平成２８年度修士論文

マルチノードFPGAによる ストリームデータ分散結合処理

大学院情報システム学研究科
情報ネットワークシステム学専攻

学 籍 番 号 : 1552013
氏 名: 多田 昂介
主任指導教員: 吉永 努 教授
指 導 教 員 : 森田 啓義 教授
指 導 教 員 : 策力木格 准教授
提出年月日 : 平成29年1月26日

(表紙裏)

目 次

図 目 次

表 目 次

第1章　序論

多様なWebサービスの普及とセンサ技術の発達にともない，データセンタに収集されるデータの速度と量は増大を続けている．金融情報処理 [1] やネットワークトラフィックの監視 [2] のようなデータ処理タスクは，途切れなく到着するデータに対して解析が行われる．これらのタスクには厳しい時間制約が課せられているため，ネットワークの速度向上が著しい現在では，従来型の“Store-and-Process”のデータ処理モデルにおいてネットワークからの入力がボトルネックとなったり，計算結果を得るまでのレイテンシが増大するなど，十分な性能が達成できているとは言えない．

Data Stream Management System (DSMS) [3] は，ストリームデータに対してSQLライクな言語で記述されたクエリ演算を実行する，高いリアルタイム性を持つ計算機構である．DSMSがストリームデータに対する演算機構であることから，データの入力元に配置したFPGAなどの計算素子の活用が提案され，その実装の性能評価がなされている [4, 5]．到着データからペイロードを取り出す通信プロトコル処理に加えて，演算自体をFPGAが担うことで，低遅延かつ高スループットな処理を実現しようという試みである．

著者らの研究グループでは，DSMSにおいて重要な処理のひとつであるSliding-window Join [6] を対象に，その並列アルゴリズムであるHandshake join [5] のFPGA実装に取り組んできた [7, 8, 9, 10]．Sliding-window Joinは，2つのストリームデータを対象に，一定のウインドウサイズごとに，条件を満たすタプルを取り出す結合処理である．Handshake Joinは，より小さなウィンドウサイズ単位で結合処理を行うJoinスレッドの並列処理により，スループットを維持したままウインドウサイズの拡張に対応するアルゴリズムである．Handshake JoinのFPGA実装においては，FPGA上にJoin Coreを複数配置するハードウェア構造を提案し，性能を評価した [8]．Handshake JoinのFPGA実装は，遅延，スループットの面で高い性能が得られる一方で，FPGA上に多数のJoin Coreが構成されることから，計算可能な最大ウインドウサイズはFPGAのロジック資源の制約を受ける．そのため，より大きなウインドウサイズでの結合演算を可能にするためには，複数のFPGAを活用する仕組みが必要である．

本研究報告では，複数のFPGAボードを活用してHandshake Joinを行うデータ通信を仕組みを提案し，性能評価した結果を報告する．具体的には，FPGAボード間ネットワークを介してJoin Core配列をマルチノードに展開することにより，スケーラブルに大きなウィンドウサイズの結合演算を実現する．最大16ノードでの処理性能の測定を行った結果，ソフトウェアによる計算と比較して，高速且つ良いスケーラビリティが得られることを確認した．

第2章　背景・関連研究

2.1　ストリームデータ処理

本研究で取り扱うストリームデータ処理を必要とするアプリケーションには，コンピュータによる高頻度取引を行う金融情報処理 [1] や，外部からの攻撃を検知するネットワークトラフィック監視 [2] などが挙げられる．これらのアプリケーションに共通する特徴として，途切れなく到着するストリームデータに対して計算を行い，高スループットな性能を要求される厳しい時間的制約を持つ．このようなリアルタイム性が重要となるストリームデータ処理を行うための仕組みとして，Data Stream Management System (DSMS) [3] が存在する．DSMS は図 2.1.1 のような構成で処理を行う．DSMS 内部にはストリームデータ処理エンジンが存在し，ストリームデータに対して行う処理内容を SQL ライクな言語で記述されたクエリを予め登録しておくことで演算を実行する．このように DSMS はストリームデータ処理をリアルタイムに計算を行う仕組みを持つが，ネットワークの速度向上が著しい現在では従来型のコンピュータのアーキテクチャでの処理では，ネットワークのインターフェイス等データの移動がボトルネックになることが多い．

このインターフェイスのボトルネックや処理性能向上の為，データの入力元に Field Programmable Gate Array (FPGA) などの計算素子を配置する試みが行われている [4, 5]．従来のコンピュータのアーキテクチャとは異なりネットワークインターフェースなどが直接接続された計算素子に通信プロコトル処理や演算自体を FPGA 上で実行することで，低遅延かつ高スループットな性能を実現している．

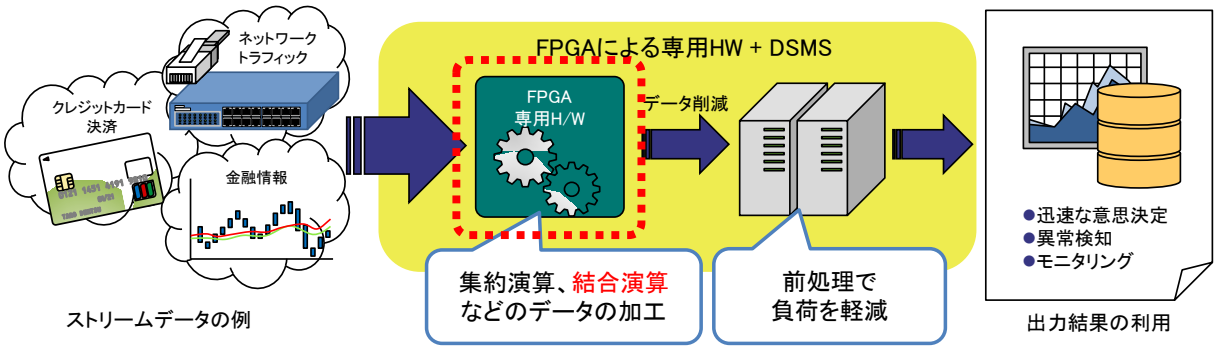

図 2.1.1: DSMS によるストリームデータ処理

2.2 ウィンドウ結合処理

DSMS が取り扱う代表的な処理の一つにウィンドウ結合処理がある．ストリームデータを取り扱う処理はストレージに格納されたデータベースに対する演算とは異なり，無限長の連続的なストリームデータを対象としている．また，これらのストリームデータを用いるアプリケーションは低レイテンシで高スループットなリアルタイム性を要求するため，結合演算を連続的に繰り返す必要がある．

データベースに格納された 2 つのテーブルを元に結合演算を行う際には，それぞれのテーブルに格納されたデータについて全ての組み合わせでクエリに基づいた比較を行う．このクエリによって定義されている条件が真である場合に，出力が生成される．このとき 2 つのテーブルのそれぞれの大きさを N，M とすると，$N \times M$ だけの計算回数と組み合わせを格納するメモリ空間を必要とする．

この $N \times M$ の組み合わせの結合演算のように，データベースに格納された有限のテーブルの結合演算とは異なり，無限長のストリームデータを扱う結合処理では，ストリームデータの 1 行のデータであるタプル同士の組み合わせから必要とされる計算回数とメモリ空間は無限となるため計算は不可能となる．

この問題に対応するため，ストリームデータ処理を扱うクエリのほとんどは，” ウィンドウ” と呼ばれる一定の大きさに分割して計算を行う仕組みを用いて行う [6]．ウィンドウ結合処理の原理を図 2.2.1 に示す．このウィンドウの大きさであるウィンドウサイズはクエリによって与えられる．

ウィンドウは 2 つのストリームデータ $StreamR$，$StreamS$ に対して $WindowR$，$WindowS$ として存在し，タプル $r \in R$，$s \in S$ が到着するとウィンドウサイズ以下のタプルがウィンドウ内に格納される．片方のタプルが到着するともう一方のウィンドウ内の全てのタプルと比較を行い，その結果を出力する．ウィンドウサイズを超えた数のタプルが到着すると，その時点でウィンドウ内の最古のタプルが押し出される形で破棄され，新たなタプルがウィンドウに挿入される．これを繰り返すことでストリームデータのウィンドウ結合処理を実現する．

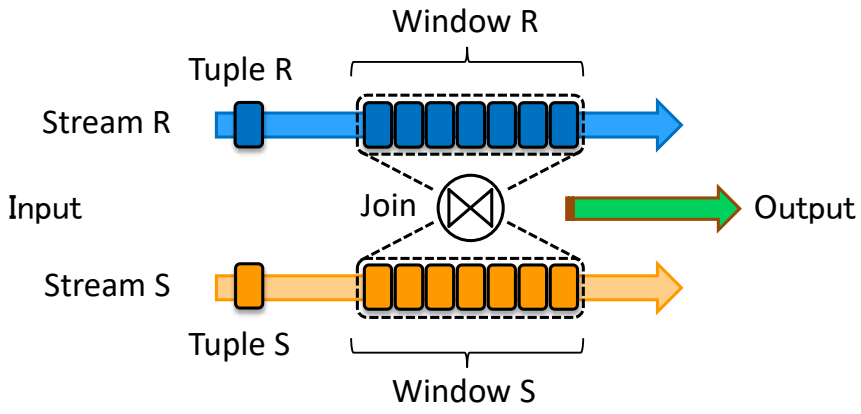

図 2.2.1: ストリーム結合処理

2.3 Handshake Join

ウィンドウ結合処理では，クエリによって示されるウィンドウサイズで結合演算を実行するが，その計算回数はウィンドウサイズの拡大によって増大する．2.2 節で示したウィンドウ結合処理では，拡大する計算量に対する対策が難しい．

ウィンドウ結合処理の高速化には，処理の並列化を進めることが有効である．Teubner らが提案した Handshake Join [5] は，FPGA やマルチコア CPU 等ハードウェアの並列性を有効活用するためのアルゴリズムである．

図 2.3.1 に Handshake Join の結合演算の様子を示す．Handshake Join は 2 つのストリームがお互いに逆方向に進んでいき，ウィンドウサイズですれ違う部分で計算を行う．入力されたストリームデータは，片方のタプルがウィンドウに格納される際に，もう一方のウィンドウに格納されたタプルと結合操作を行い，結果を出力する．ウィンドウ内で最古のタプルは押し出される形で破棄され，新たなタプルが挿入される．

Handshake Join は大きなウィンドウサイズに対して，ウィンドウを分割し”Join Core” として並列動作する特徴がある．この手法はウィンドウを複数のコアで分担し，各コアの結果を集計してウィンドウ結合の結果を得る．直列に接続したコアに対して互いに逆の方向からデータを入力することで結合演算の並列性を高め，高速化を実現した．

Handshake Join のアルゴリズムは FPGA のハードウェアの高い並列性を有効利用できることが，以前の研究 [8] によって示されている．この実装では，図 2.3.2 に示すようにシフトレジスタで実現されるウィンドウを Join Core として分割し，各 Join Core から出力される結合結果をマージ機構と呼ぶ集計機構に送ることで FPGA の並列性を活用したストリームデータ結合処理のハードウェア実行を実現した．

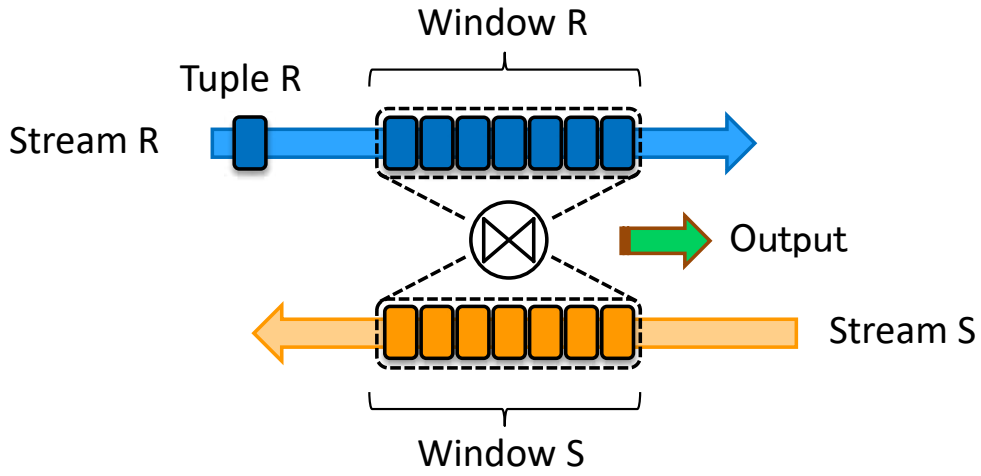

図 2.3.1: HanshakeJoin の原理

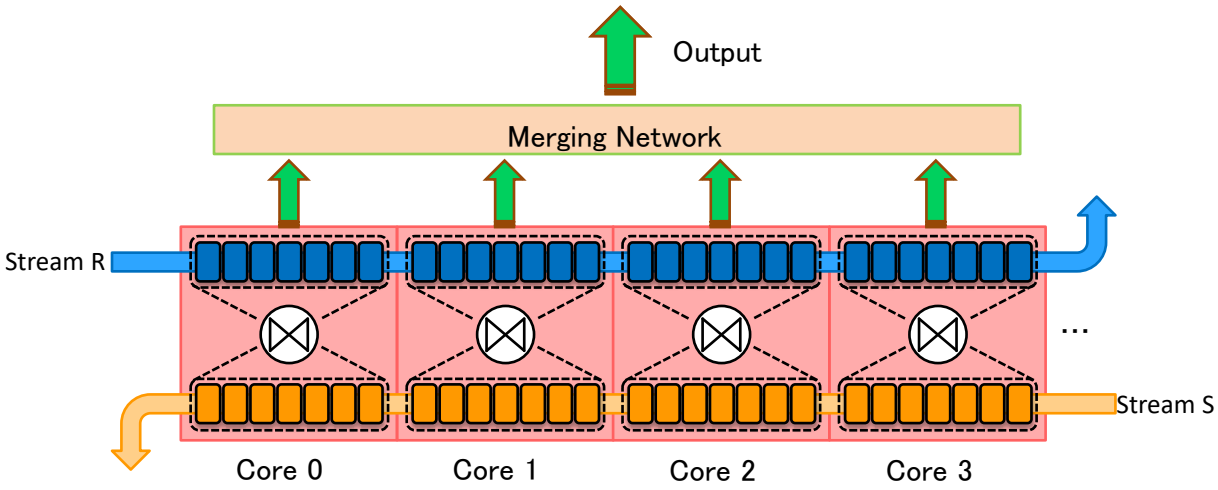

図 2.3.2: 4 コアの Join Core による HandshakeJoin

第3章　マルチノードFPGAによるHJの並列化

本研究では，ストリーム結合演算の処理の規模であるウインドウサイズ拡大を目的に，複数FPGAボードを用いた分散結合処理の仕組みを提案する．FPGAボード群が構築する高速な光ネットワークと全ボード共有するDRAM領域を活用し，ネットワーク越しに処理を分散して大きなウインドウサイズの結合処理を実現する．

3.1節にストリーム結合処理のマルチノード化の概要を示し、3.2節にFPGA間のストリームデータ共有と通信オーバーヘッドの隠蔽に用いた手法を示す。3.3節には複数台のFPGAを一括して制御するための手法について述べる。

3.1　結合処理のマルチノード拡張

提案するストリームデータ分散結合処理の概要を図3.1.1に示す．ストリーム結合処理をマルチノードへ展開するため，処理するウィンドウを複数のFPGAで分割する．各FPGAは分割されたウィンドウサイズ(Local Window Size)での結合処理を実行し，結果を集計することで大きなウィンドウサイズ(Total Window Size)での結合処理を実現する．

各FPGAは，ホストの計算機とは独立して光ネットワークのインターフェイスを持ち，リングネットワークを構築する．FPGA上のDRAMに書き込まれた内容は，自動的に書き込んだノードからリングネットワーク上に送出され，光I/Fを経由して全てのボードで常に同期され続ける．同期速度は最短約500nsと非常に短いため，本研究ではFPGA上のDRAMを全ノードで共有されたDRAM領域として扱う．

ストリームデータは，FPGAボード上のDRAMで構成する共有メモリを用いて全ノードに送付される．処理に利用するデータは予め共有メモリ内に存在しておく状態にすることによって，各ノードは処理するべきデータを共有メモリ内から読み出すことで処理を実行できる．

分散結合処理中は，全ノードが計算ノードとして処理を行うとともに，内1ノードはマスターノードとして計算ノードも兼ねる．マスターノードは計算ノードにデータを転送するとともに，計算結果を集計する役割を持つ．ノードはマスターノードの指示に従ってデータを受け取り，結合処理を実行し，その後マスターノードへと結果を転送する．各計算ノードが処理するデータをマスターノードが適切に指示することで，リングネットワーク中でHandshake Joinを実現する．

ストリームデータの転送と結合処理の制御は，光ネットワークで接続されたノード上のレジスタを包括して制御することができるFPGA向け分散処理ライブラリのTAMAMO[11]を利用しホストPCからレジスタの制御を行う．

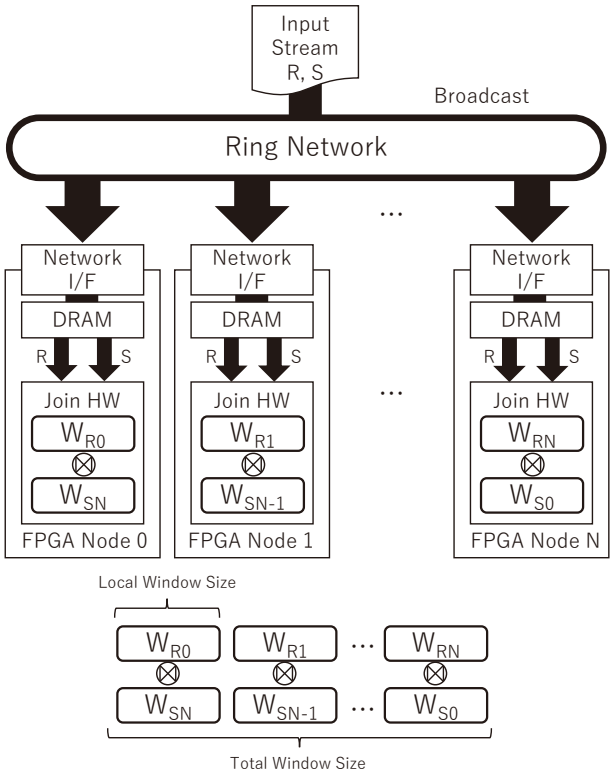

図 3.1.1: マルチノードストリーム結合処理

3.1.1 ハードウェア実装

提案する結合処理のマルチノード実装を行うハードウェアの構造図を図 3.1.2 に示す．本研究で FPGA ノードと呼ぶ 1 ノードのハードウェアは，PC と FPGA ボードの組で構成される．

FPGA ボードと PC は PCI Express バスで接続され，これを介してホスト PC の OS から FPGA ボードの DRAM 及びレジスタに対してソフトウェア制御を行うことができる．ホスト PC からはライブラリ内の関数を通じて FPGA 内の DRAM に任意のデータを読み書きでき，共有メモリを利用することができる．

FPGA 内部では，PCI Express，光ネットワークインターフェース，DRAM，結合演算モジュール（Join HW）と独自のバスを介して接続されている．これらのインターフェイスはバスに準拠したアクセスを行うことで読み書きを行う．

結合演算モジュール（Join HW）は複数の Join Core とマージ機構を使用した Oge らの実装 [8] を参考にしたモジュールである．Join HW のコア数やウィンドウサイズなどの設定は論理合成時に指定することで変更が可能である．

Join HW と DRAM の間のやり取りには，“Stream Data Controller” と呼ぶ DRAM に対するストリームデータの制御を行うモジュールである．内部には，ストリームデータを読み出し，結果を書き込む DRAM 制御モジュールが 2 器実装され，DRAM に対して 2 つのストリームを同時に読み出すことで入力ストリームを制御する．マージ機構からのストリームは一旦”Stream Data Controller“ 内部の FIFO によるバッファに溜め込まれ，一定の数の出力ごとに DRAM への書き込みを行う．

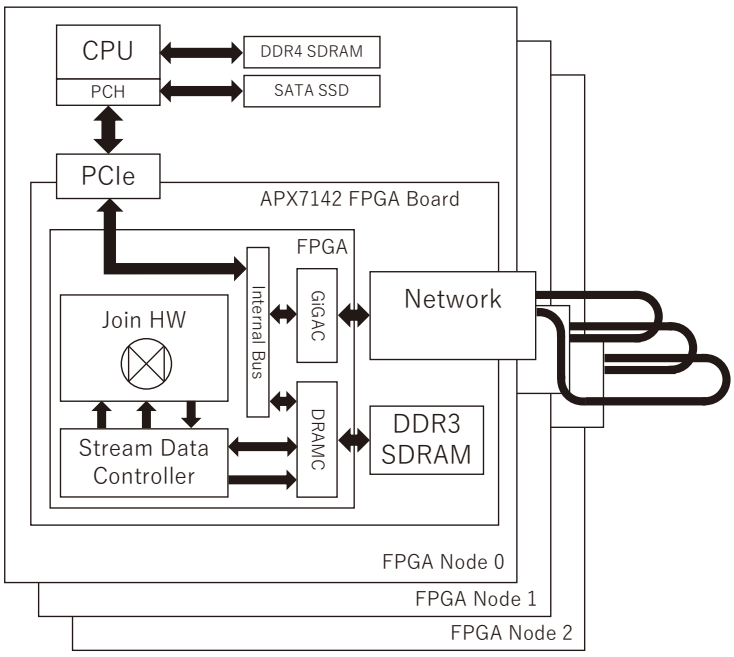

図 3.1.2: ハードウェア構造図

3.2 光ネットワークを用いたストリームデータ配布方法

3.2.1 リングネットワークによる共有メモリ

本研究で利用している FPGA ボードに実装されている光ネットワークインターフェースは，前節で述べたように接続されたボードでトークンリングネットワークを構成し通信することができる．

接続されたリングネットワーク内部では，フレームと呼ばれるトークンが周回しており，各ノードの DRAM 領域に書き込まれたデータや，共有レジスタ領域の内容はフレームに載せられる．このフレームがリングネットワーク内を一周することで内容が全ての FPGA 内部で同期され，それぞれ共有メモリと共有レジスタとして動作することができる．

これらの通信に関わるプロトコル処理などは，FPGA 上に実装されているネットワークインターフェースの IP によって自動的に制御されるため共有メモリや共有レジスタの同期と，ストリーム結合処理は独立して行うことができる．

3.2.2 DRAM 領域活用による通信オーバーヘッドの隠蔽

データ転送にかかるオーバーヘッドを削減するために，結合処理と転送処理のオーバーラップを行っている．データ共有領域を 3 つに分割してリングバッファを構築する．

リングネットワークで共有される共有 DRAM 領域は，データ共有用の領域と結果共有用の領域に分割される．前者はマスターノードのみが書き込み，各ノードが参照を行う．一方で，結果共有領域はノード台数で更に分割され，計算ノードは各々に割り当てられた領域に結果を書き込むことでマスターノードに計算結果を転送する．

図 3.2.1 に，共有メモリ領域内のバッファの様子を示す．情報源から新たに入力されたデータは，結合処理とは独立して空き領域に挿入され，全ノードの DRAM に格納される．一方で，保持する全てのデータへの処理が終わった領域は，空き領域と見なされ新たなデータの挿入を待つ．

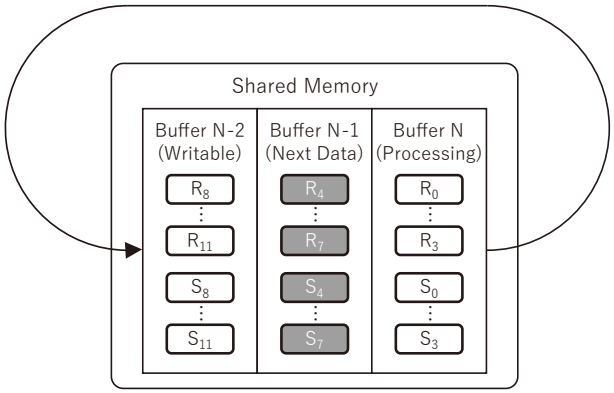

図 3.2.1: DRAM 領域を活用したバッファ

3.2.3 マルチノード実行時のデータの移動

図 3.2.2 に，4 ノード構成のマルチノード実行の例を示す．逆方向から入力される 2 つの入力ストリーム R と S に対して，各計算ノードが処理するデータを交互に進めることで，Handshake Join のマルチノード化を実現する．

(1). 計算ノードは，共有領域からそれぞれ逆方向から R と S を読み出す．しかし，この時点では．R と S 両方のデータを有するノードが存在しないため，結合処理は行われない．

(2). ストリーム R が 1 ノード分移動すると，$Node0$ から $Node2$ が順に R_2 から R_0 を読み出す．ここで初めて $Node2$ 上に R_0 と S_0 両方のデータが存在することになり，結合処理が行われる．

(3). 同様にストリーム S が 1 ノード分移動すると，$Node1$ から $Node3$ が順に S_0 から R_1 を持つ．この時点で $Node1$ 上で R_1 と S_0，$Node2$ 上で R_0 と S_1 の結合処理が行われる．

(4). ある程度処理が進むと，全ノードで結合処理が実行される．この時 R_0 は一番端のノード上にある．

(5). 次のストリーム R の移動で，R_0 は読み込まれず，代わって R_4 が読み込まれる．R_4 は，計算ノードによる処理実行中に，リングバッファにより共有されたデータである．

(6). 同様に，次のストリーム S の移動で，バッファS_0 は読み込まれなくなり，破棄されたことを示す．バッファS_4 を読みだす．S_4 も同様に，処理中に共有されていたデータである．

実際にはマッチするタプルの数により各ノードのバッファ処理時間にばらつきが生じる．そのため各ノードはマスターノードからストリームデータとともに TAMAMO によって与えられる処理対象バッファの組に従い DRAM 内にバッファが存在する限り連続して処理を実行する．

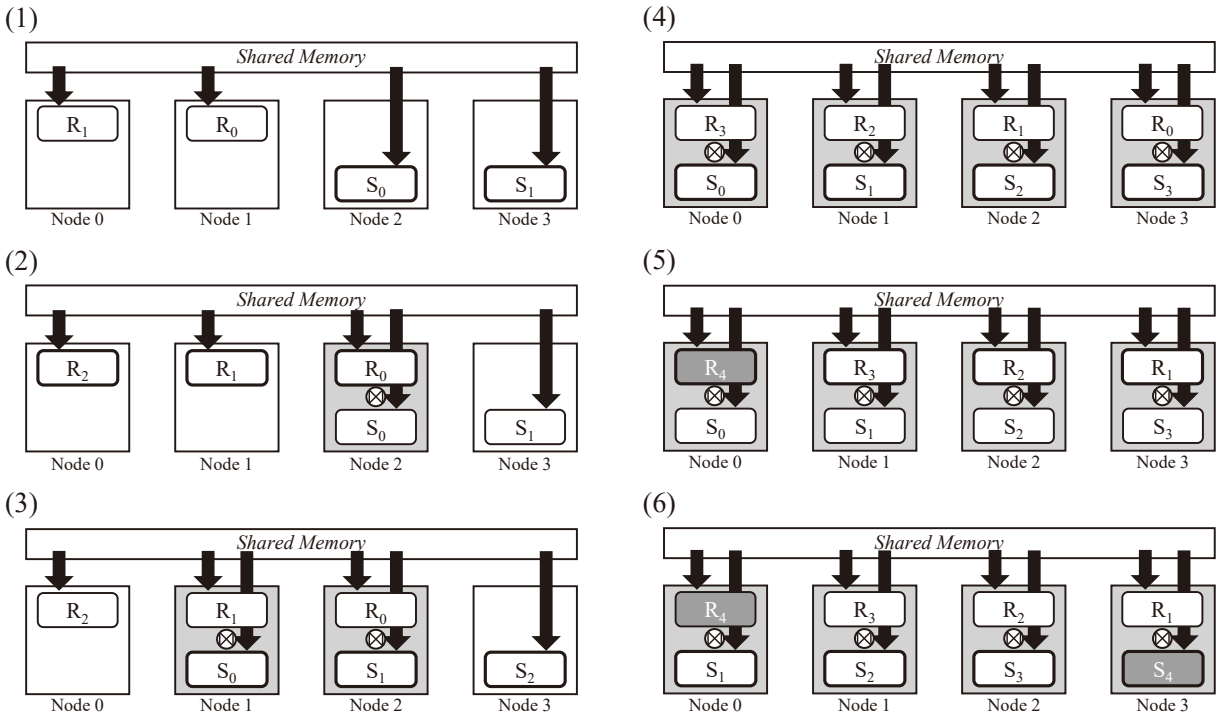

図 3.2.2: マルチノード FPGA 上でのストリーム結合処理

3.3 FPGA ノードの制御

ストリームデータの転送と結合処理の制御を行う TAMAMO は，リングネットワーク上で低遅延で共有される各ノードの共有レジスタに OpeItem 構造体を配置し制御を行う．OpeItem 構造体とは各ノード制御情報を格納，共有し，ソフトウェアで扱うことが可能な命令構造体である．

3.3.1 FPGA 向け分散処理ライブラリ”TAMAMO”

TAMAMO は，共有レジスタを介してネットワーク内の FPGA を制御する．共有レジスタ自体は PCI Express で接続されたホスト PC の OS からソフトウェアでアクセス可能であるが必要なレジスタ情報の共有には領域のアドレス管理や各 FPGA ごとの制御が煩雑になる．

そこで TAMAMO は，共有レジスタの領域内に OpeItem 構造体を展開することでアドレス管理などを一括で行い，制御を統括するマスター側のプロセスと FPGA の制御を実行するスレーブ側のプロセスに分けることで複数台の FPGA の制御を実現する．

3.3.2 マルチノード FPGA の制御

FPGA の制御の流れを図 3.3.1 に示す．

(1) マスター側のプロセスが，はじめに各ノードの処理対象バッファの組を命令として生成する．命令は OpeItem 構造体に書き込みリングネットワークを介して各ノードに通知を行う．全ノードで動作しているスレーブ側のプロセスは常に OpeItem 構造体の更新を確認し，通知されたノードは OpeItem 構造体から命令を読み込む．命令に従い DRAM からバッファを読み出しストリーム結合処理を実行する．

(2) 処理の実行時には処理対象のバッファが DRAM に格納されているかどうか，OpeItem 構造体を介した通知を確認しながら処理を進めていく．ストリーム結合処理を実行し，Join HW へストリームデータを入力している間は，常に Join HW の入出力 FIFO の空きを確認し状態によってバッファ制御を行う．

(3) 各 FPGA ノードは，割り当てられた処理対象バッファの組の処理が完了すると，スレーブ側のプロセスで OpeItem 構造体をポーリングし新たな命令が通知されるのを待つ．

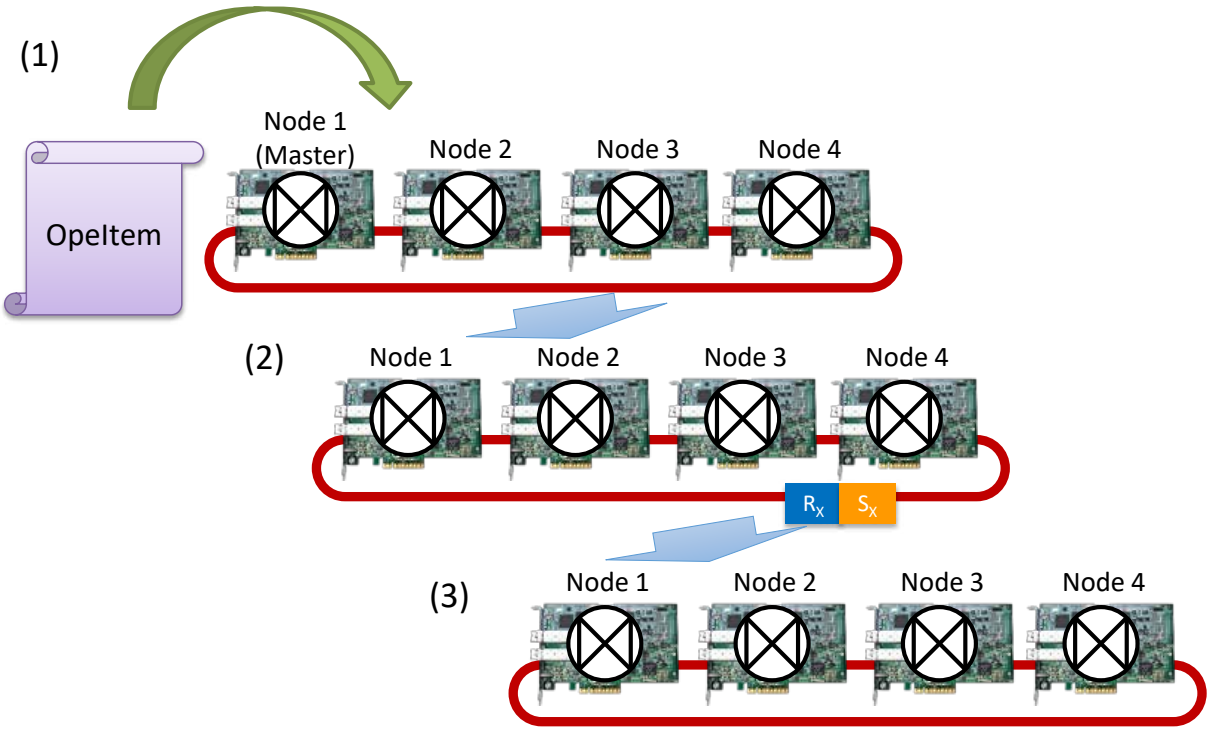

図 3.3.1: ストリーム結合処理実行中の FPGA 制御

第4章　評価

本章では，提案したマルチノードFPGAのストリームデータ結合処理の性能を，ストリームデータの入力スループットの評価を通して確認する．

本研究で実装したストリームデータ結合処理は，クエリで示された条件に一致したタプルのみが出力されるため，処理性能を示すスループットは単位時間あたりのタプルの出力数ではなく，タプルの入力数で計算される入力スループットを性能指標として取り上げる．

入力スループット評価では，2種類の方法で性能を評価する．(1)Join Coreが持つウィンドウサイズを固定し，コア数に比例しウィンドウサイズが拡大する条件での入力スループットを測定する．このパターンでは1コアが持つウィンドウサイズが一定のため，ウィンドウサイズ拡大した場合でも処理性能は一定となる．これをマルチノードへ展開した時のノード間の通信オーバーヘッドの影響を調査する．

(2) ウィンドウサイズをある固定のサイズに設定し，コア数の増加に対して1コアが持つウィンドウサイズが縮小していく条件のハードウェアで入力スループットを測定する．このパターンでは1コアあたりのウィンドウサイズが縮小していくため，コア数の増加，ノード数の増加に対して処理性能が向上していく．これをマルチノードへ展開した時の性能向上の様子を調査する．

4.1節に研究で用いたFPGAの仕様とノードを含めた計算機環境を示し，4.2節ではハードウェアに実装したクエリを紹介する．4.3節では各性能評価及び考察を述べる．

4.1　計算機環境

提案手法を実装した環境を表4.1と表4.2に示す．この計算機を1ノードとして最大16ノード用いた図4.1.1に示す環境で実験を行った．

表4.1: 計算機環境

No. of Node	1 to 16	
CPU	Intel Core i7-6700K	4.00GHz (4C8T)
Memory	DDR4 2133 MHz	32.0 GB
Network	Intel Ethernet Controller X540-AT2 on board Ethernet	10Gbps 1Gbps
SSD	Transcend TS64GSSD370S	
OS	CentOS 7.2 (Kernel 3.10.0)	
FPGA	AVALDATA APX7142 改	

表 4.2: FPGA ボードの仕様

Product	APX-7142 改
FPGA Device	Stratix V GX 5SGXMA3K1F40C2N runs at 125 MHz
DRAM (DDR3)	800 MHz, 2.0 GB
Network	Proprietary GiGA CHANNEL Optical token ring network 14 Gbps ×2ch
PCIe I/F	2.0 Gen2×8 Lane
Internal Bus	Proprietary AVAL-bus 256 bits-width

図 4.1.1: 実験に使用した計算機環境

4.2 ベンチマーククエリ

本研究で対象とするクエリを図 4.2.1 に示す．クエリは 2 つの入力ストリーム R, S を持ち，各ストリームからの入力タプル $r \in R$ と $s \in S$ は，4.2.2 に示すようなそれぞれ 4 つの 32bit データの組 $< key1, key2, value1, value2 >$ で構成される 128bit 幅のタプルを用いる．

図 4.2.1 のクエリは，入力タプル r と s の，それぞれ key1 及び key2 の 2 つの属性値を用い，WHERE 句に記述した条件を満たす結合演算を行うことを示している．この場合ではタプル r の key1，key2 がそれぞれタプル s の key1，key2 の ±10 以内に含まれている場合にタプルが出力される．

```
SELECT r.key1, r.key2, r.value1, r.value2,
       s.key1, s.key2, s.value1, s.value2
FROM   windowR AS r, windowS AS s
WHERE  r.key1 BETWEEN s.key1-10 AND s.key1+10
AND    r.key2 BETWEEN s.key2-10 AND s.key2+10
```

図 4.2.1: ストリームデータ結合処理クエリ

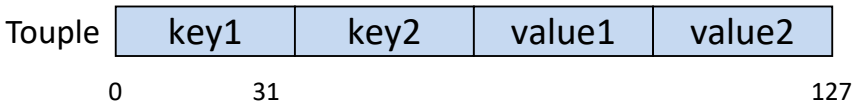

図 4.2.2: タプルの構成

4.2.1 入力データ構成

実験に用いた入力データの構成を次の図 4.2.3 に示す．

入力データはマスターノードの配布時に 1 ノードが処理するサイズに分割するとともに，動作に必要なデータを付与して配布を行う．このストリームデータの入力データは実験用に R, S の 2 種類生成して利用する．

入力データの構成は大きく分けて 3 つに分かれる．はじめに処理対象の有効なタプルを含むデータが入力されてくる．“Tuple” 部分にはタプルに含まれる 2 つのキー（key1, krey2) を乱数で生成したタプルを 1 ノードあたりのウィンドウサイズ（Local Window Size）の数だけ生成する．Join Core へ入力された際にはこの “Tuple” に含まれるデータが結合処理に利用される．

“Dummy” 部分のデータは実際には結合処理に利用されないダミーのデータとして利用される．ダミーのデータは結合処理に有効な “Tuple” データが入力し終わった後，Join Core のウィンドウ内に残っているタプルをすべて出力に押し出すために利用される．そのため，この “Dummy” 部分のデータは 1 ノードあたりのウィンドウサイズ（Local Window Size）の数だけ生成する．

“EOF” 部分は Join Core のロジック部分に対して入力データの終端を通知するための情報である．中身はすべて 1 で埋められたデータとなっており，Join Core はこの終端情報をロジック部分で検出することで処理完了のステートへ移行し，次の処理への準備のための一部回路のリセットが行われる．

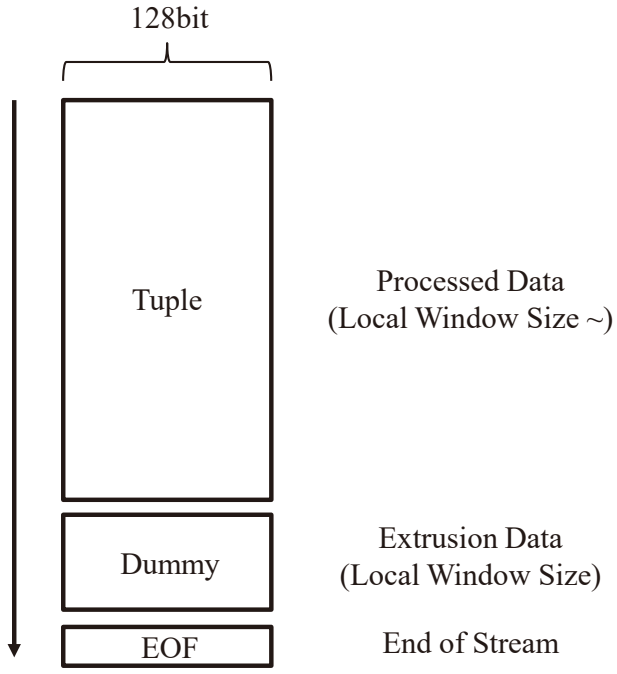

図 4.2.3: 入力データ構成

4.3 評価

4.3.1 予備評価: ハードウェアロジック使用量

予備評価として Join Core のコア数に対するリソース使用量の変化を調べた．図 4.3.1 に FPGA に実装したハードウェアのリソース使用量を示す．

”Logic utilization”は FPGA が持つ論理リソース Adaptive Logic Module(ALM) の使用量を示し，”Total block memory bits”は組み込みメモリである Block RAM の使用量を示している．また，Handshake Join 及び周辺ハードウェアは VHDL で実装されており，この値は Join Core 以外のバスなどの周辺ハードウェアを含んでいる．

1Join Core あたりのウィンドウサイズを 128 に固定した場合，論理リソースの使用量は Join Core 数に従って増大し，32 コア時に最大の 88.53%となった．一方で，組み込みメモリの利用量はウィンドウサイズに強く依存するが，Join Core 数の増大による影響は少ない．これは Stratix V で用意されている組み込みメモリが 20×1024bit のブロックサイズで与えられるためである．

本研究では，1FPGA ボードに実装できる最大の Join Core 数を 32 とする．

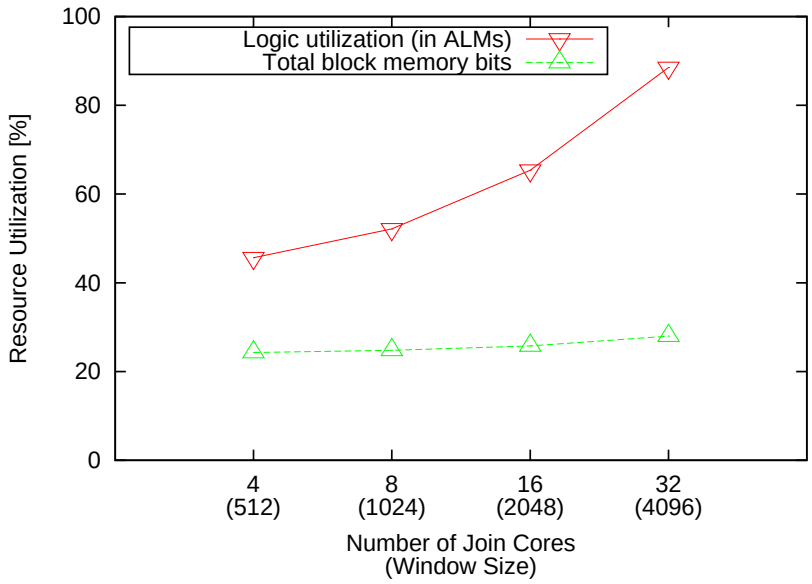

図 4.3.1: ハードウェア使用量

4.3.2 入力スループット評価

ストリームデータ分散結合処理の性能を示すために，(1)Join Core あたりのウインドウサイズを 128 に固定，および (2) 全体のウインドウサイズを 4096 に固定したそれぞれの条件下で，シングルノード及びマルチノードにおける入力スループットの関係を調査した．シングルノードでの結果から，コア数の増大が処理効率の維持または向上に貢献することを示し，マルチノードでも同様の傾向が維持されたことを示す．マルチノード評価時は，各 FPGA 上の実装コア数は 32 で固定する．

まず (1) コア毎のウィンドウサイズ固定の条件での性能を示す．シングルノードでの測定では，$4, 8, 16, 32$ コア実装したハードウェアを用いてウィンドウサイズ 512 – 4096 での入力スループット性能評価を行った．

マルチノードでは 32 コア実装したハードウェアを 1 ノードとしてノード数を $1, 2, 4, 8, 16$ ノードの構成にし，ウィンドウサイズ 4096 – 65536 での測定を行った．この実験ではコア数に比例してウィンドウサイズが拡大するが，1 コアあたりのウィンドウサイズは一定のため，入力スループットは一定になる．マルチノードへ展開した際に，FPGA をまたいだ通信のオーバーヘッドが性能低下に影響しないか確認を行った．

シングルノードの結果を図 4.3.2 に示す．ウインドウサイズはコア数に従って最大で 4096 まで拡大するが，スループットはコア数によらず一定であり，コア数増大による性能低下は見られなかった．

マルチノードの結果を図 4.3.3 に示す．図 4.3.2 で提示したシングルノード評価と比較して，最大で 16 倍のウインドウサイズに対してシングルノード時と同等の入力スループットが維持されている．ノードあたりのスループットの減少率は 0.004[Mt/s] であり，マスターノードによる包括的な制御を行った今回の実装のソフトウェアによるオーバーヘッドは極めて少ない．更にノードが増えた場合のスループットの減少を抑えるためには，各ノードの制御に複数台のマスターノードを配置する分散制御の仕組みを取り入れることが考えられる．

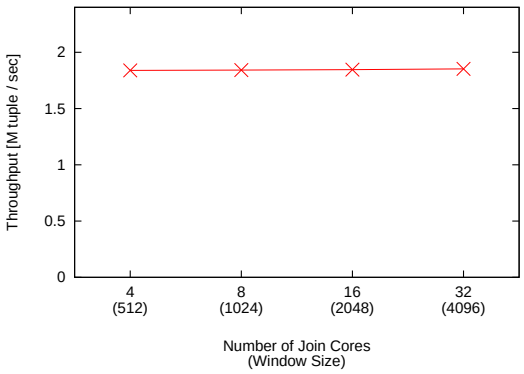

図 4.3.2: 入力スループット測定 (wsize=128/core)

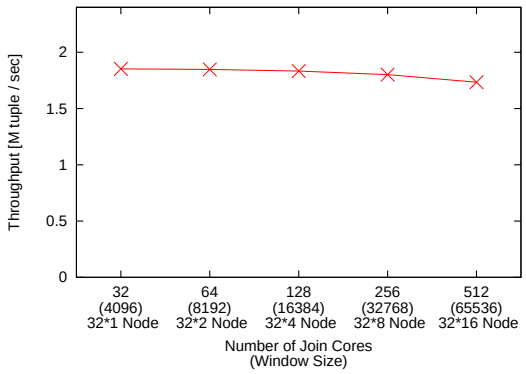

図 4.3.3: 入力スループット測定 (wsize=128/core), マルチノード

(2) 全体のウィンドウサイズ固定の条件での性能を示す．シングルノードでの測定では，4, 8, 16, 32 コアを実装し，結合処理全体のウィンドウサイズが 4096 一定となるようそれぞれのコアあたりウィンドウサイズを 1024, 512, 256, 128 設定したハードウェアを用いて入力スループット性能評価を行った．

マルチノードでは，(1) での実験と同様に 32 コア実装したハードウェアを 1 ノードとして，1, 2, 4, 8, 16 ノードの構成の入力スループット性能測定を行った．このときそれぞれのノード構成でのコアあたりウィンドウサイズは 128, 64, 32, 16 となる．この実験では同一のウィンドウサイズの処理に対して，コア数が増大していくと 1 コアあたりが処理するコアあたりウィンドウサイズが小さくなる．そのため，そのためコア数に対して入力スループットは向上していく．

マルチノードへ展開した際に入力スループット性能の向上の傾向変化の確認を行った．

シングルノードの結果を図 4.3.4 に示す．コア数に従って 1 コアあたりのウインドウサイズが小さくなるため，並列性が高まりスループットの向上が確認できる．

マルチノードの結果を図 4.3.5 に示す，全体のウィンドウサイズを固定した条件においても，図 4.3.4 で示したシングルノード評価と同様に台数に応じた性能向上が確認される．マルチノード環境において，4 ノード以上での性能向上の飽和が見られた．これは 1 コアが処理する規模が小さく，処理の負荷としては小さいため処理時間よりストリームデータの転送時間が上回っているためである．4 ノード以降での並列処理であればウィンドウサイズを 4096 より大きいサイズに設定することで効率的な処理が実現できる．

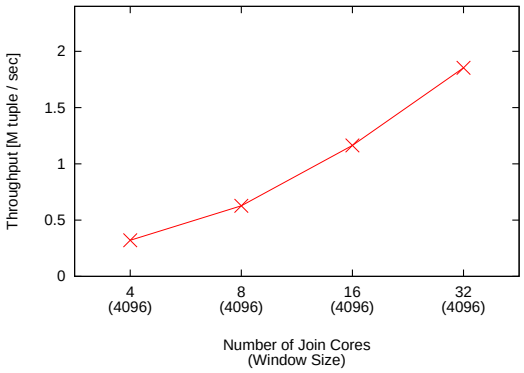

図 4.3.4: 入力スループット測定 (wsize=4096)

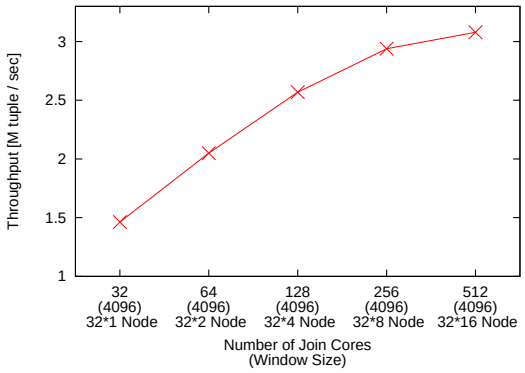

図 4.3.5: 入力スループット測定 (wsize=4096), マルチノード

4.3.3 ソフトウェア実行との比較

ソフトウェアでの分散結合処理基盤に対する優位性を示す為に，比較実験を行った．

比較に使用するソフトウェアは，Handshake Join 専用のソフトウェアを C++で実装したものを用いた．シングルノード測定の際には，タプルのマッチ率との関係を調べた．マッチ率は，入力データとして生成したタプルのうち，結合演算が行われるタプルの数の割合で示される．ソフトウェアとの評価の際には 10 – 100%の間でマッチ率を変化させた入力データを用意し，入力スループット測定を行った．

図 4.3.6 に，シングルノードかつ全体ウィンドウサイズを 4096 に設定した条件での，入力マッチ率とスループットの関係を示す．マッチ率の向上に従いマージ機構に送られる出力タプルの頻度が増大するため，ソフトウェア，FPGA ともにマッチ率の向上に従いスループットが低下する．しかし，全てのマッチ率の条件下における FPGA 実行はソフトウェアと比較して高速であり，最大 6 倍高速であることが確認された．

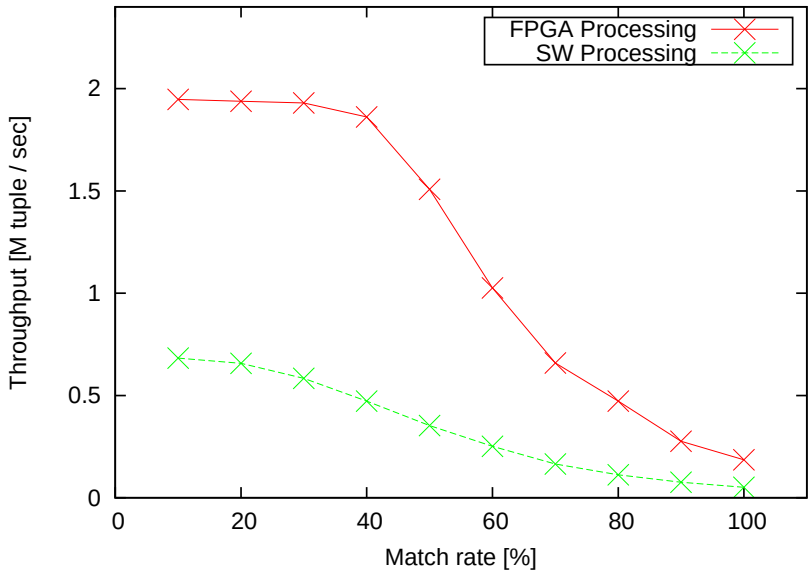

図 4.3.6: マッチ率変化による入力スループット比較

図 4.3.7 にマルチノード環境における，ノード数とスループットの関係を示す．比較の公平を期するために，ソフトウェア実行時のデータ転送には表 4.2 に示した FPGA ボードの GiGA CHANNEL を使用し，実行の制御に TAMAMO を用いた．DDT(Direct Data Transmission) と TAMAMO によるデータ転送と結合処理のオーバーラップによりシングルノード以上の性能差が開くことはなかったが，全ての台数設定下で，シングルノード時と同様に約 6 倍の高速化を維持した．

また，FPGA ボードを使用しない通常の PC クラスタとの性能比較も行った．PC クラスタは 10Gbps の Ethernet で相互接続し，ノード間のデータ転送には Ethernet を使用した TCP 通信で Point-to-Point での通信を行う．結合処理は PC 上の C++プログラムで実行する．Ethernet を用いたデータ配布は，送信側ノードから受信側ノードに対して 1 対 1 での TCP 通信を行うため，ノード数が増えると通信のオーバーヘッドが拡大していき，16 ノードの実験では図 4.3.7 に示すように FPGA でのハードウェア実行は 10G Ethernet とソフトウェアでの実行と比較して約 375 倍高速であることが分かった．

ハードウェア実行では通信オーバーヘッドの隠蔽の他，ネットワークインターフェースと FPGA が直接接続しているため，入力スループット性能の差が開いたと考えられる．これらの結果から，全てのマッチ率で，シングルノードと同等の高速化効率を維持しながらウインドウサイズ拡大を達成できると考えられる．

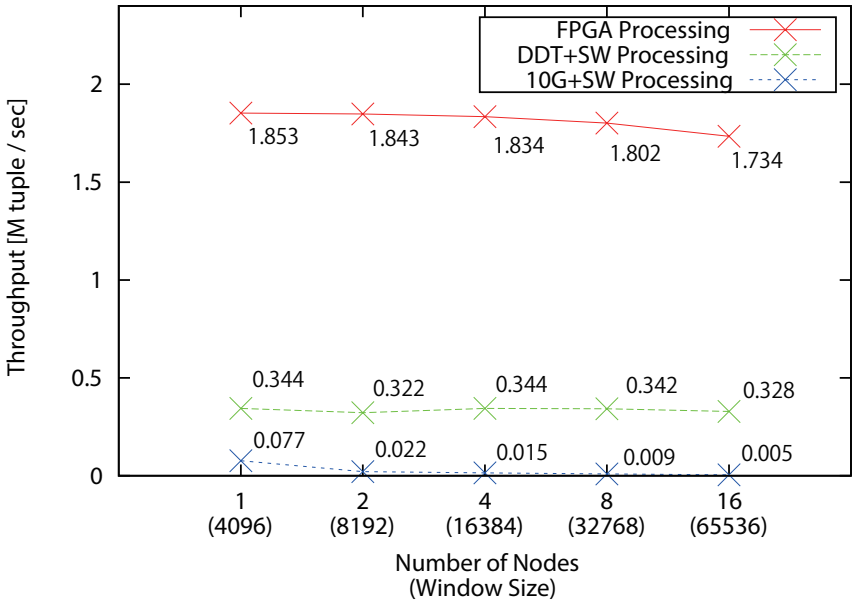

図 4.3.7: マルチノード環境における性比能較

第5章　結論

本研究では，ストリームデータ結合処理のウィンドウサイズ拡大を実現するため，複数台のFPGAを用いたマルチノードFPGA環境におけるストリームデータ分散結合処理の仕組みを提案した．FPGA同士をリングネットワークで接続し，共有メモリと共有レジスタを活用した通信オーバーヘッドの隠蔽するデータの配布と効率的なマルチノードFPGAの制御方法を実装し，従来1つのFPGAでは実現できなかった規模の問題に対するストリームデータ結合処理のハードウェア実行を実現した．

2つのパターンに基づいた入力スループット性能の評価を通じて，FPGA間の通信のオーバーヘッドの影響は極めて小さく，FPGAノードを増設することによって処理の規模，性能を向上が可能であることを示した．また，特別にチューニングされたソフトウェア実装との性能比較を通して，ハードウェア実行がマルチノードFPGAでのストリームデータ分散結合処理においても有利であることを示した．

今後の展開として，共有メモリ空間のより効率的な利用が考えられる．全体で処理を行うウィンドウサイズだけの領域が共有メモリ内に確保されているため，あるノードで処理が完了した不要なバッファ領域のデータが，全てのノードでの処理を終えるまで残留している．効率的な共有メモリ空間の利用を実現するため，ノードごとにバッファ領域の管理を行い効率的な共有メモリ空間の利用を実現する．

謝辞

本研究に際して，熱心なご指導を頂きました吉永努教授に感謝の意を表します．また，研究面・技術面ともに多大なる知識や示唆を頂いた吉見真聡助教に，感謝致します．最後に，研究生活を通じて様々な指摘，協力を下さいましたネットワークコンピューティング学講座の皆様に，厚く御礼申し上げます．

参考文献

[1] John W. Lockwood, Adwait Gupte, Nishit Mehta, Michaela Blott, Tom English, and Kees A. Vissers. A low-latency library in FPGA hardware for high-frequency trading (HFT). In *Proceedings of Hot Interconnects*, pp. 9–16, 2012.

[2] Pranav Vaidya, Jaehwan John Lee, Francis Bowen, Yingzi Du, Chandima H. Nadungodage, and Yuni Xia. Symbiote: a reconfigurable logic assisted data stream management system (RLADSMS). In *Proceedings of SIGMOD Conference*, pp. 1147–1150, 2010.

[3] Yanif Ahmad and Ugur Çetintemel. Data stream management architectures and prototypes. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pp. 639–643. Springer US, 2009.

[4] Rene Mueller, Jens Teubner, and Gustavo Alonso. Data processing on FPGAs. *The Proceedings of the VLDB Endowment*, Vol. 2, No. 1, pp. 910–921, 2009.

[5] Jens Teubner and Rene Mueller. How soccer players would do stream joins. In *Proceedings of SIGMOD Conference*, pp. 625–636, 2011.

[6] Jaewoo Kang, Jeffrey F. Naughton, and Stratis Viglas. Evaluating window joins over unbounded streams. In *Proceedings of ICDE*, pp. 341–352, 2003.

[7] Yasin Oge, Takefumi Miyoshi, Hideyuki Kawashima, and Tsutomu Yoshinaga. An implementation of handshake join on FPGA. In *Proceedings of ICNC*, pp. 95–104, 2011.

[8] Yasin Oge, Takefumi Miyoshi, Hideyuki Kawashima, and Tsutomu Yoshinaga. Design and implementation of a handshake join architecture on FPGA. *IEICE Trans. Info. & Syst.*, Vol. 95-D, No. 12, pp. 2919–2927, 2012.

[9] Yasin Oge, Takefumi Miyoshi, Hideyuki Kawashima, and Tsutomu Yoshinaga. Design and implementation of a merging network architecture for handshake join operator on FPGA. In *Proceedings of MCSoC*, pp. 84–91, 2012.

[10] Yasin Oge, Takefumi Miyoshi, Hideyuki Kawashima, and Tsutomu Yoshinaga. A fast handshake join implementation on FPGA with adaptive merging network. In *Proceedings of SSDBM*, pp. 44:1–44:4, 2013.

[11] 川原尚人. FPGA 向け分散処理ソフトウェアライブラリ”TAMAMO”. Master's thesis, 電気通信大学大学院システム学研究科情報ネットワークシステム学専攻, 2017.

発表論文

[1] 多田昂介, 川原尚人, 吉見真聡, 策力木格, 吉永 努 “マルチノード FPGA によるストリームデータ分散結合処理” 信学技報, vol. 116, no. 416, pp. 37-42, Jan. 2017.