計算機アーキテクチャ59−18
オペレーティング・システム29−18
(1985・10・25)

多重プロセッサデータベースマシンにおけるデータフローの最適化

OPTIMIZATION OF DATA FLOWS FOR A MULTIPROCESSOR DATABASE MACHINE

斎藤　直樹　　馬場　敬信
Naoki SAITO　Takanobu BABA
宇都宮大学工学部情報工学科
Utsunomiya University

1．はじめに

知識ベースや統計データベースなど、データベース（以下、ＤＢ）の発展に伴い、ＤＢシステムに対する処理の要求は増大する一方であり、これを支援するために、より高性能のＤＢマシンが求められている。ＤＢマシンに関しては、従来から、並列処理の活用を目指して数多くの提案がなされており、またその中には試作まで行なわれたものもある。これらの多くは、ディスクにおける並列トラックや、電子ディスクの活用等ハードウエアの特徴を生かして二次記憶からの高速アクセスを目指すと共に、データアクセスの過程でのマルチプロセッサによる並列処理を活用している[1,2]。

このように並列処理を取り入れる際に重要なことは、まずシステムの負荷の状況を調べ、最も効果的な所から並列化を進めることであろう。例えば、ＤＢマシンの設計に当たっては、従来ディスクＩ／Ｏのボトルネックに重点が置かれているものが多いが、問い合わせの内容によってはＣＰＵ上での処理の負荷が大きくなり、ＣＰＵネックとなる場合のあることも実験的に示されている[3,4,5]。

このような背景から、本研究ではまず関係ＤＢシステムを対象に、内部の処理におけるデータフローに着目してモデル化を行なう[6,7]。次に、このモデルのもとで与えられた問い合わせに対して可能なデータフローを生成し、空間的、時間的な並列処理を取り入れて、プロセッサ数ごとの各データフローの応答時間を評価する。この結果を用いて、最も有効なデータフローを選択すると共にそれをプロセッサに割付けるためのアルゴリズムを求める。

2．機能割付けの基本概念

2．1　ＤＢシステムの基本概念

ＤＢシステムの機能は、多くの基本的な要素から成る。図１はこれを簡単なモデルで示したものである。ターミナルのユーザから発生した問い合わせ（ＱＵＥＲＹ）は構文解析によって木構造に変換される。この結果生成された問い合わせ木は、コマンドコントローラによって実行されるが、その内容はデータ定義、データ操作、問い合わせ実行の３種類に分けられる。これらは更に、より基本的なルーチン（図では記憶システム）を起動する。

例えば、ＩＮＧＲＥＳ[8]における４つのプロセスは、

１．ターミナルモニタ
２．構文解析
３．コマンドコントローラ、問い合わせ最適化、実行
４．データ定義、データ操作

に対応する。Ｓｙｓｔｅｍ　Ｒ[9]におけるＲＤＳ（Relational Data System）とＲＳＳ（Research Storage System）はそれぞれ、

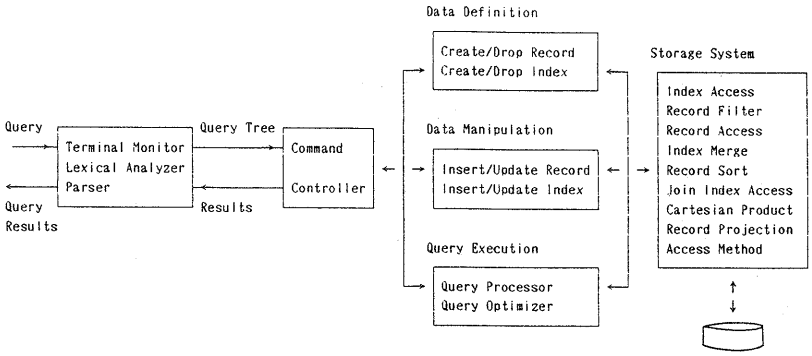

図１　データベースシステム機能の分割
FIGURE 1: A decomposition of database functions

１．構文解析、コマンドコントローラ、問い合わせ最適化、実行、選択、直積、射影
２．データ定義、データ操作、インデックス／レコード操作、マージ、ソート、結合

に対応する。

従って、多重プロセッサでＤＢマシンを実現する場合、これらの基本的な機能をいかにＤＢマシンに割付けるかが重要な課題となる。図２に、我々の提案する多重プロセッサへの割付けの過程を示した。まず問い合わせ木からはデータフローが生成されるが、これは処理のアルゴリズムや、インデックスの有無などにより種々の可能性がある。更に、このデータフローの各々に対して多重プロセッサへの割付けの可能性を検討し、最適な割付けを出力するのが多重プロセッサへの機能割付け（Function Allocator）の役目である。このように、割付けにあたっては、「データフローの生成」と「機能の割付け」の２つが要点となるが、今回は後者を中心に検討する。また、ここでは最適性の規範として、ＤＢにデータフローを与えてから結果が得られるまでの時間（応答時間）の最小化を目標とする。

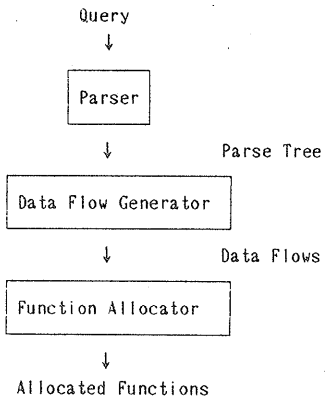

図２　基本機能割付けの基本概念
FIGURE 2: Allocation of data flow to multiprocessors

２．２　最適化の前提

最適化に当たって考慮すべきことは、基本的な機能をいかにＤＢマシンに実現するかである。もし、ハードウェアやファームウェアで実現する場合には、割付けられた機能に応じてプロセッサを特殊化することが考えられる。この時には、各機能の実行は速くなるが、１つのプロセッサに実現される機能は制限されると考えるのが妥当である。一方、ソフトウェアで実現する場合には、現在のメモリＩＣの価格と性能から考えて、１台のプロセッサにはＤＢ処理に必要な全機能が含まれると仮定してよいであろう。

前者の立場に対する最適化の方法については既に検討済み[10]のため、ここでは後者の立場に立って検討を行なう。

多重プロセッサＤＢマシンのアーキテクチャとしては、

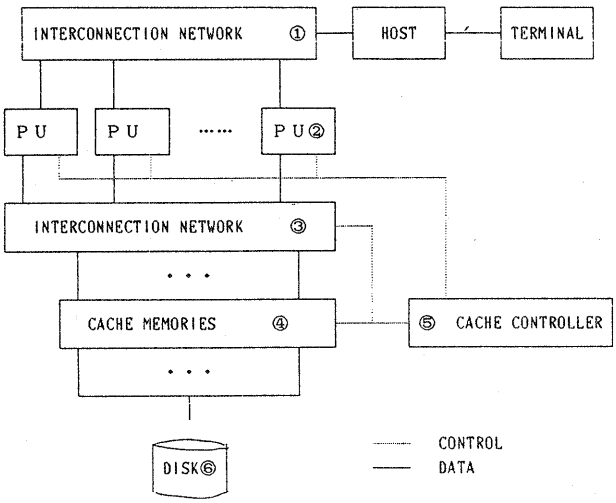

図３ データベース・マシンのアーキテクチャ
FIGURE 3: Database machine architecture

ごく標準的な図３のような構成を前提とする。

ＰＵ－ＰＵ間ネットワーク①には、ターミナルからの問い合わせを処理するホストと、実際にＤＢ処理を行なうバックエンド型のＰＵ②が結合されている。ホストは最適化を行ない選択したデータフローをＰＵに割付ける。更に、ディスク⑥に対してキャッシュ④を設け、ディスクＩ／Ｏのボトルネックを防ぐ。そして、ネットワーク③がＰＵ－キャッシュ間を結合しており、キャッシュコントローラ⑤がそれらの制御を行なう。

３． データフローとそのコスト

３．１ データフローグラフ

ＤＢ処理における基本機能を節点に対応させ、さらに基本機能間のデータの流れを、方向を持つ枝に対応させると、処理の過程は有向グラフとして表現できる。典型的な問い合わせに対する基本機能の一覧を表１に示す。

例えば、Restriction Index(RI) は、与えられた条件に従ってそれを満足するレコードへのポインタ集合を出力する。そのコストはunit(RI)･n+dio(RI)で表わされ、unit(RI)はＲＩに対するＣＰＵ上での単位演算時間、dio(RI) はディスクアクセス時間を表わす。

表１ 基本機能の動作とコスト
Table 1. Data flow and cost model of elementary operations

Elementary Function (f)	Contents	Processing Time C(f) (see footnote)
RI (Restriction Index)	according to input restriction predicate, RI accesses an index file and produces a set of pointers to records satisfying the predicate	unit(RI)･n+dio(RI)
JI(Join Index)	according to input join value, JI accesses an index file and produces a set of pointers to records which contain the same value for join attribute	unit(JI)･n+dio(JI)
RA (Record Access)	inputting a set of pointers, RI accesses a database and produces a set of records	unit(RA)･n+dio(RA)
S (Sequential Scan)	given a relation name, S sequentially scans a relation and produces a set of records	unit(S)･n+dio(S)
ST (Sort)	ST sorts and merges a set of input records so that they are grouped by ascending join values	unit(ST)･n･log(n)
P (Projection)	P extracts specified attributes of input records	unit(P)･n
RF (Restriction Filter)	according to restriction predicate, RF selects records satisfying the predicate	unit(RF)･n
JF (Join Filter)	JF receives two inputs: records grouped by a join value and join value processed by the other JF. JF checks the match and produces the records with matched value	unit(JF)･(n1+n2)
C (merge and Concatenation)	C inputs two sets of records grouped by join values and concatenates them with matching join values	unit(C)･n1･n2
I (Intersection)	I receives two sets of pointers, one of them grouped for a join value, and produces their intersection	unit(I)･(n1+n2)

unit(f) : unit CPU operation cost of function f.
dio(f) : disk I/O time for an input data segment.
n, n1, n2 : number of records or pointers to be processed.

データフローとしてここで考慮するのは、多変数に対する代表的な処理アルゴリズムとして知られているネステッドループジョインとソートマージジョインの2つである。

＜例1＞ジョインインデックスを用いたネステッドループジョイン（図4）

2変数の一方について、インデックスのアクセス（RI-2）を最初に実行する。他方の変数についてインデックスのアクセス（RI-1）を行ない、レコードをアクセス（RA-1）し、ジョインインデクスの共通項が2つの流れからとられる（I）。そして、レコードをアクセスした後（RA-2）、レコードの結合を行ない（C）、フィールドの取り出し（P）が、行なわれる。

＜例2＞ジョインインデックスを用いないネステッドループジョイン（図5）

インデックスをアクセス（RI-1）した後、レコードをアクセス（RA-1）する（インデックスがなければシーケンシャルスキャン（S））。他方の変数についてもインデックスのアクセス（RI-2）を行ない、レコードをアクセス（RA-2）し、これを与えられた条件によって選択（RF）した後、2つの流れのレコードの結合を行ない（C）、必要なフィールドの取り出し（P）を行なう。

＜例3＞ソートマージジョイン（図6）

2変数の両方について、インデックスのアクセス（RI-1 ,RI-2）を最初に実行し、レコードをアクセス（RA-1 , RA-2）する。次にそのレコードをソート（ST-1, ST-2）し、結合を行ない（C）、フィールドの取り出し（P）が行なわれる。

3．2　処理時間の計算

各データフローの処理時間の計算は次のように行なう。

まず、基本的な時間として単位データ（セグメントと呼ぶ）に対する処理時間を定義する。標準的なセグメントとしては、1ページ分のレコードなどが考えられる。

コストをディスクI／O時間、CPU時間、転送時間に分割して次のように計算する。fは基本機能を表わす。他の各変数は表2に示した。

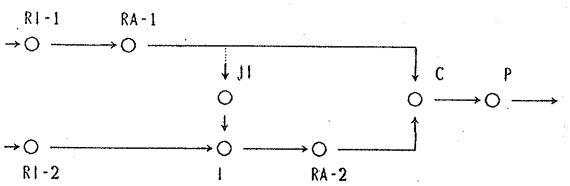

図4　JIを用いたネステッドループジョインのデータフロー
FIGURE 4: Data flow of nested loop join with join index

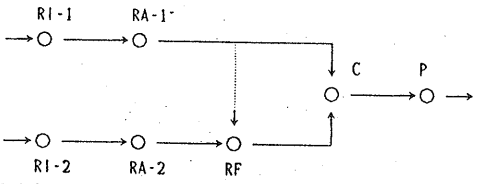

図5　JIを用いないネステッドループジョインのデータフロー
FIGURE 5: Data flow of nested loop join

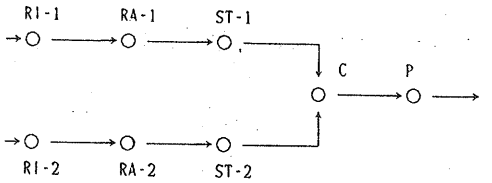

図6　ソートマージジョインのデータフロー
FIGURE 6: Data flow of sort and merge join

表2　変数表
TABLE 2: Database system parameters

Parameter	Meaning
C(f)	DIO(f)+CPU(f)+T(f)
DIO(f)	disk I/O time for a unit data segment
CPU(f)	CPU time for a data segment
T(f)	transfer time for a data segment between processors
d	average disk access time for a data segment
c	average cache access time for a data segment
rq	average time to wait for I/O. (seek and latency)
h	cache hit ratio
unit(f)	unit operation time of function f
t	unit transfer time
r(f)	selectivity of function f
wd	cache-processor bus contention time
wt	processor-processor bus contention time
N	number of segments
P	number of processors
Np	N/P (number of data segments to be processed in a processor)
k	number of records in a data segment
pipeC	pipeline processing time for a data segment
repCi	parallel processing time for a data segment, using i processors

$$C(f) = DIO(f)+CPU(f)+T(f) \quad (3.1)$$

ここで、各コストは以下のようになる。

$$DIO(f) = \begin{cases} rq+\{h\cdot c+(1-h)\cdot(d+c)\}+wd & \\ 0 \quad f\text{がディスクＩ／Ｏを行なわない時} & \end{cases} \quad (3.2)$$

$$CPU(f) = unit(f) \quad (3.3)$$

$$T(f) = t\cdot r(f)+wt \quad (3.4)$$

ディスクＩ／Ｏ時間は、シーク時間と回転待ち時間、キャッシュ又はディスクからＰＵへの転送時間、転送ネットワークの競合による待ち時間からなる。次にＣＰＵ時間は、基本機能に応じた処理時間からなる。転送時間は、基本転送時間、転送ネットワークの競合による待ち時間からなる。

4．　最適化

4．1　最適化の操作

複数の基本機能を多重プロセッサ上に実現する場合、分散の仕方によって、以下の３手法に分けられる。

（１）１台のプロセッサ上での逐次処理（複数の基本機能を一つにまとめる）
（２）異なるプロセッサ間の時間的な並列処理（パイプライン）
（３）異なるプロセッサ上の空間的な並列処理（同じ逐次処理を複数のプロセッサで実行する）

以下では単に（２）をパイプライン処理、（３）を並列処理と呼ぶ。

基本機能をＡ，Ｂ、プロセッサ数を２としたときのパイプライン処理と、並列処理の例を図７に示した。以下で、３手法によるコストを定義する。

（１）逐次処理

１つのプロセッサ内でのデータ転送を単にパラメータの受渡しで行なうものとし、その時間を無視すると逐次処理のコストとして以下の式が定義される。ただし、f1,f2,･･･fn は基本機能である。

$$C(g(f1,f2,\cdots fn)) = \sum_{i=1}^{n} \{CPU(fi)+DIO(fi)\} +T(fn) \quad (4.1)$$

（２）パイプライン処理

パイプライン処理においてセグメントは、ＰＵからＰＵへ次々と流れていく。ここでは、簡単化のために各セグメントの処理時間を一定と考える。この場合、最も時間のかかる逐次処理を行なうＰＵがネックとなり、パイプライン処理のコストもその時間で決定される。従って、各ＰＵの逐次処理のコストをC(g(f1,･･･,fi)),･･･,C(g(fj,･･･,fk)),C(g(fk+1,･･･,fn)) とすると、パイプライン処理のコストは、次の式で表わされる。

$$pipeC = \max\{C(g(f1,\cdots,fi)),\cdots,C(g(fj,\cdots,fk)),\ C(g(fk+1,\cdots,fn))\} \quad (4.2)$$

（３）並列処理

基本機能をｍ台のプロセッサに複写した時、ｍセグメントを同時に処理することが可能なので処理時間（DIO+CPU ）は１／ｍになる。ただし、ソートを並列処理で実行する場合、ソートしたセグメントをマージしなければならなくなる。このように、並列処理によって新たなコストを生じることもあるが、ソートを除きそのコストは無視できるものとする。

基本機能の逐次処理をg(f1,･･･,fn)とすると、ｍ台のプロセッサを用いた並列処理のコストは、以下のように定義される。

$$repCm = 1/m\cdot\sum_{i=1}^{n} \{CPU(fi)+DIO(fi)\} +T(fn) \quad (4.3)$$

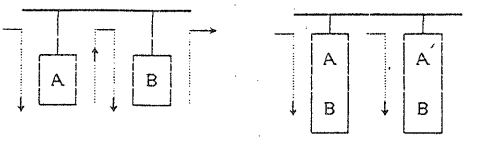

図７ パイプライン処理と並列処理
FIGUER 7: Parallel and pipeline dataflow

4．2　最適化のアルゴリズム

前節までの定義を用いることにより、次のことが言える。

「同一数のＰＵを用いて基本機能の割付けを行なう場合、並列処理は常にパイプライン処理より効果が大きい。」

この証明は付録に記した。

従って、常に並列処理を優先させて適用していけばよいのだが、処理のアルゴリズム上、並列処理だけでは済まないものがある。その代表的なものは先に述べたソートとそれに続くマージである。ソートを並列処理で実行すると必然的にマージとの間でパイプラインを形成することになる。

以上の議論をまとめると、ある与えられたデータフローに対する割付けのアルゴリズムは以下のようになる。

ルール１）

基本的には、１台のプロセッサに、全基本機能を載せる。そして、全プロセッサで並列処理を行なう。

ルール２）

ソートマージのように、アルゴリズム上ＰＵ間のデータ転送が必要となるものについては、パイプライン処理を行なう。

4．3　ソートマージジョイン

最適化において特別扱いするソートマージについて、更に具体的に考えると次の２つの方法が考えられる。

［方法１］Ｎセグメントを Ｐ台のＰＵが各々Ｎ／Ｐセグメントをソートした後、パイプラインを形成し、ソートしたセグメントを次々とマージする（図８）。

［方法２］始めからソートを行なうＰＵとマージを行なうＰＵに分けてパイプラインを形成し、ソートしたセグメントを次々とマージする（図９）。

ここで、１セグメントのソートコストをＣｓ、２セグメントのマージコストをＭ２とする。そして、ｍセグメントとｎセグメントのマージコストは、Ｍ２・ｍａｘ｛ｍ，ｎ｝と仮定する。

方法１についてコストは以下のようになる。

まず、１セグメントをｋレコードとした時のコストをＣｓ＝ｋ・ｌｏｇ ｋとする。各ＰＵは、Ｎｐ（＝Ｎ／Ｐ）セグメントのソートを行なうので、その時のソートコストは次のようになる。

$$k \cdot Np \cdot \log(k \cdot Np) \qquad (4.4)$$

ソート終了後、次に、ＰＵ間でマージを行なう過程がある。まず、Ｐ／２台の内容を他のＰ／２台に転送（図８(a)）し、空になったＰ／２台のうち、Ｐ／２－１台を用いて、パイプラインを形成（図８(b)）する。

そこでマージ過程のコストは、

（Ｐ／２台の内容転送）
＋（TOP のＰＵにセグメントが到着するまでの時間）
＋（TOP のＰＵが全セグメントを処理する時間）

$$= N \cdot t/P + (M2+t) \cdot \log(P/2) + (N-1) \cdot M2 \qquad (4.5)$$

となる。以下で式の意味を説明する。

マージ過程では、パイプラインを形成するために、まずＰ／２台の内容を他のＰ／２台に転送してＰ／２台を空にする（第１項目）。

パイプラインが形成されると、パイプラインの葉に相当するＰＵ（例えば、図８(b)ではＰＵ１）から根に相当するＰＵ（図８(b)ではTOP ）へセグメントが流れて

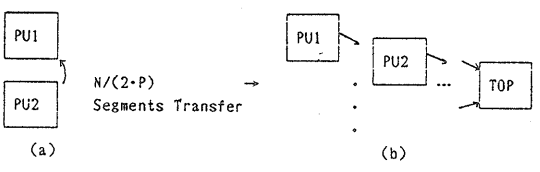

図８　ソートマージ（方法１）
FIGURE 8: Sort merge method 1

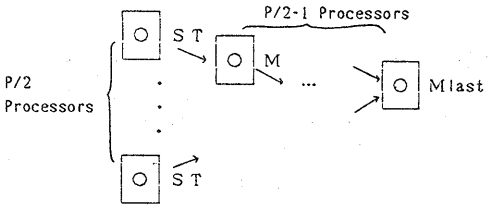

図９　ソートマージ（方法２）
FIGURE 9: Sort merge method 2

行く。TOP のＰＵに最初のセグメントが到着するまでの間、マージと転送がパイプラインの木の高さからTOP のＰＵを除いたlog （Ｐ／２）回行なわれる（第２項目）。

そして、TOP のＰＵに最初のセグメントが到着した後にTOP のＰＵでマージが行なわれる。ここでのマージはマージ済みのセグメントと流れてくるセグメントの大小関係が保証されているので、すでにマージ済みのセグメントとマージする必要がなく、Ｎ－１回のマージ（第３項目）で全セグメントのソートマージを完了する。

従って、(4.4) と(4.5) から全処理コストをＳＭＪ１とすると、以下の式で表わされる。

$$SMJ1 = k \cdot Np \cdot \log(k \cdot Np) + N \cdot t/P + (M2+t) \cdot \log(P/2) + (N-1) \cdot M2 \qquad (4.6)$$

方法２についてのコストは以下のようになる。

パイプラインの木の高さは、log （Ｐ／２）＋１であり、ソートとマージに用いるＰＵ数はそれぞれＰ／２台，Ｐ／２－１台である。

パイプラインではソートに用いたＰ／２台が葉に相当するので、Ｐ／２セグメントが１回のデータフローの単位となる。そして、流れた単位の数をステージ数と呼ぶ。

方法１と異なりセグメント内のレコードはソートされているが、セグメント間ではソートされていないため、すでにマージ済みのセグメントとマージする必要が生じる。従って、最後の段（マージ）のコストMlastは、ステージ数の増加に比例して、以下のようになる。

（Ｐ／２セグメントのマージ）
＋（前回までのマージ済みセグメントとＰ／２セグメントのマージ）

ここで、第１項目は次のようになる。

（Ｐ／２セグメントのマージ）＝ (P/2-1)・M2

次に第２項目の（前回までのマージ済みセグメントとＰ／２セグメントのマージ）では、ステージ数とコストに以下の関係が成り立つ。

ステージ数	1	2	i
コスト	0	(P/2-1)・M2	(i・P/2-1)・M2

従って、Ｍlastは常に他のマージよりコストが高いので、(4.2) からＣｓとＭlastの大小関係でパイプライン処理のコストが決まる。すなわち、このパイプライン処理のコストは、max{ Cs, Mlast }となる。そこで、パイプライン処理のコストを決めるためにＣｓとＭlastの大小関係を比較しなければならない。ｊ回目のステージのＭlastは、以下のようになる。

$$(P/2-1) \cdot M2 + \sum_{l=1}^{j} (l \cdot P/2-1) \cdot M2 = M2 \cdot \{P/2-1+j \cdot (P \cdot (j+1)/4-1)\} \qquad (1 <= j <= 2 \cdot N/P)$$

ｊ回目のステージで、Ｃｓ＜Ｍlastになったとすると、逆にｊ－１回目まではＣｓの方がコストが高いと言える。（ｊは上式とＣｓを比較して決定する）

パイプライン処理のステージ数ｊ－１回目までのコストは、ソートが決定し、

$$(j-1) \cdot (Cs+t) \qquad (4.7)$$

となる。残りステージ数ｊ回から２・Ｎ／Ｐ回までのコストは、マージ（Ｍlast）が決定するので、

$$(P/2-1) \cdot M2 \cdot (2 \cdot N/P-1) + \sum_{k=j}^{2N/P} (l \cdot P/2-1) \cdot M2 = M2 \cdot (2 \cdot N/P-j) \cdot (N/2-P \cdot j/4-1) \qquad (4.8)$$

となる。

従って、(4.7) と(4.8) から全コストをＳＭＪ２とすると、以下の式となる。

$$SMJ2 = (j-1) \cdot (Cs+t) + M2 \cdot (2 \cdot N/P-j) \cdot (N/2-P \cdot j/4-1) \qquad (4.9)$$

５．適用例

関係を以下のように定義する。

SUPPLIERS (SNO, SNAME, ITEM, CITY)
各10,20,20,10 ﾊﾞｲﾄ，計60ﾊﾞｲﾄ
ORDERS(NO, NAME, ITEM, QUANTITY)
各10,20,20,30ﾊﾞｲﾄ ，計80ﾊﾞｲﾄ

関係SUPPLIERS は、部品供給者の番号、名前、品名、都市名を示し、関係ORDERSは発注の番号、発注者名、品名、量を示している。この二つの関係に対して、「Brooks氏の発注した品物を保有し、New Yorkにいる供給者」を求める問い合わせは以下のようになる。

```
SELECT  SNAME, SUPPLIERS.ITEM,
  FROM  SUPPLIERS, ORDERS
 WHERE  NAME='Brooks, B.' AND
        SUPPLIERS.ITEM=ORDERS.ITEM, AND
        CITY='NEW YORK'
```

ここで、NAMEとCITYに対するインデックス（ＲＩ）の有無、及びITEMに対するジョインインデックス（ＪＩ）の有無に応じて種々のタイプのデータフローの可能性がある。以下で、ＮＬＪはネステッドループジョイン、ＳＭＪはソートマージジョインを意味する。

Ａタイプ　ＪＩを用いたＮＬＪ（図４に対応）	２変数共にＲＩ可
	１変数にＲＩ可
Ｂタイプ　ＪＩを用いないＮＬＪ（図５に対応）	２変数共にＲＩ可
	１変数にＲＩ可
	ＲＩ不可
Ｃタイプ　ＳＭＪ（図６に対応）	２変数共にＲＩ可
	１変数にＲＩ可
	ＲＩ不可

以下では、２変数共にＲＩ可でＪＩ可とし、ＰＵ数を７台とした時の割付けをタイプごとに示す。

＜Ａタイプ＞ＪＩを用いたＮＬＪによる割付けを示す（図１０）。

ルール１）により並列処理を適用してデータフローの全関数を全ＰＵに割付ける。ここで、割付けた基本機能の記述順に意味はない。また、括弧はデータフローに表われるものの、他の基本機能と共有できる基本機能であることを示す（例えば、RI-1とRI-2は同じRIである）。

＜Ｂタイプ＞ＪＩを用いないＮＬＪによる割付けを示す（図１１）。

ルール１）により並列処理を適用してデータフローの全関数を全ＰＵに割付ける。

＜Ｃタイプ＞　ＳＭＪによる割付けを示す。

ルール２）により４．３節で述べた２つの方法による割付けを順に示す。

まず方法１であるが、方法１を用いた割付けでは３つの段階がある（図１２）。

１）ＰＵ内ソートを完了するまでの段階（図１２の（１））。

ここで、各ＰＵはＮ／Ｐセグメント（今は、Ｎ／７セグメント）のソート処理を実行する。

２）ＰＵ間でマージする段階（図１２の（２））。

この段階では、各ＰＵがソートしたＮ／７セグメントが次々にＰＵ間を流れてマージされていく。

３）マージ結果を各ＰＵに分配して、ジョインする段階（図１２の（３））。

この段階は、マージの完了したセグメントが１つのＰＵに集中しているので、各ＰＵに分配して結合しなければならない。ここでは、ソートマージが済んでいるのでルール１）に従い、全ＰＵに並列処理が適用される。

次に、方法２による割付けを行なう。

方法２は始めからソートとマージ間をパイプラインで実現するので、方法１の１、２）の段階が図１３となる。ただし、段階３）は方法１と同一である。

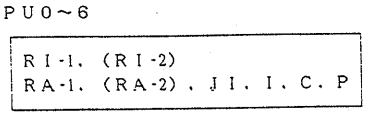

図１０　ＪＩを用いたネステッドループジョインの割付け
FIGURE 10: Allocation of functions for nested loop join with join index

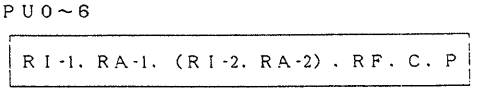

図１１　ＪＩを用いないネステッドループジョインの割付け
FIGURE 11: Allocation of functions for nested loop join without join index

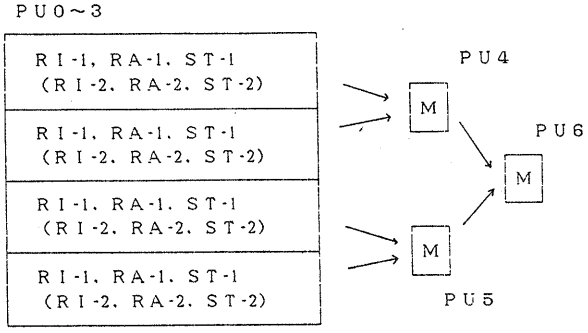

図１３　ソートマージジョインの割付け（方法２）
FIGURE 13: Allocation of functions for sort merge join method 2

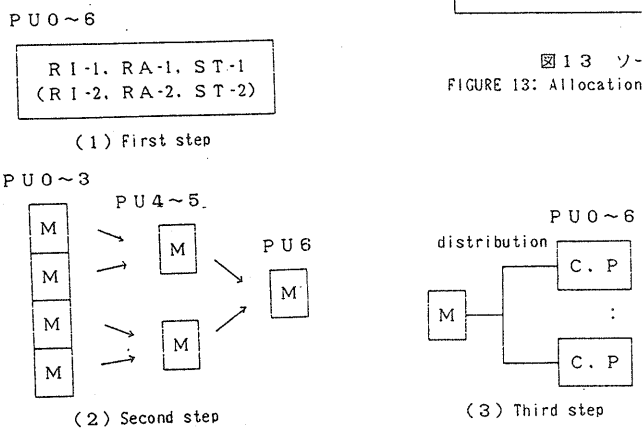

図１２　ソートマージジョインの割付け（方法１）
FIGURE 12: Allocation of functions for sort merge join method 1

前記の割付けを基本として、下記の値を用いてシミュレーションを行なった。値の設定に当っては、文献[11, 12]を参考とした。図１４は、対象とする２つの関係SUPPLIERS（１レコードは60バイト）とORDERS（１レコードは80バイト）のセグメント数（１セグメントは10レコード）を共に100 とした時の結果を示す。この時、総レコード数はSUPPLIERS 、ORDERS共に1000である。図１５は、２つの関係のセグメント数を共に10000 とした時の結果を示す。この時、総レコード数はSUPPLIERS 、ORDERS共に100000である。

rq：　100 μs
h：　0.6
d：　1　μs／byte
c：　0.5　μs／byte
k：　10　レコード
unit(f)：　0.1　μs／byte
Cs：　unit(f)・k log k　μs／byte
M2：　2・unit(f)・k　μs／byte
t：　0.1　μs／byte
r(RI-1)＝r(RI-2)＝0.1, r(P)＝0.3, r(JI)＝2, r(I)＝0.2, r(RF)＝0.4, wd＝wt＝0

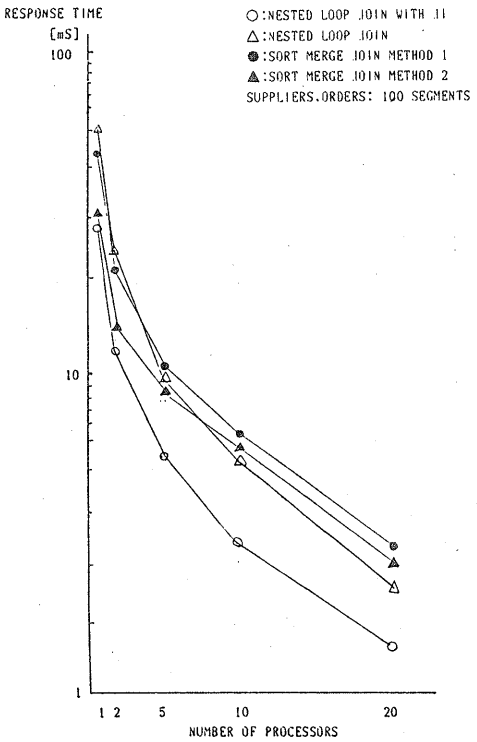

図１４　データフローとプロセッサに対する
応答時間の比較（セグメント数100 ）

FIGURE 14: Response times of the algorithms
versus the number of processors

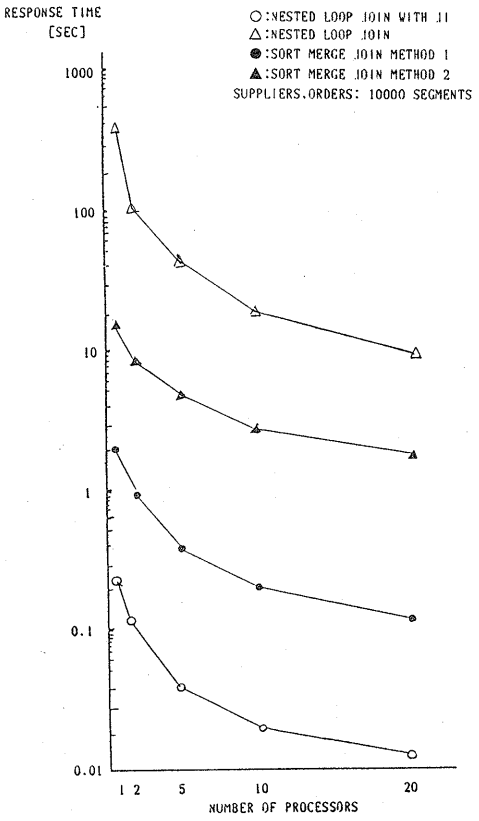

図１５　データフローとプロセッサ数に対する
応答時間の比較（セグメント数10000 ）

FIGURE 15: Response times of the algorithms
versus the number of processors

結果を以下に要約する。

１）４つの手法の比較

ＪＩを用いることが可能ならば、ＪＩを用いたＮＬＪが最適なデータフローであると言える。しかし、ＪＩを用いることができない場合、他の３つの手法の優劣については、ＰＵ数、データ量に依存して変わるため一概に言えない。

２）ＪＩの効果

ＪＩを用いたＮＬＪは、ＪＩを用いないＮＬＪより結合するデータ量が少なくなり、応答時間も短くなる。そして、処理するセグメント数が大きくなるとＪＩの効果は顕著になる。

３）ＳＭＪでの２つの方法の比較

セグメント数が多いと、方法２では最終段のマージの負荷が格段に増大して、そのためパイプラインの流れが滞り、応答時間は方法２＞方法１となる。従って、方法２では最終段のマージの高速化を行なわないと方法１に対抗できない。しかし、セグメントが少ない場合、方法２では最終段のマージの負荷が低く、パイプラインが有効に働くために方法１より有利になる。

特に１）の結果は、アルゴリズムの優劣が一意的に決まらず、ＪＩの有無、ＰＵ数やデータ量に依存して決まることを示しており、本論文で述べたような最適化の方法が有効であることを裏付けている。

6. まとめ

データベースシステム内部のデータフローのモデル化を行ない、これに基づいて、多重プロセッサ計算機を対象とした、処理機能の割付けを考えた。研究はまだ初期の段階にあり、本論文で述べたことが一般のデータベースシステムに問題なく適合するか否かはこれからの検討課題である。特に、プロセッサの多重化に伴うディスクＩ／Ｏや結合バスの競合の問題は重要な課題である。今後、実際のデータベースシステム内部の検討を行ない、より実際的なモデルを定義するとともに、多重プロセッサシステムへの応用を考えて行きたい。

謝辞

本研究を進めるに当たり貴重な御意見を頂いたメリーランド大学S.Bing Yao助教授、ならびに宇都宮大学　奥田健三教授、山崎勝弘助手に感謝する。

付録

任意の数の基本機能に任意の数のプロセッサを与えた時、常にパイプライン処理より並列処理が有利であることを示す。

基本機能を以下のようにする。（プロセッサ数＜基本機能数の時はｇ個の基本機能群に統合済みとする）

(F1,F2, …,Fg)

ここで、プロセッサＰ台（Ｐ≧ｇ）を各基本機能に割付けた場合を考える。まず、パイプライン処理で１セグメントを処理するのに必要なコストは、以下のようになる。

$$pipeC = \max\{C(F1)/N1, \cdots C(Fi)/Ni, \cdots C(Fg)/Ng\} = C(Fm)/Nm \qquad (1<=m<=g), p=\sum_{j=1}^{g} Nj$$

次に、全基本機能を１つにまとめて並列処理を適用した場合、並列処理ではＰセグメントをＰ台のプロセッサで並列に処理できるので、１セグメントを処理するのに必要なコストは、以下のようになる。

$$repCp = C(F1+F2+\cdots+Fg)/P = \sum_{j=1}^{g} Fj/P \qquad (1<=j<=g)$$

ここで、ｒｅｐＣｐの値を確定すると、パイプライン処理のコストＣ（Ｆｉ）／Ｎｉが最小値をとるのは、パイプラインの各要素（Ｃ（Ｆ１）／Ｎ１，…Ｃ（Ｆｉ）／Ｎｉ，…Ｃ（Ｆｇ）／Ｎｇ）が全て同じ値をとる時である。その時、パイプライン処理のコストの最小値は、

$$\sum_{j=1}^{g} Fj/P$$

となる。従って、

$$pipeC = C(Fm)/Nm \geqq \sum_{j=1}^{g} Fj/P = repCp$$

となり、常にパイプライン処理より並列処理が有利である。

証明終わり。

参考文献

[1] "Special Issuue on Database Machines," IEEE Computer, Vol.12, No. 3, March 1979.

[2] "Special Issuue on Database Machines," IEEE Transactions on Computers, Vol.C-28, No. 6, June 1979.

[3] C.J.Date: An Introduction to Database Systems, Vol.2, Addison-Wesley , 1983.

[4] R.Epstein and P.Hawthorn: "Design Decisions for the Intelligent Database Machine," Proceeding of National Computer Conference, pp. 237-241, 1980.

[5] M.Stonebraker: "Retrospection on a Database System," ACM Transactions on Database Systems, Vol.5, No.2, pp. 225-240, June 1980.

[6] T.Baba, S.B.Yao, A.R.Hevner, and Z.Shi: "Multiprocessor Database Machine Architectures Based on Data Flow Analysis," Proceeding of the 16th Annual Hawaii International Conference on System Sciences, January 1983.

[7] 清木、長谷川、雨宮：" 先行・遅延評価機構を用いた関係演算処理方式" 情報処理学会論文誌, Vol 26, pp 685-695,1985.

[8] M.Stonebraker, E.Wong, P.Kreps, and G.Held: "The Design and Implementation of INGRES," ACM Transactions on Database Systems, Vol.1, No.3, pp. 189-222, September 1976.

[9] M.Astrahan, et al.: "System R: Relational Approach to Database Management," ACM Transactions on Database Systems, Vol.1, No.2, pp. 97-137, June 1976.

[10]馬場, S.B.Yao, A.R.Hevner : "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回全国大会論文集, pp. 757-758, 1984.

[11] R.k.Shultz and R.J.Zingg: "Response Time Analysis of Multiprocessor Computers for Database Support," ACM Transactions on Database Systems, Vol.9, No.1, pp. 100-132, March 1984.

[12]菅, 上田, 田中："ディスク・キャッシュ装置のシミュレーションによる効果測定" 情報処理学会論文誌, pp. 22-28, Jan. 1981.

盛光印刷所