ICOT Technical Memorandom: TM-1065

TM-1065

並列推論マシンPIMにおける メタ機能「荘園」の実装

山本 礼己、川合 英夫

June, 1991

© 1991, ICOT

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列推論マシンＰＩＭにおけるメタ機能「荘園」の実装

山本礼己　　川合英夫

(財) 新世代コンピュータ技術開発機構

「荘園」と呼んでいるＯＳ記述向きメタ機能とその実装方式および実行コストの測定について述べる。荘園の機能は、ＫＬ１によるＯＳ記述のための必要性から設計されたもので、現在は、並列プログラミング環境をサポートするために拡張が続けられている。実装方式は、共有メモリと分散メモリの階層を持つ並列推論マシンＰＩＭを対象として設計した。メモリを共有するクラスタ内では、排他制御の衝突や、並列処理の追い越しが問題であり、一方クラスタ間では、ネットワーク上のメッセージの追い越しが問題となった。これらの問題解決方法について報告する。また、シミュレータ上で行なっている実行コストの測定についても述べる。

Implementation of meta functions "Shoen" for PIM

Reki Yamamoto　　Hideo Kawai

Institute for New Generation Computer Technology (ICOT)

Mita Kokusai Bldg. 21F, 4-28, Mita 1-chome, Minato-ku, Tokyo 108 JAPAN

We describe functions of "Shoen" ,its implementation on PIM and measurement of the running cost on PIM/simulator. Functions of "Shoen" are designed for OS description in KL1, and still on development for making programming environment on the PIM. Implementation of "Shoen" is designed for parallel inference machine PIM which have hierarchical structure with shared and distributed memories. In a cluster with shared memory, there are problems with collision of exclusive accessing and outrunning of parallel shoen processings. And a case of inter-clusters, there are problems with outrunning of messages. Solutions of these problems are described. And also we describe a way of measurement and some results.

1 はじめに

本稿では、始めに荘園実装の背景、問題点、評価の考え方をを概説し、続いて機能、実装方式、測定結果について述べる。

1.1 背景

現在ICOTでは並列推論マシンPIMを開発中である[2]。PIMは論理型言語KL1を並列実行する。KL1はAND並列論理型言語であるFGHCにOS記述のための機能を付加した言語である[3]。この機能は、一言でいえば、「KL1プログラムの実行制御をKL1によって記述するメタ機能」である。これを「荘園」と名付けている。

「荘園」はKL1によって記述されたPIM用のOSであるPIMOSの中で用いられている[1]。また、一般のKL1ユーザもライブラリ化されたPIMOSのユーティリティを通じて、「荘園」を使うことができる。

「荘園」自身は、資源管理や、例外処理のためのメタ機能から始まって、今ではテスト/デバッグ機能のサポートも入り、並列プログラミング環境をサポートしている。また、さらに並列実行におけるタスクモジュール機能（大粒度負荷分散の単位）へと発展しつつある。

「荘園」の実装は、疎結合型の並列推論マシンMulti-PSI上ですでに行なわれており、一方、共有メモリを持つ密結合と、それらをネットワークでつないだ疎結合からなる階層構造を持つ並列推論マシンPIMでは、現在実装が進められている[5]。

1.2 実装における問題

KL1によるOS記述を検討したとき、同期および排他制御に関しては、KL1言語の機能として、ゴールリダクションのサスペンド/レジュームにより同期の問題が、データの単一代入性により排他制御の問題が、それぞれ解決されている。

一方ユーザとシステムを切り分けるためのメタ機能は論理型言語の枠組を越えて、「荘園」の導入により解決された。

KL1が同期/排他制御の問題から解放されている分、共有メモリを持つPIMにおける荘園の実装では、両者の問題は重要である。

Multi-PSIと比較すると、単に共有メモリの排他制御が必要になっただけでなく、排他制御の衝突によるオーバヘッドや、クラスタ間でのメッセージの追い越し、さらにクラスタ内での処理の追い越しなどの問題があり、実装方式は再設計を行なった。

実装方針としては、論理型言語のリダクションは、UNIXのプロセスなどと比較すると、非常に小さな粒度の処理単位であり、PIM性能をあげるためには、一連のゴールリダクションの流れを妨げずに高速に実行する必要がある。よって、通常のゴールリダクションにおけるシステムオーバヘッドは極力押えなければならない。一方、荘園を制御する組み込み述語の実行コストは、実行頻度の低さからリダクション時間と比較してかなり高コストでも大目に見ることにした。

1.3 評価の考え方

まず評価対象である荘園を何と捉えるかというと、OSを実現するための道具というよりも、広い意味でのOS機能の1部をカーネルとして実装する試みであると考えたい。すなわち、OSカーネルとして必要最低限の機能を荘園の中に実現し、その他のOS機能はKL1で記述するPIMOSに任せる。「荘園」は、他のシステムと比較するならば、OSカーネルである。

実装の興味は、論理型言語のみをサポートした計算機上での従来OS機能の実現性の検証、実行コストの測定、さらには、従来計算機には無い論理型言語特有のサービス機能の探索とその実現方式などである。

具体的な測定では、FGHCの処理に必要なコストを物差しにして、通常のリダクションにおけるオーバヘッドおよびシステムリクエスト（荘園を制御する組み込み述語の呼出）のコストを測定する。これらの測定対象は、従来計算機OSにおける通常の計算処理におけるシステムオーバヘッドおよびシステムコールのコストに対応する。ただし、PIMのKL1処理系の評価においては、KL1で記述されたPIMOSはアプリケーションとみなし、オーバヘッドに入れないこととする。また、ハードウェアアーキテクチャはPIMに固定して考える。

2 荘園の機能

荘園の機能と論理型言語の中におけるインタフェースを簡単に解説しておく。

- PIM=KL1言語のスタンドアロンシステム

  並列推論マシンPIMはKL1言語のスタンドアロンシステムである。KL1はAND並列論理型言語であり、systemがosとusrからなる場合、例えば次のように記述できる。

  ```
  system:-true|os,usr.
  ```

  しかしこれでは、KL1のAND並列性により、usrの失敗がそのままsystemの失敗になってしまう。

- AND並列における特権プログラム保護機能

  そこで、usrに皮をかぶせてやり、usrが失敗しても成功する組み込み述語shoen:execを提供する。usr述語はこの組み込みの引数として渡され、この中で実行される。これが荘園の1番素朴なイメージである。

```
system:-true|os,shoen:exec(usr).
```

- 論理型言語におけるメタプログラミング機能

  さらに1段進んで、KL1プログラムによるKL1プログラムの制御を考えてみる。これは、荘園とosの間に通信ストリームを張ることによって実現する。Ctlはosからshoenへの、Repはshoenからosへのメッセージをリストでつないだものである。osはRepを見守り、適宜Ctlへ制御情報を送るプログラムとなる。Repにはusrプログラムの失敗をはじめ各種の例外が報告されてくる。また、ExcpMskの設定により、例外報告をエスケープすることも出来る。

```
system:-true|os(Ctl,Rep),
  shoen:exec(usr,Ctl,Rep,ExcpMsk).
```

  また、usrとして他のシステムプログラムを持ってくることも可能であり、任意レベルにネストした多階層メタプログミングが実現出来る。

- 実用的OS記述機能

  並列処理計算機は一群の協調的動作をする並列問題を効率良く実行出来れば良いが、並列処理OSは独立した並列問題を不公平なく実行させる必要がある。このために、資源の分配や、プロセッサの分配を行なう仕組みを入れた。

  ・資源の分配

  資源としては、計算時間、メモリ、I/Oなどが考えられるが、現在は、計算時間の代わりとしてリダクション数を唯一の資源として、これの上限の設定、変更などをサポートしている。例えば、OSでは、資源不足がレポートされたとき上限値を増やしてやることが出来る。

```
os(Ctl,Rep):-Rep=[rsc_low|Rep_next]|
  Ctl=[add_rsc|Ctl_next],
  os(Ctl_next,Rep_next).
```

3 荘園の実装

前節で述べた機能を統合した荘園を、共有メモリを持つ密結合と、それらをネットワークでつないだ疎結合からなる階層構造を持つ並列推論マシンPIMの上に、実装している。PIMのKL1処理系は、KL1処理系記述のためにマクロ展開型仕様記述言語PSLを定め、仮想ハードウェア上に開発している[4]。すでにPIM用のシステム記述言語PSLによるコーディングは終っており、シミュレータの上で評価を行なっている。

なお、ここで述べる実装方式のうち、ネットワークでつながれた分散環境に荘園を実装するアイデアはすでに[2]で述べられた方式である。

3.1 KL1言語における荘園

先に述べたように、荘園とOSはKL1変数を用いた2本(コントロールとレポート)のメッセージストリームにより情報を交換する。

レポートストリームの場合は、KL1で記述されたOSプログラムのゴールがレポートストリーム変数の具体化を待ち、KL1処理系がそれを具体化することにより、情報が伝わる。一方、コントロールストリームでは、OSプログラムがコントロールストリーム変数を具体化したことをKL1処理系に伝える必要がある。このため、コントロールストリームのメッセージに対応した、専用の組み込み述語を用意し、OSによるコントロールストリームの具体化を待って専用の組み込みを発行するKL1プログラム「荘園モジュール」を開発した。現在の実装では、これらの専用組み込みは、荘園を生成したクラスタで発行しなければならない。すなわち「荘園モジュール」はクラスタ間の負荷分散の範囲外である。

3.2 KL1言語処理系実現における荘園

- 荘園と里親

  荘園はKL1プログラムを実行するための一つの閉じた環境を与える。この荘園を複数のクラスタからなる分散環境に実現する方法として、クラスタ毎に、荘園に代わってKL1ゴールの実行環境を保持する「里親」方式を提案した。

- 荘園レコード、里親レコード

  荘園全体の制御情報、資源情報、および里親の配置情報などをまとめて、荘園レコードに格納する。荘園レコードは、荘園を生成したクラスタに置かれ、荘園下のすべての仕事が終り、かつ、OSからコントロールストリームが閉じられるまで存在する。

  一方クラスタ毎の実行環境は里親レコードに格納する。里親レコードは、KL1のクラスタ間負荷分散により、投げられたゴールを受けとった時、そのクラスタにまだ里親がなければ新たに割り付けられ、また里親の下のすべての仕事が終了した時解放される。

  すべてのゴールはいずれかの里親に属している。システムが最初にロードしてくるプログラムの実行のために、システム立ち上げ時に、いずれの荘園にも属さない「ルート里親」を生成する。また、すべての荘園は、荘園を生成したゴールが属していた里親の子荘園リンクにつながれている。

  図1に荘園と里親の階層構造を示す。実線によるリンクは、メモリアドレスの直接リンク、点線によるリンクは、次の荘園里親メッセージ交換で述べる荘園IDによるリンクである。

メモリアドレスで直接リンクされる、里親レコード、子荘園リンク、子荘園レコードの排他制御は、ロックの順を、子荘園リンク、里親レコード、子荘園レコードと定めてデッドロックを回避している。

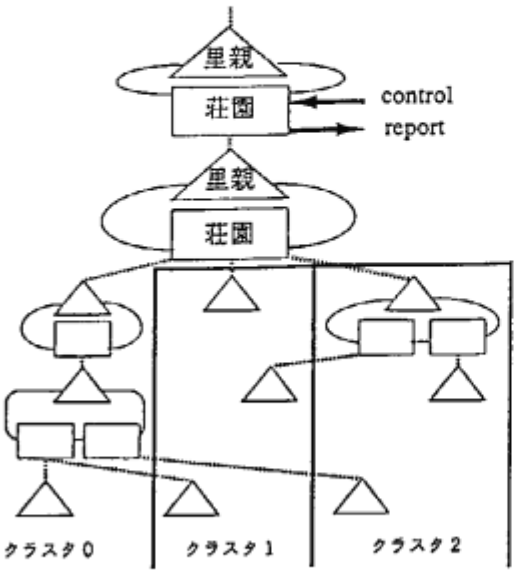

図1. 荘園および里親の階層構造

- 荘園里親メッセージ交換

KL1処理系では、KL1プログラムのデータを、メモリアドレスを共有しない他クラスタからアクセス可能にするため、クラスタ間に渡る変数にはシステムでユニークな外部参照IDをつけ、輸出入表に登録する。このIDは、クラスタ内でKL1データの置かれるヒープメモリでGCが行なわれても不変である。

クラスタ間に渡る荘園の情報は、KL1データの輸出入の枠組を借り、荘園レコードを輸出表に、里親レコードを輸入表に登録することにより実装している。また、荘園を識別する荘園IDも外部参照IDを用いている。

荘園里親間の情報の授受はすべて、荘園IDを付したメッセージの交換による。

KL1処理系の設計に当たっては、クラスタ間のメッセージ転送はオーバヘッドが大きいので第一にメッセージ数を少なくすることが目標となっている。このため、送信のたびにAckを返すようなプロトコルは採用していない。ネットワーク中を飛び交うメッセージには、追い越しがあり得るものとし、荘園がすべての仕事の終了を確認できるように、すべてのメッセージにWTCと呼ぶ重み付の参照数をつけている。また、制御状態の変化を正しく処理するために、メッセージのカウンタを設け追い越しの可能性を検知している。

- ゴール実行制御方式

現在の実装では、ゴールのリダクションを唯一の消費資源としており、リダクション数の上限を定めることで、ゴールの実行制御を行なうことができる。リダクション数の上限値は、初期値が零で、増分値を適宜コントロールストリームから与える。リダクション数が上限値に近付くと、そのことがレポートストリームに報告され、上限値の増分が指定されなければ、新たなゴールリダクションは行なわれない。また、それまでに消費した資源量のトータルの報告を求めることも出来る。なお、荘園における消費資源の積算には、荘園がネストしていた場合、子孫の荘園で消費され多分もすべて勘定する。

さらに、消費資源量によらず直接制御することも可能で、コントロールメッセージとして、スタート/ストップ/アボートを用意している。

- 荘園プロファイリング

指定された荘園の下で実行されるプログラムの、すべての述語について述語毎に実行回数を集計する。荘園がネストしていた場合、すべての子孫の情報も報告する。

里親毎に集計のためのプロファイリングエントリのハッシュテーブルを持つ。KL1プログラムでは、一度スケジュールされたゴールがサスペンドする場合があるので、デキューされた回数と、サスペンドした回数をそれぞれカウントして、その両方を報告する。また組み込み述語がサスペンドした場合には呼びだし元も区別してカウントする。

- ゴール実行環境のキャッシュ

里親レコードにはクラスタ毎の実行環境を保持しているが、クラスタ内に複数のPE(プロセッシングエレメント)を持つPIMでは、里親をアクセスするたびに排他制御していては効率が悪い。このためPE毎に、リダクション数の上限値などをキャッシュすることにより、データアクセスのローカリティを保っている。

- 例外処理

すべての例外は、荘園からレポートメッセージを通じて、KL1プログラムのレベルに報告される。例外を報告するメッセージには、例外の種類や、種類毎の付加的な情報の他に、例外を引き起こしたゴールに代わって実行すべきゴールを指定することが出来る。また、荘園生成時に例外処理マスクを設定することにより、例外の報告をより上位の荘園に回すことが出来る。

3.3 メッセージの追い越しとミスキャッチ

start/stop 制御におけるメッセージの追い越し対策の一つを具体的に解説する。

初めに、図2. を用いてメッセージミスキャッチを説明する。図は縦に時間軸をとり、左に荘園の状態を、右に里親の状態を示している。右側の実線は里親の存在を表し、左側の実線は荘園側で里親の存在を意識している期間である。

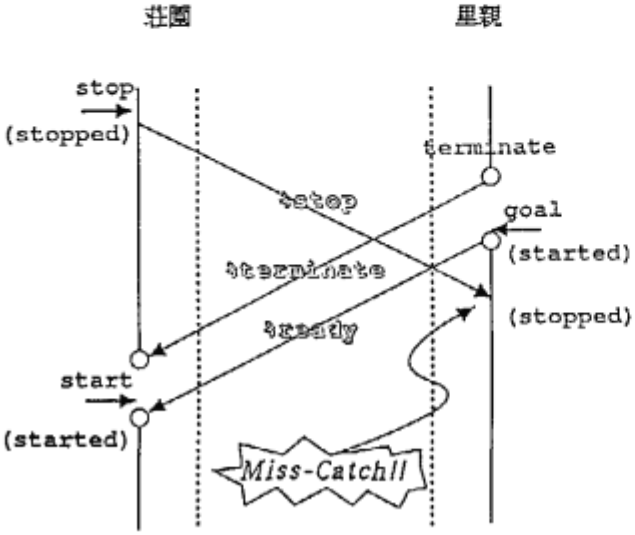

図2. メッセージのミスキャッチ

まず図の左上で、荘園に stop 組み込みが発行されると、荘園は stop 状態になり、また里親に stop メッセージを送る。一方左上で、里親はすべての仕事が終了すると荘園に terminate メッセージをおくって消滅する。この直後に他のクラスタからゴールを受信すると再び里親が生成され、荘園には ready メッセージが送られる。荘園側では、terminate メッセージと ready メッセージの間では、里親がいないと思っているので、この間に start 組み込みが発行されても自分の状態を start にするだけで、里親へはメッセージを出さない。ところが、最初の stop メッセージが、terminate および ready とクロスすると、これを受けとった里親は stop 状態になってしまう。

結果的に荘園と里親の状態が合わなくなっている。これがメッセージのミスキャッチである。Multi-PSI の様にメッセージの追い越しが無い場合には、start 状態にある荘園は、ready を受けとったらねんのため start を送るという方法で逃げることが出来る。しかし追い越しがあると、先に送った stop と後から送った start の着順すら保証されない。この問題を、メッセージの追い越しを考慮しながら解決する必要があった。

start/stop 制御の特徴は、荘園から発行された start/stop メッセージの数は、同数か或は stop が一つ多いのどちらかである。この状態を表すフラグとして stop/start ビットを設ける。そこでこのビットの状態を荘園側で正しく維持できるように、start/stop メッセージがちょうど里親がいない時に着いてしまった時にそれぞれ nak_start/nak_stop を返すようにした。

図3. に start/stop 制御に関する荘園の状態遷移図を示す。

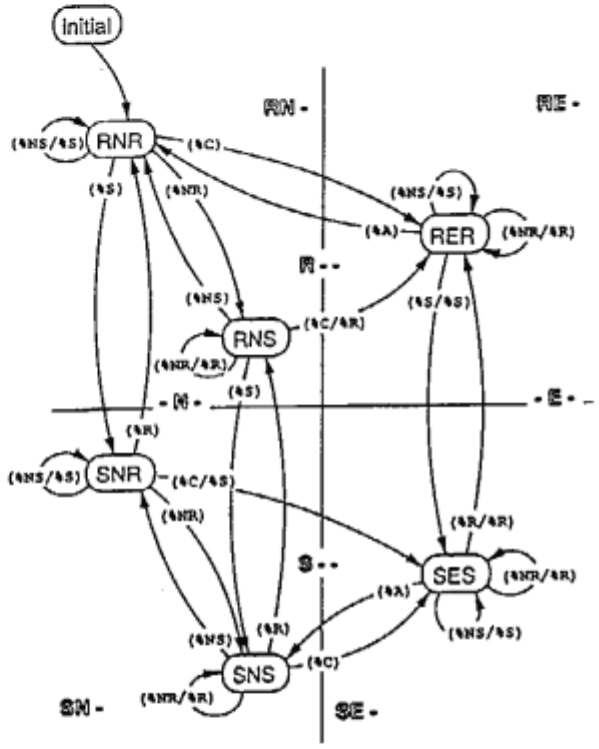

図3.start/stop 制御に関する荘園の状態遷移

遷移図中の記号を説明する。

```
Status (XYZ):
    X =  R: shoen running(started)
      =  S: shoen stopped
    Y =  E: FP existing
      =  N: non-existing
    Z =  R: stop/start bit off
      =  S: stop/start bit on

Input/Output (%X[/%Y]):
    X =  R: built-in run(start)
      =  S: built-in stop
      =  C: receive FP created(ready)
      =  A: receive FP annihilated
```

```
                    (terminated)
  =  NR: receive nak of run
  =  NS: receive nak of stop
Y =  R: send run(start)
  =  S: send stop
```

遷移の例を説明すると、左上の初期値から、状態(RNR)すなわち、荘園はstartで里親は無くstop/startビットはoff。ここで里親からreadyを受信すると、右側の(RER)に移り、さらに組み込みのstopが発行されると、里親にstopメッセージを送って下の(SES)へ移る。

この方式により、ネットワーク上のメッセージが有限時間のうちに受信されるならば、荘園側でも有限時間のうちに正しい状態を把握できる。

4 実行コストの測定

いくつかの荘園機能について、実行コストを測定した。

4.1 開発規模

PSLによるKL1処理系の開発規模は、約５０キロステップにおよび、そのうちのおよそ２割がトレース機能なども含めた、荘園関係のコーディングである。

また、マクロを仮想ハードウェアのマシン語に展開したレベルでは、約１００キロステップの４割ほどを占めている。

仮想ハードウェアのマシン語はPSLの基本マクロとして登録してある。レジスタ間転送、レジスタ上での演算、レジスタメモリ間転送、条件分岐、タグ値分岐、などをサポートする。

4.2 実行時コスト

実行コストの評価は、仮想ハードウェアをC言語で実現して、その上でシミュレータとして動作させ、計測する。ただし、この時、シミュレータの実動作時間ではなく、仮想ハードウェアの１マシン語の実行を単位としてはかることにした。以下の測定結果では、全ての仮想ハードウェアの１マシン語を１ステップと数え、また、共有メモリアクセスにおける、キャッシュミスヒット、バス待ち、ロックの衝突などは計算に入れていない。

リダクションなどのコストと、荘園関係のコストを比較するために、アペンドの１リダクション（の最良のケース）の時間を１アペンドLI（Logical Inference）或は単にLIと記すことにする。

この計り方は、Multi-PSIのKL1処理系の評価で用いられており、ハードウェアの実クロックに拘らずに、処理系の比較が行なえる利点がある[6]。

4.3 測定内容と方法

測定ユニットである１アペンドLIを与えるKL1のアペンドプログラムを以下に掲げる。

```
append([X|A],Y,Z):-true|Z=[X|B],append(A,Y,B).
append(□,Y,Z):-true|Z=Y.
```

append定義の第１節目の再呼出ループを実行するための、PSLプログラムの基本マクロステップ数、すなわち仮想ハードウェアの命令数は、69ステップ。ただし、内３ステップは荘園関係であった。

```
1 LI = 69 steps in PSL primitives
```

4.4 リダクション中のオーバヘッド

- 通常のリダクション処理

  リダクション中のシステムのオーバヘッドの切り分け方として、FGHCの処理として必要な部分と他の部分に分けるのが分かり易い。

  上述のappendのループは、PSL基本マクロで69ステップであり。この内、66ステップがFGHCの処理として必要な部分、すなわち、本来の計算処理で、残りの３ステップが荘園関係その他のシステムオーバヘッドである。66ステップをベースに考えると、オーバヘッド4.5%になる。

  リダクションの66ステップの部分には、例えばユニフィケーションの引数は、ともに単一参照のLISTとREF->UNDEFであるとかの仮定を入れているが、オーバヘッドの３ステップについても、キャッシュされているリダクション数のメンテナンス以外は、何もしなくて良い場合である。

  荘園関係の３ステップは具体的には、資源の１リダクション分消費（減算）、消費結果のテスト、プロファイリングモードであるかのテストの３ステップである。

  次に、システムとして一仕事必要な例を調べてみた。

- プロファイリングデータのインクリメント処理

  指定された荘園の下で実行されたプログラムの述語毎のリダクション数を集計する機能である。実行しようとするゴールがプロファイリングモードの荘園に属している時、述語毎にカウントする。このカウント処理のコストを調べた。

  256エントリのハッシュテーブルにおよそ1000エントリのプロファイリングデータが登録されているとし、コリジョンチェインをたどると平均３つ目で目的のエントリが見つかるとした場合、

処理コスト：96 ステップ =1.39LI

となる。

ここで、1000 エントリのプロファイリングデータとは、集計対象のプログラム中に 1000 の述語ということである。ICOT で開発した実際の大型のアプリケーションプログラム例としは、自然言語の並列構文解析を行なう「PAX」の場合、KL1 でおよそ 8000 行のプログラムであるが、ユーザ述語定義は約 700 である。

このオーバヘッドはリダクションの実行環境を、プロセッサ毎にキャッシュすることでまだ低減できる。しかし、逆に実行環境の切換え時にメインテナンスするデータ量が増えることにもなりトレードオフとなる。

数 N としてある。また、各里親の下に子荘園は無いとする。各里親の仕事は同時に行なわれる

ので、荘園と、適当な一つの里親の仕事を測定する。

処理コスト：895+77N ステップ =13.0+1.12N LI

現在開発中の PIM の最大のクラスタ数を想定した場合、システムの 64 クラスタ全体をを使っている荘園の配下で、全クラスタに里親がいるとすると、N=64 となり、5235 ステップ =75.9 LI となる。

ちなみに、単一のメッセージの送信のおよび受信部分を取り出すと、それぞれ、

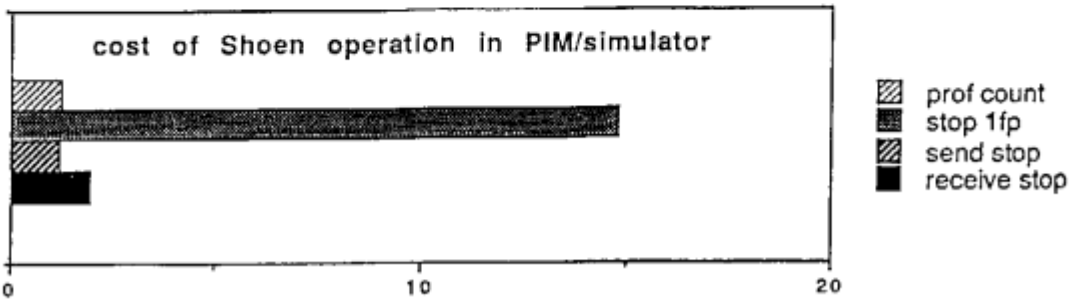

図 4.PIM/simulator における荘園機能のコスト

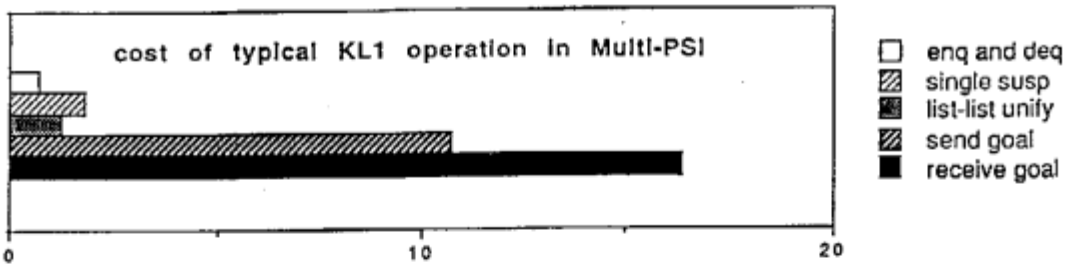

図 5.Multi-PSI における KL1 言語処理のコスト
（[6] より一部抜粋）

実行環境の切換え頻度の見積りが問題となるが、Multi-PSI と比較すると、PIM では、クラスタ内に複数のプロセッサがあり、また述語の再帰呼出を高速に実行するゴールスケジューリングを採用しているので、複数の異なる実行環境が必要なアプリケーションの場合、異なるプロセッサがそれぞれの実行環境を保持すれば、1 クラスタ 1 プロセッサの Multi-PSI よりも切換え頻度がかなり少なくなると考えられる。

4.5 システムリクエストのオーバヘッド

- 組み込み述語 stop_shoen

  ここでは、コストが里親のあるクラスタ数に比例する部分があるので、里親のあるクラスタ数を変

送信：77 ステップ =1.12 LI

受信：126 ステップ =1.83 LI

となる。

測定結果を図 4. にまとめた。参考のため、中島等が行なった Multi-PSI 上の KL1 処理系の評価 [6] からデータの一部を図 5. にまとめてある。LI といっても違うシステムであり、オーダーの比較程度の意味であるが、荘園に停止を命じる組み込み

stop_shoen のコストは、クラスタ数によらない部分でゴールの送受信とほぼ同じオーダのコストを要しているが、ストップメッセージ自身の送受信にはそれほどのコストが掛かっていないことが分かる。

5 まとめ

- 通常のリダクションにおけるオーバヘッド

  荘園機能として必要最低限の処理のみが行なわれた場合、オーバヘッドはアペンドで4.5%である。ユーザ定義述語は、アペンドと比較すると大きいものが多く、このオーバヘッドはもう少し小さく見積もることが出来る。

  プロファイリングデータの採取時には、アペンドでは139%のオーバヘッドである。これはかなり実用の限界に近いところであり、より高効率な代替案の検討が必要である。

- 荘園制御組み込み述語のコスト

  stop_shoen は組み込み述語の中では、処理の簡単な部類に入る。荘園がネストしている場合のコストは、およそ荘園数に比例すると見込める。

  消費資源量の問い合わせの様に、答を集計して報告するような処理のコストの計測はこれから行なう予定。

- 当面の課題

  リダクション中のオーバヘッドのさらなる低減が課題である。PE単位のキャッシュと実行環境の切換えの効率化のトレイドオフとなる。クラスタ内のPE数ほどの異なる実行環境を持つことは、それぞれのPEにそれぞれの里親のプログラムがスケジュールされている限り、効率に悪影響は少ない。

  クラスタ内のPE数は共有メモリバスネックにより、PIMでは8と決めた。しかしこれは、単一代入型のKL1データのアクセスパタンを考慮したものであり、現在の荘園のように、ランダムアクセス可能なシステムデータを大量に抱えるようになってくると、アクセスパタン自身も変化してきている可能性が高い。これはバスネックをより深刻なものにする方向である。

- 長期の課題

  大きな機能がまだカーネルに入り過ぎている。機能の代案、実現方式の代案、まだいくらでも有るはずである。OSをサポートする機能のプリミティブのハード化は良いが、OSをまるごとハード化することは労を多くするのみである。実は、FGHC処理の部分についても同じことがいえると考えている。

- 荘園の新しいとらえ方

  ここまで述べた荘園の機能は主にOSの記述性にあったが、すでに実装された荘園単位の優先度指定や、クラスタ分配機能は、荘園内の実行を大粒度の部分仕事とするような、大型の並列応用プログラムの開発を可能にする。

  このような考えに沿って、荘園に対する優先度指定や、クラスタ分配指定をダイナミックに制御していく方式を検討中である。

6 謝辞

貴重な御意見を頂いたICOTのPIMOS,Multi-PSIおよびVPIMグループ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、ICOT渕所長、内田研究部長、瀧第1研究室長に感謝致します。

参考文献

[1] T. Chikayama, H. Sato, and T. Miyazaki. Overview of the Parallel Inference Machine Operating System (PIMOS). In *Proc. of the International Conference On Fifth Generation Computing Systems 1988*, Tokyo, November 1988.

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

[3] T. Miyazaki. Parallel Logic Programming Language KL1 – Its Implementation and an Operating System in It –. *Transactions of the Institute of Electronics Information and Communication Engineers*, J71-D(8):1423–1432, August 1988. (In Japanese).

[4] 山本礼己 他. 並列推論マシンPIMにおける抽象機械語KL1-Bの実装「高級機 械語を実装するための道具立」. 信学技報 168, 電子情報通信学会, Aug 1989.

[5] 川合英夫 他. 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」. 情処研報 29, 情報処理学会, Apr 1990.

[6] 中島克人, 稲村 雄, and 市吉 伸行. 疎結合並列マシンMulti-PSI上でのKL1分散処理系におけるプロセッサ間通信の評価. In 並列処理シンポジウム *JSPP'90*, pages 369–376. 情報処理学会, May 1990.