ICOT Technical Report: TR-556

TR-556

並列推論マシンにおけるKL1の実行制御方式
—分散ゴール管理の課題と対策

川合英夫，伸瀬明彦，今井　明，
後藤厚宏，六沢一昭

May. 1990

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

Institute for New Generation Computer Technology

並列推論マシンにおける KL1 の実行制御方式
- 分散ゴール管理の課題と対策 -

川合 英夫 †　　仲瀬 明彦 ††

今井 明 †　　後藤 厚宏 †　　六沢 一昭 †††

†新世代コンピュータ技術開発機構
††(株) 東芝
†††沖電気 (株)

概要

並列推論マシン PIM は、共有メモリに複数のプロセッサを密結合したクラスタを、メッセージの追い越しが生じ得るようなネットワークによって疎結合した構成を持つマシンである。このようなマシン上に、並列論理型言語 KL1 を実行する処理系を実装する際には、KL1 自身が持つメタ制御機能をクラスタに分散し、なおかつ分散によって生じる制御の時間遅れや誤差といったデメリットを現実的な範囲に抑えることが重要である。本報告では、共有メモリ環境においては共有データに対するアクセス集中を避け、分散環境においてはメッセージの追い越しに対応することによって制御を分散する方式について述べる。

Distributed KL1 Goal Management for the Parallel Inference Machine

Hideo KAWAI †　Akihiko NAKASE ††
Akira IMAI †　Atsuhiro GOTO †　Kazuaki ROKUSAWA †††

†Institute for New Generation Computer Technology (ICOT)
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
††Toshiba Corporation
†††Oki Electric Industry Co.,Ltd

Abstract

The Parallel Inference Machine PIM is a computer system which has a two level hierarchy. The lower is a cluster which has multi-processors with shared memory. The higher is a system which is consisted of clusters connected with message overtakable network system. To implement KL1 language execution system on such a computer system, it is very important to distribute meta control function of KL1 and to minimize the demerits caused by distribution. This paper describes how to distribute KL1 meta control function. This scheme is avoiding concentration of access to specific data and corresponding to message overtakable network system.

1 はじめに

並列推論マシン PIM[4] は、AND 並列論理型プログラミング言語 KL1 を高速に並列実行するマシンであり、第5世代コンピュータプロジェクトの一環として、現在その開発が進められている。

PIM は、多数のプロセッサを2レベルの階層で結合した構成をとる。つまり、8台程度のプロセッサを共有バスで密結合してクラスタと呼ぶサブシステムを作り、そのクラスタを更にネットワークで疎結合することによって全体システムを構成する。

このため、クラスタ内に注目した場合には PIM は密結合共有メモリマシンであり、またクラスタ間に注目した場合には PIM は疎結合分散メモリマシンである。

一方 KL1[2] は、KL1 自身によって OS(Operation System) を記述できるように、言語自身にプログラムの実行制御や資源管理、例外処理などのメタ制御機能を導入した言語である。

この KL1 言語処理系を、PIM のようなシステム構成を持つマシン上に実装しようとする時、単純に制御機能を一つのクラスタに集中して管理するような実装を取ると、分散メモリ環境においてはクラスタ間の制御メッセージの増大、共有メモリ環境においては共有データ構造へのアクセス集中といった問題が生じる。

そこで、制御機能をクラスタに分散することを考えるが、この時機能を分散することによって生じる制御の時間遅れや誤差といったデメリットを、できるだけ現実的な範囲に抑えることが重要である。

このうち、追い越しの生じないネットワークで結合された分散環境における制御機能の分散方式については、PIM に先立って開発された並列推論マシンパイロットモデル Multi-PSI[6] 上の KL1 言語処理系 [7] によって既に解決されている。

本稿では、2章、3章において Multi-PSI 上の KL1 言語処理系での KL1 ゴールの管理方式とその実装方式について、4章、5章において Multi-PSI 上の KL1 言語処理系では未検討であった、共有メモリ環境における排他制御を用いた実装方式および追い越しの生じ得るネットワークによって結合された分散環境でのゴールの管理について述べる。

2 KL1 におけるゴール管理機能

KL1 は、Flat GHC(Guarded Horn Clauses)[1] を基にした Commited-Choice 型の AND 並列論理型プログラミング言語である。

Flat GHC のような全てのゴールが平板な論理積となっているような言語では、一つのゴールの失敗がシステム全体の失敗となってしまうため、大規模で複雑なプログラムを作成するのは困難である。このため KL1 では、Flat GHC に失敗の範囲を局所化できるような構造を持ち込み、この構造ごとにゴールの実行を制御できるようした。このような構造を荘園と呼ぶ。

2.1 荘園

荘園とは、ゴールの実行制御、資源管理および例外処理の単位となる機構である。KL1 のゴールは必ずいずれかの荘園に属しており、そのゴールから発生した全てのゴールもまた同じ荘園に属する。荘園内のゴールが更に荘園を生成することも可能であり、新たに生成された荘園はゴールが属する荘園の子荘園となる。このように、荘園は一般に図1に示されるような親子関係を持った階層構造となっている。

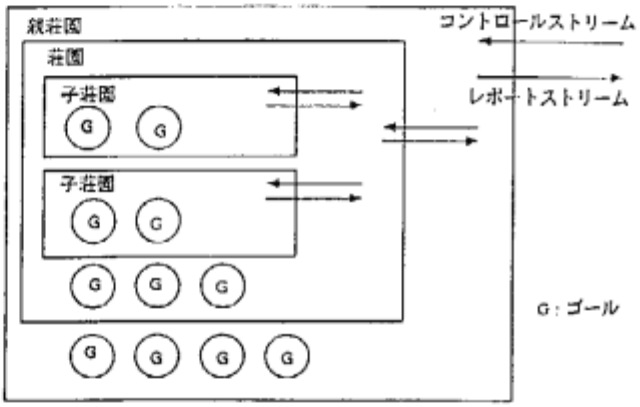

図 1: 荘園の親子関係

2.2 コントロール / レポートストリーム

荘園を制御するには、コントロールストリームと呼ばれるストリームから荘園の制御のための指示 (起動 / 停止 / 実行放棄 / 資源追加など) を入力する。コントロールストリームは図2のような KL1 プログラムで実現されており、コントロールストリームへの入力は対応する組み込み述語に変換されて実行される。

荘園がコントロールストリームからの停止指示で停止した場合、その荘園に属する全てのゴールの実行が停止されると共に、荘園の子孫荘園に属するゴールも全て停止される。また、荘園の再起動や実行放棄についても同様であり、親荘園の状態の変化は全ての子孫荘園に反映される。

また、コントロールストリームからの入力に対する応答やゴール実行中に発生した例外の情報などはレポートストリームから報告される。

```
control([start|CNTL],Shoen):-
      start_shoen(Shoen,NewShoen),
      control(CNTL,NewShoen).
control([stop|CNTL],Shoen):-
      stop_shoen(Shoen,NewShoen),
      control(CNTL,NewShoen).
control([abort|CNTL],Shoen):-
      abort_shoen(Shoen,NewShoen),
      control(CNTL,NewShoen).
control([add_resource(Res)|CNTL],Shoen):-
      add_shoen_resource(Shoen,Res,NewShoen),
      control(CNTL,NewShoen).
```

図 2: コントロールストリームの例

このレポートストリームは、荘園生成時に作られる。

2.3 資源管理

資源とは、プログラムの実行による計算機使用量を、総合的かつ具体的に表現するために導入された概念であり、プログラムが消費した資源量を監視することによってプログラムの挙動を把握したり、プログラムに供給する資源量を調節することによってプログラムの実行を制御したりすることができる。

なお、資源に反映されるべき計算機使用量としては、CPU時間やメモリ消費量などが考えられるが、現在はCPU時間の代用としてリダクション数のみを資源として用いている。

資源を管理する単位は荘園であり、各荘園ではその荘園内で消費できる資源量の上限が定められている。荘園がその資源量の上限近くまで資源を消費すると、荘園のレポートストリームから資源値少報告がなされる。

これに対して、荘園のコントロールストリームから資源を追加することができる。資源値少報告後荘園に資源を追加しないでおくと、荘園の資源はやがて使い果たされ、資源が使い果たされた荘園ではゴールのリダクションは行われない。

さらに、コントロールストリームからは、荘園が現在までにどれくらいの資源を消費したかを問い合わせることもできるため、問い合わせ結果をもとにして荘園を制御することも可能である。

ただし、荘園への消費資源量問い合わせ時には一般に荘園ではゴールが実行されているため、消費資源量問い合わせに対する応答には一般に誤差が含まれており、次の2点だけが保証されている。

- 資源消費量は単調増加であり、減少はしない。
- 終結した荘園から得られる消費資源量は誤差を含んでいない。

3 荘園機能の実装

荘園をMulti-PSIやPIMのような分散環境を持つマシン上に実装する際には、分散してゴールを管理する方式や、クラスタ間メッセージを増加させずに効率良く終了判定する方式が課題となる。以下に、Multi-PSI上のKL1言語処理系開発時に提案された、里親およびWTCを用いた管理方式について述べる。

3.1 里親

ゴールは、負荷分散などにより他のクラスタに投げられることがある。このため、荘園の存在するクラスタ以外のクラスタでも、荘園に代わってゴールを管理する実体が必要であり、これを里親と呼んでいる。里親は荘園によって管理されており、荘園の状態変化は里親にも反映される。しかし、コントロールストリームやレポートストリームは持たず、里親の管理するゴールの実行中に発生した例外の情報などは荘園のレポートストリームを通じて報告される。また、里親の管理するゴールが子荘園を生成した場合、この子荘園は里親に双方向リンクによってつながれ、里親によって管理される。

また、荘園の存在するクラスタと里親の存在するクラスタを処理系レベルで区別して処理を分けると処理系が複雑になるため、図3に示すように荘園の存在するクラスタにも里親を置き、荘園と里親間の通信は全てクラスタ間メッセージを用いて行うものとしている。(実際にネットワークに出すか否かはネットワークハンドラのレベルで決める)

このため、荘園と同一クラスタにある里親も他クラスタにある里親と同様にゴールや子荘園を管理しており、実質的にはゴールや子荘園は里親に属していると考えてよい。

里親はゴールを受け取ったクラスタに、そのゴールが属する里親が存在しない場合に動的に生成され、その里親に属するゴールや子荘園がなくなった時に消滅する。

3.2 終了判定

里親はゴール及び子荘園の終了を判定するためにチャイルドカウントと呼ばれるカウンタを持っている。チャイルドカウンタはその里親に属するゴールの総数と子荘園の総数との和であり、里親が他のクラスタから投げられてきたゴールを受け取ったり、ゴールが新たなゴールや子荘園を生成した時にカウントアップされ、ゴールや子荘園が終了した時にはカウントダウンされる。このため、チャイルド

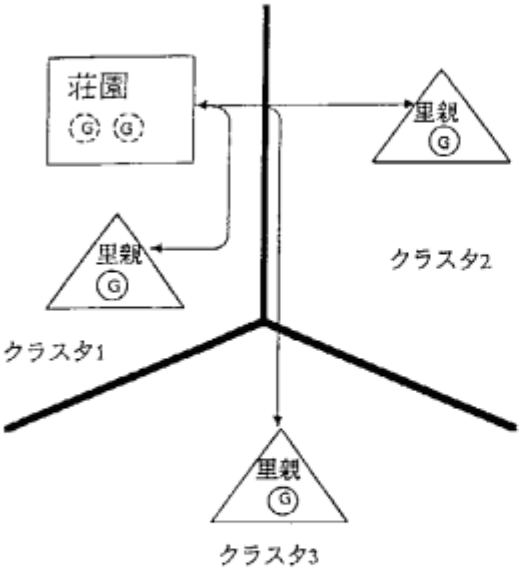

図 3: 荘園と里親

カウントが0になったことで、この里親の中で行なう処理がなくなったことが検出できるため、この時点で里親は終結し、終結メッセージを荘園に送信する。

また、荘園の実行放棄(アボート)によって里親が終結する時は、里親はすべての子荘園に対してアボートメッセージを送り、全ての子荘園が終結した時点で里親も終結する。この場合一般に里親には実行されないまま放棄されるゴールが存在するため、チャイルドカウントは0にはならない。

一方、荘園は図4に示されるようにWTC(Weighted Throw Count)と呼ばれる値を用いて、里親及びクラスタ間メッセージ処理の管理を行なう。WTCはトータルが0になる値で、里親の生成時に一定量のWTCが里親に与えられ、それと同じ量のWTCが荘園から引かれる。また、里親が終結した時にはその里親のWTCが荘園に返される。

全てのクラスタ間メッセージにもWTCが付加され、メッセージ送信側ではメッセージに付加した量のWTCを減じ、受信側ではメッセージに付加されてきたWTCを加算する。里親のWTCが不足した場合、里親は荘園にWTC要求を出すが、WTCが供給されるまでの間メッセージの送信処理は中断される。

荘園が実行放棄(アボート)する場合は、里親の存在するクラスタにアボートメッセージが送られる。アボートメッセージを受信したクラスタの里親は自発的に終結し、既に里親の終結したクラスタにアボートメッセージが到着した場合にはメッセージが送り返される。また、アボートメッセージ送信後に生成が通知された里親には追加してアボートメッセージが送信される。

このようにしてWTCを管理することにより、荘園に里親があったりクラスタ間を渡るメッセージが未処理である内はWTCが0にならないことが保証できる。従って、荘園はメッセージが到着するたびに荘園のWTC値を調べ、WTC値が0になった時点で終結すればよい。

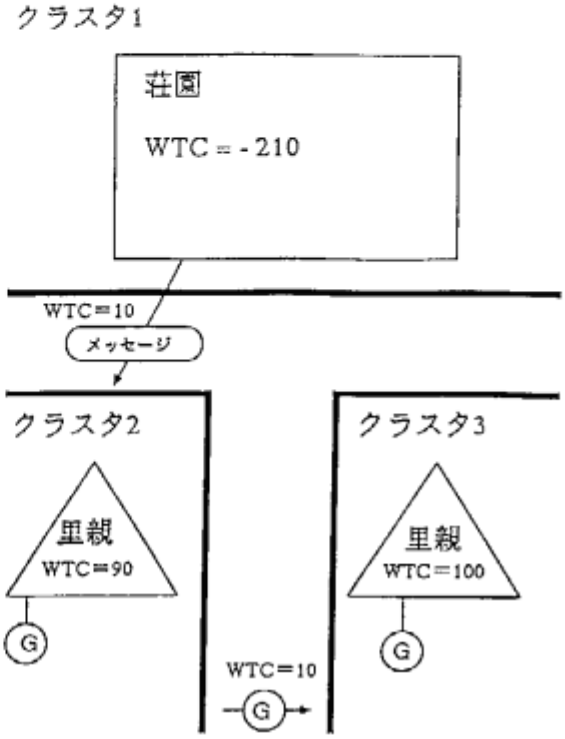

図 4: 荘園のWTC管理

なお、Multi-PSIでは個々のPE(Processing Element)がそれぞれメッセージの追い越しの生じないネットワークで結合されていたのに対して、PIMでは共有メモリを介して複数のPEが密結合されたクラスタが、メッセージの追い越し現象が生じうるネットワークで結合されている。

このため、PIMにおいてはクラスタ内処理における共有データアクセスに伴うプロセッサ間排他制御や実行時オーバヘッドの低減、クラスタ間処理においてはネットワークメッセージの遅延や追い越し対応などの課題がある。

これらの課題とその対策については次章以降で述べる。

4 共有メモリ環境における課題と対策

荘園及び里親はそれぞれ共有メモリ上の構造体(レコード)として実現されており、クラスタ内の各PEからは共有データとして見えるものである。従って、荘園および里親にアクセスする時にはPE間で競合が起こる可能性があり、PE間の排他制御が必要になる。

また、ゴールは実質的には里親で管理されているため、里親レコードには里親の状態や資源残量、チャイルドカ

ウント値などゴールの実行時に必要となるデータが格納されている。従って、クラスタ内の各PEがゴールのスケジュール毎にそのゴールが属する里親にアクセスような方式をとると、クラスタ内の複数PEで同じ里親に属するゴールを同時に実行する場合など、一つの里親レコードに対してアクセスが集中してしまい、里親の排他制御やPE間のバストラフィックが増大することなどにより、オーバヘッドが大きくなってしまう。

このためPIMでは、できるだけ各PEが里親のアクセス頻度の高いデータにアクセスせずに済むように、資源残量やチャイルドカウント値をPE毎にキャッシュして管理し、里親の状態については、キャッシュした資源が尽きた時と実行するゴールの里親が交替する場合にチェックするものとした。

しかし、このような処理方式を取ることによって里親の消費資源量を調査する際の誤差や里親の状態変化への対応の遅れが生じるため、これらの場合にはPE間相互割り込み機能を利用して対応する。

以下、それぞれの方式について述べる。

4.1 PE毎の資源キャッシュ

ゴールを実行する前に、図5に示すように、そのゴールが所属する里親の資源から一定量(キャッシュユニット)をPE個別のレジスタに取り分けておき、以後連続して同じ里親に属するゴールを実行する時は、このPE個別のレジスタからデクリメントする。

キャッシュした資源が尽きた時には、里親から再び資源をキャッシュして来ると同時に里親の状態をチェックする。

この時、里親に資源がなければ実行しようとしていたゴールは中断ゴールとして里親レコードにつながれ、次のゴールがスケジューリングされる。なお、里親レコードにつながれた中断ゴールは、里親に資源が補給された時点で里親レコードから外され、再びスケジューリングの対象となる。

また、直前まで実行していたゴールとは違う里親に属するゴールがスケジュールされた時は、PEごとにキャッシュされていた資源を、直前まで実行していたゴールの里親に加算する。その後、スケジュールされたゴールが属する里親の資源をキャッシュしてゴールを実行する。

なお、この時もこの里親が実行可能状態であるか否かのチェックも行ない、もし里親が実行可能状態でなければ、さらに次のゴールをスケジュールする。

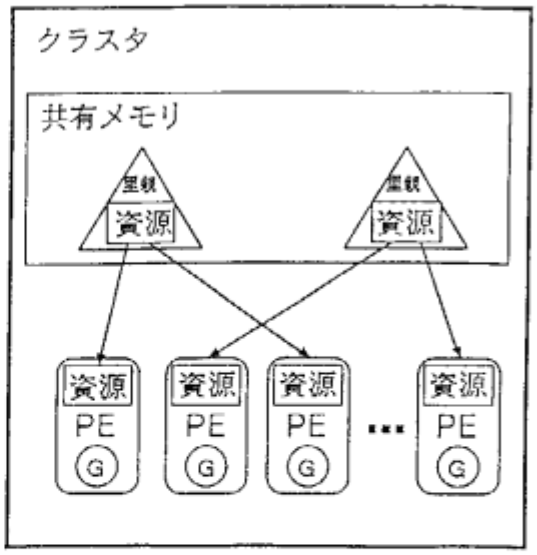

図5: 里親の資源キャッシュ

4.2 PE毎のチャイルドカウント値キャッシュ

資源と同じ様に、チャイルドカウント値をPE個別のレジスタにキャッシュする。

ゴールを実行する前に、PE毎のチャイルドカウント値を0にしておく。そして、連続して同じ里親に属するゴールを実行する際に、ゴールの中で新たなゴールを生成した場合にはこのチャイルドカウント値を一つ加算し、ゴールの実行が終了した場合には一つ減算する。

また、違う里親に属するゴールがスケジュールされた時は、キャッシュされていたチャイルドカウント値を、直前まで実行していたゴールが属する里親に加算し、PE毎のチャイルドカウント値を0にクリアした後にゴールを実行する。

このように管理することで、PE個別のチャイルドカウント値を保持するレジスタと、里親の持っているチャイルドカウントとの和が、その時点での正確なチャイルドカウント値となる。

なお、里親にチャイルドカウント値を加算した結果この値が0をなった場合には、里親は終結する。

4.3 PE間相互割り込み機能の利用

上記のように、できるだけ各PEが里親のアクセス頻度の高いデータにアクセスせずに済むような処理方式を取ることによって、次のような点に対して新たな対応が必要となる。

- 里親の状態変化に対する対応の遅れ
- 里親の消費資源量の誤差

以下、それぞれについて述べる。

1. 里親の状態変化に対する対応の遅れ

荘園の状態変化に伴って、里親の状態が実行可能状態から実行不可能状態に変化した場合、ゴールのスケジュール毎に里親の状態をチェックしていれば、里親が実行可能状態から実行不可能状態に変化した時にその里親に属するゴールの実行を防ぐことができる。

しかし、上記のように、同一の里親に属するゴールが連続して実行される時には原則として里親にアクセスしないような処理方式を取ると、里親の状態をチェックできるタイミングが

- キャッシュ資源が尽きた時
- 今までと違う里親に属するゴールがスケジュールされた時

に限られてしまうため、これら以外のタイミングでは実行不可能状態になったはずの里親に属するゴールがそのまま実行されてしまう可能性がある。

このため、クラスタ内に存在するいずれかの里親が実行不可能状態に変化した時には、共有結合されているPE間でのソフトウェア割り込み機能[5]を用いてクラスタ内の全PEに通知する。(図6参照)

各PEは、その時点で実行不可能状態に変化した里親に属するゴールを実行していた場合には、キャッシュしていた資源やチャイルドカウント値を里親に加算した後に、次のゴールをスケジュールして里親の状態をチェックすることによって、実行不可能状態に変化した里親に属するゴールの実行を防ぐ。

一方、その時点で実行していたゴールの里親が実行不可能状態に変化した里親ではなかったPEでは、そのままゴールを実行する。

この方式では、ある里親の状態が変化した時には、その里親とは関係のないゴールを実行していたPEでも、現在実行しているゴールの里親が実行不可能状態に変化した里親であるか否かを一旦チェックする必要があり、新たなオーバヘッドとなる。

しかし、里親の状態変化の頻度は毎回のゴールスケジューリングの頻度に比べて充分小さいことが予想されるため、この方式を採用することとした。

2. 里親の消費資源量の誤差

里親の資源の一部がクラスタ内の任意のPEにキャッシュされるため、消費資源量の問い合わせ時などには、里親にアクセスするだけでは里親の正しい消費資源量が分からなくなる。

また、終結していない荘園の消費資源量にはもともと誤差が含まれるため、このキャッシュ分の資源を誤差として扱ってしまうと、最悪の場合この誤差はPIMのシステム内の全PE数に比例した値となってしまう。

このため、里親の消費資源量を調査する場合には、調査対象の里親に属するゴールを実行していたPEのキャッシュ資源を里親に一旦戻して貰う。この場合にも里親の状態変化の場合と同様に、クラスタ内PE間の相互割り込み機能を用いてクラスタ内の全PEに通報した後、PE間の同期を取って各PEの処理を再開する。

この方式でも里親の状態変化の場合と同様に、資源調査と関係のない里親のゴールを実行していたPEにとってはオーバヘッドが生ずるが、資源問い合わせの頻度が大きくないことが予想されるため、この方式を採用することとした。

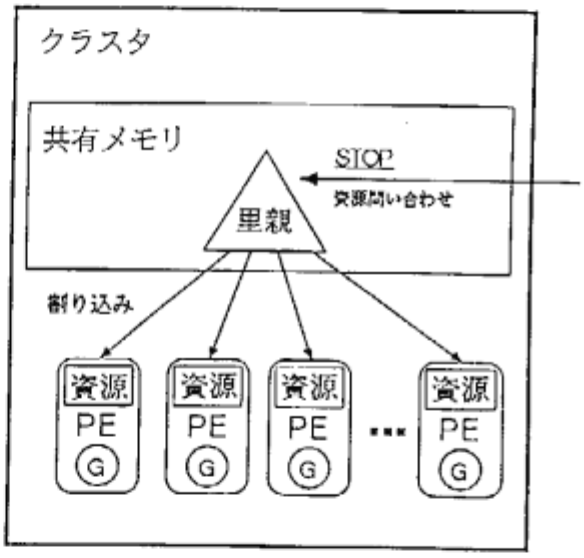

図 6: 割り込み機能によるクラスタ内PEへの通知

5 分散メモリ環境における課題と対策

分散メモリ環境では、分散データの管理や送受信などの操作が必要となるため、一般にクラスタ内処理に比べてコストが高くなる。このため、疎結合分散メモリ環境を持つマシンでは、できるだけクラスタ間の通信量を抑えて、各クラスタが独立に処理を行えるようにすることが重要である。

また、クラスタ間通信はネットワークの混雑状況などにより有限時間の遅延が生ずるため、これにも対応できるように処理系を設計しておく必要がある。

実際、Multi-PSI上のKL1言語処理系においても、上記の点が十分考慮された設計となっている。

荘園の状態	stop	⇒	start	⇒	stop	⇒	start	⇒	stop	⇒	start	⇒	stop	⇒	start
CS への入力		start		stop		start		stop		start		stop		start	
里親の状態	stop	⇒	stop	⇒	stop	⇒	start	⇒	start	⇒	start	⇒	start	⇒	stop
到着 MSG		stop		stop		start		start		start		start		stop	

表 1: 荘園と里親の状態変化 (その 1)

荘園の状態	stop	⇒	start	⇒	stop	⇒	start	⇒	stop	⇒	start	⇒	stop	⇒	start
CS への入力		start		stop		start		stop		start		stop		start	
里親の状態	stop	⇒	stop	⇒	stop	⇒	stop	⇒	stop	⇒	start	⇒	start	⇒	start
到着 MSG		stop		stop		start		start		start		start		stop	
StartCount		0		0		0		0		0		1		0	
StopCount		1		2		1		0		0		0		0	

表 2: 荘園と里親の状態変化 (その 2)

しかし、PIM では前述した様に Multi-PSI では生じなかったネットワークメッセージの追い越し現象が生じる可能性があるため、これに対応する部分を新たに設計する必要がある。

ネットワークメッセージの追い越しが生じた場合であっても、ネットワークの遅延によるメッセージの遅れと同等に扱うことができることがある。

例えば、里親への資源補給メッセージを、里親への実行放棄メッセージ (アボートメッセージ) が追い越した場合、資源補給メッセージは里親が終結してしまったクラスタに到着することになるが、これは資源補給メッセージの到着が遅れている間に里親が終結してしまった場合と同等と考えられる。

また、全てのメッセージに WTC を付加しておくため、メッセージの追い越しが生じてもメッセージの到着前に荘園が終結することもない。

しかし、次の場合についてはネットワークメッセージの追い越しを遅れと同等に考えることはできない。

- stop/start メッセージ間の追い越し
- ready/terminate メッセージ間の追い越し

以下、それぞれの場合について述べる。

5.1 stop/start メッセージ間の追い越し

荘園がコントロールストリームからの命令などで停止する場合、荘園の状態変化を反映するために、荘園から里親に向かって stop メッセージが送られる。また、停止していた荘園が再起動された時も同様に start メッセージが送られる。今、ある荘園が停止と再起動を繰り返したとすると、荘園から里親へは複数の stop メッセージと start メッセージが送られる。この時、ネットワークメッセージの追い越しを考慮せずに到着したメッセージの順に里親の状態を変化させると、追い越しが生じた時には停止状態の里親に stop メッセージあるいは実行状態の里親に start メッセージといった、(その時の里親にとって) 論理的に正しくないメッセージが到着することになる。

この時、この (一見) 論理的に正しくないメッセージを無視してしまうと、次の例のような問題が生じる。

[例]

いま、荘園とその里親が stop 状態にあるとして、荘園のコントロールストリームに

start、stop、start、stop、start、stop、start をこの順序で流したとする。この時、ある里親に到着したメッセージの順序が追い越しによって

stop、stop、start、start、start、start、stop に変わってしまったとすると、荘園と里親の状態は表 1 のように変化する。

つまり、荘園は最終的に start 状態となるのに対し、里親は最終的に stop 状態となってしまう。

このような問題が発生するのは、論理的に正しくないように見えるメッセージを無視することに原因がある。そこで、里親に到着するメッセージの内、start 状態での start、stop 状態での stop といった論理的におかしく見えるメッセージを受け取った時はそれらをカウントしておき、次にそのメッセージに対応するメッセージが到着した時点でカウントを減らし、カウントが 0 の時に論理的に正しいメッセージが到着した時だけ里親の状態を変化させることによってこの問題に対応する。

先の例では表 2 のようになる。

なお、この場合コントロールストリームへの入力が start/stop 同数であった場合、どんなに start/stop の数が多くても里親の状態が一回も変化しない可能性もあるが、

これは並列分散処理系の許容範囲内であると考える。

5.2 ready/terminate メッセージ間の追い越し

ready メッセージは、里親の存在しないクラスタにゴールが投げられた時に、そのクラスタに新たに里親が生成されたことを荘園に通知するためのメッセージである。また、terminate メッセージは、里親が終結してそのクラスタから消滅したことを荘園に通知するためのメッセージである。

いま、あるクラスタで里親の生成と終結が繰り返し行われた場合、複数の ready/terminate メッセージが里親から荘園に向かって送信される。

この時、これらのメッセージの間で追い越しが生じると、上記の start/stop メッセージの場合と同じように荘園側ではそのクラスタに里親が存在するか否かが判断できなくなる。

この場合にも、start/stop メッセージの場合と同じように、荘園にクラスタごとの ready/terminate メッセージをカウントするカウンタをつけることによって対応する。

6 まとめ

本報告では、密結合共有メモリ環境を持つクラスタと、このクラスタをメッセージの追い越しが生じ得るようなネットワークで結合したマシン上の KL1 ゴール管理方式について述べた。

この方式のポイントは

- 共有メモリ環境においてアクセス頻度の高いデータを PE 毎に分散することによって特定データへのアクセス集中を避けること
- 分散メモリ環境において追い越しが生じたと考えられるメッセージをカウントして管理することによってメッセージの追い越し現象に対応すること

である。

現在 PIM 上の KL1 言語処理系はシミュレータ上でその一部が動作し始めた段階である。

今後はシミュレータ上での動作結果や Multi-PSI の評価結果などを参考としながら処理系をより洗練して PIM 実機上に実装し、さらに実機上でも最適化を行っていく予定である。

謝辞

本研究を進めるに当たり、いつも貴重なご意見を頂く ICOT の Multi-PSI 及び PIMOS 開発メンバの方々ならびに ICOT 第四研究室の研究員の方々に感謝致します。また、本研究の機会を与えて頂いた渕一博 ICOT 所長、内田俊一第四研究室室長に感謝致します。

参考文献

[1] K. Ueda. Guarded Horn Clauses : A Parallel Logic Programming Language with the Concept of a Guard. Technical Report TR-208, ICOT 1986.

[2] 宮崎 敏彦 並列論理型言語 KL1 の実現方式と並列 OS の記述. 信学会論文誌 '88/8 Vol.J71-D-No8 pp1423-1432.

[3] K. Rokusawa, N. Ichiyoshi, T. Chikayama et al. An efficient termination detection and abortion algorithm for distributed processing systems. Proceedings of the 1988 International Conference on Parallel Processing, Vol.I, 1988.

[4] A. Goto, M. Sato, K. Nakajima, K. Taki et al. Overview of the Parallel Inference Machine Architecture(PIM). Proc. of the International Conference On Fifth Generation Computing Systems 1988, Tokyo, Japan, November 1988.

[5] 中川貴之、後藤厚宏、近山隆. プロセッサ間ソフトウェア割り込み処理を高速化するスリットチェック機構. 計算機アーキテクチャ研究会, July 1989.

[6] K. Taki. The parallel software research and development tool : Multi-PSI system. France-Japan Artificial Intelligence and Computer Science Symposium 86, pp 365-381, October 1986.

[7] K. Nakajima, Y. Inamura, I. Ichiyoshi et al. Distributed Implementation of KL1 on the Multi-PSI/V2. Proceedings of the Sixth International Conference on Logic Programming, pp436-451, June 1989.