動的なロジック切り替えによる結合演算高速化の検討

上田 高徳　伊藤 愛　小原 盛幹

FPGA による処理エンジンは，回路面積の制約に加えて，外部メモリアクセスのレイテンシやスループットを考慮して設計する必要がある．本論文で議論の対象とするデータベースの結合演算は，演算内容が同一であっても，処理対象データの性質やサイズに応じて，処理時間が最も短い処理戦略が変わる．しかし，FPGA 上に複数の戦略をプログラムすると，それぞれの実装が利用できる回路面積が狭くなり，FPGA のリソースを最大限に利用することができず，FPGA チップ全体を利用した場合よりも性能が劣化する．そこで本論文では，ソフトウェア側から FPGA をプログラムしなおすことで，1 つの結合演算が FPGA のリソースを占有できるようにし，より高速にデータベースの結合演算を実行する技法について議論する．

1　はじめに

ハードウェアを設計する際には，利用可能な回路面積を効率よく利用して性能を引き出すことが求められる．回路を占有できる専用ハードウェアによる処理エンジンであったとしても，利用可能な回路面積を最大限に利用して，いかに高い性能を実現できるかが，そのハードウェア処理エンジンの価値を左右するといえる．ただし，一概に性能といっても，電力効率の場合もあれば，レイテンシやスループットの場合もあり，その性能目標はアプリケーションと合わせて議論を行う必要がある．

本論文では関係データベースの結合演算を FPGA (Field Programmable Gate Array) を用いて処理することを考える．大規模データを処理する関係データベースにおいては，処理スループット，すなわちクエリの実行時間がもっとも重要な指標となる．データベースの結合演算は，演算内容が同一であっても，処理対象データの性質やサイズに応じて，処理時間が最も短い処理戦略が変わる．しかし，FPGA 上に複数の戦略をプログラムすると，それぞれの実装が利用できるハードウェア面積が狭くなり，1 つの実装で FPGA チップ全体を利用した場合よりも利用できるリソース量が少なくなり性能が劣化する．

本論文で対象とするのは，キーが等しい行を結合する等価結合である．等価結合の代表的なアルゴリズムであるハッシュ結合とソート・マージ結合の実行時間を予測し，高速に実行できると予想される方の結合回路を FPGA にプログラムすることで，FPGA チップ全体のリソースを最大限に利用してより高速に結合演算を実行できることを示す．そして，実機における提案手法の効果を立証するために，計算機のバスに接続された FPGA カードを用いて性能を評価した．

データベースの演算に FPGA を用いる試みは，近年になって始まったばかりである [7]．データストリームに対するウィンドウ結合 [10] [11] は，FPGA がデータストリームに対して効果的に処理できることを示す良い例である．しかし，本論文が対象とするのは永続化された，あるいはメモリ上に存在するテーブルに対する結合演算であり，半永続的に到着するデータストリームに対するウィンドウ結合とは演算が異なる．FPGA による結合演算機構 [4] や，クエリ処理エンジン [1] [2] [8] [9] も開発されているが，本論文のような結合演算の選択方法については議論されていない．[3] は本論文と同様に動的に FPGA をプログラムするこ

This is a non-refereed paper.

Ueda, Ito, Ohara, 日本 IBM 東京基礎研究所, IBM Research - Tokyo.

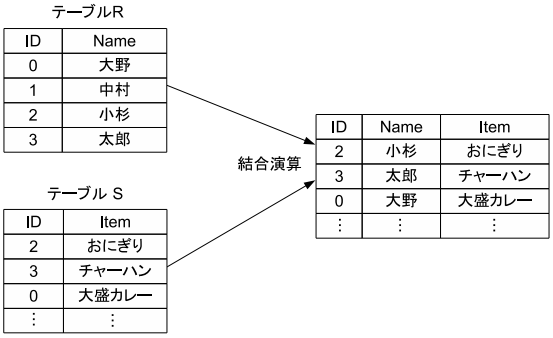

図 1　同一の ID を結合する結合演算

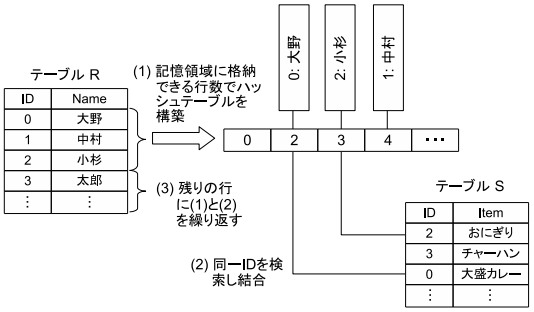

図 2　ハッシュ結合の処理の流れ

とを提案しているが，対象が選択演算であり，結合演算ではない．

以下，2 節では，結合演算の基本とアルゴリズムについて確認する．3 節においては，ハードウェア処理エンジンにおいて結合演算を用いる場合の課題について議論する．4 節においては，結合演算を切り替えるために，実行時間を予測するモデルについて述べる．5 節において実験結果を述べ，6 節において関連研究について議論する．そして，7 節においてまとめる．

2　結合演算とアルゴリズム

本節では本論文の対象である結合演算とそのアルゴリズムについて説明する．結合演算は，関係代数における基本演算のひとつであり，複数のテーブル (Relation) から，指定された条件にマッチした行（Tuple）同士を結合して出力する演算である．以下，本論文ではキーが等しいものを結合するものとする．また，テーブル R と S の行数をそれぞれ $|R|$，$|S|$ と書くこととし，$|R| \leq |S|$ とする．

図 1 はテーブル R とテーブル S から ID 列が同一の行を結合する結合演算の例を示している．結合演算は関係代数の演算子であるから，その定義に計算アルゴリズムは含まれない．よって同一の結合演算を実行するためのアルゴリズムが様々に考えられ，代表的なものにハッシュ結合とソート・マージ結合がある [6]．

ハッシュ結合では図 2 のように，(1) 容量が小さいテーブル R に対して ID 列をキーとしてハッシュテーブルを構築する．この時，テーブル R 全体に対してハッシュテーブルを構築すると記憶領域が不足する場合，テーブル R のまだ処理していない行のみでハッシュテーブルを構築する．そして，(2) もう一方のテーブル S の行を順に読み込み，各行の ID 列でハッシュテーブルを検索し，同一の ID を持つ行を発見して結合する．必要に応じて，(3) テーブル R の残りの行に対して (1) と (2) を繰り返す．ここで，ハッシュテーブルが記憶領域を使い切った時に格納できる行数を C とすると，図 2 (3) の繰り返しは $\left\lceil \frac{|R|}{C} \right\rceil$ 回発生し，その度にテーブル S の全ての行が読まれる．テーブル R の各行は一度しか読まれないから，読まれる行数の合計は $\left\lceil \frac{|R|}{C} \right\rceil |S| + |R|$ となり，実行時間のオーダーは，

$$\mathrm{O}\left(|R||S|\right) + \mathrm{O}\left(|R|\right) \tag{1}$$

となる．

ソート・マージ結合では図 3 のように，まず ID 列をキーとみなして両方のテーブルをそれぞれマージソートした後，両テーブルを昇順あるいは降順に比較してマージすることにより，同一の ID を持つ行を発見して結合する．マージソートの計算量は $\mathrm{O}\left(|R| \log |R|\right)$ と $\mathrm{O}\left(|S| \log |S|\right)$ であり，マージにおいては両テーブルを読み込むことになるから，ソート・マージ結合の実行時間のオーダーは

$$\mathrm{O}\left(|R| \log |R|\right) + \mathrm{O}\left(|S| \log |S|\right) + \mathrm{O}\left(|R| + |S|\right) \tag{2}$$

となる．

以上のことから，ハッシュ結合とソート・マージ結合の実行時間は，テーブルのサイズに依存することになる．例えば，ハッシュ結合においては，小さい方のテーブルのハッシュテーブルが記憶領域に収まるならば，図 2 (3) の繰り返しはなく，両テーブルに対する

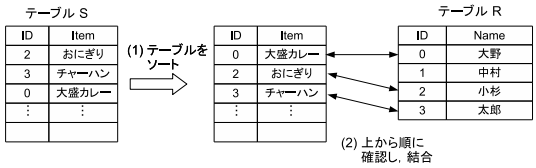

図 3　ソート・マージ結合の処理の流れ

一度の読み込みのみで結合演算が完了する．しかし，テーブルのサイズが大きく，ハッシュテーブルが記憶領域に収まらない場合は，図 2 (3) の繰り返し回数が増え，実行時間が増加する．結合演算においては，入力に与えられるデータに応じて計算時間が変わるため，適切な結合アルゴリズムを用いることが高速な実行を実現する鍵である．

3　FPGA による処理エンジンと結合演算

前節においてはソフトウェアの視点から結合演算を説明したが，本節では，FPGA で結合演算を実行する場合でも処理対象のテーブルサイズに応じて結合演算の処理時間が変わることを確かめる．そして，本論文のテーマである，結合演算の内容に応じて実装を切り替える利点について議論する．

結合演算に用いるデータ構造の記憶先として FPGA の外部にある記憶領域を利用すると，処理中にデータ構造へアクセスした際のレイテンシが問題になる．アクセスレイテンシにより処理が止まることで，1 クロックあたりに処理できるテーブルの行数が減少し，スループットが低下する．よって本論文では，外部メモリはデータの入出力に利用するのみとし，チップ内やチップに近い低レイテンシでアクセスできる記憶領域にデータ構造を配置することを考える．

前節で確認したように，ハッシュ結合においてはハッシュテーブルのサイズを大きく確保するほど，実行時間を短くできる．よって，低レイテンシでアクセスできる記憶領域をできるだけ大きく確保することが目標になる．また，ソート・マージ結合においてはソートの計算時間が支配的であることは明らかである．本論文においては我々の開発したソートツリー [5] を用いる．図 4 に 8 ウェイのソートツリーを示した．このソートツリーは 8 つのソート済み配列から

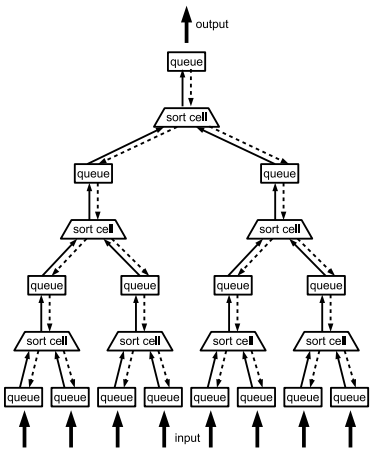

図 4　8 ウェイのソートツリー

1 つのソート済み配列を生成できる．未ソートの配列に対して，ソートツリーを繰り返し適用することで，ソート済みの配列を生成することができる．繰り返し回数は大きなウェイ数のソートツリーを用いるほど少なくなるので，やはりできるだけ大きなソートツリーを FPGA に確保することが目標になる．

ここで，ASIC のような専用ハードウェア処理エンジンでは，アルゴリズムごとに専用の回路を実装する必要があることに注意すべきである．ハッシュ結合とソート・マージ結合の回路の双方をハードウェア上に配置すると，それぞれの実装が利用できる回路面積が狭くなり，ハードウェアチップを最大限に利用することができず，チップ全体を利用した場合よりも，使用できるレジスタ数や記憶容量に制約を受ける．よって，使用する結合戦略を決定し，動的に回路を変更してチップ全体を利用すれば，ハードウェアリソースを最大限に利用できるという着想が得られる．動的な回路の変更は FPGA にユニークな特徴であり，ASIC のような専用ハードウェアを用いた処理エンジンでは不可能である．本論文では，ハッシュ結合とソート・マージ結合に対して議論するが，他の結合アルゴリズムにおいても適用できる．

4　結合回路の実行時間予測

本節ではハッシュ結合とソート・マージ結合のどちらが高速か決定する手法について議論する．まず，結

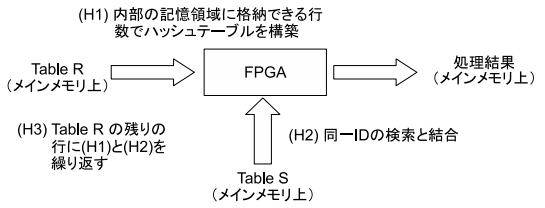

図 5　FPGA によるハッシュ結合の実行の流れ．FPGA 内部に記憶できる量だけテーブル R を読み込み，テーブル S を読み込んで結合演算を行う

合対象のテーブルサイズを元に両結合方法の実行時間を予測するためのモデルを説明する．実行時間を予測することができれば，処理時間が短い方の実装を選択すればよい．以下では，ある結合演算のステップ s において，FPGA が単位時間あたりに処理できるテーブルの行数を v_s とする．v_s はメモリ帯域や行のサイズ，FPGA の周波数によって変わることになる．詳細は 4.3 項で議論する．

4.1　ハッシュ結合の実行時間

ハッシュ結合では，FPGA の内部記憶領域を利用してデータを格納し，ハッシュ結合を行う．まず，(H1) 容量が小さいテーブル R を FPGA 内部に格納できる量だけ読み込んだ後，(H2) もう一方のテーブル S を読み込み結合する．そして，(H3) テーブル R が FPGA 内部に収まらない場合は (H1) と (H2) を繰り返す．

ここで，(H1) においては (H3) での繰り返し回数によらず，処理全体で テーブル R の各行が 1 度ずつ読まれることになる．よって，処理全体における (H1) の処理時間 [s] の合計 T_{H_1} は以下の式で表される．

$$T_{\mathrm{H}_1} = \frac{|R|}{v_{\mathrm{H}_1}} \tag{3}$$

(H2) においては (H3) の繰り返しごとに，テーブル S の全ての行を読み込むことになる．FPGA 内に記憶可能なテーブルの行数を C としたとき，(H3) の繰り返し回数は $\left\lceil \frac{|R|}{C} \right\rceil$ 回となる．よって，結合処理全体を通した (H2) の実行時間の合計は以下の式で表される．

$$T_{\mathrm{H}_2} = \left\lceil \frac{|R|}{C} \right\rceil \frac{|S|}{v_{\mathrm{H}_2}} \tag{4}$$

最後に，処理の開始やステップの切り替えに定数時間の制御オーバーヘッド E_{H} が掛かるとする．以上の結果から，ハッシュ結合の実行時間 T_{H} は

$$\begin{aligned} T_{\mathrm{H}} &= T_{\mathrm{H}_1} + T_{\mathrm{H}_2} + E_{\mathrm{H}} \\ &= \frac{|R|}{v_{\mathrm{H}_1}} + \left\lceil \frac{|R|}{C} \right\rceil \frac{|S|}{v_{\mathrm{H}_2}} + E_{\mathrm{H}} \end{aligned} \tag{5}$$

で計算することができる．

4.2　ソート・マージ結合の実行時間

前節で説明したように，FPGA にソートツリーを構築し，以下の手順でソート・マージ結合を実現する．(S1) テーブル R をソートする．(S2) テーブル S をソートする．(S3) ソート済みの 2 つのテーブルの行を順に確認してマージする．

ここで，(S1) のテーブル R をソートする際のイテレーション回数は，K ウェイのソートツリーを用いれば $\lceil \log_K |R| \rceil$ 回となる．(S2) についても同様に $\lceil \log_K |S| \rceil$ 回となる．各イテレーションにおいてはテーブルの全ての行を読み込むことになる．よって，(S1) と (S2) の実行時間は以下の式で表される．

$$T_{\mathrm{S}_1} = \frac{|R| \lceil \log_K |R| \rceil}{v_{\mathrm{S}_1}} \tag{6}$$

$$T_{\mathrm{S}_2} = \frac{|S| \lceil \log_K |S| \rceil}{v_{\mathrm{S}_2}} \tag{7}$$

次に (S3) については，昇順にソートされたテーブルを順に確認して，マージソートのマージフェーズと同様の手順で結合する．この時，注目している行について，(S3-a) テーブル R と S のキーが同一，(S3-b) テーブル R のキーの方が小さい，(S3-c) テーブル S のキーの方が小さい，の 3 パターンが考えられる．

(S3-a) は同一のキーを持つ行がテーブル R と S の

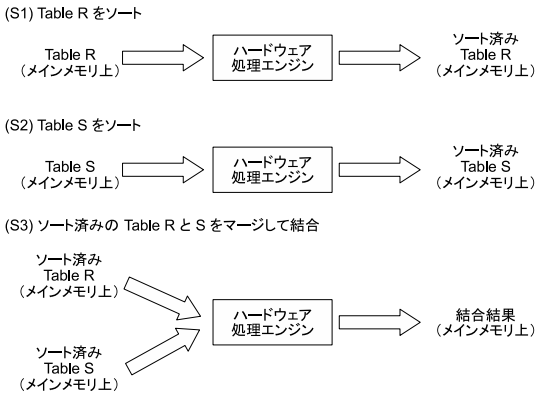

図 6　ハードウェアによるソート・マージ結合の流れ

両方に見つかったことを意味するから，結合演算の出力となる．何個の行が結合されるかは入力データに依存するため事前には分からないが，テーブルの統計を用いる従来手法で予想することができる [6]．そこで，(S3-a) が発生する回数，すなわち結合される行の個数を M とする．すると，(S3-b) が発生する回数は $|R| - M$ 回となる．また，(S3-c) も同様に $|S| - M$ 回発生することになる．従って，ステップ S3 の実行時間 T_{S_3} は以下の式で表すことができる．

$$T_{\mathrm{S}_3} = \frac{M}{v_{\text{S3-a}}} + \frac{|R| - M}{v_{\text{S3-b}}} + \frac{|S| - M}{v_{\text{S3-c}}} \tag{8}$$

最後に，処理の開始やステップの切り替えに定数時間の制御オーバーヘッド E_{S} が掛かるとする．以上の結果から，ソート・マージ結合の実行時間 T_S は

$$\begin{aligned} T_{\mathrm{S}} &= T_{\mathrm{S}_1} + T_{\mathrm{S}_2} + T_{\mathrm{S}_3} + E_{\mathrm{S}} \\ &= \frac{|R|\lceil \log_K |R| \rceil}{v_{\mathrm{S}_1}} + \frac{|S|\lceil \log_K |S| \rceil}{v_{\mathrm{S}_2}} \\ &\quad + \frac{M}{v_{\text{S3-a}}} + \frac{|R| - M}{v_{\text{S3-b}}} + \frac{|S| - M}{v_{\text{S3-c}}} + E_{\mathrm{S}} \end{aligned} \tag{9}$$

で計算することができる．

4.3　回路の再プログラムとパラメータ v

以上の議論から，処理テーブルのサイズとパラメータとなっている v と E が分かれば，(5) 式と (9) 式からハッシュ結合とソート・マージ結合の実行時間が分かり，処理時間が短い戦略を選択することができる．

本論文では，E については経験的に求めた値を用いることとし，以下，我々の実装を前提とした時のパラメータ v について説明する．本論文で用いた FPGA カードは計算機の専用バスに接続され，FPGA から計算機のメインメモリに直接アクセスできる．理論上，メインメモリと 128bit のデータを各クロックごとに送受信できるインターフェースを持つ．また本論文の実験では，テーブルの 1 行は 1 つの 64bit キーと 1 つの 64bit 値の合計 128bit から構成している．よって，メモリ帯域がボトルネックでなく，必ず各クロックでデータを受信できるとすれば，FPGA は各クロックに 1 行ずつのデータ読み込むことができる．以下では FPGA のクロック周波数 [Hz] を f とし，v を f で表現することを考える．

我々のハッシュ結合の実装では，(H1) でのテーブル 1 行の処理に 2 クロック必要である．1 クロック目でキーがすでに格納されているか確認し，格納されていなかった場合は 2 クロック目で格納するためである．そして，(H2) においては 1 クロックで 1 行処理できる．以上から，$v_{\mathrm{H}_1} = f/2$，$v_{\mathrm{H}_2} = f$ となる．

ソート・マージ結合の実装においては，(S1) と (S2) については 1 クロックで 1 行を処理できるようになっている．また，(S3) については (S3-a)，(S3-b)，(S3-c) のいずれの場合にも 1 クロックで処理を 1 ステップ進めることができるようになっている．ただし，(S3-a) については，両方のテーブルから行を読み込むことになるため，倍のメモリ帯域を消費し，実際には 2 クロック必要になると考えられる．よって，$v_{\mathrm{S}_1} = v_{\mathrm{S}_2} = v_{\text{S3-b}} = v_{\text{S3-c}} = f$，$v_{\text{S3-a}} = f/2$ となる．

さて，上記の説明においては，FPGA は各クロックにおいて必ずデータを受け取ることができるとしている．しかし，実機における処理においてはメモリ帯域やバスのボトルネックにより，必ずしもクロック毎にデータを受け取れるとは限らない．よってパラメータ v は実機上の測定結果に基づいてキャリブレーションする必要がある．本論文では以下の実験において上記パラメータをそのまま用いている．実機を用いた v の決定とその時のモデル精度の評価は今後

の課題である．

5 評価実験

本節ではまず，4 節で導いた計算時間の式を用いてハードウェア実装を切り替える利点について確認する．そして，切り替えることで性能上の利益が実機でも得られることを確かめるため，計算機に接続された FPGA カードを用いて性能を評価した．

5.1 回路再プログラムの利点の確認

まず，回路を切り替える利点について 4 節で導いた実行時間の式に基づいて考察する．ハッシュ結合の実装において記憶領域に格納できる行数を 10 万行，8 万行，4 万行の 3 種類，ソート・マージ結合で用いるソート実装のウェイ数を 8, 4, 2 の 3 種類として，それぞれの処理時間について式 (5) と式 (9) を用いて比較する．以下では，$E_{\mathrm{H}} = 0, E_{\mathrm{S}} = 0, M = 0$ としている．また通常，ハッシュ結合では，小さい方のテーブルでハッシュテーブルを構築した方が性能が良いが，以下では常に固定長のテーブル，すなわち 20 万行と 1000 万行に対してハッシュテーブルを構築している．

図 7 はハッシュテーブルの構築対象であるテーブルを 20 万行とし，もう一方のテーブルの行数を変化させた場合の式 (5) と式 (9) による理論実行時間である．20 万行は記憶領域に格納できる行数である 10 万行に近いため，ハッシュ結合の方がソート・マージ結合よりも有利になる．具体的には，記憶領域に格納できる容量が 10 万行の場合，ハッシュ結合におけるテーブル再構築回数は 2 回であり，8 万行の時は 3 回，4 万行の時は 5 回となる．4 節で確認したとおり，記憶領域の容量が少なくなるほど構築回数が増えるため，処理時間が長く掛かることが確認できる．よって，ハッシュ結合が優位な場合においては，できるかぎり高速記憶領域をハッシュ結合に利用することが必要である．

これに対して，図 8 は一方のテーブルを 1000 万行とし，もう一方のテーブルの行数を変化させた場合の式 (5) と式 (9) による理論実行時間である．記憶領域に格納できる行数よりも大きいため，ソート・マージ結合の方がハッシュ結合よりも有利になる．もっとも

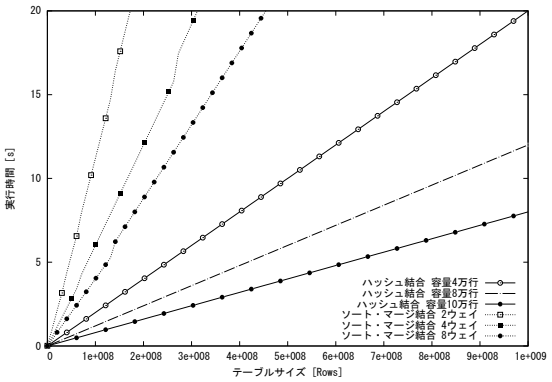

図 7　片方のテーブルが 20 万行の時の理論実行時間．左上から順にソート・マージ結合の 2 ウェイ，4 ウェイ，8 ウェイ，続いてハッシュ結合のメモリ容量 4 万行，8 万行，10 万行．

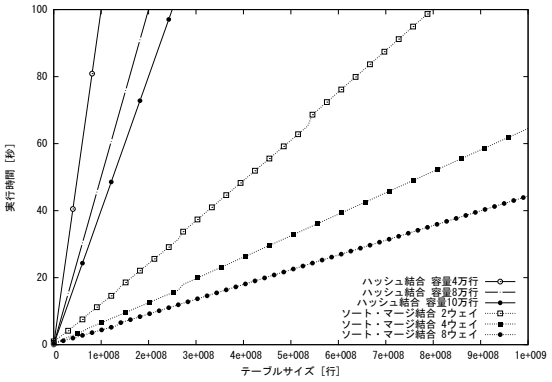

図 8　片方のテーブルが 1000 万行の時の理論実行時間．左上から順にハッシュ結合のメモリ容量 4 万行，8 万行，10 万行，そしてソート・マージ結合の 2 ウェイ，4 ウェイ，8 ウェイ．

性能が良いのは 8 ウェイの場合であり，2 ウェイの場合は，もっとも性能が悪くなる．

我々のハッシュ結合の実装では 10 万行の格納領域を実現するためには，FPGA のロジック数の 86%を使用する必要があった．ハッシュ結合とソート・マージ結合を同一の FPGA に共存させると，ハッシュ結合用の記憶容量が圧迫され，格納容量が低下する可能性がある．FPGA を回路を占有して結合演算を実行することで，そのような低下の可能性を排除することができる．我々の FPGA の再構成時間は，ハッシュ

結合，ソート・マージ結合のどちらであっても 1.44 秒であった．実験結果においては，ハッシュ結合の 8 ウェイと 4 ウェイ，また，ソート・マージ結合の容量 10 万行と 8 万行の実行時間において，特にテーブルサイズが大きい場合に，1.44 秒以上の差がある．よって，FPGA を 1 つの結合実装で占有することで，より高速に結合演算を実行可能であると考えられる．

5.2 実ハードウェアでの結合演算

次に，実機でも議論が成り立つことを確認するため，FPGA カードを用いて実験を行った．実験に用いた FPGA カードは表 1 の計算機の専用バスに接続され，FPGA から計算機のメインメモリに直接アクセスできる．Linux 上からハードウェアに対して，結合演算対象の 2 つのテーブルのメモリアドレスと，結果の格納先のアドレス，計 3 つのメモリアドレスを FPGA に送信する．すると FPGA がメインメモリに直接アクセスして結合演算を実行する．ハッシュ結合の実装は，FPGA 内に 10 万行のデータを格納でき，ソート・マージ結合に用いたソートツリーは 8 ウェイである．本実験での理論値のグラフにおいては，$E_{\mathrm{H}} = 0.0001[\mathrm{s}], E_{\mathrm{S}} = 0.005[\mathrm{s}], M = 0$　とした．

図 9 は両方のテーブルを同一サイズとして変化させた時の実験結果である．テーブルの行数が少ない時は，ハッシュ結合が優位であるが，テーブルサイズが大きくなるとソート・マージ結合が優位になることも確認できる．ハッシュ結合においては実行時間を概ね予測できているが，ソート・マージ結合においては誤差が見られる．なお，2 つの実装を使い分けるという観点からは，特に 2 つの曲線が交わる周辺のテーブルサイズにおいて正確な実行時間の予測が必要になり，それ以外のテーブルサイズにおいて常に正確な実行時間予測が求められるわけではない．様々なテーブルサイズにおいて本モデルの精度を評価することと，パラメータ v の実機を用いた決定方法の検討が今後の課題である．

6 関連研究

専用ハードウェアを用いたデータベース処理は古くからの研究課題であるが，動的再構成が可能な FPGA をデータベースの演算に使用する研究は近年になって始まったばかりである [7]．データストリームに対するウィンドウ結合 [10] [11] は，FPGA がデータストリームに対して効果的であることを示す良い例である．しかし，本論文が対象するのは永続化あるいは，すでにメモリ上に存在するテーブルに対する結合演算であり，半永続的に到着するデータストリームとは演算が異なる．FPGA による結合演算機構 [4] や，クエリ処理エンジン [1] [2] [8] [9] も開発されているが，本論文のように結合演算の選択方法については議論されていない．[3] は本論文と同様に動的に FPGA をプログラムすることを提案しているが，対象が選択演算であり，結合演算に適用できるものではない．

表 1　評価実験の環境

CPU	POWER7 3.55GHz
Memory	256GB
OS	Fedora release 18 (3.9.4-200.fc18.ppc64p7)

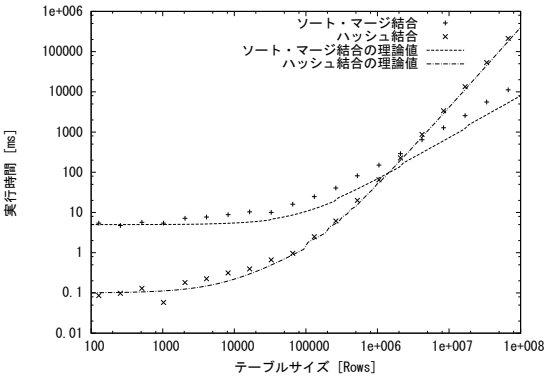

図 9　FPGA による結合演算の性能比較．破線はモデルによる推測値，＋はソート・マージ結合，×印はハッシュ結合の実測値

7 おわりに

本論文では，結合演算の回路をクエリ処理時に FPGA にプログラムし直し，1 つの結合演算回路が FPGA のハードウェアリソースを占有できるようにすることで，より高速に結合演算を実行する方法について議論した．データベース分野において，クエリ

の性質に応じて結合演算のアルゴリズム決定することは従来からの研究分野である．しかし，ハードウェアの回路面積に着目して同様の最適化を考える必要があることを示したことが，本研究のユニークな点である．本論文ではハッシュ結合とソート・マージ結合を対象に議論を行ったが，同様のアイデアを他の結合アルゴリズムに適用することも可能である．今後，実際のデータベースエンジンに組み込むなどして，本切り替え手法の利点を実証すると共に，対象とする結合演算アルゴリズムを増やして詳しい性能評価を行いたいと考えている．本研究がハードウェア処理エンジンの発展の一助になれば幸いである．

謝辞 本研究で利用した FPGA カードの主開発チームである IBM Research - Austin の研究チームに感謝する．

参 考 文 献

[1] Casper, J. and Olukotun, K.: Hardware Acceleration of Database Operations, *FPGA 2014*, FPGA '14, New York, NY, USA, ACM, 2014, pp. 151–160.
[2] Chung, E. S., Davis, J. D., and Lee, J.: LINQits: Big Data on Little Clients, *Proc. ISCA 2013*, 2013, pp. 261–272.
[3] Dennl, C., Ziener, D., and Teich, J.: Acceleration of SQL Restrictions and Aggregations through FPGA-Based Dynamic Partial Reconfiguration, *FCCM 2013*, April 2013, pp. 25–28.
[4] Halstead, R., Sukhwani, B., Min, H., Thoennes, M., Dube, P., Asaad, S., and Iyer, B.: Accelerating Join Operation for Relational Databases with FPGAs, *Proc. FCCM 2013*, pp. 17–20.
[5] Hayashizaki, H., Ito, M., Ueda, T., and Ohara, M.: Variable-Length-Key Sorter on FPGA using Tries, *Proc. IWLS 2014*, 2014.
[6] Hector Garcia-Molina, Jeffrey D. Ullman, J. D. W.: *Database Systems: The Complete Book*, 2002.
[7] Mueller, R. and Teubner, J.: FPGA: What's in It for a Database?, SIGMOD '09, 2009, pp. 999–1004.
[8] Mueller, R., Teubner, J., and Alonso, G.: Streams on Wires – A Query Compiler for FPGAs, *Proc. VLDB Endow.*, Vol. 2, No. 1(2009), pp. 229–240.
[9] Najafi, M., Sadoghi, M., and Jacobsen, H.-A.: Flexible Query Processor on FPGAs, *Proc. VLDB Endow.*, Vol. 6, No. 12(2013), pp. 1310–1313.
[10] Oge, Y., Miyoshi, T., Kawashima, H., and Yoshinaga, T.: An Implementation of Handshake Join on FPGA, *Proc. ICNC 2011*, Nov 2011, pp. 95–104.
[11] Teubner, J. and Mueller, R.: How Soccer Players Would Do Stream Joins, *SIGMOD 2011*.