TM-0293

並列論理型言語による検索処理プロセスの並列制御とその評価

武脇敏晃，伊藤英則

March, 1987

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列論理型言語による検索処理プロセスの並列制御とその評価

武脇敏晃、伊藤英則

(財)新世代コンピュータ技術開発機構

1. はじめに

著者らは、並列論理型言語による知識ベースシステムの構築を行っている。

本稿では、並列論理型言語GHC[Ueda 86]によって複数の検索プロセスを並列に制御する方法について述べ、それを評価する。ここで考えているプロセスの処理対象は、ファクト・ベース(またはルール・ベース)である。また、並列制御の要素として、プロセッサの数、与えられたゴールに対する検索処理コマンドの種別、および、その対象オブジェクトの量などがある。ここで述べる方式の評価の重点は、処理粒度と並列度の相関である。

ここで用いたGHCは、ガード部に記述できる述語をシステム述語だけに制限したFlat GHC (FGHC)である。また、評価に用いた処理系は、ゴールの処理される順序によってサイクル数が変化しないものである。

2. 知識ベース・システム

知識ベース・システムの最初の段階として、ホーン節の集合を知識とし、これを蓄え管理する演繹データベース・モデル(図1)を考える。これは、知識ベース・システム内で、外部データベース(ファクトの集合:以下、EDB)および内部データベース(EDBの使い方に関するルールの集合[view定義]:以下、IDB)を管理し、ユーザまたは応用プログラムは、知識ベース・システムに対してホーン節でIDBおよびEDBに対して問い合わせを行う三階層結合モデル[宮崎 85]である。また、データベース管理層は、検索プロセッサ並列制御部、複数の検索プロセッサ、EDBの3つの部分からなる(図2)。

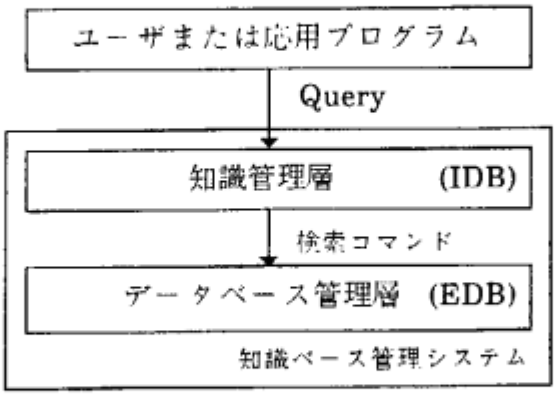


図1 三階層結合の知識ベース・モデル

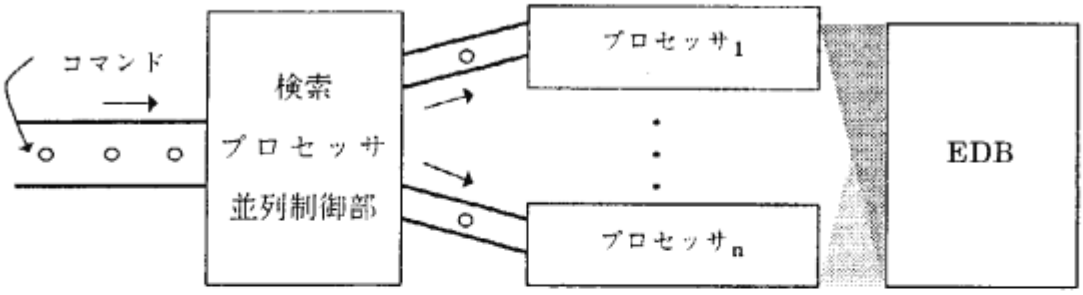


図2 データベース管理層

以下では、複数ある関係演算プロセスをどのように並列制御すればよいかについて述べる。

3. プロセスの並列制御

プロセスを並列に制御するために関係するパラメータは、プロセッサの数、コマンドの種別、対象データ量などである[Itoh 87]。本節では、プロセッサの数を考慮した並列制御について述べ、次節では、コマンドの種別と処理粒度から並列性について述べる。

データベース管理層において、プロセスを並列に制御するために3つの戦略を用いた。以下に出現する記号n, iはそれぞれ、検索プロセッサの総数、ある時点における空きプロセッサ数を示している。

(1)データ非分割法(以下、戦略1)
これは、知識管理層からコマンドを1つ受け取ると、空きプロセッサを1台見つけ、そのプロセッサにコマンドを割り付ける方法である。

(2)データ可変分割法(以下、戦略2)
これは、知識管理層からコマンドを1つ受け取ると、その時点での空きプロセッサをすべて(i台)見つけ、対象データをi分割することによって、i個のサブ・コマンドを生成し、各空きプロセッサにサブ・コマンドを割り付ける方法である。

(3)データ固定分割法(以下、戦略3)
これは、知識管理層からコマンドを1つ受け取ると、対象データを一定数(例えばn)で分割することによって、n個のサブ・コマンドを生成し、戦略1のように空きプロセッサを1台見つけては、サブ・コマンドを1つずつ割り付ける方法である。

各プロセッサの状態は仕事中(busy)または空き(free)のいずれかをとる。戦略1と3はプロセッサの状態が空きであることだけがわかればよく、仕事中のプロセッサを知る必要がない。このため並列制御部は、プロセッサからの要求で駆動させることができる。

戦略2は、プロセッサの状態が仕事中か空いているかの両方を知る必要があるため、制御部内で状態を管理し、空きプロセッサに仕事を割り当てていくようにする。図3は、FGHCで記述したプログラムで、プロセッサの状態を保持し、そのプロセッサの状態問合せにたいして答える述語'RP'と述語responseである。述語'RP'の引数はそれぞれ、プロセッサ番号、コマンド列である。述語

```
'RP'(N,[term  |Cs]) :- true | Command=[].
'RP'(N,[ins(C)|Cs]) :- true |
    C=free, 'RP'(N,Command).
'RP'(N,[cmd(C)|Cs]) :- true |
    sendToRP(N,C,Res),
    response(Res,Cs,NCs), 'RP'(N,NCs).

response(end,Cs         ,NCs) :- true |
    Cs=NCs.
response(R  ,[ins(C)|Cs],NCs) :- true |
    C=busy, response(R,Cs,NCs).
```

図3 プロセッサの状態問合せに答える述語

sendToRPはプロセッサへのコマンド送信で、処理が終了すると第三引数にendがインスタンシェートされる。プロセッサが仕事をしている時に状態問合せコマンド(ins(C))が来ると述語responseによってbusyを返す。

図4は、プロセッサ数を1から10台まで変化させて、処理に100リダクション、100サイクル必要なコマンドを10個処理したときのリダクション数、サスペンション数、サイクル数を示したものである。すべてのコマンドを処理するために最小限必要なリダクション数は1000でサイクル数は1000/プロセッサ数である。これと実際に必要となった数との差がプロセッサの並列制御に使われた部分である。また、図の中で戦略1dと示したものは、プロセッサからの要求とコマ

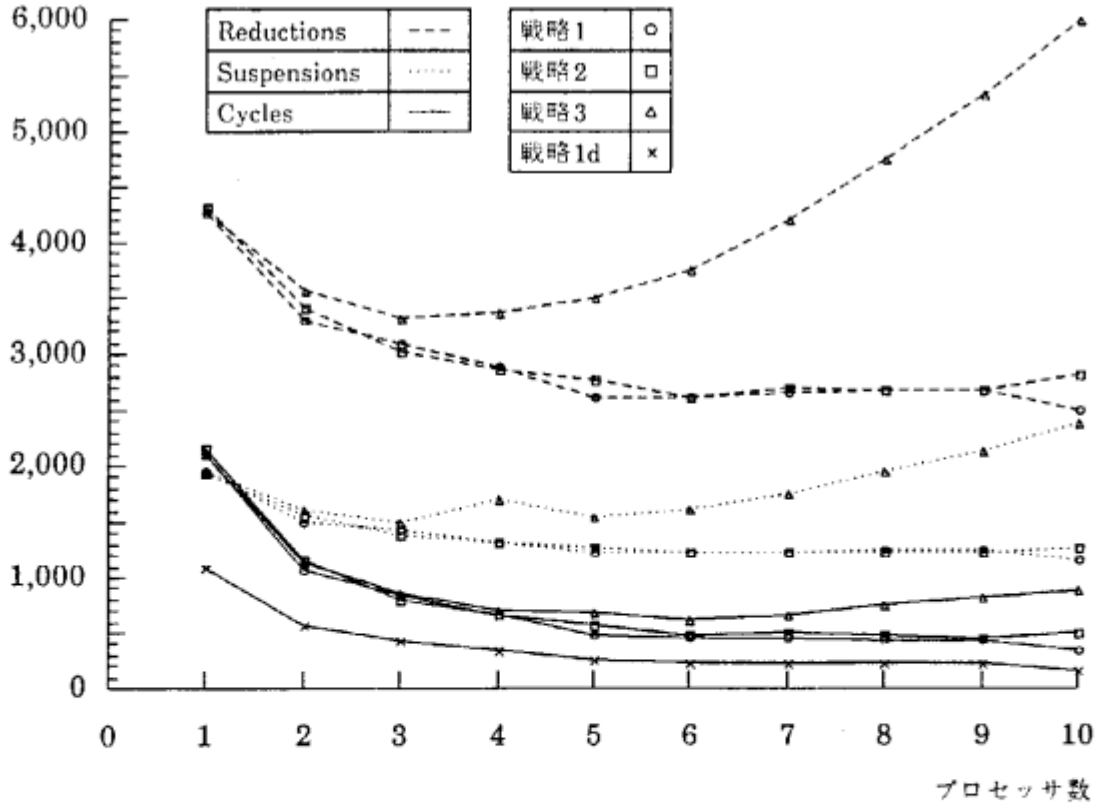


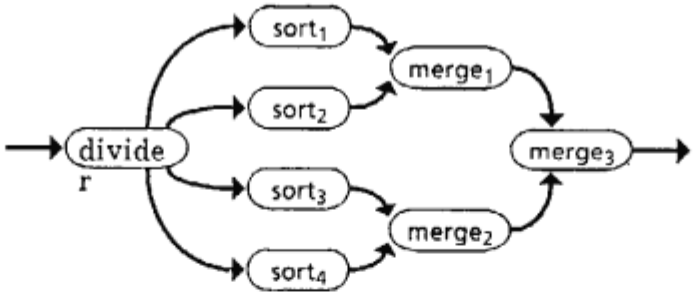
図4 各戦略のReductions,Suspensions,Cycles数

ンドの受付によって制御部を駆動するようにインプリメントしたときの結果である。他はコマンドを受付けた後、定期的にプロセッサの状態をポーリングするようにインプリメントしたときの結果である。

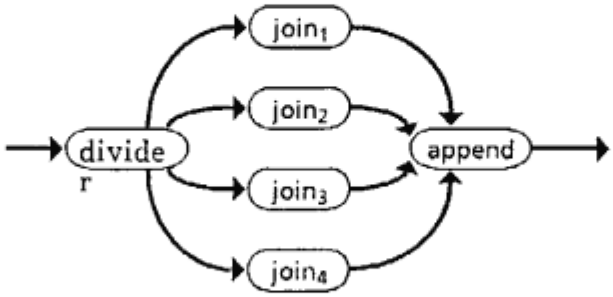
4. 処理粒度と並列性

戦略2、3では、データを分割することによって対象データの粒度を小さくし、これによって処理粒度を細分することにより、プロセッサをきめ細かく稼働させその空きを少なくして、並列性を高めようとしている。関係演算処理において、大きくウエイトを占めるものにソートやジョインがある。以下ではソートとジョイン演算処理を中心にして述べる。

ソート・コマンドをn分割して処理する場合には、n個のソート・プロセスと$\log_2 n$段n(またはn-1)個のマージ・プロセスが必要となる。つまり、ソート・プロセスは処理粒度を小さくすればするほど処理は速くなるが、マージ・プロセスのコストが増加してくる。図5はソートとジョインを4分割処理したときに必要なプロセスとデータの流れを示している。各プロセス間はストリームによってつながれており、パイプライン的にデータを流している。(a)は、$sort_1$と$sort_2$の結果を$merge_1$に、$sort_3$と$sort_4$の結果を$merge_2$に入力として与え、$merge_1$と$merge_2$の出力を$merge_3$の入力としてデータを流して行く。このように行えば、3つのマージ・プロセスを並列に動かすことができ、マージ・プロセスの処理時間(サイクル数)は分割数によって変化せず、データ量に比例する。しかしながら、分割数を大きくするとマージ間の通信コストを無視できなくなってくる。あとの評価では、通信コストを考慮していない。なお、ソート・コマンドに出現するマージ・プロセスは、整列された2つのストリームをデータの大小関係を考慮して併合するものである。



(a)ソート・コマンド



(b)ジョイン・コマンド

図5 コマンドのデータ分割処理

```
make_join(Div,Op,Data,Result,C1-Cn) :-true |
  data_division(Div,Data,DD),
        div_join(DD,Op,Result-[],C1-Cn).
div_join([F|R],Op,R1-Rn,C1-Cn) :- true |
    C1=[cmd(join(Op,F,R1-R2))|C2],
    div_join(R,Op,R2-Rn,C2-Cn).
div_join([]   ,Op,R1-Rn,C1-Cn) :- true |
    R1=Rn, C1=Cn.
```

図6 差分リストによる結果の収集

ジョイン・コマンドやセレクト・コマンドの場合は、処理粒度を小さくする程、各々のプロセスを並列に動かすことができ、処理時間(サイクル数)が短縮される。なお、値を部分的にもつことができる論理変数の性質を利用して、図6に示すように、分割時に結果の部分を差分リストで連結しておけばアペンド・プロセスは必要がない。

5. コマンド分割の評価

図7は、コマンドの対象データ量を分割して処理したときの各サイクルにおけるリダクション数を示したものである。表1は、処理にかかった総リダクション数、サスペンション数、サイクル数、そして実行時間を示している。ソート処理およびセレクト処理は200個のデータを対象とした結果であり、ジョイン処理は25個と20個をデータをジョインした結果である。

(a)はソートの結果である。ソート・コマンドを分割処理する場合に、マージ・プロセスがデータ長だけ必要であるため、分割処理を行った方が処理サイクルが多くかかている。しかしながら、総リダクション数はほぼ同じで、実行時間は分割するほうが短い。これは、マージ・プロセスのほうがソート・プロセスにくらべて軽い処理であることを示している。現在は、逐次マシン上の処理系を使用しているため、すべてのゴールのリダクションは同じ重さで実行されると仮定している。このため、分割するほうがしないものに比べて悪い結果(サイクル数)となっている。しかし、実行時間からみて実際の並列計算機の上では分割ソートの方が良くなると推測される。

(b)はジョインの結果である。ジョイン・コマンドは、は両方のデータの組み合わせだけ対を生成し、それぞれの対を並列に実行してジョインをとる。このため元々並列度をもっている。分割処理することによって、データから対を作成することを早め、それによってより高い並列性を引き出すことができる。実験では、1リダクションで分割された全てのジョイン・プロセスに1つずつデータを流すようなdividerを用いた。

(c)はセレクトの結果である。セレクト・コマンドは、ソートなどの処理にくらべて手間の軽い処理であるが、データ長だけ処理サイクルが必要である。このため、データを分割すればするほど並列性がでてくる。実験では、1リダクションで分割された全てのセレクト・プロセスに1つずつデータを流すようなdividerを用いた。

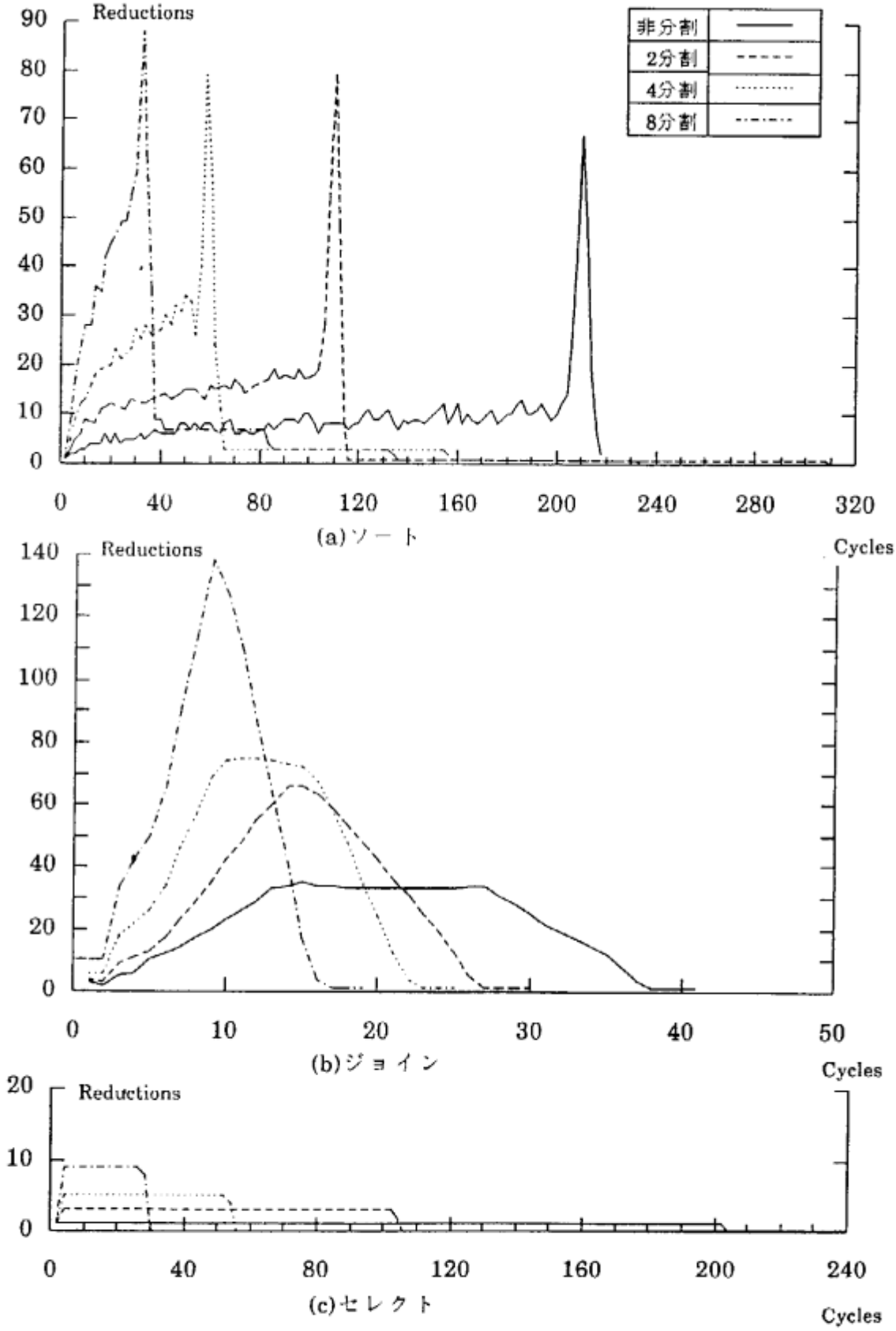


図7 コマンドの分割処理における各サイクルのリダクション数

表1 コマンドの分割処理の結果

(a)ソートの分割処理

分割数	Reductions	Suspensions	Cycles	実行時間 (sec)
1	2038	1580	218	13.663
2	2050	1349	308	7.987
4	2017	1075	257	4.658
8	2027	893	236	2.960

(b)ジョインの分割処理

分割数	Reductions	Suspensions	Cycles	実行時間 (sec)
1	854	253	41	1.120
2	876	255	30	1.043
4	975	260	25	1.108
8	1005	268	19	1.065

(c)セレクトの分割処理

分割数	Reductions	Suspensions	Cycles	実行時間 (msec)
1	202	1	202	345
2	304	3	103	326
4	256	5	53	269
8	235	9	28	252

6. おわりに

ここでは、並列論理型言語GHCによって複数の検索演算処理プロセスを並列に制御する方式として、データ非分割法、データ可変分割法、データ固定分割法の3つの戦略を示し、それぞれをGHCでインプリメントし、簡単なデータで評価をおこなった。また、コマンドの対象データ量を分割することによる並列性や処理時間(サイクル)について、ソート、ジョイン、セレクトなどの代表的なコマンドで示した。

ここではデータベース管理層に重点をおいて述べたが、知識ベースシステム

の他の部分も並列論理型言語によって構築中である。たとえば、知識管理層における知識コンパイル処理の一部であるホーン節変換[Miyazaki 86]もGHCで記述している[Itoh 86]。

[謝辞] 本研究の機会を与えて下さったICOT研究所渕一博所長および有益な討論を頂いたICOT第1研究室、第3研究室の各氏に深謝いたします。

参考文献

[Ueda 86] Ueda, K., "Guarded Horn Clauses", Logic Programming'85, Wada, E. (ed.), Lecture Notes in Computer Science 221, Springer-Verlag, pp. 168-179, 1986.

[宮崎 85] 宮崎、阿比留 他 、KBMS PHI (2) 情報処理学会第32回全国大会論文集、1985.

[Itoh 87] Itoh, H., Abe, M. et al., "Parallel Control Techniques for Dedicated Relational Database Engines", Proc. of Data Engineering' 87, 1987.

[Itoh 86] Itoh, H., Takewaki, T. et al., "A Deductive Database System Written in Guarded Horn Clauses", ICOT TR-214, 1986.

[Miyazaki 86] Miyazaki, N., Yokota. H. et al., "Compiling Horn Clause Queries in Deductive Databases : A Horn Clause Transformation Approach", ICOT TR-183, 1986.