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††

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．RBU は知識を項（変数を含んだ構造体）で表現し，その集合を関係（テーブル）として管理し，単一化 (Unification) と呼ばれる高機能なパターンマッチング機能を使って検索するモデルである．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．試作システムでは，RBU の機構として大量の知識に対し効率よく検索するための専用インデックスと，検索コマンド単位で並列に処理するための同時実行メカニズムを採用している．また，GHC と RBU とは共有メモリ領域を使って結合され，ストリーム的に通信が行われる．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．

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，システムを使いやす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．

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，並列知識検索モジュールと，並列推論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，知識ベースの管理については専用の並列知識検索モジュールを用意し，そこで検索された知識を並列言語で高速に処理をしながら全体として推論を進めるシステムである．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．

このような検索機構と推論機構の結合は，演繹データベース・システム[1]のアプローチに近い．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はかっている．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，構造を持つ知識を格納することを前提としてない．つまり，知識を表現するための手段としての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．例えば，構造体や変数を文字列として格納したのでは，効率的な検索機能の提供は困難である．また，並列処理の点からは，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[2]．

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，検索することにより効率の良い知識ベース管理を実現することを目指す．さらに，検索も推論も並列に実行して高速化をはかることを重視する．このため，知識ベースのモデルとして関係データベースモデルを拡張した RBU[3] を採用し，その並列検索機構を実現する．格納対象をより一般的に拡張したオブジェクト指向データベースの研究[4]なども盛んに行われているが，我々は各知識の持つ構造に注目した場合の知識ベース処理を対象とする．

システム全体を並列に動作させるために，並列言語によって検索/更新の発令処理や検索結果の解析処理等を記述する．並列言語処理系としては，ストリームベースで論理型指向の GHC[5),6)], Parlog[7], Concurrent Prolog[8], オブジェクト指向の ABCL/1[9], Concurrent Smalltalk[10], 並列 Lisp の MultiLisp[11]など数多くの処理系が提案されている．KOPPS のように

† Knowledge-base-Oriented Parallel-Processing System by HARUO YOKOTA, HAJIME KITAKAMI and AKIRA HATTORI (FUJITSU LIMITED).

†† 富士通(株)

* 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．

2つのモジュールを結合して並列システムを構成する場合には，ストリームベースで処理を記述する方法が適している．さらに，RBU で扱う知識の構造が述語論理における項そのものであるため，論理型指向の言語と相性がよい．このため，並列論理型言語の GHC で並列推論モジュールを記述することにした．

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 で知識ベースのモデルとして採用した 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[3]．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[12]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．RBU 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．このようなリアルタイムの更新機能は，KOPPS 上でプロダクション・システムを実現したり，学習などの研究をする場合には，欠くことができないものである．

並列推論モジュールの処理を記述する 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．並列に動作させたい処理単位の制御を記述するのに適した言語と言える．そこで，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 に渡し，システム全体が並列に動作するように制御する．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[6]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

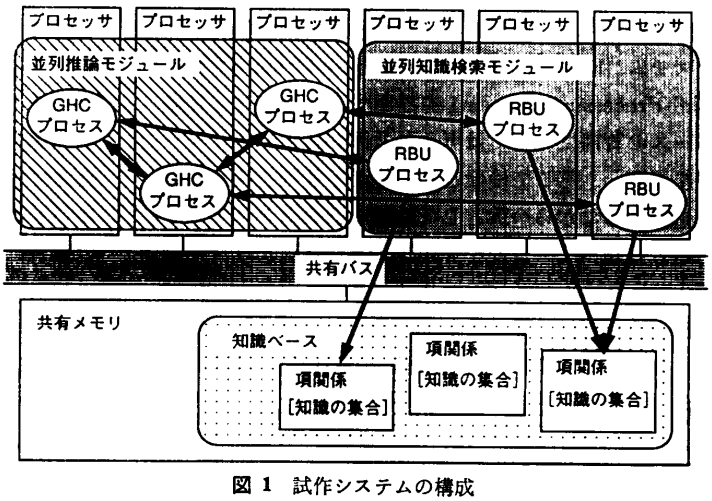

図 1 試作システムの構成
Fig. 1 A multiprocessor-based system configuration.

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 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．このとき，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はない．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，検索処理か更新処理かによって，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．

また，項関係の中で検索条件を満足する構造体をしぼり込むため，トライ（Trie）構造[13]とハッシュを用いた専用インデックスと，その上での単一化機能を実現している．このインデックスにより，検索が高速化されるだけでなく，更新時の維持コストを低く抑えることができる．インデックスおよび単一化機構の詳細については，文献 12) を参照されたい．

並列推論モジュールと並列知識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を渡すため，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．なお，将来的には並列推論モジュールで使うデータ構造全体も知識ベースとして捕え，推論も知識ベースも1つの枠組みの中で扱う方法も考えられる．

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．つまり，並列知識検索モジュールの各プロセスは，検索/更新処理中でない場合，自分に対応するコマンドページに書き込まれたコマンドを優先的に読み取って実行するほか，自分に対応するコマンドページにコマンドが書き込まれていない場合には，他のコマンドページにコマンドを探しに行くことができる．また，並列推論モジュールは，コマンドを書き込む時点で書き込まれているコマンド数がなるべく少ないコマンドページを選択する．

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．意味ネットワークは，ノード数を増減することにより知識ベースのサイズを調整することができる上，比較的容易に容量の大きな意味のある知識ベースを準備することができる．ここでは，**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ことにする．このような計算機に関する知識ベースは，各種の計算機が LAN を介して多数接続されるようになった昨今，計算機環境を把握したり維持する上で重要になってきていると考える．

図2の知識を項関係に格納したものを**表1**に示す．ここで，$(1)や $(2)は，RBU 内で使われる論理変

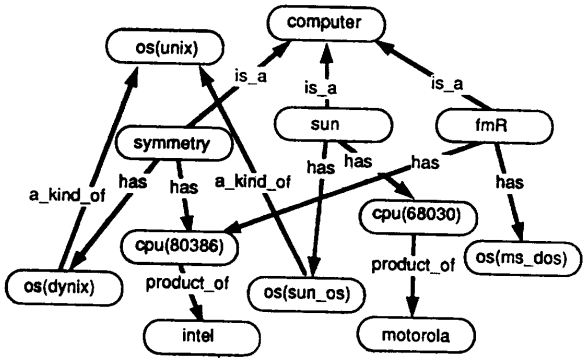

図 2 計算機に関する知識の意味ネットワーク表現の一部
Fig. 2 A partial semantic network knowledge-representation of computers.

表 1 図2の知識の項関係
Table 1 A term relation for the knowledge of Fig. 2.

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,RBU1,OUT1) :- Q1 ≠ nil |
    RBU1 = [urs(KB,[1=Q1,2=Q2,5=Q3,6=Q4],[3,4,5,6],S)|RBU2],
    S = [_,_|_],
    search_loop(S,KB,RBU3,OUT2),
    L = [_,_|_],
    search_loop(L,KB,RBU4,OUT3),
    merge(RBU3,RBU4,RBU2),
    merge(OUT2,OUT3,OUT1).
search_loop([[nil,empty,Q1,Q2]|L],KB,RBU,OUT1) :- true |
    OUT1 = [writeterm([Q1,Q2])|OUT2],
    L = [_,_|_],
    search_loop(L,KB,RBU,OUT2).
search_loop([-1,_|L],KB,RBU,OUT) :- true |
    L = [],
    RBU = [],
    OUT = [].
```

並列知識検索モジュール・インタフェース用組込述語
入出力用組込述語
RBUのコマンド（単一化制約）
ダブルバッファリングを使った要求駆動用リスト
ストリームのマージ
出力条件判定
終了条件判定

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
Fig. 3 A sample program for traversing the semantic network.

数を表し，メタとオブジェクトの間の変数の区別を行っている[14]．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[15] の方法と同様のものである．

このとき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[3]）．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

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．この結果のうち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．

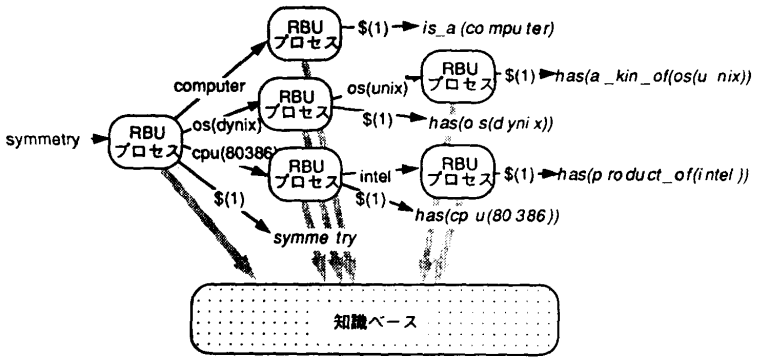

図 4 並列検索の様子
Fig. 4 Parallel traversal of the semantic network.

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，ほとんどすべ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．

このほか，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 sun 3 シリーズの has 関係にだけ注目して，

traverse(kb, sun(3, $(20)), has($(30))).

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 556 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（64 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．

このとき，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[16]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．

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．

単に実行時間だけでは対象とした問題の難易度やサイズを示すことができないため，処理速度の目安として，完成度が高いと言われ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

表 2 プロセッサ台数と実行時間（単位：秒）
（インデックスを張らない場合）
Table 2 Execution times corresponding to number of processors (without indexing).

	RBU:1	RBU:2	RBU:3	RBU:4	RBU:5	RBU:6
GHC:1	4.24	3.21	3.18	3.20	3.13	3.15
GHC:2	4.08	2.36	1.87	1.84	1.82	1.83
GHC:3	4.00	2.26	1.68	1.48	1.41	1.38
GHC:4	3.99	2.22	1.62	1.40	1.26	1.23
GHC:5	3.98	2.18	1.61	1.36	1.22	1.14
GHC:6	3.97	2.17	1.60	1.32	1.17	1.10
GHC:7	3.96	2.16	1.59	1.30	1.16	1.08
GHC:8	3.97	2.17	1.60	1.31	1.15	1.07
GHC:9	3.96	2.20	1.56	1.31	1.13	1.06
GHC:10	3.97	2.19	1.58	1.32	1.12	1.06

表 3 プロセッサ台数と実行時間（単位：秒）（インデックスを張った場合）
Table 3 Execution times corresponding to number of processors (with indexing).

	RBU:1	RBU:2	RBU:3	RBU:4	RBU:5	RBU:6
GHC:1	2.98	2.96	3.03	3.02	3.02	2.98
GHC:2	1.59	1.55	1.55	1.58	1.58	1.60
GHC:3	1.12	1.10	1.08	1.12	1.09	1.12
GHC:4	0.87	0.84	0.85	0.83	0.85	0.85
GHC:5	0.75	0.69	0.69	0.69	0.68	0.71
GHC:6	0.68	060	0.58	0.58	0.60	0.60
GHC:7	0.65	0.53	0.52	0.52	0.54	0.51
GHC:8	0.62	0.48	0.48	0.47	0.48	0.47
GHC:9	0.63	0.46	0.44	0.43	0.43	0.42
GHC:10	0.64	0.44	0.40	0.39	0.39	0.39

ンタプリタと比較する．比較対象をインタプリタとしたのは，本システムが知識の対話的な更新を1つの特徴とし，更新の度にコンパイルし直すコンパイラでは比較の対象とならないためである．また，すべてをGHC のみで記述したものと比較も考えられるが，現実的には不可能である．これは，GHC で同様の機能を実現するためには単一化処理自身をGHC で記述し，知識ベース全体をリスト等で保持しながら永久プロセス[6]で検索モジュールを実現する必要がある[14]ため，実行速度が比較にならないほど遅くなる上，メモリ使用量の関係から同程度のサイズの問題ではメモリ容量不足となり実測できないためである．

同じ内容の知識ベース（556 個分のホーン節）を SUN 3/60 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，11.28 秒であった．SUN 3/60 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 SUN 3/60 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，9.49 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2）と，張った場合（表3）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．ここで用いた例の場合，インデックスを張らないときには並列知識検索モジュール2に対して並列推論モジュール1，インデックスを張ったときには並列知識検索モジュール1に対して並列推論モジュール4程度が適当である．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．

また，インデックスを張ってない場合に，ある程度以上並列知識検索モジュールのプロセッサを増やすと，並列推論モジュールの台数比を上げても一様に処

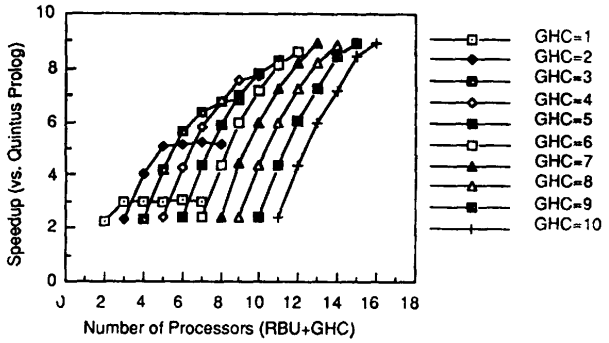

図 5 プロセッサ台数による処理速度比の推移（インデックスを張らない場合）
Fig. 5 Speedup versus number of processors (without indexing).

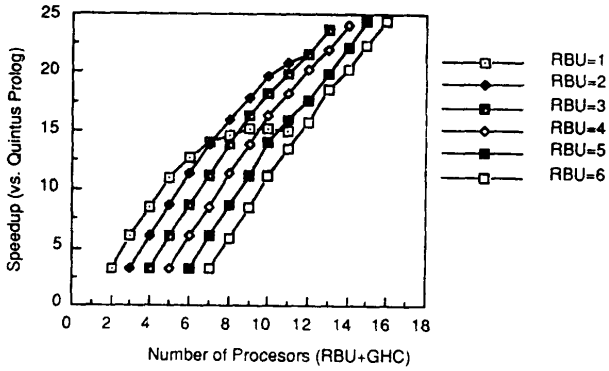

図 6 プロセッサ台数による処理速度比の推移（インデックスを張った場合）
Fig. 6 Speedup versus number of processors (with indexing).

理速度の向上率が低下している．この原因としては，インデックスが張ってない場合にはすべてのタプルを読み込む必要が生じるため，並列キャッシュのヒット率が下がり，共有メモリのアクセスネックになっている可能性が考えられる．並列知識ベース処理の場合，共有メモリのアクセス頻度を下げる上でも，ますますインデックスの重要性が高くなるといえる．

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[17]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．

謝辞 日頃ご指導をいただく内田俊一 ICOT 研究部長，ICOT KL 1-TG メンバ，林 弘 富士通研究所人工知能研究部長，試作にご協力いただいた人工知能研究部の小沢，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) Gallaire, H., Minker, J. and Nicolas, J-M.: Logic and Databases: A Deductive Approach, *ACM Comput. Surv.*, Vol. 16, No. 2, pp. 153-185 (1984).
2) 特集：演繹データベース，情報処理，Vol. 31, No. 2 (1990).
3) Yokota, H. and Itoh, H.: A Model and an Architecture for a Relational Knowledge Base, *Proc. of the 13th Int'l Symp. on Computer Architecture*, pp. 2-9 (1986).
4) Kim, W., Ballou, N., Chou, H.T., Garza, J.F. and Woelk, D.: Integration and Object-Oriented Programming System with a Database System, *Proc. of Object Oriented Programming Systems, Languages and Applications '88*, pp. 142-152 (1988).
5) Ueda, K.: Guarded Horn Clauses, *Logic Programming '85*, Wada, E. (ed.), *Lecture Notes in Computer Science 221*, Springer-Verlag (1986).
6) 淵 一博監修，古川康一，溝口文雄共編：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，p. 277，共立出版 (1987).
7) Clark, K. L. and Gregory, S.: PARLOG: Parallel Programming in Logic, *ACM Trans. Prog. Lang. Syst.*, Vol. 8, No. 1, pp. 1-49 (1986).
8) Shapiro, E. Y.: Concurrent Prolog: A Progress Report, *Computer*, Vol. 19, No. 8, pp. 44-58 (1986).
9) Yonezawa, A., Shibayama, E., Takada, T. and Honda, Y.: Modeling and Programming in an Object-Oriented Concurrent Language ABCL/1, *Object-Oriented Concurrent Programming*, Yonezawa, A. and Tokoro, M. (eds.), pp. 55-90, MIT Press (1988).
10) Yokote, Y. and Tokoro, M.: Concurrent Programming in Concurrent Smalltalk, *Object-Oriented Concurrent Programming*, Yonezawa, A. and Tokoro, M. (eds.), pp. 129-158, MIT Press (1988).
11) Halstead, R.: MultiLisp: A Language for Concurrent Symbolic Computation, *ACM TOPLAS*, Vol. 7, No. 4, pp. 501-538 (1985).
12) Yokota, H., Kitakami, H. and Hattori, A.: Term Indexing for Retrieval by Unification, *Proc. of 5th Int'l Conf. on Data Engineering*, pp. 313-320 (1989).
13) Knuth, D.E.: *The Art of Computer Programming, Vol. 1, Fundamental Algorithm*, p. 634, Addison-Wesley (1973).
14) 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).
15) 小山，田中：Definite Clause Knowledge Representation, *Proc. of the Logic Programming Conference '85*, pp. 95-106 (1985).
16) 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).
17) Lloyd, J. W.: *Foundation of Logic Programming*, p. 124, Springer-Verlag (1984).

(平成2年3月12日受付)
(平成2年9月11日採録)

横田　治夫（正会員）

1957 年生．1980 年東京工業大学工学部電子物理工学科卒業．1982 年同大学院情報工学専攻修士課程修了．同年 4 月，富士通(株)入社．同年 6 月より(財)新世代コンピュータ技術開発機構に出向．1986 年 4 月(株)富士通研究所に帰属．データベースマシン，演繹データベース，知識ベースシステム，並列記号処理の研究・開発に従事．電子情報通信学会，人工知能学会各会員．

北上　　始（正会員）

1952 年生．1976 年東北大学大学院修士課程修了．同年富士通(株)入社．1978 年～82 年(株)富士通研究所において関係データベース管理システムの研究・開発に従事．1982 年 6 月～85年 5 月(財)新世代コンピュータ技術開発機構において知識ベース管理システムの研究に従事．1985 年 6 月(株)富士通研究所に帰属．静止衛星の姿勢制御ソフトウェアの開発，関係データベース管理システムの研究・開発，知識ベース管理システムの研究．日本認知学会，人工知能学会各会員．

服部　　彰（正会員）

昭和 24 年生．昭和 47 年大阪大学工学部電子工学科卒業．昭和 49 年同大学院工学研究科修士課程修了．同年(株)富士通研究所入社．階層メモリ，記号処理マシンの研究を経て，並列コンピュータの研究開発に従事．現在，人工知能研究部第 3 研究室長．電子情報通信学会，人工知能学会各会員．