ストリーム型関係演算処理方式

Stream-oriented Relational Database Operation Scheme

清木 康† 長谷川 隆三 雨宮 真人

日本電信電話公社 武蔵野電気通信研究所

1．まえがき

E.F. Codd によって提案された関係モデル〔1〕，〔2〕は，データ独立性，アクセス対称性，非手続き的ユーザ・インタフェースを実現する優れたデータモデルとして注目されている。しかし，関係モデルは，その基本演算である関係演算の処理効率の点で問題を抱えており，この点が関係データベース・システムを実現する際の最大の課題である。

関係演算処理を高速化するために，現在までに数多くの関係演算処理方式（アルゴリズムおよびデータベースマシン・アーキテクチャ）が提案されてきた〔4〕〔5〕,〔6〕,〔7〕。しかし，それらのどの関係演算処理方式も，データベースのような巨大なデータを扱う場合に必要となる計算機資源の管理，例えば，結合演算結果の中間リレーションのメモリ・オーバーフローの制御等の問題を解決していない。

本論文では，リレーションに対する関係演算処理をストリームに対する関数の実行に対応させ，関数の引評価を先行・遅延評価（eager and lazy evaluation）〔8〕を用いて行うことにより，有限の計算機資源の中でメモリ・オーバーフローを引き起こすことなく関係演算処理を行い，さらに，問合せを構成する複数の関係演算間でパイプライン並列処理の効果を最大限に引き出す方式を提案する。また，本方式を関数型言語 Valid〔9〕により記述したプログラムを提示する。

過去に提案されたほとんどの関係演算処理方式では複数の関係演算から構成される問合せを処理する場合に，関係演算間で引き渡されるデータの大きさの単位（granularity）はリレーションであった。リレーションを単位とした場合には，図1に示すように1つの関係演算が演算結果の中間リレーション（intermediate relation）を完全に生成し終えた後に，その中間リレーションを演算対象とする関係演算が起動される。従って，問合せを構成する関係演算間でのパイプライン性はなく，また，中間リレーションのメモリ・オーバーフローに対処するために主記憶と二次記憶間でスワッピングを行う等のオーバーヘッドが問題となる。そこで，関係演算間で引き渡されるデータの大きさの単位をリレーションを分割したページとし，関係演算をページ単位に起動することにより関係演算間でのパイプライン性を引き出すことが考えられる〔3〕。ページを単位とした場合，問合せを構成する各関係演算に対応させてプロセッサを配置することにより，関係演算間でのパイプライン処理による並列性が引き出される。データ駆動制御によりページ単位でのパイプライン処理を行う方式〔5〕が提案されているが，この方式では次のような問題点が生じる。

（1）ページ単位に関係演算が起動されると，図2に示すように，問合せを構成している各関係演算ノード内に大量のデータが残され，リレーションを単位とした場合よりもかえって大きな中間データとなる可能性がある。従って，計算機資源の管理が複雑になる問題は解決されない。

（2）問合せを構成する各関係演算ノードにプロセッサの台数およびメモリ領域を動的かつ最適に割り当てることはできない。従って，関係演算間のパイプラインを流れるデータ量を制御できないので，パイプラインによる並列性を十分に引き出すことはできない。

このように，関係演算間で引き渡されるデータの単位をページとして単にデータ駆動制御を行うだけではパイプライン処理による効果を十分に引き出すことはできず，また，計算機資源の管理の問題は解決されない。

本論文で提案する方式は，関係演算間で引き渡されるデータの単位をページとし，データ駆動型の制御とデマンド駆動型の制御を組み合わせた新しい関係演算処理方式である。本方式は，有限資源の中でメモリ・オーバーフローを起こすことなく関係演算を実行し，さらに，パイプラインによる並列性を最大限に引き出すことを可能にする。

2．基本機構

ここでは，本関係演算処理方式の基本的な機構を提示する。

図1に示す問合せを処理する場合，図3に示すように，各関係演算ノードは上位の関係演算ノードからリレーションの生成要求（デマンド）を受け取ると入力バッファから演算対象リレーションの1ページを読み出し，関係演算を実行する。そして，その結果を出力バッファ（上位関係演算ノードの入力バッファ）へ転送する。また，入力バッファからページを読み出した時点で，この関係演算ノードでは入力バッファに空きが生じるので，それを埋めるために下位の関係演算ノードへデマンドを先出ししておく。1デマンドに対して，これらの動作を1ページ分の出力データを生成するまで繰り返す。この機構により，1リレーションを

† 現、筑波大学電子・情報工学系

演算対象とする単項関係演算（選択演算（selection），制約演算（restriction），射影演算（projection）），また，2リレーションを演算対象とする2項関係演算（結合演算，準結合演算（semi-join），直積　（cartesian product），和演算（union），積演算（intersection），差演算（difference））をメモリ・オーバーフローなく，パイプラインの効果を最大限に引き出して処理することが可能となる。特に結合演算，和演算（これらの関係演算は，演算対象のリレーションよりも大きな中間リレーションを生成する可能性がある），直積のように大きなリレーションを生成する関係演算の処理では有効性を発揮する。

各関係演算ノードは，単項関係演算ノードの場合には1つ，2項関係演算ノードの場合には2つの論理的な入力バッファを持ち，そのバッファはその関係演算ノードの下位の関係演算ノードの出力バッファとして扱われる。

（1）単項関係演算

単項関係演算ノードは上位の関係演算ノードからデマンドを受け取ると，1ページ分の出力データを生成するまで以下の動作を繰り返す。まず，入力バッファから1ページを読み出す。このとき，入力バッファには空きが生じるので，それを埋めるためにこの関係演算ノードは下位の関係演算ノードにデマンドを送出しておく。そして，入力バッファから読み出したページに対して関係演算を実行し，演算結果を出力バッファ（上位関係演算ノードの入力バッファ）へ格納する。ただし射影演算はタプル間で重複の排除を必要とするので，出力バッファに格納された演算結果のデータは，上位の関係演算ノードによって読み出された後も消滅することはできない。従って，射影演算ノードの出力バッファに関してだけは，中間リレーションをすべて格納できる大きさが必要となる。射影演算ノードは上位関係演算ノードからのデマンドにより，入力バッファから1ページを取り出し，すでに得られている射影演算の結果と重複しているタプルを排除した後，出力バッファへ格納する。

（2）2項関係演算

2項関係演算では，一方の演算対象リレーション（アウタ・リレーション）の各ページと他方（インナ・リレーション）の全ページを付き合わせることにより処理が完了する。2項関係演算ノードは以下

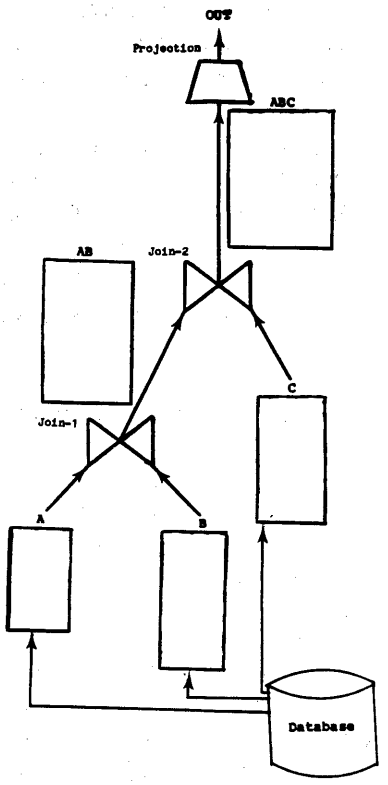

図1　問合せ処理
Fig.1　Query Processing

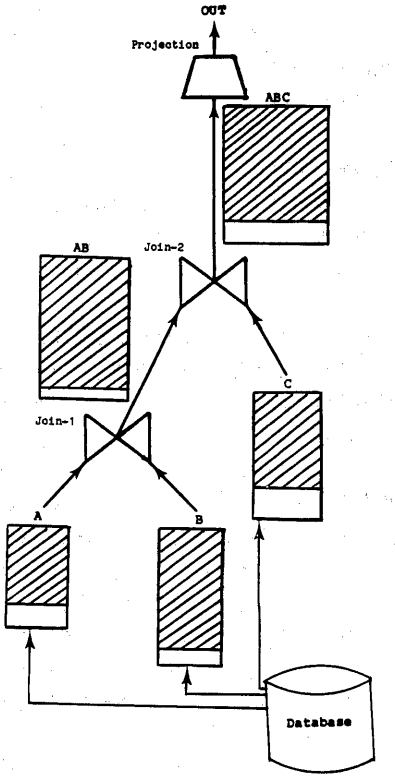

図2　データ駆動制御による関係演算処理
Fig.2　Relational Operation Exection by Data-driven Control

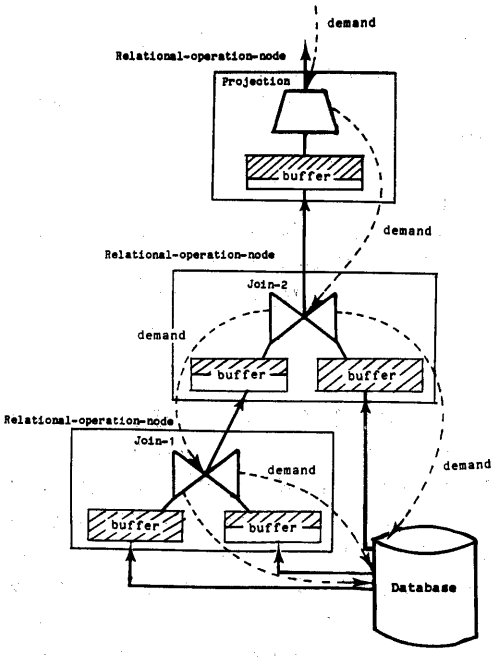

図3　先行・遅延評価を用いた関係演算処理
Fig.3　Relational Operation Execution with Eager and Lazy Evaluation

のアルゴリズムで動作する。

①上位関係演算ノードからデマンドを受け取ると，1ページ分の出力データを生成するまで②，③の処理を行う。

②アウタ・リレーションのページを格納している入力バッファから1ページを読み出す。その時点でアウタ・リレーションを格納する入力バッファに空きが生じるので，アウタ・リレーションを生成する下位関係演算ノードへ次のページの生成を要求するデマンドを送出しておく。

③インナ・リレーションを格納している入力バッファから1ページを読み出す。その時点でインナ・リレーションを生成する下位関係演算ノードへ次のページの生成を要求するデマンドを送出しておく。②で読み出したアウタ・リレーションのページと，ここで読み出したインナ・リレーションのページとの間で関係演算を実行し，演算結果を出力バッファへ格納する。1回のデマンドに対して1ページ分の出力結果を生成し終えたならば次のデマンドを受け取るまで処理を停止する。さもなければ，もしもインナ・リレーションのページが最終ページであったならば，アウタ・リレーションの次のページに対してインナ・リレーションの全ページの付き合わせを行うために，インナ・リレーションを生成する下位関係演算ノードを初期化した後②へ戻る。もしも，アウタ・リレーション，インナ・リレーションの両ページが最終ページであった場合には関係演算処理を終了する。さもなければ，③の処理を繰り返す。

3．　関係演算処理の実現

3．1　先行・遅延評価の関係演算への適用

従来の問合せの実行を関数計算の観点から見ると，図1に示したような問合せの処理では，結合演算Join-1を実行した後，その中間リレーションに対して結合演算 Join -2を実行するというように，問合せを最内側の関数（関係演算）から1つずつ実行していた。この場合，2．で述べたように，中間リレーションを演算対象とする上位の関係演算は大きな演算対象データを扱うのでメモリ・オーバーフローを引き起こす可能性が高まり，また，関係演算間のパイプラインによる並列性は引き出せない。

これらの点を解決するために，ここでは先行・遅延評価を応用した関係演算を提示する。ストリーム・データおよび先行・遅延評価に関しては文献〔8〕に詳しく述べているので，ここではそれらを関係演算に対応させて簡単に説明する。

（1）ストリーム

データ要素を生成順に並べたもので，データの系列（データ要素の並び）である。本関係演算処理方式では，ストリームの各データ要素はリレーションを構成するタプルに対応し，タプルの系列がストリームとなる。

（2）先行評価

一般に値呼び（call-by-value ）による関数評価においては，引数をすべて評価した（値を得た）後に関数本体の評価が開始される。しかし，引数の評価と併行して関数本体の処理を進めることができれば関数間に内在する並列性をさらに引き出すことが可能となる。この機構をここでは先行評価とよぶ。関係演算処理においては，この機構は関係演算間のパイプラインを作ることに対応する。関係演算を関数で定義すると関数の引数はリレーションに対応することになる（3．2で示す）。リレーションをストリームとしてとらえ，ストリーム全体が求まるまで（引数をすべて評価し終えるまで）待つことなくストリームの一部（リレーション中の一部のタプル群すなわち1ページ）が得られた時点で関係演算の評価を開始することにより，関係演算間でのパイプラインによる並列性を引き出すことが可能となる。

（3）遅延評価

引数の評価にデマンド駆動の概念を取り入れ，データが必要となった時点で必要な個数だけデータ要素

を生成する（すなわち，引数を評価する）と，資源量に合わせて引数の評価を制御でき，ストリームの流れを制御できるようになる。すなわち，先行評価によって引き出される関数間の並列性（パイプライン性）に対して，ストリームの流れをデマンド駆動により制御できるようになる。この機構をここでは遅延評価とよぶ。関係演算処理においては，この機構は関係演算間のパイプラインを流れるタプル群のストリームをデマンド駆動により制御することに対応する。この機構により，計算機資源量に応じてストリームの流れを制御可能となるので，関係演算処理で問題となる中間リレーションのメモリ・オーバーフローの問題を解決できる。関係演算（関数）を評価する場合，その関係演算結果のデータが必要となった時点（すなわち，関係演算結果を受け取る上位関係演算ノードが計算を行う時点）で関係演算に対してデマンドを送出する。デマンドを受け取った関係演算は，１回のデマンドに対応するだけの引数評価を行い，中間リレーションの一部を生成し，再び引数評価を停止する。この機構により，各関係演算はその演算結果を格納する上位関係演算ノードの計算機資源量に応じて独立に計算の進行を制御できるようになる。従って，有限資源の中でメモリ・オーバーフローを起こすことなく，しかもストリームの流れを制御してパイプラインによる並列性を効果的に引出しながら関係演算処理を行える。

3．2　関係演算処理の実現

２．で示した関係演算処理方式は，先行・遅延評価を用いた関数計算により実現できる。ここでは，先行・遅延評価機構をもつ関数型言語 Valid を用いて記述した関係演算処理プログラムを示す。Valid では遅延評価する対象の引数の前に ' delay ' を付加することにより，遅延評価機構を実現する。ストリームを生成する場合の基本となる演算 cons に対しては，特に，第２引数の評価を遅延する strcons (stream cons) を用意している。strcons は次のように定義される。

strcons (x,y) = cons (x,delay y)

例えば，

f (x) = strcons (x, f (x + 1))

とすると，x = 1とした場合には（f (1) は無限の自然数列），

f (1) = (1 . delay f (2))

のように第２引数の評価が遅延される。これに対してデマンドにより評価を再開させるには rest (cdr に対応し，cdr 部の遅延された引数を１回だけ評価する）を用いる。

rest (f (1)) = (2 . delay f (3))

このように，関数型言語に先行・遅延評価機構を取り入れることにより，ストリームの生成，ストリームの流れの制御を非常に簡潔に記述できる。実際，各関係演算プログラムは，Program-1 に示すように大変簡潔である。しかも，このプログラムは，有限資源（有限バッファ）の中で問合せのパイプライン処理を実現する機能をすべて含んでいる。従って，従来のメモリ・オーバーフロー時の制御等のための複雑なプログラムは，本方式ではいっさい排除できる。

4．　性能評価

本関係演算処理の性能評価をするために，本方式と従来の代表的な関係演算処理方式をモデル化し，性能評価式を設定して，各方式を比較検討する。比較対象とする関係演算処理方式は次のとおりである。

（1）ストリーム方式（本関係演算処理方式）
（2）nested-loop 方式　〔4〕，〔5〕
（3）sort-search 方式　〔6〕

なお，（2），（3）のモデルは，文献〔4〕，〔5〕，〔6〕を参考にして，（1）との差異が明確になるようにここで設定したものであり，各文献に示されたアルゴリズムおよびアーキテクチャの環境とは一致しない。

4．1　評価モデル

各方式を評価するために，２項関係演算を処理する

program-1

```
Relational operations descriptions with the functional language Valid

-- x : outer relation structured in a stream form.
--     Each element of the stream is a tuple.
-- y : inner relation structured in a stream form.
--     Each element of the stream is a tuple.
-- n,n1,n2 : list of operand-attribute identifiers
-- strcons(r,s) is defined as cons(r,delay s).
-- rest(x) starts evaluating the cdr-part of delayed cons-cell x.
--  (rest(strcons(r,s)) = s)
-- opitems(tuple,n) extracts the operand attributes specified by n
-- from a tuple.
-- pagesize : size (the number of tuples) of a page stored
--            in the input or output buffer.
-- count : amount of the available output-buffer space.
```

```
selection: function (x:stream,n,a:list,count,pagesize:integer) return(list)
=if null(x) then nil
 elsif opitems(car(x),n) == a   -- comparison operation for retrieval
        then if count == 1      -- check the available space of output-buffer
              then strcons(car(x),selection(rest(x),n,a,pagesize,pagesize))
                                -- selection operation evaluation is delayed.
             else cons(car(x),selection(rest(x),n,a,count-1,pagesize))
    else selection(rest(x),n,a,count,pagesize);

projection: function (x:stream,n:list,count,pagesize:integer) return(list)
=if null(x) then nil
  else if count == 1             -- check the available space of output-buffer
       then strcons(opitems(car(x),n),
                    projection(dupeliminate(rest(x),n,
                               opitems(car(x),n)),n,pagesize,pagesize))
                                --projection operation evaluation is delayed.
       else  cons(mmcar(car(x),n),
                    projection(dupeliminate(rest(x),n,
                               opitems(car(x),n)),n,count-1,pagesize))
                                -- duplicate elimination

dupeliminate: function (x:stream,n,a:list) return(list)
                        -- duplicate elimination for projection operation
=if null(x) then nil
 elsif opitems(car(x),n) == a
        then dupeliminate(cdr(x),n,a)
        else cons(car(x),dupeliminate(cdr(x),n,a));

join: function(x:stream,n1:list,y:stream,n2:list,pagesize:integer) return(list)
=if null(x) then nil
  elsif null(y) then nil
   else append(concatenate(car(x),
                           selection(y,n2,opitems(car(x),n1),pagesize,pagesize)),
               delay join(cdr(x),n1,y,n2));
                               -- join operation evaluation is delayed.

concatenate: function(x1,y1:stream) return(list)
                             -- tuples concatenation for join operation
=if null(y1) then nil
    else cons(append(x1,car(y1)),concatenate(x1,rest(y1)));


union: function (x,y:stream,count,pagesize:integer) return (list)
=if null(y) then x
 elsif null(x) then y
 else clause
         b = car(x);
         if member(b,y)
          then {temp = delete(b,y);   -- elimination of the duplicate tuple
                if count == 1    -- check the available space of output-buffer
                 then strcons(b,union(rest(x),temp,pagesize,pagesize))}
                                    -- union operation evaluation is delayed.
                else  cons(b,union(rest(x),temp,count-1,pagesize))}
           else cons(b,union(rest(x),y,count,pagesize))
       end;
```

場合の各方式の並列処理アルゴリズムを以下のように設定する。

（1）ストリーム方式

基本アルゴリズムは2．2で述べたとおりである。ただし，問合せを構成する各関係演算ノードはデータフロー制御による並列処理を実現するマルチプロセッサ構成になっているものとし，問合せを構成する関係演算ノード間には動的にリンク付けを行うことができるものとする。これにより，関係演算ノード内ではページ単位にデータ駆動型データフロー制御の並列処理を行え，関係演算ノード間ではストリームのパイプライン転送を行える。関係演算ノード内では，上位関係演算ノードからデマンドを受け取ると，アウタ・リレーションおよびインナ・リレーションの1ページが揃った時点でそれらのページを入力バッファから読み出し，アウタ・リレーションの1ページを各プロセッサに分割配置し，インナ・リレーションのページ内のタプルを次々に各プロセッサへブロードキャストする。各プロセッサ内では，ブロードキャストされたインナ・リレーションのタプルと，アウタ・リレーションのタプル間で関係演算を行い，演算結果を出力バッファに格納する。

（2）nested-loop 方式

問合せを構成する関係演算を1つずつマルチプロセッサによって処理し，演算結果の中間リレーションを完全に生成し終えた時点で，次の関係演算を同じマルチプロセッサで処理する。各プロセッサには，アウタ・リレーションをマルチプロセッサの台数で等分割した数のタプル群（分割リレーション）を配置する。インナ・リレーションの各タプルを全プロセッサへブロ

ードキャストし，各プロセッサ内ではアウタ・リレーションのタプル群との間で比較演算（関係演算）を行い，演算結果を出力バッファへ格納する。出力バッファがオーバーフローした場合には，入り切らない出力データを2次記憶へスワップアウトする。1関係演算処理が終了すると，次の関係演算処理の対象リレーションを入力バッファへロードし（出力バッファ内の中間リレーションが次に実行する関係演算の演算対象リレーションとなる場合には，出力バッファを入力バッファと切り換え）関係演算処理を開始する。このアルゴリズムは全タプルを比較する単純なものであるが，関係モデルを指向した多くのデータベースマシンで採用されている。

（3）search-sort 方式

関係演算を処理する場合，sorting を前処理として行うと，一般に計算回数（タプル間の比較演算回数）が少なくなる。2つの演算対象リレーションの一方を演算対象属性上で sort し，他方のリレーションの各タプルと比較演算を行うことによって関係演算処理を行う。 sorting をマルチプロセッサによって行う場合には，タプル数 t のリレーションを sort するのに log t 台のプロセッサと t 個のタプルを格納する主記憶が必要となる〔6〕。リレーションが主記憶に入り切らない場合には，主記憶と2次記憶の間でスワッピング処理が必要となる。また，演算対象リレーションの全タプルが揃わないと sorting 結果は得られないので，関係演算結果の生成は全タプルが揃うまで待たされる。従って，関係演算間のパイプライン効果はない。search-sort 方式を用いた場合の2項関係演算の処理アルゴリズムは以下のようになる。

①アウタ・リレーションを sort moduleで sort し，2進木構造の search module へそれらのタプル群を格納する。アウタ・リレーションの全タプルが sort module へ入り切らない場合には， sort module の容量を1ページとして，1ページずつsortした後，それらを2次記憶へスワップアウトする。この動作をアウタ・リレーションの全タプルに対して繰り返す。

② sort ずみのアウタ・リレーションの1ページを格納している search-module へインナ・リレーションの各タプルを次々と送り込み，関係演算処理を行う。もしもアウタ・リレーションが複数ページから構成される場合には，1ページずつ2次記憶から読み出して search-moduleへ格納し，各ページに対してインナ・リレーションの全タプルを送り込み，関係演算処理を行う。

③関係演算結果の中間リレーションを出力バッファへ格納する。中間リレーションが出力バッファ容量よりも大きい場合には，オーバーフローしたタプル群を2次記憶へスワップアウトする。

4．2 関係演算処理効率

4．1に示した並列処理アルゴリズムの特徴を明らかにするために，関係演算処理（結合演算，選択演算）を行う場合の総計算回数，メモリ使用量，マルチプロセッサでの処理時間，メモリ・オーバーフロー時のスワッピング時間を計算する評価式を図4に示す。

関係演算の中で，問合せの処理効率に最も影響を与える結合演算を処理する各アルゴリズムの特徴は次のとおりである。

（1）総計算回数

search-sort 方式がO（N1＊log N1+ N2＊log N1），nested-loop方式がO（N1＊N2），ストリーム方式がO（N1＊N2＊M ／B ）：（N1,N2 ：インナおよびアウタ・リレーションのタプル数，M ：演算結果リレーションのタプル数，B ：バッファに格納できるタプル数）となり，本ストリーム方式の総計算回数が最も多い。本ストリーム方式は有限バッファ内で関係演算処理を進める。従って，アウタ・リレーションの各ページとインナ・リレーションの全ページの付き合わせを行うために，アウタ・リレーションを格納するバッファ内にアウタ・リレーションの全ページが入り切らない場合には，インナ・リレーションを M／B 回，再計算する必要がある。本ストリーム方式は，これに対して次の2点から対処している。

1）関係演算間のパイプライン処理

本ストリーム方式の特徴の一つは，関係演算ノード間のパイプライン効果である。問合せを処理する場合，個々の関係演算の総計算回数が多くとも，パイプライン上を常にデータが流れていれば問合せ処理は効率よく，無駄な計算は行っていないことになる。例えば，図5に示すような3つの結合演算からなる問合せを考える。結合演算－3（join-3）ノードに着目し結合演算－3のアウタ・リレーション，インナ・リレーションを格納する入力バッファの大きさをそれぞれ bs1,bs2とする。結合演算－3ノードで，ページ間（両入力バッファに入っているインナ・リレーションおよびアウタ・リレーションの各1ページ間）の比較演算を行っている間に，結合演算－2ノードがインナ・リレーションの次のページを生成し入力バッファに格納できれば，パイプラインの遅延はなくなる。すなわち，結合演算－3ノードにおいて，入力バッファにインナ・リレーションの次のページが用意されないことによって起こる関

係演算処理の中断が，パイプラインの遅延である。もしもこの中断がなければ，結合演算－3ノードはnested-loopと同じ回数の比較演算を行うことになるので，パイプライン効果の分だけ効率が良くなる。パイプラインの遅延をなくすには，次の式からページサイズ（granularity）を設定すればよい。

・結合演算－3でのページ間比較処理に要する時間

(bs1 *bs2 /n3) * (jsf1*Tt*Tu/bs1) * (jsf2*Tr*Ts/bs2)

= (jsf1*Tt*Tu) * (jsf2*Tr*Ts) /n3

----------①

・インナ・リレーションのページbs2を生成するのに要する時間

(bs2 / (jsf2*n2)) * (jsf1*Tt*Tu/bs1) * (jsf2*Tr*Ts/bs2)

= (jsf1*Tt*Tu/bs1) *Tr*Ts/n2

----------②

・パイプラインの遅延をなくすための条件

① > ②

bs2 > n3 / (jsf2*n2) ----------③

③のようにページサイズを設定すれば，問合せを構成する各関係演算ノードにおける遅延（関係演算ノードが関係演算処理を中断し，入力バッファに次ページが用意されるのを待っている状態）はなくなり，再計算によるオーバーヘッドはパイプライン処理の中に隠せる。

N1,N2 : number of tuples (inner and outer relation,respectively)
jsf : join selectivity factor,
M : intermediate relation size (= jsf*N1*N2) of JOIN,
ssf : selection selectivity factor (ssf*N1 : intermediate relation size of SELECTION),
psf : projection selectivity factor (psf*N1 : intermediate relation size of PROJECTION),
P : number of processors (P = P1 + P2),
B : buffer size (number of tuples) for each relation (inner, outer and intermediate relations),
Io: swapping time per tuple
Ao: broadcasting time of a tuple to every PE.

[JOIN]

STREAM SCHEME

total computation times	memory requirement
N1*N2*(M/B)	3*B

processing time	swapping time
Ao + B/(jsf*P1) + (N1/B - 1)*N2* max(Ao*B + (B*B)/P1 ,Ao + B/(jsf*P2))	0

NESTED-LOOP SCHEME

total computation times	memory requirement
N1*N2	N1 + N2 + M

processing time	swapping time
Ao*N1 + N1*N2/P + jsf*N1*N2	Io*((N1 - B) + (N1/B - 1)*N2 + (jsf*N1*N2 - B)

SEARCH-SORT SCHEME

total computation times	memory requirement
N1*log N1 + N2*log N1	N1 + N2 + M

processing time	swapping time
N1 + N1 + jsf*N1*N2 + if N1 < B then log N1 + N2 else (N1/B)*(log B + N2)	Io*((N1 - B) + (N1/B - 1)*N2 + (jsf*N1*N2 - B))

[SELECTION]

STREAM-SCHEME

total computation times	memory requirement
N1	2*B

processing time	swapping time
N1/P1	0

NESTED-LOOP SCHEME

total computation times	memory requirement
N1	N1 + ssf*N1

processing time	swapping time
N1/P1	Io*((N1 - B) + (ssf*N1 -B)

SEARCH-SORT

total computation times	memory requirement
N1	N1 + ssf*N1

processing time	swapping time
N1/P1	Io*((N1 - B) + (ssf*N1 -B)

図4 性能評価式

Fig.4 Performance Evaluation Formulas

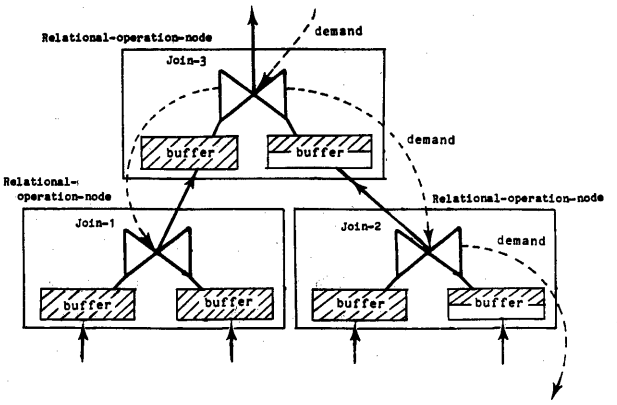

図5　パイプライン処理
Fig.5　Pipeline Processing

2）メモリ・オーバーフローの回避

2．で述べたように，ストリーム方式ではメモリ・オーバーフローは生じない。nested-loop および search-sort 方式では，中間リレーションがバッファ・サイズより大きくなった場合には2次記憶へのスワッピングが行われる。両方式では，スワッピング時間を表す式が示すように，O（N1＊N2／B）回のタプルのスワッピングが必要となる。これは，nested-loop の場合にはマルチプロセッサでの計算回数O（N1＊N2／P）とほぼ同じオーダーであり，search-sort の場合にはO（2＊N1＋N2）よりも多い。しかも1タプルに対するスワッピング処理はマルチプロセッサでの1タプルの処理よりも一般的に長い処理時間を要するので，メモリ・オーバーフローによるオーバーヘッドは非常に大きい。この点が本ストリーム方式との間で処理効率を比較する場合のトレードオフ点となる。すなわち，本ストリーム方式の再計算と，他の方式のスワッピング処理オーバーヘッドが，関係演算処理効率の優劣を決定することになる。

（2）メモリ使用量

本ストリーム方式のメモリ使用量は，演算対象であるインナおよびアウタ・リレーションのページを格納する入力バッファと，関係演算結果の出力リレーションのページを格納する出力バッファ分の容量だけである。すなわち，用意されたメモリ資源の中だけで関係演算処理を進めることができる。一方，nested-loop およびsearch-sort 方式では，インナおよびアウタ・リレーションと関係演算結果の出力リレーションを格納するメモリ領域が必要となる。

（3）マルチプロセッサでの処理時間

プロセッサ台数p台の場合の関係演算処理時間のオーダーは，図4に示すように各方式とも総計算回数のオーダーを反映している。

4．3　問合せ処理効率

各方式による実際の問合せ処理効率を評価するために，2つの問合せ例，問合せ-1（query-1，図6）と問合せ-2（query-2，図6）を処理する場合の処理時間，メモリ使用量，および最初の演算結果が得られるまでの応答時間について，図4に示した評価式を基に性能評価を行った。パラメータとその設定値は図6に示すとおりである。問合せ処理時の主記憶容量（バッファ・サイズ）は固定とする。従って，nested-loop，search-sort方式ではメモリ・オーバーフローを引き起こす場合があり，その場合には2次記憶との間でスワッピング処理を行うものとする。各方式による問合せ-1，問合せ-2の処理時間をそれぞれ図7，図8に示す。横軸は結合演算結合率 jsf（join selectivity factor）を変化させている。jsf は 0～1.0 までの値をとり，中間リレーションの大きさ（タプル数）jsf ＊（T1＊T2）（T1,T2 ：それぞれインナ，アウタ・リレーションのタプル数）を決めるファクタとなる。問合せを構成する各結合演算の jsf は各々同じ値に設定する。

問合せ1,2 とも,4.2の③式を満たすのは，

$$jsf > 0.005$$

の場合である。また，nested-loop，search-sort でメモリ・オーバーフローを引き起こすのは，

$$jsf > 0.000002$$

の場合である。

図7，図8より，ストリーム方式は jsf が大きい

```
Query-1   Join(Join(Relation-1,A#,Relation-2,B#),E#,
                 Join(Relation-3,C#,Relation-4,D#),F#)
Query-2   Join(Relation-1,A#,Join(Relation-2,C#,Relation-3,D#),B#)

  parameter-settings
  number of processors : (p1,p2,p3 = 100);
  number of tuples (cardinarity) : (Relation-1,2,3,4 =  10000);
  join-selectivity-factor : (jsf1,jsf2,jsf3 = 0.000001 ~ 0.1);
  buffer-size : 200 tuples;
```

図6　問合せ
Fig.6　Queries

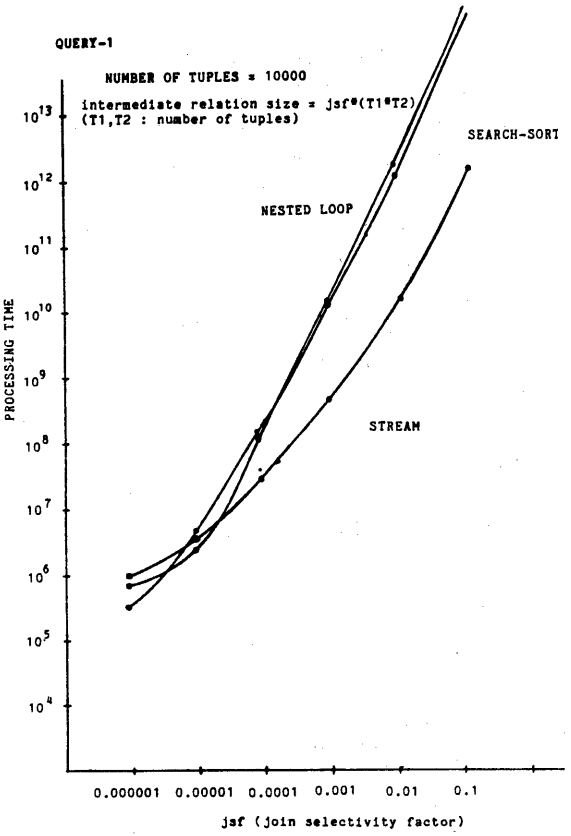

図7　問合せ処理時間 (問合せ-1)

Fig.7　Query Processing Time　(Query-1)

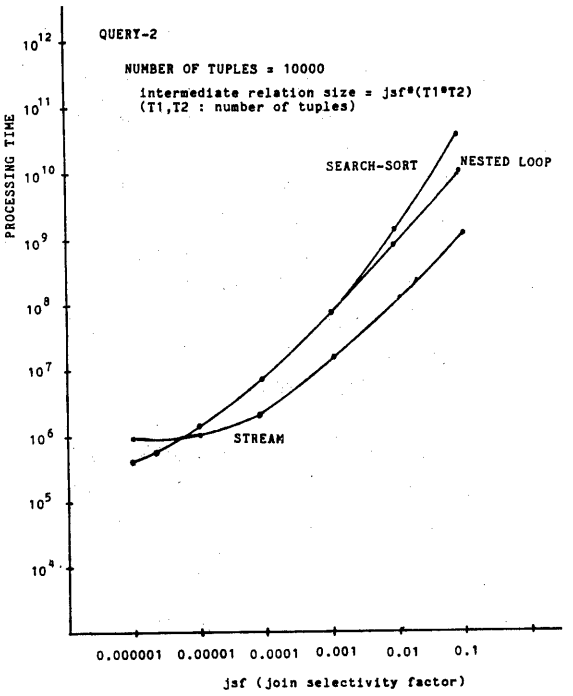

図8　問合せ処理時間 (問合せ-2)

Fig.8　Query Processing Time　(Query-2)

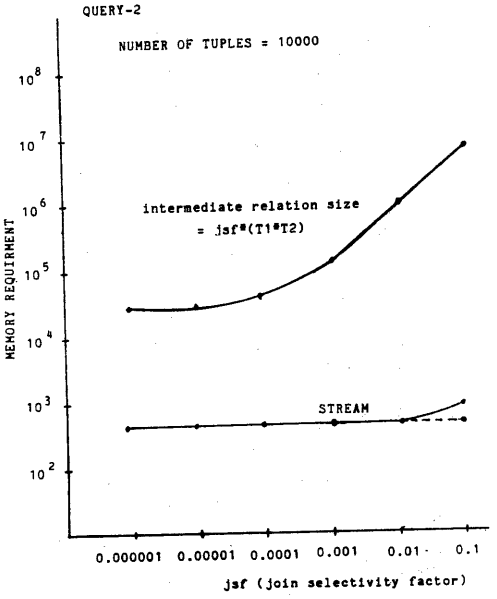

図9　メモリ使用量

Fig.9　Memory Requirement

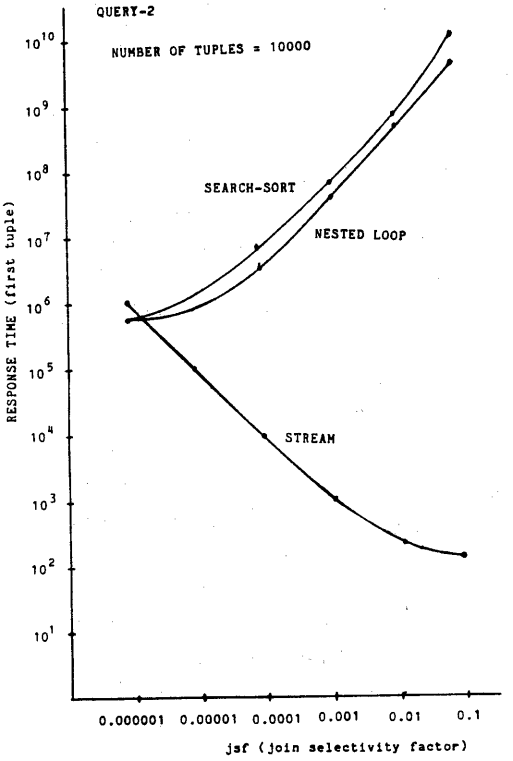

図10　応答時間

Fig.10　Response Time

場合ほど効果が大きい。これは，jsfが大きい場合には，4.2 （1）の1）で述べたように，関係演算ノード間のパイプライン上をデータストリームが途切れなく流れるからである。一方，nested-loop，search-sort方式では jsfが大きくなるほど中間リレーションが大きくなるので，マルチプロセッサでの処理時間の増加に加え，スワッピング処理が増大してしまう。従って，jsf が大きい範囲（目安としては，4.2 の③式を満たす範囲，もしくはそれに近い範囲）においてストリーム方式は有効性を発揮する。一方，jsf が小さい場合には，パイプライン上のデータストリームの流れが悪くなり，各関係演算ノードは関係演算処理を中断している時間が長くなる。すなわち，下位の関係演算ノードにデマンドを送出しても，バッファが詰りにくい状況が起こる。しかし，jsf が小さい場合には，再計算回数は少なくなる（なぜならば，4.2 （1）で示したように再計算の回数は M／B 回であり，jsf が小さいと M が小さくなる）。すなわち jsfが小さい場合には，ストリーム方式は nested-loop方式とほとんど同じ実行形態で問合せを処理することになる。従って，jsf が小さい場合にも，本ストリーム方式はnested-loop 方式と差異のない時間で問合せ処理を行うことができる。

図9に問合せ処理途中の中間結果を表すためのデータ量（タプル数を単位）を示す。これは，jsf が大きい状況でもメモリ使用量が増大しないという本ストリーム方式の特徴を良く表している。

図10に問合せ結果の最初のタプルが得られるまでの応答時間を示す。ストリーム方式では，問合せを構成する関係演算ノード間でのパイプライン処理により，リレーションの一部分が存在すれば処理が進行し，結果の一部が得られるという特徴をもつ。この性質は，問合せのすべての結果を必要としないような応用に対して特に効果を発揮する。

5．むすび

本論文では，関係データベース・システムを実現する際に最大の課題となる関係演算の処理効率の向上を目指した新しい関係演算処理方式を提案した。ここで提案した方式は，有限の計算機資源の中で大量データを扱うデータベース処理を効率よく行うことを目的としたものであり，関係演算処理と関数計算の融合性，ひいてはデータフロー・マシンとの融合性を考慮したものである。本方式は，リレーションに対する関係演算処理をストリームに対する関数の実行に対応させ，関数の引数評価を先行・遅延評価を用いて行うことにより，有限の資源の中でメモリ・オーバーフローを引き起こすことなく関係演算処理を行うことを特徴とする。そして，問合せを構成する複数の関係演算間でパイプラインによる並列処理を実現し，パイプライン上のデータの流れを先行・遅延評価を用いることに効率よく制御する。また，本論文では，従来は非常に複雑であったデータベース処理におけるメモリ管理を 先行・遅延評価機構をもつ関数型言語によって記述することにより関係演算処理アルゴリズムの中に融合し，簡潔に記述できることを示した。

筆者らは今後，本関係演算処理方式をもとに，推論制御とデータベース処理との融合性について検討を進める予定である。

謝辞

本研究に当たり，有益な討論と御協力を頂いた武蔵野通研・情報通信基礎研究部第二研究室の諸氏に感謝致します。

参考文献

〔1〕E.F. Codd : A Relational Model of Data for Large Shared Data Banks, Comm. ACM, Vol. 13, No.6, 1970.

〔2〕E.F. Codd : Relational Completeness of Database Sublanguages, Courant Computer Science Symposia 6, "Database Systems", 1972

〔3〕J.M. Smith and P. Chang : Optimization the Performance of a Relational Algebra Database Interface, Comm. ACM, Vol. 18, No.10, pp.568 - 579, 1975.

〔4〕S.A. Schuster, H.B. Nguyen, E.A. Ozkarahan and K.C. Smith : RAP.2 - An Associative Processor for Database and Its Applications, IEEE Trans. on Compt., Vol.c-28, No.6, pp. 446 - 457, 1979.

〔5〕H. Boral and D.J. DeWitt : Processor Allocation Strategies for Multiprocessor Database Machines, ACM Trans. on Database Systems, Vol.6, No.2, 1981.

〔6〕Y. Tanaka, Y. Nozawa and A. Masuyama : Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer, Proc. IFIP - 80, pp.427 - 432, 1980.

〔7〕Y. Kiyoki, M. Isoda, K. Kojima, K. Tanaka, A. Minematsu and H. Aiso : Perfomance Analysis for Parallel Processing Schemes of Relational Operations and a Relational Database Machine Architecture with Optimal Scheme Selection Mechanism, Proc. 3rd International Conference on Distributed Computing Systems, 1982.

〔8〕M. Amamiya and R. Hasegawa : Dataflow Computing and Eager and Lazy Evaluations, New Generation Computing, Vol.2, No.2, 1984.

〔9〕長谷川，雨宮 : Valid 言語システム：遅延評価機構とその実現，情報処理学会・ソフトウエア基礎論 8 - 4, 1984.

〔10〕雨宮，長谷川，清木 : データフロー・アーキテクチャに於ける先行評価・遅延評価機構とその並列推論制御への応用，信学技法 EC-83-36, 1983