次のとおりである．**2.** ではウィンドウ結合について説明する．**3.** では研究課題を述べる．**4.** では研究課題に対する新しい手法を提案し，更にそれらの手法を実現するアーキテクチャを **5.** で提案する．**6.** では提案を実験を通して評価する．最後に **7.** でまとめを行う．

2. ウィンドウ結合

ウィンドウ結合 [3], [12] とは，二つのストリームデータについてユーザが指定したフィールドを比較し，条件を満足するタプル同士を結合する演算である(注2)．タプルとはストリームデータの一行のデータを指し，比較するフィールドは結合キーと呼ばれる．ウィンドウ結合の処理では，まずユーザが指定した数以下のタプルがキューに保存される．この数は図 1 で示されるように ROWS という変数により指定される．そして片方のタプルが到着すると，もう片方のキューと照合処理を行い，その結果を出力する．キューが一杯であるときに新たなタプルが到着すると，最古タプルがキューから押し出されて破棄される．結果タプルの数は結合選択率により変動する．例えばストリーム A からタプルが到着し，B のキューについて照合処理を行うとき，B のキューサイズが k タプルであれば，結果タプル数は 0 から k までの値をとり得る．結果タプル数が 0，k のとき，結合選択率はそれぞれ最小，最大である．

図 2 に，ウィンドウ結合におけるタプルの制御を示す．図 2 では，二つの情報源 A と B から到着するストリームをウィンドウ結合する様子を表している．図中の括弧順に説明を行う．(1) “Stream A” からタプル “α” が一つ到着する．タプル “α” は “Volume” フィールドが “a9” であり，結合キーを有する “Key” フィールドが “05” である．(2〜3) タプル “α” は “Queue B” と照合処理を行う．(4) 照合処理で得られた結果タプルが出力される．(5) タプル “α” は “Queue A” に

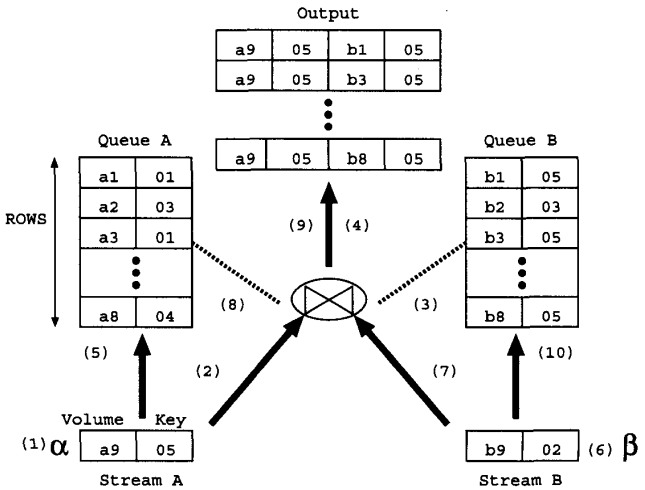


図 2 逐次的なウィンドウ結合用のタプル制御
Fig. 2 Tuple control for sequential window join.

(注1)：Time based Window と Tuple based Window の違いはリレーションサイズの可変性にある．その可変性は時間をタプル数に変換することにより解決される．この変換作業は容易だが，本論文ではウィンドウ結合の FPGA 化に関する議論に集中するため，Tuple based Window のみを取り上げる．
(注2)：本論文では等結合 (equi-join) のみ扱い，θ 結合は扱わない．

挿入される．キューは一杯であるから，これに伴い最古タプルが破棄される．(6～10) “Stream B” からタプル “β” が到着すると，同様な処理が行われる．

3. 研究課題

ウィンドウ結合をFPGAに実現するためには，二つの課題がある．下記でこれらを述べる．

3.1 第一の課題：並列化技法

第一の課題はウィンドウ結合の並列化技法を見出すことである．FPGAを用いた専用ハードウェアでウィンドウ結合を高速化するためには，並列処理が必要である．しかしウィンドウ結合のFPGAによる並列化技法は，我々が知る限り存在しない．ここでは，通常行われているウィンドウ結合の流れを述べる．この流れは逐次的である．

通常のウィンドウ結合の流れを図3に示す．図3の説明を以下に述べる．時刻 “t0” で “stream A” から到達したタプル “a” を受信する．時刻 “t1” から照合処理が始まる．照合処理とは，結合キーが等しい入力タプルとキュー内タプルを結合して結果タプルを作成する処理を指す．照合処理により得られた結果タプルは外部インタフェースへ出力される．この場合には，照合処理はBキュー内の全タプルについて行われる．時刻 “t2” で全てのキュー内タプルとの最後の照合処理が実行される．なお，Bキュー内のタプル数はNとしてあり，“B1” はBキューの先頭タプル，“Bn” はBキューの末尾タプルを表す．その後，時刻 “t3” で “a” を “queue A” に挿入する．“stream A” と “stream B” の到着率は等しいとは限らないため，タプルの受信は交互に行われるとは限らない．また，二つのキューのROWSの値が等しいとは限らない．

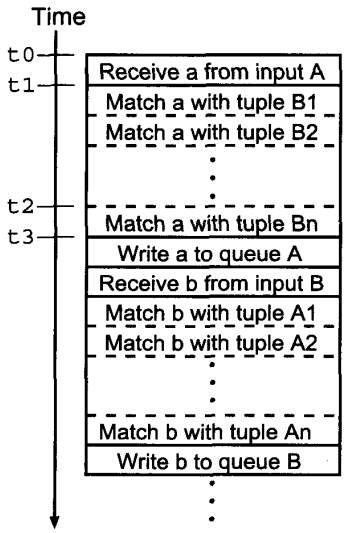


図3 逐次的ウィンドウ結合の流れ
Fig. 3 Flow of sequential window join.

図3に示したウィンドウ結合の流れは，入力タプルとキュー内タプルとの照合処理と，それにより発生した結果タプルの出力処理を逐次的に行っている．これではFPGAを用いた性能向上を実現できない．ウィンドウ結合の並列化技法を新たに提案する必要がある．

3.2 第二の課題：結果欠落

第二の課題は，結果タプルが大量に生成された場合に，それらをFPGAから外部へ送信するインタフェースの容量があふれることである．

FPGAを用いたウィンドウ結合を行うためには，FPGAの内外間でデータを送受信するインタフェースが必要になる．ウィンドウ結合において並列化が成功したとすると，生成される結果タプルの量は，入力タプルの量よりも増大することになる．そのとき，結果タプルを滞りなく出力するためには，通信インターフェイスには十分な通信容量が必要である．十分な通信容量が確保されない場合，結果タプルは欠落する可能性が生じる．このように結果タプルが欠落してしまう状況を回避する必要がある．

4. 提案手法

前章で述べた2課題を解決手法を本章では提案する．

4.1 第一の課題（並列化）に対する提案手法

入力タプルとキュー内タプルとの照合処理を並列に行う方式を提案する．提案は結合処理の並列化と，パイプライン化の二つから構成される．

4.1.1 結合処理の並列化

まず，ウィンドウ結合の並列化について述べる．FPGA上では，図2における “source A” と “queue B” との照合処理を行う回路と，“source B” と “queue A” との照合処理を行う回路を独立して構成できる．この場合，1サイクル当りに生成される結果タプルの数は最大で(注3)二つとなる．この並列化ウィンドウ結合の流れを図4に示す．

図4に示される並列化ウィンドウ結合は次のように動作する．時刻 “t0” で “a” を受信する．時刻 “t1” で “a” を “queue A” に書き込むと同時に “b” を受信する．時刻 “t2” からは照合処理を二並列で行う．これにより，1サイクル当りに生成される結果タプル数

(注3)：結合選択率が100%の場合を指す．

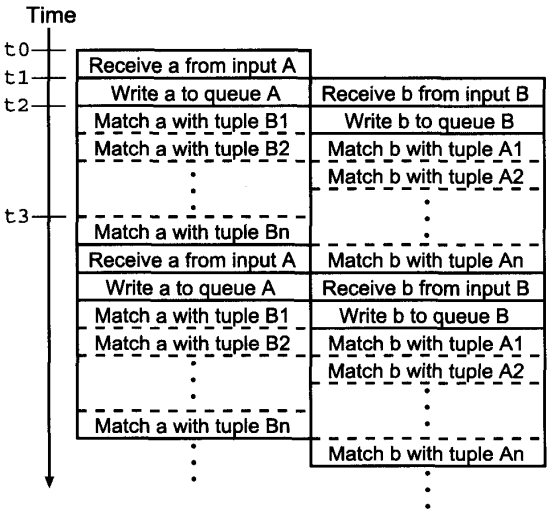


図 4　並列化ウィンドウ結合の流れ
Fig. 4　Flow of parallelized window join.

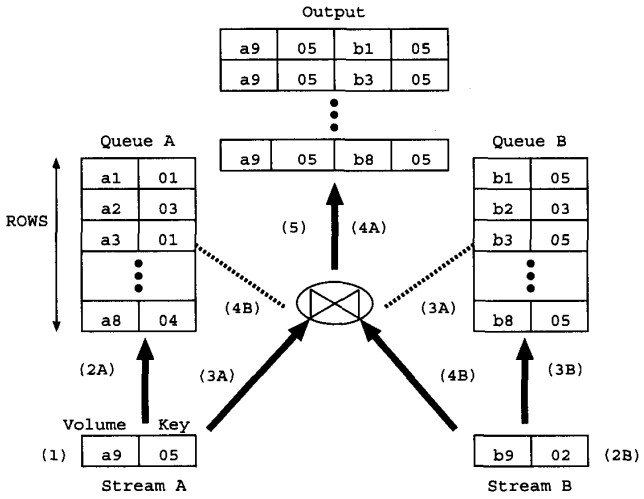


図 5　並列化ウィンドウ結合用のタプル制御
Fig. 5　Tuple control for parallelized window join.

は 2 となる．図 3 に示される逐次方式では，1 サイクル当り生成される結果タプル数は 1 だから，提案方式の結合処理速度は逐次方式の 2 倍となる．

図 4 に示される並列化ウィンドウ結合を実現するには，ウィンドウ結合におけるタプル制御を変更する必要がある．なぜならば最新のタプル “a” と “b” 同士の照合処理を行う必要があり，それを実現するには “a” と “b” が共にキュー内に存在する必要があるからである．並列化ウィンドウ結合用のタプル制御を図 5 に示す．図中の数字はステップ数を表し，数字に続くアルファベットはそのステップで動作する側のストリームを表す．例えば (2A) はステップ 2 におけるストリーム A の動作を表す．並列化ウィンドウ結合用タプル制御方式（図 5）が逐次的ウィンドウ結合用タプル制御方式（図 2）と異なる点は，上述のように，新規タプルは到着すると即座にキューへ書き込まれる点である．これは図 5 のステップ (2A) 及び (3B) に示されている．

ここで注意されるべきは，並列化ウィンドウ結合用のタプル制御（図 5）をナイーブに実現しては，ウィンドウ結合が正しく行われない点である．例えば “stream A” から “a, b” という二つのタプルが到着し，“stream B” から “1, 2” という二つのタプルが到着する状況を考える．到着順序は a, 1, b, 2，キューサイズは 1，当初キューは空だとする．このとき，正しい結果は “{ a, 1}, { b, 1 }, { b, 2 }” である．しかし，並列化ウィンドウ結合用のタプル制御をナイーブに適用すると，この結果は “{ a, 1 }, { b, 2 }” となる可能性がある．この場合，“{ b, 1 }” が消失するため，ウィンドウ結合の結果は正しくない．

この問題を解決するために，照合処理である (1)“Match a with B1” とストリームから新たに受理したタプルのキューへの書込み処理である (2)“Write b to queue B” が同期順序回路の同一クロック内で同期して処理されるように設計する．(1) と (2) の処理が同期順序回路の同一クロック内の処理として設計されるとき，(1) の処理で B1 の値を読み出す前に (2) の書込みが完了することは起こり得ない．すなわち，(1) の処理では，処理が開始した直後の時刻 t2+Δ でキューに保存されている B1 の値の読み込みを実行し，その処理を完了する．一方，(2) の処理では，時刻 t2+Δ でキュー B に受理したタプル b の書込み処理を開始するが，実際に書込みが完了するのは，時刻 t3 の直前 (>t2+Δ) である．

これにより，論理的な処理の順序である (1) → (2) が維持され，前段落で述べた消失問題は解決される．同様に，照合処理 “Match b with A1” とキューへの書込み処理 “Write a to queueA” が同一クロック内で同期して実行されるよう設計することで，“stream B” から到達するタプルが “stream A” から到着するタプルに先行する場合にも，データの読み書きの論理的順序を維持できる．

4.1.2　パイプライン化

キュー内タプルの読込みと，二つのタプルのマッチと結果タプルの出力は，いずれも 1 サイクルで実行可能である．これに基づいて我々はタプル制御をパイプライン化した．パイプライン化されたタプル制御の

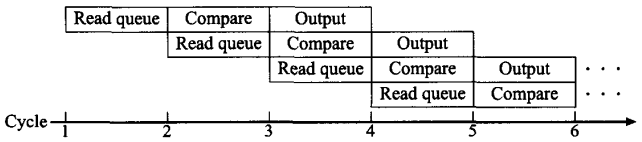


図 6　パイプライン化された並列ウィンドウ結合
Fig. 6　Pipelined parallel window join.

流れを図 6 に示す．図 6 に示されるように，パイプライン化により，キューからのタプル読込み (“Read queue”)，結合キー比較 (“Compare”)，そして結果タプル出力 (“Output”) の 3 処理を 1 サイクルで実行可能になる．

4.2　第二の課題（結果欠落）に対する提案手法

必要な通信容量が不足した場合，生成された結果タプルが外部インタフェースへ送出されずに破棄されてしまう．したがって，ユーザは間違った結果を得ることになってしまう．

この問題を解決するため，我々はアドミッション制御 [13] を適用する．すなわち，システムに新しく到着したタプルをシステム内部に入れずに破棄することで，結果タプルがあふれてしまう状況を回避する．アドミッション制御によりユーザに届けられるデータのレートは低下するが，生成されたタプルは欠落することなくユーザに届くようになる．

アドミッション制御を我々のシステムにおいて実現するために，ウィンドウ結合処理を一時的に停止する手法を提案する．この停止処理により，結果タプルの生成量が抑制される．そしてこの抑制により，結果タプルの欠落が防がれる．

5.　提案手法を実現するアーキテクチャ

本章では，前節で提案した手法を実現するシステムのアーキテクチャを提案する．提案アーキテクチャを図 7 に示す．このアーキテクチャは，(1) ストリームタプルを入力する入力層 (Input layer)，(2) 受信したタプルをキューに保存し，他方のキュー内タプルと結合処理を行う照合層 (Match layer)，そして (3) 結果タプルを出力する出力層 (Output layer)，の 3 層に分かれている．FPGA の並列性により，各層を構成する回路は並列に動作することができる．

提案するアーキテクチャは二つのストリームの入力を対象とする．三つ以上のストリームに対する結合演算を行う場合には提案アーキテクチャを階層的に組み合わせることで対処できる．**4.1.1** に示したように，

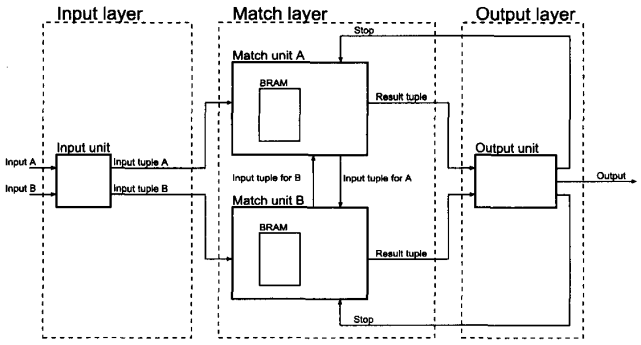


図 7　ウィンドウ結合モジュールのブロックダイアグラム
Fig. 7　Block diagram of window join module.

提案するアーキテクチャでは入力ストリームの到着順によらず処理の順序関係が維持され，タプルの消失は発生しない．したがって，三つ以上のストリームを処理するためにどのように組み合わせた場合でも，タプルが消失することなく正しく結合演算が実現される．

以下，各層について説明する．

5.1　入力層 (Input layer)

入力層は一つの入力ユニットからなる．入力ユニットは，ストリームデータのタプルを受信し，新たなタプルを受信するまでの一時的な保存を行う．入力ユニットは他の層の回路と並列に動作し，常に新たなタプルを受信する．保存された入力タプルは，前の入力タプルのマッチを終えた照合層に読み込まれるが，このとき，読み込まれたタプルを消去することはなく，新たなタプルを受信するまで保存されたままとなる．ここで，マッチ層は前の入力タプルと同一の入力タプルの読込みを制限する構成となっているため，新たな入力タプルの受信による過去の入力タプルの更新がなかった場合でも，同一の入力タプルに対して二度以上の照合処理を行うことはない．

5.2　照合層 (Match layer)

照合層は，入力 A と入力 B の照合処理を行う二つの照合ユニット (match unit A，match unit B) と，各入力タプルを保持する二つのキュー (queue A，queue B) で構成される．照合ユニット A では，入力 A からのタプルと，キュー B 内のタプルとの照合処理を行う．照合ユニット B では，入力 B からのタプルと，キュー A 内のタプルとのマッチを行う．キュー A は，照合ユニット B からのみデータの読み書きが行われる．キュー B は，照合ユニット A からのみデータアクセスが行われる．

図 8 に，パイプライン処理を行う照合ユニット内の

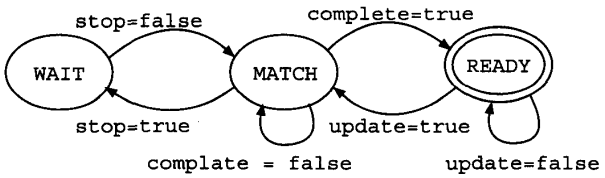


図 8 結合処理を行うステートマシンの状態遷移図
Fig. 8 State machine diagram of join operation.

ステートマシンの状態遷移を示す．ステートマシンはまず，READY において入力層のキューに保存されている入力タプルを読み込む．入力タプルが更新されている場合 (updated が真の場合)，照合処理を開始するキュー内タプルのアドレスを指定し，MATCH へ遷移する．MATCH では，結合の一連の処理，すなわち，(1) キュー内タプルと入力タプルのキー値の比較，(2) 前サイクルの結果タプルの出力層への書き出し，(3) 次サイクルで読み込むキュー内タプルのアドレスの指定，を同時に行う．これにより，照合処理の一連の動作はパイプライン処理される．

ステートマシンは結果タプル欠落を防ぐための機構も有する．MATCH 時に，出力層が照合処理を一時停止させるための信号 (stop) を真にセットした場合，WAIT へ遷移する．WAIT では照合処理は行われない．stop が偽になると再び MATCH に遷移する．全てのキュー内タプルとの照合処理を完了する (complete が真になる) と READY に遷移する．

提案アーキテクチャは **4.1.1** 最後に述べた，順序制御機構も有する．照合処理が行われている間はキューは更新されてはならない．なぜなら，照合処理の対象となるキュー内タプルは，入力タプルが到達した時点でキューに格納されている必要があるからである．そこで，照合処理が行われていないとき (ステートマシンが READY のとき) にのみキュー更新を許可する機構を提案アーキテクチャは有する．ステートマシンが READY 以外のときは，キュー更新は待たされる．これにより，順序制御が実現される．

5.3 出力層 (Output layer)

出力層は一つの出力ユニットで構成される．このユニットは照合層から出力された出力タプルの出力を行う．提案アーキテクチャでは，二つのストリームデータに対して並列に照合処理を行う．つまり，1 サイクルで最大二つ(注4)の結果タプルを出力する．

出力ユニットが結果タプルを受け付けたとき，それが出力インタフェースの通信容量を超えていれば，出力ユニットは照合ユニットを一時停止させる信号を真にセットする．そして結果タプルの出力が終了すると，これを偽にセットする．

照合ユニットの一時停止により，結果タプルの生成が停止される．これにより結果タプルは全て出力される．ただし，この方式はアドミッション制御により入力タプルを欠落させることになる．

6. 評　　価

本章では提案手法及びアーキテクチャの評価を行う．

6.1 準　　備

6.1.1 評価条件

ここでは評価条件としてキューの実現方法，タプルサイズ，メモリ幅，ROWS の設定について述べる．

提案アーキテクチャでは，キューは FPGA 上の BRAM を用いて実現した．この理由を下記に述べる．FPGA 上のレジスタを使用してこの記憶領域を構成すると，サイクルレベルで高速にデータアクセス可能だが，使用可能な量の制限が強い．したがって，キューの実現には大容量の記憶領域を確保できる FPGA 内部のメモリリソースである BRAM か分散メモリを利用する方式が考えられる．ここで，分散メモリの構成には，回路構成用リソースが使われる一方，BRAM の構成には BRAM 専用のプリミティブとして用意されたリソースが使用される．したがって，BRAM は分散メモリよりも FPGA のリソースを効率良く使用できる．したがって我々はキューを BRAM で構成した．FPGA 上の BRAM は 1 タプル/サイクルで書込みと読込みを行うことができる．

FPGA に搭載されているリソース量は FPGA によって異なるため，用いた FPGA に搭載されているリソースのみで実現できる ROWS の範囲で評価を行う．タプルは 100 bit までのものを扱うとし，メモリの帯域幅は 100 bit とする．ROWS の値は固定であり，二つのキューの ROWS の値は同じとする．

6.1.2 実験環境

Xilinx 社の FPGA である XC6VLX240T-1 を対象に，提案アーキテクチャの構成回路を合成する．表 1 に，XC6VLX240T の主な仕様を示す．開発ツールとして Xilinx ISE 12.1 Logic Edidion を用い，XST コンパイラで合成し，配置配線を行う．

(注4)：結合選択率が 100%の場合を指す．

6.1.3 評価指標

本論文では，ウィンドウ結合の入力レートと出力レートを評価指標とする．提案方式の性能評価に実トラヒックを用いることは，その非一定性から不適当である．それゆえ我々は仮想情報源を用いて生成される人工トラヒックにより性能評価を行う．入力ストリーム情報源として，FPGA 上にストリームデータを発生させる回路を実装する．発生するストリームタプルのキーを設定することで，結合選択率を制御する．

6.2 実験結果

本節では，第一に XC6VLX240T に搭載されているリソースのみを用いて実現できるキューの ROWS についての評価を行う．第二に提案するアーキテクチャを今回用いる FPGA に実装したときの動作周波数の評価を行う．第三に，提案アーキテクチャのウィンドウ結合が，入力タプルを取りこぼしなく全て処理できる最大の入力レートについて評価する．第四に，この入力レートのデータストリームを処理するために必要な出力の通信容量を明らかにするために，出力レートについて評価する．第五に，この通信容量を実現できなかった場合における，入力タプルの処理率の減少について評価する．

6.2.1 使用リソース量

表 2 に，合成結果から得られた使用リソース量を示す．これらの値は上記のストリームデータを発生させる回路と出力タプルを格納する記憶領域のリソース量は含まない．表から，ROWS を 2^{17} 以上とすると，FPGA の BRAM が不足し，XC6VLX240T には実装できないことが分かった．

表 1 XC6VLX240T
Table 1 XC6VLX240T.

項目	個数
Slice Registers	301,440
Slice LUTs	150,720
Slice	37,680
BRAM (32 KB)	416
DSP48	768

表 2 ウィンドウ結合のリソース使用量
Table 2 Resource usage of window join operation.

ROWS [個]	Slice [個]	BRAM [個]
2^{10}	43	12
2^{12}	49	24
2^{14}	410	92
2^{16}	793	356
2^{17}	—	—

6.2.2 動作周波数

提案するアーキテクチャのウィンドウ結合回路を FPGA に実装した結果，205.17 MHz の動作周波数で動作することが分かった．Streams on Wires [8] では，実験室レベルで生成可能な IP パケット発生率は 100 MHz 程度であり，その周波数が FPGA 上のストリーム処理の現実的な最高性能だとしている．彼らの主張に従えば，我々のウィンドウ結合回路の実装は現実的な最高性能を達成しているといえる．

入力タプルを 100% 取りこぼしなく処理できる最大の入力レートを考える．入力ストリームのタプル到達間隔よりも，1 タプルの処理時間が長い場合，入力ストリームのタプルの取りこぼしが発生する．本構成のウィンドウ結合では，一つの入力タプルの処理にかかる時間は $ROWS+2$ サイクルであるから，入力レートの理論値 ($Exact$) は次式で表される．ただし ω は FPGA の動作周波数である．

$$Exact(input) = \omega/(ROWS+2) \quad [\text{Tuple/s}] \tag{1}$$

ここで，$\omega = 100\,\text{MHz}$，$\text{ROWS} = 2^{16}$ を式 (1) に代入すると，

$$100\text{M}/(2^{16}+2) \simeq 100/2^6 \cdot 10^3 > 10^3 \tag{2}$$

であるから，100%取り零しなく処理可能なタプルの到着率は 1K Tuple/秒より大きい．すなわち，提案アーキテクチャは 1 ms 周期で到着する入力タプルであれば，取り零しなく処理できる．

6.2.3 入力レート

図 9 に上記の理論値と実験値を示す．この図からは

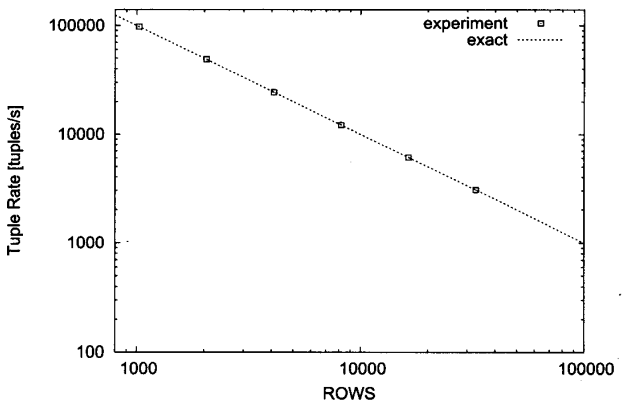


図 9 入力レート
Fig. 9 Tuple rate.

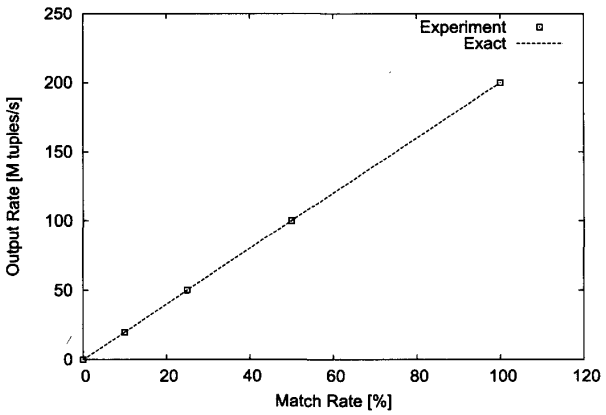


図 10 出力レート
Fig. 10 Output rate.

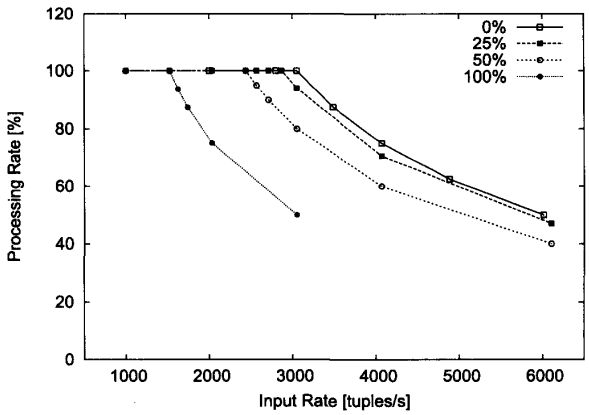


図 11 入力タプルの処理率
Fig. 11 Operation rate of input tuples.

実験値 (Experiment) と理論値 (Exact) が一致することが観察される．これより，我々が FPGA に実現したウィンドウ結合の実装は設計モデルを正確に反映していることが示唆される．

6.2.4 出力レート

提案アーキテクチャの出力レートに関する実験結果を図 10 に示す．出力レートの理論値は提案ウィンドウ結合のアーキテクチャから次のように求められる．ただし θ は結合選択率である．

$$Exact(output) = 2\theta \cdot \omega[\text{Tuple/s}] \qquad (3)$$

提案アーキテクチャは同時に二つの照合処理を行うため，1 サイクルで生成する結果タプルの数は結合選択率の 2 倍となる．図 10 より，出力レートについて，実験値と理論値が一致することが観察される．これより，我々が FPGA に実現したウィンドウ結合の実装は設計モデルを正確に反映していることが示唆される．

6.2.5 通信容量不足時の入力タプル処理率

式 (3) では，結果タプルの出力レートは ROWS に依存しないと考えられる．しかし，これは結果タプルを出力するのに十分な通信容量を有するインタフェースが存在することを仮定しているからである．

そこで，出力用の通信容量が不足した場合について考える．提案アーキテクチャでは，同時に発生する結果タプルの数は最大で 2 である．したがって FPGA から結果タプルを出力するインタフェースが 2 タプル/サイクルで動作可能であれば，滞りなく結果タプルを出力できる．

FPGA から結果タプルを出力するインタフェースが十分な通信容量がもたない場合，照合層を一時停止させることで，結果タプルの生成を制限する．これにより，生成した結果タプルを全て出力することが可能となる．しかし，照合層を一時停止させると，照合処理に遅延が生じる．

1 タプル/サイクルで出力可能なときの入力タプルの処理率を図 11 に示す．ただし ROWS は 2^{15} と設定した．図 11 から，結合選択率が高いほど入力タプルの処理率が減少することが観察される．ここで，ウィンドウ結合演算器のキューのサイズが N[tuple]，出力容量が C_{out}[tuple/cycle] の場合，t[cycle] 間隔で到達するタプル（到着率は $1/t$）を処理できる割合 $Rate_{op}$ と結合選択率 M の関係は次のように定式化できる．

まず，出力されるタプルの総数 $nMatchedTuples$ は $nMatchedTuples = N \cdot M$ で得られる．また，出力容量 C_{out}[tuple/cycle] のウィンドウ結合演算器が照合処理により得られる結果タプルをすべて出力するために必要なサイクル数 $nOutputCycles$[cycle] は，$nOutputCycles = nMatchedTuples/C_{out}$ になる．ただし，提案するアーキテクチャでは到着したタプルとキュー内の N 個のタプルを比較するためには $N + 2$[cycle] 必要である．したがって，タプルを出力するサイクルに比べ照合処理にかかるサイクル数が大きい ($nOutputCycles \leq N + 2$) 場合には，$nOutputCycles$ は $N + 2$ に抑えられる．これは，タプルを出力するサイクルが照合処理にかかるサイクル数を超えることはないことを意味する．

提案するアーキテクチャでは，入力タプルに対して，すべてのキュー内のタプルの照合処理が完了するまで次の入力タプルは受理されず破棄される．したがって，入力タプルが到着する間隔 t[cycle] が処理サイクルより短い場合 ($t < nOutputCycles$) には，入力タプルのうち処理可能なタプルの割合，すなわち処理率 $Rate_{op}$

は $Rate_{op} = t/nOutputCycles$ で得られる．ただし，t が処理サイクルより長い ($t \geq nOutputCycles$) 場合には，全てのタプルが処理可能で，また，到着するタプル以上のタプルが処理されることはあり得ないため，$Rate_{op} = 1$ と定義する．

以上より，キューのサイズが N[tuple]，出力容量が C_{out}[tuple/cycle] のウィンドウ結合演算器に対し，t[cycle] 間隔で到達するタプルの処理できる割合 $Rate_{op}$ は，結合選択率 M に対して次のように定式化される．

$$Rate_{op} = \begin{cases} t/nOutputCycles & t < nOuputCycles \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

ここで，$nOutputCycles$ は次のようになる．

$$nOutputCycles = \begin{cases} (N \cdot M)/C_{out} & (N \cdot M)/C_{out} \leq N + 2 \\ N + 2 & \text{otherwise} \end{cases} \tag{5}$$

これは，図 11 に示した実験結果と一致する．

結合選択率が 0%である場合には，結果タプルが生成されないから，遅延が生じない．このとき，減少率は ROWS の値によらず一定の結果が得られた．照合処理に遅延が生じると，入力タプルの取りこぼしが発生する．つまり，入力タプルを取りこぼしなく処理することができなくなる．したがって，全ての入力タプルの処理が可能なタプルレートが減少する．図 11 から，処理可能な最大のタプルレートの値は最大で 50%減少することが分かった．

また，出力されたタプルを確認した結果，結果タプルについては漏れなく出力がされていることを確認した．

6.3 提案アーキテクチャの性能の考察

提案アーキテクチャで実現した ROWS と処理速度が，実際の DoS 検知に十分であるかどうか考察する．

まず，提案アーキテクチャの ROWS に関して考察する．文献 [14] には，DoS 攻撃を検知するためには 1 秒間当りのパケット数が用いられていること，またそのパケット数が 2 万〜3 万 5 千パケット，あるいは 5 万パケットと記されている．これより，実際の DoS 検知には最低限 5 万タプルが必要だとみなせる．したがって，実際の DoS 検知に最低限必要な ROWS は 5 万であるといえる．表に示したとおり，提案アーキテクチャの実装が許す最大 ROWS は $2^{16} = 65536$ であり，実際の DoS 検知に最低限必要な ROWS を上回る．ただし，市場で入手可能なより大きな BRAM をもつ FPGA を用いることで，最大 ROWS を更に大きくすることが可能である．

次に，提案アーキテクチャの処理速度について考察する．DoS 攻撃を検知するには 1 秒間当りのパケット数が用いられる [14]．提案アーキテクチャは 6.2.2 節に示したように 1 ms 周期で発生するストリームデータを取りこぼしなく処理できる．この処理時間は上述の検知時間である 1 秒間よりも短い．したがって，提案アーキテクチャの処理速度は DoS 検知に十分だと考えられる．

7. むすび

本論文では，ストリームデータに対するウィンドウ結合を FPGA 上で効率的に実現する手法と，その手法を実現するアーキテクチャを提案した．提案手法は照合処理の並列化と，並列化照合処理のパイプライン化であった．提案アーキテクチャはそれらの処理を実現するために，3 層のアーキテクチャにより構成された．性能向上に特に関連したのは照合層であった．実験の結果，提案アーキテクチャは，2^{16} 個のタプルに関するウィンドウ結合を実現し，1 ms 周期で発生するストリームデータを処理できることが示された．高性能化に起因して，出力結果タプルが欠落する問題が生じたが，照合層に配したアドミッション制御機構によりその問題を防げることを示した．

今後の課題としては，複数ストリームに対するウィンドウ結合への本研究の発展が挙げられる．

謝辞 本研究の一部は科研費（#22700090），新世代ネットワーク技術戦略の実現に向けた萌芽的研究，を受けたものである．ここに記して謝意を表す．

文　献

[1] K. Inoue, D. Akashi, M. Koibuchi, H. Kawashima, and H. Nishi, "Semantic router using data stream to enrich services," Proc. CFI, pp.20–23, 2008.

[2] H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. Zdonik, "Retrospective on aurora," The VLDB Journal, vol.13, pp.370–383, Dec. 2004.

http://dx.doi.org/10.1007/s00778-004-0133-5

[3] A. Arasu, S. Babu, and J. Widom, "The cql continuous query language: semantic foundations and query execution," The VLDB Journal, vol.15, pp.121–142, June 2006. http://dx.doi.org/10.1007/s00778-004-0147-z

[4] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, "Gigascope: a stream database for network applications," Proc. 2003 ACM SIGMOD International Conference on Management of Data international conference on Management of data, pp.647–651, 2003.

[5] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, F. Reiss, and M.A. Shah, "Telegraphcq: continuous dataflow processing," Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp.668–668, SIGMOD '03, ACM, New York, NY, USA, 2003. http://doi.acm.org/10.1145/872757.872857

[6] B. Gedik, H. Andrade, K.-L. Wu, P.S. Yu, and M. Doo, "Spade: the system s declarative stream processing engine," Proc. SIGMOD, pp.1123–1134, 2008.

[7] Y. Ahmad, B. Berg, U. Cetintemel, M. Humphrey, J.-H. Hwang, A. Jhingran, A. Maskey, O. Papaemmanouil, A. Rasin, N. Tatbul, W. Xing, Y. Xing, and S. Zdonik, "Distributed operation in the borealis stream processing engine," Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp.882–884, SIGMOD '05, ACM, New York, NY, USA, 2005. http://doi.acm.org/10.1145/1066157.1066274

[8] R. Mueller, J. Teubner, and G. Alonso, "Streams on wires: a query compiler for fpgas," Proc. VLDB Endow., vol.2, no.1, pp.229–240, 2009.

[9] R. Mueller, J. Teubner, and G. Alonso, "Data processing on fpgas," Proc. VLDB Endow., vol.2, pp.910–921, Aug. 2009.

[10] L. Woods, J. Teubner, and G. Alonso, "Complex event detection at wire speed with fpgas," PVLDB, vol.3, no.1, pp.660–669, 2010.

[11] R. Mueller, J. Teubner, and G. Alonso, "Glacier: a query-to-hardware compiler," SIGMOD '10: Proc. 2010 international conference on Management of data, pp.1159–1162, ACM, New York, NY, USA, 2010.

[12] M.A. Hammad, W.G. Aref, and A.K. Elmagarmid, "Stream window join: tracking moving objects in sensor-network databases," Proc. 15th International Conference on Scientific and Statistical Database Management, pp.75–84, SSDBM '03, IEEE Computer Society, Washington, DC, USA, 2003. http://dx.doi.org/10.1109/SSDM.2003.1214967

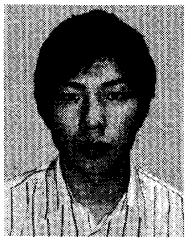
[13] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," Proc. 29th international conference on Very large data bases - Volume 29, pp.309–320, VLDB '2003, VLDB Endowment, 2003. http://portal.acm.org/citation.cfm?id=1315451.1315479

[14] Microsoft Technet: DDoS の実体と対策, "http://www.microsoft.com/japan/technet/security/secnews/columns/column060426.mspx".

(平成 23 年 1 月 31 日受付, 5 月 25 日再受付)

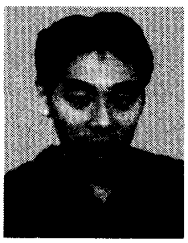
三好 健文 (正員)

2003 東工大・工・電気電子卒, 2005 同大大学院電子機能システム専攻修士課程了. 2007 同大学院物理情報システム専攻博士課程了. 博士 (工学). 同年東京大学情報理工学系研究科特任助教. 東京工業大学情報理工学研究科産学官連携研究員を経て, 2009 より電気通信大学大学院情報システム学研究科助教. コンパイラ, HW/SW 協調設計に関する研究に従事. IEEE, ACM, 情報処理学会各会員.



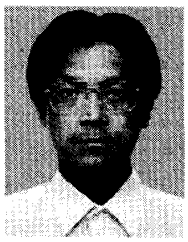
寺田 祐太

2011 電通大・電気通信卒. 同年より (株) アバールデータへ入社. 組込みシステム開発に従事. 情報処理学会会員.



川島 英之

1999 慶大・理工・電気卒. 2005 同大大学院理工学研究科開放環境科学専攻後期博士課程了. 同年, 慶應義塾大学理工学部助手. 2007, 筑波大学大学院システム情報工学研究科講師, 並びに計算科学研究センター講師. 博士 (工学). センタデータ管理に関する研究に従事. 日本データベース学会, 情報処理学会, ACM, IEEE 各会員.



吉永 努 (正員)

1986 宇都宮大・工・情報工学卒. 1988 同大大学院工学研究科修士課程了. 同年より宇都宮大学工学部助手. 1997 から翌年にかけて電子技術総合研究所・客員研究員. 2000 より電気通信大学大学院情報システム学研究科助教授. 現在, 同教授. 博士 (工学). 並列分散処理, 計算機アーキテクチャなどに興味をもつ. 情報処理学会, IEEE 各会員.