TM-1320

PIM/p上のKL1処理系におけるｸﾗｽﾀ間通信方式および荘園里親処理方式の評価

武井　則雄（富士通SSL）
熊谷　勝宏（富士通SSL）
橋本　竜一（富士通SSL）
畑澤　宏善

February, 1995

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

Institute for New Generation Computer Technology

PIM/p上のKL1処理系におけるクラスタ間通信方式および荘園里親処理方式の評価

Evaluation of the Inter-Cluster Communication Method and the Meta Control Method in the KL1 Implementation on PIM/p

武井 則雄　　熊谷 勝宏　　橋本 竜一
(株) 富士通ソーシアルサイエンスラボラトリ †
畑澤 宏善
(財) 新世代コンピュータ技術開発機構 ††

並列推論マシン PIM/p 上の KL1処理系において、クラスタ間通信方式および荘園/里親処理に関しての評価を行なった。クラスタ間通信方式に関する評価では、自輸出と里輸出の割合や、メッセージ別の発生頻度などを調べた。その結果、輸出形態では自輸出、特に %read メッセージの返信先を送る自輸出が多く、メッセージの種類では %read などデータを参照するためのメッセージが全体の9割以上を占めていることを明らかにした。荘園/里親処理に関する評価では、荘園/里親の生成/消滅や里親の切替といった基本的な処理にかかる時間などを調べた。その結果、荘園/里親の生成/消滅全体では、PIM/p 上の 300 append リダクション以上、里親の切替えでは、6 append リダクション程度に相当する時間がかかることが分かった。

1　はじめに

ICOTでは、第五世代コンピュータプロジェクト[1]において、5つの並列推論マシン(PIM)のモジュールを開発し、その上に並列論理型言語 KL1の処理系を実装した。これらの PIMのうち4つのモジュール(PIM/p, PIM/c, PIM/i, PIM/k) は、プロセッサが共有メモリで結合されたクラスタ構造を有している。ICOTでは、これらのクラスタ構造を持った PIMに共通のKL1処理系として VPIM[2]を開発し、これを各PIMに移植/実装した[3]。

このうち PIM/p[4, 5] は、8台の要素プロセッサ(PE) が並列キャッシュメモリを介してメモリ共有結合されたクラスタを、ネットワークによってハイパーキューブ結合する二階層構成をとっている。PIM/p の処理系を含めた VPIM に基づく KL1処理系の特徴として、

- クラスタ内の自動負荷分散機構
- 荘園/里親を使ったメタ制御機構
- 輸出入表を使ったクラスタ間通信方式

を挙げることが出来る。このうち、自動負荷分散機構に関しては、これまでに簡単な評価を行ない、発表した[6]。本報告では、これ以外のクラスタ間通信方式および荘園/里親を使ったメタ制御機構について、PIM/p上での評価方法および評価結果について述べる。

これらの評価を行なうために、従来の KL1処理系とは若干処理方式を変更した**新処理系**を PIM/p上に実装した。従来の処理系と新処理系との主な相違点は 2章に説明した。以下、クラスタ間通信に関する評価について3章に、荘園/里親処理に関する評価について 4章に述べる。

2　新処理系と従来の処理系との相違点

PIM上の KL1処理系のクラスタ間通信方式および荘園/里親処理方式に関する評価を行なうため、新処理系を PIM/p上に実装した[8]。新処理系の処理方式上の特徴[1]は次の二点である。

- 荘園表および里親表の実装
- クラスタ間データ参照の最適化

荘園表および里親表は、荘園/里親間の制御関係の通信を通常の輸出入表とは別の荘園表/里親表を介して行

† Norio TAKEI, Katsuhiro KUMAGAI, Tatsukazu HASHIMOTO, Fujitsu Social Science Laboratory, Ltd.
†† Hiroyoshi HATAZAWA, ICOT.

[1] 新処理系では、ここに述べた処理方式の変更の他に、ソースコードの保守性を高めるための変更も行なった。しかし、これらの変更は性能には影響しないと考えられる。

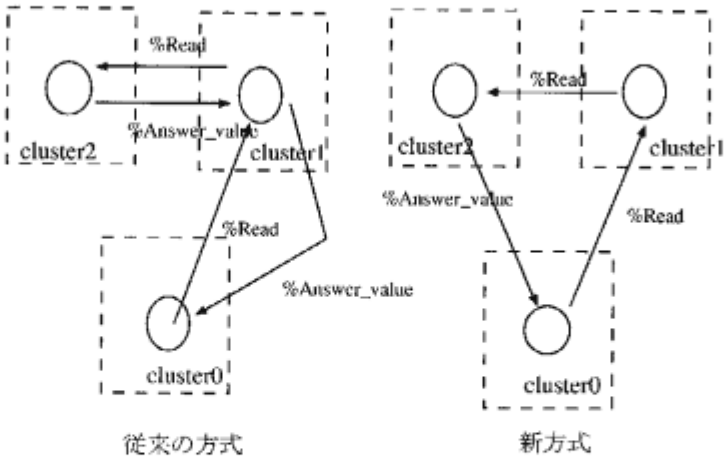


図 1: %readに対する%answer_valueの返信方式

なうために導入した。従来の処理系では、KL1データの通信と輸出入表を共用していたため、輸出入表の排他制御の競合が起こりやすくなっていたと考えられる。そこで、祖園/里親間の制御メッセージに関しては独立のテーブルを用意することにより、排他制御コストの軽減を図った。今回の評価では、新処理系の里親関係の排他制御コストについても調べてみた。

クラスタ間の KL1データの参照は、%readメッセージの送信と%answer_value メッセージによる返信で行なわれる。%readメッセージを受信したクラスタで、読み出しの対象が更に別のクラスタにある場合には、そのクラスタに%read メッセージを中継して送信する。従来の方式では、読み出しデータの返信のための%answer_valueメッセージも、%readメッセージの逆経路で中継していた(図1の左側)。このため、中継したクラスタで当該データの具体化が必要でない場合、%answer_value メッセージの中継は余分な処理となる。新処理系では、これを変更し、このような場合の%answer_valueメッセージを、データの要求元に直接返信できるようにした(図1の右側)。これは、%readメッセージを中継する際に、データの要求元をもともとの要求元にして中継することで実現している。

これによって、返信時の%answer_valueメッセージの中継を省くことができる。ただし、%readを中継したクラスタで、その後当該データの具体化が必要になった場合には、改めて %readメッセージを送信するため、逆にオーバヘッドが生じる可能性もある。従って、変更した方式の妥当性を検証するためには、後者の状態の発生率を測定する必要がある。しかし、この測定を行なうためには、全外部参照セルに %readメッセージを受信したかどうかを示す記録が必要であり、今回の評価では行なっていない。

なお、以下に述べる評価結果は、特に注記したもの以外はすべて、PIM/p の新処理系上で測定した結果に基づいている。

3 クラスタ間通信方式の評価

3.1 評価項目

この報告では、クラスタ間通信方式に関し、以下の項目について評価を行なう。

- 白輸出と黒輸出の比率

  KL1では、クラスタ間のデータ参照について、他クラスタのデータを読み出すことを輸入、他クラスタにデータを送ることを輸出という。このうち、単一参照データの輸出を白輸出、多重参照データの輸出を黒輸出と区別する。そこで、KL1プログラム実行時に発生する輸出形態の内訳について測定した。

- メッセージの発生頻度と処理時間

  KL1処理系のプロセッサ・プロファイル組込述語を用いて、メッセージ別の発生頻度と処理時間を測定した。また、新処理系と従来の処理系とでの発生するメッセージ数の差についても測定した。

- データ型別通信速度

  種々のKL1データ型を通信するプログラムの実行時間から、データ型毎の通信速度を推定し、違いを比較した。

表 1: ベンチマーク・プログラム

プログラム名	内容
life	ライフゲーム[6]
mastermind	全解探索型数当てパズル[9]
pentomino	詰め込みパズル[10]
bestpath	最短経路探索[11]
LPIA	遺伝子情報処理
MGTP	定理証明

- クラスタ間データ参照の最適化の効果

  新処理系で最適化したデータ参照方式に関して、どの程度の性能向上が見込まれるかを評価した。

以下のクラスタ間通信方式の評価に用いるベンチマーク・プログラムを表1に示す。

3.2 白輸出と黒輸出の比率

3.2.1 測定の目的

KL1 データの輸出形態は、データの参照数により**白輸出**と**黒輸出**に大別される。前者は単一参照データの輸出、後者は多重参照データの輸出である。クラスタ間参照データは、重みつき参照カウント方式 (WEC 方式) を用いて参照数管理と GC を行なっている。白輸出の場合、WEC を 1ビットのフラグによって実現し、参照数の管理コストの軽減を図っている。黒輸出の場合、各メッセージに WEC を付加し、輸出側・輸入側双方で参照数を厳密に管理して、GC の際に不要なデータを解放する方式をとっているが、白輸出に比べて処理が重い。データの参照数はプログラムに依存するが、処理系実装方式に対する基礎データを示すために、白輸出/黒輸出の発生割合を測定した。

他クラスタのデータを読み出す際、`%read`メッセージへの返信先アドレスとして、外部参照セルへのポインタを割り付け、これを白輸出する。このポインタは、`%answer_value` の受信時に解放される一時的な輸出データである。この**`%read`返信先の白輸出**は、`%read`メッセージの送信毎に必要なので、発生頻度が高いと予測される。このため、測定ではこれ以外の白輸出と区別した。

一方、黒輸出の中には**間接輸出**が含まれる。間接輸出は、外部参照セルを更に他のクラスタに輸出する際に、解放輸出・分割輸出ともできない場合に行なう。解放輸出とは、外部参照セルに対するクラスタ内の参照が単一参照の場合に、自クラスタの参照を解放して輸出することである。分割輸出とは、手持ちの WEC を分割し、参照を分け与える形で輸出することである。これ以外、つまりクラスタ内で多重参照かつ WEC の分割が不可能な場合が間接輸出となり、外部参照セルへのポインタを黒輸出する。間接輸出は、データの参照関係とその処理を複雑にするが、WEC 補給のために処理が中断してしまうのを防ぐ目的で実装している。その発生頻度を調べることによって処理系実装方針を見直すことができると考え、黒輸出と区別して測定した。

表 2: ベンチマーク・プログラムと実行時のPE構成

プログラム	CL	PE	プログラム	CL	PE
life	32	8	LPIA	32	8
mastermind	32	8	MGTP	32	8
pentomino	32	8			
bestpath	8	8			

CLはクラスタ数、PEはPE数。

3.2.2 測定方法

測定項目には、前節に説明した以下の 4項目を取り上げた。

- 白輸出
- 黒輸出
- `%read`メッセージ返信のための白輸出
- 間接輸出

測定時のPIM/pのPE構成は、表2のとおりである。なお、MGTP と LPIA は PIMOS 上で実行し、それ以外はPIMOSなしで実行した。

測定は、処理系内部にカウンタを組み入れて、各測定項目の頻度の計数を行ない、実行後に集計した。各プログラムとも 5回ずつ実行し、結果を平均した。

3.2.3 測定結果

測定結果を図2に示す。

まず、白輸出と黒輸出の割合は、データの単一/多重参照に依存するため、プログラムにより異なる。白輸出全体 (`%read`メッセージの返信先のための白輸出とその他の白輸出の合計) と黒輸出全体 (間接輸出とその他の黒輸出の合計) について見ると、life と LPIA では、輸出データのほとんどが白輸出であり、逆に、mastermind と MGTP では、黒輸出の占める割合が大きくなっていた。しかし、ここで取り上げたすべてのプ

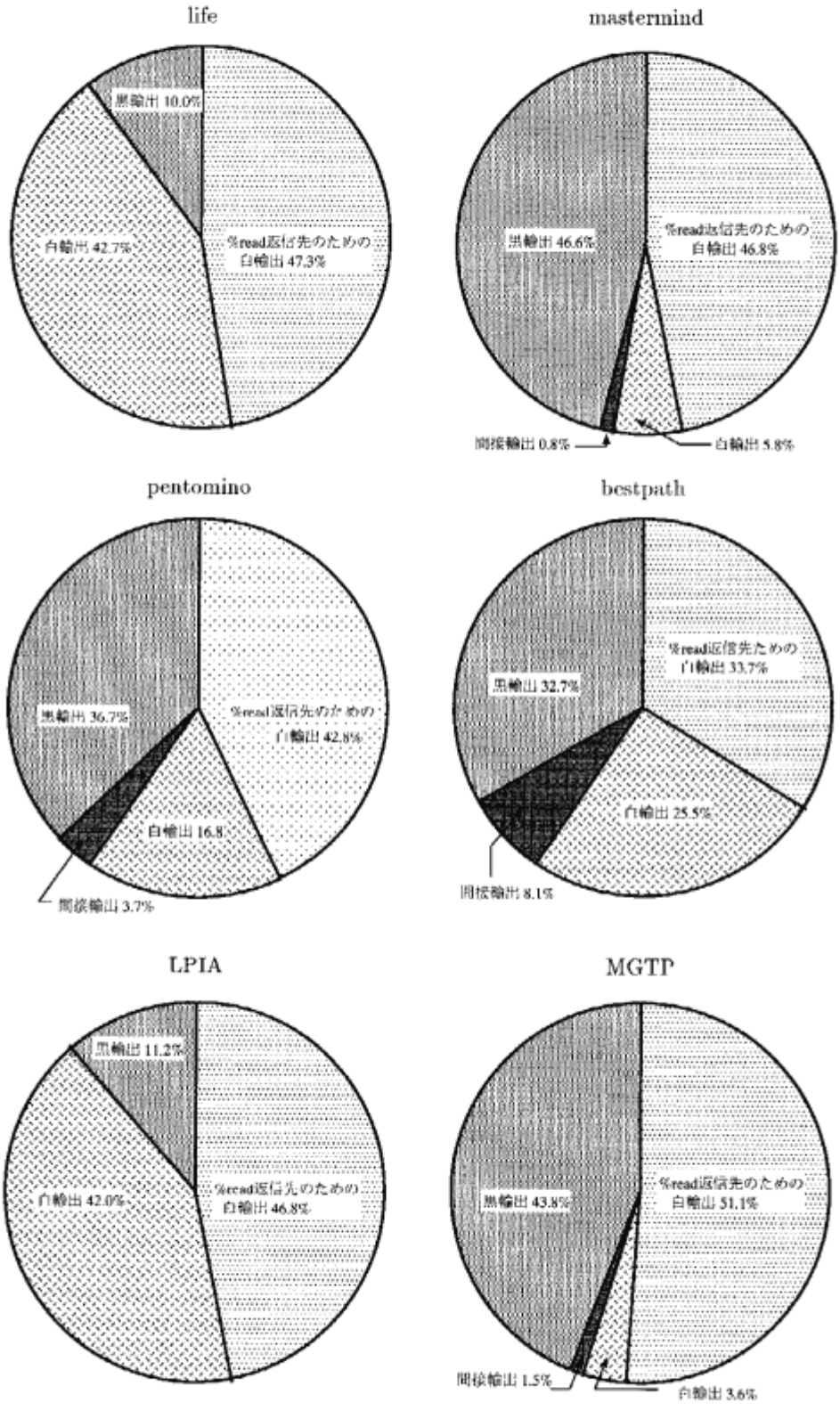

life
黒輸出 10.0%
%read返信先のための
白輸出 47.3%
白輸出 42.7%
mastermind
黒輸出 46.6%
%read返信先のための
白輸出 46.8%
間接輸出 0.8%
白輸出 5.8%
pentomino
黒輸出 36.7%
%read返信先のための
白輸出 42.8%
白輸出 16.8
間接輸出 3.7%
bestpath
黒輸出 32.7%
%read返信先のための
白輸出 33.7%
白輸出 25.5%
間接輸出 8.1%
LPIA
黒輸出 11.2%
%read返信先のための
白輸出 46.8%
白輸出 42.0%
MGTP
黒輸出 43.8%
%read返信先のための
白輸出 51.1%
間接輸出 1.5%
白輸出 3.6%

図 2: 輸出形態の分布

ログラムで白輸出が5割以上を占めているのは、%readメッセージの返信先のための白輸出の割合が大きいためと考えられる。

%readメッセージの返信先以外の普通の白輸出は、通常の単一参照データの輸出を表すが、これと黒輸出全体とを比較すると、lifeとLPIA以外は、黒輸出の方が多い。ただし、黒輸出に関しては、コード・モジュールの輸出分が含まれるため、プログラムを構成するモジュール数が多く、それらが複数のクラスタから参照される場合には、黒輸出の頻度が大きくなる。ここでは、life, mastermind以外のプログラムは、複数モジュールから構成されている。

黒輸出に関しては、以下のことが指摘できる。第一に、クラスタ間負荷分散によって生じるコード・モジュールの輸出は、すべて黒輸出となる。従って、どのようなプログラムでも、他クラスタにゴールを分散する場合には黒輸出が発生する。第二に、データが最初に輸出された時点で多重参照であったものの他に、輸出後に輸出データへの参照が増えて、黒輸出の形態をとる[2]場合もある。このように、輸出後に参照が延びるに従ってデータの属性が変化する場合や、参照の延び方については、今後調べるべき点である。第三に、評価に用いた新処理系では、荘園/単親の通信に輸出入表を使用しなくなった(2章参照)ことで、従来黒輸出扱いをしていた荘園の数の分だけ黒輸出が減少しているはずである[3]。ただし、通常のプログラムでは、荘園を生成することはごく稀である。

白輸出全体のうちに%readメッセージの返信先のための白輸出が占める割合は、いずれも5割以上であった。この部分には、実際には%readメッセージの返信先のための白輸出の他に、WEC補給関係のメッセージの返信先のための白輸出も含んでいる。WECの補給要求に対する供給方法は、新たに黒輸出表を割り当てて輸出を行なう場合と、既にある黒輸出表からWECを分配して供給する場合がある。前者の場合には、WEC補給要求メッセージの送信時に生じる返信先のための白輸出と、これに対する供給時の黒輸出が1回ずつ生じる。後者の場合には、供給時の黒輸出が行なわれないため、白輸出だけが1回カウントされる。MGTPにおいて、%read返信先のための白輸出が、輸出全体の5割以上になっているのは、後者の影響と考えられる。

最後に、間接輸出の全輸出形態に対する割合は、少ないことが分かった。ただし、外部参照セルへのユニファイが許される場合があるため、間接参照がない訳ではない。このユニファイによる間接参照パスの生成については、今回は測定を行なってないため、その発生頻度は不明である。しかし、間接参照セルに対する参照頻度は、%readメッセージの転送回数として測定し、3.5節に結果を示す。

[2]3.3節に述べるWECの補給についての記述を参照

[3]実際の検証はしてない。

3.3 メッセージの発生頻度と処理時間

3.3.1 測定の目的

メッセージ発生頻度は、処理系を設計する上で基礎的なデータとなる。もし、プログラムによらずに発生頻度の高いメッセージが存在すれば、それはクラスタ間通信処理における中心的な処理であると考えられる。これによって実装方式の設計や改良をする上での優先課題を知ることが出来る。

メッセージ処理時間の測定結果は、現状の実装方式の妥当性を評価するための基礎データである。今回は個々のメッセージの処理時間に関しては、十分なデータがとれなかったため、いくつかのベンチマークプログラムについて、プログラムの実行時間全体に占めるメッセージ処理時間の割合を調べた。

3.3.2 測定方法

KL1処理系のプロセッサ・プロファイル組込述語を用いて、各メッセージの発生頻度と処理時間を測定した。制御プログラムによって、ベンチマーク・プログラムの実行開始前に、プロセッサ・プロファイルを開始し、実行終了後にプロセッサ・プロファイラを停止させた。なお、計測の開始/終了はKL1プログラムによって行なうため、制御プログラムの実行自体によるメッセージの発生や、時間の遅れについても測定に含まれてしまう。しかし、その効果は測定結果にはほとんど影響しない。測定はlife, mastermind, pentomino, bestpathの各プログラムについて10回ずつ行ない平均を求めた。

基本的な測定はすべて新処理系上で行なったが、メッセージ発生数については、従来の処理系でも測定を行ない、その結果を比較した。

3.3.3 メッセージ発生頻度の測定結果

各ベンチマーク・プログラム実行時の、新処理系上でのクラスタ間メッセージ発生頻度を、表3と4に示す。この結果は以下のように解釈できる。

- %throw_goalは他のクラスタへ負荷分散されたゴールの数を表す。

表 3: メッセージ別発生頻度(1)

	life(32cl×8pe)				mastermind(32cl×8pe)			
	送信	(%)	受信	(%)	送信	(%)	受信	(%)
%throw_goal	1368	(0.53)	1368	(0.53)	70520	(2.79)	70520	(2.79)
%ready	0	(0.00)	31	(0.01)	0	(0.00)	31	(0.00)
%terminated	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%start	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%stop	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%abort	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%read	124153	(47.93)	124215	(47.94)	782521	(30.91)	782583	(30.91)
%answer_value	124164	(47.94)	124184	(47.93)	781533	(30.87)	781564	(30.87)
%release	2299	(0.89)	2330	(0.90)	793214	(31.33)	793245	(31.33)
%unify	6948	(2.68)	6948	(2.68)	94043	(3.71)	94043	(3.71)
%exception	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%shutdown	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%request_wtc	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%supply_wtc	31	(0.01)	0	(0.00)	31	(0.00)	0	(0.00)
%return_wtc	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%req_resource	3	(0.00)	3	(0.00)	0	(0.00)	0	(0.00)
%sup_resource	34	(0.01)	3	(0.00)	31	(0.00)	0	(0.00)
%ret_resource	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%ask_stat	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%answer_stat	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%sh_profile	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%sh_prof_req	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%synchro_gc	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%req_BEXID_WEX	2	(0.00)	2	(0.00)	0	(0.00)	0	(0.00)
%req_BEXID_BEX	0	(0.00)	0	(0.00)	4867	(0.19)	4867	(0.19)
%sup_BEXID	2	(0.00)	2	(0.00)	4867	(0.19)	4867	(0.19)
%nak_start	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%nak_stop	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
総数	259004		259086		2531627		2531720	

- 本来 %read と %answer_value は一対一に発生し、同数になるはずである。小さな違いについては、計測の開始 / 終了のタイミングの問題と考えられる。bestpath では、他のプログラムに比べてその差が著しいが、これは新処理系で導入した %readメッセージの転送が多発したためと考えられる(3.3.4節参照)。

- %release は外部参照セルへの参照が全くなくなり、クラスタ間の輸出入関係が不要になった時に、データの輸出元に対して輸出表の解放を依頼するために送信される。

- データへの参照を表す %read, %answer_value, %unify, %releaseの4つのメッセージを合計すると、すべてのプログラムで全メッセージ数の9割を越えた。これより、データ参照に関するメッセージ処理が、クラスタ間通信の中心であると言える。以下、これらをデータ参照系メッセージと呼び、%throw_goalと合わせて後に解析する。

- %ready は新たに里親が生成されたことを示す。bestpath 以外では、実行開始時のクラスタ以外の各クラスタに里親が 1つずつ生成された。

表 4: メッセージ別発生頻度(2)

	pentomino(32cl×8pe)				bestpath(8cl×8pe)			
	送信	(%)	受信	(%)	送信	(%)	受信	(%)
%throw_goal	1366	(1.89)	1366	(1.88)	8242	(6.71)	8242	(6.71)
%ready	0	(0.00)	31	(0.04)	45	(0.04)	52	(0.04)
%terminated	0	(0.00)	0	(0.00)	46	(0.04)	46	(0.04)
%start	0	(0.00)	0	(0.00)	1	(0.00)	1	(0.00)
%stop	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%abort	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%read	23839	(32.92)	23901	(32.93)	38677	(31.50)	38691	(31.50)
%answer_value	22094	(30.51)	22187	(30.57)	17869	(14.55)	17911	(14.58)
%release	22768	(31.44)	22799	(31.42)	33355	(27.16)	33362	(27.16)
%unify	2102	(2.90)	2102	(2.90)	16630	(13.54)	16630	(13.54)
%exception	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%shutdown	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%request_wtc	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%supply_wtc	31	(0.04)	0	(0.00)	52	(0.04)	45	(0.04)
%return_wtc	0	(0.00)	0	(0.00)	79	(0.06)	79	(0.06)
%req_resource	0	(0.00)	0	(0.00)	10	(0.01)	10	(0.01)
%sup_resource	31	(0.04)	0	(0.00)	62	(0.05)	55	(0.04)
%ret_resource	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%ask_stat	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%answer_stat	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%sh_profile	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%sh_prof_req	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%synchro_gc	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%req_BEXID_WEX	93	(0.13)	93	(0.13)	3862	(3.15)	3862	(3.14)
%req_BEXID_BEX	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%sup_BEXID	93	(0.13)	93	(0.13)	3862	(3.15)	3862	(3.14)
%nak start	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
%nak_stop	0	(0.00)	0	(0.00)	0	(0.00)	0	(0.00)
総数	72417		72572		122792		122848	

表 5: 1リダクションあたりのメッセージ発生頻度

	life	mastermind	pentomino	bestpath
%throw_goal	0.0038	0.0372	0.0002	0.0023
%read	0.3435	0.4124	0.0027	0.0110
%answer_value	0.3435	0.4119	0.0025	0.0051
%release	0.0064	0.4180	0.0025	0.0095
%unify	0.0192	0.0496	0.0002	0.0047
総メッセージ	0.7166	1.3341	0.0081	0.0081
リダクション数	361416.8	1897593.5	8953085.4	3524251.5

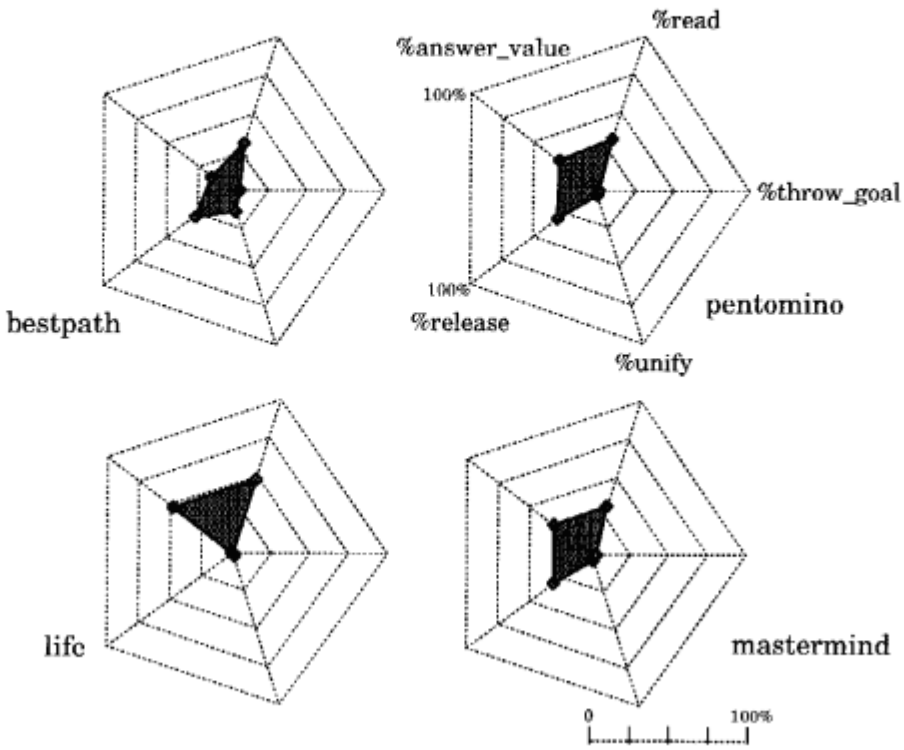


図 3: ゴールの負荷分散とデータ参照系メッセージの総メッセージ数に占める割合

- %ready, %supply_wtc, %supply_resource の送受信数の違いも、計測の開始 / 終了のタイミングの問題であると考えられる。

- WEC の補給要求である %request_BEXID には2つの種類がある。%request_BEXID_WEX は輸入した白外部参照データの参照数が、2以上に増えた場合に黒外部参照への変換を要求するもの。%request_BEXID_BEX は輸入した黒外部参照データの参照数の増加により、WEC の分割で参照数を増やせなくなった場合に WEC の補給を要求するものである。また、%supply_BEXID は %request_BEXID による要求に対して WEC を補給するものである。 pentomino, bestpath では白外部参照から黒外部参照への変換要求が多かったことがわかる。 mastermind の結果は黒外部参照の WEC が頻繁に不足したことを示しているが、これは黒外部参照への参照数がかなり増加したことを意味している。

  輸入したデータを分割輸出できる回数は、 WEC の初期値とその分割方法に依存している。現在の実装では、32クラスタ構成では、全てのクラスタに分割輸出することはできない。 WEC の初期値については輸出側で管理可能な範囲でパラメータ化できるので、システム構成などに合わせてチューンするべきものと考えられる。

- %request_resource は、里親から荘園に対して資源の補給要求を行なう時に送信する。逆に、荘園から里親に対して資源を供給する時には、%supply_resource を送信する。mastermind と pentomino では里親生成時に供給された資源で十分であり、life では3回補給が行なわれた。

ここでは、ゴールの負荷分散を表す %throw_goal メッセージと、4つのデータ参照系メッセージ(%read, %answer_value, %release, %unify) に注目する。この5つのメッセージの送信頻度[4]を、各ベンチマーク・プログラムの通常実行時のリダクション数で割り、1リダクションあたりのメッセージ発生頻度を求め表5に示した。リダクション数は 10回実行した時の平均値を使用した。

その結果、life や mastermind は特に通信の多いプログラムであることがわかった。最も多い mastermind

[4]以下の議論は、メッセージの送信を元に行なうが、受信数を使っても大差ない。

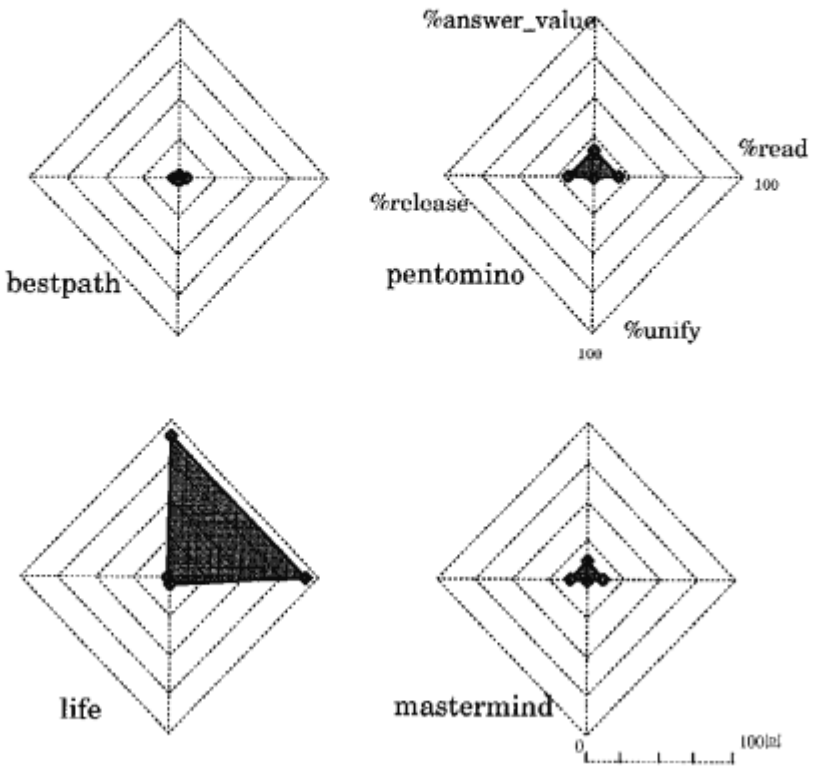


図 4: %throw_goal 1回あたりのデータ参照系メッセージの頻度

表 6: %throw_goal 1回あたりのデータ参照系メッセージの頻度

	life	mastermind	pentomino	bestpath
%read	90.76	11.10	17.45	4.69
%answer_value	90.76	11.10	16.17	2.17
%release	1.68	11.25	16.67	4.05
%unify	5.08	1.33	1.54	2.02

では、ほぼ 1リダクションに 1回の通信がある。また life プログラムは、ストリームを使ってプロセス間でデータを交換しながら計算を進めるプログラムなので、通信が多いと考えられる([6]参照)。

また、これらの 5つのメッセージの発生頻度 (ここでは総メッセージ数に対する割合)を、グラフ化したものを図3に示す。この図から、life では他の 3つのベンチマーク・プログラムに比べ %release メッセージが少ないことが見てとれる。これは自輸出データが多く、%readメッセージ送信時に解放輸出できた場合が多かったためと推測される。mastermind と pentomino ではメッセージ発生の状態がよく似ている。%unify メッセージが少ないのは、他クラスタからデータを具体化されることが少なかったためと推測される。%release メッセージが %read, %answer_value と同程度発生しており %answer_value の受信によって、輸出表がほとんど解放できていると考えられる。 bestpath では %read メッセージに対する%answer_value メッセージが少ない。これは3.3.4節や3.5.3節に記したように、他のベンチマーク・プログラムに比べて %read の転送が多かったためと考えられる。

%throw_goal メッセージの発生頻度を 1 とした時のデータ参照系メッセージの発生頻度を表6に示した。また、図4は、図3と同形式でグラフ化したものである。

図4では、色のついた部分の面積の大きさが、1回の %throw_goal あたりどれだけ多くの通信が起こったのかを表し、

bestpath < mastermind < pentomino < life

の順番で多いことがわかる。

また、これらのベンチマーク・プログラムでは、%readの送信に比べて write (図4では%unify) するプロセスは少ないことが読みとれる。 life は、一旦負荷分散を行なうと、それによって生成されたプロセスが存続し、プロセス間でストリーム通信が行なわれる。その結果、他のプログラムに比べてメッセージ数が多くなる。

3.3.4 新旧処理系でのメッセージ数の比較結果

新旧処理系上でメッセージ発生数を測定し、そのうちデータ参照系メッセージの送信数について表7に比較結果を示した。測定は pentomino と bestpath のプログラムを 8クラスタ 8PE構成の PIM/p 上で 10回走らせ、結果を平均した。

この結果、以下のようなことが分かった。従来の処理系では、%readメッセージと %answer_value メッセージの発生数はほぼ同程度であったが、新処理系では %read の送信数は変化しなかったが、%answer_value の数が大きく減少した。これは、データの返信を直接要求元のクラスタに送るようにしたためである(2章参照)。特に、bestpath でその効果が大きい。また、%releaseメッセージの減少については次のように考えられる。従来の処理系では、%readを中継したクラスタが返信である%answer_valueも中継し、その時に輸出表を解放するための%releaseメッセージを送出していた。これに対して、新処理系では %answer_value が直接要求元に送られるようになったため、%readを中継したクラスタからの %release送出が減少した。

3.3.5 メッセージ処理時間の測定結果

プロセッサ・プロファイラで得られるメッセージのエンコード / デコード時間を合計して、メッセージ処理時間とした。また、ベンチマーク・プログラムの実行時間をプロセッサ台数倍することで、システム内の総実行時間の近似値[5]とし、総実行時間に対するメッセージ処理時間の割合を求めた(表8)。

この結果を見ると、life が最大の割合を示している。life は、1リダクションあたりのメッセージ発生頻度でも高い数値を示しており(表5を参照)、やはり通信の多いプログラムであることがわかる。一方、1リダクションあたりのメッセージ発生頻度が最も高かった mastermind は、メッセージ処理時間では life の半分以下の割合に留まっている。この原因について本節の測定結果からは説明できないが、総実行時間に含まれるアイドル時間が大きいためと推測される。

2番目にメッセージ処理時間の割合の高い bestpath は、1リダクションあたりのメッセージ発生頻度は、100リダクションに1回未満と life などに比べ低い。bestpath は、3.2節で述べた間接輸出の割合(図2を参照)や、%read メッセージの転送の頻度(表15を参照)から、外部参照セルを介した間接参照の多いことが分かっている。このことから、1回の参照で必要となるメッセージが複数のクラスタを経由して、メッセージ数を増幅している可能性を指摘できる。また、表4に示したメッセージ別の頻度から、bestpath では、荘園/里親間の通信メッセージが他のプログラムより多いことがわかる。ここで計測している処理時間には、通信メッセージに対応した荘園/里親の操作時間が含まれるため、全体のメッセージ処理時間が大きくなる可能性がある。但し、双方の事象の検証をしていないため、これに対する適切な説明は、より詳細な調査・解析を必要とする。

全体としてみると、メッセージ処理時間の割合は、総実行時間の 2割から 4割を占めている。メッセージの発生数は、プログラムに強く依存するものであるが、これが高いもの程、クラスタ間処理の効率が実行性能に影響し易くなる。また、3.3.3節の結果から、データ参照に関わるメッセージの発生頻度が総じて高いことから、データ参照系メッセージの処理効率が実行性能に大きく影響すると考えられる。

3.4 データ型別通信速度

3.4.1 測定の目的

KL1 データの型毎に、クラスタ間の通信速度を測定した。クラスタ間データの参照では、輸出形式の違いから、データ型によって読み出しの操作回数も異なる。このため、用いるデータ型によって通信に要する時間も異なることが予想される。また、データの参照数によって輸出形態が白輸出 /黒輸出に分かれ、輸出処理そのもののコストの違いが通信時間にも影響すると思われる。以上のことから、KL1 プログラム設計の際に、最適なデータ構造を選択する目安として、通信速度の評価を行なった。ここでは、各データ型の白輸出および黒輸出データを用いた通信プログラムの実行時間を測定することで、各々の通信時間を算出した。

[5]ただし、この中には各プロセッサが同期待ちのためにアイドルしていた時間もかなりの割合で含まれることに注意すべきである。

表 7: 新旧処理系でのメッセージ送信数の変化

	pentomino				bestpath			
メッセージ種別	従来版	(%)	新処理系	(%)	従来版	(%)	新処理系	(%)
%read	18852	(31.6)	18558	(32.8)	38466	(27.2)	38581	(31.5)
%answer_value	18838	(31.6)	17090	(30.2)	33919	(24.0)	17788	(14.5)
%release	18880	(31.7)	17679	(31.3)	41236	(29.2)	33259	(27.2)
%unify	1841	(3.1)	1815	(3.2)	16660	(11.8)	16622	(13.6)

メッセージ種別は、データ参照系のみ抜粋。百分率は、データ参照系以外も含む全メッセージ送信数に対する割合である。

表 8: メッセージ処理時間と総実行時間に占める割合

	メッセージ処理時間(msec)	総実行時間 (msec)	メッセージ処理時間の割合(%)	システム構成 CL×PE
life	37729.30	89260.80	42.27	8×8
bestpath	46942.74	154649.60	30.35	8×8
mastermind	331089.46	1771212.80	18.69	32×8
pentomino	118367.39	684518.40	17.29	32×8

3.4.2 データ型別の輸出形式

データの輸出はレベル 0 輸出、レベル 1 輸出の 2段階にわかれる。レベル 0 輸出では、データのポインタのみを輸出する。レベル 1 輸出では、データの実体を輸出する。但し、これは表9に示すようにデータ型によって適用が異なる。

この輸出形式とデータ参照操作との対応は、次のようになる。

1. %throw_goal メッセージで送られる負荷分散ゴールの引数データは、レベル 0 輸出を行なう。
2. 最初に輸出されたデータがポインタであった場合、その参照の際に %read メッセージでデータ本体が問合わされる。その返信の %answer_value メッセージでは、レベル 1 輸出を行なう。
3. ここで輸出されたデータに更にポインタが含まれている場合、必要とするデータ構造すべてが得られるまで、%read　%answer_value を繰り返す。

この3番目の項目では、リストやベクタなどの構造を持つデータを想定している。ネスト構造を持たないベクタでは、1回の読み出しですべての要素が取り出せるのに対して、リストやネスト構造を持つベクタの場合には、すべてのデータ構造を得るにはネスト段数分の読み出しが必要になる。

3.4.3 測定方法

以下に示す各データ型について、各々単一参照/多重参照の場合の通信速度を測定する。

- アトミック(アトム/整数)
- 浮動小数点数
- リスト
- ストリング
- ベクタ
- モジュール

1回の通信で送受するデータ数は、10~50の範囲で変化させてみた。ベクタでは幅方向(要素幅)と深さ方向(ネスト段数)の両者を変化させた。ストリングおよびモジュールは、データ本体のダブルワード数[6]を変化させた。また、モジュールは常に多重参照扱いとなる。

上記の各データ型を 2クラスタ間で送受するプログラムを作成した。リストとベクタの通信では、構造体中に未定義変数を格納し、これを用いてストリーム処理で通信を繰り返す。アトミックおよび浮動小数点数の通信では、通信 1回毎にゴールを投げ、その引数を用いて通信を行なう。この場合、データの通信に要する時間のみを推定するためには、ゴール分散などのコストを差し引く必要がある。そこで、輸入したデータの参照を行なわない比較プログラムを作成して用いた。測定は、通信を

[6]1ダブルワードは 8バイト。

表 9: データ型別の輸出形式

データ型	レベル 0 輸出	レベル 1 輸出
整数	–	常にレベル 1 輸出
アトム	–	常にレベル 1 輸出
リスト	ポインタ	先頭のリスト・セルの car 部と cdr 部のレベル 0 輸出
ベクタ	ポインタ	すべてのベクタ要素のレベル 0 輸出
浮動小数点数	–	常にレベル 1 輸出
ストリング	ポインタ	ストリング本体
モジュール	ポインタと構造体 ID	制御情報部分のレベル 0 輸出とコード本体

1000回繰り返すプログラムを 10回ずつ実行し、結果を統計的に処理した。

3.4.4 データ型別の測定結果

各プログラムの測定結果を図5 ～ 8 のグラフに示した。グラフには、各データ数での測定値の平均値と、通信時間のデータ数に関する回帰直線を示した。図中の記号の意味は以下の通り。測定値の平均値には、以下に示す average と diff の二通りの場合がある。

- wex ··· 白輸出
- bex ··· 黒輸出
- no susp ··· サスペンド回避プログラム
- average ··· 各データ数での測定値の平均値
- diff ··· 各データ数での測定値の平均値 (通信プログラムと比較プログラムの差)

サスペンド回避プログラムは、後に述べるように多重参照時に、KL1 プログラムでalternatively を用いて、アイドル状態にならないようにした場合である。

表10および11に、通信での測定結果の回帰直線式を示す。表中の回帰直線式で、y は通信に要する時間、x はデータの送信個数を意味する。

表10と11の各回帰直線式において、x の係数を各データの通信速度と考えると、表12に示すような結果を得る。表では、データの送信個数に関わらず、参照が 1 回の%read – %answer_value で完了するものと、データの送信個数分の通信が繰り返されるものとに分けて示した。

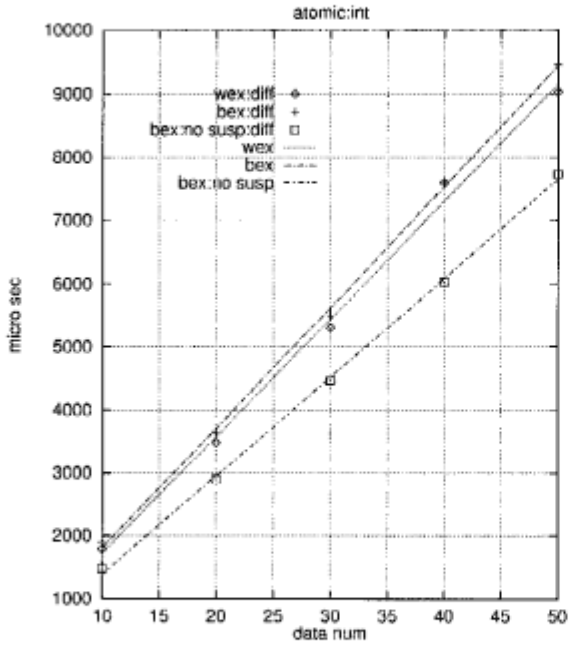
最も通信速度が速いものは、ストリングおよびモジュールとなっているが、これは両者ともデータの送信個数に関わらず、通信は1回しか行なわれないためである。つまり、この速度は 1ダブル・ワードの転送速度に相当すると考えられる。この点で、2つのデータ型には差が見られない。表中のストリングの通信では、データの受信後に要素の参照を行なっている。モジュールではデータの具体化のチェックのみ行ない、要素の参照は行なっていない。この 2つを比較するため、ストリングの要素の参照を行なわない場合を測定したところ、5.6μsec程度になった(図8 の "bcx:wait")。これを見ると、モジュールの通信速度より若干速くなっている。これは、データ構造の生成コストの差とモジュールでの構造体管理コストによるものと考えられる。

ベクタのネスト構造を持たないものは、ストリング、モジュールに次ぐ通信速度になった。前者と後者の差は、前者ではデータ構造の生成後に要素の書き込みを行なっているのに対して、後者ではほぼデータ構造の生成コストしかかかっていないことが原因と考えられる(ストリングでは要素が設定されているが、これはストリング全体を構造体定数としているため、コンパイル時に設定されている)。

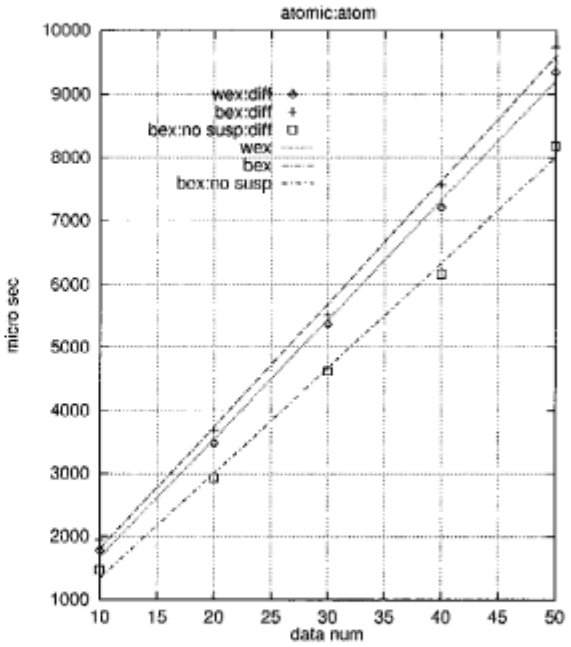
アトミック・データと浮動小数点数データの通信では、個々のデータは1回の輸出でデータ本体を送ることができるが、各データをゴールの引数としているため、1データずつ %read – %answer_value の通信を行なっている。浮動小数点数データの通信速度は、アトミック・データよりも遅くなっているが、これは、データの受信後にガード・ユニフィケーションを行なっていることと、1つのデータが 1ダブルワード (アトミック・データの 2倍、但し通信量としては、1.5倍[7])の大きさを持つことが原因であると考えられる。

リストやベクタのネスト構造を持つものは、通信速度が最も遅いという結果が出た。これは、データを参照するのにネスト段数回の %read – %answer_value の通信を必要とするためと推測される。この場合、各通信では参照する要素のデータ本体と次の要素へのポインタが対になって送受されている。ネスト構造を持つデータの方

[7]KL1 のデータ表現と通信時のコード化したデータ表現との違いによる。PIM/p の通信時には、タグ部は 4バイトの表現となる。このため、アトミック・データは 8バイト、浮動小数点数は 12バイトの通信量になる。



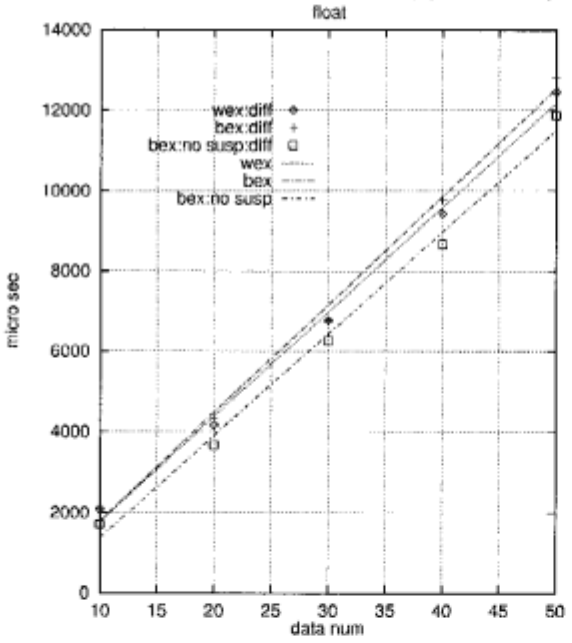
整数



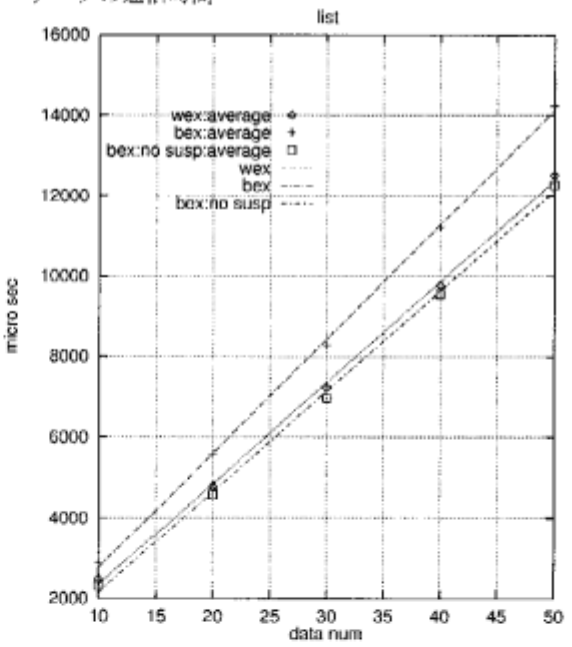
アトム

wex：単一参照 bex：多重参照　横軸はデータの送信個数 縦軸は通信時間(μsec)

図 5: アトミック・データの通信時間



浮動小数点数



リスト

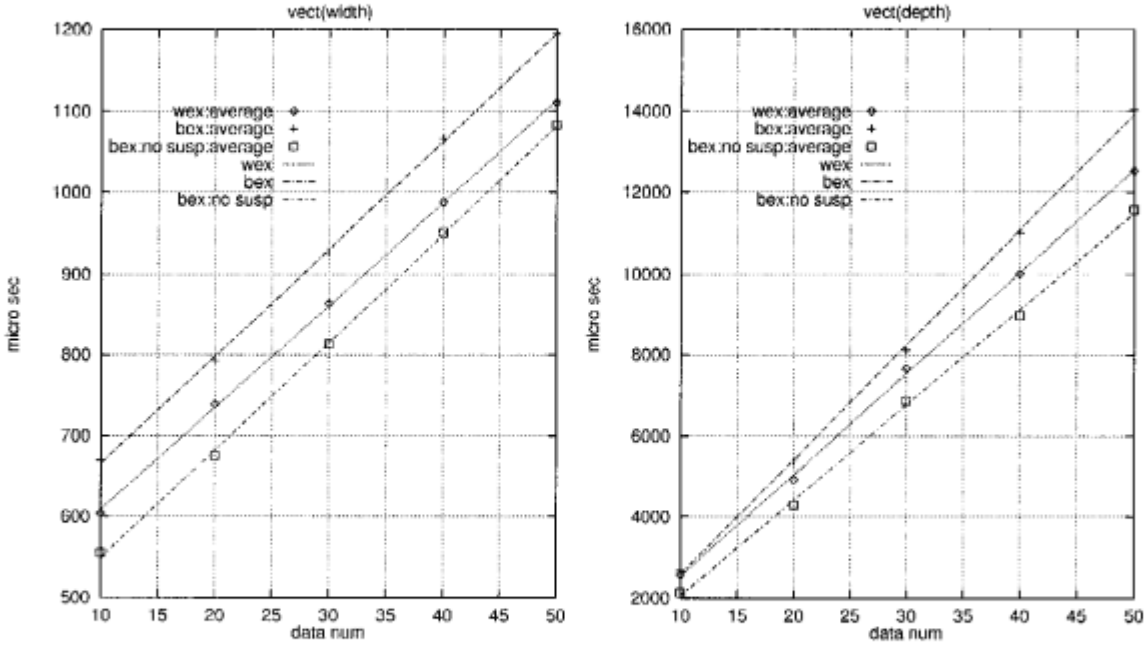
wex：単一参照 bex：多重参照　横軸はデータの送信個数 縦軸は通信時間(μsec)

図 6: 浮動小数点数 / リストの通信時間

が、持たないデータよりも参照コストが大きいことは、クラスタ内でも同様であるが、クラスタ間参照ではその差が非常に大きいと言える。

次に、単一参照と多重参照との比較をすると、ほぼすべてのデータ型で多重参照の場合の方が遅いことがわかる。これは、多重参照の場合の方が、データの輸出入に関わる処理が複雑なためである。但し、表中に示したように、多重参照でも通信処理の間にアイドル状態を作らないようにすると、単一参照よりも速くなるという結果が出ている。この結果は、輸出入処理の違いよりも、アイドル状態になるかどうかの方が性能に大きく影響することを示している。

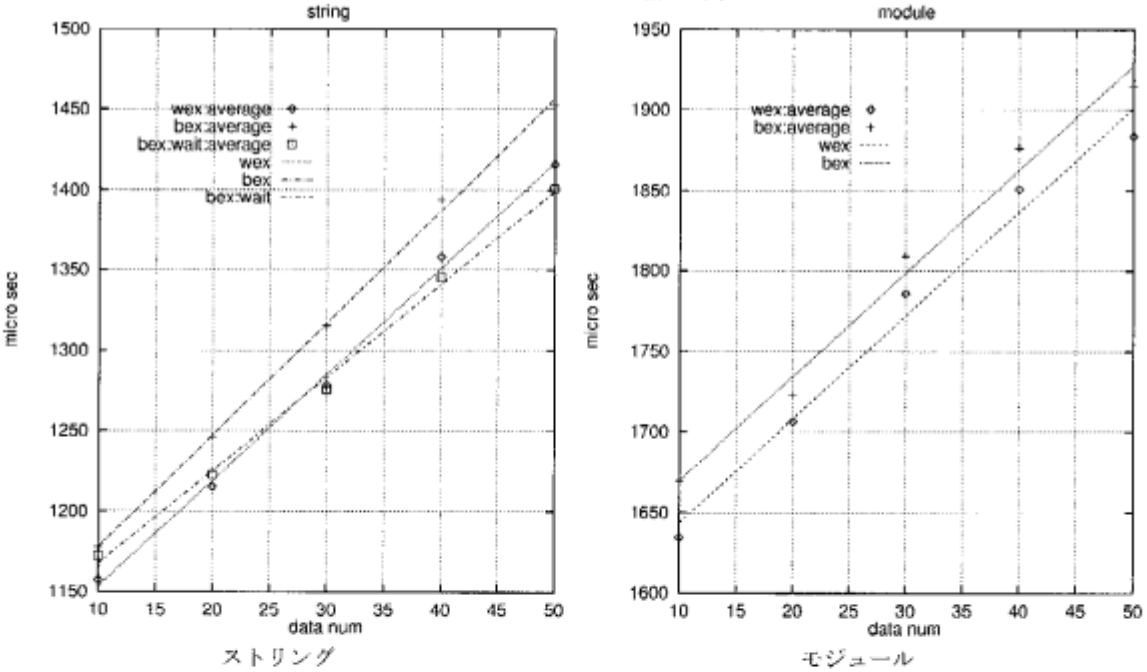
以上のような結果から、KL1 プログラムの通信には、ベクタやリストが使われることが多いが、できるだ



ベクタ(幅)　　ベクタ(深さ)

wex：単一参照　bex：多重参照　　横軸はデータの送信個数　縦軸は通信時間(μsec)

図 7: ベクタの通信時間



ストリング　　モジュール

wex：単一参照　bex：多重参照　　横軸はデータの送信個数　縦軸は通信時間(μsec)

図 8: ストリング/モジュールの通信時間

け平坦なベクタ構造がクラスタ間通信には有利であると指摘できる。

3.5　データ参照方式の最適化の効果

3.5.1　測定の目的

ここでは、新処理系で導入した外部参照セルに対する`%read`メッセージの転送処理(2章参照)について、その効果を評価する。

新旧処理系上で実行した、同じベンチマーク・プロ

表 10: 単一参照データの通信時間の回帰直線式

x：データの送信個数　y：通信時間(μsec)

データ	読み出し	回帰直線式
整数	あり	$y = 694.5900 + 236.0350x \pm 173.7148$
	なし	$y = 838.4800 + 50.0180x \pm 13.4612$
	差分	$y = -143.8900 + 186.0170x \pm 208.0957$
アトム	あり	$y = 623.9200 + 238.4180x \pm 94.3580$
	なし	$y = 842.8300 + 49.9510x \pm 12.5395$
	差分	$y = -218.9100 + 188.4670x \pm 130.8581$
浮動小数点数	あり	$y = 17.8700 + 309.7950x + 236.4567$
	なし	$y = 842.0100 + 49.8010x \pm 10.9410$
	差分	$y = -824.1400 + 259.9940x \pm 308.4114$
リスト		$y = 171.7100 + 250.7610x \pm 121.0167$
ベクタ(幅)		$y = 482.3400 + 12.6160x \pm 6.4563$
ベクタ(深さ)		$y = 38.7800 + 249.9420x \pm 91.1944$
ストリング		$y = 1087.0600 + 6.5960x \pm 7.4619$
モジュール		$y = 1579.1600 + 6.4300x \pm 14.5228$

表 11: 多重参照データの通信時間の回帰直線式

x：データの送信個数　y：通信時間(μsec)

データ	読み出し	回帰直線式
整数	あり	$y = 759.6700 + 264.6410x \pm 94.1158$
	なし	$y = 869.9400 + 73.6720x \pm 8.8396$
	差分	$y = -110.2700 + 190.9690x \pm 108.0787$
アトム	あり	$y = 721.9800 + 268.2600x \pm 129.3138$
	なし	$y = 868.2800 + 73.8000x \pm 7.1678$
	差分	$y = -146.3000 + 194.4600x \pm 161.7413$
浮動小数点数	あり	$y = -63.5400 + 345.4280x \pm 277.3045$
	なし	$y = 835.0200 + 76.8520x \pm 9.7779$
	差分	$y = -898.5600 + 268.5760x \pm 349.9411$
リスト		$y = -78.6300 + 283.7230x \pm 117.0930$
ベクタ(幅)		$y = 533.2800 + 13.2300x \pm 5.7634$
ベクタ(深さ)		$y = -250.6600 + 283.1500x \pm 102.5961$
ストリング		$y = 1107.7800 + 6.9720x \pm 5.0435$
モジュール		$y = 1605.5200 + 6.4280x \pm 13.0302$

グラムの実行時間からは、その効果による差が認められなかった。これは、%readメッセージの転送によって%answer_valueメッセージが減少しても、そのような効果が他の同期待ち処理等に隠れてしまい結果に表れないためと考えられる。

そこで、この効果を評価するために、1回の%readメッセージの転送により得られる効果を、転送を行なう場合と行なわない場合での実行時間の比較より求める。これを、ベンチマーク・プログラムで測定した%readメッセージの転送回数に適用することで、最適化による効果の見積りを算出した。

表 12: データ型別通信速度

データの参照が1回の通信で完了するもの			
データ種別	単一参照(μsec)	多重参照(μsec)	多重参照:アイドルなし(μsec)
ベクタ(幅)	12.6	13.2	13.3
ストリング	6.6	7.0	5.8
モジュール	6.4	6.4	4.2
データの送信個数分の繰り返しの通信が必要なもの			
データ種別	単一参照(μsec)	多重参照(μsec)	多重参照:アイドルなし(μsec)
整数	186.1	191.0	156.5
アトム	188.5	194.5	166.6
浮動小数点数	260.0	268.6	253.2
リスト	250.8	283.7	249.1
ベクタ(深さ)	250.0	283.1	235.7

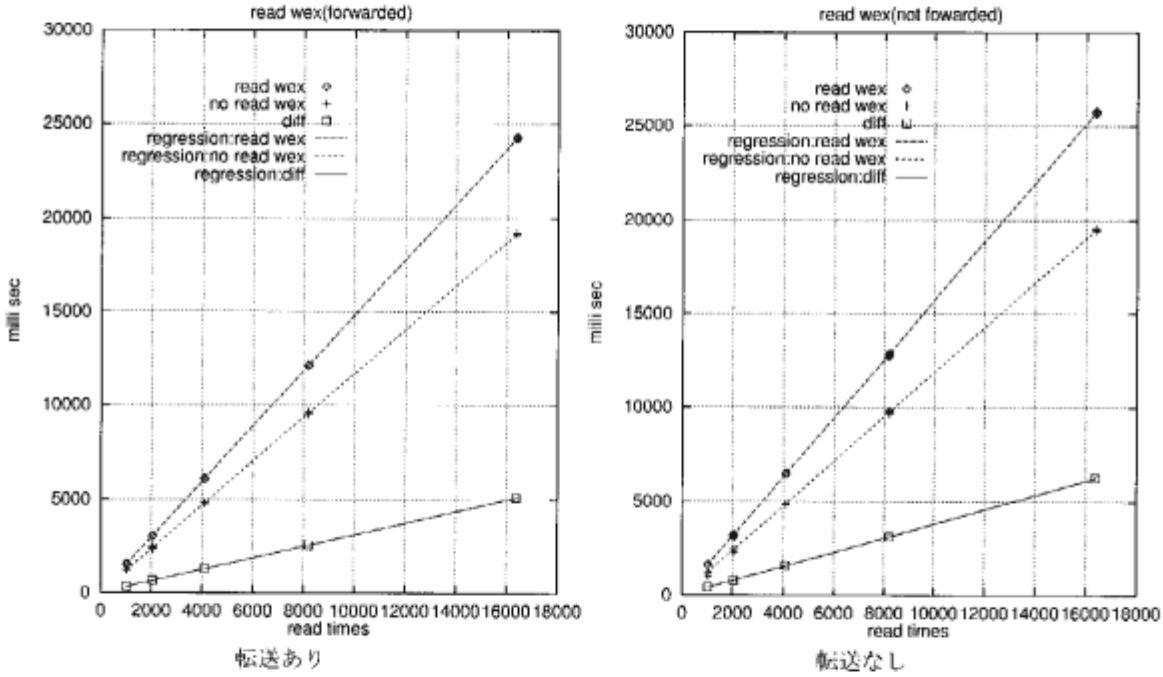


図 9: 自外部参照データの参照に要する時間

3.5.2 測定方法

外部参照セルに対する他クラスタからの参照のコストを求めるため、表13に示す各条件で測定を行なった。

表13の測定項目の 1 および 2 を満たすプログラム群を作成し、測定項目の 3 については実行時の条件とした。各プログラムにおいて、データの参照回数を変化させ、10回ずつ実行して、その実行時間を測定した。

3.5.3 測定結果

各条件でのすべての測定値 (実行時間) と実行時間の参照回数による回帰直線、および間接参照を行なう場合と行なわない場合の差とその回帰直線のグラフを、図9および図10 に示す。

各回帰直線式を表14に記す。表において、x はデータの参照回数、y は実行時間を示す。

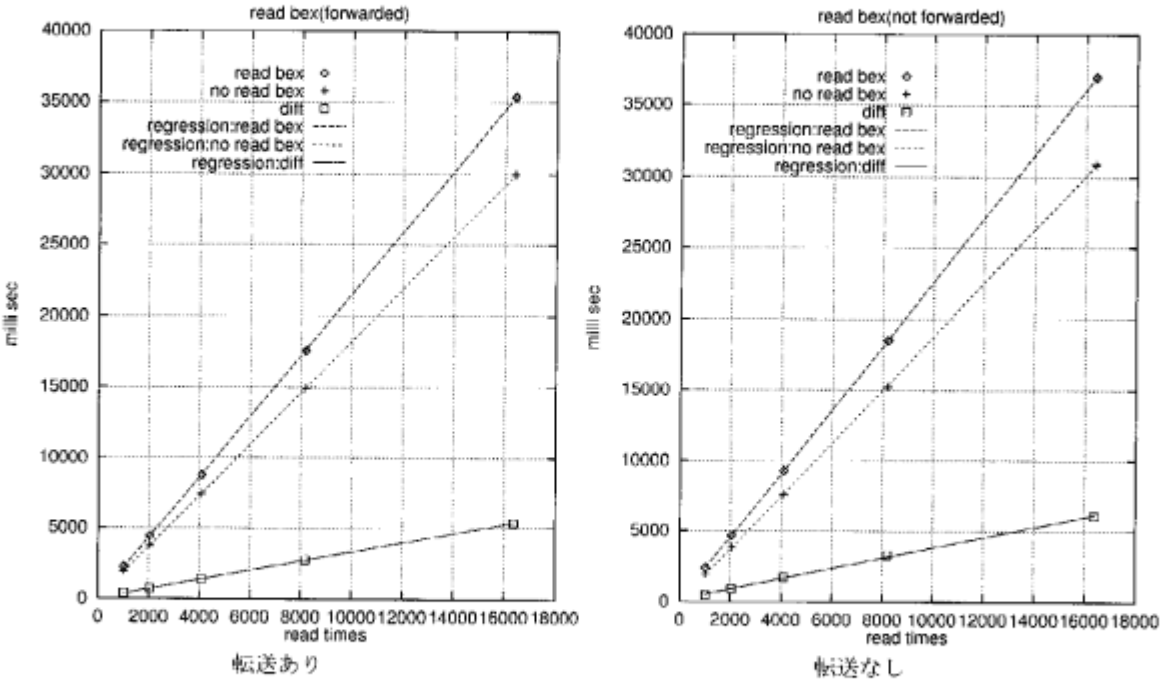


図 10: 黒外部参照データの参照に要する時間

表 14: 間接参照コスト算出のための回帰直線式

x : データの参照回数 y : 実行時間(msec)

白外部参照			図9
転送	参照	回帰直線式	グラフ記号
あり	あり	$y = 14.2708 + 1.4794x \pm 14.1503$	read wex
あり	なし	$y = 8.5458 + 1.1687x \pm 21.0431$	no read wex
差分		$y = 3.4477 + 0.3109x \pm 10.2479$	diff
なし	あり	$y = 18.9917 + 1.5687x \pm 28.2697$	read wex
なし	なし	$y = 10.0208 + 1.1879x \pm 43.4444$	no read wex
差分		$y = 8.9708 + 0.3808x \pm 8.6787$	diff
黒外部参照			図10
転送	参照	回帰直線式	グラフ記号
あり	あり	$y = -38.8292 + 2.1587x \pm 43.9054$	read bex
あり	なし	$y = -44.9083 + 1.8300x \pm 43.3697$	no read bex
差分		$y = 6.0792 + 0.3287x \pm 3.8083$	diff
なし	あり	$y = 33.7875 + 2.2549x \pm 15.9395$	read bex
なし	なし	$y = -99.0458 + 1.8875x \pm 83.6954$	no read bex
差分		$y = 132.8333 + 0.3674x \pm 107.1207$	diff

最適化の効果　表14において、間接参照を行なう場合と行なわない場合の差分に関する回帰直線式のxの係数を、1回の間接参照のコストと考えると、表16のようにデータ参照最適化の効果が算出できる。この結果を見ると、白外部参照の場合の方が黒外部参照の場合よりも効果が大きいことがわかる。

KL1プログラム別の効果　表15に、ベンチマーク・プログラムにおける%readメッセージの転送回数を測定し、先に求めた最適化の効果から実行時間の短縮率を概

表 15: KL1プログラム別の最適化の効果

プログラム	構成 (CL×PE)	転送回数 (白外部参照)	転送回数 (黒外部参照)	効果 (msec)	実行時間 (msec)	向上 (%)
mastermind	32×8	699.2	59.8	48.98	7707.40	0.6
pentomino	32×8	18826.6	0.0	1315.98	134350.20	1.0
mastermind	8×8	307.2	1.0	21.47	9886.80	0.2
pentomino	8×8	621.2	0.0	43.42	3361.20	1.3
bestpath	8×8	8114.0	0.0	567.17	2444.80	23.2

表 13: 最適化効果の測定条件

	項目	条件
1	外部参照セルの属性	白外部参照
		黒外部参照
2	間接参照の実施	参照あり
		パスのみ生成、参照なし
3	%read の転送	転送あり
		転送なし

表 16: データ参照最適化の効果の算出 (単位は μsec)

	転送なし	転送あり	差分 = 最適化の効果
白外部参照	380.8	310.9	69.9
黒外部参照	367.4	328.7	38.7

算した。実行時間は、10回実行した際の平均値を使用した。

これより、今回測定したプログラムでは、最大で20% 以上の効果が得られることがわかった。但し、ここで算出に用いた実行時間は、プログラム開始から終了までの実時間である。一方、%read の転送による最適化の効果は、並列に起こり得る事象についての値を積み上げたものである。このため、実際にはこれほどの性能向上は得られない。しかしながら、程度の差こそあれ %read メッセージの転送が起こるプログラムにおける利点は示していると考える。

4 荘園/里親処理の評価

4.1 評価項目

この報告では、荘園/里親処理に関する以下の項目について、評価を行なう。

- 荘園/里親における基本機能のコスト
  荘園/里親の生成/消滅、里親の切替え、荘園/里親関係のメッセージ通信などの処理時間をベンチマークプログラムの実行時間から算出する。また、荘園/里親の生成/消滅コストについては、新処理系と従来の処理系とで比較を行なう。

- 里親における基本機能の処理頻度
  里親の生成、切替え、里親ハッシュ表の排他制御、里親レコードの排他制御についての処理頻度を測定する。

- メッセージ発生数の資源配給量依存性
  %supply_resource メッセージや %throw_goal メッセージによって荘園/里親に与えられる資源配給量と、資源要求/補給のためのメッセージ発生数の関係を調べる。

4.2 荘園/里親における基本機能のコスト

4.2.1 評価方法

荘園/里親の生成/消滅コスト ここでは、荘園と里親の生成から消滅までの間に、荘園/里親処理に要した時間を、生成/消滅コストとする。測定誤差を小さくするため一連の処理を多数繰り返し、一回当たりのコストを算出した。実行時の構成は1クラスタ1PEである。プログラムは以下の3種類を用いる。

(a) 荘園/里親の平板的生成
荘園/里親のある階層下で、荘園および里親を 1組生成し、簡単なゴールを 1つ実行し、そのゴールが終了すると荘園/里親を消滅させる。この一連の処理を多数繰り返し実行する。

(b) ゴールの平板的生成
荘園/里親のある階層下で、簡単なゴールを 1つ実行する。このゴールが終了すると再び同一里親下

でゴールを実行する。この処理を多数繰り返し実行する。

(c) 荘園/里親の階層的生成
荘園/里親を生成し簡単なゴールを1つ実行する。この一連の処理を再帰的に多数繰り返して荘園/里親を階層的に生成する。指定回数の処理の終了後に、荘園/里親を最下層から順に消滅させる。

これらのプログラムの実行時間の差分から、荘園/里親の生成/消滅コストを算出する。

里親の切替えコスト　ある里親に属するゴールの実行が終了し、次に実行するゴールが異なる里親に属しているなら、その時点で里親の切替えが発生する。ここでは、この里親の切替えのみに要した時間を切替えコストとする。測定誤差を小さくするため一連の処理を多数繰り返し、一回当たりのコストを算出した。実行時の構成は1クラスタ1PEである。プログラムは以下の2種類を用いる。

(d) 異なる里親下のゴール間で通信を行なう
ある階層の里親下で荘園を2つ生成し、それぞれの荘園の下に里親が生成される。この2つの里親に属するゴール間でメッセージ通信を行なう。

(e) 同一里親下のゴール間で通信を行なう
ある階層の里親下で荘園を1つ生成し、その荘園下に里親を1つ生成する。この里親に属する2つのゴール間でメッセージ通信を行なう。

上記2つの項目における測定結果の差分から、里親の切替えコストを算出する。

クラスタ間負荷分散コスト　クラスタ間での負荷分散に要する時間をコストとして測定する。プログラムは以下の4つを用いる。

(f) クラスタ内でゴールを実行する
ゴールをクラスタ内に多数生成させる。生成方法は、荘園/里親の生成/消滅コストの項目(b)で述べた方法を用いる。実行時の構成は1クラスタ1PEである。

(g) クラスタ間でゴールを実行する
ゴールを多数生成し、他クラスタに負荷分散させる。ゴールの生成方法は、荘園/里親の生成/消滅コストの項目(b)で述べた方法とほぼ同じであるが、クラスタ間処理のために次の方法をとった。ゴールを1つ他クラスタに分散する。分散先のクラスタでは、里親が生成されそのゴールが実行され、ゴールの終了とともに里親も消滅する。ゴールの終了時に、分散元に終了合図のメッセージを送る[8]。そのメッセージが分散元に届いてから、再びゴールを1つ分散する。この一連の処理を多数繰り返す。実行時の構成は2クラスタ1PEである。

(h) パフォーマンスアナライザ(PA)機能を1つ用いてクラスタ間でゴールを実行する
PA機能を1つ用いて、上記(g)の方式を用いて測定する。ここで使うPA機能とは、KL1の組込述語としてプログラム中に埋め込み、実行時のクロック数を計測するものである。この機能を埋め込んだ場合、この機能を含まずに実行した時よりも全実行時間が増加すると考えられる。これを補正するために次の項目(i)を試み、結果を比較する。

(i) PA機能を2つ用いてクラスタ間でゴールを実行する
PA機能を1つ用いた時と、PA機能を2つ続けて用いた時の実行時間の差分を取り、PA機能を含まない測定結果を推定する。

4.2.2　測定結果

PIM/p実機上での測定結果を表17に示す。以下にこの表の見方を示す。

- 各プログラムは5回測定した。表中の数値は、その平均値である。
- ループ回数とは、測定誤差を小さくするため各プログラム中で一連の処理を繰り返した回数である。
- 実行時間とは、各プログラム中で一連の処理を繰り返した部分のみで費やされた時間である。
- 標準偏差は、実行時間の標準偏差である。
- 1ループ当たりの平均実行時間とは、実行時間をループ回数で割ったものである。
- 1ループ当たりのクロック数とは、1ループ当たりの平均実行時間を、クロック数に換算したものである。PIM/pは、1クロック当たり80[nsec]で動作している。

[8] ゴールを他クラスタで実行中、分散元のPEはアイドル状態となる。

表 17: 荘園/里親の基本機能測定プログラムの実行結果

項番	プログラム	ループ回数	実行時間 [msec]	実機の標準偏差	1ループ当たりの平均実行時間[μsec]	1ループ当たりのクロック数
(a)	荘園/里親の平板的生成	65536	55314	0.04	844	10550
(b)	ゴールの平板的実行	65536	815	0.21	12.4	155
(c)	荘園/里親の階層的生成	1024	225371	0.01	2.20×10^5	2750000
(d)	異なる里親下での里親切替え	1048576	61652	0.00	58.8	735
(e)	同一里親下での里親切替え	1048576	28438	5.60	27.1	339
(f)	クラスタ内ゴール実行	1048576	13009	3.43	12.4	155
(g)	クラスタ間ゴール実行	262144	144776	2107.77	552	6900
(h)	クラスタ間ゴール実行 (PAの機能 1 つ)	255	149	0.0	586	7325
(i)	クラスタ間ゴール実行 (PAの機能 2 つ)	255	152	0.0	596	7450

表 18: 現処理系と新処理系の荘園/里親生成速度の比較

プログラム	現処理系	新処理系	速度向上比
荘園/里親の平板的生成(1ループ当たり)	872[μsec]	830[μsec]	5.06%
荘園/里親の階層的生成(1ループ当たり)	260[msec]	219.2[msec]	18.6%

4.2.3 評価

荘園/里親の生成/消滅コスト 荘園/里親の生成/消滅の総コストは、荘園/里親を平板的生成した方法による実行時間(表 17の(a))と、ゴールを平板的生成した方法の実行時間(b)の差分で表せる。この結果は 10,400クロック (832[μsec]) となる。これは、PIM/p上の 315 appendリダクションに相当する[9]。

現処理系と新処理系で荘園/里親を平板的に生成した場合と、階層的に生成した場合についての結果を表 18に示す。表 18の速度向上比とは、現処理系の処理速度に対して新処理系の処理速度がどれだけ向上したかの割合である[10]。この結果、新処理系では、荘園/里親の生成に関しては5~18%の速度改善が認められた。

荘園/里親の階層的生成については、以下のように考えられる。荘園/里親を階層的に生成させた場合、`%request_resource`と`%supply_resource`の資源関係のメッセージが発生する。荘園/里親は、上階層の里親と 1RSU (Resource Supply Unit) という単位で資源をやりとりする。PIM/p の KL1処理系では、里親の手持ちの資源量1RSU 以上に保つという制約を設けている。i段目の荘園/里親に属しているゴールが、その下に荘園/里親を生成する場合、最上層まで再帰的に資源要求メッセージ(`%request_resource`)が送られ、その数はiである。生成された$i+1$段目の荘園/里親の下でゴールを実行するためには、再び最上層まで資源要求メッセージを送る必要があり、その数は$i+1$である。結局、i段目の荘園/里親に属しているゴールが、荘園/里親を生成し、その下でゴールを実行するために発生する資源要求メッセージは、合計で$2i+1$となる。n段の荘園/里親を生成した場合の資源要求メッセージの発生数は、

$$\sum_{i=0}^{n}(2i+1)=(n+1)^2$$

で表される[11]。$n=0$の階層から積算するのは、初期里親によるメッセージ発生を含めるためである[12]。ここで使ったように、単純に荘園/里親の生成を繰り返すプログラムの場合、nが大きくなると、これらのメッセー

[9] PIM/p上の 1appendリダクションは、28 命令、33 クロック、2.64[μsec]である。

[10] 表 17の荘園/里親の平板、階層的生成と比較して、表 18の結果は、若干高速化されている。これは測定時に使用した処理系の版数の違いによるものである。変更点は、メッセージ通信に関する若干の高速化であるが、現処理系と新処理系のどちらも同様にその高速化を施してあるので、改良効果の比較には特に問題ない。

[11] 資源要求に対する応答として、同数の資源補給メッセージ(`%supply_resource`)が、最上層から生成された荘園/里親まで送られる。

[12] 生成された荘園/里親の最上層には、初期荘園/里親が存在する。この初期荘園/里親から、その上階層に存在するルート里親への資源要求メッセージが送出される数も含んで考えている

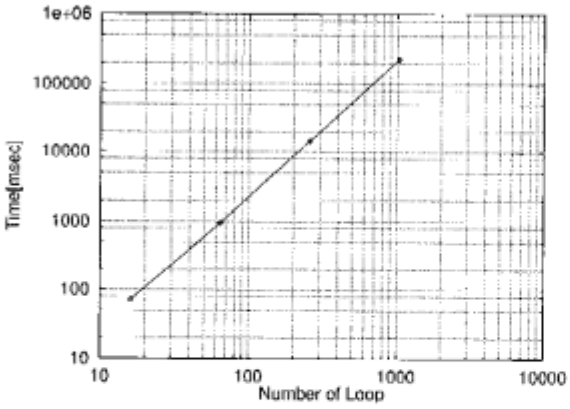


図 11: 荘園/里親を階層的に生成した時の実行時間

ジ処理時間が、荘園/里親の生成/消滅に要する時間よりも大きくなり、実行時間 $\propto n^2$ という比例関係になると考えられる。測定結果で、ループ回数を横軸に、実行時間を縦軸にとると、図 11に示すような比例関係が見られ、この議論が裏付けられた。

上式を用いると、1,024回ループを回した場合、約1,050,000回のメッセージが発生することになる。対象となる資源メッセージは、要求と補給の2種類あるため、資源メッセージの総発生数は約2,100,000回である。表17(c)から、実行時間が225[msec]であるから、1メッセージの処理に1,340クロック (107 [μsec]) を要していると考えられる[13]。

さらに、この処理時間の内訳を考える。1回のメッセージ送受信処理には、里親の切替が3回ある[14]。1回の切替え時間は次項で述べるように、198クロック (15.9[μsec]) となっている。このため、1回のメッセージ送受信処理中の里親の切替えコストは、594クロック (47.5[μsec]) となる。また、命令コードの静的解析から、里親ハッシュ表と里親レコードの資源更新処理に36クロック (2.88[μsec]) 必要になっている。これらの時間を除くと、正味の1メッセージ送受信に要するコストが算出され、710クロック (56.8[μsec]) となる。

里親の切替コスト　異なる里親下でメッセージの通信を行なった結果から、同一里親下でメッセージの通信を行なった結果を引けば、里親の切替えコストが算出される。表 17の(d)と(e)から、その差は396クロック (31.7[μsec]) である。ここで使ったプログラムでは、1回のループで2度里親の切替えが行なわれるので、1回当たりの里親の切替えコストは、198 クロック (15.9[μsec]) となる。これは、PIM/p上の 6 appendリダクションに相当する。

表 19: 荘園/里親に関する基本機能処理時間

基本機能	クロック数	時間 [μsec]
荘園/里親の生成/消滅	10395	832
荘園/里親の 1メッセージの通信	710	56.8
里親の切替え	198	15.9
クラスタ間負荷分散	7045	564

クラスタ間負荷分散コストの測定　クラスタ間でゴールを実行した結果からクラスタ内でゴールを実行した結果を引けば、クラスタ間の通信を行なうために要したコストが算出される。表 17の(f)と(g)からその差は、6,745クロック (540[μsec]) である。

一方、PA機能を使用して計測されるクロック数には、PA機能そのものの実行時間、結果を格納するリスト処理、および結果の出力が含まれている。PA機能1つ当たりの実行時間は、PA機能 2つを導入した結果から、PA機能1つを導入した結果を引けば算出される。その差は、表 17の(h)と(i)から 125クロック (10.0[μsec]) である。従って、PA機能 1つの結果からこの 125クロックとクラスタ内でのゴール実行時間を引けば、クラスタ間負荷分散に要したコストは、7,045クロック (564[μsec]) と見積もられる。この結果は、表17の(f)と(g)から推定した結果とそれほど変わらない。

ここでクラスタ間負荷分散とは、ゴールを負荷分散するための %throw_goal メッセージ処理と、分散先クラスタでの里親の生成/消滅処理、分散元に終了メッセージを送る処理などを含んでいる。

これらの結果のまとめを表 19に示す。

4.3　里親における基本機能の処理頻度

4.3.1　評価方法

処理系にプローブを導入して、里親の各機能の処理頻度データを収集した。プローブの処理は単純に、測定したいルーチン中に埋め込み、そのルーチンの実行回数をカウントするものである。ここで測定対象とした機能を以下に示す。

- 里親の生成頻度
- 里親の切替え頻度

[13] 要求と補給に関するメッセージの処理時間が等しいと仮定できない場合、一組の要求と補給メッセージ処理に、2,680クロック要したと考えれば良い。

[14] 初期里親からメッセージを出そうとしている里親に変わり、メッセージを出したらリスペンドするため初期里親に変わり、最後にメッセージを受けとった里親に変わるため。

- 里親ハッシュ表の排他制御の頻度
- 里親レコードの排他制御の頻度

ベンチマークプログラムは PIMOS 上で life, LPIA, MGTP を実行した[15]。このうち、lifeについては、PIMOSなしでも測定した。これは、PIMOS 使用時に必要な処理、つまり資源更新やロックのための処理が増加することを測定するためである。実行時の PIM/p の構成は 32クラスタ 8PEである。各プログラムは 5回測定して、評価には平均値を用いた。

4.3.2 lifeの測定結果

PIMOSなしの場合 プログラムlifeを実行した時の結果を図 12 ～ 16に示す。図はそれぞれ、里親の生成回数 (図 12), 里親の切替え回数 (図 13), メッセージ受信時に里親の存在しなかった回数 (図 14), 里親ハッシュ表の排他制御回数 (図 15), 里親レコードの排他制御回数 (図 16)についてクラスタごとの分布を表している。図 15 の記号の意味は、FPTableLock は里親ハッシュ表の排他制御のため、ロック機構のルーチンに入った回数、FPTableLockWait1 は一回目のロックに失敗した回数、FPTableLockWait2 は二回目のロックに失敗した回数、FPTableLockWait3 は三回目以降のロックに失敗した回数である。図 16 についても同様である。なおロック機構 (図17) に関しては、以下の考察の中で説明する。

以下に測定結果について考察する。

図 12から、里親はクラスタ(以下 CL)CL0~20で1つずつ生成されていることが分かる。これはlifeのプログラム中の負荷分散の指示によるものである。

図 15で、里親ハッシュ表の排他制御回数が CL16~19 では、他のクラスタの 20 倍にもなる。この結果は次の理由によるものと考えられる。CL16~19 では、プログラムの指示によってゴールが実行されるが、これらのゴールは、変数同士のユニファイを行なってすぐに終了し、里親も消滅する。ところが、CL0~15 で未定義変数の具体化が行なわれると、これらのクラスタに %unify メッセージが送られ、ユニファイの処理が行なわれる。%unify メッセージの受信処理では、ユニファイゴールを生成する場合と、生成しない場合がある。前者の場合には、ユニファイゴールの里親が必要となり、里親が不在の場合には新たに生成する。後者は、ユニファイ処理のための compare & swap 操作による書き換えが成功した場合に限られる。図 12より、CL16~19 では里親は最初の 1つしか生成されないことが分かる。また、図 14からメッセージ到着時に里親不在となる場合が多いことからみて、CL16~19 に送られた %unify メッセージの受信処理では、ユニファイゴールを生成していないことが推測される。このとき、%unify に付随してきた WTC の値は、加算するべき里親がすでに消滅しているので %return_WTC メッセージによって庄園に戻される。図 15は、この時の里親ハッシュ表へのアクセスがカウントされたことを表している。

[15] これらのベンチマークプログラムについては、表1参照。

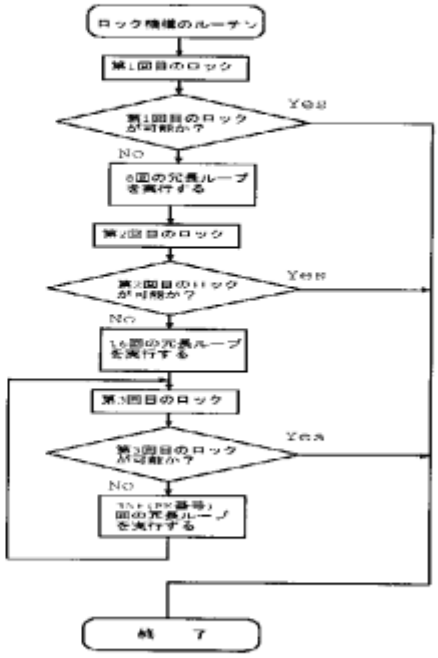


図 17: 排他制御のロック機構の処理手順

表 20: 里親ハッシュ表の排他制御回数 (life)

測定項目	呼び出し回数	全呼び出しに対する割合
全ロック機構呼び出し	8355.6	
第一回目のロック不可能	1054.6	12.6%
第二回目のロック不可能	819.6	9.80%
第三回目以降のロック不可能	1090.0	13.0%

里親ハッシュ表、里親レコードの排他制御の方法の概略を図 17に示す。里親ハッシュ表の排他制御の実行頻度の集計を表 20に示す。この表から分かることは、第二回目でロック可能となるのは僅か 3% である。第三回目のロックの無限ループの平均繰り返し回数が 1.33 回であることから、この無限ループの繰り返しを無駄にしないために、第二回目のロックの無限ループの平均

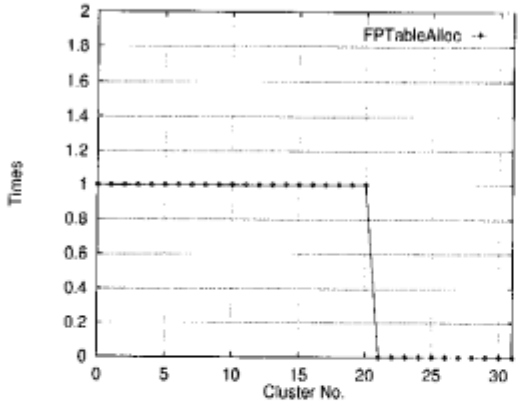


図 12: 里親の生成回数 (life)

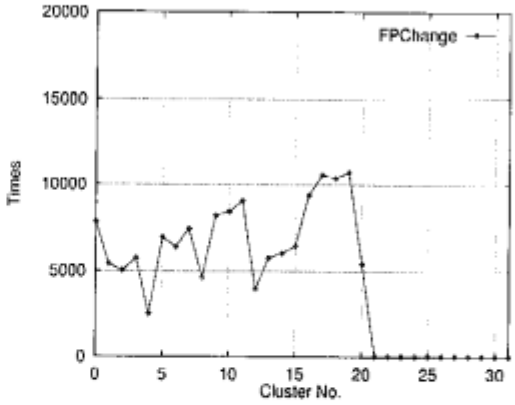


図 13: 里親の切替え回数 (life)

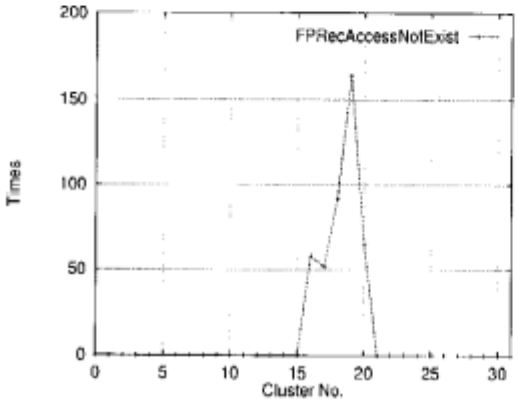


図 14: メッセージ受信時の里親不在回数 (life)

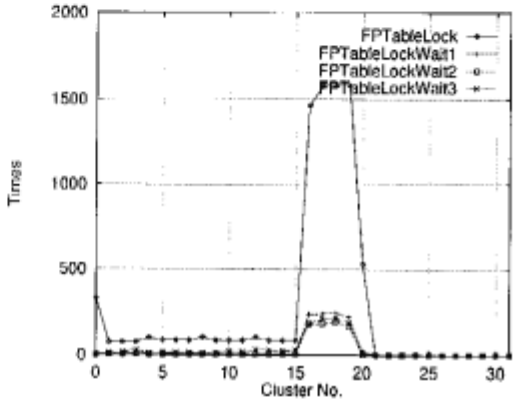


図 15: 里親ハッシュ表の排他制御回数 (life)

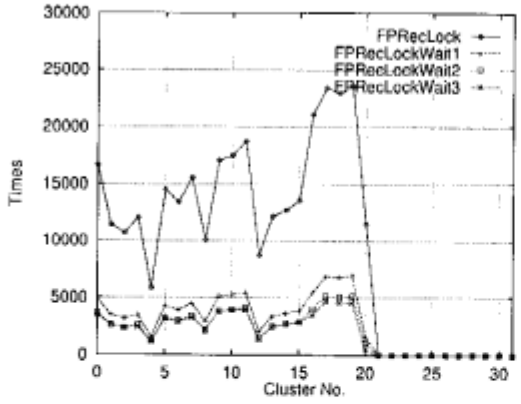


図 16: 里親レコードの排他制御回数 (life)

繰り返し回数を 1(もしくは整数)以下に抑える必要がある。その対策として、第二回目でロック不可能だった時の後に実行される 16 回の冗長ループ回数を増やせば良いと推察される。

里親ハッシュ表の排他制御は表 20 から、380,000 リダクションに対して、全クラスタで、約 8,400回であった。約 50リダクションに一回の間隔で里親ハッシュ表の排他制御が行なわれたことになる。しかし、図 14から分かるように、クラスタによるばらつきが大きい。

里親切替えの全クラスタでの合計回数は、146,000回であった。里親の切替えコストは表 19から 198クロック (15.9[μsec]) であるから、全体として見れば、里親の

表 21: 里親レコードの排他制御回数 (life)

測定項目	呼び出し回数	全呼び出しに対する割合
全ロック機構呼び出し	313172.2	
第一回目のロック不可能	88346.2	28.2%
第二回目のロック不可能	65877.4	21.0%
第三回目以降のロック不可能	62242.6	19.9%

切替えに必要だったコストは2.32 [sec] である。256PEで実行した場合、 life の実行時間が 500[msec] であるため、全PEの実行時間を合計すると 128[sec] である。実行時間に対する里親の切替えに要した時間の割合は約3% と見積もられる。

里親レコードの排他制御に関する集計を表 21に示す。里親レコードは里親ハッシュ表の排他制御に対して幾程度ロックウェイトが発生している。これは里親レコードのアクセスが多発しているからと推察される。

PIMOS上で実行した場合 PIMOS上で life を実行した時の結果を図 18 ～ 21 に示す。図はそれぞれ、里親の切替え回数、メッセージ受信時に里親が存在しなかった回数、里親ハッシュ表の排他制御回数、里親レコードの排他制御回数についてクラスタごとの分布を表している。

PIMOS 上で実行した life と PIMOS なしで実行した life と比較して、里親へのアクセス回数全般[16]が、約 2割程度増加している。この原因は次のように考えられる。PIMOSのルーチンは、その上で走らせるアプリケーション (この場合 life) とは、別の里親に所属している。このため lifeプログラムと実行制御のためのPIMOSのルーチンが交互に走ることによって、里親の切替えが増え、全般に里親へのアクセス回数が増える。また PIMOS では、ユーザ管理、ジョブ管理、資源管理などのために、CL0 上にネストした荘園/里親が生成され、その下でユーザ・プログラムが実行される。そのため、資源関係のメッセージが最上層の荘園/里親まで送られるような場合には、CL0 上での里親に対するアクセスが増加すると考えられる。図 14と図 19で比較すると、メッセージを受信したクラスタに、里親がすでに存在しなかった数はクラスタによって若干異なっているが、全体としてはほとんど変わらなかった。

里親ハッシュ表、里親レコードの排他制御に関する集計を、それぞれ表 22, 23に示す。PIMOS なしで実行したものとほぼ同じ結果である。

[16] 里親の切替え回数、メッセージ受信時に里親が存在しなかった回数、里親ハッシュ表の排他制御回数、里親レコードの排他制御回数

表 22: 里親ハッシュ表の排他制御回数 (life on PIMOS)

測定項目	呼び出し回数	全呼び出しに対する割合
全ロック機構呼び出し	10085.4	
1回でロック不可能	1240.8	12.3%
2回でもロック不可能	928.0	9.20%
3回以降でロック不可能	807.2	8.00%

表 23: 里親レコードの排他制御回数 (life on PIMOS)

測定項目	呼び出し回数	全呼び出しに対する割合
全ロック機構呼び出し	415941.4	
1回でロック不可能	118996.2	28.6%
2回でもロック不可能	88809.0	21.4%
3回以降でロック不可能	84233.2	20.3%

4.3.3 LPIAの測定結果

PIMOS 上で LPIA を実行した時の結果を図 22 ～ 25 に示す。図はそれぞれ、里親の生成回数、里親の切替え回数、里親ハッシュ表の排他制御回数、里親レコードの排他制御回数についてクラスタごとの分布を表している。

図 22から里親は各クラスタ均等に生成されていることが分かる。

里親ハッシュ表、里親レコードの排他制御に関する集計を、それぞれ表 24, 25 に示す。

里親の切り替えは全クラスタで合計 1,480,000回発生している。里親の切替えコストは表 19から198 クロック (15.9[μsec]) であるから、全体として見れば、里親の切替えに必要だったコストは 2.93×10^8 クロック (23.5[sec]) である。256PE で実行した場合、 LPIA の実行時間が 56.2[sec] であるため、全PEの実行時間を合計すると 4[hour] である。実行時間に対する里親の切替えに要した時間の割合は約 0.16% となる。

表 24: 里親ハッシュ表の排他制御回数 (LPIA)

測定項目	呼び出し回数	全呼び出しに対する割合
全ロック機構呼び出し	1803.4	
1回でロック不可能	0.0	0.0%
2回でもロック不可能	0.0	0.0%
3回以降でロック不可能	0.0	0.0%

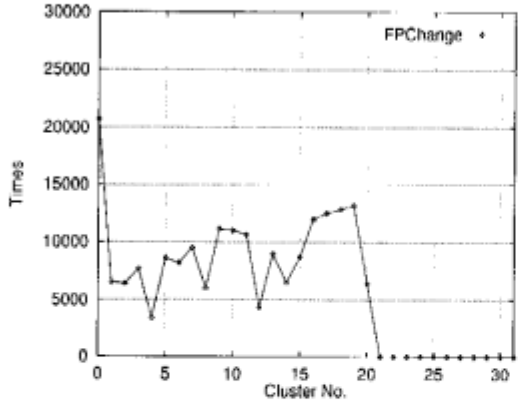


図 18: 里親の切替え回数
(life on PIMOS)

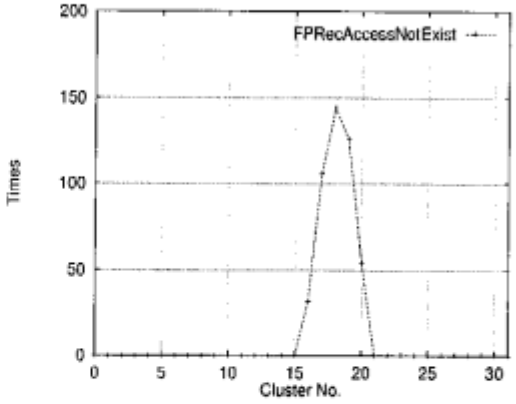


図 19: メッセージ受信時の里親不在回数
(life on PIMOS)

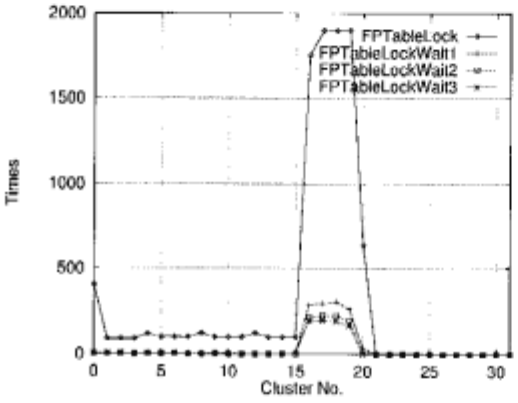


図 20: 里親ハッシュ表の排他制御回数
(life on PIMOS)

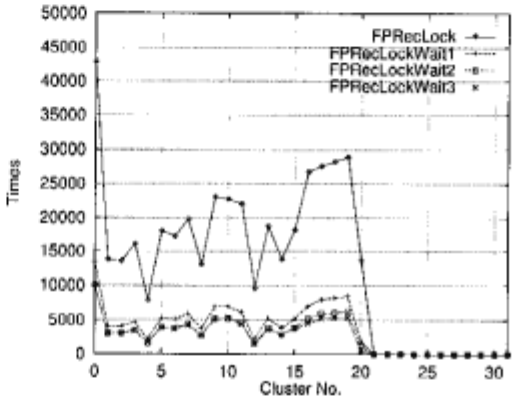


図 21: 里親レコードの排他制御回数
(life on PIMOS)

表 25: 里親レコードの排他制御回数 (LPIA)

測定項目	呼び出し回数	全呼び出しに対する割合
全ロック機構呼び出し	2960631.2	
1回でロック不可能	963587.2	32.5%
2回でもロック不可能	720626.8	24.3%
3回以降でロック不可能	856102.6	28.9%

表 24, 25から、里親ハッシュ表ではロックウェイトが発生せずに里親レコードの方でロックウェイトが多発している。これは里親のハッシュリンクが短く、里親ハッシュ表へのアクセスの衝突がないためと推察される。

4.3.4 MGTPの測定結果

PIMOS 上で MGTP を実行した時の結果を図 26 ～ 29に示す。図はそれぞれ、里親の生成回数、里親の切替え回数、里親ハッシュ表の排他制御回数、里親レコードの排他制御回数をクラスタごとの分布を表している。

里親は各クラスタ均等に生成されている。また、里親の切替え等の処理も各クラスタでほぼ均等である。

里親の切り替えは全クラスタで合計 180,000,000 回発生している。里親の切替えコストは表 19から 198クロック(15.9[μsec]) であるから、全体として見れば、里親の切替えに必要だったコストは 3.56×10^{10} クロック (47.6[min]) である。256PE で実行した場合、 MGTP の実行時間が 12.9[min] であるため、全PEの実行時間を合計すると 54.9[hour] である。実行時間に対する里親の切替えに要した時間の割合は約 1.5% となる。MGTP では実行時にグラフィック表示を行なうために

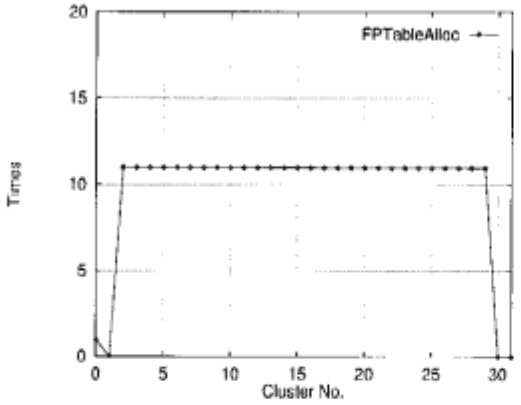


図 22: 里親の生成回数 (LPIA)

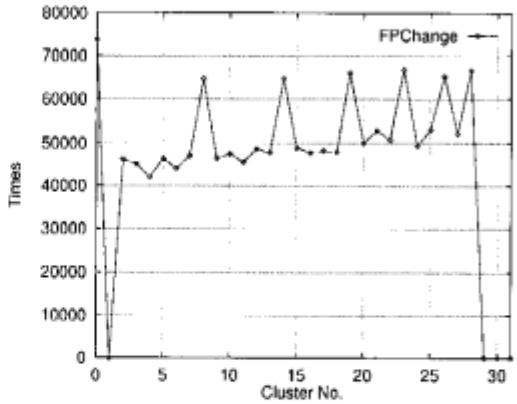


図 23: 里親の切替え回数 (LPIA)

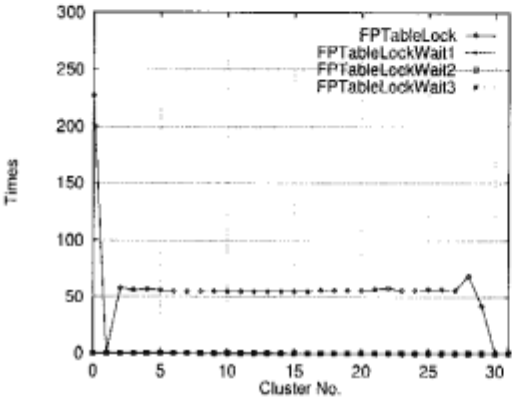


図 24: 里親ハッシュ表の排他制御回数 (LPIA)

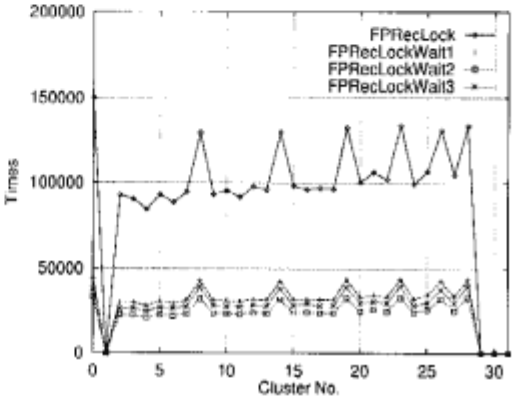


図 25: 里親レコードの排他制御回数 (LPIA)

表 26: 里親ハッシュ表の排他制御回数 (MGTP)

測定項目	呼び出し回数	全呼び出しに対する割合
全ロック機構呼び出し	266243.8	
1回でロック不可能	655.2	0.246%
2回でもロック不可能	456.8	0.172%
3回以降でロック不可能	211.4	0.0794%

表 27: 里親レコードの排他制御回数 (MGTP)

測定項目	呼び出し回数	全呼び出しに対する割合
全ロック機構呼び出し	360069469.2	
1回でロック不可能	89229219.0	24.8%
2回でもロック不可能	62638430.2	17.4%
3回以降でロック不可能	50355047.2	14.0%

データの転送等にかかる時間が多くなる。データ転送の時にPIMOS の処理およびアイドルしている状態が増加するため、里親の切替え処理が多くなる。

里親ハッシュ表、里親レコードの排他制御に関する集計を、それぞれ表 26, 27に示す。LPIA や MGTP などの比較的大きなアプリケーションプログラムでは、里親ハッシュ表への排他制御が競合することはほとんどないが、里親レコードへの排他制御は競合しやすくなっている。対策を検討すべきである。

4.4 メッセージ発生数の資源配給量依存性

4.4.1 評価方法

各クラスタにゴールを負荷分散した時、そのゴールはクラスタ毎の里親によって管理される。里親のゴール管理の一つに資源という概念がある。里親では、ゴールの生成と実行の際に、資源を 1ずつ消費する。資源がなくなれば、上の階層の荘園/里親に資源関係のメッセージを送出し、資源を得る。その資源のやりとりの単位を RSU (Resource Supply Unit)という。この RSU を当

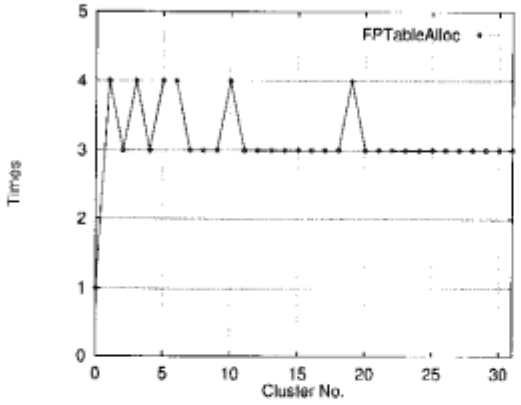


図 26: 里親の生成回数 (MGTP)

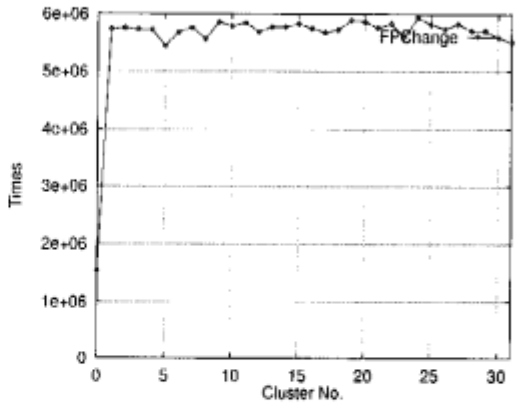


図 27: 里親の切替え回数 (MGTP)

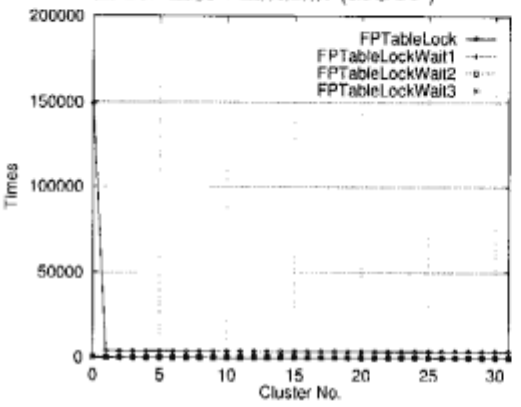


図 28: 里親ハッシュ表の排他制御回数 (MGTP)

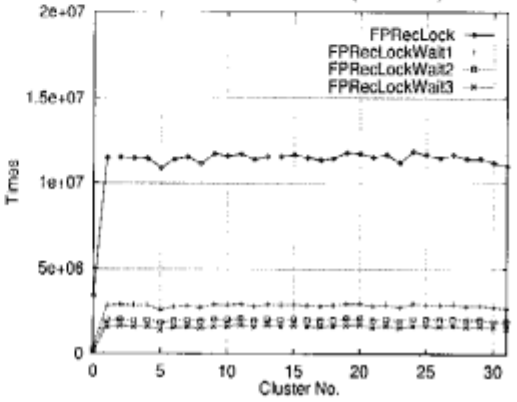


図 29: 里親レコードの排他制御回数 (MGTP)

初 0x1000000 としていた。今回、この RSU の値を変化させて、資源関係のメッセージ発生回数の変化について調べた。

また、ゴールを他のクラスタに負荷分散した場合、%throw_goal メッセージに資源を付加している。この単位を RTU (Resource Throwgoal Unit) という。その値は 0x10000 である。RTU に関しても、荘園/里親の資源関係のメッセージ発生頻度との関係について調べた。評価用プログラムとして、PIMOS 上で LPIA, MGTP のプログラムを用いた。

4.4.2 荘園/里親の資源関係のメッセージ発生数の RSU依存性測定結果

LPIA, MGTP における %request_resource, %supply_resource メッセージ発生頻度の RSU 依存性の結果を、それぞれ図30, 31 に示す。

図 30 から LPIA では、RSU の値 0x100000 から 0x1000000 までを見ると、ほぼ RSU の -1 乗に比例してメッセージ数が減少していることが分かる。また、RSU を 0x1000000 以上にした場合は、この比例傾向が成り立っていないが、これは、RSU がある値以上になると資源が不足しなくなるために、資源関係のメッセージの発生が少なくなるためと考えられる。このことから、LPIA では RSU のデフォルトの値である 0x1000000 で十分と言うことが出来る。

図 31 から MGTP では、RSUを変化させてみた全域に関して資源関係のメッセージ生成数は、ほぼ RSU の -1 乗で減少していることが分かる。RSUの値を今回の測定範囲よりももっと大きくすれば、LPIA と同様にメッセージ発生数が飽和してくると考えられる。

このように、RSU の値と発生するメッセージ数との間に -1 乗の比例関係が存在するのは、例えば、RSU の値を 1/10 にすれば、同じ分量の資源を確保するために 10倍の数の資源要求メッセージが必要なためと理解できる。どの程度の資源量が各里親で必要になるかは、プログラムによって異なるが、ある程度大きな単位で補給した方が処理効率が良くなると考えられる。

4.4.3 資源関係のメッセージ発生数のRTU依存性測定結果

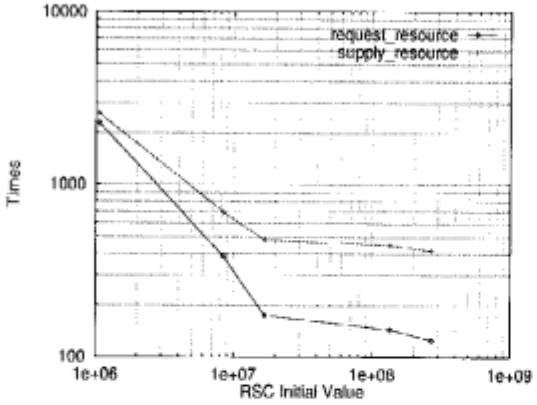


図 30: 資源メッセージの RSU 依存性 (LPIA)

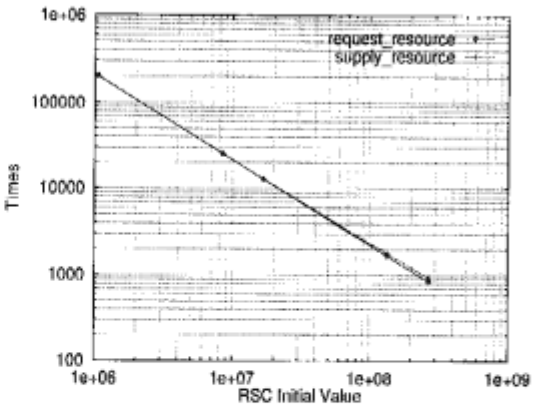


図 31: 資源メッセージの RSU 依存性 (MGTP)

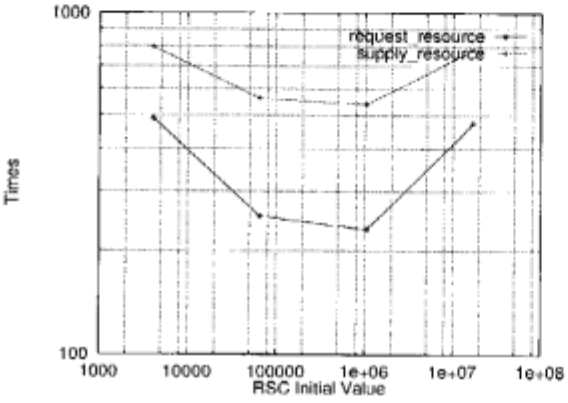


図 32: 資源メッセージの RTU 依存性(LPIA)

LPIA を実行した時の %request_resource、%supply_resource メッセージ発生頻度の RTU 依存性の結果を、図 32に示す[17] [18]。

LPIA では、RTU による資源要求メッセージ生成数が、RTU の値 0x100000で最小値となっている。これは、ゴールを負荷分散する時に里親から資源をもらって分散先の里親に資源を与えるが、この値があまり大きくなると分散先の里親の資源は増えるが、分散元の里親の資源が減るため、資源要求のメッセージ数が増加してしまうためと推察される。このことから、現状の RTU のデフォルト値は、LPIAに関しては適切であると言えるが、他のアプリケーションプログラムに関しても評価を行なう必要がある。

5 まとめ

本報告では、並列推論マシン PIM/p 上の KL1処理系に実装されているクラスタ間通信方式および荘園/里親処理に関しての評価結果を示した。

クラスタ間通信方式に関する評価では、以下のようなことを明らかにした。

- 白輸出と黒輸出では、全般に白輸出の方が多く、特に%readメッセージの返信先を送信するための白輸出が頻発する。
- メッセージの発生頻度としては、データ参照系のメッセージが全体の9割以上を占めている。また、メッセージの処理時間は、全実行時間の 2~4割を占める。
- KL1のデータ型別の通信速度では、ストリングやモジュールなどのデータ構造が高速である。
- 新処理系では旧処理系に比べて %answer_valueや %releaseメッセージの数を減らすことが出来た。また、%readメッセージの転送によって、プログラムによって最大で20%程度実行時間を短縮する効果があった。

また、荘園/里親処理に関する評価では、以下のようなことを明らかにした。

- 荘園/里親の生成/消滅処理はトータルで、PIM/p 上の 315 appendリダクション相当の時間がかかる。また、里親の切替えには、6 appendリダクション相当の時間がかかる。
- 里親の切替え処理のプログラム実行時間に占める割合は、lifeでは3%と高く、LPIA では低く0.16%程度である。また、里親ハッシュ表への排他制御では、LPIA や MGTP などある程度大きなアプリケーションにおいては競合がほとんど

[17] RSU の値はデフォルトの値 0x1000000 に設定している。
[18] MGTP に関しては測定を行なっていない。

ない。しかし、里親レコード自体へのアクセスでは、排他制御の競合が起きている。

- 荘園 / 里親の資源関係のメッセージ発生数は、プログラムの消費する資源量の範囲内で、`%supply_resource`による補給単位 (RSU) に逆比例する。一方、`%throw_goal`による補給単位 (RTU) に対しては、プログラムによって最適な値が存在している。

これらの知見は、今後並列論理型言語の処理系を設計する場合や、アプリケーションプログラムを開発する際に、有益なものと考える。

謝辞

本研究の機会を頂いたICOT第1研究部 近山隆部長に感謝します。また、有意義な御討論をして頂いたICOT の PIM 評価タスクグループの委員の方々に感謝します。

参考文献

[1] 瀧 和男編: bit別冊 第五世代コンピュータの並列処理, 共立出版 (1993).

[2] ICOT 第 1 研究室編: VPIM 処理方式解説書, TM-1044, ICOT (1991).

[3] Hirata, K. *et al.*: Parallel and Distributed Implementation of Concurrent Logic Programming Language KL1, In *Proc. of International Conference on the Fifth Generation Computer Systems*, pp.436-459 (1992).

[4] 服部, 篠木, 久門, 後藤: 並列型推論マシンPIM/pのアーキテクチャ, 情報処理学会論文誌, Vol.30, No.12, pp.1584-1592 (1989).

[5] Kumon, K. *et al.*: Architecture and Implementation of PIM/p, In *Proc. of International Conference on the Fifth Generation Computer Systems*, pp.414-424 (1992).

[6] 畑澤 宏善: 共有メモリ並列マシン上の細粒度自動負荷分散方式の評価, 情報処理学会論文誌, No. 35(10). pp. 2069-2077 (1994).

[7] ICOT: 平成 4 年度電算基礎技術開発成果報告書 ハードウェア技術編 (1993).

[8] ICOT: 平成5年度発電設備診断システムの開発調査研究開発報告書 融合化設計・試作編 (I 融合型基本ソフトウェア技術) (1994).

[9] Tick, E.: *Parallel Logic Programming*, M.I.T. Press, Cambridge MA (1991).

[10] Furuichi, M., Taki, K. and Ichiyoshi, N. : A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Programs on the Multi-PSI, In *Proc. 2nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 50-59 (1990).

[11] Wada, K. and Ichiyoshi, N. : A Study of Mapping of Locally Message Exchanging Algorithms on a Loosely-Coupled Multiprocessor, TR-587, ICOT (1989).