データベースの多様な応用分野に対応可能な関数型並列処理システムSMASH
--演繹データベースの並列処理系--

3H－8

波内みさ　清木 康
筑波大学

1　はじめに

近年のデータベース・システムの普及に伴い、データベースの応用分野が多様化してきている。従来のデータベース・システムは、事務情報の処理を主要な対象として構成されているので、応用分野の多様化に十分に対応することができない。そこで、データベース自身の枠組みの拡張に関する研究が行われている。

我々は、データベースの多様な応用分野および知識ベースに柔軟に対応することができる並列処理システムSMASHを設計し、その実現を行っている[1]。SMASHは、問い合わせを実行するための任意の基本演算を、ストリーム・データを引数とする関数としてシステム内に統合し、それらを関数型計算の枠組みの中で並列に実行する環境を実現するソフトウェア・システムである。このシステムでは、関数型計算の評価方式の1つとして知られる要求駆動型評価とストリームの概念をデータベース処理に適用している。

本稿では、共有メモリ型汎用マルチプロセッサ・マシンに実現したSMASH上での演繹データベースの処理方式と、その処理系における資源割り当て方式について述べる。

2　演繹データベースの並列処理方式

演繹データベースは、関係データベースに一階述語論理の推論機能を導入することにより、より高度な問い合わせの支援を可能とするデータベースである。

本稿で述べる演繹データベースの問い合わせ処理方式は、並列処理システムSMASH上に実現されている。問い合わせは、後向き推論によるインタプリテーション方式を実現する3種類の基本演算により処理される。

2．1　並列性の抽出

SMASHでは、要求駆動型評価のもとで、関数計算に内在する次の2種類の並列性が引き出される[1]。

(1) 関数引数の並列評価
(2) 関数の適用側とその本体側での並行評価（ストリーム型並列処理）

演繹データベースの問い合わせに内在する並列性は、基本的に、論理型計算におけるホーン節レベルのＯＲ並列性およびＡＮＤ並列性に対応する。本処理方式では、処理のための基本演算を、ホーン節の集合としてみなせるリレーションを対象とした集合演算のレベルに設定し、集合を単位として、それらの基本演算あるいは基本演算群を並列に処理する。

2．2　問い合わせ処理のための基本演算

演繹データベースの問い合わせを実行するために、図1に示す3種類の基本演算（AND、R_unify、F_unify）を設定した([2],[4])。これらの3つの基本演算は、各々関数として定義され、引数の生産者側関数からホーン節中の変数と値の束縛環境を表す集合をストリームの形式で受け取る。そして、消費者側の関数に対して、新しい束縛環境の集合を返り値としてストリームの形式で返す。問い合わせのインタプリテーションは、各R_unify、F_unify関数中で、これらの束縛環境の書換えを行うことにより進められる。

各関数では、次のような処理を行う。

(1)AND
内包データベースあるいは問い合わせ中のルール節のインタプリテーションを実行する。

(2)R_unify
関数ANDからホーン節のボディ部のリテラルを1つ受け取り、それに対するインタプリテーションを行う。

(3)F_unify
ホーン節ボディ部中の1リテラルをANDから受け取り、そのリテラルに関する変数の束縛環境の集合と、そのリテラルと同一のリレーション名を持つタプルとの間で、集合演算として単一化を実行する。

末尾再帰的に定義されているルールを対象とした問い合わせについては、1回の再帰呼び出しに対応して生成されるルール／ゴール・グラフの部分を1プロセッサに割り当てる。そして、プロセッサ台数に対応したルール／ゴール・グラフを生成し、並列処理を行う。この場合、そのルール／ゴール・グラフは、図1に示すように、再帰呼び出しに対応して繰り返し用いることになる([2],[4])。

要求駆動型評価における1回のデマンドに対して生成されるストリーム・データの量を、以下では、グラニュラリティ（granularity）とよぶ。各関数間には、ストリーム型の並列性の制御を行うために中間バッファを設ける。この中間バッファのサイズが、ストリーム・データのグラニュラリティに相当する。したがって、このサイズを設定することが、関数間で受け渡されるグラニュラリティを決定することになる。

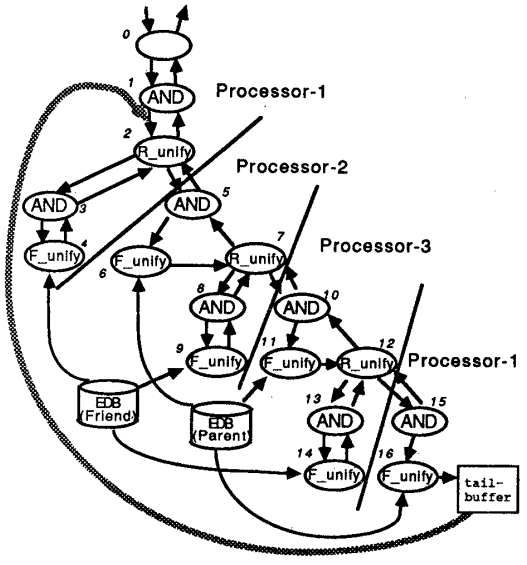

図1　基本演算と再帰的問い合わせの処理方式

3　演繹データベースにおけるメモリ資源割り当て方式

問い合わせを構成する各関数へのメモリ資源の割り当て

A Functional Parallel Processing System SMASH supporting a wide variety of database applications
-- Parallel processing for deductive databases --
Misa Namiuchi, Yasushi Kiyoki
University of Tsukuba

に関して、関数間の中間バッファを次の２種類に分類する。

(a)入力バッファ群
(b)出力バッファ群

入力バッファ群は、各関数の処理対象である入力ストリーム（変数の束縛環境）を各関数に伝達するバッファ群である。このバッファ群の大きさは、各関数でのコンテクスト・スイッチの回数とストリーム型並列性の抽出に影響を与える他に、関数F_unifyでのリレーション・アクセスの回数にも影響を与える。

一方、出力バッファ群は、各関数での処理結果の束縛環境をストリームとして伝達するバッファ群である。

ここでは、(a)の入力バッファ群に関するメモリ資源の割り当てについて、(1)各関数における束縛環境群の処理時間を最小にするための割り当て方式と、(2)各関数での処理対象データの待ち時間を最小にするための方式の２方式について考察する。

ここで、入力バッファ群に属する中間バッファの内で、F_unifyへ入力データを伝達するバッファとF_unifyでの処理結果を出力するバッファ以外は、１つの関数に対する入力用バッファと出力用バッファのサイズを同じ大きさに設定することにする。

３．１　束縛環境の処理時間を最小にするための方式

文献[3]では、関係演算において、処理のグラニュラリティと処理時間の関係を式で表し、関係データベースにおけるメモリ資源の最適なアロケーションを与えるアルゴリズムを提案している。このアルゴリズムを、上述の演繹データベースの並列処理方式におけるメモリ資源の割り当てに適用する。すなわち、F_unifyでの処理時間を中心に、各世代毎の問い合わせの処理時間を漸化式で表し、[3]のアルゴリズムを適用することによって、全ての関数F_unifyに関係する中間バッファについて、処理時間を最小にするためのメモリ資源の割り当てを求めることができる[4]。

３．２　各関数での待ち時間を最小にするための方式

関数間の中間バッファのサイズが大きく設定されている時、そのバッファから伝達される引数データの生成完了を待って、関数がその処理を一時中断されることがある。すなわち、中間バッファのサイズ分の引数データが消費される時間よりその引数データが生成される時間の方が長い場合、その時間差分の待ち時間が関数に生じる。

ここで、図２の様なモデル化を行う。このモデルでは、１つのプロセッサに割り当てられた関数群のうち、ＤＢにアクセスを行う負荷の重い関数が１つだけ存在するものとする。このモデルにおいて、入力データのストリーム型並列性が最大限に引き出される条件を式で表すと、次のようになる。

t_s = sort の係数
t_{bs} = search の係数
t_{io} = 1block の I/O にかかる時間
t_{cs} = 1 回の context switch の時間
Bin_k = 関数kへの入力用のバッファの大きさ [tuple]
DB_size_k = データベースkのサイズ [record]
DB_sizeb_k = データベースkのサイズ [block]
sf_k = データベースkの選択率

$$T_G = (t_{bs} * (log_2 Bin_i + sf_i * Bin_i) * DB\_size_i + t_s * Bin_i * log_2 Bin_i + t_{cs} + t_{io} * DB\_sizeb_i) * \frac{Bin_j}{sf_i * Bin_i * DB\_size_i}$$

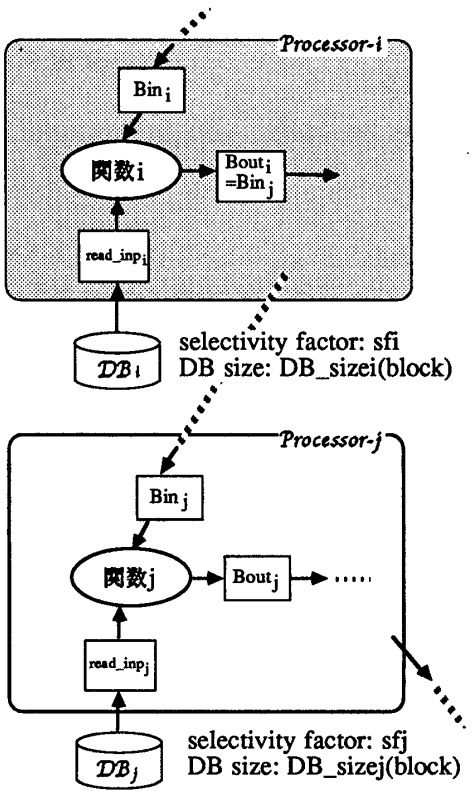

図２　資源割り当てのためのモデル

$$T_C = t_{bs} * (log_2 Bin_j + sf_j * Bin_j) * DB\_size_j + t_s * Bin_j * log_2 Bin_j + t_{cs} + t_{io} * DB\_sizeb_j$$

入力データの生成時間：$T_G \leq$ 入力データの消費時間：T_C

４　おわりに

本稿では、演繹データベースの問い合わせ処理を、関数型計算の枠組みの中で並列に実行する方式について述べた。

また、提案方式に関して、関数での処理時間および待ち時間をそれぞれ最小にするためのメモリ資源の割り当て方式を示した。一般に、この２つの立場からは異なるメモリ割り当てが導かれるが、両者を組み合わせることによって、よりきめの細かい割り当てができることが可能である。現在、この割り当て方式について検討を行っている。

参考文献：

[1] Y. Kiyoki, K. Kato and T. Masuda, "A relational database machine based on functional programming concepts," Proc. 1986 ACM-IEEE Computer Society Fall Joint Computer Conf., pp. 969-978, 1986.

[2] Y. Kiyoki, P. Liu and M. Namiuchi, "An Implementation Method for Logical Query Processing based on the Parallel Interpretation Approach," The 2nd Joint Scandinavian-Japanese Seminar on Information Modelling and Knowledge Bases, 1989.

[3]劉、清木、益田、”ストリーム指向型関係演算処理におけるバッファ資源割り当ての計算方式、”情報処理学会論文誌、Vol. 29, No. 8, pp. 770-781, 1988.

[4]波内、清木、劉、”演繹データベースの並列処理方式の実現と資源割り当て方式、”電子情報通信学会データ工学研究会報告、DE89-19、pp. 9-16, 1989.