先行・遅延評価機構を用いた関係演算処理方式†

清木 康††* 長谷川 隆三†† 雨宮 真人††

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，関数の引数評価を先行・遅延評価機構を用いて行う関係演算処理方式を提案する．本方式は，巨大なデータを扱うデータベースの処理におけるメモリ資源管理の複雑さの問題を解決し，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処理による並列性を引き出すことを特徴とする．本論文では，本方式の有効性を過去に提案された方式との間で性能評価を行うことにより明らかにする．

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)~7)]．しかし，それらのどの方式も，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メモリ資源の管理，すなわち結合演算結果の中間リレーション (intermediate relation) のメモリ・オーバフロー等の問題を解決していない．

これまで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，一つの関係演算が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，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．したがって，問合せを構成する関係演算間でのパイプライン処理による並列性はなく，また，中間リレーションのメモリ・オーバフローに対処するために主記憶と2次記憶間でスワッピングを行う等のオーバ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効果を引き出すことが考えられる[3),11)]．ページを単位とした場合，問合せを構成する各関係演算ノード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が可能となる．この実現法としてデータ駆動型制御によりページ単位に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．すなわち，ページ単位に関係演算が起動されると，**図1**の斜線部に示すように，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．たとえば，結合演算等の2項関係演算では，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を付き合わせて比較操作を行う必要があるので，インナ・リレーションのページが1ページでも生成され終わっていないとアウタ・リレーションの各ページを消すことはできない．したがって，データ駆動型制御によりページ単位にパイプライン処理を行うと，どこか一つの関係演算ノードの処理が遅れた場合（図1では結合演算-1 (Join-1) ノード），各ノードに大量のデータ（図1斜線部）が中間結果として残されることになる．

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，単にデータ駆動型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，メモリ・オーバフローの問題は解決されない．

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，関数の引数評価を先行・遅延評価 (eager and lazy evaluation)[8)]

† An Execution Scheme for Relational Database Operations with Eager and Lazy Evaluations by YASUSHI KIYOKI, RYUZOU HASEGAWA and MAKOTO AMAMIYA (Musashino Electrical Communication Laboratory, NTT).

†† 日本電信電話公社（現，日本電信電話(株)）武蔵野電気通信研究所

* 現在 筑波大学電子・情報工学系

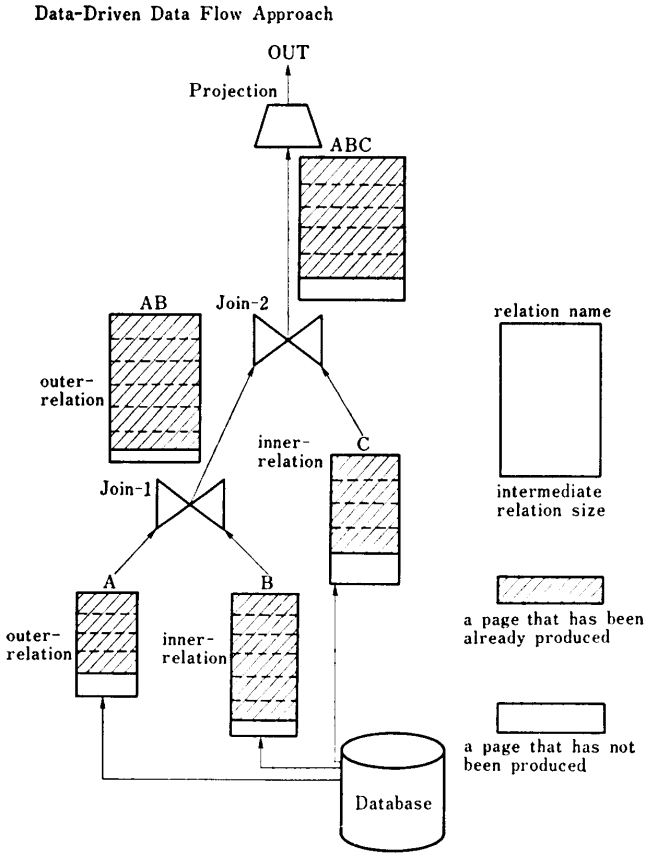

図 1 関係演算処理
Fig. 1 Executions of relational operations.

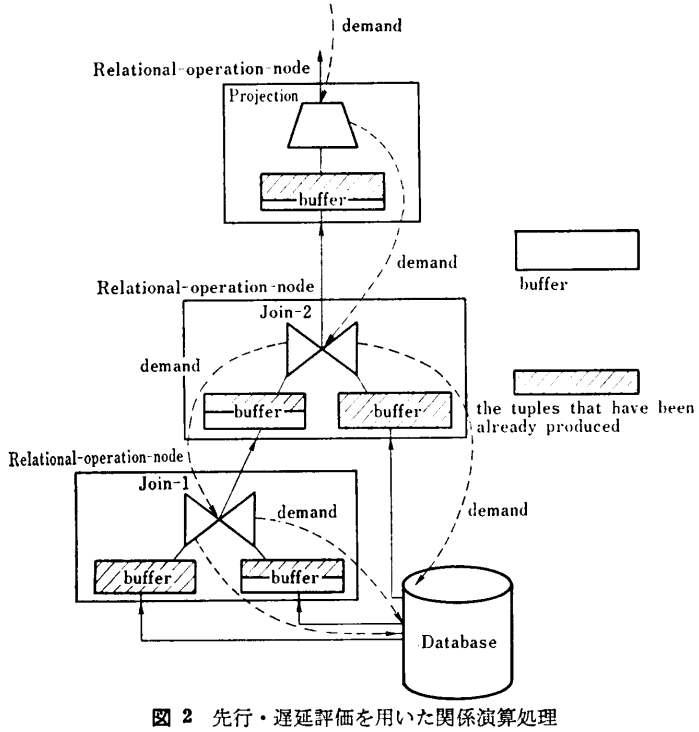

図 2 先行・遅延評価を用いた関係演算処理
Fig. 2 Relational operation execution with eager and lazy evaluation.

機構を用いて行うことにより，有限のメモリ資源量に合わせて問合せ処理を行い，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．本方式は，従来の方式では有限のメモリ資源内で処理できないような問合せの処理を可能にする．

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2**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）へ格納する．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（各バッファにはダブルバッファリング機構があるものとする）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．これにより，下位の関係演算ノードと並行して関係演算処理を行うことができる．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(join)，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．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，とくに効果を発揮する．

各関係演算ノードは，単項関係演算ノード

の場合には一つ，2項関係演算ノードの場合には二つの論理的な入力バッファをもち，その入力バッファは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次のアルゴリズムで動作する．

① 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②，③の処理を行う．

② 演算対象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．その時点で入力バッファには1ページ分の空き領域が生じるので，それを埋めるために下位の関係演算ノードにデマンドを送出しておく．

③ 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．①で受けた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，①へ戻り次のデマンドを待つ．さもなければ，②，③の処理を繰り返す．もしも演算対象リレーションのページが最終ページであったならば，関係演算処理を終了する．

ただし，射影演算についてはタプル間の重複の排除を必要とするので，出力バッファへ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，後続のタプルの重複排除を行うために，消滅させることができない．したがって，射影演算ノードの出力バッファについては中間リレーションを格納する容量が必要となる．

（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次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1ページと，ここで読み出したインナ・リレーションの1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，①に戻り次のデマンドを待つ．さもなければ，③の処理を繰り返す．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．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．

3. 関係演算処理の実現

本章では，ストリームに対する関数計算の概念を用いて2章で示した関係演算処理方式を実現する一方式を提示する．

3.1 先行・遅延評価の関係演算への適用

問合せの従来の実行方法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では，総合演算 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，問合せを構成する各関数（関係演算）を一つずつ実行していた．この場合，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・遅延評価機構を関係演算処理に適用する．データフロー方式による先行・遅延評価機構の実現法に関しては文献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述べる．

（1） ストリーム

データ要素を生成順に並べた系列（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

引数の評価と並行して関数本体の処理を進めること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．関係演算は，リレーションを引数として受け取り結果として新しいリレーション（中間リレーション）を返す関数として定義し，リレーションはタプルを要素とするストリームとしてとらえる．先行評価の導入により，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が可能となる．これにより，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できる．

（3） 遅延評価

引数評価を遅延し，要求に応じて評価を開始することを遅延評価という．これによりメモリ資源量に合わせて，先行評価によって生成されるストリームの流れを制御できるようになる．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．関数として定義された関係演算を評価する場合，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中間リレーション（1ページ分）を生成するために引数評価を行い，生成し終えるとその評価を遅延させる．この機構により，各関係演算ノードはその演算結果を格納する上位関係演算ノードのメモリ資源量（有限なバッファ・サイズ）に応じて各々独立に計算の進行を制御できるようになる．

3.2 関係演算処理の実現方式

2章で示した関係演算処理方式は，先行・遅延評価機構を用いた関数計算により実現できる．先行・遅延評価機構をもつ関数型言語 Valid[9] を用いて記述した関係演算処理プログラム（選択演算，結合演算）を付録に示す．関数間で授受されるストリームは，データ構造を生成する cons オペレータに先行評価の概念を適用することにより実現される．付録に示すプログラムでは，ストリームを生成する関数はストリームの各要素（car 部）が作られるたびに cdr 部の生成を待つことなく，これを待つ関数に引き渡すことができる．一方，ストリーム生成の中断，再開は cons の遅延評価により実現される．遅延評価の指定は，遅延評価の対象となる引数の前に delay を付加することにより行う．ここでは，cons に対して第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 $y)$

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，付録に示すようにストリームの生成，ストリームの流れの制御を簡潔に記述できる．このプログラムでは，cons の先行評価によるパイプライン処理が前提とされ，また有限資源（有限バッファ）内での問合せ処理がカウンタを用いた遅延評価の導入により簡潔に記述されているため，従来のメモリ・オーバフロー時の制御のための複雑な記述は排除されている．

4. 性能評価

2章および3章で述べた関係演算処理方式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

（3） sort-search 方式

なお，(2), (3)のモデルは，文献4)～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場合の各方式の並列処理アルゴリズムを次のように設定する．

（1） ストリーム方式

基本アルゴリズムは2章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する．また，2章で述べたように，関係演算ノード間ではページを単位としてデマンド駆動型データフロー制御の並列処理を行う．関係演算ノード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ページがそろ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た後，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・リレーションの各タプルと自プロセッサに格納されているアウタ・リレーションの各タプル間で関係演算を行い，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）に格納する．

（2） nested-loop 方式

問合せを構成する関係演算を一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，次の関係演算を再びマルチプロセッサで処理する．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）を各プロセッサの入力バッファへ配置する．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に格納する．アウタ・リレーションがマルチプロセッサの入力バッファに入りきっていない場合には，入力バッファの容量を1ページとしてアウタ・リレーションの各ページを2次記憶から読み出し，インナ・リレーションの全ページとの間で比較演算を行い，演算結果を出力バッファへ格納する．出力結果が出力バッファをオーバフローする場合には，入りき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し）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．二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．ここでは，t 個のタプルを sort するために $\log_2 t$ 台のプロセッサと t 個のタプルを格納する主記憶があるものとする[6]．リレーションが主記憶内に入りきらない場合には，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．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次のようになる．

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 に入りきらない場合には，sort module の容量（入力バッファ容量）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 節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）を行う場合の比較演算の総計算回数（total computation times），メモリ使用量（memory requirement），マルチプロセッサでの処理時間（processing time），メモリ・オーバフロー時のスワッピング時間（swapping time）を計算する評価式を**図3**に示す．ここではページ・サイズは各バッファの容量と等しく設定している（ダブルバッファリング機構をサポートするので，実際にはバッファ・サイズはページ・サイズの2倍あるものとする）．また，search-sort 方式におけるプロセッサ台数は，sort-module, search-module ともに $\log_2 B$ 台（B はバッファに格納できるタプル数）とし，ストリーム方式および nested-loop 方式におけるプロセッサ台数は P 台とする．

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場合の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 $O(N_1 * \log_2 N_1 + N_2 * \log_2 N_1)$, nested-loop 方式が $O(N_1 * N_2)$, ストリーム方式が $O(N_1 * N_2 * M/B)$：(N_1, N_2：インナおよびアウタ・リレーションのタプル数，M：中間リレーションのタ

N1,N2 :アウタおよびインナ・リレーション各々のタプル数
jsf :結合演算におけるタプルの結合率(join selectivity factor)
M :中間リレーションのタプル数(=jsf*N1*N2)
B :入力バッファ,出力バッファ各々のサイズ(単位はタプル数)
P,P1,P2:プロセッサ台数(P=P1+P2)(ただし,search-sort方式のプロセッサ台数はlog B台とする.) P1:関係演算ノードに割り当てられたプロセッサ台数, P2:下位関係演算ノードに割り当てられたプロセッサ台数.
Io :1タプルをスワップ・インまたはスワップ・アウトする時間(400)
Co :2タプル間の比較時間(1)
Ao :1タプルを並列プロセッサへブロードキャストする時間(1)
ssf :選択演算における選択率(selection selectivity factor)
(中間リレーションのタプル数=ssf*N1)

[JOIN]

STREAM SCHEME

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

processing time	
Ao+Co*B/jsf*P2)	:下位関係演算ノードが最初のB個のタプル(バッファ・サイズ)を生成するのに要する時間(インナ・リレーションの最初のページを生成する時間).
+(N1/B)*(N2/B)	:ページ間付き合わせの回数.
*max{Ao*B+Co*(B*B/P1),	:ページ間の比較時間.
Ao+Co*B/(jsf*P2))	:インナ・リレーション1ページの生成時間.
swapping time	
0	:スワッピングは起こらない.

NESTED-LOOP SCHEME

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

processing time	
Ao*N2	:インナ・リレーションの各タプルのブロードキャスト時間.
+Co*N1*N2/P	:アウタ・リレーションとインナ・リレーション間のタプルの比較時間.
+Co*jsf*N1*N2/P	:演算結果のリレーションの書込み時間.
swapping time	
Io*((N1-B)	:アウタ・リレーションがバッファに入り切っていない場合にアウタ・リレーションを2次記憶から読み出す時間.
+(N1/B)*(N2-B)	:インナおよびアウタの両リレーションともにバッファに入り切っていない場合, アウタ・リレーションの各ページに対してインナ・リレーションの全ページを毎回2次記憶から読み出す時間.
+(jsf*N1*N2-B))	:出力バッファに出力リレーションが入り切らない場合, 出力リレーションを2次記憶へスワップアウトする時間.

SEARCH-SORT SCHEME

total computation times	memory requirement
N1*log₂N1+N2*log₂N1	N1+N2+M

processing time	
Co*N1	:アウタ・リレーションのsort時間(sort module に入り切るタプル数をページサイズとし, ページ単位に分割してsort).
+Co*N1	:sortされたアウタ・リレーションの各ページをsearch-moduleへ格納する時間.
+(if N1 ≤B then Co*(log₂N1+N2)	:アウタ・リレーションに対してインナ・リレーションの全ページを送り込み結合演算を行う時間.
else Co*(N1/B)*(log₂B+N2))	:アウタ・リレーションの各ページに対してインナ・リレーションの全ページを送り込み結合演算を行う時間.
+Co*jsf*N1*N2	:演算結果のリレーションの書込み時間.
swapping time	
Io*((N1-B)	:アウタ・リレーションがバッファに入り切っていない場合にアウタ・リレーションを2次記憶から読み出す時間.
+(N1/B)*(N2-B)	:インナおよびアウタの両リレーションともにバッファに入り切っていない場合, アウタ・リレーションの各ページに対してインナ・リレーションの全ページを毎回2次記憶から読み出す時間.
+(jsf*N1*N2-B))	:出力バッファに出力リレーションが入り切らない場合, 出力リレーションを2次記憶へスワップアウトする時間.

[SELECTION]

STREAM SCHEME

total computation times	memory requirement
N1	2*B

processing time	swapping time
Co*N1/P1	0

NESTED-LOOP SCHEME

total computation times	memory requirement
N1	N1+ssf*N1

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

SEARCH-SORT SCHEME

total computation times	memory requirement
N1	N1+ssf*N1

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

図 3 性能評価式
Fig. 3 Performance evaluation formulas.

プル数(上位の2項関係演算ノードのアウタ・リレーションのタプル数), B: バッファ・サイズ(バッファに格納できるタプル数))となり, バッファ・サイズが中間リレーションのサイズよりも小さい場合は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ので, 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, 上位の2項関係演算ノードのアウタ・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きらない場合には, インナ・リレーションを生成する関係演算を 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, パイプライン上をつねにデータが流れていれば問合せ処理は効率よく行われる. たとえば, **図4**に示すような三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・リレーションおよびインナ・リレーションを格納する入

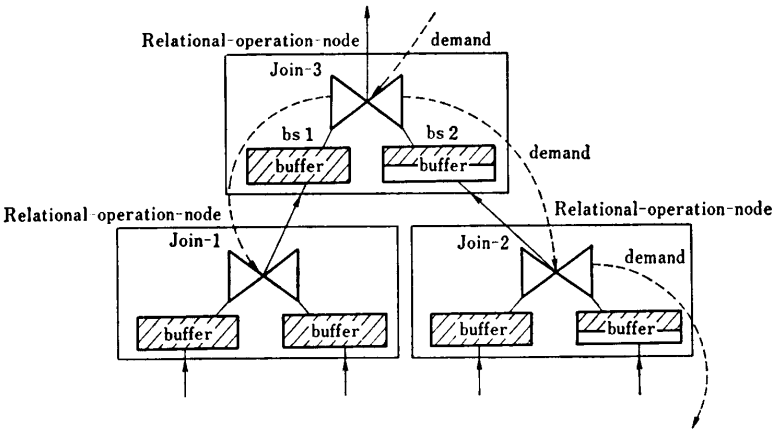

図 4 パイプライン処理
Fig. 4 Pipeline processing.

力バッファの大きさをそれぞれ bs_1, bs_2 (単位はタプル数) 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ノードが次に比較対象となるインナ・リレーション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. 結合演算-3 ノードは, nested-loop と同じ比較回数で演算を行うので, もしもこの中断がなければ, nested-loop よりもパイプライン効果によって効率がよくなる. 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. (P_2, P_3: 結合演算-2, 結合演算-3 ノードおのおののプロセッサ数. N_{11}, N_{12}: 結合演算-1 のアウタ, インナ・リレーションおのおののタプル数, N_{21}, N_{22}: 結合演算-2 のアウタ, インナ・リレーションおのおののタプル数, jsf (join selectivity factor): 0〜1.0 の値をとる変数で, 中間リレーションの大きさ (タプル数) $jsf*(N_1*N_2)$ (N_1, N_2: インナ, アウタ・リレーションおのおののタプル数) を決めるファクタとなる.)

① 結合演算-3 でのページ間比較処理に要する時間

$$\underbrace{(bs_1*bs_2/P_3)}_{\text{ページ間比較}}*\underbrace{(jsf_1*N_{11}*N_{12}/bs_1)}_{\text{アウタ・リレーションのページ数}}*\underbrace{(jsf_2*N_{21}*N_{22}/bs_2)}_{\text{インナ・リレーションのページ数}}$$

② 結合演算-2 がインナ・リレーションのページ bs_2 を生成するのに要する時間

$$\underbrace{(bs_2/(jsf_2*P_2))}_{\substack{\text{インナ・リレーション}\\ \text{の1ページ生成}}}*\underbrace{(jsf_1*N_{11}*N_{12}/bs_1)}_{\text{再計算回数}}*\underbrace{(jsf_2*N_{21}*N_{22}/bs_2)}_{\text{インナ・リレーションのページ数}}$$

③ パイプラインの遅延をなくすための条件

①≧②, すなわち

$bs_2 \geq P_3/(jsf_2*P_2)$

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ヘッドはパイプライン処理の中に隠せる.

2) メモリ・オーバフローの回避

2章で述べたように, ストリーム方式ではメモリ・オーバフローは起こらない. 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 $O(N_1*(N_2-B)/B)$ 回のタプルのスワッピングが必要となる. これは, nested-loop の場合のマルチプロセッサでの計算回数 $O(N_1*N_2/P)$, および, search-sort の場合の $O(2*N_1+N_1*N_2/B)$ (メモリ・オーバフローの場合, すなわち $N_1>B$) とほぼ同じオーダである. 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 ($I_0>>C_0$) ので, メモリ・オーバフローによるオーバ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の合計で

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．

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，二つの問合せ例，問合せ-1 (Query-1, 図5) と問合せ-2 (Query-2, 図5) 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のタプルが得られるまでの応答時間について，図3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5に示すとおりである．問合せ処理時のバッファ・サイズは固定とする．したがって，nested-loop, search-sort 方式ではメモリ・オーバ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6，図7に示す．問合せを構成する各結合演算の jsf はすべて同じ値に設定する．

問合せ1，2とも，4.2節の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・オーバフローを引き起こすのは，

$jsf > 0.000002$

の場合である．

図6，図7より，ストリーム方式は jsf が大きい場合ほど他方式に比べて効率がよい．それは，jsf が大きい場合には，4.2節(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

```
Relation-1 (A#, E#, A1#)
Relation-2 (B#, C#, B1#)
Relation-3 (D#, C#, C1#)
Relation-4 (B#, F#, D1#)

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

parameter-settings
 buffer-size : (B = 200 tuples)
 the number of processors : (P1,P2,P3 = 100 : STREAM SCHEME,
                             P = 300        : NESTED-LOOP,
                             2*log(B)       : SEARCH-SORT)
 the number of tuples (cardinarity) : (Relation-1,2,3,4 = 10000)
 join selectivity factor : (jsf1,jsf2,jsf3 = 0.000001 ~ 0.1)
```

図5 問合せ
Fig. 5 Queries.

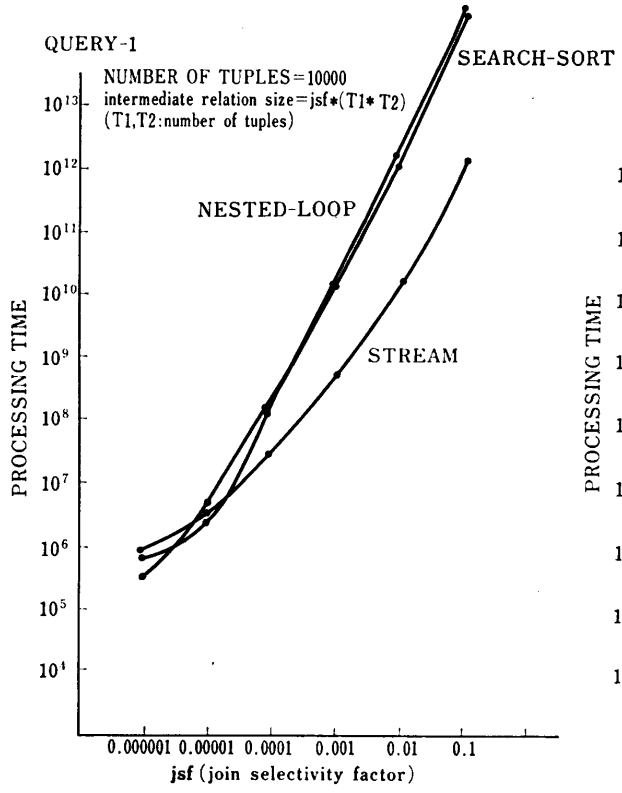

図6 問合せ処理時間 (問合せ-1)
Fig. 6 Query processing time (query-1).

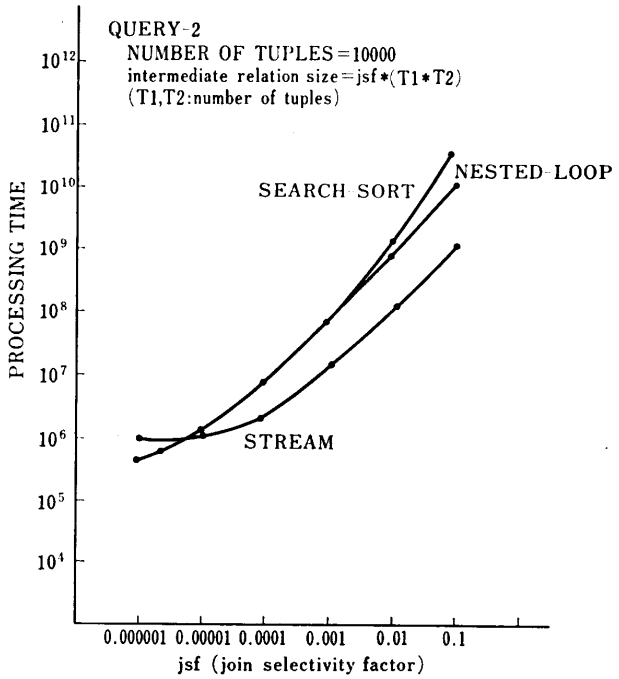

図7 問合せ処理時間 (問合せ-2)
Fig. 7 Query processing time (query-2).

タストリームが間断なく流れるからである．一方，nested-loop および 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．したがって，jsf が大きい

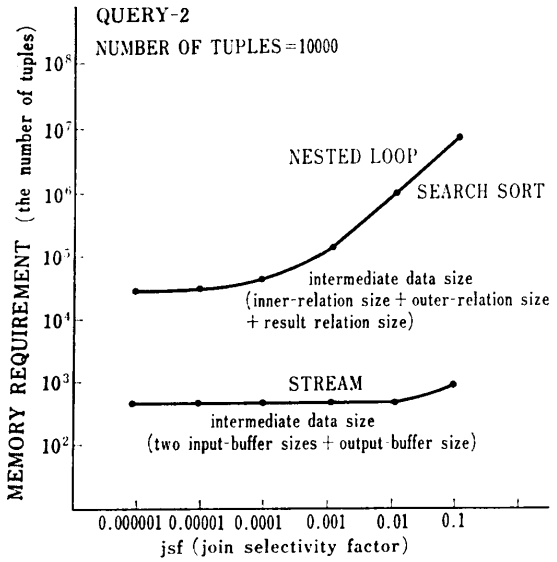

図 8 メモリ使用量
Fig. 8 Memory requirement.

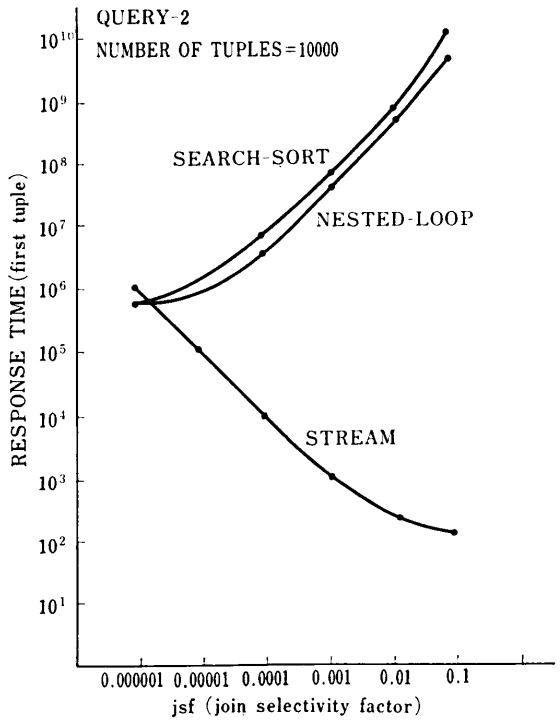

図 9 応答時間
Fig. 9 Response time.

範囲（目安としては，4.2 節の③式を満たす範囲，あるいはそれに近い範囲）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ま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 節(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）のでストリーム方式は nested-loop 方式とほぼ同じ実行形態で問合せを処理することになる．したがって，jsf が小さい場合には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．また，search-sort 方式については，メモリ・オーバフローが生じない (search-module, sort-module 内に演算対象リレーションが入りきる) 限り，本ストリーム方式よりも少ないプロセッサ台数 (P: $(2*\log_2 B) \fallingdotseq (20:1)$) で同等の効率を実現できる．

図 8 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よく表している．

図 9 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とくに効果を発揮する．

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，有限のメモリ資源の中で，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より制御する．性能評価では2段の2項関係演算からなる問合せを対象としたが，それが多段となった場合にも個々の連続した2ノードが4.2節の③に示した条件式を満たせば，パイプラインは最大の効率で流れる．

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[10]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，有益な討論とご協力をいただいた武蔵野通研・情報通信基礎研究部第二研究室の諸氏に感謝いたします．

参考文献

1) Codd, E. F.: A Relational Model of Data for Large Shared Data Banks, *Comm. ACM*, Vol. 13, No. 6, pp. 377-397 (1970).
2) Codd, E. F.: Relational Completeness of Database Sublanguages, Courant Computer Science Symposia 6, "Database Systems", pp. 65-98 (1972).
3) Smith, J. M. and Chang, P.: Optimizing the Performance of a Relational Algebra Database Interface, *Comm. ACM*, Vol. 18, No. 10, pp. 568-579 (1975).
4) Schuster, S. A., Nguyen, H. B., Ozkarahan, E. A. and Smith, K. C.: RAP. 2—An Associative Processor for Database and Its Applications, *IEEE Trans. Comput.*, Vol. C-28, No. 6, pp. 446-457 (1979).
5) Boral, H. and DeWitt, D. J.: Processor Allocation Strategies for Multiprocessor Database Machines, *ACM Trans. Database Syst.*, Vol. 6, No. 2, pp. 227-256 (1981).
6) Tanaka, Y., Nozawa, Y. and Masuyama, A.: Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer, Proc. IFIP-80, pp. 427-432 (1980).
7) Kiyoki, Y., Isoda, M., Kojima, K., Tanaka, K., Minematsu, A. and Aiso, H.: Performance 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, pp. 196-203 (1982).
8) Amamiya, M. and Hasegawa, R.: Dataflow Computing and Eager and Lazy Evaluations, *New Generation Computing*, Vol. 2, No. 2, pp. 105-129 (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・ソフトウェア基礎論 8-4 (1984).
10) 雨宮，長谷川，清木：データフロー・アーキテクチャにおける先行評価・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).
11) 馬場，Yao, S. B., Hevner, A. R.: データフロー解析に基づく多重プロセッサ・データベースマシンの設計，情報処理学会・第29回全国大会，pp. 757-758 (1984).

付録

```
Program-1
     Relational Operations Descriptions  in  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
        --   pagesize  : size (the number of tuples) of a page stored in the
        --               input-buffer or output-buffer. (pagesize = buffersize)
        --   strcons(r,s) : delayed cons operator defined as cons(r,delay s)
        --   rest(x)   : evaluating the cdr-part of delayed cons-cell x
        --               (As the result, a head  element of the cdr-part of x is
        --                produced and then the evaluation is delayed again.)
        --   opitems(tuple,n) : extracting the operand-attribute values
        --                      specified by the operand-attribute idenifiers n
        --
```

```
selection : function(x:stream.n,a:list,count,pagesize:integer) return(stream)
= if null(x) then nil              -- check the end of the stream
  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))
                          -- buffer space is exhausted.
                          -- the evaluation of selection operation is delayed.
             else cons(car(x),selection(rest(x),n,a,count-1,pagesize))
    else selection(rest(x),n,a,count,pagesize);


join : function(x:stream.n1:list,y:stream.n2:list,pagesize:integer)
       return(stream)
= if null(y) then nil          --  check the existence of the inner-relation
  elsif null(x) then nil       --  check the end of the outer-relation stream
    else append(concatenate(car(x),
                        selection(y.n2.opitems(car(x),n1)
                                  ,pagesize,pagesize),pagesize,pagesize),
               delay join(rest(x),n1.y.n2.pagesize);
                          --  the evaluation of join operation is delayed.

concatenate : function(x1:list,y1:stream.count.pagesize:integer)
              return(stream) --  tuples concatenation for join operation
= if null(y1) then nil         --  check the existence of the tuples which
                               --  will be concatenated.
   elsif count == 1            --  check the available space of output-buffer
       then strcons(append(x1,car(y1)),
                    concatenate(x1,rest(y1),pagesize,pagesize))
                               --  buffer space is exhausted.
                               --  the evaluation of concatenate is delayed.
     else cons(append(x1.car(y1)),        --  concatenation of two tuples
               concatenate(x1.rest(y1),count-1,pagesize));
```

(昭和59年12月14日受付)
(昭和60年 1 月17日採録)