プログラミング－言語・基礎・実践－ 13－14
（１９９３．８．２０）

データ並列論理型言語とその応用

松田　秀雄，金田　悠紀夫
神戸大学工学部情報知能工学科

本稿では，データ並列方式に基づいて，静的に分割可能なデータ集合を複数のプロセスに分配して並列に処理するデータ並列論理型言語を提案し，その並列実行方式，および処理系の実現手法について述べる．データ並列はコントロール並列と比べて，制御が単純であるため理解しやすく，デバッグも比較的容易であると考えられる．並列実行方式では，データ分割並列と節分割並列の２つの方式を示し，得られる解の順番を制御するためプライオリティ制御を導入する．応用としては，分子系統樹推定問題を取り上げ，本方式での並列化手法を示す．

A Data-Parallel Logic Programming Language and Its Application

Hideo MATSUDA and Yukio KANEDA,
Department of Computer and Systems Engineering,
Faculty of Engineering, Kobe University

In this paper we propose a data-parallel logic programming language based on data-parallel approach. A large amount of input data or database facts are partitioned in advance then a number of processes execute Prolog programs in parallel with a part of data or facts. Whole solutions can be obtained by combining partial solutions of each process. In order to sort the order of solutions, we introduce a priority control mechanism. The implementation method of its processing system is discussed. Phylogenetic inference is raised as an application of the language.

1 はじめに

数1000台以上のプロセッサを結合する超並列計算機の開発が各所で進められている．これらの計算機が持つ力を最大限に発揮するためには，解くべき問題に応じた並列性を可能な限り抽出することが必要となる．

論理型言語による並列処理は，これまでに論理式に内在する並列性を取り出すAND並列・OR並列[1]，共有変数を介して明示的に同期・通信を行なう並行論理型言語[2]などが提案されている．これらは，いずれも実行制御の並列化であるコントロール並列方式であり，探索問題のように実行中に探索空間が動的に変化するような応用に向いている．

しかし，超並列計算機の応用分野では，静的に分割可能な膨大な量のデータを処理する問題も多く，そのような応用ではデータ分割による並列化であるデータ並列方式が有効であると考えられる．

そこで，本研究では，データ並列方式により並列処理が記述できるデータ並列論理型言語を提案し，その処理系の実現方式について述べる．また，データ並列論理型言語の応用としては，ゲノム情報処理の一分野である最尤法による分子系統樹推定問題を取り上げ，これを並列に求める手法を示す．

2 並列実行方式

2.1 並列実行モデル

本稿で述べるデータ並列論理型言語は，分散メモリ型の並列計算機上での実行を前提としている．すなわち，各プロセッサのメモリ空間は独立しており，相互のデータ交換は結合ネットワークを通したメッセージ通信のみとする．

並列実行は，コントローラ，プロセス，タスクの3つの概念に基づいて行なわれる(図1参照)．コントローラは処理系全体でただ一つ存在し，同期・通信制御，プロセスへのデータ分配または放送，プロセスからのデータ回収などを担当する．プロセスは基本的にSPMD(Single Program, Multiple Data Stream)で動作し，静的に分割されたデータまたはプログラムを受け取って実行する．

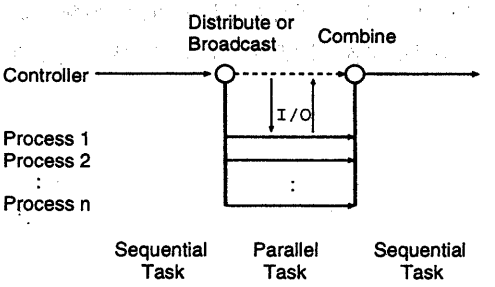

図1: 並列実行モデル

タスクはプログラム中での同期点から同期点までの実行であり，逐次タスクと並列タスクとがある．逐次タスクは，コントローラにより逐次的に処理され，主に前の並列タスクで回収したデータの処理，次の並列タスクで分配すべきデータの準備を行う．並列タスクは，プロセスにより並列に処理されるタスクであり，同一のプログラムを異なる入力データに対して並列に処理するデータ分割並列タスクと，同一の述語をヘッドに持つ節集合を静的に分割してこれらを並列に処理する節分割並列タスクの2種類がある．

並列タスクで生成されるプロセスの個数は，今のところ単純に，データ分割並列タスクでは

入力データの数，節分割並列タスクでは節集合内の節の数としている．このため，実際の実行ではプロセスの数はプロセッサの数を上回ることが一般的と考えられるので，複数のプロセスをプロセッサに割当てて並列タスクを処理する．

データ並列論理型言語の言語仕様は，Prologをベースにしている．並列タスクの実行は，Prologの bagof 内部の実行と類似した以下のような手順で行われる．

(1) 並列タスクの開始点にはデータ分配命令(データ分割並列タスクの場合) またはデータ放送命令(節分割並列タスクの場合)があり，コントローラにより各プロセスにデータが分配または放送される．

(2) その後，コントローラからの指令により，各プロセスは並列タスクを実行する．このとき，入力データまたは処理すべき節がプロセスにより異なるので各プロセスは最初のゴールが同一でもそれぞれ別々の動作をする．

(3) 並列タスクの終了点にはデータ回収命令があり，ゴールの実行に成功し，終了点まで到達したプロセスはデータ回収命令内で指定された変数の束縛情報を解としてコントローラに報告し，バックトラックにより別解の探索に移る．

(4) ゴールの実行に失敗したプロセスはもう解のないことをコントローラに報告し，次のタスクの実行指令がコントローラから来るのを待つ．

Prologには組込み述語によりファイル入出力やプログラムの書き換えなどの副作用を伴う操作があるが，分散メモリ型並列計算機ではメモリやI/Oなどの資源はプロセッサごとに独立している場合が多く，複数のプロセスが任意のタイミングでこれらの操作をすることを許すには，何らかの一貫性制御が必要となる．

本稿で述べる並列実行方式では，これらの一貫性制御は全てコントローラで行うことにしている．プロセスはファイル入出力などの操作をコントローラに要求し，コントローラは処理結果をプロセスに返す．従って，順番が問題になるような操作でもコントローラが要求を解析することにより順序づけを行なって処理することができる．

このとき，コントローラはあるプロセスから要求のあった操作の結果を他のプロセスに通知することはしない．プログラム書換えでは，その結果を他のプロセスにも知らせる必要があるため，本方式ではプロセスがプログラム書換えを直接実行またはコントローラに要求して間接的に実行することはできないことにしている．

2.2 並列実行命令

並列タスクの開始・終了は，プログラム中で以下のような組込み述語により指示される．

- `distribute(List,ProcVar)`
  データ分配命令：データ列 List (リスト形式)の各要素をプロセスに分配する．分配されたデータはプロセスからは変数 ProcVar を通して参照される．

- `broadcast(Data,ProcVar)`
  データ放送命令: コントローラが持つデータ Data をプロセスに放送する．放送されたデータはプロセスからは変数 ProcVar を通して参照される．

- `combine(ProcVar,List)`
  データ回収命令: プロセスの持つデータを変数 ProcVar (プロセスごとに値が異なる)

を通してリスト List にまとめる．まとめられたデータ列はコントローラから参照される．

この他に，節分割並列タスクでは，節をプロセスに分配する必要があるが，それは以下のような節分配宣言をプログラム中に書いておくことにより，どの節を並列タスクで処理するかがコンパイル時に静的に決定される．

- `:- distribute Pred/Arity.`
  節分配宣言: Arity 個の引数を持つ述語 Pred をヘッドに持つ節集合のプロセスへの分配を宣言する．

これらの命令を使った簡単なプログラム例を以下に示す．まず，データ分割並列タスクのプログラムは Program 1 のようになる．

Program 1

```
goal(Ans) :-
    distribute([a,b,c],In),
    p(In,Out),
    combine(Out,Ans).

p(a,1).
p(a,2).
p(b,3).
p(c,4).
```

Program 1 では，データ分配命令によりアトム a，b，c がそれぞれ別々のプロセスに分配され，並列タスクが開始される．これにより，`p(a,Out)`，`p(b,Out)`，`p(c,Out)`の 3 つのゴールが並列に実行され，結果として変数 Ans に 1,2,3,4 を要素に持つリスト(但し，リストの要素の順番は不定)が入る．

これに対して節分割並列タスクのプログラムは例えば Program 2 のようになる．

Program 2

```
:- distribute q/2.

goal(Ans) :- broadcast(a,In),
             q(In,Out),
             combine(Out,Ans).

q(a,1).
q(a,2).
q(a,3).
q(a,4).
```

Program 2 では，節分配宣言により述語 q をヘッドに持つ節がプロセスに分配され，データ放送命令によりアトム a が全プロセスに放送されて，並列タスクが開始される．これにより，各プロセスは自分に分配された節をもとに，同じゴール `q(a,Out)` を実行し，結果として変数 Ans に 1,2,3,4 を要素に持つリスト(但し，リストの要素の順番は不定)が入る．

2.3 プライオリティ制御

前節の並列実行命令では，並列に実行されるプロセスの間での制御は特に行なっていないため，得られる解の順番は不定であった．しかし，応用によっては解に対して評価関数が設定され，その評価関数を最適にするような解を求めたい場合もある．

このような応用に対応するため，筆者等が OR 並列の制御に導入したプライオリティ制御機構[3]をデータ並列論理型言語にも導入する．プライオリティは次の 3 つの組込み述語により設定される．

`setPriority`(プライオリティ値)
`upPriority`(プライオリティ増加値)
`downPriority`(プライオリティ減少値)

`setPriority` は引数の値(整数)を新しいプライオリティとして設定する．`upPriority`，`down-`

Priorityはそれぞれ現在のプライオリティに引数の値を加えるかまたは引いたものを新しいプライオリティとして設定する.

以上の述語により設定されたプライオリティを使って得られる解の順番を決めるため, 以下のデータ回収命令を新たに追加する.

- combinePriority(ProcVar,List)
  プライオリティ付きデータ回収命令: プライオリティの高いゴールから順に変数ProcVarの値をリストListにまとめる.

このプライオリティ付きデータ回収命令を使ったプログラム例を図2に示す. 図2は図3で表されるグラフの最短経路を求めるプログラムであり, 述語arcにより与えられたグラフを探索して, 開始点から到達可能な点までの経路情報をデータ分割並列タスクにより次々に生成する.

このとき, コストに負号をつけたものをプライオリティとしているので, コストの小さいものから経路情報がコントローラに回収される. このため, データ列の先頭に目標点までの経路情報がくれば, それが目標点までの最もコストの小さい経路, すなわち最短経路を示すものとなる.

図2のプログラムで?- go(a,d,Path,Cost). というゴールを実行すると,

```
Path = [a,c,e,d],
Cost = 17
```

という最短経路とそのときのコストが表示される.

ここで, 図2のプログラムは, グラフの各アークを逆方向にたどることを許しているため, 生成される経路は無限にあるが, searchPathの最初の節で目標点までの最短経路が求められたと

```
go(From,To,Path,Cost) :-
    searchPath([[From,To,[From],0]],
               RevPath,Cost),
    reverse(RevPath,Path).


searchPath([[To,To,OptPath,OptCost]|_],
           OptPath,OptCost) :- !.
searchPath(PathRecs,OptPath,OptCost) :-
    distribute(PathRecs,ProcPathRec),
    search1(ProcPathRec,
            [From,To,Path,Cost]),
    Priority is -Cost,
    setPriority(Priority),
    combinePriority([From,To,Path,Cost],
                    ViaRecs),
    searchPath(ViaRecs,OptPath,OptCost).


search1([To,To,Path,Cost],
        [To,To,Path,Cost]) :- !.
search1([From,To,Path,Cost1],
        [Via,To,[Via|Path],Cost]) :-
    path(From,Via,ViaCost),
    Cost is Cost1 + ViaCost.


path(From,To,Cost) :- arc(From,To,Cost).
path(From,To,Cost) :- arc(To,From,Cost).


arc(a,b, 5).  arc(a,c,10).  arc(b,c,7).
arc(b,d,13).  arc(c,d,10).  arc(c,e,3).
arc(d,e, 4).
```

図2: 最短経路を求めるプログラム

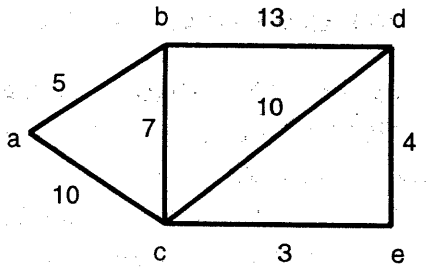

図3: 経路のグラフ表現

きに実行が終了するようにしている．従って，上記の例では，答が一つ出ただけで実行が終了する．

3 応用例

データ並列論理型言語の応用例として，筆者等が文献[3, 4]でOR並列Prologのプライオリティ制御機構の応用として示した分子系統樹推定問題を取り上げる．

分子系統樹とはDNAやRNAの塩基配列など分子レベルでのデータを使用してつくられた生物または遺伝子の系統樹である．推定アルゴリズムとしては最尤法[5]を使用する．

最尤法では，n種の系統樹は以下のように求められる．

(1) 3種のDNA配列から系統樹を作成し，種の数を表す変数kに4を代入する．

(2) $k-1$種の系統樹にk番目の種を加えてk種の系統樹の樹形($2k-5$個ある)を生成する．それぞれについて進化の確率モデルから尤度が最大になるように系統樹の枝長を計算し，k種の系統樹を構成する．

(3) $k<n$の間，kを1ずつ増やしながら(2)を繰り返す．$k=n$になったとき，尤度が最大の系統樹が最終的な系統樹となる．

ここで樹形とは系統樹の分岐パターンであり，これに枝の長さ(進化に要した時間に対応する)を加えたものが系統樹となる．

文献[3, 4]ではA*アルゴリズムのOR並列実行用に拡張したアルゴリズムを使っている．種の数を増やすにつれて調べるべき系統樹の樹形が組合せ的に増えるので，系統樹の尤度を基にプライオリティを設定して効率的に枝刈りを行なっているが，それでも尤度の近い系統樹が多数生成される場合には並列度が大きくなり過ぎ，メモリ容量の限界から多数の種の系統樹推定には対応が困難となっていた．

本稿では，文献[3, 4]とは異なる方式により系統樹の並列推定を行なう方法を示す．すなわち，最尤法による推定アルゴリズムの(2)において，生成された$2k-5$個ある系統樹の樹形を入力データとして，データ分割並列タスクにより各樹形の枝長および尤度を並列に計算する．得られた尤度の値をもとにプライオリティを設定することにより，データ回収の時点で自動的に尤度最大の系統樹が得られる．

この方法では，種の数を増やしていきながら系統樹を求める過程で，個々の種の数の系統樹を尤度最大のもの一つに限定しているので，最終的に尤度最大の系統樹が得られるとは限らない．そこで，文献[6]で筆者等が行なった系統樹の部分木の再配置(rearrangement)を行ない，これにより尤度が向上するかどうかを調べる(図4参照)．

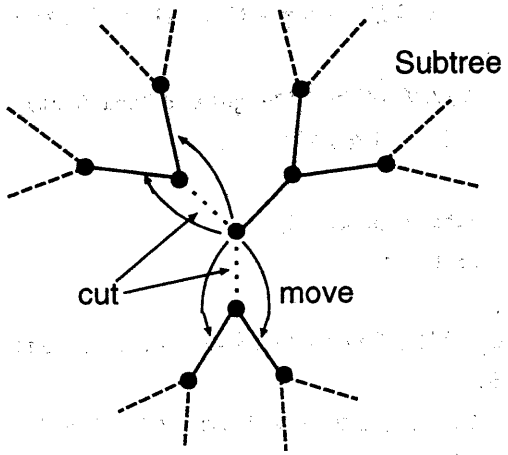

図4: 系統樹の再配置

再配置には，部分木の再配置を隣接した枝への移動だけに限定した局所再配置と，限定せずに系統樹全体に渡る移動を行なう大域再配置の2種類がある．k種の系統樹では，局所再配置では $2k-6$ 個，大域再配置では近似的に $4n^2-26n+38$ 個の樹形が生成される[6]．これらの処理においても，種の数を増やしていく段階と同様に，データ分割並列タスクにより各樹形の枝長および尤度を並列に計算し，尤度最大の系統樹を得ることができる．

4 処理系の実現

データ並列論理型言語の処理系は，筆者等が作成したOR並列Prolog処理系[3]をベースにして作成する．この処理系は，密結合型並列計算機上で稼働しており，Prologプログラムを WAMコード[7]を中間コードとしてCプログラムに変換するコンパイラと，OR並列で実行される複数のゴールをプライオリティ制御によりスケジューリングするPE (Prolog Engine)から成っている．

コンパイラは現在，OR並列宣言によりOR並列実行用のコードを生成するようになっているが，これを修正して節分配宣言を取り扱えるようにする．

また，PEにメッセージ交換によるデータ分配・放送・回収機能を付加して拡張する．これらの機能は，米国アルゴンヌ国立研究所で開発された並列プログラミングツール p4[8, 9]を使うことにより，並列計算機だけでなく，ネットワークでつながれたワークステーション等の上でも動作する移植性に優れたものとする．

PEは各プロセッサごとに1個ずつ置かれ，他のPEとはメッセージ交換のみで通信する．PEのうちの1台がコントローラとなり，他のPEの実行制御，入出力処理等を担当する．残りのPEは並列タスクでのプロセス実行を担当する(図5参照)．

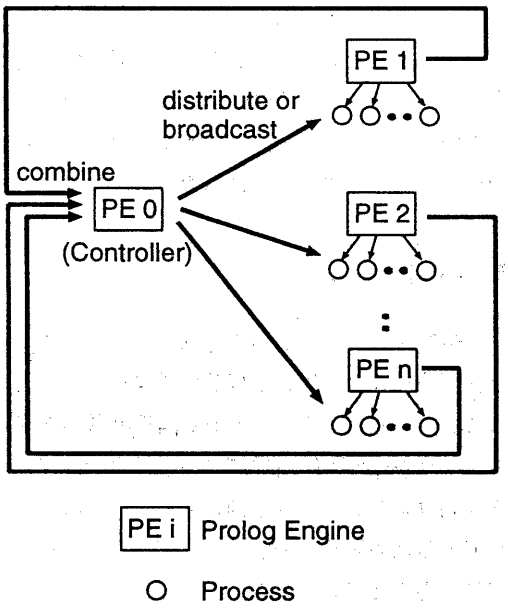

図 5: 処理系の実現

2.1節で述べたように，並列タスクで生成されるプロセスの個数は，今のところ単純に，データ分割並列タスクでは入力データの数，節分割並列タスクでは節集合内の節の数としている．従って，一般的に，プロセッサの数を上回るプロセスが生成されるので，プロセスを担当するPEには並列タスク開始時に，コントローラより複数のプロセスを割り当てる指令がメッセージとして送られる．

各PEはプライオリティ付きのスケジューリングを行なって，複数のプロセスを並行に実行する．各プロセスから解が得られるたびに，データ回収命令によりコントローラへ解を送信する．このとき，データ回収命令が `combinePriority`

であれば解とともにそれを求めたプロセスのプライオリティをもコントローラに送る.

コントローラは, 他のPEからの解を来た順番に回収する. もしその中にプライオリティの情報が含まれていれば, プライオリティの高い順にソートしながら解の回収を行なう.

5 おわりに

本稿では, データ並列方式に基づいて, 静的に分割可能なデータ集合を複数のプロセスに分配して並列に処理するデータ並列論理型言語を提案し, その並列実行方式について述べた. データ並列はコントロール並列と比べて, 制御が単純であるため理解しやすく, デバッグも比較的容易であると考えられる.

また, 筆者等が以前OR並列に実行制御方式として提案したプライオリティ制御がデータ並列論理型言語についても適用できることを示し, 最短経路探索問題でのプログラミングの手法について述べた.

データ並列論理型言語の応用としては, 分子系統樹推定問題を取り上げ, これを並列に求める手法を示した.

処理系の実現については, 以前, 筆者等が作成したOR並列Prolog処理系をベースにメッセージ通信機能を持つよう拡張して実現する方法を示した.

参考文献

1) Conery, J. S.: *Parallel Execution of Logic Programs*, Kluwer Academic Publishers, Massachusetts (1987).

2) Shapiro, E.: The Family of Concurrent Logic Programming Languages, *ACM Comput. Surv.*, Vol.21, No.3, pp.413–510 (1989).

3) 松田秀雄, 鈴鹿重雄, 金田悠紀夫: OR並列Prologにおけるプライオリティ制御機構とその応用, 情報処理学会論文誌, Vol.34, No.4, pp.773–781 (1993).

4) 松田秀雄, 鈴鹿重雄, 金田悠紀夫: プライオリティ制御機構を有するOR並列Prologの探索問題への応用, Proc. JSPP '93, pp.55–62 (1993).

5) Felsenstein, J.: Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach, *J. of Molecular Evolution*, Vol.17, pp.368–376 (1981).

6) Matsuda, H., Olsen, G. J., Hagstrom, R., Overbeek, R. and Kaneda, Y.:, Implementaion of a Parallel Processing System for Inference of Phylogenetic Trees, *Proc. of Pac. Rim Conf. on Commun., Comput. and Signal Processing*, pp.280–283, 1993.

7) Warren, D. H. D.: An Abstract Prolog Instruction Set, SRI Technical Note 309 (1985).

8) Butler, R and Lusk, E.: User's Guide to the p4 Programming System, Tech. Report ANL-92/17, Math. and Comp. Div., Argonne National Laboratory (1992).

9) 松田秀雄: アルゴンヌ国立研究所における並列処理研究, 情報処理, Vol.34, No.6, pp.789–794 (1993).