計算機アーキテクチャ 81－2
（１９９０．３．２３）

知識ベース指向並列処理システム*

横田 治夫、 北上 始、 服部 彰
富士通株式会社

知識ベースが利用できる並列処理環境を提供するため，並列推論モジュールと並列知識検索モジュールからなる知識ベース指向の並列処理システム (KOPPS: Knowledge-base-Oriented Parallel-Processing System)を提案する．KOPPS では，知識検索のモデルとして RBU を採用し，並列論理型言語 GHC から RBU に並列にアクセスして推論を行う．本論文では，KOPPS の概要と，マルチマイクロ計算機上に試作したシステムの構成，および試作システム上での意味ネットワーク検索の実験とその結果について報告する．実験の結果，本構成で十分実用になる性能で知識を処理することが可能で，検索と推論の処理負荷のバランスを取りながら並列化することにより，プロセッサの数に従って全体の処理速度が向上することを確認できた．

Knowledge-Base-Oriented Parallel-Processing System*

Haruo Yokota, Hajime Kitakami, and Akira Hattori
FUJITSU LIMITED
1015 Kamikodanaka, Nakahara-ku, Kawasaki 211, Japan

We propose a knowledge-base-oriented parallel-processing system (KOPPS) consisting of parallel inference and parallel knowledge retrieval modules. The KOPPS uses RBU as its knowledge retrieval model and makes inferences by accessing RBU in parallel from guarded Horn clauses (GHC), a parallel logic programming language. This paper gives an overview of the KOPPS and a multiprocessor-based configuration. It also presents test results from an application for traversing semantic networks. Results show the configuration to be effective for actual knowledge processing. Parallel execution speedup corresponding to the number of processors is achieved by load-balancing the knowledge retrieval and inference.

* 本研究は第五世代コンピュータプロジェクトの一環として行われたものである．

1．はじめに

計算機システムを人間により身近なものとするためには，人間の常識や経験に関する知識などを利用して処理を進めることが要求される．そのような知識は，システムを人間に近づけようとすればするほど量を増すことになる．このため，個々の知識をまとめて知識ベースとして管理し，要求される知識に対する高速アクセス手段を提供することが必要となってくる．また，知識に対する処理自身を高速化することも，システムを使い易くするためには重要である．そこで，処理を並列化して性能を上げることも必須となってきている．

知識を利用して処理を行うシステムを構成する場合に，LISP や Prologなどの逐次型の記号処理言語でシステム全体を記述する方法がまず考えられる．しかし，その場合には並列処理が不可能となり，速度的な要求を満たすことができない．次にGHC[1,2]，Parlog[3]，Concurrent Prolog[4] などの並列型の記号処理言語でシステム全体を記述する場合を考えてみる．上に挙げたような並列記号処理言語では，いかなる並列動作をしても矛盾が生じないように，並列処理単位間でグローバルなデータが共有できない．つまり，内容が更新されうる知識を静的に保持して，それらに対して複数処理単位から並列にアクセスすることが許されていない．知識ベースを永久プロセス[2]などを用いて動的に実現する方法も考えられるが，更新した内容を恒久的に保持できない上，知識の容量が大きくなった場合の高速アクセス手段が与えられていない．このように，逐次あるいは並列の記号処理言語のみで，知識ベースを使った高速システムを構成することには困難が伴う．

我々は，知識ベースが利用できる並列処理環境を提供することを目指し，並列推論モジュールと，並列知識検索モジュールからなる知識ベース指向の並列処理システム（KOPPS: Knowledge-base-Oriented Parallel-Processing System）を提案する．つまり，推論する部分は並列型の記号処理言語を使い，知識ベースの管理については専用の並列検索モジュールを用意するものである．知識ベース管理を専用化することにより，知識検索用にカスタマイズした高速アクセス手段と，排他制御機構を提供することができる．また，並列記号処理言語を利用することにより，検索された知識を高速に操作することができる．

この様な推論機構と検索機構の結合は，演繹データベース・システム[5]の考え方に近い．演繹データベースは，関係データベースを基本にして，述語論理との結合を図っている．ただし，演繹データベースでは，格納する対象がデータのみであり，知識を格納することを前提としていない．つまり，知識を表現するための手段（例えば構造体や変数など）をその機能を保持したまま格納・検索することができない．また，並列化を指向している演繹データベース・システムの研究もあるが，実際の並列処理の結果は現在のところ報告されていない[5]．KOPPS では，構造あるいは変数をそのまま構造や変数として格納し，検索する．さらに，検索も推論も並列に実行することを重視している．

本論文では，まず第２章でシステムの概要を述べ，第３章でマルチマイクロ計算機上に試作したシステムの構成について紹介する．第４章では，応用の１つとして KOPPS 上で意味ネットワークをトラバースする方法について述べる．さらに，試作システム上でプロセッサ台数を変化させて実行した場合のそのトラバースの処理性能について，第５章で報告する．

2．システムの概要

KOPPSでは，知識ベースのモデルとして，RBU[6]を採用した．RBU は，Retrieval By Unification の頭文字を取ったもので，関係データベースモデルを基に，格納対象と検索機能を知識ベース用に拡張したものである．RBU では，知識は項（変数を含んだ構造体）で表現され，その集合は項関係（項を要素とするテーブル）に格納されて管理される（排他制御は，項関係単位で行われる）．項関係の横１列をタプル，縦１列を属性と呼ぶ．検索では，項の構造と変数バインディングを対象とする

単一化(Unification)と呼ばれる高機能なパターンマッチング操作を使う．項関係のある属性の各要素と検索条件との間で単一化を行い，成功したタプルを取り出す．専用のインデックス[7]を利用することにより，高速にこの検索を実行できる．

知識ベースは決して不変なものではなく，内容は絶えず更新されることが想定できる．これは，項関係のタプルを削除したり，新たなタプルを追加することに対応する．このためには，対話的な動作が重要となる．KOPPS では，インデックス全体を作り替えずに部分修正だけで更新処理に対応することができるため，知識ベースの更新に対しても強力である．

並列推論モジュールでは，GHC[1]で処理を記述する．GHC は，述語単位でプロセスを生成し，その間で変数のバインディングによりメッセージのストリームを生成して処理を進める．このため，小粒度に並列処理を記述することができ，同期処理などを明示的に書く必要がなく自然に並列性が抽出できる．ただ，先に述べた並列言語特有の制約から，GHC だけで知識ベースを使った処理を記述することは効率的でない．そこで，検索された知識に対する操作や，ユーザインタフェースの部分だけを GHC プロセスが担当する．GHC で記述されたプログラムの中から専用の組込述語を使って知識ベースの検索や更新のコマンドを RBU に渡す．このとき，GHC 内で使われるデータ構造が RBU と同様の項そのものであるため，GHC プログラムからの知識ベース利用が容易になっている．

並列推論モジュールと並列知識検索モジュールの間のデータ転送は，リスト状につながれた GHC の論理変数を用いた要求駆動によるストリーム通信である．この論理変数は，コマンドとともに未完成メッセージ[2]として渡される．並列推論モジュールは，未定義変数をリストセルに入れて，転送要求として並列検索モジュールに渡す．並列検索モジュールは，この要求に従って検索結果を１つ１つ変数にバインドさせる．この転送は，検索処理と並列に，すべての検索結果がそろうのを待つことなく，結果が得られた時点で行われる．

３．試作システムの構成

我々は，KOPPS を Sequent 社製の共有メモリ型マルチマイクロ計算機 Symmetry 上に試作し，その試作システムを使って実験を行った．今回実験に使ったSymmetry の構成は，インテル 80386 のプロセッサボードが共有バスに１８台接続されているもので，このプロセッサやメモリ等の資源は UNIX

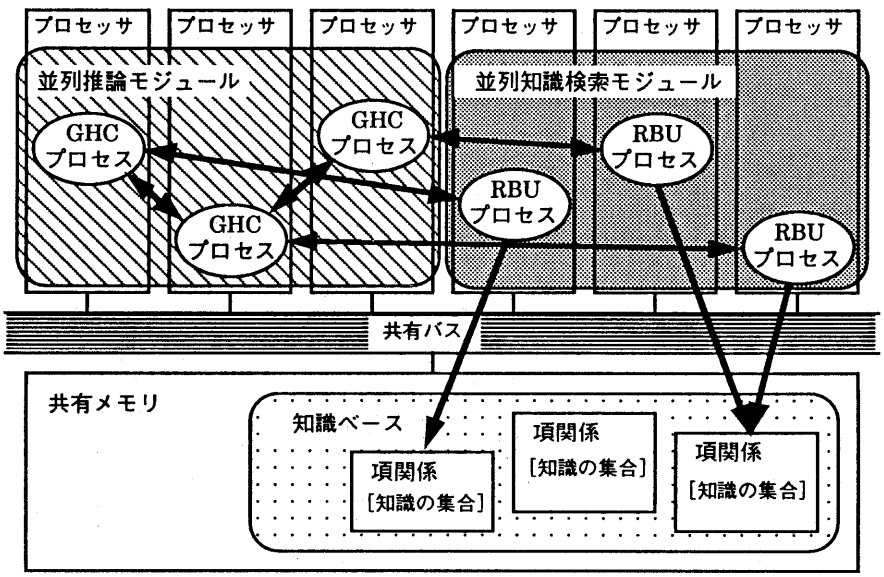

図１　試作システムの構成

ベースの並列OSであるDYNIXが管理する．

図1に，試作システムの構成を示す．Symmetryの複数のプロセッサ上に，GHCのプロセスとRBUプロセスを置いて，それぞれのプロセス間で通信をしながら処理を進める．利用するプロセッサの台数は，実験のため変化させる．この時，あるプロセッサは，並列推論モジュール用か並列知識検索モジュール用に振り分けられ，1つのプロセッサ上で2種類のプロセス両方を実行することがない．GHCは述語単位で，RBUは検索／更新コマンド単位でプロセスとして実行される．実際に起動をかけられるプロセスは，負荷分散に従ってそれぞれのモジュール内で動的に決定される．

知識ベース（項関係の集まり）は，共有メモリ上に置かれ，複数のRBUのプロセスから同時にアクセスされる．この並列アクセスに対する排他制御は項関係単位で行われ，更新処理か検索処理かによって，共有モードと排他モードを使い分ける．この排他制御のための情報は，ディクショナリに格納され，セマフォによって排他機構は実現されている．

推論モジュールと検索モジュールの間の接続も，共有メモリ上の共有領域を用いて実現している．共有メモリ上に，知識ベースの領域とは別に，コマンドページと呼ぶ領域を用意して，双方のモジュールからアクセスする．並列推論モジュールからは，検索や更新の指示をこのコマンドページを介して並列知識検索モジュールに渡す．並列知識検索モジュールで行われた検索の結果も，同様にコマンドページを介して並列推論モジュールに渡される．このとき，文字列で送るのではなく，ポインタでつながれたセル構造で情報が渡されるため，アトムテーブルはそれぞれのモジュールの間で共有している．アトムテーブルを共有することにより，通信容量や変換処理を大幅に減らすことができる．

コマンドや結果が上書きされたり，複数プロセスに重複して読み込まれたりしないように，アクセスするときにコマンドページ単位に排他制御を行う．このためコマンドページが1つだと，そのコマンドページへのアクセスが並列処理のボトルネックになる．そこで，コマンドページを複数設けて，アクセスを分散させている．試作システムでは，並列知識検索モジュールに割り当てられたプロセッサの台数と同じ個数のコマンドページを用意した．ただし，負荷が片寄らないよう，両モジュールの各プロセスはどのコマンドページにもアクセスできる．

この複数コマンドページを用いた動的コマンド割り振りにより，コマンド間の実行順序が保証されないことになる．並列推論モジュール側でコマンド間の順序関係を明示的に制御したい場合には，検索／更新コマンド終了時に並列知識検索モジュール側から返されるステータス情報をコマンド生成に利用する．

4．意味ネットワークのトラバース

KOPPSの応用の1つとして，意味ネットワークで表現されるような知識を知識ベースに格納して，並列に検索することを考える．例えば，図2のように表現される知識に対して，ネットワークを手繰りながら計算機に関する情報を収集する．

図2の知識を項関係に格納したものを表1に示

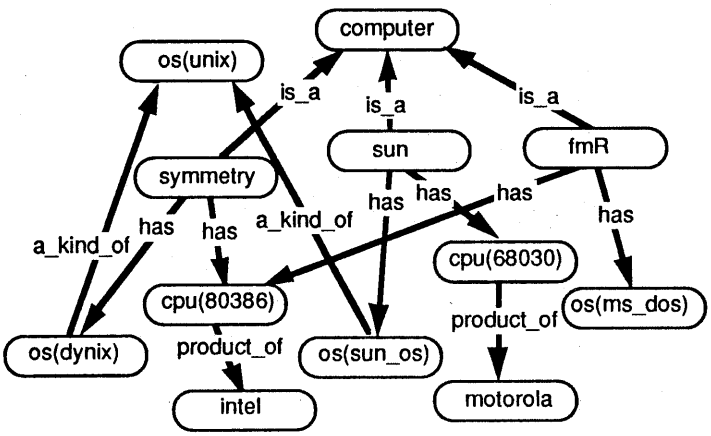

図2　計算機に関する知識の一部

す．ここで，$(1) や $(2) は，RBU 内で使われる論理変数を表わし，メタとオブジェクトの間の変数の区別を行っている[8]．項関係の第1属性と第3属性はオブジェクトを，第2属性はその間の関係を示している．第4，5，6属性は，意味階層をたどる場合に，問い合わせ中の変数のバインディング情報を保存するための内部変数である．このような，論理変数による意味階層の履歴保持は，DCKR[9] の方法と同様のものである．

この時のトラバースするための並列推論モジュール側のプログラムを図3に示す．プログラム中の rbu_stream という述語が並列知識検索モジュールにコマンドを渡すための組込述語である．また，file_stream は入出力のための組込述語である．search_loop という述語の第3引き数によって，処理ループ中で発生する RBU コマンドを rbu_stream にストリームとして渡している（ここで，urs は RBU のコマンドの1つで，単一化制約のコマンドである[6]）．なお，プログラムの中で，S=[_,_|_] あるいは L=[_,_,|_] と記述してある部分は，ダブルバッファリングを使った要求駆動処理のために未定義変数の入ったリストを2つ分用意しているところである．

並列知識検索モジュールでは，コマンド単位で複数の検索処理を並列に実行する．このため，変数 RBU にバインドされるコマンド列は，そのときの並列知識検索モジュールに割り当てられたプロセッサ分だけ並列に実行される．

例えば，symmetry に関する知識を検索するため

表1　図2の知識の項関係

オブジェクトA	関係	オブジェクトB	ワーク用		
symmetry	is_a($(1))	computer	$(1)	$(2)	$(3)
symmetry	has($(1))	os(dynix)	$(1)	$(2)	$(3)
symmetry	has($(1))	cpu(80386)	$(1)	$(2)	$(3)
sun($(1),$(2))	is_a($(3))	computer	$(3)	$(4)	$(5)
sun($(1),$(2))	has($(3))	os(sun_os)	$(3)	$(4)	$(5)
sun(3,$(1))	has($(2))	cpu(68030)	$(2)	$(3)	$(4)
fmR($(1))	is_a($(2))	computer	$(2)	$(3)	$(4)
fmR($(1))	has($(2))	os(ms_dos)	$(2)	$(3)	$(4)
fmR(70)	has($(1))	cpu(80386)	$(1)	$(2)	$(3)
cpu(68030)	product_of($(1))	motorola	$(1)	$(2)	$(3)
cpu(80386)	product_of($(1))	intel	$(1)	$(2)	$(3)
os(dynix)	a_kind_of($(1))	os(unix)	$(1)	$(2)	$(3)
os(sun_os)	a_kind_of($(1))	os(unix)	$(1)	$(2)	$(3)
$(1)	$(1)	nil	empty	$(2)	$(3)

```
traverse(KB,Arg1,Arg2) :- true |
    rbu_stream(RBU),
    file_stream(OUT),
    search_loop([[Arg1,Arg2,Arg1,Arg2],-1],KB,RBU,OUT).

search_loop([[Q1,Q2,Q3,Q4]|L],KB,R1,O1) :- Q1 \= nil |
    R1 = [urs(KB,[1=Q1,2=Q2,5=Q3,6=Q4],[3,4,5,6],S)|R2],
    S = [_,_|_],
    search_loop(S,KB,R3,O2),
    L = [_,_|_],
    search_loop(L,KB,R4,O3),
    merge(R3,R4,R2),
    merge(O2,O3,O1).
search_loop([[nil,empty,Q1,Q2]|L],KB,R,O1) :- true |
    O1 = [writeterm([Q1,Q2])|O2],
    L = [_,_|_],
    search_loop(L,KB,R,O2).
search_loop([-1,_|L],KB,R,O) :- true |
    L = [],
    R = [],
    O = [].
```

図3　トラバースするための並列推論モジュール側のプログラム

```
traverse(kb,symmetry,$(10)).
```

という問い合わせを行うと，

```
urs(kb,[1=symmetry,2=$(10),5=symmetry,
        6=$(10)],[3,4,5,6],S)
```

というコマンドが並列知識検索モジュールに渡さ

れる．このコマンドは，「kb と言う項関係の第1属性と第5属性を symmetry と単一化し，第2属性と第6属性同士を単一化できるタプルの3，4，5，6属性を取り出す」という意味で，表の 1，2，3番目と最後の4つのタプルのが得られる．この結果の内1つ（第1属性が nil になったもの）が出力され，残りの3つは新たなコマンド生成に使われる．それらの3つのコマンドは並列に動作することが可能である．この様子を図4に示す．RBUプロセスの左側の矢印が検索条件，右側がその検索結果を概念的に（検索条件や結果の正確な記述ではない）示している．ある RBU プロセスによって検索された結果は，次の RBU プロセスの検索条件となっている．これらのプロセス間の受け渡しは，GHC が行う．このため，動作順序等を記述することなく自然に並列に処理が進んでいく．また，RBU プロセスは検索が完了しないうちに，条件を満足するタプルが見つかったところから検索結果を返すため，結果的に図中の RBU プロセスは，ほとんど全てが並列に動作することになる．

図で，斜体で書かれた部分が出力である．最終的に，

[symmetry,symmetry]
[symmetry, is_a(computer)]
[symmetry, has(os(dynix))]
[symmetry, has(cpu(80386))]
[symmetry, has(a_kind_of(os(unix)))]
[symmetry, has(product_of(intel))]

という単なるデータベース検索では得られない意味階層を反映した答えが出力される．

この他，構造が直接知識ベースの中に格納されているため，例えば has 関係に注目して，

traverse(kb,symmetry,has($(20))).

のように問い合わせに構造を含ませることもできる．また，対話的な更新ができるため，検索結果を見ながら項関係に対するタプルの削除／挿入を行うことができる．

5．実験結果

性能を評価するため，表1にあるような計算機に関する項関係の知識を5 5 6タプル分用意した．その項関係に対して，並列推論モジュールと並列知識検索モジュールのプロセッサの割り当て台数をそれぞれ1台から振らせた場合の，ある（6 4個の結果が得られる）問い合わせに対する検索時間を測定した．

この時，GHC はヒープ領域が少なくなってくると GC を行い，構造体の位置をずらす[10]ため，このモジュール間インタフェースは，その位置変更に対応する必要がある．このため，GC が起こると，GHC のプロセスだけでなく，RBU のプロセスも止まることになる．そこで，以下の評価用の実験では，GC が起こらないよう十分広いヒープ領域を確保して行った．

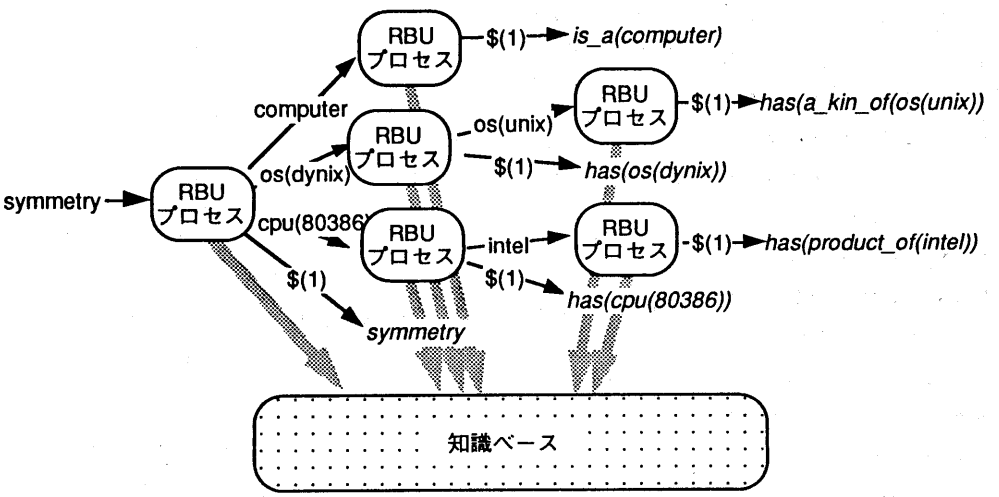

図4　並列検索の例

表2　プロセッサ台数と実行時間
（インデックスを張らない場合）　　単位：秒

	RBU:1	RBU:2	RBU:3	RBU:4	RBU:5	RBU:6
GHC:1	4.239	3.207	3.178	3.198	3.126	3.154
GHC:2	4.080	2.362	1.870	1.837	1.822	1.830
GHC:3	3.997	2.261	1.677	1.484	1.405	1.384
GHC:4	3.986	2.219	1.624	1.403	1.257	1.227
GHC:5	3.976	2.176	1.613	1.359	1.217	1.143
GHC:6	3.969	2.165	1.600	1.324	1.170	1.097
GHC:7	3.964	2.159	1.585	1.301	1.158	1.081
GHC:8	3.966	2.170	1.600	1.307	1.148	1.070
GHC:9	3.963	2.197	1.562	1.308	1.125	1.061
GHC:10	3.966	2.192	1.584	1.321	1.120	1.061

表3　プロセッサ台数と実行時間
（インデックスを張った場合）　　単位：秒

	RBU:1	RBU:2	RBU:3	RBU:4	RBU:5	RBU:6
GHC:1	2.983	2.959	3.033	3.024	3.018	2.977
GHC:2	1.589	1.548	1.553	1.576	1.582	1.600
GHC:3	1.117	1.096	1.080	1.115	1.090	1.115
GHC:4	0.871	0.838	0.846	0.833	0.846	0.850
GHC:5	0.747	0.687	0.689	0.685	0.678	0.705
GHC:6	0.678	0.595	0.582	0.584	0.597	0.602
GHC:7	0.652	0.532	0.523	0.521	0.537	0.514
GHC:8	0.623	0.484	0.476	0.468	0.479	0.468
GHC:9	0.627	0.455	0.441	0.433	0.430	0.423
GHC:10	0.636	0.440	0.401	0.394	0.390	0.390

プロセッサ台数と実行時間の関係を，項関係の第1属性にインデックスを張った場合と張らない場合について，それぞれ表2と表3に示す．表の横方向にRBUのプロセッサ数を，縦方向にGHCのプロセッサ数を変化させている．この表から，インデックスがない場合は，検索処理が全体の処理時間を制御することになり，検索モジュールのプロセッサを増やさないと，推論モジュールのプロセッサを増やしても処理速度が向上しないことがわかる．逆に，インデックスを張った場合には，推論処理が全体の処理時間を制御していることがわかる．

処理速度の目安として，現在最も広く使われているPrologの処理系の1つであるQuintus Prologのインタープリタと比較する．比較対象をインタープリタとしたのは，知識の更新を対話的にできるものを対象としたためである．

同じ内容の知識ベース（556個分のホーン節）をSUN3/60上のQuintus Prolog 1.6上に乗せ，setofで同様の問い合わせで検索した場合の実行時間は，11.283　秒であった．SUN3/60　とSymmetryの1プロセッサの処理速度がDhrystone 1.1で比較すると，オプションなしの場合でSUN3/60が3582，Symmetryが4258であったことから，Symmetry　上の1プロセッサでQuintus Prolog 1.6を使って実行した場合を想定すると，9.492秒かかる計算になる．

この値を1とした場合に，試作システムでインデックスを張らなかった場合（表3）と，張った場合（表4）のプロセッサ数の変化に対する処理速度比の推移グラフをそれぞれ図5と図6に示す．処理性能を抑えているモジュールを明確にするために，図5ではGHCプロセッサ台数を固定してRBUプロセッサ台数による変化を折れ線とし，図6ではRBUプロセッサ台数を固定してGHCのプロセッサ台数による変化を折れ線とした．

インデックスを張らない場合は並列知識検索モジュール側の，張った場合には並列推論モジュール側のプロセッサ割り当てを増やすことにより，プロセッサ台数に対応して直線的に処理速度が向上することがわかる．このように，検索と推論の処理負荷の比率に合わせてプロセッサ台数を増やすことが重要である．ただし，この比率は知識ベースの大きさや問題の性質に依存するため，問題によってカスタマイズする必要がある．

6．おわりに

知識ベースが利用できる並列処理環境として，並列推論モジュールと並列知識検索モジュールからなる知識ベース指向並列処理システムKOPPS を提案し，その構成と，マルチマイクロ計算機上に試作したシステムを用いた意味ネットワーク検索の実験について報告した．実験の結果，検索と推論の負荷バランスを取りながら並列化することにより，十分実用になる性能で処理可能であることを示した．

我々は，ここで報告した意味ネットワークの検索以外にも，SLD演繹[11]やプロダクション・システムの実行をKOPPS 上で行い，良好な結果を得ている．これらより，KOPPS を並列マシン上での知識検索を伴う並列処理の実行環境として利用できる見通しを得ることができた．

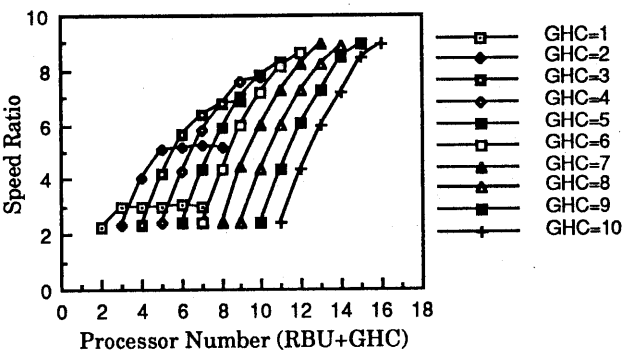

図５　プロセッサ台数による処理速度比の推移
（インデックスを張らない場合）

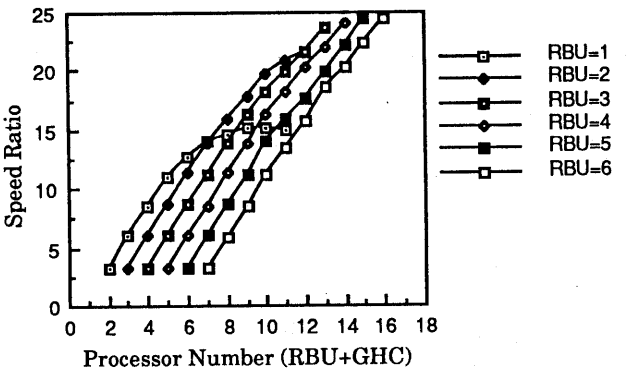

図６　プロセッサ台数による処理速度比の推移
（インデックスを張った場合）

謝辞

日頃ご指導をいただく ICOT 内田 第４研究室長，ICOT KL1-TG メンバ，富士通研究所 林 人工知能研究部長，試作に御協力頂いた人工知能研の小沢，細井両氏，ならびに富士通 SSL の山崎，北島，竹中の３氏に感謝します．

参考文献

1. K. Ueda, "Guarded Horn Clauses," *Logic Programming '85*, E. Wada (ed). Lecture Notes in Computer Science 221, Springer-Verlag, 1986.
2. 淵一博監修，古川康一，溝口文雄供編，並列論理型言語 GHC とその応用，知識情報処理シリーズ第６巻，共立出版，1987.
3. K. L. Clark and S. Gregory, "PARLOG: Parallel Programming in Logic," *ACM Trans. on Programming Language System*, Vol. 8, No. 1, pp.1-49, 1986.
4. E. Y. Shapiro, "Concurrent Prolog: A Progress Report," *Computer*, Vol. 19, No. 8, pp.44-58, 1986.
5. 特集：演繹データベース，情報処理学会誌，Vol. 31, No. 2, 1990.
6. H. Yokota and H. Itoh, "A Model and Architecture for a Relational Knowledge Base," *Proc. of the 13th Int'l Sympo. on Computer Architecture*, pp. 2-9, 1986.
7. H. Yokota, H. Kitakami, and A. Hattori, "Term Indexing for Retrieval by Unification," *Proc. of 5th Int'l Conf. on Data Engineering*, pp.313-320, 1989.
8. 北上，横田，服部：知識処理向き並列推論エンジン，電子情報通信学会研究会資料，CPSY, 88-50, 1988.
9. 小山，田中：Definite Clause Knowledge Representation, *Proc. of the Logic Programming Conference '85*, pp.95-106. 1985.
10. 小沢，細井，服部：FGHC処理システムのメモリ使用特性と世代別ガーベージ・コレクション，情報処理学会論文誌，Vol. 30, No. 9, pp.1182-1188, 1989.
11. J. W. Lloyd, *Foundation of Logic Programming*, Springer-Verlag, 1984.