Vol. 26 No. 2 情報処理学会論文誌 Mar. 1985

並列 Prolog 処理系 "*K*-Prolog" の実現†

松田秀雄†† 田村直之†† 小畑正貴†††
金田悠紀夫†††† 前川禎男††††

試作マルチマイクロ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述べる．まずマルチプロセッサ上で Prolog 処理系を実現するための並列実行モデルを与えそのモデルをもとにパイプライニング並列と OR 並列という二つの並列処理方式の記述を行う．パイプライニング並列とは後戻り処理のときに必要となる別解を他のプロセッサがあらかじめ求めておくもので解の求められる順番が逐次実行の場合と同じになるという特徴をもっている．OR 並列とはゴール節中の述語からの入力節の呼出しを並列に行うものでデータベース検索等の問題に有効な方式だと考えられる．処理系の実装は筆者の所属する研究室で試作されたブロードキャストメモリ結合形並列計算機上に行った．これは 16 ビットマイクロプロセッサ 8086 を CPU にしており，共通バスにより結合されている．いくつかの例題プログラムを両並列処理方式で実行した結果，パイプライニング並列ではプロセッサ台数が小さいときに良好なデータが得られており実行プロセス数の急激な増大もなく安定している．OR 並列では全プロセッサ台数を通じて台数に比例した値に近い実行速度の向上が見られるが，実行プロセス数が急激に増大する場合があり大容量のメモリが必要となるという結論が得られている．

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．とくに 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実現のための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むずかしく，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．本論文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．実装の方針としては，言語仕様は pure Prolog（文献 1）にあげられているもの）にいくつかの組込み述語（算術演算，比較演算等を行う．入出力などの副作用がある述語はない）を付加したものとし，これを実行するインタプリタはパイプライニング並列と OR 型列という二つの並列処理方式についてそれぞれ作成することにした．

以下，これら 2 種類の並列処理方式を実現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二つの並列方式の評価を行う．

2. 並列処理方式

2.1 並列実行モデル

本論文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，与えられた 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．OR プロセスはゴール節中の各述語を受け持ち，AND プロセスはそれらの述語が AND 記号により結ばれた項 (term) 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 (unification) により生成される空節の実行を受け持つ．

本モデルと Conery らの AND-OR モデル[2] とを比べると，本モデルのほうがプロセスの種類が多いためよりきめ細かい並列実行の記述ができる．たとえば AND-OR モデルでは，AND プロセスと OR プロセスとが必ず交互に親子関係で現れなければならないが，本モデルでは各プロセスが受け持つ項によってプロセス間の親子関係が形成されるため，このような制

† Implementing Parallel Prolog System "*K*-Prolog" by HIDEO MATSUDA, NAOYUKI TAMURA (Graduate School of Science and Technology, Kobe University), MASAKI KOHATA (Faculty of Science, Okayama University of Science), YUKIO KANEDA and SADAO MAEKAWA (Faculty of Engineering, Kobe University).
†† 神戸大学大学院自然科学研究科
††† 岡山理科大学理学部電子理学科
†††† 神戸大学工学部システム工学科

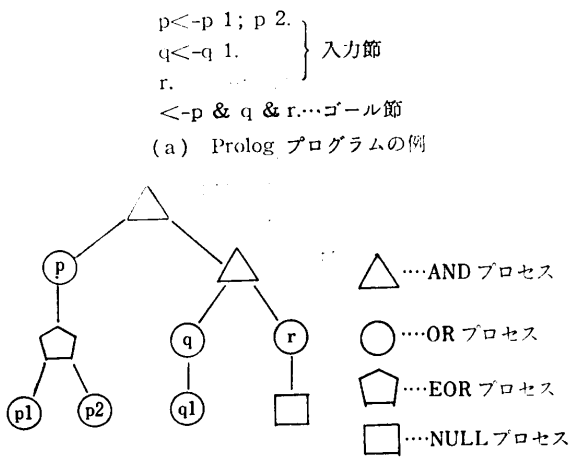

図 1 Prolog プログラムとプロセスとの関係
Fig. 1 Relation of Prolog program and process.

約はない．しかし AND 記号と明示的 OR 記号とは構文解析時に2項演算子として処理されるため，AND プロセスと EOR プロセスの数はつねに2に決められている．これらの記号の連鎖がある場合には，右結合的に結ばれた形で子プロセスが生成される．例として**図1**(a)に示すプログラム（AND 記号が '&'，含意記号が '<-' であることを除けば Edinburgh 版 Prolog と同じ記法）を実行したときにできるプロセスの木を図1(b)に示す．木の上にあるのが親プロセス，下が子プロセスとなっている．

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蓄え，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．

本モデルでは，原理的には AND 並列の記述も可能であるが，以下ではパイプライニング並列と OR 並列に限定して，AND 並列は考えないことにする．

2.2 変数への値の束縛

変数束縛情報としては，Boyer と Moore による構造体共有 (Structure Sharing) 方式[3] を用いている．すなわち，項は〈構造，添字，環境〉という三つ組で表される．構造とは項を表す骨組み (skeleton) のことであり，その項の Prolog プログラム中での元の形を示す．添字とは単一化の番号を与えるもので，これと次に述べる環境中の変数の添字とが照合することによって，どの単一化で変数に値が代入されたかがわかる．並列実行の場合には，同じ変数に一度にいくつかの値が代入されることがあるため，このような単一化ごとの変数束縛情報の分類が必要になる．環境とは変数束縛情報を蓄えている部分で，**図2**のような連想リスト（このリスト一つ一つがそれぞれ1回分の代入を表す）の連なりになっている．実際には，項の環境部分はこの連想リストの連なりを指すポインタとなっている．

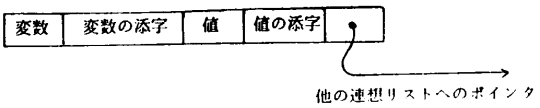

図 2 連想リスト
Fig. 2 Association list.

並列実行の場合には，プロセスは木構造状に広がっていくので，それぞれのプロセスのもつ項の環境も木構造になってつながっていく．新しい変数束縛情報つまり新しいプロセスの環境は，このすぐにできた環境に付け加わっていくだけで，以前の情報が書き換えられることはなく，複数のプロセスで一つの環境を共有することが可能である．しかし，束縛されている変数の値を得るには，環境内で変数と添字とを照合しながら探索を繰り返す必要があり時間がかかる．また，木構造状の環境の不要になった枝を回収する必要がある．

2.3 プロセスの制御とプロセス間通信

プロセスの制御および通信の実現のため，次のような命令が用意されている．

(1) new (項)

与えられた項を実行するプロセスを子プロセスとして生成する．

(2) receive (子プロセス)

指定された子プロセスからメッセージを受け取る．子プロセスの send と同期を取っているため，子プロセスがメッセージを送ってくるまで待たされる．

(3) send (メッセージ)

親プロセスにメッセージを送る．親プロセスの receive と同期をとっているが，通信は固定長のバッファを介して行われるため，バッファに空きがある限

り待たされずに次の実行に移れる．

（4） terminate (子プロセス)

指定された子プロセスの実行を終了させる．子プロセスの done と同期をとっている．

（5） done

プロセスの実行を終了する．親プロセスの terminate と同期をとっている．

2.4 パイプライニング並列

パイプライニング並列とは筆者らの提案[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．新田らの提案した Backup 並列[5]と基本的に同じものであるが，並列実行のためのモデルが Backup 並列では AND-OR モデルとなっているため，プロセスの生成の様子が異なっている．すなわち，Backup 並列では AND プロセスと OR プロスしかなく，それらが必ず交互に親子関係で結ばれているが，パイプライニング並列では4種類のプロセスがあり，各プロセスの受け持つ項によって親子関係で結ばれるプロセスの種類が変わってくる．また，Backup 並列では AND プロセスは任意の数の子プロセスをとれるのに対して，パイプライニング並列では AND プロセスと EOR プロセスの子プロセスの数は2に限定されている．

この方式の特徴であるパイプラインの動作は，AND プロセスの実行で行われる．AND プロセスのアルゴリズムをC言語風の疑似言語で記述したものを**図3**(a)に示す．AND プロセスが引数として受け取るのは前述した三つ組表現の項で，その構造 (structure) 部分は and (P, Q) といった関数形式をしている．この関数の二つの引数は，元のプログラム中で AND 記号の両側にあった項を示し，それぞれ get-1st-argument と get-2nd-argument とによって取り出される．AND プロセスは，まず AND の第1引数の項を受け持つプロセスを子プロセスとして生成 (new) し，そこから解 (message) を受け取る (receive)．次にその解を新しい環境として，AND の第2引数の項を受け持つプロセスを子プロセスとして生成し，そこから解を受け取る．この2番目の子プロセスの解は AND プロセスの親プロセスに送られる (send)．この間に最初の子プロセスは継続として別の解を求めており，これをメッセージバッファに蓄えている．そして2番目の子プロセスの解が尽きる (fail が送られてくる) と，AND プロセスはこのプロセスを消去 (terminate) して，あらかじめバッファに蓄えられた最初の子プロセスの別解を受け取り，先ほどと同様にその解を新しい環境として AND の第2引数の項のプロセスを生成する．最初の子プロセスからの解が尽きれば，そのプロセスを消去し fail メッセージを送って実行を終了 (done) する．パイプライニング並列では，つねに直前に生成された子プロセス (youngest-child-process：この変数の値は new を行うたびに更新される) からしか解を受け取らない．

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すべて逐次実行の場合と同じ動作をする．例として EOR プロセスのアルゴリズムを記述したものを図3(b)に示す．EOR プロセスの実行でも，EOR プロセスが受け持つ項の構造 (structure) 部は eor (P, Q) という関数形式をしており，その第1，第2引数は元のプログラム中で明示的 OR (';') の左右の項となっている．そして，この二つの引数を受け持つ子プロセスが順々に生成されるが，最初の子プロセスの実行と2番目の子プロセスの実行とは重なることがなく，一時には一つの子プロセスしか存在しない．これらの子プロセスの求めた解はそのまま EOR プロ

```
and_process(st, ix, ev)
    /* structure, index, environment */
{
    get_1st_argument(st, ix, ev, &st_arg1, &ix_arg1);
    new(st_arg1, ix_arg1, ev);
    while ((message = receive(youngest_child_process)) != FAIL)
    {
        get_2nd_argument(st, ix, message, &st_arg2, &ix_arg2);
        new(st_arg2, ix_arg2, message);
        while ((message = receive(youngest_child_process)) != FAIL)
            send(message);
        terminate(youngest_child_process);
    }
    terminate(youngest_child_process);
    send(FAIL);
    done;
```

(a) AND プロセス

```
eor_process(st, ix, ev);
    /* structure, index, environment */
{
    get_1st_argument(st, ix, ev, &st_arg1, &ix_arg1);
    new(st_arg1, ix_arg1, ev);
    while ((message = receive(youngest_child_process)) != FAIL)
        send(message);
    terminate(youngest_child_process);
    get_2nd_argument(st, ix, ev, &st_arg2, &ix_arg2);
    new(st_arg2, ix_arg2, ev);
    while ((message = receive(youngest_child_process)) != FAIL)
        send(message);
    terminate(youngest_child_process);
    send(FAIL);
    done;
```

(b) EOR プロセス

図3 パイプライニング並列でのプロセスのアルゴリズム
Fig. 3 Algorithm of process in pipelining parallelism.

```
eor_process(st, ix, ev);
    /* structure, index, environment */
{
    get_1st_argument(st, ix, ev, &st_arg1, &ix_arg1);
    new(st_arg1, ix_arg1, ev);
    get_2nd_argument(st, ix, ev, &st_arg2, &ix_arg2);
    new(st_arg2, ix_arg2, ev);
    while (exist_child_process())
    {
        msg_process = get_process_which_sent_message();
        if ((message = receive(msg_process)) != FAIL)
            send(message);
        else
            terminate(msg_process);
    }
    send(FAIL);
    done;
}
```

(a) EOR プロセス

```
and_process(st, ix, ev)
    /* structure, index, environment */
{
    get_1st_argument(st, ix, ev, &st_arg1, &ix_arg1);
    new(st_arg1, ix_arg1, ev);
    1st_process = youngest_child_process;
    while (exist_child_process())
    {
        msg_process = get_process_which_sent_message();
        if ((message = receive(msg_process)) != FAIL)
        {
            if (msg_process == 1st_process)
            {
                get_2nd_argument(st, ix, message, &st_arg2, &ix_arg2);
                new(st_arg2, ix_arg2, message);
            }
            else
                send(message);
        }
        else
            terminate(msg_process);
    }
    send(FAIL);
    done;
}
```

(b) AND プロセス

図 4 OR 並列でのプロセスのアルゴリズム
Fig. 4 Argorithm of process in OR parallelism.

セスの親プロセスに送られる.

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まったく同じになるという特徴がある (ただし入出力や assert, retract によるプログラムの書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.

2.5 OR 並列

本論文で示す OR 並列は, Conery らの提案[2] した OR プロセスの実行の並列化 (ゴール節中の述語と単一化可能なすべての入力節の呼出しを同時に行うもの) の他に EOR プロセスの並列化 (明示的 OR ';' で結ばれた述語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.

OR 並列での EOR プロセスのアルゴリズムを図 4 (a) に示す. EOR プロセスが受け持つ項の表現等については, まったくパイプライニング並列の場合と同じである. パイプライニング並列との違いは, 同時に二つの子プロセスが生成され, しかもそれらの子プロセスの両方から解を受け取る点である. このため, 図 3 にはなかった, すでに解を求めてメッセージバッファにそれを置いている子プロセスを探しそのプロセスへのポインタを返す関数 (get-process-which-sent-message) が追加され, この子プロセス (msg-process) のところから解を受け取るようにしている (まだどの子プロセスも解を求めていないときには, この関数のところで待つ). 子プロセスからの解は EOR プロセスの親プロセスに送られ, 解が尽きれば (fail が送られてくれば) その子プロセスは消去される. 子プロセスがすべて消去されてしまうと (exist-child-process が偽になると) fail メッセージを送って実行を終了する.

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直前に生成された子プロセスからしか解を受け取らないのではなく, すべての子プロセスから解を受け取るようにしている点である. OR 並列での AND プロセスのアルゴリズムを図 4 (b) に示す. AND プロセスが受け持つ項の表現および生成される子プロセスが受け持つ項については, パイプライニング並列の場合とまったく同じであるが, 子プロセスとの通信に関しては, EOR プロセスのときと同様, 解を求めた子プロセスを探し, そこから解を受け取るという形になっている. そして, AND の第 1 引数を受け持つ最初に生成された子プロセス (これは一つしか作られない) からの解は, 次の第 2 引数を受け持つ子プロセス (これは最初の子プロセスからの解に応じて, 次々に作られる) を生成するための環境とし

て用いられ，第2引数を受け持つ子プロセスからの解は，AND プロセスの親プロセスに送られる．fail メッセージを送ってきたプロセスは消去され，すべての子プロセスが消去されると，AND プロセスの親プロセスに fail メッセージを送り，実行を終了する．

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おそれがある．これを防ぐには，プロセスのプロセッサへの割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．

3. ハードウェア構成

K-Prolog の実装は，われわれ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[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 64 k バイト)，バススイッチ (バスコントーラを含む) 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，どのプロセッサからでも使用可能となっている．本システムでは，8086 CPU がもつ 1 M バイトのアドレス空間を

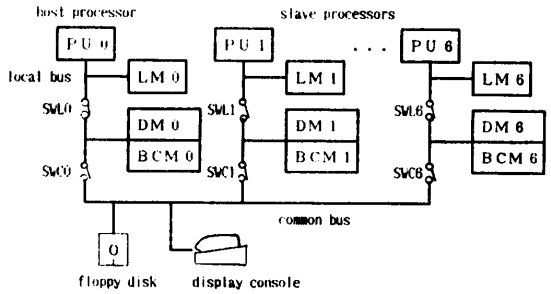

図 5 マルチマイクロプロセッサシステムのハードウェア構成

Fig. 5 Hardware structure of multimicroprocessor system.
PU: Processing Unit, LM: Local Memory, DM: Data Memory, BCM: Broadcast Memory

64 k 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．この領域はおも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共通バスを通じて参照できる．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プロセス領域 (後述) 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，書込み時には同一のデータを共通バスを通じてすべてのブロードキャストメモリに転送する．この領域は，すべ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．

4. ソフトウェア構成

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分かれている．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，そのコピーをすべての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初期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．ここでプロセス領域とは，各プロセスの実行に必要な情報 (プロセス制御ブロック) が置かれている領域であり，それぞれのスレーブプロセッサのデータメモリ中に設けられている．

スレーブ側処理系は各スレーブプロセッサ上に同一のものが実装されているが，この初期プロセスを受け取ったスレーブプロセッサ (現在のところ，あるプロセッサに固定されている) を中心にして並列実行を開始し，その解が得られるたびにホストにそれを返していく．初期プロセス以外のプロセスの生成は，ブロードキャストメモリ中に設けられたニュープロセス領域にニュープロセスと呼ばれるプロセス生成に必要な情報が置かれ，手の空いているスレーブプロセッサがそれを自分のプロセス領域に取り込むという形で行われ

る．このプロセスのプロセッサへの割当てと実行管理は，スレーブ側処理系中のプロセススケジューラが行っている．各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（正確にはそのプロセスの制御ブロック）を round robin 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（たとえば，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．

このように本処理系では，プロセスのプロセッサへの割当て方法として，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．とく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．

5. 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ときの実行時間，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（プログラムについては，文献 7）参照）．

5.1 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．

この問題を実行したときの，実行時間と最大プロセス数（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）のスレーブプロセッサ台数による変化を**表1**に示す．両並列方式ともにこの問題に関しては，問題の規模の増大に伴ってプロセッサ台数の増加に伴う実行時間の減少の比率が大きくなり，並列化の効果がよりはっきりと出てくる．最大プロセス数と実行時間の増減は相関しており，最大プロセス数が急激に減っている箇所では実行時間も急激に減少している．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のときの値で，実行速度は1台の約2.1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台のときより小さい（40までの素数のとき）．これは，2台のプロセッサで前進処理と後戻り処理とを分担し，一方のプロセッサが前進処理をしている間に他方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，1台による実行の場合と比べてプロセスが早めに消滅しスケジューリング効率が上がるためだと考えられる．

5.2 ギリシア神話データベース

これは，ギリシャ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．

実行時間と最大プロセス数のスレーブプロセッサの台数による変化を**表2**に示す．パイプライニング並列による実行ではまった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（Zeus の子孫を求めた台数6の時の値で1台の約4.9倍）．これは，この問題が

表1　素数生成問題の実行時間と最大プロセス数

Table 1　Execution time and maximum numbers of process in problem of generating prime numbers.

	問題	並列方式	スレーブプロセッサ台数 1	2	3	4	5	6
実行時間 (sec)	20までの素数	パイプライニング	12.3	6.9	6.8	5.8	5.7	5.6
		OR	11.8	6.8	4.8	3.8	3.2	3.1
	40までの素数	パイプライニング	56.4	26.8	22.2	19.4	18.2	17.7
		OR	48.8	25.8	15.7	12.2	10.8	10.1
最大プロセス数	20までの素数	パイプライニング	269	99	69	48	39	35
		OR	169	128	60	44	38	32
	40までの素数	パイプライニング	539	175	116	92	73	65
		OR	328	265	192	135	106	71

表 2 ギリシア神話データベース問題の実行時間と最大プロセス数
Table 2 Execution time and maximum numbers of process in problem of Greek mithology database.

	問題	並列方式	スレーブプロセッサ台数 1	2	3	4	5	6
実行時間 (sec)	Zeus の子孫	パイプライニング	8.0	7.6	7.5	7.9	8.0	7.7
		OR	7.9	3.7	3.4	2.8	1.8	1.6
	Gaea の子孫	パイプライニング	66.5	66.0	66.8	67.3	67.3	67.4
		OR	64.6	34.0	25.1	19.5	16.9	15.0
最大プロセス数	Zeus の子孫	パイプライニング	14	8	7	6	5	5
		OR	30	27	28	30	28	35
	Gaea の子孫	パイプライニング	32	17	13	9	9	8
		OR	60	56	70	55	49	41

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とも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（Gaea の子孫を求めた台数 3 のときの値など）さえ見られる．したが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おそれがある．

OR 並列のこのような問題点を改良する方法としては，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[5]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ときにだけパイプライニング並列を使うことが考えられる．

6. おわりに

本論文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，解くべき問題の性質や規模，またそれを実行する並列処理計算機のプロセッサ台数，メモリ容量などの構成によって両並列処理方式の使い分けが必要であることがわかった．

パイプライニング並列による実行ではプロセッサ台数が小さい場合には実行速度等の面で有効な結果が得られるが，台数が増加するに従ってすぐ頭打ちとなってしまう．また問題の性質によってはまったくプロセッサ台数の増加に伴う実行速度の向上が見られない（データベース検索等の問題）．しかし，実行プロセス数はプロセッサ台数を増やしても増大せず安定した値を示すためメモリを大量に消費することはなく，小容量のメモリしか実装されていないシステムでも実現可能であると考えられる．

OR 並列による実行では問題の性質によらずにほぼプロセッサ台数に比例した実行速度の向上が見られ，とくにデータベース検索の問題には有効な結果が得られた．しかし，実行プロセス数の爆発的な増大の可能性が示唆されており，その場合には大容量のメモリが必要となる．このプロセス数の増大を防止する方法としてはパイプライニング並列との併用などをあげたが，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) Kowalski, R.: Predicate Logic as Programming Language, Proc. of IFIP Congress 74, pp. 569-574 (1974).
2) Conery, J. S. and Kibler, D. F.: Parallel Interpretation of Logic Programs, Proc. of the 1981 Conference on Functional Programming Languages, pp. 163-170 (1981).
3) Boyer, R. and Moore, J.: The Sharing Structure in Theorem Proving, *Machine Intelligence*, Vol. 7, pp. 101-116 (1972).
4) Tamura, N. and Kaneda, Y.: Implementing Parallel Prolog on a Multi-Processor Machine, Proc. of International Symposium on Logic Programming, Atlantic City, pp. 42-48 (1984).
5) 新田克己，松本裕治，古川康一：Prolog インタプリタの記述と並列化への拡張，電子通信学会論文誌，Vol. J 66-D, No. 11, pp. 1310-1317 (1983).

6) 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, pp. 351-356 (1983).
7) 松田秀雄，田村直之，小畑正貴，金田悠紀夫，前川禎男：K-Prolog の並列処理方式とその評価，情報処理学会計算機アーキテクチャ研究会報告，54-4 (1984).
8) 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, pp. 830-837 (1983).

(昭和59年5月23日受付)
(昭和59年9月20日採録)