OR 並列 Prolog におけるプライオリティ制御機構とその応用

松田秀雄[†] 鈴鹿重雄[†] 金田悠紀夫[†]

本論文では Prolog の OR 並列実行におけるプライオリティ制御の方式を提案している．従来の OR 並列 Prolog 処理系では並列に実行されるゴールの数が組合せ的に増大する可能性があり，得られる解の順番についても処理系に依存していた．本方式では並列実行されるゴールにプライオリティを付加し高いプライオリティのものから実行する．これにより実行時の探索空間を不必要に広げずに最適解に近いゴールのみを選択して実行できる．処理系の実現については共有メモリ型並列計算機上での方法を示した．この処理系を使って遺伝子情報処理の一分野である分子系統樹の推定を行ったところ，全解を求めるのに比べて約 1/14 の時間で最適解を求めることができ本方式の有効性が確認された．

A Priority Control Mechanism for OR-Parallel Prolog and Its Application

Hideo Matsuda,[†] Shigeo Suzuka[†] and Yukio Kaneda[†]

We propose a method for performing priority control in Or-parallel Prolog, with an evaluation function specified by the user as the priority. In Or-parallel Prolog systems there is a possibility that the number of goals executed in parallel could increase in combination. By assigning priorities to the goals and executing in order of priority, only goals near the optimum solution are scheduled for running and the number of goals is not unnecessarily expanded. The effectiveness of this method is demonstrated by implementing the processing system on a shared memory multiprocessor machine (Sequent Symmetry) and applying the system to the inference of a molecular phylogenetic tree, which is a field of genetic information processing. To infer a phylogenetic tree of 6 species, the execution speed with the priority control is about 14 times as fast as that without it.

1. はじめに

知識情報処理の応用分野が多様化，大規模化するにつれて，処理速度のさらなる向上が求められている．このためには並列処理技術の導入が不可欠と考えられており活発に研究が進められている．その中でも，特に論理型言語 Prolog による並列処理については，論理式に内在している非決定性を利用した AND 並列・OR 並列[1]を代表とする種々の方式が提案されている．

OR 並列は，実行可能な複数の候補節を並列に呼び出して実行していくもので，探索問題などで特に効果を発揮する．しかし，単純にすべての呼出しを並列に行うと並列度が爆発的に増え，プロセス切替えなどのオーバヘッドの増大により台数効果が抑えられてしまう．これを解決するためには，並列処理の効果が高い部分のみを並列に実行し残りは逐次に実行するなど，プログラムの特性に応じたスケジューリングを行う必要がある．このようなスケジューリングを行っている処理系としては Aurora[2]，Muse[3] などがある．

本論文では OR 並列を制御するためのプライオリティ制御機構を提案する．ある評価関数を最大に（または最小に）するような最適解を求める問題では，この評価関数に基づいてゴールにプライオリティを設定すれば，プライオリティ制御機構により最適解に近いゴールのみが選択されて実行される．これにより並列実行時の並列度の組合せ的な増大を抑制できる．

プライオリティ制御機構を持つ OR 並列 Prolog の応用として分子系統樹 (molecular phylogenetic tree) 推定問題をとり上げた[4]．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．

† 神戸大学工学部情報知能工学科
Department of Computer and Systems Engineering, Faculty of Engineering, Kobe University

以下ではOR並列の制御方式と，その処理系の並列計算機上での実現について述べ，分子系統樹の推定に適用した結果について述べる．

2. プライオリティによる OR 並列の制御

Prolog の実行過程は，ユーザが入力した最初のゴールを根とし，実行の各時点でのゴールを節点とする探索木の展開ととらえることができる (**図 1**)．各節点から下向きに出る枝は，その節点のゴールと単一化可能な候補節を表し，探索木の一番下の葉はすべてのゴールのリダクションに成功，あるいはこれ以上ゴールのリダクションができずに失敗したことを表す．成功した結果得られた変数の束縛値が最初のゴールの解となる．この探索木を，根から幅優先に並列に展開するのが **OR 並列**である．

OR 並列は，与えられた条件を満たす解をすべて求めるような問題で特に有効であるが，ある評価関数を最大に（または最小に）する最適解を求めるようなときには，通常の逐次 Prolog と同様 setof, bagof といった組込み述語により解の集合を求めた後，それらの中から最適解を選択することになる．しかし，探索木が非常に大きく広がるような問題では結果的に無駄な処理を数多く行う可能性が高く，実行時間，メモリ消費ともに膨大なものとなってしまう恐れがある．

OR 並列での効率の良い最適解探索のため，プライオリティによる実行制御を提案する．プライオリティは，概念的には探索木の各枝についてラベルと見なすことができる．全体の実行はスケジューラにより管理され，プライオリティの高い枝から順に選択されて，その下のゴールが実行される．これにより探索木の展開は理想的には常に最適解の方向に向かうことになり，無駄な展開を抑えることができる．

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．

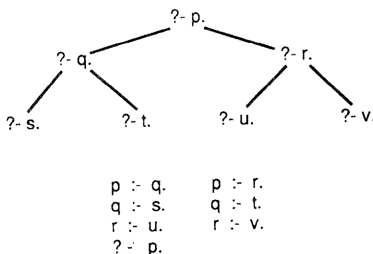


図 1 探索木
Fig. 1 Search tree in execution of a Prolog program.

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, downPriority 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値を新しいプライオリティとして設定する．これらによりプログラムは次のように記述される．

ヘッド :-
評価関数計算，プライオリティ設定，ボディ．

節の呼出しごとにその時点での評価関数の値が計算され，プライオリティが設定される．プライオリティは新たに設定されるまで以前の値を引き継ぐことにしているので，すべての節で評価関数計算とプライオリティ設定を書く必要はなく，書かなかった場合は節を呼び出したゴールのプライオリティがそのままボディゴールのプライオリティとなる．

さらに，プライオリティ制御をしても並列に実行すべきゴールの数が膨大になってしまう場合があるので，OR 並列で実行すべき候補節を並列宣言により指定するようにしている．この宣言がされていない節は逐次 Prolog と同様，バックトラックにより逐次的に呼び出される．OR 並列実行と逐次実行を組み合わせることにより，並列に実行されるゴールの数を減らし，後述するタスク生成・切替えのオーバヘッドを抑えるようにしている．

3. 並列計算機上での実現

3.1 並列計算機の構成

処理系の実現を，Sequent 社製の並列計算機 Symmetry S 81 上で行った．Symmetry は，要素プロセッサとして Intel 80386（クロック 16 MHz）を使用した共有メモリ型の並列計算機である．使用した並列計算機では28台のプロセッサが実装されている．

Symmetry では，すべての要素プロセッサが対等であり，一本のバスにより結ばれている．プロセッサ間の通信はバスに接続された共有メモリを介して行う．バスと共有メモリでの競合を抑えるため各プロセッサには 64 KB のコピーバック方式のキャッシュがつけられている．OS は DYNIX と呼ばれる UNIX をベースにしたもので，プロセスを単位としてプロセッサへの割当てなどの並列実行管理を行い，相互排除のためのロック（専用ハードウェアを備えている），プ

ロセス間の共有メモリ割当てなどの機能を提供する.

3.2 実行モデル

本処理系では, **PE** (**Prolog Engine**) と**タスク**という2つの概念で Prolog プログラムを並列に実行する. PE はプロセッサを抽象化したものであり, 逐次 Prolog 処理系と同様, プログラムを深さ優先で実行するソフトウェア仮想マシンである. PE は DYNIX から見ればユーザプロセスの1つであり, 並列実行に先だってプロセッサ台数以下の個数だけ生成され, 次に述べるタスクを処理していく. DYNIX では実行可能なユーザプロセスの数がプロセッサ台数以下の時には, プロセスとプロセッサとは1対1に対応する.

タスクはゴールの逐次的な実行過程である. 2節で述べた並列宣言されていない節はすべて1つのタスクにより実行される. 並列宣言されている節の実行では節の数に応じて複数のタスクが生成され, プライオリティでソートされた実行可能キュー (Ready Queue) につながれる (図 2).

実行可能キューは共有メモリ上におかれ, 処理系全体で1つだけである. タスクの PE への割当ては, グローバルなスケジューラではなく個々の PE が実行可能キューを見て, そこから先頭のタスクを取り出すことにより行われる. PE は(1)タスクが割り当てられていない, (2)割り当てられたタスクを完了した, (3)割り当てられたタスクより高いプライオリティを持つタスクが新たに実行可能キューにつながれた, のいずれかで, 実行可能キューから新たにタスクを獲得してそれを実行する. (3)の条件は, 各 PE により2節で述べた組込み述語によるプライオリティ設定時および候補節の呼出しごとにチェックされる.

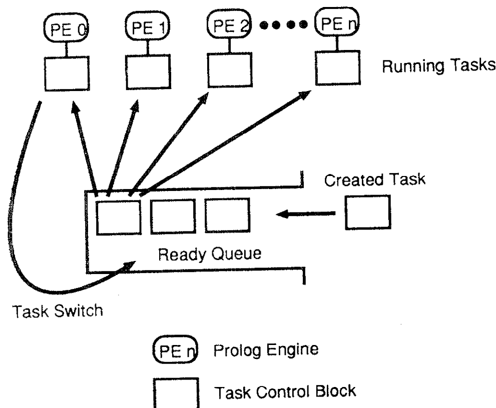


図 2　並列実行モデル
Fig. 2　Parallel execution model with Prolog engines and tasks.

この実行モデルは Aurora[2] で採用されている SRI モデルとほぼ同じであるが, (a)変数束縛が SRI モデルでは束縛アレイであるのに対して本モデルでは分割スタック (後述) である. (b)タスク切替えのタイミングが SRI モデルでは基本的にタスク終了時であるのに本モデルではプライオリティ制御により節の呼出しごとに起こり得る, などの点が異なっている.

3.3 処理系の実現

タスクの実現のため, タスクごとに **TCB** (Task Control Block) を設け実行に必要な情報 (PE のレジスタ退避領域, 子タスクの数など) を持たせている. タスク切替えは前節で述べたタイミングで生じるが, その操作は, (1)実行可能キューの先頭にある TCB を1つ獲得, (2) PE のレジスタの値を現タスク TCB 内のレジスタ退避領域に保存, (3)現タスクの TCB を実行可能キューにつなぐ, (4)獲得した TCB のレジスタ退避領域の値を PE レジスタに設定の順に行われる.

PE は WAM[5] を拡張する形で実現されている. Prolog プログラムは WAM コードを拡張した中間コードにコンパイルされる. 中間コードはエミュレータで実行されるのではなく, 各命令をCの関数とみなしてさらにCコンパイラによりコンパイルされ, PE の起動, タスクのスケジューリングなどを行うランタイム・モジュールとリンクされて直接実行可能なオブジェクトに変換される. 以下に WAM からの拡張点を示す.

para_try 命令　WAM の try 命令を OR 並列用に拡張したもの. 並列宣言された複数の候補節を呼び出す時実行され, (1)並列選択点の作成, (2)候補節の数だけの子タスクの生成と実行可能キューへの登録, (3)現在のタスクの切離し (すべての子タスクが終了するまで実行を中断する) と次のタスクの獲得を行う.

unite 命令　同一の親から生成された複数のタスクを統合する. あるタスクが並列選択点を越えてバックトラックする時, そこでは複数のタスクが生成されたはずなので, 単純にバックトラックすることはできない. そこで, この unite 命令により, (1)中断している親タスクの TCB 中の子タスクの数を減らしていき, (2)自分が最後の子でなければタスクの消滅, 最後であれば並列選択点を除去しさらにその前の選択点までバックトラックする.

3.4 メモリ管理

Gupta と Jayaraman[6] は，OR 並列では，(a)束縛環境の生成，(b)変数アクセス，(c)(新たなタスク生成による)タスク切替え，の3つを同時に定数時間で抑えられるようなメモリ管理方式が存在しないことを証明した．さらに本処理系ではプライオリティ制御を行うので，この3つ以外に，(d)プライオリティの変更によるタスク切替えが生じる．これらのトレードオフをどのように選ぶかによって処理系が性格付けられることになる[7]．

本処理系では，メモリ管理については分割スタックを使ってコピー方式をとっている(**図 3**)．メモリは固定長のブロックに分割され，タスクの実行に必要な領域はこのブロックを最初に必要な個数だけ割り当てることによって確保される．スタックなど実行時に動的に増加する領域については，それがブロックの境界からあふれると，新たなブロックを確保しそこへリンクを張って領域拡張を行う．

コピー方式では，新たなタスクを生成する場合，束縛環境や選択点などの情報を探索木の根からすべてコピーする必要があるので上述の(c)が定数時間内に行えないが，他の3つは定数時間内に行える．本処理系の特徴である(d)が定数時間内に抑えられるようにすることが重要と考え，この方式を選んだ．

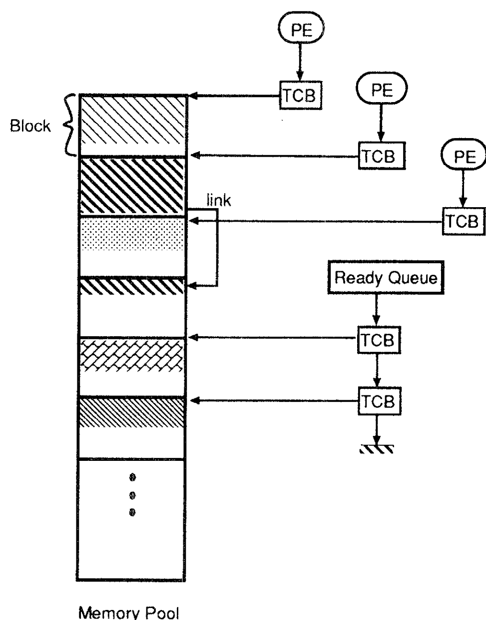


図 3 分割スタックによるメモリ管理
Fig. 3 Memory control using divided stack segments.

4. 分子系統樹の推定

4.1 最尤法による推定

プライオリティ制御付きOR並列の応用として分子系統樹の推定を行った．分子系統樹の推定アルゴリズムとして，距離行列法，節約法，最尤法などが提案されている[4]．距離行列法はすべての種の対についての遺伝的距離を距離行列の形で求め，これらの距離の値の関係から系統樹を推定する方法である．また，節約法は現存種の遺伝データから祖先種の遺伝データを推定し，系統樹全体での進化による変化の数を最小化するように系統樹の樹形(topology)を選ぶ方法である．

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．樹形は根(すべての種の祖先である種を表す)のある有根木と根のない無根木のいずれかに表現され，どちらの表現をとるかは推定アルゴリズムに依存する．なお，節約法では系統樹の枝長を求めるのは難しく，樹形のみを推定するのに使われるのが一般的である．

これらに対して，最尤法は個々の遺伝データを距離行列に変換せずに直接使い，進化の確率モデルに基づく統計的手法により系統樹を推定する方法である．距離行列法，節約法と比べてより推定の精度が高いとされているが，膨大な計算量を必要とするのが欠点である[8]．そこで，本研究では，この最尤法による推定をOR 並列実行により高速化することを試みた．

最尤法では，基本的には以下の3ステップにより系統樹を推定する．

[ステップ1] 与えられた種を組み合わせて，可能な樹形(無根木で表現される)を生成する．

[ステップ2] 樹形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枝長を計算する．

[ステップ3] 尤度の最大値は樹形によって異なるので，それらの中で最も大きい値をとるものを選ぶ(これが最終的な系統樹となる)．

樹形の数は単純に生成すると，種の数 n のとき，

$$\prod_{k=3}^{n}(2k-5)=(2n-5)!/(2^{n-3}(n-3)!) \qquad (1)$$

という膨大な数になる．**図 4** に $n=3$ と $n=4$ の時の樹形を示す．無限木では $n=3$ の樹形はただ1つである．$n=4$ の樹形は $n=3$ の樹形の3本の枝のいずれかに4番目の種を付加したもので3通りある．以下順

に，$n=i$ の樹形は $n=i-1$ の樹形の枝（$2i-5$ 本）に i 番目の種を加えることにより構成される．

式(1)の数の樹形ををすべて調べるのは現実的ではないので，Felsenstein[9] は，最尤法の各ステップをすべての種について一度に行うのではなく，$n=3$ の樹形から始め1つずつ種を増やした中間的な樹形について繰り返し行うアルゴリズムを提案している．ステップ3で尤度が最大の樹形を1つだけ選ぶので，生成される樹形の数は，

$$\sum_{k=3}^{n}(2k-5)=n^2-4n+4 \qquad (2)$$

にまで減らすことができる．

しかし，これは式(1)だけある可能性を式(2)に抑えているため，最終的に必ずしも尤度が最大の樹形が得られるとは限らない（ある時点で生成された中間的な樹形の尤度の大小関係が，それらに次の種を加えると逆転することがありうる）．このため，ステップ3の後，局所再配置（local rearrangement）という樹の各部の接続を変更する処理を，より尤度の大きい樹形が生成される間繰り返すようにしている．1回の局所再配置処理時間と局所再配置の回数はともに n にほぼ比例するため，Felsenstein のアルゴリズムは実際には n^3 に比例した処理時間を要する[10]．

4.2 OR 並列による並列実行

著者らは，OR 並列 Prolog により，樹形を並列に生成し，それらの尤度に基づいてプライオリティを設定する方法により系統樹推定の高速化を試みた（プログラムを付録に示す）．Felsenstein のアルゴリズムと同様，種を加えながら中間的な樹形を生成していくが，ステップ3で1つを残して他の樹形を切り捨てることはしない．プライオリティ制御機構により，尤度の小さい樹形の処理は後回しにされるため，理想的な状況では式(1)よりもはるかに少ない数の樹形を調べるだけで尤度が最大のものが最初に求まるはずである．

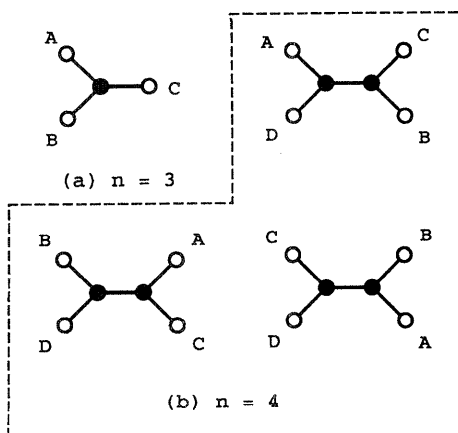


図 4 無根木で表された分子系統樹の樹形
Fig. 4 Topologies as unrooted trees for phylogenetic trees.

プライオリティは尤度の値（正確にはそれの対数を整数化したもの）にある重みを加えたものとしている．これは，尤度は，新たに種を加えると種を加える前と比べて（たとえそれが最適な樹形を持つ樹であっても）必ず小さくなるからである．そこで，最適な樹形を選んでいく限りプライオリティがだいたい一定になるよう，種を加えてできる新しい樹形を生成するごとに単調に増加する重みを加えたものをプライオリティとしている．この重みの値は，現在のところ種ごとの塩基配列データの差分の値から経験的に導き出したものを使用している（差分が大きければ尤度低下が大きくなる）．また，プライオリティの下限値としてある境界値を設け，ゴールのプライオリティの値がこの値より小さくなった時そのゴールを強制的に失敗させている．

尤度に重みを加えたものをプライオリティとし，ある境界値でタスクを強制終了させているので，元の Felsenstein のアルゴリズムと同様，このプログラムで得られた結果も必ずしも大域的に最適とは言えない．Felsenstein は彼のアルゴリズムで得られる解を改善するため，塩基配列データのランダム・サンプリングによる手法[11]を提案しているが，これは著者らの開発した方法にもそのまま適用できると考えられる．

付録で示したプログラムでは，重みと境界値は最初のゴールで次のように設定する．

?- go（[重み1，重み2，…，重み $n-2$]，境界値，変数）．

ここで，重み1，2，…，$n-2$ の値は段階的に種を増やしながら樹形を生成していく過程での重みの値であり，種の数がそれぞれ3，4，…，n の時の樹形に対応している．プログラムではこれらの値のリストが変数 WeightList に代入される．

境界値は変数 Limit に代入され，述語 limitSearch で参照される．最後の引数の変数（プログラムでは変数 Likelihood）にはゴールが成功し系統樹が求められた時に，その系統樹の尤度が代入される．

付録の c_ で始まる組込み述語は，この応用のため

に，C言語で書かれた組込み述語である．樹形の個々の枝の長さや尤度の計算は大量の数値計算を必要とする[12]のでこれらをC言語で記述し，Prolog からは組込み述語として使用している．なお，これらの組込み述語の実現には Felsenstein のアルゴリズムに基づいてイリノイ大学で作成されたプログラム[10]を共有メモリ型並列計算機向けに修正したものを使用している．

プログラムの入力である塩基配列データは，あらかじめファイルに格納されている．今のところファイル名が固定になっているのでプログラム中でその指定はない．並列処理は，あるタスクがステップ1で樹形を生成すると（述語 c_buildNewTip），ステップ2の計算（述語 treeEval）を複数のタスクで並列実行することにより行われる．treeEval は並列宣言されており，ある樹形での計算と残りの樹形を処理する treeEval の再帰呼出しとが並列に実行される．

5. 実行結果

5.1 実行時間と台数効果

本研究で実現した処理系の性能評価を行うため，クィーン問題と系統樹推定問題を実行した．**表1**に PE 1 台での実行時間を，**図5**に台数効果（PE 1 台の実行時間をその PE 台数での実行時間で割った値）を示す．

表 1 PE 1 台の実行時間
Table 1 Execution times of application programs on 1 PE.

	10 Queens	Tree 6 Optim	Tree 6 All	Tree 20 Optim
時間（秒）	170	153	2354	44453

クィーン問題では，プライオリティ制御なしの OR 並列で，文献 13）の MBqueen のプログラムにより 10 クィーンの全解を求めた（10 Queens）．また，系統樹推定問題ではプライオリティ制御により 6 種の古細菌（Archaeabacteria）について最初の樹が見つかるまで（Tree 6 Optim）と可能な樹をすべて生成したとき（Tree 6 All），および 20 種の古細菌について最初の樹が見つかるまで（Tree 20 Optim）のそれぞれの実行時間と台数効果を示した．Tree 6 Optim で求められた樹は，Tree 6 All で生成される 105 個の樹の中で最大の尤度の樹であることを確認しており，全解を求めるのに比べて約 1/14 の時間で最適解を求めることができた．

台数効果は，Tree 6 All でほぼ線形に近い値となっているのに対し，Tree 6 Optim では低い値しか得られず PE 16 台でも 3.6 である．Tree 6 Optim は表1からわかるように 10 Queens と同程度の規模の問題なので，本来ならより大きな台数効果がでるはずである．これとは逆に，Tree 20 Optim では線形以上の大きな台数効果が得られた（26 台のとき 41）．これは単なる問題の規模の増大にともなう台数効果の向上だけでは説明できない大きな値である．

このように種の数によって大きく台数効果が違う原因を調べるため，生成されたタスクの総数を各問題で計測した（**図6**）．この結果，Tree 6 Optim と Tree 20 Optim ではタスク数が PE 台数により大きく変化することがわかった．Tree 6 Optim では PE 台数の

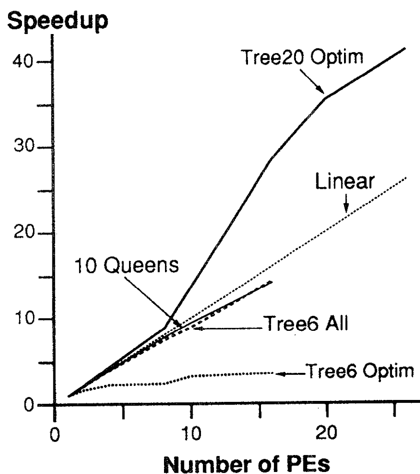


図 5 台数効果
Fig. 5 Speedup of application programs.

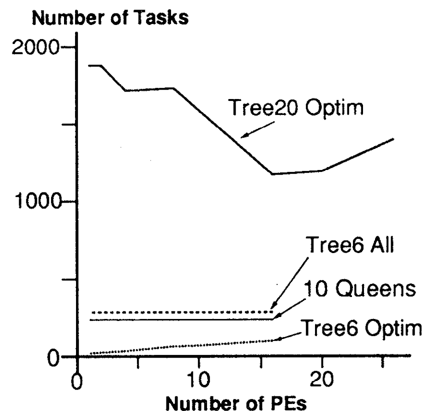


図 6 生成タスク数
Fig. 6 Number of tasks for executing application programs.

増加に対してタスク数が単調に増大している．これは，PE 台数が増えると，本来優先して実行しなければならないプライオリティ最高のタスクの実行とともに，よりプライオリティの低いタスクをも並列に実行してしまうためであると考えられる．すなわち，探索木において PE 台数が1台のときは調べなかった余分な枝まで探索しているわけである．

しかし，Tree 20 Optim では，ある PE 台数までは逆にタスク数が減少している．これは，4.1 節で述べたように，系統樹推定問題では尤度の大小関係の逆転，つまりある時点で生成された樹形の間での尤度の大小関係がそれらに次の種を加えると逆転することがあるためと考えられる．このとき，複数の PE による並列実行では最高より低いプライオリティを持つタスクをも同時に実行するので，次の時点でプライオリティ最高になるタスクを PE 1 台のときよりも早く生成する．各 PE はタスクのプライオリティを候補節の呼出しごとにチェックするので，ある PE がよりプライオリティの高いタスクを生成すると，他の PE は現在実行中のタスクを中断してそのタスクの実行に移る．

複数の樹形の処理を並列に実行すれば，尤度の大小関係の逆転が起こらないときは Tree 6 Optim のときのようにむしろ余分な樹形を調べてしまうことになる．しかし，種の数が増えるにしたがって逆転が起こる可能性は高くなるので，そのときは Tree 20 Optim で示されているように線形以上の台数効果が得られる．ただし，このときもある程度以上 PE 台数が増えると余分な樹形を調べることが多くなり，図6からわかるようにタスク数が増え台数効果が落ちていく．

線形以上の台数効果が得られるときには，一般に PE 1 台でのタスクのスケジューリングに問題があり，複数の PE 台数でのスケジューリングと同じようにタスクを並行に実行すれば，PE 1 台での実行速度を上げることができるはずである．しかし，この問題の場合は，種の数によって台数効果が大きく変化するため，どのスケジューリングが良いかは問題に依存する．さらに種の数が多い時，どのようなスケジューリングをすればよりタスクの数を減らすことができるのかなどの詳細な考察については今後の課題である．

5.2 他の処理系との比較

表1の 10 Queens と Tree 6 All を，Symmetry 上で SICStus Prolog による逐次実行で実行したときの時間はそれぞれ 51 秒，2370 秒であった．系統樹の推定では，SICStus Prolog にはプライオリティ制御機構がないため，付録のプログラムからプライオリティ制御の述語を除いて，バックトラックにより単純にすべての樹を生成した．

10 Queens では SICStus の方が約 3.3 倍速いが，Tree 6 All ではほとんど同じである．これは，系統樹の推定では実行時間の大部分がCで記述された尤度計算で費やされるのと，尤度計算部分を本処理系では組込み述語で実現しているのに対して SICStus では他言語インタフェースによりリンクしているので呼出しのオーバヘッドが大きいためと考えられる．

本処理系の基になったイリノイ大学で開発されたプログラム[10]により Symmetry 上で Tree 6 Optim と Tree 20 Optim の系統樹を推定したときの実行時間は，それぞれ 210 秒，37738 秒であった．このプログラムは 4.1 節の Felsenstein のアルゴリズムをそのまま実現しており，ステップ3で尤度最大の樹を選んだ後，局所再配置を行っている．6種の系統樹では局所再配置により表1の Tree 6 Optim より時間がかかっているが，20種ではステップ3で尤度最大の樹を残して他の樹を捨てているため Tree 20 Optim よりも時間が短くなっている．

6. おわりに

本論文では OR 並列実行をプライオリティにより制御する方式を提案し，その処理系の共有メモリ型並列計算機上での実現について述べた．応用例として，分子系統樹の推定問題をとりあげ，本方式によるプログラミングと実行時間を示した．それによると，種の数が6個と少ないときでも可能なすべての樹を調べるのに比べて，約 1/14 の時間で尤度最大の樹が得られた．

さらに 20 種の系統樹ではいくつかの PE 台数による実行で線形以上の大きな台数効果が得られた．これは，中間的な樹形の生成時に尤度の大小関係の逆転が起こっているためと考えられるが，詳しい考察については今後の課題である．

以上により，本方式の有効性が示された．

謝辞　古細菌の塩基配列データを提供いただいた米国イリノイ大学 Carl Woese 教授，Gary Olsen 博士に感謝いたします．また，本研究で使用した分子系統樹推定プログラムは Gary Olsen 博士の開発したプログラム[10]に基づいて作成されました．同博士に重ねて感謝いたします．同プログラムの並列化について数々の有益な御助言をいただいた米国アルゴンヌ国立研究所 Ross Overbeek 博士に感謝いたします．

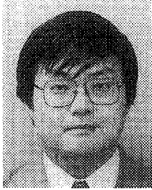
参 考 文 献

1) Conery, J. S.: *Parallel Execution of Logic Programs*, Chap. 3, pp. 35-61, Kluwer Academic Publishers, Norwell, MA (1987).
2) Lusk, E. et al.: The Aurora OR-Parallel Prolog System, *New Generation Computing*, Vol. 8, No. 7, pp. 243-271 (1990).
3) Ali, K. and Karlsson, R.: Full Prolog and Scheduling Or-Parallelism in Muse, *Int'l J. Parallel Programming*, Vol. 19, No. 6, pp. 445-475 (1990).
4) Nei, M.: *Molecular Evolutionary Genetics*, Chap. 11, Columbia University Press (1987). (邦訳 五條堀孝, 斎藤成也 (訳): 分子進化遺伝学, 培風館 (1990))
5) Warren, D. H. D.: An Abstract Prolog Instruction Set, SRI Technical Note 309 (1985).
6) Gupta, G. and Jayaraman, B.: On Criteria for Or-Parallel Execution Models of Logic Programs, *Proc. of NACLP '90*, pp. 737-756 (1990).
7) 市吉伸行: 論理型言語の並列処理方式, 情報処理, Vol. 32, No. 4, pp. 435-449 (1991).
8) Weir, B. S.: *Genetic Data Analysis*, Chap. 8, Sinauer Associates, Inc., Sunderland, MA (1990).
9) Felsenstein, J.: Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach, *J. of Molecular Evolution*, Vol. 17, pp. 368-376 (1981).
10) Olsen, G., Woese, C., Hagstrom, R., Matsuda, H. and Overbeek, R.: Inference of Phylogenetic Trees Using Maximum Likelihood, *Proc. of the First Delta Application Workshop*, pp. 247-262 (1992).
11) Felsenstein, J.: Confidence Limits on Phylogenies: An Approach Using the Bootstrap, *Evolution*, Vol. 39, No. 4, pp. 783-791 (1985).
12) 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, 情報処理学会研究報告, 92-PRG-8, pp. 219-226 (1992).
13) Tick, E.: Performance of Parallel Logic Programming Architectures, ICOT Tech. Rep. TR-421, ICOT, pp. 38-43 (1988).

付録 分子系統樹生成プログラム

```
:- para treeEval/7.

go(WeightList,Limit,Likelihood) :-
  c_init, c_allocTreeArea(NumSpec),
  c_buildFirstTree(NumTips,Tree),
  makeDenovoTree(WeightList,Limit,NumTips,
                 NumSpec,Tree,RestTree),
  c_getLikelihood(ResTree,Likelihood),
  c_showBestTree(ResTree), c_freeTreeArea.

makeDenovoTree([Weight|WeightList],Limit,
        NumTips,NumSpec,Tree,ResTree) :-
  NumTips<NumSpec,
  c_buildNewTip(Tree,TreeBuffer,NumTrees),
  NumTips1 is NumTips+1,
  treeEval(Weight,Limit,NumTips1,TreeBuffer,
         0,NumTrees,NewTree),
  makeDenovoTree(WeightList,Limit,NumTips1,
                 NumSpec,NewTree,ResTree).
makeDenovoTree(_,_,NumTips,NumSpec,ResTree,
              ResTree) :-
  NumTips>=NumSpec.

treeEval(Weight,Limit,NumTips,TreeBuffer,TreeId,
        NumTrees,Tree) :-
  TreeId<NumTrees, c_allocTreeArea(_),
  c_treeEvaluate(TreeBuffer,TreeId,Tree,Likelihood),
  Priority is Likelihood+Weight,
  limitSearch(Limit,Priority), setPriority(Priority).
treeEval(Weight,Limit,NumTips,TreeBuffer,TreeId,
        NumTrees,Tree) :-
  TreeId<NumTrees, TreeId1 is TreeId+1,
  treeEval(Weight,Limit,NumTips,TreeBuffer,TreeId1,
          NumTrees,Tree).

limitSearch(Limit,Priority) :- Priority<Limit, !,
  fail.
limitSearch(_,_).
```

(平成4年6月18日受付)
(平成5年1月18日採録)



松田　秀雄（正会員）

昭和 34 年生．昭和 57 年神戸大学理学部物理学科卒業．昭和 59 年同大学院工学研究科システム工学専攻（修士課程）修了．昭和 62 年同大学院自然科学研究科（博士課程）修了．同年同大学工学部助手となり，現在同大学講師．この間，平成 3 年 4 月より 10 か月間米国アルゴンヌ国立研究所客員研究員．学術博士．論理型言語による並列処理，遺伝子情報処理の研究に従事．IEEE CS, ACM 各会員．

鈴鹿　重雄（正会員）

昭和 42 年生．平成 2 年神戸大学工学部システム工学科卒業．平成 4 年同大学院工学研究科システム工学専攻（修士課程）修了．現在(株)野村総合研究所に勤務し，システムコンサルティング事業に従事．

金田悠紀夫（正会員）

昭和 15 年生．昭和 39 年神戸大学工学部電気工学科卒業．昭和 41 年神戸大学大学院電気工学専攻修士課程修了．昭和 41 年電気試験所（現電総研）入所．電子計算機研究に従事．昭和 51 年神戸大学工学部システム工学科，現教授．工学博士．コンピュータシステムのハードウェア，ソフトウェアの研究に従事．高級言語マシン，並列マシン，AI に興味を持っている．