プログラミング 8－3
（1996. 8. 28）

並列論理型言語 KL1 分散処理系の外部参照管理方式の評価

六沢 一昭　　市吉 伸行
rokusawa@okilab.oki.co.jp　ichiyoshi@mri.co.jp
沖電気工業 (株)　　(株) 三菱総合研究所

外部参照の管理は並列言語処理系の実装における重要な課題のひとつである．実際的な応用プログラムを実際の並列計算機で実行させることによって KL1 分散処理系の外部参照管理方式を評価した．評価結果は，本方式のオーバヘッドが小さいことを示している．使用された輸出表エントリの数やプロセッサ間に渡るゴミループについても考察した．

Evaluation of Remote Reference Management in a Distributed KL1 Implementation

Kazuaki Rokusawa　　Nobuyuki Ichiyoshi
rokusawa@okilab.oki.co.jp　ichiyoshi@mri.co.jp
Oki Electric Industry　　Mitsubishi Research Institute

Remote reference management is one of the main issues in parallel language implementations. This paper evaluates remote reference management in a distributed KL1 implementation. Measurements have been taken on a real parallel machine using experimental or genuine application programs. The evaluation result indicates low overhead of the management scheme. The number of export table entries and interprocessor garbage reference loops are also discussed.

1 はじめに

本稿では, 並列論理型言語 KL1 [1] の分散処理系における外部参照管理方式の評価を述べる. 外部参照とは他のプロセッサ (以下 PE と略す) への参照であり, その管理は並列言語処理系実装の基本のひとつである. 本稿では, まず, 処理系の設計方針を述べ, 処理方式を簡単に説明する. そして評価プログラムを説明し, 評価を述べる.

2 設計方針

KL1 で記述する処理は一般に**動的かつ不均質**であり, PE によって振舞いは異なる. また, PE 内処理に比べて PE 間メッセージ通信には長い時間がかかる. これらに対処するため「各 PE が独立に GC を行なう」「大域 GC は行なわない (で済ませたい)」「メッセージの送信数を抑える」という方針で分散処理系を設計した.

3 分散処理方式の概要

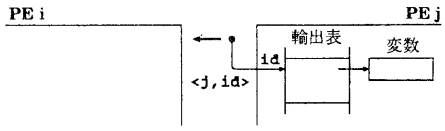
3.1 ゴールの分散実行

KL1 プログラムの実行の単位は**ゴール**という細粒度プロセスである. 粒度はスレッドよりも小さく, ゴール間の実行の切替えにかかるコストは非常に小さい. ゴール群はメモリ空間を共有し, 共有変数によって通信を行なう. PE は一般に多くの実行可能ゴールを抱えており, あるゴールの実行が終了もしくは中断すると別のゴールの実行へ移る. 各 PE は独立にゴールを実行し, 投げ出し指定をされたゴールに出会うとメッセージ (`%throw`) を組み立てて他の PE へ送る.

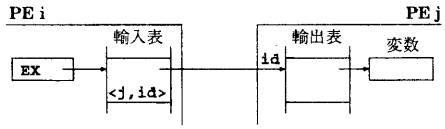
3.2 外部参照ポインタの管理

生成 変数を引数に持つメッセージを他の PE に送ると「他の PE を指すポインタ」が生まれる. これが**外部参照ポインタ**である (図 1 参照).

表現形式 各 PE はアドレス変換テーブル (**輸出表**と**輸入表**) 持ち, 各外部参照ポインタはこれらを介してデータセルを指す. 外部参照ポインタは PE 番号と輸出表エントリ番号の対 (図 1 で

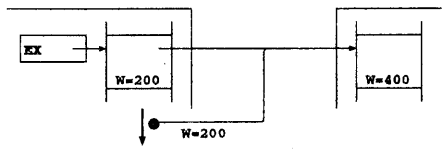


(1) 変数への参照を輸出表に登録して送出する.

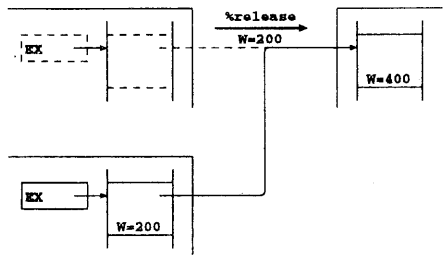


(2) 外部参照ポインタが PEi に生まれた.

図 1: 外部参照ポインタの生成



(1) 重みを分割し外部参照ポインタに付けて送る.

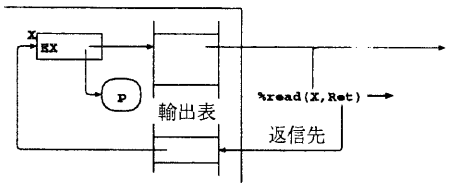


(2) 解放した外部参照ポインタの重みを返却する.

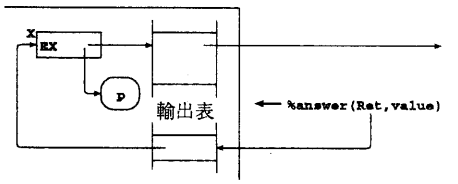
図 2: 重みを用いた分割 GC 方式

は `j` と `id`) で表現し, これらは輸入表エントリが記憶する. 輸出表の存在により各 PE は**独立に GC** を行なうことができる.

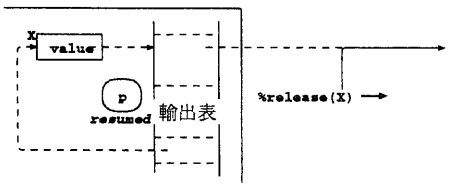
コピーと解放 重み付け参照カウント [2, 3] による分散 GC を採用した. 各外部参照ポインタ (メッセージ中のものも含む) が重みを持ち, 「ある輸出表エントリの重み」と「そのエントリを指すポインタの重みの合計」が等しくなるように制御する. ポインタのコピーを他の PE に渡す時は重みを分割し, 解放した時は`%release` で重みを返却する (図 2). `%release` を受信すると, 返却された重みをエントリから引く. その結果ゼロになったならば, そのエントリを解放する.



(1) 返信先を設け %read を送信する.



(2) %answer で参照値が返信された.



(3) 参照値を書き込み %release を送信する.

図 3: 参照値の読み出し

間接輸出 重みが分割できない外部参照ポインタを他の PE に渡す時は, そのポインタ自身を輸出表に登録し, 新しい外部参照ポインタを生成する. これを**間接輸出**と呼ぶ.

3.3 参照値の読み出し

ゴール p の実行において, ある外部参照ポインタ X の指す値が必要になると, 参照値の読み出し処理を行なう (図 3).

まず %read(X,Ret) を送信する. Ret は参照値の返信先であり X を指す外部参照ポインタである. p は実行を中断して X にフックする.

%read(X,Ret) を受信した PE は, X の**参照先が具体化されていれば** %answer を直ちに返信して参照値を送る. **参照先が変数**の場合は, 返信先を記録した制御構造を生成し変数にフックする. 参照値の返信は具体化がなされた時点で行なう. **参照先が外部参照ポインタ** X' の場合は, X' が指す PE へ %read(X',Ret) を転送する. 返信先はそのままであり, 最初に %read を送信したプロセッサへ参照値が直接送られる.

%answer(Ret,value) を受信すると, Ret が指す外部参照ポインタを具体値 value で書き換え, フックしていたゴールは実行を再開し, 外部参照ポインタは解放する.

3.4 ユニフィケーション

これは従来型言語の書き込みに相当する処理である. 外部参照ポインタ X とデータ Y のユニフィケーションは通常 %unify(X,Y) の送信によって行なう. %unify(X,Y) を受信した PE は, X の参照先と Y のユニフィケーションを行なう.

4 性能向上のための工夫

4.1 %release 送信の削減

MRB [5] により, あるデータセルへの参照が単一であるかどうかを検出することができる. 単一参照の外部参照ポインタ X に対して %read, %unify, %answer を送信する時は, それらの送信時に X を解放し重みを返却してしまう. %release は送らない. この結果 %release の送信を減らすことができる.

4.2 新しい外部参照ポインタ生成の抑制

外部参照ポインタ生成の度に輸出表エントリを割付けたり, 受信の度に輸入表エントリを割付けたりすると, 無駄な「読み出し」「書き込み」処理を招いてしまう. 参照先が同じでも外部参照ポインタが異なると参照元では異なったものに見えてしまうからである. この状況を防ぐため, 輸出表, 輸入表のそれぞれで, ハッシュ機構を用いて同じデータセル, 同じポインタに対しては同じエントリを使う処理を行なっている.

5 評価プログラムと実行結果

実行した評価プログラムを以下に示す.

MGTP 並列モデル生成法による定理証明系. 準群に関する未解決問題を解いた.

Gene ダイナミックプログラミングを並列に実行しタンパク質の配列解析を行なう.

表 1: 評価プログラム

プログラム		実行時間 (sec)	リダクション数	メッセージ数 総数	%read	%unify	%throw	3つの比
MGTP (Ground)	32 PEs	1,582	5,003 M	106 M	37,498 K	51 K	51 K	700 : 1 : 1
	256 PEs	273	5,003 M	108 M	38,171 K	52 K	52 K	700 : 1 : 1
MGTP (NonGr)	32 PEs	1,751	6,486 M	14.6 M	4,170 K	554 K	285	15K : 2K : 1
	256 PEs	413	7,035 M	109 M	37,384 K	647 K	750	50K : 1K : 1
Gene	32 PEs	5,457	12,962 M	4,390 K	473 K	9.0 K	8.7 K	50 : 1 : 1
	256 PEs	793	12,963 M	4,504 K	486 K	9.2 K	8.7 K	50 : 1 : 1
DisPool	32 PEs	54	26 M	5,452 K	2,405 K	32 K	1.0 K	2K : 30 : 1
	256 PEs	249	231 M	53.0 M	22,899 K	1,972 K	7.8 K	3K : 300 : 1
Pento	32 PEs	665	1,974 M	890 K	137 K	21 K	14 K	10 : 2 : 1
	256 PEs	134	1,974 M	1,902 K	479 K	69 K	57 K	10 : 1 : 1
BestPath	32 PEs	207	4,436 K	3,330 K	706 K	499 K	162 K	4 : 3 : 1
Router	64 PEs	104	32 M	1,561 K	634 K	237 K	701	700 : 100 : 1
LogSim	256 PEs	356	983 M	47.0 M	19,118 K	8,219 K	2.6 K	7K : 3K : 1
COMP	21 PEs	282	43 M	96.1 K	14 K	6.8 K	151	100 : 50 : 1

DisPool 分散プールのベンチマーク.

Pento 6×10 の詰め込みパズルの全解探索. 多階層動的負荷分散方式を用いている.

BestPath 最短経路問題を解く. 優先度機構を利用して見込み計算を減らしている.

Router LSI の配線を行なう. 並列オブジェクトモデルに基づいている.

LogSim VLSI の論理シミュレーションを行なう. タイムワープ機構を採用している.

COMP AYA 言語コンパイラの並列コンパイル. コンパイラは 21 のファイルから成る.

上記の多くは実際的な応用プログラムである [6]. 分散メモリ型並列計算機 PIM/m [4] による実行結果を表 1 に示す. `%read`, `%unify`, `%throw` は, 代表的な PE 間処理 (「参照値の読み出し」「ユニフィケーション」「ゴール投げ出し」) を行なうものである. これらの比から評価プログラムの特徴が多岐にわたっていることがわかる.

6 評価

本方式であるが故のオーバヘッド (`%release`, 間接輸出, 輸出表) と, 本方式では対処できない「PE 間に渡るゴミループ」を考察する.

6.1 %release 送受信によるオーバヘッド

`%release` は外部参照の管理のためだけのメッセージである. 従ってこれの送受信コストはそのまま本管理方式のオーバヘッドになる. `%release` 送信について以下を算出し, 表 2 に示した.

割合 全メッセージに対する割合.

頻度 各 PE における平均送信間隔.

削減率 「MRB による削減 (4.1 参照)」の効果[1].

`%release` 送受信コストは他のメッセージに比べて低いもしくは等しい. このため, **割合**はメッセージ送受信処理全体における `%release` 送受信コストの割合の上限を示す. 表から, MGTP が 30% 前後である他はすべてほぼ 10% 以下であることがわかる. このことから, `%release` 送受信によるオーバヘッドは他のメッセージ送受信によるコストに対して相対的に小さいと言える.

頻度は, `%release` 送受信処理のプログラム実行処理全体への影響を示す. 送信間隔は, 最も短い MGTP (Ground) 32 PEs でも 1.7 msec である. これは, 機械命令を 26K ステップも実行できる[2] 長い時間であり, `%release` 送受信の全体への影響は充分小さいと言える.

削減処理を行なわないと, `%read`, `%answer`, `%unify` の合計と同じ数の `%release` が送信されてしまう[3]. 表を見ると, ほとんどの場合 **削減率**は 75% を越えており, MRB による削減は充分効果があったと言える.

[1] 算出式は表の脚注を参照のこと.

[2] PIM/m の要素 PE のクロックは 65 nsec である.

[3] 外部参照ポインタに対してメッセージが送信されないまま解放され `%release` が送信される可能性もあるので, この**削減率**は実際に削減できた率の下限を示す.

表 2: %release の送信数と割合, 頻度, 削減率

プログラム		送信数	割合[a] (%)	頻度[b] (msec)	(red)	削減率[c] (%)
MGTP (Ground)	32 PEs	29,589 K	28	1.7	1.0 K	61
	256 PEs	30,108 K	28	2.3	1.4 K	61
MGTP (NonGr)	32 PEs	4,092 K	28	13.7	8.2 K	54
	256 PEs	36,316 K	33	2.9	1.7 K	49
Gene	32 PEs	47.5 K	1.1	3,680	2.2 M	95
	256 PEs	57.8 K	1.3	3,510	2.1 M	94
DisPool	32 PEs	592 K	11	2.9	1.7 K	88
	256 PEs	5,072 K	9.6	11.7	7.0 K	89
Pento	32 PEs	67.4 K	7.6	320	192 K	77
	256 PEs	214 K	11	160	96 K	79
BestPath	32 PEs	345 K	10	19.2	11.5 K	81
Router	64 PEs	38.4 K	2.5	173	102 K	97
LogSim	256 PEs	159 K	0.3	573	342 K	100
COMP	21 PEs	8.46 K	8.9	700	420 K	75

[a] %release の送信数 ÷ メッセージ総数
[b] (実行時間 ÷ %release の送信数) × PE 台数
リダクション数 (red) は, PIM/m の 要素 PE 性能 = 600 append KLIPS で算出.
[c] 1−(%release の送信数 ÷ %read, %answer, %unify の送信数の合計) =
1−(実際の送信数 ÷ 削減処理を行なわなかった場合の送信数)

6.2 間接輸出の影響

間接輸出を行なうと「外部参照ポインタが再び外部参照ポインタを指す」という状況が生まれる. これを**間接参照**と呼ぶ[4]. 間接参照の存在自体は問題ではないが, これにメッセージが飛び込むと**転送**という余計な処理が起きてしまう. ここでは %read と %unify の転送の影響を考察する.

%read の転送 以下を算出し, 表 3 に示した.

転送数 %read と %answer の差[5].

割合 全メッセージに対する転送 %read の割合.

頻度 各 PE における %read の平均転送間隔.

割合を見ると最高でも数% である. 多くはゼロに近い. このことから, %read 転送によるオーバヘッドは他のメッセージ送受信によるコストに対して相対的に非常に小さいと言える. **頻度**も, 最も頻繁なものでも 20 msec を越えており %read の転送頻度は充分低いと言える.

以上のことから「%read 転送によるオーバヘッドは充分小さい」と結論づけることができる.

%unify の転送 %read と比べて送信数がかなり少ない (表 1 参照) ので, %unify 転送によるオーバヘッドは充分小さいことが期待できる.

6.3 必要とされた輸出表エントリ数

どのくらいの輸出表エントリが必要であったかは実用的な処理系の設計において興味深い点である. ここでは輸出表エントリ数を考察する.

%read, %answer, %unify, %release は外部参照ポインタに対して送信されるものである. ひとつのポインタに対して複数送信されることがある[6] ので, 送信数の合計はポインタの総数の上限を示す. また, ひとつの輸出表エントリは 1 つ以上の外部参照ポインタから指される. 従って, 上述したメッセージの送信数の合計は使用されたエントリの延べ数の上限を示す.

表 3 に, 外部参照ポインタの総数 (の上限) を示す. 前述したように, 表の値は実際に使われたエントリ数に対して充分大きいのであるが, この程度の数で済んでいる. 評価プログラムの実行で

[4] ユニフィケーションによっても間接参照は生まれるので, これは本管理方式特有のものではない.

[5] 非決定的プログラムの場合, (%read の返答である) %answer が返信されなくてもプログラムが正常終了することがある. このため「%read と %answer の差」は, 厳密には, 「%read 転送数の上限」である.

[6] 例えば, ある外部参照ポインタに対して%read と %release が送信される.

表 3: 転送された %read と外部参照ポインタの総数

プログラム		転送数[a]	割合[b] (%)	頻度[c] (msec)	(red)	外部参照ポインタの 総数[d]	総数/PE 台数
MGTP (Ground)	32 PEs	0	0	—	—	104,636 K	3,269 K
	256 PEs	0	0	—	—	106,505 K	416 K
MGTP (NonGr)	32 PEs	85.3 K	0.6	659	395 K	12,901 K	403 K
	256 PEs	4,511 K	4.1	23.4	14 K	107,220 K	419 K
Gene	32 PEs	2.97 K	0.1	58,800	35 M	1,000 K	31.2 K
	256 PEs	8.33 K	0.2	24,400	15 M	1,025 K	4.00 K
DisPool	32 PEs	0	0	—	—	5,434 K	170 K
	256 PEs	0	0	—	—	52,838 K	206 K
Pento	32 PEs	0	0	—	—	362 K	11.3 K
	256 PEs	0	0	—	—	1,241 K	4.85 K
BestPath	32 PEs	110 K	3.3	60.2	36.1 K	2,145 K	67.0 K
Router	64 PEs	1.67 K	0.1	3,990	2.4 M	1,541 K	24.1 K
LogSim	256 PEs	5.74 K	0.0	15,900	9.5 M	46,608 K	182 K
COMP	21 PEs	1,562	1.6	3,790	2.3 M	42.2 K	2.01 K

[a] %read 送信数 − %answer 送信数
[b] %read の転送数 ÷ メッセージ総数
[c] (実行時間 ÷ %read の転送数) × PE 台数.
リダクション数 (red) は, PIM/m の 要素 PE 性能 = 600 append KLIPS で算出.
[d] %read, %answer, %unify, %release の送信数の合計.

は 768 K エントリ分の領域を確保し再利用も行なっているが, 一例を除いて再利用を行なわなくてもエントリは足りていたことがわかる.

6.4 PE 間に渡るゴミループ

本管理方式では PE 間に渡るゴミループを解放できない. しかし, 本稿で取り上げている評価プログラムの実行においてゴミループが問題になることはなかった. 輸出表エントリが不足したり, 回収できないゴミによってメモリが埋め尽くされ GC が頻発するようなことは起こらなかった. これは評価に用いたプログラムの実行においてだけではなく, KL1 分散処理系上での数年間に及ぶプログラム開発においても同様である. これらのことから「ゴミループの問題は深刻ではない」と言うことができよう.

7 おわりに

KL1 分散処理系の外部参照管理方式のオーバヘッドを考察した. メッセージ処理全体のコストと比べてオーバヘッドは小さく, 必要な輸出表エントリも適度な数で済んだ. またゴミループも問題にはならなかった. 本稿で示した測定値は様々な設計において役立つものと信じる.

参考文献

[1] K. Ueda and T. Chikayama, "Design of the Kernel Language for the Parallel Inference Machine," *The Computer Journal*, Vol.33, No.6, pp.494–500, 1990.

[2] D. I. Bevan, "Distributed Garbage Collection Using Reference Counting," *Parallel Computing*, Vol.9, No.2, pp.179–192, 1989.

[3] N. Ichiyoshi, *et al.*, "A New External Reference Management and Distributed Unification for KL1," *New Generation Computing*, Ohmsha Ltd., pp.159–177, 1990.

[4] 瀧 和男 編, 第五世代コンピュータの並列処理, bit 別冊, 共立出版, 1993.

[5] T. Chikayama and Y. Kimura, "Multiple Reference Management in Flat GHC," *ICLP'87*, pp.276–293, 1987.

[6] 瀧 和男, 市吉 伸行, マルチ PSI における並列処理とその評価 — 小粒度高並列オブジェクトモデルに基づくパラダイムについて —, 電子情報通信学会論文誌, Vol.J75-D-I, No.8, pp.723–739, 1992.