計算機アーキテクチャ 54－4
（1984. 7. 13）

Ｋ－Ｐｒｏｌｏｇの並列処理方式とその評価

松田秀雄，田村直之（神戸大・自然科学），小畑正貴（岡山理大・理），
金田悠紀夫，前川禎男（神戸大・工）

１．はじめに

述語論理型言語Prologに対する関心が、近年急激に高まってきている。特にPrologの並列処理に関する研究が盛んに行われ、データフロー、マルチプロセッサ、リダクションといった並列実行モデルが提案されている。しかし、これらのモデルに基づくシステムは一般に実現が難しく、シミュレーションによる推定で性能評価をしているのが現状であり、実現例は少ない。

本稿では試作マルチプロセッサシステム上への並列Prolog処理系"K-Prolog"の実装とその評価について報告する。K-Prologについては、既に何回か報告している[1,2] が、その後ハードウェア構成が変化し、より大規模な問題が解けるようになった。また、今までのK-Prologの並列処理方式であるパイプライニング並列（[1,2] ではストリーム並列と呼んでいた）の他に、ＯＲ並列に基づく処理系も作成した。以下、本稿ではこれら２種類の並列処理方式を記述するためのモデルについて述べ、いくつかの簡単な例題プログラムを実行した結果により２つの並列方式の評価を行う。

２．並列実行モデル

本稿で示す並列実行モデルは、マルチプロセッサシステムの各プロセッサが並行プロセス処理を行うものである。各プロセスはメッセージの送受により通信を行いながら、解くべきPrologプログラムを並列に実行する。これらのプロセス間には親子関係が存在し実行の過程で木構造状に連なっていく。

本モデルでは、プロセスの種類としてＯＲプロセス、ＡＮＤプロセス、ＥＯＲ（Explicit OR)プロセス、ＮＵＬＬプロセスの４種類を設けている。このうちＯＲプロセスとＡＮＤプロセスとは、それぞれConeryら[3] のＡＮＤ－ＯＲモデルにおけるＯＲプロセス、ＡＮＤプロセスと同じもので、前者はゴール文中の各述語を受け持ち、後者はそれらの述語がＡＮＤ記号により結ばれた項を受け持って実行する。ＥＯＲプロセスとは、Edinburgh 版Prologなどに備わっている明示的なＯＲの記号（Edinburgh 版と同様 '；' で表す）で結ばれた項の実行を受け持ち、ＮＵＬＬプロセスはPrologの実行過程で単一化により生成される空節の実行を受け持つ。

プロセス間で通信されるメッセージとは、単一化の結果とそれにより生じる変数束縛情報であり、子プロセスから親プロセスへ送りだされる。メッセージは子プロセスが受け持った論理式を真にする代入を含んでおり、それゆえ子プロセスの受け持った論理式の解と考えることができる。

メッセージには、success とfailの２種類がある。success は解すなわち変数束縛情報であり、failは解がなくなったことを示す。メッセージの伝達には各プロセス中に設けられたFIFOのメッセージバッファを用いる。子プロセスは自分のバッファにメッセージを貯え、親プロセスはその子プロセスのバッファにメッセージが入るのを待ってそれを取り出すという形で通信が行われる。

３．パイプライニング並列

パイプライニング並列とは筆者等の提案[4] した並列処理方式で、後戻り処理の時に必要となる別解をパイプライン的にあらかじめ求めておくものである。

このパイプラインの動作は、前で述べたプロセスのうちＡＮＤプロセスの実行で行われる。例えば、Ｐ＆Ｑ（Ｐ、Ｑは適当な項、＆はＡＮＤを示す）を受け持つＡＮＤプロセスでは以下のように実行が行われる（図１）。

① Ｐ＆Ｑの解を求めるＡＮＤプロセスは、まず子プロセスとしてＰの解を求めるプロセスを生成し、そこから最初の解$\sigma 1$ を受け取る。

② 次に$\sigma 1$ に含まれている変数束縛情報によりQ$\sigma 1$ の解を求めるプロセスを子プロセスとして生成し、そこから解$\tau 11$, $\tau 12$, …を順次受け取る。そしてQ$\sigma 1$ の解が得られる度にP＆Qの親プロセスに$\sigma 1 \circ \tau 11$, $\sigma 1 \circ \tau 12$, …（oは合成を表す）を送る。このときPのプロセスは次の解$\sigma 2$, $\sigma 3$,…を求めており、順次自分のバッファに貯えていく。

③ Q$\sigma 1$ のプロセスが失敗すれば（Q$\sigma 1$ のプロセスの解がなくなりfailメッセージが送られてくれば）、Q$\sigma 1$ のプロセスを消去し、Pのプロセスから次の解$\sigma 2$ を受け取る。

④ $\sigma 2$ についても②と同様の手順を繰り返す。以下$\sigma 3$, $\sigma 4$,…についても同じことを行う。

⑤ Pのプロセスが失敗すればそのプロセスを消去し、P＆Qのプロセスが失敗したことを親プロセスに伝える（failメッセージを送る）。

パイプライニング並列では、ANDプロセス以外のプロセスは全て逐次実行の場合と同じ動作をする。例えばP；Q（；はORを示す）の解を求めるEORプロセスの実行は次のようになる（図2）。

① P；Qの解を求めるEORプロセスは、まず子プロセスとしてPの解を求めるプロセスを生成し、そこから解$\sigma 1$, $\sigma 2$,…を受け取る。受け取った解は全てそのままP；Qのプロセスの親プロセスに送られる。

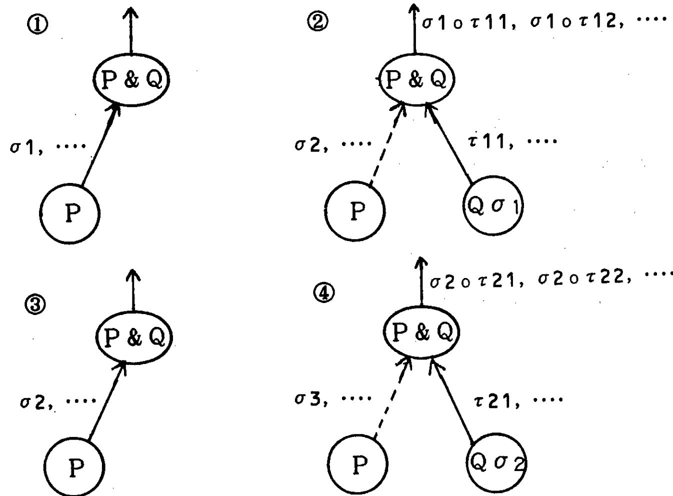


図1 パイプライニング並列のANDプロセスの実行

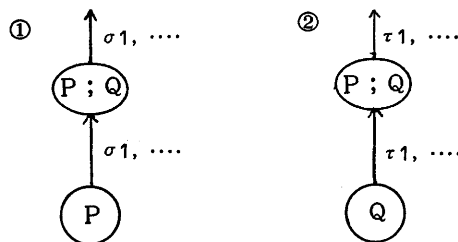


図2 パイプライニング並列のEORプロセスの実行

② Ｐのプロセスが失敗すれば、そのプロセスを消去する。そして、子プロセスとしてＱの解を求めるプロセスを生成し、そこからτ1,τ2,…を受け取る。受け取った解はそのままＰ；Ｑの親プロセスに送られる。

③ Ｑのプロセスが失敗すればそのプロセスを消去し、Ｐ；Ｑのプロセスが失敗したことを親プロセスに伝える。

パイプライニング並列は逐次型に近い並列処理方式で解の得られる順番が逐次実行の場合と全く同じになるという特徴がある（但し入出力やassert、retract によるプログラムの書き換えなど副作用のある場合には必ずしも同じになるとは限らない）。このため逐次型Prologのcut operatorのようなバックトラック制御も可能である。

4．ＯＲ並列

本稿で示すＯＲ並列は、Coneryらの提案[3] したＯＲプロセスの実行の並列化（ＯＲプロセスが受け持つ項と単一化可能な全ての入力節の呼び出しを同時に行うもの）の他にＥＯＲプロセスの並列化（；で結ばれた項の実行を同時に行うもの）を含んでいる。ＯＲプロセスとＥＯＲプロセスの実行とでは、子プロセスを生成するのにＯＲプロセスは単一化してから、ＥＯＲプロセスは単一化をせずに行う点が違うが、子プロセスの生成以降は同一の実行手順となる。そこで、ＥＯＲプロセスのみに限定して実行手順を説明する。

例えば、Ｐ；Ｑの解を求めるＥＯＲプロセスでは以下に示すような形で実行が行われる（図３）。

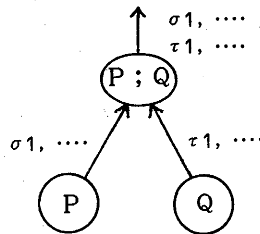


図３　ＯＲ並列のＥＯＲプロセスの実行

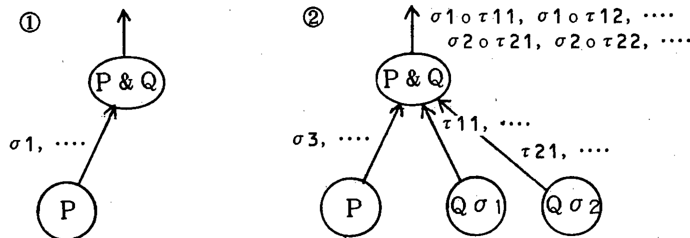


図４　ＯＲ並列のＡＮＤプロセスの実行

① P；Qの解を求めるEORプロセスは、子プロセスとしてPの解を求めるプロセスとQの解を求めるプロセスとを同時に生成する。
② 次にPのプロセスからPの解σ1,σ2,…を、QのプロセスからQの解τ1,τ2,…を得られた順に受け取り、それをそのままP；Qのプロセスの親プロセスに送る。
③ PとQのプロセスのうちいずれかが失敗すれば（解がなくなってfailメッセージが送られてくれば）そのプロセスを消去して実行を続ける。そして両方の子プロセスが消去されてしまうと、P；Qのプロセスが失敗したことを親プロセスに伝える。

本稿で示すOR並列では、ANDプロセスの実行でも若干の改良がなされている。それは、パイプライニング並列のようにP＆Qの解を求めるANDプロセスがPの解σ1を受け取った後Qσ1 の解が尽きるまでPの解を受け取らないのではなく、Pの解が得られるやいなやすぐにそれを受け取るようにしている点である（eager evaluation）。例えばP＆Qの解を求めるANDプロセスの実行は次のようになる（図4）。

① P＆Qの解を求めるANDプロセスは、まず子プロセスとしてPの解を求めるプロセスを生成し、そこから最初の解σ1 を受け取る。
② 次にσ1 の変数束縛情報に基づいてQσ1 の解を求めるプロセスを子プロセスとして生成する。この後PのプロセスとQのプロセスの両方から順々に解を受け取る。
③ 受け取った解がPからの解σ2 であれば、子プロセスとしてQσ2 のプロセスを生成し、Qσ2 からも解を受け取ろうとする。受け取った解がQσ1 からの解τ11であれば、σ1 ∘τ11をP＆Qの親プロセスに送る。以下Pからの解σi（i=3,4, …）はQσi の親プロセスを生成するのに使い、Qσj（j=1,2, …）からの解はP＆Qの親プロセスに解を送るのに使われる。
④ P，Qσ1 ，Qσ2 ，…のうちいずれかが失敗すればそのプロセスを消去して実行を続ける。そして、全ての子プロセスが消去された時にはP＆Qのプロセスが消去されたことを親プロセスに伝える。

以上述べてきたOR並列の実行では、解くべき問題の性質によって並列に実行できるプロセスの数が爆発的に増えてそれ以上の実行が不可能になってしまう恐れがある。これを防ぐにはプロセスのプロセッサへの割り当てとスケジューリングを工夫することなどが考えられる。本システムでのプロセスのプロセッサへの割り当てとスケジューリングについては以降で述べることにする。

5．ハードウェア構成

K-Prologの実装は、我々の所属する研究室で開発されたブロードキャストメモリ結合形並列計算機[5,6] 上で行った。これは本来は行列計算などの数値計算を並列に実行するために作られたもので、16ビットマイクロプロセッサ8086（Intel 社製）を用いたマルチマイクロプロセッサシステムである。

K-Prologの実装では、この並列計算機を1台のホストプロセッサと6台のスレーブプロセッサに分けて行った（図5）。各プロセッサは共通バスにより結合されていて、プロセッシングユニット（8086+8087）、メモリ（ローカル、データ、ブロードキャストの3種類があり容量はそれぞれ64Kバイト）、バススイッチ（バスコントローラを含む）から成っている。また入出力機器は共通バスにつながっているため、どのPUからでも使用可能となっている。

本システムでは、8086CPUが持つ1Mバイトのアドレス空間を64Kバイトずつのセグメントに分け、各セグメントにローカルメモリ、データメモリ、ブロードキャストメモリを割り当てている。

ローカルメモリはプロセッサごとに独立しており、それぞれ同じアドレス空間に割り当てられている。この領域は主にプログラムの格納、スタックに用いられる。

データメモリは各プロセッサごとに別々のアドレス空間に割り当てられており、他のプロセッサに実装されているデータメモリも参照できる。プロセッサは自分のデータメモリに対してはローカルバスを使用してそれぞれ別々にア

クセスし、他のデータメモリに対しては共通バスを通してアクセスする。この領域は、ホストプロセッサについてはローカルメモリと同様プログラム領域、スレーブプロセッサについては並列実行を行うプロセス領域としている。

ブロードキャストメモリは全プロセッサの共有メモリであり、それぞれ同じアドレス空間に割り当てられている。プロセッサは、読み出し時には自分のブロードキャストメモリからそれを行い、書き込み時には同一のデータを共通バスを通じて全てのブロードキャストメモリに転送する。この領域は全てのプロセッサで共有されるデータを格納するのに用いられる。

6．ソフトウェア構成

K-Prologのソフトウェア構成を図6に示す。K-Prologはホストプロセッサ側処理系とスレーブプロセッサ側処理系とに別れている。ホスト側処理系はユーザからの入力を受け付け、それが入力節（Prologプログラム）なら保持しそのコピーをスレーブプロセッサに分配する。またユーザから入力されたものがゴール節であれば、そのゴール節を受け持って実行するプロセスをスレーブプロセッサのプロセス領域中に生成する。スレーブ側の処理系はホストプロセッサが生成したプロセスをもとに並列実行を開始し、その解が得られる度にホストにそれを返していく。

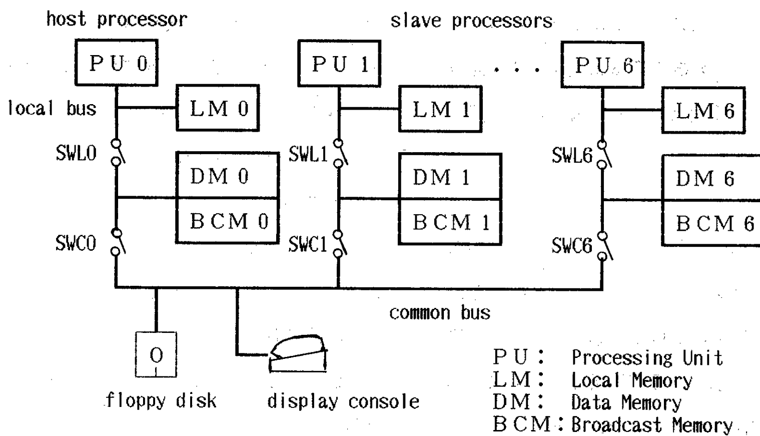


図5．ブロードキャストメモリ結合形並列計算機システム

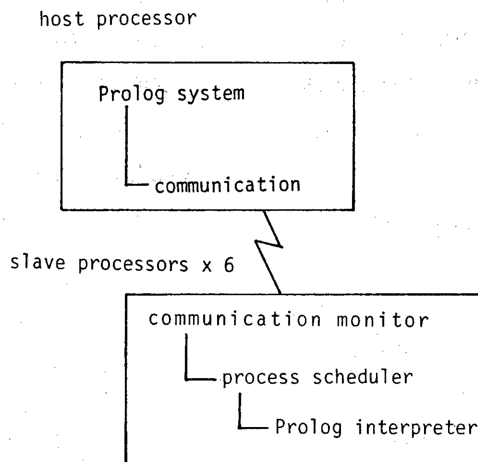


図6　K－Ｐｒｏｌｏｇのソフトウェア構成

ホスト、スレーブ各プロセッサのメモリ配置は次のようになっている。

① ホストプロセッサでは、ローカルメモリとデータメモリとをまとめて主メモリとし、そこにホスト側処理系と入力節が置かれる。

② スレーブプロセッサでは、ローカルメモリにスレーブ側処理系と入力節のコピーが置かれている。データメモリはプロセス領域となっていて、プロセスの実行を管理するのに必要な各種の情報（プロセス制御ブロック）が貯えられている。

③ ブロードキャストメモリには、これから実行されるプロセスの骨組みになる情報（ニュープロセス）を貯えているニュープロセス領域、変数の束縛状況を表す連想リスト、ホスト・スレーブ間通信のための通信領域が置かれている。

プロセスのプロセッサへの割り当てと実行管理は、各スレーブプロセッサのプロセススケジューラが行っている。プロセススケジューラは自分のプロセス領域中のプロセスを順々に走査し、ready 状態にある実行可能プロセスをrunning 状態にしてそこに制御を渡す。その後そのプロセスが何らかの原因でwaiting 状態になる（例えば、まだ実行されていない子プロセスから解を受け取ろうとした場合）と制御は再びプロセススケジューラに渡り、まだ見ていないプロセスの走査を行う。プロセス領域を一回り走査し終えるとプロセススケジューラはニュープロセス領域を見にいき、そこにニュープロセスがあればそれを自分のプロセス領域に取り込んで、再びプロセス領域の先頭から走査を始める。

このように本処理系では、プロセスのプロセッサへの割り当て方法として各スレーブプロセッサによる動的な争奪方式を採用している。この方式だと、静的な割り当てに比べバス競合による待ち時間の増加などの欠点があるが、生成されるプロセスの個数やプロセッサの台数の変化に伴う処理系の手直しの必要がないという長所がある。特に本処理系の実装ではニュープロセス領域をブロードキャストメモリに設けているため、同時に複数のプロセッサがニュープロセス領域を見にいってもバス競合は生じない。

プロセスは初めニュープロセス領域にscheduled という状態で貯えられ、手の空いたスレーブプロセッサのプロセススケジューラによってプロセス領域に移されready 状態となる。この後このプロセスに制御が渡るとrunning 状態となり、以下 running→ waiting→ ready→ runningという状態を繰り返し、実行が終了するとterminating となって消滅する。

7．性能評価

実際に例題プログラムをパイプライニング並列とＯＲ並列とで実行させた時の実行時間、生成プロセス数の計測結果をもとに両並列方式の評価を行う。使用したプログラムは素数の生成とギリシア神話データベースの問題である（プログラムリストを付録に示した）。

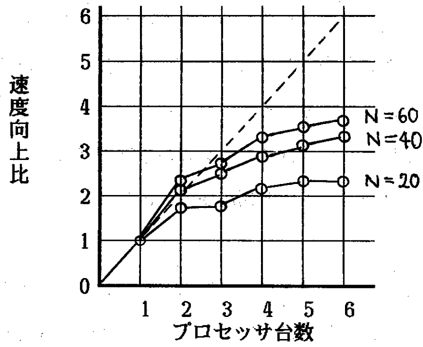
① 素数の生成

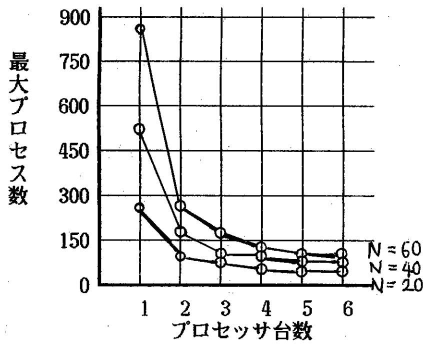
このプログラムはエラトステネスのふるいにより素数を見つけるもので、１から指定された数までの整数を生成し、その後それが２から始まってその数より１小さい数までの整数で割り切れないかどうかチェックするものである。なお、このプログラムでは１も素数に入れている。

この問題の規模を表１に、実行した結果得られた速度向上比（１台での実行時間を各台数の実行時間で割ったもの）と最大プロセス数（各スレーブプロセッサが受け持ったプロセスの瞬間的な個数のうちで最大のもの）の台数による変化を図７に示す。両並列方式ともこの問題に関しては規模の増大に伴って速度向上比が増加し、並列化の効果がよりはっきりと現れてくる。最大プロセス数と速度向上比

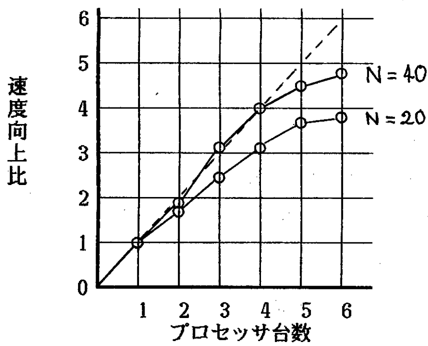
表１　素数生成問題の規模

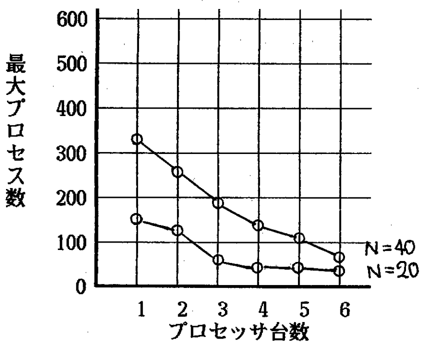
問題（ゴール文）	プロセッサ１台の実行時間(msec) パイプライニング	ＯＲ	総プロセス数
genprime(10,X)	2330	2933	401
genprime(20,X)	12303	11844	1530
genprime(30,X)	29531	25491	2967
genprime(40,X)	56363	48763	4864
genprime(50,X)	104803	-----	7677
genprime(60,X)	164497	-----	10614





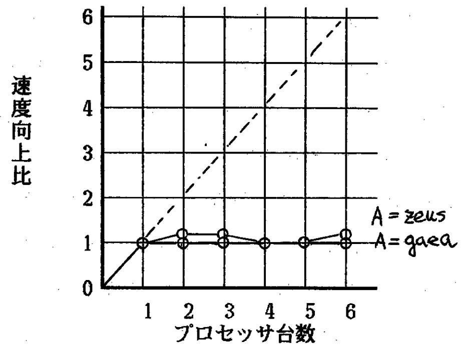
（a） パイプライニング並列

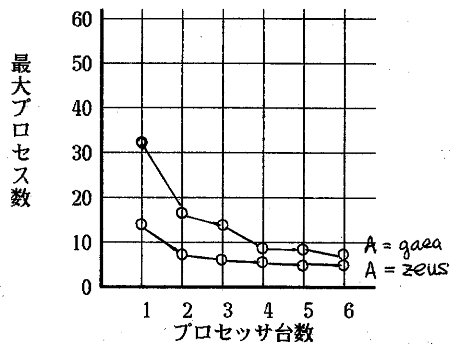




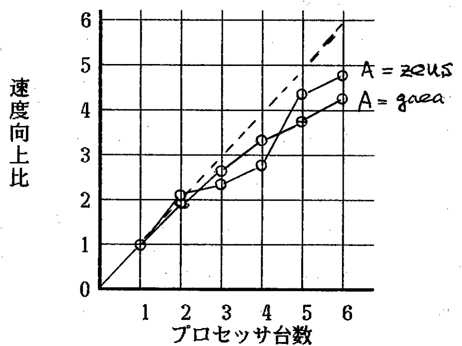
（b） OR並列

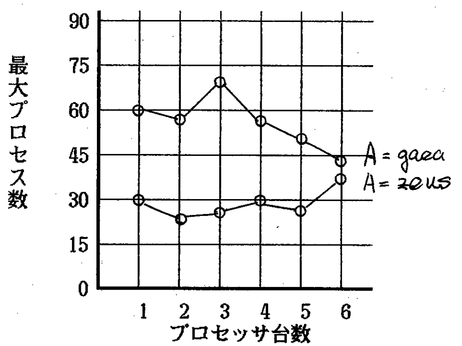
図7 素数生成問題の速度向上比と最大プロセス数





（a） パイプライニング並列





（b） OR並列

図8 ギリシア神話データベース問題の速度向上比と最大プロセス数

の増減は逆の関係にあり、最大プロセス数が急激に減っている箇所では速度向上比は急激に増大している。なお興味深いのは、パイプライニング並列のプロセッサ台数が２の時の値で、速度は１台の約２．３倍であり、最大プロセス数は１台の約１／３と２台分合わせても１台の時より小さい（N=60の時）。これは、１台のプロセッサが前向きの処理を行っている時に、別のプロセッサが別解を調べ尽くして枝刈りをしてくれるためプロセスのスケジューリング効率が上がるためだと考えられる。

② ギリシア神話データベース

これは、ギリシア神話に出てくる神々の親子関係から、ある神の子孫を求めるものである。プログラム中でｘは非存在、？は不明を表す（仮にこう決めただけで、Prologの文法とは全く関係ない）。

この問題の規模を表２に、実行した結果得られた速度向上比および最大プロセス数の台数による変化を図８に示す。パイプライニング並列による実行では全く実行速度の向上が見られないのに対して、ＯＲ並列では台数に比例した値に近い実行速度の向上が見られる（A=zeusのプロセッサ台数６の時で１台の約４．９倍）。これは、この問題がほとんどＯＲの処理ばかりで、ＡＮＤの処理はわずかしかないためである。

最大プロセス数については、パイプライニング並列による実行では台数の増加と共に単調に減少するのに対して、ＯＲ並列による実行では横ばいもしくは増大する場合（A=gaeaのプロセッサ台数３の時など）さえ見られる。従って問題によってはプロセス数が爆発的に増大し、それ以上の実行が不可能になる恐れがある。

ＯＲ並列のこのような問題点を改良する方法としては、

表２ ギリシア神話データベース問題の規模

問題（ゴール文）	プロセッサ１台の実行時間(msec) パイプライニング	OR	総プロセス数
ancestor(zeus, X)	8036	7911	233
ancestor(cronus,X)	11696	12072	363
ancestor(uranus,X)	32976	31214	1026
ancestor(gaea, X)	66539	64610	2079

プロセスのスケジューリングをうまく行ってすぐに消滅しそうなプロセスを優先的に実行する方法[7]などがあげられている。別の方法としては、ＯＲ並列とパイプライニング並列とを併用し、実行プロセス数が急激に増えそうな時にだけパイプライニング並列を使うことが考えられる。

８．おわりに

本稿では、マルチマイクロプロセッサシステム上へ実装した並列Prolog処理系について、その並列処理方式とハードウェアおよびソフトウェア構成について述べた。また、例題プログラムの実行結果から二つの並列方式に基づいた処理系の評価を行った。その結果、パイプライニング並列はプロセッサ台数、メモリ容量共に小規模のシステムに向いており、ＯＲ並列は大規模なシステムでデータベース検索等の処理をするのに向いているという特徴が得られた。

ＯＲ並列による実行で引き起こされる可能性のあるプロセス数の爆発的な増大については、パイプライニング並列との併用による防止などをあげたが、その実現方法等については今後の研究課題である。

［参考文献］

[1] 田村直之，有尾隆一，松田秀雄，金田悠紀夫，前川禎男： K-Prolog: 並列マシン上でのPrologの実現，情報処理学会記号処理研究会報告　２０－１(1982).

[2] 田村直之，松田秀雄，金田悠紀夫，前川禎男：K-Prolog（並列Prolog）の実現方法について，Proc. of Logic Programming Conference '83(1983).

[3] Conery,J.S. and Kibler,D.F.: Parallel Interpretation of Logic Programs, Proc. of the 1981 Conference on Functional Programming Languages, pp163-170 (1981).

[4] Tamura,N. and Kaneda,Y.: Implementing Parallel Prolog on a Multi-Processor Machine, Proc. of International Symposium on Logic Programming,Atlantic City(1984).

[5] 小畑正貴，金田悠紀夫，前川禎男：ブロードキャスト

メモリ結合形マルチマイクロプロセッサシステムの試作，情報処理学会論文誌，Vol.24 No.3，pp351-356(1983).

[6] 小畑正貴，金田悠紀夫，田中敏幸，前川禎男：並列プログラミング言語の設計と実現，信学技報ＥＣ84-5(1984).

[7] 相田仁，田中英彦，元岡達：並列Prolog処理システム"Paralog"について，情報処理学会論文誌，Vol.24 No.6，pp830-837(1983)．

[付録]　評価用プログラム

（ａ）　素数の生成

```
* generate prime numbers
genprime(I, P) <-
  gennum(I, I, P) & notdivide(P, 2).
gennum(I, J, 1).
gennum(I, J, N) <-
  I>0 & I1 is I-1 &
  gennum(I1, J, N1) & N1<J & N is N1+1.
notdivide(X, N) <- X<=N.
notdivide(X, N) <-
  X> N & Mod is X mod N &
  Mod!=0 & N1 is N+1 & notdivide(X, N1).
```

（ｂ）　ギリシア神話データベースの検索

```
* Greek Mithology Data Base.
ancestor(X, Y) <- parent(X, Y) ;
  (parent(X, Z) & ancestor(Z, Y)).
parent(X, Z) <-
  parents(X, Y, Z) ; parents(Y, X, Z).

parents(x, gaea, uranus).
parents(x, gaea, erebus).
parents(x, gaea, pontus).
parents(uranus, gaea, oceanus).
parents(uranus, gaea, tethys).
parents(uranus, gaea, coeus).
parents(uranus, gaea, phoebe).
parents(uranus, gaea, hyperion).
parents(uranus, gaea, iapetus).
parents(uranus, gaea, cronus).
parents(uranus, gaea, rhea).
parents(uranus, gaea, themis).
parents(uranus, gaea, mnemosyne).
parents(coeus, phoebe, leto).
parents(coeus, phoebe, hecate).
parents(hyperion, ?, hellios).
parents(hyperion, ?, selene).
parents(hyperion, ?, eos).
parents(iapetus, ?, atlas).
parents(iapetus, ?, prometheus).
parents(iapetus, ?, epimetheus).
parents(cronus, rhea, poseidon).
parents(cronus, rhea, hades).
parents(cronus, rhea, zeus).
parents(cronus, rhea, hestia).
parents(cronus, rhea, demeter).
parents(cronus, rhea, hera).
parents(zeus, hera, ares).
parents(zeus, hera, hephaistus).
parents(zeus, x, athena).
parents(zeus, hera, hebe).
parents(zeus, leto, appolo).
parents(zeus, leto, artemis).
parents(zeus, maia, hermes).
parents(zeus, dione, aphrodite).
parents(zeus, demeter,  persephopne).
parents(zeus, leda, castor).
parents(zeus, leda, pollux).
parents(zeus, leda, hellen).
parents(zeus, leda, clytemnestra).
parents(zeus, europa, minos).
parents(zeus, alcmene,  hercules).
parents(zeus, danae, perseus).
parents(zeus, semele, dionysus).
```