TR-164

核言語KL1ソフトウェアシミュレータの試作

大原有理，鳥井　悟，小野越夫，岸下　誠
(富士通)
田中二郎，宮崎敏彦
(ICOT)

April, 1986

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

核言語ＫＬ１ソフトウェアシミュレータの試作

大原有理　鳥居悟　小野越夫　岸下誠　　田中二郎　宮崎敏彦
富士通　株式会社　　ＩＣＯＴ

はじめに

ＫＬ１［古川　85］は，５Ｇプロジェクト中期において開発される並列推論マシン（ＰＩＭ）の機械語として位置付けられる言語系であり，並列推論，同期メカニズム，並列処理等の機能を持つ．

ＫＬ１の並列推論機能のベースとして，初め Concurrent Prolog［Shapiro 83］が採用された．しかし，いくつかの問題点が指摘され［上田　85a］，それにかわってConcurrent Prolog と同等の記述力を持つ言語モデルＧＨＣ(Guarded Horn Clauses)［上田　85b］を採用することになった．

現在ＧＨＣの処理系としては，

① Dec-10 Prolog 上のＧＨＣ処理系［古川　85］

② ＰＳＩ上のＧＨＣサブセット逐次型処理系［江崎　86］

等があり，

マルチプロセッサ上の分散処理系の実行モデルとして［村上　85］の方式が提案されている．

今回筆者は，［村上　85］を参考にＧＨＣを簡単化したＦＧＨＣ（Ｆlat ＧＨＣ）を分散環境下で実行する処理系，分散処理系の検討を行った．この検討は，ＫＬ１の並列実行モデルの明確化を目指したものであり，ＫＬ１の中核をなすＫＬ１－Ｃ（ＦＧＨＣ）の分散環境下での実行モデルを，分散ユニフィケーション，ゴール管理方式，メモリ参照方式等の面から検討を行った．

また，それらの動作原理の確認のため，逐次計算機上にＫＬ１の分散処理系ソフトウェアシミュレータを試作した．このシミュレータは，ＫＬ１－Ｃプログラムを実行コードに変換するコンパイラと，その実行コードに基づいてシミュレーションを行う実行系からなる．

このソフトウェアシミュレータの特徴としては，

① ＦＧＨＣプログラムにゴール割付の指示（以後，プラグマと呼ぶ）を加えたこと

② 任意個のＰＥ：Processing Elementからなる分散ユニフィケーションが可能であること

③ ＦＧＨＣの持つ並列性，同期メカニズムに適した分散ユニフィケーション，リダクション・アルゴリズムを採用していること

④ ゴールのサスペンドは，変数に bind hookする方式を採用していること

⑤ 実行コードは，分散ユニフィケーションのための特別なコードを加えた Prolog コードとしていること

などがある．

本論文は，分散環境下での実行に起因する変数の共用，分散ユニフィケーション，リダクションを中心に分散処理系の動作原理，ソフトウェアシミュレータの実現方式について述べ，そしてシミュレーションの結果について述べる．

1　並列論理型言語

並列論理型言語ＫＬ１－Ｃとしては，ＧＨＣを簡単化した，ＦＧＨＣを採用している．

ＦＧＨＣでは，ガードを持つホーン節によって述語が定義され，ゴールはand 並列に実行される．また，ゴール間の同期メカニズムが提供される．

さらに今回は，複数のＰＥのうち，どのＰＥにゴールを割付け，実行させるかを示すプラグマを付加している．

1.1　ＫＬ１－Ｃ

ＧＨＣの節形式を以下に示す．

```
H(Arg1, Arg2, ···, Argk) :- G1, G2, ···, Gm | B1, B2, ···, Bn .
  Head 部              Guard 部        Body 部
      passive part                  active part
```

ＧＨＣは，passive partの実行において，アーギュメント変数つまり呼び出し元の変数に値を設定するようなユニフィケーションは，行うことができない．このようなユニフィケーションは，active part に組込み述語 " = " を陽に呼び出して行わなければならない，という特徴を持っている．

今回ＫＬ１－Ｃとして採用したＦＧＨＣは，このＧＨＣに対し，以下のような特徴，評価規則を持っている．

- Guard 部内では（一部を除く）組込み述語しか書けない．
- passive partは，左から右に逐次的に実行される．
  (head unification -> G1 -> G2 -> ... -> Gm)
- head unificationは，各引数を左から右に逐次実行される．
- 候補節のpassive partは，上から下に逐次的にテストされる．
  但し，ある候補節のテストがsuspend もしくは fail した時には，次の候補節もテストされる．
- サスペンドしていた複数の候補節が resume された時には，一番上の候補節から再びテストを行う．
- or並列は行わない．
- 逐次orは行わない．

ただし今回は，meta-call-frozen-call は考えておらず，メタ述語も取り入れていない．

1.2　プラグマ

プラグマは，分散処理用のゴールの割付の指示を与えるものである．今回は，ＫＬ１－Ｃプログラムにおいて，どのＰＥにゴールを割付けるか，投けるかを，ユーザが陽にゴールとして指定するものとした．

指定形式を以下に示す．

throw(ゴール群, [up / down / left / right])

このプラグマは，実行時にこの（プラグマ）ゴールを処理しているＰＥの（ネットワーク上）上（up），下（down），左（left），右（right）のＰＥに指定したゴール群を投げることを意味する．

2　分散処理系モデル

分散処理系モデルは，複数のＰＥから成り，個々のＰＥが並列に実行する．ＰＥ間は，ネットワークでつながれており，さらに各ＰＥの中には，ゴールを管理するゴールキューが存在している．

2.1　分散処理系の基本モデル

分散処理系はその性格上，ＫＬ１の処理系を内蔵し，ＫＬ１－Ｃプログラムを受け付け，その実行を行う．

分散処理系について検討する際の基本モデルとして，次のようなものを想定した．

・分散処理系は，ＫＬ１処理系を内蔵するＰＥがネットワークで結合した構成である．
・各ＰＥは，２本のチャネル（in/out）によってネットワークとつながれており，ＰＥ間のメッセージの送受信は，ネットワークを経由して行われる．
・各ＰＥは並列動作し，さらにＰＥ内部では複数ゴールが並列実行する．
・各ＰＥはローカルにメモリを持っており，グローバルなメモリは存在しない．
・プログラム（述語定義）は，全てのＰＥが同一のもののコピーを持つ．

各ＰＥは同期のつながったグリッド状に配置されているものと仮定した．ＰＥの数に制限はないが，以降では，説明を簡単にするため，ＰＥの数を９台とする．

図２－１に分散処理系の基本モデルのイメージを示す．

2.2　分散処理系の基本構成

図２－２に示すように，本分散処理系は複数のＰＥ（ＰＥ＃１～ＰＥ＃９）とＰＥ間のメッセージを管理するネットワークマネージャから構成される．

各ＰＥは，入力(in)と出力(out)の２本のチャネルによって，ネットワークマネージャにつながれている．

ネットワークマネージャは，各ＰＥの出力チャネルに置かれたメッセージを回収し，そこに記述されている通信先に従って，対応するＰＥの入力チャネルにそのメッセージを置く．これによって，ＰＥ間のメッセージ交換が行われる．

ＰＥにおける処理の単位はゴールである．ゴールキューにスケジューリングされることによって，ゴールは実行可能な状態になる．

本分散処理系においては，ＰＥ＃１に解かれるべきゴールを渡すことから，処理が開始する．そして，処理が進むに並行して，他のＰＥに（サブ）ゴールがばらまかれて行くことになる．

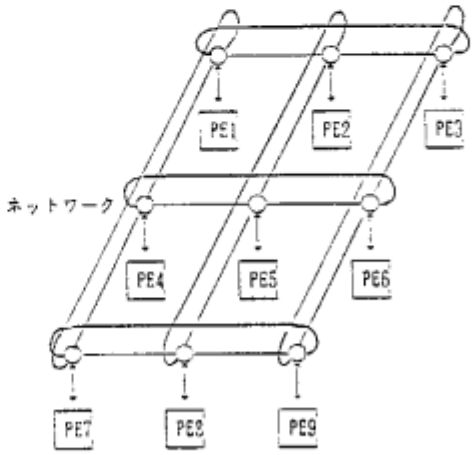


図２－１　分散処理系の基本モデル

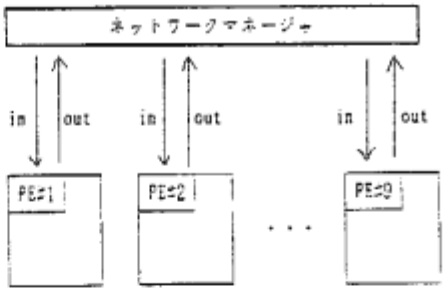


図２－２　分散処理系の基本構成

3　分散処理系における処理の概要

分散処理系は，ＫＬ１－Ｃプログラムを入力し，分散環境下で実行するものである．

分散処理系の設計においては，以下の基本アルゴリズムを明確にする必要がある．

①　ゴールの送受信，メッセージ通信等のＰＥ間通信
②　ゴールスケジューリング，ゴール実行等のＰＥ内での処理

本章では，これらについて述べる．

3.1　ＰＥ間通信

分散処理系においては，図３－１に示すように，複数のＰＥがＰＥ間で入出力チャネルを介してメッセージを交換しながら処理を行って行くことによって，全体としての機能が実現される．

複数のＰＥ間での複数の共用にはいくつかの方法が考えられるが，

ここでは［村上 85］と同様に，ＰＥのIDとそのＰＥ中の変数によるメモリ参照方式を想定して，検討を進めることにした．

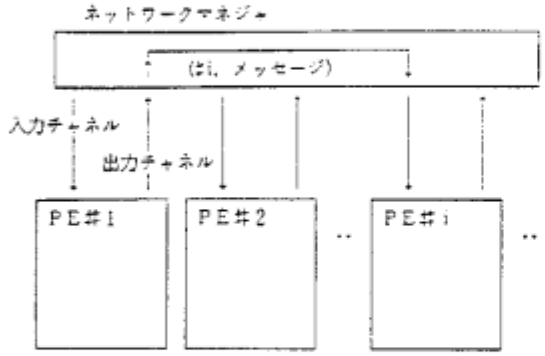
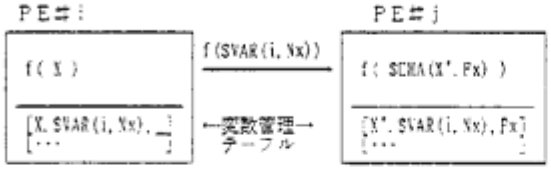


図３－１　ＰＥ間のメッセージ通信

(1)　ゴールの送受信

あるＰＥが他ＰＥへゴール実行を依頼する時には，ネットワークを経由してゴールを送る．ゴールを外に送ることを export すると言い，ゴールを受け取ることを import すると言う．

ゴールを export するＰＥは，そのゴールが含む変数を，一意に識別可能なチャネル通過用の形式に変換して転送する．import側のＰＥは，そのゴールが含むチャネル通過用の形式を，自ＰＥ内で処理可能なチャネル変数に変換する．チャネル変数は，チャネルを通過してきたことを表す特別な形式を持っている．変数とチャネル通過用の形式との対応は，ＰＥ毎に一つずつ存在する変数管理テーブルによって管理される．この変数管理テーブルは，そのＰＥが export 及び import した変数とチャネル通過用の形式との対応を保持する．以下に例を用いて詳述する．



注）変数管理テーブルのエントリの第３フィールドは，通信元の時は使用されず，チャネル変数の時は，そのチャネル変数の値を問い合せ中であることを示す問い合せフラグとして使用される．

図３－２　チャネル変数の例

図３－２はＰＥ＃iからＰＥ＃jへゴール f(X) を送信する例である．ここで，X は，他ＰＥからimportしたチャネル変数ではなく，普通の変数であるとする．

ＰＥ＃i は，exportするゴールが含むすべての変数をチャネル通過用の形式 SVAR(自PEのID，PE内変数番号) に変換し，変数管理テーブルには［変数，チャネル通過用の形式，_］の形式で格納する．つまり，変数 Xがチャネル通過用の形式 SVAR(i, Nx) に変換され，変数管理テーブルに［X, SVAR(i, Nx), _］が格納される．そして，ゴールが f(SVAR(i, Nx)) に変換され，ＰＥ＃j に送られる．

ＰＥ＃j は，importしたゴールが含むすべてのチャネル通過用の形式から新たにローカル変数を生成し，変数管理テーブルに［新ローカル変数，チャネル通過用の形式，問合せフラグ］の形式で格納する．さらに，ゴール内のチャネル通過用の形式は，チャネル変数 SCHA(新ローカル変数，問合せフラグ) の形に置き換えられる．つまり，チャネル通過用の形式 SVAR(i, Nx) からローカル変数 X' が生成され，変数管理テーブルに［X', SVAR(i, Nx), Fx］が格納される．また，チャネル変数 SCHA(X', Fx)が作成され，ゴールは f(SCHA(X', Fx)) となる．

以上の処理によって，ＰＥ＃j上のチャネル変数とＰＥ＃i上の変数とがリンクづけられる．

ＰＥ＃jからＰＥ＃iにチャネル変数 SCHA(X', Fx)の値を要求する場合，ＰＥ＃jにおいて，変数管理テーブルからそのチャネル変数に対応するチャネル通過用の形式 SVAR(i, Nx) を得て，それをＰＥ＃i に送る．ＰＥ＃i では，送られてきたチャネル通過用の形式に対応する変数 Xを変数管理テーブルから得る．

(2)　メッセージ通信

前述の通り，ゴールの送受信，分散ユニフィケーション等は，ネットワークを介して複数のＰＥ間で行われる．このネットワークを流れるメッセージの形式として，以下の５つのプリミティブを考えた．

他ＰＥに対する要求は，①～④で表したプリミティブの形に変換され，ネットワークを介して要求先ＰＥに送られる．要求先ＰＥは，これらのプリミティブで表された要求をゴールとして処理し，その結果を⑤のプリミティブによって要求元ＰＥに通知する．これによって，ＰＥ間のメッセージ通信が実現される．

①　チャネル変数の値を要求するプリミティブ

・get_value (Ch_Var, Result, Return_Value)

チャネル変数Ch_Varの元のＰＥに対し，Ch_Varの値を要求する．値は reply_result によって，Return_Valueに返される．

②　ゴールを転送するプリミティブ

・send_goal (送り先ＰＥ, Goal, Result)

Goalで示されるゴール群を送り先ＰＥで示されるＰＥに送り，実行を依頼する．

③　ユニファイを要求するプリミティブ

・unify (Ch_Var, Value, Result)

チャネル変数Ch_Varの元のＰＥに対し，Ch_Varと値 Valueとのユニフィケーションを要求する．

ユニフィケーション結果 (succeeded/failed) は reply_result によって，Resultに返される．

④　チャネル変数同士のユニフィケーションを要求するプリミティブ

・unify_channels (Ch_Var1, Ch_Var2, Value, Result)

チャネル変数 Ch_Var1の元のＰＥに対し，Ch_var2 とのユニフィケーションを要求する．

ユニフィケーション結果 (succeeded/failed) とチャネル変数の値は reply_result によって返される．

⑤　要求された処理の実行結果を通知するプリミティブ

・reply_result（〔Result，Result値〕，〔Value，Value値〕）

要求元のＰＥに対し，要求された処理の実行結果を通知する．Result値には実行結果，Value 値には値（①，④の時）が返される．

3.2　ＰＥ内処理

ＰＥにおける処理の単位はゴールであり，ゴールキューにスケジュールされたゴールを疑似的に並列に実行することで，マルチプロセスをサポートする．以下にＰＥ内での処理について述べる．

(1)　ゴールのスケジューリング

各ＰＥにおいて，ゴールは，ゴールキューに適当なスケジューリング・アルゴリズムに基づいてスケジューリングされることによって，実行可能状態となる．

今回は，ＰＥ毎に１つのゴールキューが存在するものとした．「ゴールキュー内のゴールを先頭から順次取り出して実行し，その結果リダクションされたゴールをゴールキューにキューイングし直す」ということを繰り返し行う．この様子を図３－３に示す．

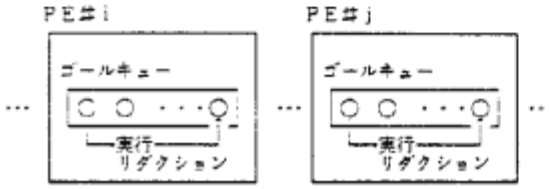


図３－３　ゴールのスケジューリング

但し，他ＰＥから要求されたプリミティブゴールは，ゴールキューの先頭につながれ，先に実行される．

(2)　サスペンド

ゴール実行が何らかの原因でサスペンドすると，そのゴールはサスペンド原因となった変数に hook される．つまり，そのサスペンド原因となった変数 Xを，susp(X'，〔g(X')，NEXT〕）の形に実体化する．ここで，〔g(X')，NEXT〕は，サスペンドゴールリストを表す．g(X') は，サスペンドしたゴールg(X)のX をX'に置き換えたものであり，NEXTは，サスペンドゴールリストのリンクを表す．

変数 Xを実体化しようとする時，hookされているゴールのサスペンドが解かれ，ゴールキューにつなげられる．

既にサスペンド原因となっている変数が原因で他のゴールがサスペンドすると，そのゴールはその変数に hook されているサスペンドゴールリストに追加される．

4　ソフトウェアシミュレータ

これまでに述べた分散処理系の設計及び動作原理の検討・確認のため，Prologを使用して逐次計算機上にソフトウェアシミュレータを試作した．

本ソフトウェアシミュレータは，ＫＬ１－Ｃプログラムを逐次計算機上に構成した分散環境下で実行し，その実行結果及びトレース情報を出力するものである．

入力であるＫＬ１－Ｃプログラムは，並列性等の特徴を持っており，これを逐次計算機で直接実行することは難しい．そこで，入力ＫＬ１－Ｃプログラムをシミュレートされた分散環境下で実行できるようなコード，実行コードに変換し，その実行コードに基づいてシミュレーションを行うこととした．

また，ＰＥの並列動作は，ＰＥ管理キューというものを持つことにより，各ＰＥをプロセスとしてスケジュールすることでシミュレートしている．

4.1　ソフトウェアシミュレータの構成

本ソフトウェアシミュレータは，ＫＬ１－Ｃプログラムを実行コードに変換するコンパイラと，その実行コードに基づいてシミュレーションを行う実行系の２つの部分から構成される．図４－１に，本ソフトウェアシミュレータの基本構成を示す．

前述したように，ＫＬ１－ＣとしてはＦＧＨＣを想定している．さらに，分散処理用のゴール割付の指示をプラグマを用いることで可能としている．

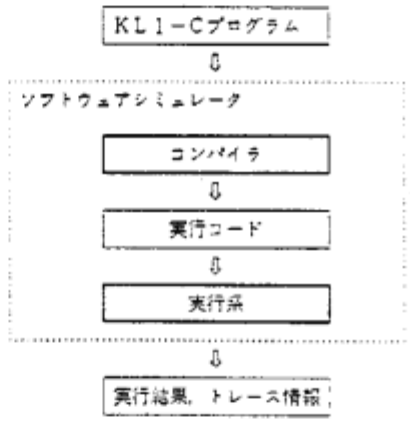


図４－１　ソフトウェアシミュレータの構成

4.2　コンパイラ

本コンパイラでは，〔上田　94〕を参考に，ＫＬ１－Ｃプログラムを Head部・Guard部・Body 部に分け，それぞれ差分リストの形に表現し，コンパイルする．そして最後にそれらを結合して，実行コードを出力する．

(1)　Guard 部

Guard 部では，アーギュメント変数，つまり呼出し元の変数に値を設定するようなユニフィケーションは行えない．そこで，実行系で Guard部を実行する時に，アーギュメントに関するユニフィケーションとローカル変数に関するユニフィケーションとの区別をする必要があるが，それでは実行時の処理効率が悪く，さらに処理系特にユニフィケーションが複雑になってしまう．ところで，アーギュメントに関するユニフィケーションをコンパイル時に行うということは，アーギュメントにそのユニフィケーションの結果が入った実行コードが出力されることを意味する．しかし，これはＦＧＨＣの計算規則から元のプログラムと意味的に等しい．そこで，Guard 部

のゴールをアーギュメントに関するユニフィケーションゴールと，その他のゴールに分け，アーギュメントに関するユニフィケーションゴールは，コンパイル時に実行する．以下に Guard部のコンパイル例を示す．

ＫＬ１－Ｃプログラム

```
f(X,Y) :- X=1, Y>0 | B.
```

実行コード

```
f(節番号, g(1,A2), R, [Bh | Bt ], Bt ) :-
        '>'(A2, 0, R1),
        ( not(R1==succeeded), R=R1 ; R=succeeded), !.
```

(2) Body部

ＦＧＨＣでは，and 並列に動作する他のゴールによって値が設定される変数が原因で，あるゴールの実行がサスペンドすることがある．本コンパイラでは，このようなサスペンドの可能性を減らすために，ユニフィケーションに関するゴールが先に処理されるように，ゴールの順序を変更する．

(3) ヘッドユニフィケーションゴール

値を持つアーギュメントに対しては，アーギュメントのマッチングと呼出し元に値を設定してはいけないというＦＧＨＣの評価規則をチェックする，ヘッドユニフィケーションゴールを作成する．

(4) 候補節選択

ＫＬ１－Ｃでは，各候補節の passive part は，上から下に逐次的に実行される．そのため，どの候補節が実行されているか実行系で認識できるようにするため，各節に番号を付加する．

また，候補節の実行結果を実行系に通知するため，各節の実行コードに結果が格納されるフィールドを設定する．

以下にコンパイル形式を示す．

```
KL1-C       Head :- Guard | Body.
                          ⇩
コード  Head( 節番号, アーギュメント, 結果,Body') :-
                ヘッドユニフィケーションゴール,Guard', !.
```

4.3 実行コード

実行コードは，ＫＬ１－Ｃ（ＦＧＨＣ）の持つ並列性等の特徴を生かし，実行系で効率良く実行できるコードである必要がある．今回は，ＫＬ１－Ｃ（ＦＧＨＣ）との言語仕様の類似，処理系作成の容易さ等から，実行コードとして Prolog を採用した．

ＫＬ１－Ｃでは，and 並列に動作する複数のゴール間の通信・同期は，ヘッドユニフィケーション及び Guard部ゴールの実行によって行われる．しかし，実行コードとして採用した Prolog では，単純なユニフィケーションしか行われないので，コンパイラによって，ヘッドユニフィケーションと Guard部ゴールに関して特別なコードが生成される．以下に，付録１の述語 part の実行コードの一部を示す．

述語 part の定義

```
part(X, Xs, A2, A3, A4) :- A2<X | A4=X, A5=
                           throw( part(Xs, A2, A3, A5), down).
```

実行コード

```
part(1,g(P1,P2,P3,P4),R,
        , &S(ulist(P4,X,A5)),
          throw( &part(Xs,P2,P3,A5)),down) Bt ] , Bt):-
 ulist(P1,X,Xs,R1), (not(R1==succeeded),R=R1; '<'(P2,X,R)).
```

4.4 実行系

実行系は，逐次計算機上のＫＬ１分散処理系であり，ＰＥの並列動作は適当なスケジューリング・アルゴリズムによってシミュレートされる必要がある．

今回は，ＰＥ管理用キューを持ち，各ＰＥをプロセスとしてスケジュールすることで，ＰＥの並列動作をシミュレートする．図４－２に，ＰＥ管理キューのイメージを示す．

実行系は，ＰＥ管理キューの最左のＰＥを取り出し，そのＰＥに実行権を割り当てる．実行権を割り当てられたＰＥは，ゴールキュー内のゴールを処理する．ゴールキューを１サイクル実行し終ると，ＰＥは実行権を実行系に返却する．実行系は，そのＰＥをＰＥ管理キューの最後部にスケジュールし直す．実行系は，すべてのＰＥ内に実行すべきゴールが無くなる，つまり，与えられたゴールが解かれたか，あるＰＥでのゴール実行が失敗し，システム全体が失敗するまで，上記の処理を繰り返す．

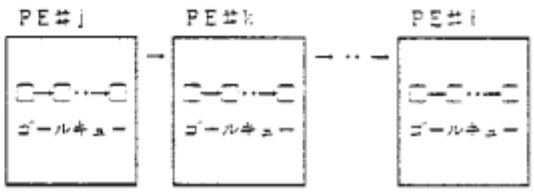


図４－２　ＰＥ管理キュー

5 ユニフィケーション・アルゴリズム

ユニフィケーションは，分散ユニフィケーションを行う分だけ，複雑になっている．すなわち，一般的な逐次処理におけるユニフィケーションに加え，チャネル変数とのユニフィケーション，チャネル変数同士のユニフィケーションを考慮する必要がある．

ゴールの実行時，チャネル変数とのユニフィケーションが発生すると，チャネル変数の実体は元のＰＥにあるので，チャネル変数とのユニフィケーションは，元のＰＥで行うことを原則とする．つまり，あるＰＥでチャネル変数とのユニフィケーションが発生すると，元のＰＥに対してユニフィケーション要求が出される．ユニフィケーション要求を受け取ったそのＰＥは，チャネル変数の実体に対して，要求されたユニフィケーションを行い，その結果（成功・失敗）を要求元に通知する．普通の変数同士のユニフィケーションは，通常のＦＧＨＣと同じであるので，普通の変数とチャネル変数及び，チャネル変数同士のユニフィケーションだけをGuard 部，Body 部に分けて述べる．

(1) Guard 部

Guard 部においては，ユニフィケーションに関するゴールは，コンパイル時に，ユニフィケーションが実行される．

例えば，プログラム

```
consume ( [ H | Hs ] , B ) :- H = 'EOS' | B = [ ] .
```

は，

```
consume ( [ 'EOS' | Hs ] , B ) :- true    | B = [ ] .
```

として，コンパイルされる．

つまり，ヘッドユニフィケーション，すなわちアーギュメント変数とローカル変数とのユニフィケーションのみを考慮すればよい．

① 普通の変数とチャネル変数（ $CHA(Xc,\_)$ ）とのユニフィケーションの時

そのチャネル変数の値の転送要求(get_value) を送り，このユニフィケーションを要求した候補節のテストがサスペンドする．

② チャネル変数同士のユニフィケーションの時

チャネル変数同士のユニフィケーションは，必ずサスペンドする．

(2) Body部

① 普通の変数とチャネル変数（ $CHA(Xc,\_)$ ）とのユニフィケーションの時

普通の変数が未定義ならば，チャネル変数自身を代入し，このユニフィケーションは成功する．

また，普通の変数に値が実体化されていれば，まず Xc にその値を代入し，次にそのチャネル変数の元のＰＥに対して，その値とのユニフィケーションを要求する(unify)．そして，このユニフィケーションは成功する．それと同時に，結果を受け取るために使用する変数に，unify 要求の結果をチェックするためのプロセスを hook する．

結果が戻され変数が実体化されると，hookされていたプロセスがゴールキューにスケジューリングされる．このプロセスは，戻された結果が failed であると，自分自身を fail させる．これによって，システム全体が fail する．

② チャネル変数（$CHA(Xc,\_)$）とチャネル変数（$CHA(Yc,\_)$）とのユニフィケーション時

まず，XcとYcとの間に referenceチェーンを張り，次に一方のチャネル変数の元のＰＥに対し，他方のチャネル変数とのユニフィケーションを要求する(unify_channels)．そして，このユニフィケーションは成功する．

以降の処理は，①と同様である．

6 リダクション・アルゴリズム

プログラム実行の基本原理は，ゴールを実行し，その結果のゴール群をリダクションすることを繰り返すことである．

つまり，ゴールキューの最左ゴールを取り出し，そのゴールに対応する候補節の数だけの要素を持つResult_list を作成する．そして，その Result_listを用いて候補節選択テストを行い，選択テストが最初に成功した候補節を選び出し，そのBody部ゴールをゴールキューにスケジュールする．

このように，候補節選択テストの結果をリストとして表現することが今回の特徴である．これにより，どのような効果が得られるか以下に示す．また，リダクション・アルゴリズムを図６－１に示す．

① ＫＬ１－Ｃプログラムを，実行コードにコンパイルするときに，候補節選択のための特別なコードを生成する必要がない．

② 再度，候補節選択を行わなければならない場合，以前の選択テストの結果が，候補節ごとに保存されているので，どの候補節を実行すべきかが容易にわかる．

例えば，候補節選択テストが失敗した候補節は，再度実行する必要がない．また，選択テストがサスペンドした候補節は，その原因（サスペンド原因の変数）がResult_list に記憶されるので，その候補節の選択テストをリジュームしてよいかが容易にわかる．

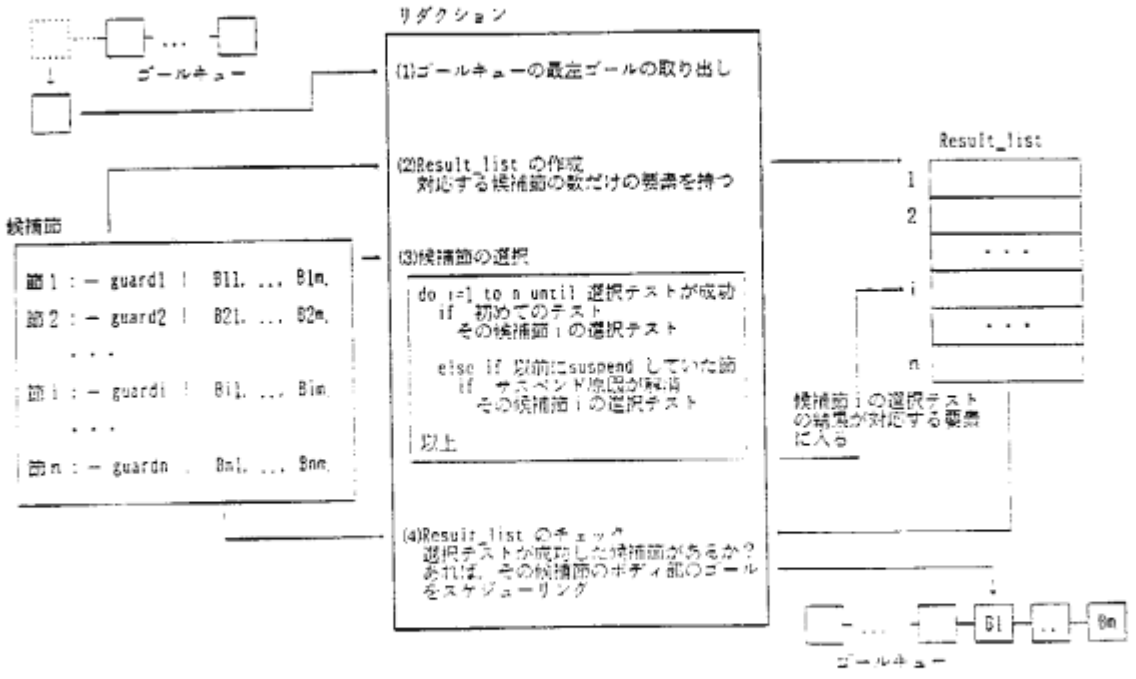


図６－１ リダクション・アルゴリズム

7　シミュレーション結果

前述のシミュレータを利用して，いくつかのサンプルプログラムを実行させ，分散環境下でのプログラム実行と一つのプロセッサ上でのプログラム実行との比較を行った．

今回のシミュレーションでは，すべてのＰＥにひと通り実行権を渡した時を１サイクルとし，ＰＥ間のメッセージ送受信は，このサイクル毎に１度ずつ行うこととした．また，実行の評価基準としては，ＰＥにおいて処理されるゴールの数を考え，サイクル毎の各ＰＥのゴール処理量の最大値をすべてのサイクルに渡って合計したものを全体としての処理量とした．さらに，ゴールサスペンドの影響を見るため，サスペンドしたゴールがリジュームされるまでの時間（サイクル数）を考えることとした．

以下の説明で，サイクル数は各ＰＥに何回実行権が渡されたかを表すサイクル回数，総ゴール数は実行されたゴールの総数，サスペンドゴール数はサスペンドしたゴールの数，通信量はネットワークを流れたプリミティブの総数，処理量はプログラム実行に要したゴール処理量，プリミティブを除く処理量はプリミティブをゴールとしてではなく，ネットワークマネージャのレベルで処理したと仮定した時のゴールの処理量，リジューム待ち時間は各サスペンドゴールがリジュームまでに要したサイクル数の総和をサスペンドゴール数で割ったものである．

例１　算術演算プログラム（付録１）

算術演算プログラムに，ゴールを分散させる命令（プラグマ）を付加したものと，付加しないものとを比較した．これは，ＰＥ９個での分散処理と，ＰＥ１個での集中処理との比較である．

実行結果を表７－１に示す．この表が示すように，分散処理した方がＰＥ間の通信及びサスペンドの処理のため，総ゴール数，処理量が増加する．また，ＰＥ１個の時はサスペンドは起らないが，分散させるとメッセージ送受信のタイミングのためサスペンドが発生してしまう．この例では，分散の効果が得られないだけではなく，分散させることがかえって逆効果であった．

表７－１　例１の実行結果

項　目	ＰＥ９個	ＰＥ１個
サイクル数	6	6
総ゴール数	34	11
サスペンドゴール数	15	0
通信量	13	0
処理量	18	11
プリミティブを除く処理量	11	11
リジューム待ち時間	1．4	0

例２　エラトステネスのふるい（付録２）

このプログラムは，エラトステネスのふるいといって素数を生成し，求まった素数を使ってさらに倍数を取り除いていく方法である．実行結果を表７－２に示す．

分散処理した方が通信量，サスペンドゴール数が増え，リジューム待ち時間もほとんど変らないにもかかわらず，処理量は減少している．また，メッセージ通信用のプリミティブをネットワークマネージャのレベルで処理することも，ある程度効果があることがわかる．この例では，例１とは逆に総じて分散の効果が現れている．

表７－２　例２の実行結果

項　目	ＰＥ９個	ＰＥ１個
サイクル数	48	46
総ゴール数	104	99
サスペンドゴール数	21	7
通信量	21	0
処理量	76	99
プリミティブを除く処理量	67	99
リジューム待ち時間	4．67	5．0

この２つの例からは，次のことが言える．

例１では，分散処理の方が集中処理よりも処理量が増え，これはＰＥ間の通信量及びゴールのサスペンドによることが考えられるが，例２からは，処理量が必ずしも通信量やサスペンドゴール数に依存しないことが分る．すべてのプログラムで分散の効果が必ずしも現れるのではなく，かえって逆効果になることもある．これは，そのプログラムの持っている並列性やプログラムの記述方法にかかわった問題である．

一般に，「分散処理は効果がないのでは？」といわれているが，今回のシミュレーションから，ある種のプログラムでは効果があることがわかった．今後は，多くのサンプルプログラムによるデータを収集し，詳細な検討を行っていくつもりである．

おわりに

今回，ＫＬ１－Ｃ（ＰＣＨＣ）の分散処理系の実行モデルの検討を行い，その動作原理の確認のため，逐次計算機上にＫＬ１の分散処理系ソフトウェアシミュレータを試作した．

本ソフトウェアシミュレータは，１Ｋステップ強であり，４人月で完成した．

そしていくつかのサンプルプログラムに適用し，定性的な評価を行った．その結果，分散処理に伴う通信量，サスペンドゴール数の増加によって必ずしも実行時間が悪化していくのではなく，改善されるという評価が得られた．定量的な評価を行うには，さらに多くのプログラムに適用し，多くのデータを収集する必要がある．

また，ＰＩＭ上での実行モデルを明確化するために，プラグマ方式，分散ユニフィケーション方式，ＰＥ及びゴールのスケジューリング方式，ＰＥ台数等を考慮に入れた検討も今後の課題である．

謝辞

本研究は、第５世代コンピュータの一環として行われた。本研究開発にあたり、多くの示唆、援助をしていただいたICOTの方々に感謝致します。

参考文献

〔古川　85〕　古川他、核言語第一版説明資料(ICOT, 1985).

〔Shapiro 83〕 Shapiro, E. Y., "A Subset of Concurrent Prolog and Its Interpreter " ICOT TR-003.

〔上田　85a〕 上田, "Concurrent Prtolog Re-examined" ICOT Technical Report TR-102.

〔上田　85b〕 上田, "Guarded Horn Clauses" ICOT Technical Report TR-103.

〔江崎　86〕　江崎他、ＧＨＣサブセット逐次型処理系の作成、情報処理学会第32回全国大会 (1986).

〔村上　85〕　村上、マルチプロセッサにおけるユニファイアの検討、ICOT内部メモ (1985).

〔上田　84〕　上田、近山、並列論理型言語の実用処理系、日本ソフトウェア科学会第一回大会論文集 (1984).

付録１　算術演算プログラム

```
?- test(2).
test(A) :- true | write(A),
                  throw( do1(A, B), up),
                  throw( do2(B, C), down),
                  throw( do3(C, D), left).
do1(A, B) :- true | plus(B, A, 2), write(B).
do2(B, C) :- true | times(C, B, 2), write(C).
do3(C, D) :- true | div(D, C, 2), write(D).
```

付録２　エラストテネスのふるい

```
?- primes(2, 2,3,4,5,6,7,8,nil).
primes(A, _) :- A == 'End' | true.
primes(A, List) :- A \== 'End' | filter(A, List, Next),
                  primes(Next, List).
filter(A, Head,List, Next) :- Head <= A |
                  filter(A, List, Next).
filter(A, Head,List, Next) :- Head > A, Head mod A =:= 0 |
                  filter(A, List, Next), write(Head).
filter(A, Head,List, Next) :- Head > A, Head mod A =\= 0 |
                  filter(A, List,   _ ), Next = Head.
filter(A,      nil , Next) :- true | A = nil, Next = 'End'.
```