ストリーム指向データベース処理における最適メモリ資源割当て方法

陳 幸[†] 清木 康[††]

データベースの多様な応用分野に適応可能な並列処理方式として，我々は，ストリーム指向型並列データベース処理方式を提案してきた．この方式を並列処理システム上で実現する場合の重要な課題は，計算機資源の割当てである．本論文では，共有メモリ型並列マシンにおいて，問合せをストリーム指向型並列処理方式により処理する場合に，問合せ処理効率を向上するためのメモリ資源割当て方式を提案する．また，提案するメモリ資源割当て方式を用いて問合せ処理を行った実験の結果を示し，本方式の有効性を明らかにする．

A Method for Optimal Memory Allocation for Stream-oriented Database Processing

XING CHEN[†] and YASUSHI KIYOKI[††]

In order to process various database applications in parallel, we have proposed a stream-oriented parallel database processing scheme. An important issue for implementing this scheme in a parallel processing environment is the optimal computer resources allocation. In this paper, an optimal memory resources allocation method is proposed for queries processed by the stream-oriented parallel database processing scheme in a shared memory parallel machine. Several experimental results are shown to clarify the efficiency of the memory allocation method.

1. まえがき

近年，データベースシステムが扱うデータ量は増大し，また，対象とする応用分野は多様化している．多様なメディアによって表現された様々な情報がネットワーク上に広く存在しており，動画，音声などのコンティニュアスメディア・データをデータベース化し，効率的に検索，加工，および，合成するシステムの実現が重要な課題となっている[1)~4)]．

コンティニュアスメディア・データをデータベース化するために，多くのデータ・フォーマットおよび記録，再生，処理ツールが提案されてきた[2),5)~14)]．それらのデータ・フォーマットやツールは，利用者の利用環境や目的などの状況に応じて選択され，利用されている．我々は，それらのデータ・フォーマットに対応し，また，新しいコンティニュアスメディア，および，新しいフォーマットに適応できる方式として，ストリーム指向型並列処理方式を提案し，その方式によるデータベースシステムの実現を行っている[15),16)]．

ストリーム指向型並列処理方式は，関数型計算[17)~20)]の概念をデータベース処理に適用することにより，データベース処理の対象となるデータをストリームとして表現し，それらを対象としたデータベース処理をストリームの関数として実現する．この方式の特徴は，多様な応用分野に適した任意のデータベース演算およびデータ構造を柔軟に記述できること，ならびに，データベースへの問合せを並列に処理できることである．

ストリーム指向型並列処理方式によるコンティニュアスメディア・データ処理の実現方式の特徴を次に列挙する．

(1) データベースから，任意のフォーマットのコンティニュアスメディア・データを収集することができる．すなわち，データベースへの問合せ処理を柔軟に記述でき，かつ，任意のフォーマットから復号化できる．

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† 慶應義塾大学環境情報学部
Department of Environmental Information, Keio University

(2) 多様なコンティニュアスメディア・データを対象することができる．今後，新しいコンティニュアスメディアの出現，および，新しいフォーマットの設定が想定される．また，新しいコンティニュアスメディアでは，そのメディアに特有の加工操作の存在が想定される．本システムでは，設計者が定義したコンティニュアスメディア・データに対する操作を関数として取り込むことにより，新しいコンティニュアスメディア，および，多様なフォーマットに柔軟に対応することが可能となる．

(3) 利用者は，システムが生成するデータのフォーマットを自由に選択することができる．これは，利用者が用いるフォーマットへの変換操作を関数として柔軟にシステムに取り込むことができることによる．

(4) システムの実現に際して，ハードウェア環境を特定しない．すなわち，利用者が求める性能に応じて，自由にハードウェアを選択できるように，個々のハードウェアに依存しない，上位レベルに基本操作群を設定している．これにより，ハードウェア技術の進歩に柔軟に対応することが可能となる．

複数のコンティニュアスメディア・データから新しいマルチメディア・データを生成する処理に，ストリーム指向型並列処理を適用するために，コンティニュアスメディア・データの流れ，すなわち，コンティニュアスメディア・データ自身，および，生成処理途中の中間結果をストリームとして扱う．システムの設計者は，システムで取り扱う各種のコンティニュアスメディア・データの操作を関数群によって定義する．利用者は，それらの関数を組み合わせることにより，マルチメディア・データ生成のための問合せを記述する．

そのようなストリーム指向型問合せ処理において，メモリ資源は，ソースデータや中間結果の格納，および，ディスクへのアクセス回数を軽減するために用いられる．メモリ資源の割当ては，問合せの処理効率に重大な影響を与える．

最適なメモリ資源割当てのための方法が数多く提案されている[21)~24)]．それらの方法は，関係データベースの問合せを処理対象としている．関係データベースシステムにおいて，使用されるデータベース演算は，固定的に定められた関係データベース演算である．したがって，従来のメモリ資源割当て方式では，各関係データベース演算の性質に依存したメモリ資源割当てを求めることにより，問合せの処理効率を向上させる方法を用いていた．しかし，ストリーム指向型並列処理方式は，データ構造やデータベース演算を柔軟に適応するため，利用されるデータベース演算が固定されていないので，任意のデータベース演算を対象としたメモリ資源割当ての方法が必要となる．

ハードウェア技術の進歩にともない，共有メモリ型並列処理システムを用いた並列処理環境が容易に利用できるようになっている．これらの環境を利用して，データベース処理を高速に行うための研究も活発である．本論文では，共有メモリ型並列処理マシン上で，ストリーム指向型並列データベース処理方式を対象とするメモリ資源割当て方式を提案する．分散環境におけるメモリ資源割当て問題については文献 25)，26) に詳しい．

文献 27) において，ストリーム指向型並列処理方式を対象としたメモリ資源割当て方式が提案されている．この方式は，ストリーム指向型並列処理方式によって，処理中に生成している中間結果をメモリやディスクに格納しない場合に，その処理時間を最短にするメモリ資源割当てを求めるための方式である．

本論文では，ストリーム指向型並列処理中に生成される中間結果をディスクに格納する場合を対象として，問合せの処理効率を向上するためのメモリ資源割当てを求める方式を提案する．まず，問合せの処理中の中間結果ストリームをディスクに格納するために，ストリーム指向型並列データベース処理方式を拡張する．次に，拡張した処理方式により，中間結果ストリームをディスクに格納する場合において，最適なメモリ資源割当て方式を提案する．また，提案するメモリ資源割当て方式を用いて問合せ処理を行った実験の結果を示し，本方式の有効性を明らかにする．

2. ストリーム指向型データベース処理におけるメモリ資源割当て

本章では，まず，ストリーム指向型並列データベース演算処理方式の概要を述べる．その処理方式については文献 28) に詳しく述べている．また，ストリーム指向型並列データベース演算処理方式におけるメモリ資源割当て問題を説明する．

2.1 ストリーム指向型並列処理方式

ストリーム指向型並列データベース演算処理方式では，データベース演算は関数として実行される．関数は，主記憶に常住し，軽量プロセスとして実行される．関数の計算は，要求駆動型評価[18)]によって実現される．

関数型計算の枠組みの中で次のような 2 種類の並列性を抽出できる．

(1) ストリーム型並列性

データベースの問合せ処理においては，ストリームを生成する演算とそれを消費する演算が並列に実行されることに対応する．

以下では，中間結果のストリームを生成する関数を生産者関数，それらを消費する関数を消費者関数とする．ストリーム型並列性を実現するために，バッファが生産者関数と消費者関数間に設定される．

ストリーム指向型並列データベース処理を逐次的に実現する場合には，消費者関数は，入力バッファ内の1グレイン分のストリーム要素群の処理を終えると，生産者関数へデマンドを発行する．生産者関数は，デマンドを受け取ると，出力バッファ内に演算結果の1グレイン分のストリーム要素群の生成を完了するまで関数演算を実行し，生成を完了する時点で実行を停止する．1デマンドに対してストリーム全体を生成せず，あらかじめ定められた1グレイン分のストリーム要素群を生成して停止するので，大量データを扱う演算処理を限られたメモリ資源の中で行うことが可能となる．

並列処理を行う場合には，生産者関数と消費者関数を異なるプロセッサに配置し，それらの関数間のバッファには，ダブルバッファリング機構を実現する．各バッファには2領域を用意する．一方の領域の大きさは，1グレイン分のストリーム要素群を格納できるように設定される．生産者関数は，1デマンドに対し，1グレイン分のストリーム要素群を生成する．消費者関数は，一方の領域に格納されているストリーム要素群に対して関数演算を実行する前に，他方の領域に次のグレインの格納を要求するために，生産者関数へデマンドを先行的に発行する．生産者関数は，デマンドを受け取ると次のグレインを生成するために関数演算の実行を開始する．消費者関数は，一方の領域内のストリーム群に対する処理を完了すると，その領域への次グレインの格納を要求するために，生産者側関数に再びデマンドを発行する．このようにして，ストリーム型並列性を抽出する．

(2) 関数引数の並列評価

データベースの問合せ処理においては，互いに独立な演算が並列に実行されることに対応する．この並列性は，複数の引数データの消費を行う消費者関数が，それらの引数データを生成する各生産者関数に対して，デマンドを同時に発行することによって抽出される．

CPUの割当ては，以下の方法により行う．関数は，主記憶上に常駐され，問合せが完了する前には，主記憶からスワップ・アウトされないことを前提とする．まず，関数キューとCPUキューを作る．関数キューは，CPUの処理を待つ関数の列であり，CPUキューは，アイドル状態にあるCPUの列である．

CPU数が関数数より少ない場合，関数は擬似的に並列に実行される．その場合には，出力バッファ内に演算結果の1グレイン分のストリーム要素群の生成を完了するまで関数演算を実行し，生成を完了した時点で実行を停止する．実行を停止した関数は，関数キューの末尾に加えられる．

CPUは，関数に対する演算が終わった後，アイドル状態になり，CPUキューの末尾に加えられる．これらのキューの生成後，CPUキューの先頭にあるCPUを関数キューの先頭の関数に割り当てる．

本論文では，メモリ割当てがストリーム指向型処理効率に与える影響について述べ，最適なメモリ資源割当て方式を提案する．この割当て方式は，問合せを構成する全関数を対象とし，各関数が主記憶上に常駐され擬似的に並列に実行される状況を前提とする．

2.2 ストリーム指向型処理方式におけるメモリ割当て問題

本論文では，**図1**に示すような木構造型問合せ処理を汎用の共有メモリ並列処理マシンで実現する場合において，最適なメモリ資源割当てを求める方式を提案する．

大量データを扱う演算処理を限られたメモリ資源の中で行うことを可能にするために，本論文では，関数型計算におけるcall-by-name[17]による引数評価方式により，要求駆動型評価を実現する．call-by-nameでは，関数本体の実行において，引数の参照に出会うた

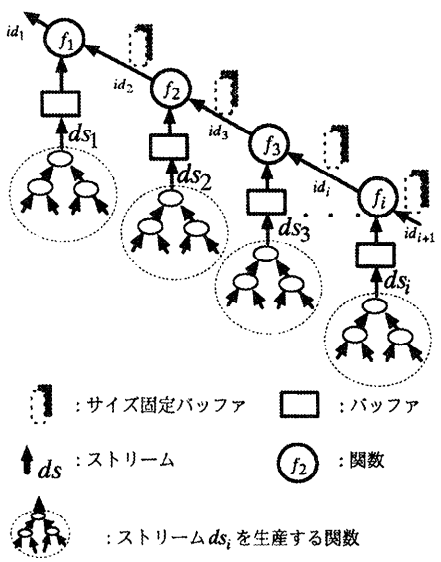


図1 ストリーム指向型問合せ処理

Fig. 1 The stream-oriented query processing.

びに，その実引数が評価される．このため，実引数値を参照した後に，その値を消去することができるが，同じ引数が再参照されるたびに，その引数の生成を行う必要がある．木構造型問合せ処理の場合，関数本体の実行において，引数は，下記の方式で再参照される．

ここでは，例として，ストリーム A とストリーム B を消費する関数をデータベース操作の関数として対象とする．関数本体の実行中に，バッファ *Buffer_A* に格納されたストリーム A の1グレイン分の要素群の処理を終えると，関数は，ストリーム A の次のグレインの生成を要求するために，デマンドを発行する．また，ストリーム A の全要素の処理を終えると，バッファ *Buffer_B* に格納されるストリーム B の要素群の処理を終える場合に，関数は，ストリーム B の次のグレインの生成を要求するために，デマンドを発行する．同時に，関数は，ストリーム A の次のグレインの生成のために，ストリーム A の生産者関数にデマンドを発行する．そのデマンドの発行により，ストリーム A の最初のグレインが再度バッファ *Buffer_A* に格納される．ストリーム B の各グレインの処理とストリーム A の全グレイン間の処理を終えると，関数の実行が終了する．

その処理により，問合せの処理中にストリーム A を2回以上生成する（以下，ストリームの再生成という）操作が必要となる．ストリーム A の再生成回数は次のとおりである．ストリーム B のサイズを *Stream_Size* として，バッファ *Buffer_B* のサイズを *Buffer_Size* とすると，その再生成回数は，$\lceil Stream\_Size / Buffer\_Size \rceil$ である．

ストリームの再生成回数が増えると，ディスクへのアクセス回数も増える．一般的に，コンティニュアス・データに対するディスクへのアクセス操作は，関数の処理効率に重大な影響を与える．そこで，本方式は，ストリームの再生成の回数とディスクへのアクセス回数をパラメータとして最適メモリ資源割当てを求めることを目的とする．

メモリ資源の制限がある場合，すべてのバッファに必要なサイズを割り当てることができない状況が発生する．本方式は，その状況において，最適なメモリ資源割当てを求めるものである．

図2 は，図1の問合せ構造から，ストリームの再生成と関連がある関数を抽出する構造である．本方式では，図2の構造により，ストリームの再生成と関連するバッファ（以下バッファと略）の最適なサイズを求める．

図2において，関数 f_1 が生成するストリーム id_1 は問合せ処理の結果である．関数 f_{i+1} が生成する中間結果ストリーム id_{i+1} は，関数 f_i によって消費される（$i = 1, 2, \ldots, N-1$）．関数 f_N 以外は，すべての関数が2ストリーム（id_{i+1} と ds_i）を消費する．関数 f_N は1ストリーム ds_N だけを消費する．ストリーム ds_i は1回だけ生成され，中間結果ストリーム id_{i+1} の生成回数はバッファサイズ bs_i に依存する．

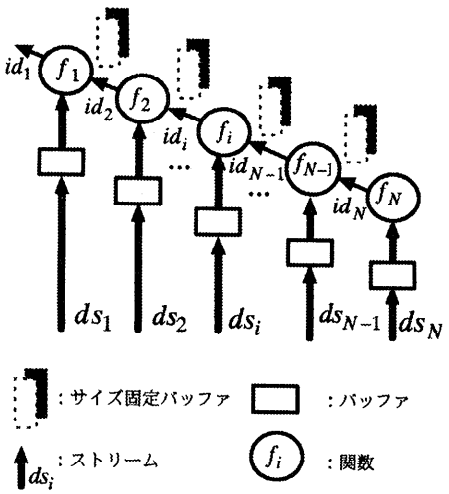


図2 最適なメモリ資源割当てを求める構造
Fig. 2 The calculating structure for obtaining the optimal memory allocation.

以下では．次の記述を用いる，

- M：利用可能なメモリ資源（単位：ページ），
- f_i：関数，
- N：関数の数，
- ds_i：関数 f_i の引数ストリームとそのサイズ（単位：ページ），
- bs_i：ストリーム ds_i を格納するバッファとそのサイズ（単位：ページ），
- id_i：関数 f_i の出力ストリームとそのサイズ（単位：ページ），
- $Ac(ds_i)$：ストリーム ds_i を生成するために必要なディスクへのアクセス回数，
- $Ac(id_i)$：ストリーム id_i を生成するために必要なディスクへのアクセス回数，
- $Sv(p)$：p ページ分のストリーム要素群をディスクに保存するため，あるいは，ディスクから呼び出すために必要なディスクアクセス回数．

上述の記述と図2の構造を利用すると，問合せ処理中に必要なディスクアクセス回数 T_Q $(= Ac(id_1))$ は，次式 (1) に示される．

$$
\begin{aligned}
T_Q &= Ac(id_1)\,,\\
Ac(id_1) &= Ac(ds_1) + \left\lceil \frac{ds_1}{bs_1} \right\rceil * Ac(id_2)\,,\\
&\cdots\\
Ac(id_i) &= Ac(ds_i) + \left\lceil \frac{ds_i}{bs_i} \right\rceil * Ac(id_{i+1})\,,\\
&\cdots\\
Ac(id_N) &= Ac(ds_N)\,.
\end{aligned}
\tag{1}
$$

最適メモリ資源割当て問題は，次のように定式化できる．

目的関数

$$\min\left(T_Q\left(bs_1, bs_2, \ldots, bs_N\right)\right)\,,$$

制約条件

$$M = \sum_{i=1}^{N} bs_i\,.$$

3. メモリ資源割当て方式

本章では，問合せのディスクアクセス回数を最小にするためのメモリ資源割当て方式を提案する．ストリーム指向型並列処理におけるメモリ資源割当て方式として，次の2種類が考えられる．

(1) 静的割当て方式

問合せが与えられたとき，あらかじめ獲得している統計情報をもとに，静的にメモリ資源を割り当てる方式．

(2) 動的割当て方式

問合せの実行時に，各関数の実行状態をモニタリングすることにより，統計情報を獲得し，問合せ処理の実行途中において，メモリ資源割当て計算を行った後，メモリ資源の動的割当てを行う方式．

動的メモリ資源割当て方式を利用する場合，割当てアルゴリズムの計算複雑さは，問合せの処理効率に影響を与える．文献27)において，動的割当て計算のオーバヘッドが小さい方法が提案されている．

従来の提案は，中間結果ストリームがディスクに保存されない場合において，最適なメモリ資源割当てを求める場合に対応している．本方式は，中間結果ストリームをディスクに保存する場合において，ディスクアクセス回数への影響を考慮した最適なメモリ資源割当てを求める方式である．

この方式では，中間結果ストリーム id_i がディスクに保存されている場合，そのストリームの再生成は必要ない．ストリーム id_i が再び消費されると，そのストリームがディスクから呼び出される．

式(1)により，ストリーム id_i を生成するために必要なディスクへのアクセス回数は，次式で表される．

$$Ac(id_i) = Ac(ds_i) + \left\lceil \frac{ds_i}{bs_i} \right\rceil * Ac(id_{i+1})$$

中間結果ストリームが k 回目に生成される場合に必要なアクセス回数は，$k * Ac(id_i)$ となる．

ストリーム id_i が1回だけ生成され，ディスク領域に保存される．その後，そのストリームが $k-1$ 回目にディスク領域から呼び出される場合に，必要なディスクアクセス回数は，次式で表される．

$$D_{id_i} = Ac(id_i) + Sv(id_i) + (k-1) * Sv(id_i)$$

D_{id_i} の値は，中間結果ストリームをディスクに保存する場合におけるディスクアクセス回数である．

D_{id_i} の値が，$k * Ac(id_i)$ の値より小さい場合には，中間結果ストリームをディスクに保存することにより，問合せの処理効率が向上する．

3.1 特殊関数によるストリーム要素群のディスクへの保存

中間結果ストリームをディスクに保存するために，本方式では，ストリーム指向型並列データベース処理方式を拡張し，生産者関数と消費者関数の間に特殊関数 f' を設定する．ストリームの再生成の回数を最小にするために，特殊関数 f' を関数 f_1 と関数 f_2 の間に設定する．

中間結果ストリーム要素全体を格納できるディスク領域がある場合に，ストリーム id_2 が第1回目に消費されるときに，特殊関数 f' は，関数 f_1 からデマンドを受け取り，関数 f_2 へ渡す．また，関数 f_2 の出力ストリーム id_2 の要素群を消費者関数 f_1 に渡すのにともない，そのストリームをディスク領域に保存する．ストリーム id_2 が再消費されるとき，そのストリームの全要素をディスク領域から呼び出すことができるので，ストリーム id_2 の再生成は必要なくなる．関連するストリーム $id_3, id_4, \ldots, id_N$ の再生成も必要なくなる．また，関数 $f_2, f_3, \ldots, f_N$ の計算処理を実施するために必要なメモリ資源を解放できる．解放されたメモリ資源を対象として，関数 f_1 のバッファ bs_1 に再割当てを行う．式(2)は，ストリーム id_2 の第1回目の生成のとき，問合せのディスクアクセス回数（$Ac(id_{1,1})$）を表す．式(3)は，ストリーム id_2 の第2回目の生成開始から問合せ処理の終了までのディスクアクセス回数（$Ac(id_{2,1})$）を表す．

$$
\begin{aligned}
Ac(id_{1,1}) &= Ac(bs_1) + Sv(id_2) + Ac(id_2)\,,\\
Ac(id_2) &= Ac(ds_2) + \left\lceil \frac{ds_2}{bs_2} \right\rceil * Ac(id_3)\,,\\
&\cdots\\
Ac(id_i) &= Ac(ds_i) + \left\lceil \frac{ds_i}{bs_i} \right\rceil * Ac(id_{i+1})\,,\\
&\cdots
\end{aligned}
\tag{2}
$$

$$Ac(id_N) = Ac(ds_N)\,.$$

式 (3) において，ストリーム ds_1 の bs_1 ページ分の要素群は，ストリーム id_2 が生成されたときに消費されているので，そのページ分の要素群は，式の計算から取り除く．

$$Ac(id_{2,1}) = Ac(ds_1 - bs_1) + \left\lceil \frac{ds_1 - bs_1}{M} \right\rceil * Sv(id_2)\,. \tag{3}$$

中間結果ストリーム要素全体を格納するディスク領域がない場合に，特殊関数 f' は，受け取ったデマンドに対応するストリーム要素群がディスク領域に保存されているかをチェックする．対応するストリーム要素群がディスク領域に保存されている場合，その要素群をディスク領域から呼び出し，関数 f_1 に送る．ディスク領域に保存されていない場合，受け取ったデマンドを生産者関数 f_2 へ渡す．関数 f_2 の出力ストリーム id_2 の要素群を消費者関数 f_1 に渡すのにともない，ディスク上の空き領域をチェックする．ディスク上に空き領域がある場合，そのストリームをディスク領域に保存する．空き領域がない場合には，ストリームをディスク領域に保存せず，ストリームの再生成を行う．

式 (4) は，ストリーム id_2 の第 1 回目の生成のとき，問合せのディスクアクセス回数（$Ac(id_{1,1})$）を表す．式 (5) は，ストリーム id_2 の第 2 回目の生成開始から問合せ処理の終了までのディスクアクセス回数（$Ac(id_{2,1})$）を表す．ディスク領域に保存されている id_2 の要素群のサイズは，pid_2（ページ）で表される．

式 (4) と式 (5) において，ストリーム id_2 が第 1 回目に生成されている場合のバッファサイズを $bs_{1,i}$ と表し，第 2 回目の生成では，バッファサイズを $bs_{2,i}$ と表す．

$$\begin{aligned}
&Ac(id_{1,1}) = Ac(bs_{1,1}) + Sv(pid_2) + Ac(id_{1,2})\,,\\
&Ac(id_{1,2}) = Ac(ds_2) + \left\lceil \frac{ds_2}{bs_{1,2}} \right\rceil * Ac(id_{1,3})\,,\\
&\quad \cdots \qquad (4)\\
&Ac(id_{1,i}) = Ac(ds_i) + \left\lceil \frac{ds_i}{bs_{1,i}} \right\rceil * Ac(id_{1,i+1})\,,\\
&\quad \cdots\\
&Ac(id_{1,N}) = Ac(ds_N)\,.
\end{aligned}$$

式 (5) において，パラメータ φ は 0 と 1 間の実数（$0 < \varphi < 1$）であり，ストリーム id_2 の pid_2 ページ分の要素群を生成するために，ストリーム ds_2 の $ds_2 * \varphi$ ページ分の要素群を消費することを意味する．

$$\begin{aligned}
&Ac(id_{2,1}) = Ac(ds_1 - bs_{1,1}) + \left\lceil \frac{ds_1 - bs_{1,1}}{bs_{2,1}} \right\rceil * Ac(id_{2,2})\,,\\
&Ac(id_{2,2}) = Sv(pid_2) + Ac(ds_2 - ds_2 * \varphi) + \left\lceil \frac{ds_2 - ds_2 * \varphi}{bs_{2,2}} \right\rceil * Ac(id_{2,3})\,,\\
&\quad \cdots \qquad (5)\\
&Ac(id_{2,i}) = Ac(ds_i) + \left\lceil \frac{ds_i}{bs_{2,i}} \right\rceil * Ac(id_{2,i+1})\,,\\
&\quad \cdots\\
&Ac(id_{2,N}) = Ac(ds_N)\,.
\end{aligned}$$

特殊関数 f' により，問合せ処理中に必要なディスクアクセス回数は，式 (6) に示すように，ストリーム id_2 を第 1 回目に生成するために必要なディスクアクセス回数と第 2 回目の生成開始から問合せの処理終了までに必要なディスクアクセス回数の和である．

$$Ac(id_1) = Ac(id_{1,1}) + Ac(id_{2,1})\,. \tag{6}$$

次節以降，特殊関数 f' を利用する問合せ処理方式（中間結果ストリームをディスク領域に保存する方式）を f' **方式**，そうではないストリーム指向データベース処理方式（中間結果ストリームをディスク領域に保存しない方式）を**一般方式**と呼ぶ．

3.2 メモリ資源の割当て

問合せ処理中に必要なディスクアクセス回数は，一般方式を利用する場合には，式 (1) で示され，f' 方式を利用する場合には，式 (2)，(3)，(4)，(5)，(6) で示される．ここでは，式 (1)，(2)，(4)，(5) に対する最適なメモリ資源割当て方法を述べる．式 (3) の場合は，利用可能な全メモリ資源を関数 f_1 のバッファに割り当てるので，メモリ資源割当て計算の対象とならない．

ストリーム指向型データベース処理には，次の特徴がある．

特徴 1： 問合せの処理中に，最初のストリーム要素から最後の要素までを 1 個ずつ連続的に処理する．または，ストリームの第 i 番目の（最後の要素ではない）要素から最後の要素までを連続的に処理する．しかし，第 i 番目の要素を処理した後，第 j $(j \neq i+1)$ 番目の要素を処理するような飛び越し処理と逆順（$i > j$）の処理をしない．

特徴 2： ストリーム id_i の再生成回数が減ると，ストリーム $id_{i+1}, id_{i+2}, \ldots, id_N$ の再生成回数も減る．

本方式では，その特徴を利用し，バッファ $bs_1, bs_2, bs_3, \ldots, bs_i, \ldots, bs_{N-1}$ の順に各バッファの最適サイズを求める．

バッファ bs_N のサイズがストリーム ds_N への再生成回数に影響を与えないので，bs_N を 1 ページと設定する．

バッファ bs_i の最適なサイズを求めるために，まず，

利用可能なメモリ資源を，均等方式（利用可能なメモリ資源を各バッファに均等的に配分する方式）により，各バッファに割り当てる．その割当てサイズを初期値 x_0 とする．

$$x_0 = \frac{(M-1) - \sum_{k=1}^{i-1} bs_k}{N-i}, \quad \sum_{k=1}^{0} bs_k = 0 .$$

次に，一般方式の場合，式(1)に，x_0+x を bs_i として代入し，$x_0 - x/((N-1)-i)$ を $bs_{i+1}, id_{i+2}, \ldots, id_{N-1}$ として代入する．x は，0，または，0より大きい実数（$x \geq 0$）である．その結果は，式(7)を得る．

$$\begin{aligned} &Ac(id_i) = f_i(x) , \\ &f_i(x) = Ac(ds_i) + \left\lceil \frac{ds_i}{x_0 + x} \right\rceil * Ac(id_{i+1}) , \\ &Ac(id_{i+1}) = Ac(ds_{i+1}) \\ &\qquad + \left\lceil \frac{ds_{i+1}}{x_0 - x/(N-i-1)} \right\rceil * Ac(id_{i+2}) , \\ &\cdots \\ &Ac(id_N) = Ac(ds_N) . \end{aligned} \tag{7}$$

したがって，バッファ bs_i の最適なサイズを求める問題は，関数 $f_i(x)$ を最小値とする y_i を求める問題として表される．

$$f_i(y_i) = \min(f_i(x)) .$$

このとき，最適バッファサイズは，x_0 と y_i の和となる．

f' 方式の場合，最適なバッファサイズを求めるために，式(2)，(4)，(5)に x_0+x と $x_0 - x/((N-1)-i)$ を代入する．

$f_i(x)$ は，$\lceil \alpha \rceil$ により，非連続的な特性を持っている．次に，その非連続的な特性を考え，$f_i(x)$ の最小値に対応する y_i を求める方法を述べる．**図3**は，その方法を説明したものである．y_i を求めるときに，利用可能なメモリ資源は，$(N-i)*x_0$ である．バッファ $bs_{i+1}, id_{i+2}, \ldots, id_{N-1}$ を各1ページに割り当てると，x の最大値 $x_{\max}$ は $(N-i)*x_0 - ((N-1)-i)$ である．

x の最大値 $x_{\max}$ を均等的にL分割し，1つ分のサイズを δ とする．

バッファ bs_i の最適サイズを求める方法は，次のとおりである．

1. $0, \delta, 2\delta, \ldots, L\delta$ $(= x_{\max})$ に対する各 $f_i(j\delta)$ $(0 \leq j \leq L)$ の値を求め，$f_i(x)$ の最小値を獲得する．
2. $k\delta$ $(0 \leq k \leq L)$ を前のステップ1で求めた $f_i(x)$ の最小値になる値とすると，$(k-1)\delta$ から $(k+1)\delta$ まで（$k=0$ の場合，0から δ まで，$k=L$ の場合，$(L-1)\delta$ から $L\delta$ まで），1ページずつ加えて，$f_i(x)$ を求める．求めた $f_i(x)$ の各値の最小値を y_i とする．
3. バッファ bs_i の最適サイズは，$x_0 + y_i$ である．

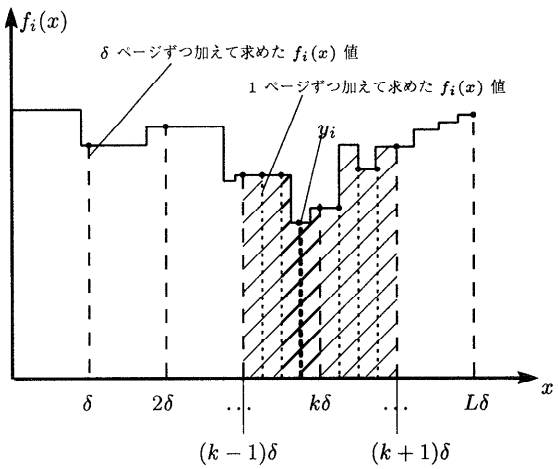


図3 最適なバッファサイズを求める方法
Fig. 3 The method for calculating the optimal buffer size.

δ を1ページ以上に設定すると，y_i を求めるために必要な計算量が減少するが，δ 値の設定が大きいと，求めた y_i は最適なバッファサイズと対応しない（$f_i(y_i) \neq \min(f_i(x))$）可能性がある．したがって，$\delta$ の設定には，計算結果の精度と計算量を考えなければならない．

本方式では，ディスクへのアクセス回数を参考基準として，δ の範囲を設定する．メモリを各関数に均等に割り当てると，各バッファサイズは $M/(N-1)$ となる．すなわち，$bs_1 = bs_2 = \ldots = bs_{N-1} = M/(N-1)$，$bs_N = 1$ となる．さらに，式(1)を利用し，ディスクへのアクセス回数 $T_{Q_{equ}}$ を求める．

$$\begin{aligned} T_{Q_{equ}} &= Ac(ds_1) \\ &\quad + \left\lceil \frac{ds_1}{M/(N-1)} \right\rceil * Ac(id_2) , \\ &\cdots \\ Ac(id_i) &= Ac(ds_i) \\ &\quad + \left\lceil \frac{ds_i}{M/(N-1)} \right\rceil * Ac(id_{i+1}) , \\ &\cdots \\ Ac(id_N) &= Ac(ds_N) . \end{aligned} \tag{8}$$

関数 f_1 の最適なバッファサイズを求めるために，M/δ 回の計算が必要である．bs_N がすでに1に設定されたので，各関数において最適なバッファサイズを求めるために，最大，$(N-1)*M/\delta$ 回の計算が必要である．最適なバッファサイズを求める計算の1回の計算時間は，ディスクを1回アクセスするのに必要な時間より短い．バッファサイズを求める処理時間の最大

値をディスクアクセス時間と同じと仮定すると，バッファサイズを求める必要な計算回数は，ディスクアクセス回数より多い．つまり，$((N-1)*M/\delta) > T_{Q_{equ}}$ となる．また，δ の最小値は，1であるので，δ の範囲は，次式で表される．

$$1 \leq \delta < \frac{(N-1)*M}{T_{Q_{equ}}}.$$

$((N-1)*M)/T_{Q_{equ}}$ が1より小さい場合，δ を1と設定する．

次に，疑似コードを用いて，提案方法の具体的な手続きを示す．

Query Processing
  Calculate buffer sizes;
  Execute the functions of the query
    until total stream elements of id_2 are generated;
  If (total elements of id_2 is saved in disk) then
    Set N as the number of functions;
    Release the memory occupied by f_2, f_3, ..., f_N;
    Add the released memory to bs_1;
    Execute functions f_1 and f' until query is finished;
  Else
    Get parameter φ;
    Set the consumed size of stream ds_2 to be $\varphi * ds_2$;
    Calculate buffer sizes;
    Execute functions of the query until query is finished;
  End If;
End Query Processing

Calculate buffer sizes
  Set N as the number of buffers;
  Set M as the number of pages of available memory;
  Set bs_N as 1;
  Set x_0 as
  $$x_0 = \frac{M - bs_N}{N-1};$$
Set $x_{\max}$ as
  $$x_{\max} = (N-1)*x_0 - ((N-1)-1);$$
Set i as 1;
Calculate the minimal value of y_i;
Set the optimal buffer size bs_1 as $x_0 + y_i$;
Set i as 2;
Loop while $i < N-1$
  Set x_0 as
  $$x_0 = \frac{(M - bs_N) - \sum_{k=1}^{i-1} bs_k}{N-i};$$
  Set $x_{\max}$ as
  $$x_{\max} = (N-i)*x_0 - ((N-1)-i);$$
  Calculate the minimal value of y_i;
  Set the optimal buffer size bs_i as $x_0 + y_i$;
  Set i as $i+1$;
End Loop;
Set the optimal buffer size bs_{N-1} as
$$M - bs_N - \sum_{k=1}^{N-2} bs_k;$$
End Calculate buffer sizes

Calculate the minimal value of y_i
  Get i;
  Get δ;
  Get $x_{\max}$;
  $L = x_{\max}/\delta$;
  Calculate $f_i(0)$ by Formula (7);
  $Minf_i = f_i(0)$;
  Set y_i as 0;
  $k = 1$;
  Loop while $k*\delta \leq L*\delta$
    Calculate $f_i(k*\delta)$ by Formula (7);
      If $(f_i(k*\delta) < Minf_i)$ then
        $Minf_i = f_i(k*\delta)$;
        Set y_i as $k*\delta$;
      End If;
    $k = k+1$;
  End Loop;
  $k = y_i/\delta$;
  $j = (k-1)*\delta$;
  If $(k \leq 0)$ then
    $k = 0$;
    $j = 0$;
  End If;
  If $(k \geq L)$ then
    $k = L-1$;
    $j = (L-1)*\delta$;

End If;
Loop while $j \leq (k+1) * \delta$
Calculate $f_i(j)$ by Formula (7);
If $(f_i(j) < Minf_i)$ then
$Minf_i = f_i(j)$;
Set y_i as j;
End If;
$j = j + 1$;
End Loop;

4. 実　　験

本章では，提案したメモリ資源割当て方式を適用した問合せ処理の実験結果を示す．実験は，一般方式と f' 方式の両者を対象として行う．比較評価は，問合せ処理中のディスクアクセス回数と利用可能なメモリ資源およびディスク領域について行う．

実験は，一般方式を用いた問合せ処理，および，f' 方式を用いた問合せ処理について行う．3.2 節で述べた $f_i(x)$ の最小値を求める方法の有効性を確認するために，一般方式を用いた実験を行った．f' 方式に関する実験は，中間結果ストリームをディスク領域に保存する場合の最適なメモリ資源割当て方式の有効性を確認するために行った．

f' 方式を適用する問合せ処理においては，バッファサイズを動的に求める必要がある．そのため，高速にバッファサイズを求める方法が必要である．本論文では，3.2 節において，高速にバッファサイズを求める方法を提案した．提案した方法により求めたバッファサイズは最適なサイズではない場合もある．つまり，求めた結果には，誤差が生じる．その誤差の範囲を求めるために，本実験では，最適メモリ資源割当てアルゴリズムを設定する．そのアルゴリズムは，文献 22) において提案した関係データベース問合せ処理に適用するメモリ資源割当て方式により求めたものである．実験では，まず，提案方法を用いて，バッファサイズを求める．次に，最適メモリ資源割当てアルゴリズムを用いて，同じパラメータの問合せ処理に対する最適なバッファサイズを求める．それらにより，求めた結果を比較し，誤差範囲を定める．

問合せは，図 2 に示すような構造を持つ．実験において，任意のデータベース演算を擬似的に実現するために，問合せを構成する関数はフイルタとマージ演算を用いた．関数 f_N はフイルタ演算であり，他の関数はマージ演算である．フィルタ演算は，1 入力ストリームを持ち，1 出力ストリームを生成する．マージ演算は 2 入力ストリームを持ち，1 方のストリーム id_{i+1} は生産者関数から生成され，他方のストリーム ds_i はディスクから読み込まれる．マージ演算は，ストリーム id_{i+1} の全要素に対し，ストリーム ds_i の全要素を参照し，マージ結果を生成する．

実験において，各データストリームのサイズを 1024 ページと設定した．ディスクアクセス操作はページ単位とする．δ の値は，1 と設定した．f' 方式に適用する問合せ処理について，中間ストリーム id_2 をディスクに保存できる領域がない場合，中間ストリームのサイズを，200 ページから 1050 ページまで変動させている．

ds_i：1024,
δ：　1,
id_2：200 ページから 1050 ページまで変動
（f' 方式かつ id_2 を保存できるディスク領域がない場合に適用する）．

データ量は，ページサイズに依存しており，ページサイズが 1 KB の場合，1024 ページで約 1 MB，ページサイズが 10 KB の場合 10 MB のデータサイズとなる．ページサイズは，I/O 処理システムに依存している．提案方式についての評価を I/O 処理システムに依存せずに行うために，本論文では，ページサイズを規定していない．

図 4 は，一般方式の場合において，利用可能なメモリ資源全体に対し，提案方式を用いて求めたメモリ割当て，および，最適なメモリ資源割当て各々の場合における問合せ処理時のディスクアクセス回数を示す．**図 5** は，図 4 の一部を拡大して示すために行った実験結果を示している．実験結果により，提案方式を用いて求めたメモリ資源割当ては，最適なメモリ資源割当てとほぼ同じであることが分かる．

表 1 は，利用可能なメモリ資源が 500，550，600，700，800，900，1000 ページの各々の場合において，提案方式により求めた各バッファサイズと最適なメモリ資源割当ての各バッファサイズを比較している．メモリ資源が 1000 ページである場合を除き，他の場合には，提案方式により求めた各バッファサイズと最適なメモリ資源割当てにおける各バッファサイズは同じである．

提案方式により求めたバッファサイズは，最適なバッファサイズと必ずしも一致しない場合がある．表 1 に示すように，メモリ資源 1000 ページの場合，本方式により求めたバッファサイズは最適ではない．求めたバッファサイズは，bs_1 が 512 ページ，bs_2 が 256 ページ，bs_3 が 231 ページである．各バッファの最適サイズは，bs_1 が 342 ページ，bs_2 が 342 ページ，bs_3 が 315

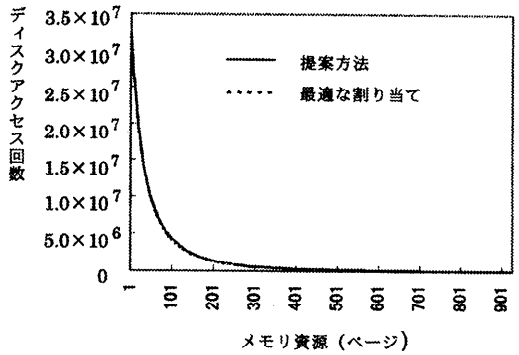


図 4 中間結果ストリームをディスクに保存しない場合のディスクアクセス回数（1）

Fig. 4 The number of disk accesses in the case that the intermediate stream is not stored into disk (1).

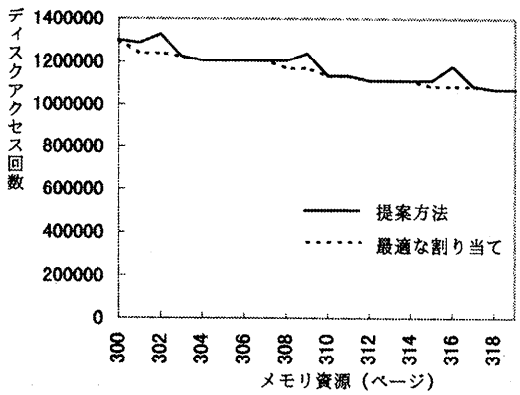


図 5 中間結果ストリームをディスクに保存しない場合のディスクアクセス回数（2）

Fig. 5 The number of disk accesses in the case that the intermediate stream is not stored into disk (2).

表 1 計算されたバッファサイズと最適なバッファサイズ（単位：ページ）

Table 1 Calculated buffer sizes and the optimal buffer sizes (unit: page).

メモリ	bs_1	最適 bs_1	bs_2	最適 bs_2	bs_3	最適 bs_3
500	205	205	147	147	147	147
550	205	205	171	171	173	173
600	256	256	171	171	172	172
700	256	256	256	256	187	187
800	256	256	256	256	287	287
900	342	342	342	342	215	215
1000	512	342	256	342	231	315

ページである．この場合，提案方式を適用する問合せ処理において必要とされるディスクアクセスは52224回である．メモリ資源が最適に割り当てられた場合，必要なディスクアクセス回数が50176であり，提案方式を用いる場合，ディスクアクセス回数が4%増えている．

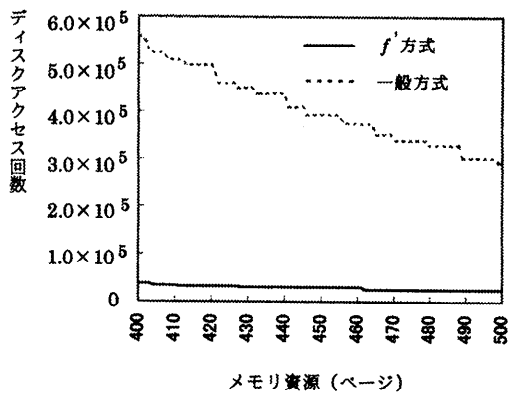


図 6 中間結果ストリーム全体をディスクに保存できる場合のディスクアクセス回数

Fig. 6 The number of disk accesses in the case that the total intermediate stream elements can be stored into disk.

この場合，メモリ資源1000ページの中で，6%のメモリ資源を有効に利用できない．まず，52224回のディスクアクセスを比較の基準点とする．これは，提案方式に適用する問合せ処理において，メモリ資源1000ページの場合に必要なディスクアクセス回数である．ここで，メモリ資源を1000ページから940ページまで減少させ，メモリ資源をこの減少に従い最適に割り当てる．メモリ資源の減少に従い，必要なディスクアクセス回数は50176から59392に増加する．メモリ資源が940ページより多い場合，必要なディスクアクセス回数は52224回よりも少ない．メモリ資源が940ページより少ない場合，必要なディスクアクセス回数は52224回よりも多い．そのため，この場合提案方式を用いて求めたメモリ割当ては，最適なメモリ割当てと比べて，6%のメモリ資源を有効に利用できないことになる．

図 6 は，中間結果ストリーム全体をディスクに保存できる場合の実験である．図6において，実線は f' 方式を適用する場合のディスクアクセス回数であり，点線は一般方式の場合のディスクアクセス回数である．実験結果により，f' 方式と一般方式を比べて，f' 方式では，ディスクアクセス回数が1/10に減少し，問合せ処理を大幅に効率化できている．

中間結果ストリーム要素の全体をディスクに保存できない場合における実験結果を，**図 7** に示す．実験において，中間結果ストリームを保存するためのディスクス領域を，400ページ，800ページ，1000ページと設定した．保存したページ分の中間結果ストリームを生成するために，924ページ分のストリーム ds_2 の要素群（$ds_2 * \varphi = 924$，$\varphi \approx 0.9$）が消費されるよ

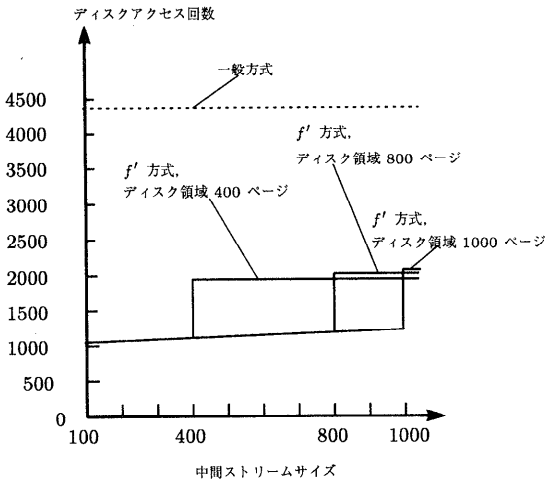


図 7 中間結果ストリームをディスクに保存できない場合のディスクアクセス回数

Fig. 7 The number of disk accesses in the case that the total intermediate stream elements can not be stored into disk.

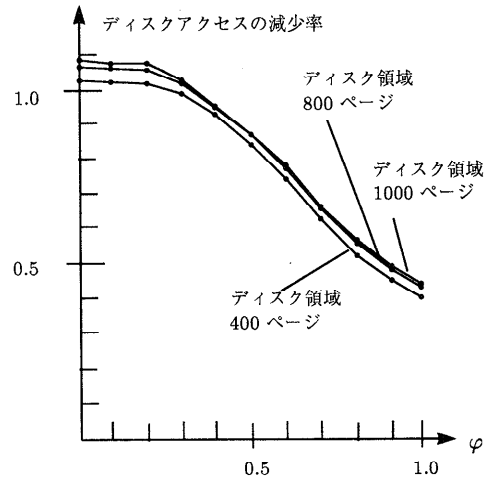


図 8 f' 方式を用いてディスクアクセス回数と中間結果消費量レート φ の関係

Fig. 8 The relationship between the number of disk accesses and φ, the rate of intermediate stream consuming.

うに設定した．実験では，中間結果ストリーム要素の数が徐々に増えていく．実験結果により，中間結果ストリームのサイズが，それを保存するためのディスク領域のサイズを超えない場合，ディスクアクセス回数は，ストリーム要素の数の増加に従って徐々に増えていく．中間結果ストリームのサイズが，それを保存するためのディスク領域のサイズを超えると，ディスクアクセス回数が急激に増加している．どちらの場合においても，f' 方式を用いることにより，ディスクへのアクセス回数が減少している．

中間結果ストリーム要素の全体をディスクに保存できない場合，f' 方式を適用する問合せ処理のディスクへのアクセス回数は，保存したページ分の中間結果ストリームを生成するためのストリーム ds_2 の消費量 $(\varphi * ds_2)$ の増加に従い，減少する．**図 8** はディスクへのアクセス回数が φ の増加に従って減少することを示している．図 8 に表示しているディスクへのアクセス回数は，f' 方式を用いるディスクへのアクセス回数と一般方式を用いるディスクへのアクセス回数の比である．φ が最小の場合 $(\varphi \to 0)$，ディスクス領域が 400，800，1000 ページの場合，f' 方式を用いるディスクへのアクセス回数は，一般方式を用いるディスクへのアクセス回数より，それぞれ，1.036，1.073 と 1.091 倍に増加する．φ が最大の場合 $(\varphi \to 1)$，f' 方式を用いるディスクへのアクセス回数は，一般方式を用いるディスクへのアクセス回数より，それぞれ，0.399，0.427，0.440 に減少する．図 8 に示すように，φ が 0.35 以上になる場合，f' 方式を用いて問合せを処理することにより，その処理効率が向上する．

5. むすび

本論文では，ストリーム指向型並列データベース処理方式を共有メモリ並列処理マシン上で実現する場合の動的メモリ資源割当て方式を提案した．まず，データベースに関する統計情報が提供されていない場合，および，ディスクを利用し，問合せ処理の途中で生成される中間結果ストリームを格納する場合において，問合せ処理を継続して行うために，ストリーム指向型並列データベース処理方式を拡張した．また，拡張した方式により，中間結果ストリームをディスクに格納する場合における最適なメモリ資源割当て方法を提案した．提案した方法は，中間結果ストリームをディスクに格納できるディスクスペースがある場合，および，そうではない場合のどちらにおいても有効である．最後に，問合せ処理の実験を行い，本論文で提案したメモリ資源割当て方式の有効性を明らかにした．

今後は，複数の中間結果ストリームを格納する場合における最適なメモリ資源割当てについて，検討を行っていく予定である．

参考文献

1) Bulterman, D.C.A., Hardman, L. and van Rossum, G.: Structured multimedia authoring, *Proc. 1st International Conf. on Multimedia,*

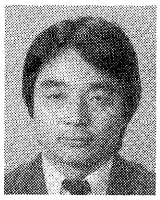
pp.283–289 (1993).
2) Little, T.D.C. and Ghafoor, A.: Interval-based conceptual models for time-dependent multimedia data, *IEEE Trans. Knowledge and Data Engineering*, pp.551–563 (1993).
3) Newcomb, S.R., Kipp, N.A. and Newcomb, V.T.: Hytime' the hypermedia/time-based document structuring language, *Comm. ACM*, Vol.34, No.11, pp.67–84 (1991).
4) Ripley, G.D.: Digital video interactive-a digital multimedia technology, *Comm. ACM*, Vol.32, No.7, pp.154–159 (1989).
5) White, C.: Heavy Metal Video: EDLs, Edit Suites and Nonlinear Editors, *Digital Video*, pp.76–79 (1994).
6) Breiteneder, C., Gibbs, S. and Tsichritzis, D.: Modeling of Audio/Video Data, *Proc. 11th International Conference on the Entity-Relationship Approach*, pp.322–339 (1992).
7) Little, T.D.C. and Ghafoor, A.: Synchronization and Storage Model for Multimedia Objects, *IEEE Journal on Selected Areas in Communication*, pp.413–427 (1990).
8) Little, T.D.C.: Time-Based Media Representation and Delivery, *Multimedia Systems*, Buford, J.F.K. (Ed), pp.175–200, ACM Press and Addison-Wesley (1994).
9) Duda, A.: Structured Temporal Composition of Multimedia Data, *Proc. 1995 International Workshop on Multimedia Database Management Systems*, pp.136–142 (1995).
10) Schloss, G.A. and Wynblatt, M.J.: Building Temporal Structures in a Layered Multimedia Data Model, *Proc. ACM Multimedia 94*, pp.271–278 (1994).
11) Hamakawa, R. and Rekimoto, J.: Object Composition and Playback Models for Handling Multmedia Data, *Multimedia Systems*, Vol.2, No.1, pp.26–35 (1994).
12) Adiba, M.: STORM: Structural and Temporal Object-oRiented Multimedia database system, *Proc. 1995 International Workshop on Multimedia Database Management Systems*, pp.12–19 (1995).
13) Weiss, R., Duda, A. and Gifford, D.K.: Composition and Search with a Video Algebra, *IEEE Multimedia*, Vol.2, No.1, pp.12–15 (1995).
14) Oomoto, E. and Tanaka, K.: OVID: Design and Implementation of a Video-Object Database System, *IEEE Trans. Knowledge and Data Engineering*, Vol.5, No.4, pp.629–643 (1993).
15) 清木, 端山：マルチメディア・データベースを対象としたシステム・アーキテクチャの検討, 情報処理学会データベースシステム研究会報告, 93-DBS-93, pp.75–81 (1993).
16) Sato, A., Hiroki, M. and Kiyoki, Y.: A stream-oriented parallel processing system for continuous media integration, *Proc. 14th IASTED International Conference on Applied Informatics*, pp.123–129 (1996).
17) Henderson, P.: *Functional Programming: Application and Implementation*, Prentice-Hall, Englewood Cliffs (1980).
18) Treleaven, P.C., Brownbridge, D.R. and Hopkins, R.P.: Data-driven and Demand-driven Computer Architecture, *ACM Comput. Surv.*, Vol.14, No.1, pp.93–144 (1982).
19) Amamiya, M. and Hasegawa, R.: Dataflow Computing and Eager and Lazy Evaluations, *New Generation Computing*, Vol.2, No.2, pp.105–129 (1984).
20) Friedman, D.P. and Wise, D.S.: Aspects of Applicative Programming for Parallel Processing, *IEEE Trans. Comput.*, Vol.C-27, pp.289–296 (1978).
21) Kim, W.: A New Way to Compute the Product and Join of Relations, *Proc. ACM-SIGMOD Conference on Management of Data*, pp.179–187 (1980).
22) 劉, 清木, 益田：関係演算のストリーム指向型並列処理における動的資源割り当て方式, 情報処理学会論文誌, Vol.29, No.7, pp.656–668 (1988).
23) Cornell, D.W. and Yu, P.S.: Integration of Buffer Management and Query Optimization in Relational Database Environment, *Proc. 15th VLDB Conf.*, pp.247–256 (1989).
24) Wolf, J.L., Iyer, B.R., Pattipati, K.R. and Turek, J.: Optimal Buffer Partitioning for the Nested Block Join Algorithm, *Proc. 7th IEEE International Conference on Data Engineering*, pp.510–519 (1991).
25) 村岡, 佐藤, 清木：ストリーム指向型並列データベース処理を対象とした分散メモリ資源割り当て方式, 情報処理学会論文誌, Vol.36, No.12, pp.2831–2843 (1995).
26) Chen, X. and Kiyoki, Y.: A New Memory Allocation Algorithm for Distributed and Stream-Oriented Query Processing, *14th IASTED International Conference on Applied Informatics*, pp.168–173 (1996).
27) Chen, X. and Kiyoki, Y.: Optimal Memory Resource Allocation for Stream-Oriented Database Processing Applied to Continuous Media Manipulation, *14th IASTED International Conference on Applied Informatics*, pp.232–237 (1996).

28) Kiyoki, Y., Kurosawa, T., Kato, K. and Masuda, T.: The Software Architecture of a Parallel Processing System for Advanced Database Applications, *Proc. 7th IEEE International Conference on Data Engineering*, pp.220–229 (1991).

(平成 9 年 9 月 12 日受付)
(平成 10 年 11 月 9 日採録)

陳　　幸（正会員）

1982 年中国西安交通大学電子工程系卒業．1987 年同大学計算機科学・工程系修士課程修了．1996 年筑波大学工学研究科博士課程修了．1982～1992 年中国西安交通大学計算機科学・工程系講師を経て，1996 年より筑波大学電子・情報工学系に勤務，現在，同学系助手．データベースシステム，マルチメディアシステム，並列分散システムの研究に従事．IASTED 会員．



清木　　康（正会員）

1978 年慶應義塾大学工学部電気工学科卒業．1983 年同大学院工学研究科博士課程修了．工学博士．同年，日本電信電話公社武蔵野電気通信研究所入所．1984～1995 年筑波大学電子・情報工学系講師，助教授を経て，1996 年より慶應義塾大学環境情報学部に勤務，現在，同学部教授．データベースシステム，知識ベースシステム，マルチメディアシステムの研究に従事．ACM，IEEE，電子情報通信学会，日本ソフトウェア科学会各会員．