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

信学技報
TECHNICAL REPORT OF IEICE.

マルチノードFPGAによるストリームデータ分散結合処理

多田　昂介† 　川原　尚人† 　吉見　真聡†† 　策力　木格†† 　吉永　努††

† 電気通信大学大学院情報システム学研究科
†† 電気通信大学大学院情報理工学研究科
〒 182–8585 東京都調布市調布ケ丘 1–5–1
E-mail: †{ktada,nkawahara}@comp.is.uec.ac.jp, ††{yoshimi,clmg,yosinaga}@is.uec.ac.jp

あらまし　本研究報告では，ストリームデータの結合処理を行う Handshake Join の FPGA アクセラレータをマルチノードに拡張する手法を提案し，その性能評価を報告する．マルチノード拡張は，データ通信の 2 つの工夫によって実現する．FPGA 上に複数の Join Core を実装すると共に，FPGA ボード上の DRAM を介して FPGA 間で Join Core を接続する仕組みを導入する．また，データ通信と結合演算をオーバラップすることで，通信に要するオーバヘッドを隠蔽する．これらのデータ配布方法の工夫により，単一の FPGA では実装できなかった大きなウインドウサイズの結合演算が可能となる．最大 16 ノードでの性能評価の結果，マルチノードに拡張した場合においても，結合演算のスループットが維持されるとともに，並列化効果が得られることが確認された．

キーワード　FPGA，ストリームデータ処理，ウィンドウ・ジョイン，Handshake Join，結合演算

Distributed Handshake-Join Processing for Stream Data on Multiple FPGA Nodes

Kousuke TADA†, Naoto KAWAHARA†, Masato YOSHIMI††, Celimuge WU††, and Tsutomu YOSHINAGA††

† The University of Electro-Communications
†† The University of Electro-Communications　Chofugaoka 1–5–1, Chofu-shi, Tokyo, 182–8585 Japan
E-mail: †{ktada,nkawahara}@comp.is.uec.ac.jp, ††{yoshimi,clmg,yosinaga}@is.uec.ac.jp

Abstract　This paper proposes an FPGA-based Handshake join acceleration using multiple-FPGA boards. The proposed multi-node extension devises two ideas. Firstly, join cores implemented on each FPGA are interconnected via DRAM on the FPGA boards. Secondly, join operation is overlapped with data transmission between FPGAs in order to hide communication latency. The proposed architecture performs Handshake join algorithm well on multiple FPGA boards, and a window size can be expanded linearly as the number of FPGAs. Our experiments up to 16 FPGA nodes show that the proposed implementation can handle considerably high input tuple rates, especially at low match rates, without degrading performance even for a large window size.

Key words　FPGA, Stream Data Processing, Window Join Operator, Handshake Join

1. はじめに

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，データストリームに対して 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 がデータストリームに対する演算機構であることから，

IEICE-CPSY/RECONF/VLD,IPSJ-ARC/SLDM 1月研究会 「FPGA応用および一般」
信学技報, vol. 116, no. 416, CPSY2016-112, pp. 37-42, 2017年1月. https://search.ieice.org/

```
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 1: ストリームデータ結合処理クエリ

データの入力元に配置した FPGA などの計算素子の有効性が提案され，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]〜[10]．Sliding-window Join は，2 つのデータストリームを対象に，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，良いスケーラビリティが得られることを確認した．

2. Handshake Join

2 節では，対象とするストリームデータ結合処理のクエリと，Handshake Join のアルゴリズムについて述べる．

2.1 ストリームデータ結合処理クエリ

本研究では，ストリームデータ結合処理の性能評価を行うため図 1 に示すようなクエリを用いる．2 つの入力ストリームを R, S とし，各ストリームからの入力タプル $r \in R$ と $s \in S$ は，それぞれ 4 つの 32bit データの組 $< key1, key2, value1, value2 >$ で構成される．図 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．

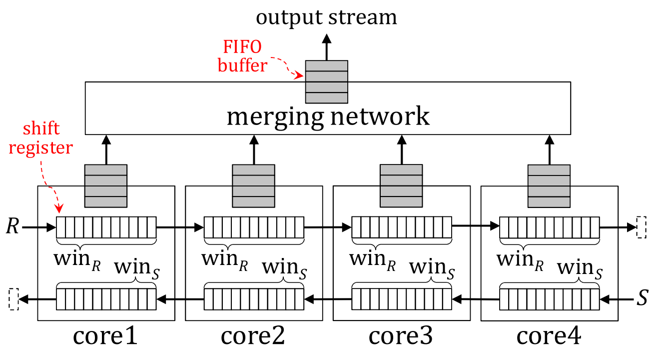

図 2: HandshakeJoin アーキテクチャ

2.2 Handshake Join の設計

一般に無限長であるストリームデータの処理は，入力をウィンドウと呼ぶ一定の区間に分割して処理する．ウィンドウ結合処理では，2 つの入力ストリームからそれぞれウィンドウサイズの入力タプルについて結合演算を行うため，ウィンドウサイズの拡大は計算時間が増大する要因となる．

Teubner と Mueller らは，ストリームデータ結合処理において高度な並列性と高速化を実現する Handshake Join [5] を提案した．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，その原理から 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 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．

しかし，ストリームデータ結合処理の FPGA 実装では，実装可能なウィンドウサイズが FPGA 上のリソース量に制約されてしまい，巨大なウィンドウサイズに対応できない．

3. マルチノード FPGA による HJ の並列化実装

3.1 結合処理のマルチノード拡張

本研究では，ストリーム結合演算の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した．

提案するストリームデータ分散結合処理の概要を図 3 に示す．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．

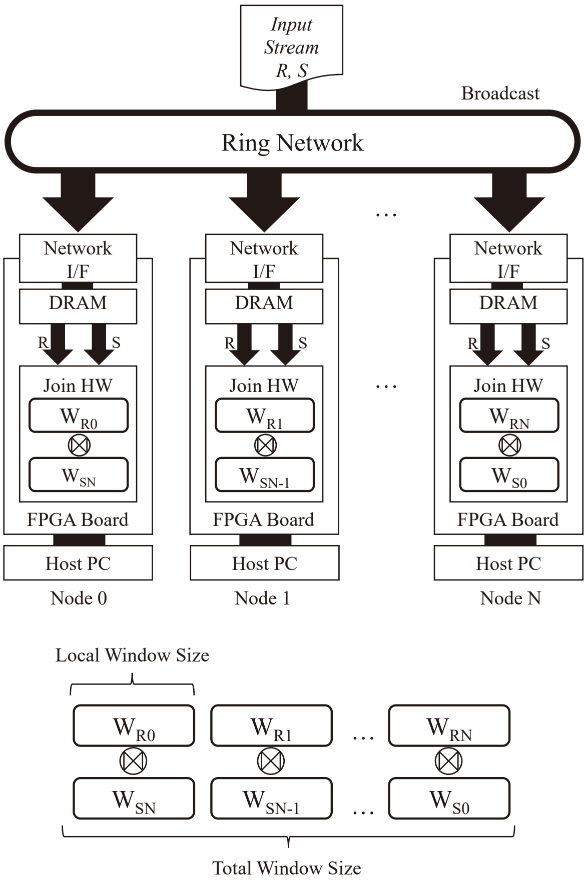

図 3: マルチノード HandshakeJoin の概要

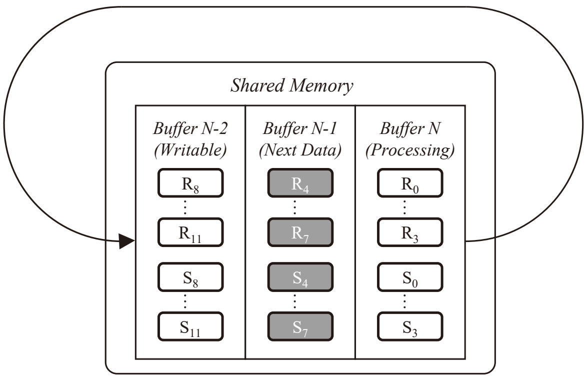

図 4: 共有メモリ内のストリームデータバッファの様子

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共有し，計算ノードが返す処理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．

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

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．

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．データ共有領域を三分割してリングバッファを構築する．図 4 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 に共有される．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 5 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した．(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 を読み出していく．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 を読みだす．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．

3.3 Join Core 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．

マスターノードは，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．ノードは常に 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．

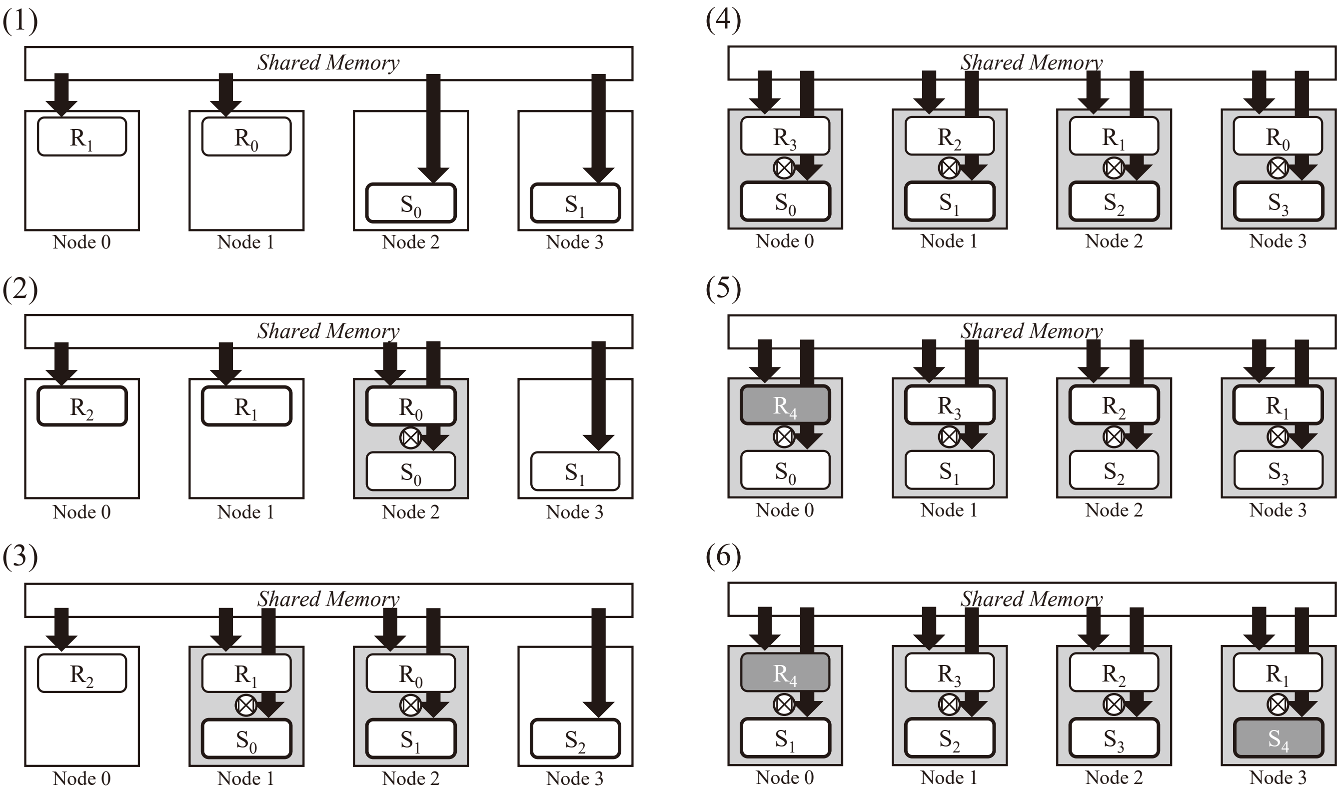

図 5: マルチノード実行時のデータ移動の様子

表 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	10Gbps
	on board Ethernet	1Gbps
SSD	Transcend TS64GSSD370S	
FPGA	AVALDATA APX7142 改	

表 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

Join HW へストリームデータのバッファ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．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．

4. 評　　価

前章で述べたハードウェアを実装し，シングルノード及びマルチノード計算機環境でストリームデータ結合処理の性能評価を行った．計算機ノードの環境を表 1 に，実験に用いた FPGA ボードを表 2 に示す．

4.1 ハードウェア使用量

図 6 に FPGA に実装したハードウェア使用量を示す．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

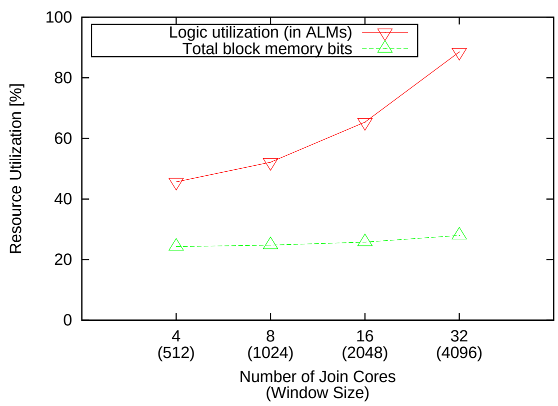

図 6: ハードウェア使用量

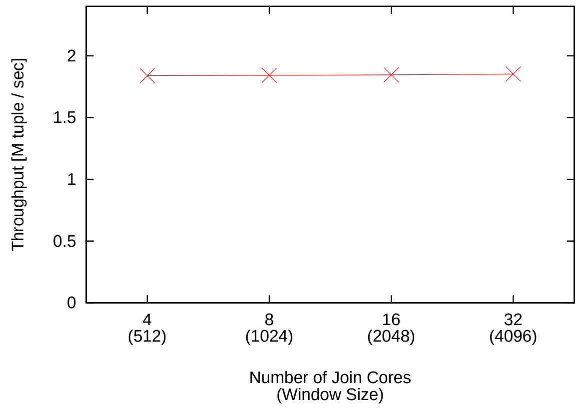

図 7: 入力スループット測定 (wsize=128/core)

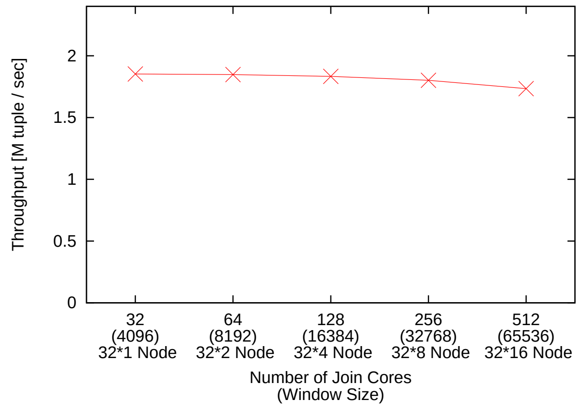

図 8: 入力スループット測定 (wsize=128/core), マルチノード

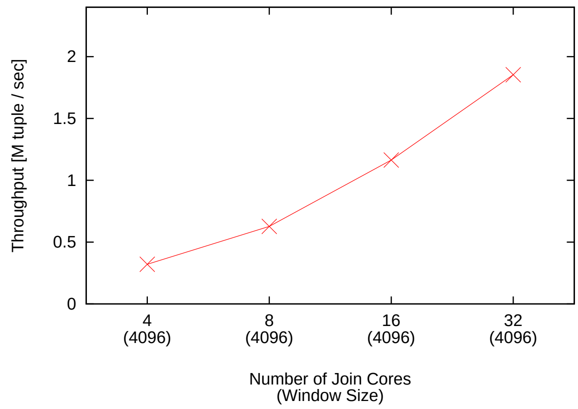

図 9: 入力スループット測定 (wsize=4096)

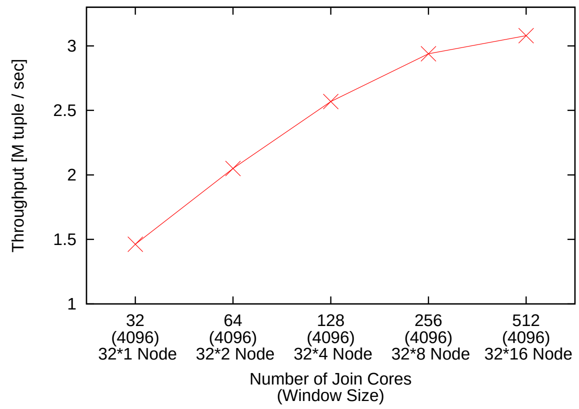

図 10: 入力スループット測定 (wsize=4096), マルチノード

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．

4.2 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．シングルノードの結果を図 7 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 8 に示す．図 7 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た．

同様に (2) 全体のウィンドウサイズ固定の条件での性能を示

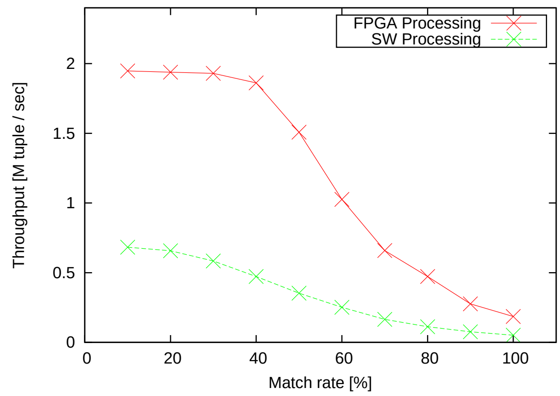

図 11: マッチ率変化による入力スループット比較

す．シングルノードの結果を図 9 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が向上した．

マルチノードの結果を図 10 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 9 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た．

本実験を通してマルチノードを用いたストリームデータの分散結合処理が有効であることが示されたと言える．

4.3 ソフトウェアとの性能比較

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．

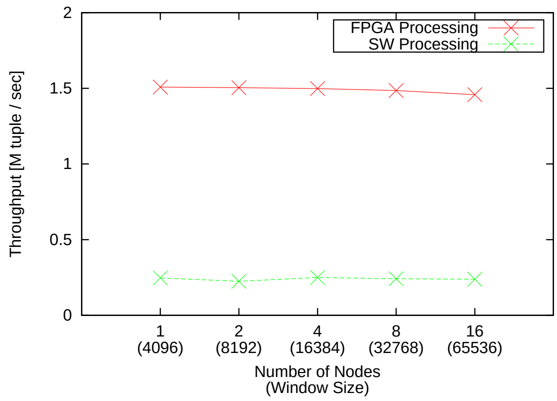

図 12: マルチノード環境における性比能較

図 11 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 7 倍高速であることが確認された．

さらに，図 12 にマルチノードかつマッチ率を 50%に固定した際の，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と実行の制御にも TAMAMO を用いた．TAMAMO による転送と処理の隠蔽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．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．

5. まとめと今後の課題

本研究では，ストリームデータ結合処理の効率向上を目的に，マルチノード分散結合処理機構を提案し，性能測定を行った．測定の結果，ノード数の増大に応じて，性能を維持した最大ウインドウサイズの大幅な拡大，およびウインドウサイズ固定時の処理高速化を実現した．

また，ソフトウェア実行との比較で最大 7 倍程度の高速化を確認し，この比較を通じて，処理性能は FPGA ノードの数に応じてスケールするという結論を得た．

提案手法では，全てのノードが共有メモリとして全てのノードが同一のストリームデータのバッファ領域と内容を保持している．そのためノード数を増加させた際にバッファ領域が拡大し，ノードが利用しないバッファが存在し利用効率が悪い．今後の発展として，ノードごとに保持するデータのパーティショニングを行い，効率よくバッファを行う仕組み等が考えられる．

謝辞　本研究の一部は JSPS 科研費 26330061, 63003088 の助成を受けたものです．

文　　献

[1] J.W. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K.A. Vissers, “A low-latency library in FPGA hardware for high-frequency trading (HFT),” Hot Interconnects, pp.9–16, 2012.
[2] P. Vaidya, J.J. Lee, F. Bowen, Y. Du, C.H. Nadungodage, and Y. Xia, “Symbiote: a reconfigurable logic assisted data stream management system (RLADSMS),” SIGMOD Conference, pp.1147–1150, 2010.
[3] Y. Ahmad and U. Çetintemel, “Data stream management architectures and prototypes,” Encyclopedia of Database Systems, eds. by L. Liu and M.T. Özsu, pp.639–643, Springer US, 2009.
[4] R. Mueller, J. Teubner, and G. Alonso, “Data processing on FPGAs,” PVLDB, vol.2, no.1, pp.910–921, 2009.
[5] J. Teubner and R. Mueller, “How soccer players would do stream joins,” SIGMOD Conference, pp.625–636, 2011.
[6] J. Kang, J.F. Naughton, and S. Viglas, “Evaluating window joins over unbounded streams,” ICDE, pp.341–352, 2003.
[7] Y. Oge, T. Miyoshi, H. Kawashima, and T. Yoshinaga, “An implementation of handshake join on FPGA,” ICNC, pp.95–104, 2011.
[8] Y. Oge, T. Miyoshi, H. Kawashima, and T. 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] Y. Oge, T. Miyoshi, H. Kawashima, and T. Yoshinaga, “Design and implementation of a merging network architecture for handshake join operator on FPGA,” MCSoC, pp.84–91, 2012.
[10] Y. Oge, T. Miyoshi, H. Kawashima, and T. Yoshinaga, “A fast handshake join implementation on FPGA with adaptive merging network,” SSDBM, pp.44:1–44:4, 2013.
[11] K. Naoto，Y. Masato，W. Celimuge, and Y. Tsutotmu, “ネットワーク結合型マルチ fpga ボードを用いた集約演算クエリ処理,” 信学技報，vol.116，no.240，pp.29–34，2016.