メッセージの追い越しがある分散環境における低コストな外部参照管理

六沢 一昭　　仲瀬 明彦　　川合 英夫　　今井 明
沖電気工業(株)総合システム研究所　(株)東芝総合研究所　(財)新世代コンピュータ技術開発機構

1 はじめに

本稿では、並列論理型言語の分散処理系における低コストな外部参照管理について述べる。

分散環境における外部参照管理の手法として重み付き参照カウントを用いた方式[1]が知られているが、重み付け管理のコストが大きい。単一参照であるデータを輸出する際は単純に外部参照アドレスだけを用いるべきであろう。ところが外部参照を輸入した側で多重参照になることがあり、そのような外部参照に対して読み出し／書き込み(unify)処理を行なう場合は、メッセージ(%read/%unifyメッセージ)を送信する時に外部参照ポインタを解放することができない。その結果、後に外部参照ポインタを解放した時に送信する%releaseメッセージに%readあるいは%unifyが追い越されてしまい、処理が正しく行なわれない恐れがある。

本稿では、1ビット情報を用いることによって輸入後の多重参照に対処する方式について述べる。

2 計算モデル

以下に示す計算モデルを仮定する。

- 局所メモリを持つ有限個のプロセッサ(PE)がネットワークで結合されている。
- 非同期なメッセージ通信によってPE間処理を行なう。メッセージの追い越しの起こる可能性があり、送信した順序でメッセージが到着するとは限らない。
- PEは自分の局所メモリへの参照ポインタを他のPEへ輸出できる。このため、他PEのデータを参照するポインタ("外部参照ポインタ")を各PEは持ちうる(図1)。
- PEごとの局所GCを可能にするため、各PEは他のPEからの参照を一括管理する表("輸出表")を持つ。
- 外部参照ポインタをもつPEは、%readあるいは%unifyを送信することにより、ポインタが指すデータの読み出し及び書き込み(unify)を行なう。%readは引数として"返信先"を持ち(図3)、%readを受信したPEは、その返信先へ%answerメッセージを送信して参照値を返す。

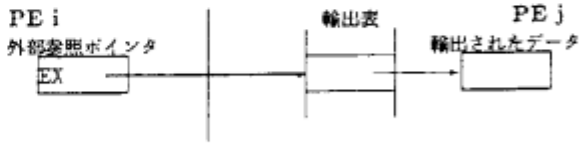


図 1: 外部参照

3 読み出し / 書き込み (unify) 処理の問題点

3.1 単一参照ならば問題ない

外部参照ポインタへの参照がひとつならば、%readあるいは%unifyを送信する時にポインタを解放することができる(図2)。そのような%readあるいは%unifyを受信すると対応する輸出表エントリを解放する。

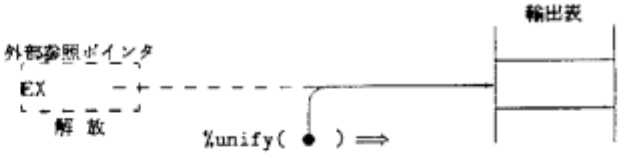


図 2: 単一参照時の書き込み(unify)

3.2 多重参照になると ...

外部参照ポインタへの参照が複数存在する場合は、%readあるいは%unifyの送信時にポインタを解放することができない。メッセージ送信後もPE内に外部参照ポインタへの参照が存在するからである(図3)。

外部参照ポインタは以下の場合に解放し、%releaseを送信する。

- 局所GCによって外部参照ポインタへの参照のないことがわかった時。
- %answerを受信し、参照値を得た時。

メッセージの追い越しのある環境ではこの%releaseが%readあるいは%unifyを追い越してしまう可能性があ

A Low Cost External Reference Management for Distributed Processing Systems with Non-FIFO Communication
Kazuaki ROKUSAWA (OKI), Akihiko NAKASE (TOSHIBA), Hideo KAWAI, Akira IMAI (ICOT)

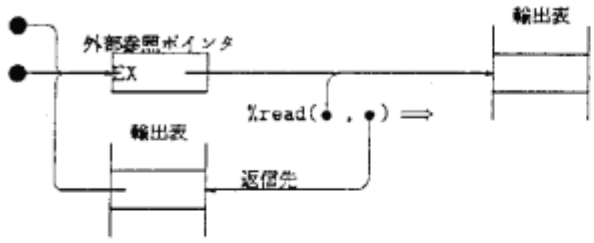


図 3: 多重参照時の読み出し

る。このため「単純に %read 及び %unify を送信する。%release を受信したら対応する輸出表エントリを解放する」のでは、「%read あるいは %unify が到着した時、対応する輸出表エントリが解放されてしまっている」ことが起こりそうである。

%read は送信時に外部参照ポインタを指す返信先を用意する (図 3) ので %read 到着前に局所 GC が起こっても外部参照ポインタが解放されることはない。一方 %unify 送信後はいつでも局所 GC あるいは %answer 受信が起こり外部参照ポインタが解放される可能性がある。以上のことから解決すべき課題は

> 『%unify が到着する前に輸出表エントリが解放されてしまうのを防ぐ』

ことになる。

4 解決方法

4.1 望まれること

並列論理型言語で書かれたプログラムでは、外部参照ポインタが指すデータに対して書き込み (unify) を 2 回以上行なう (%unify を 2 回以上送信する) ことは稀である。このため、多重参照になった場合、最初の %unify は単一参照の時と同じコストで送信できることが望ましい (2 回目からはコストが増えてもかまわない)。

4.2 *Ack* を用いる方式ではコストが高くなる

%unify の到着を確認してから %release を送信するのであれば、%unify 到着前の輸出表エントリの解放を防ぐことができる。しかしこの方式はすべての %unify に対して到着確認メッセージ (*Ack*) が必要になるのに加え、以下に示すコスト増を招く。

- *Ack* 受信前に送信しようとしたメッセージを *Ack* 受信まで待たせておく仕組みが必要。
- *Ack* 受信前に %answer を受信しても外部参照ポインタを解放しない仕組みが必要。

4.3 1 ビット情報を用いる方式

%unify の到着を確認することなく %release を送信できるのならば、%unify 送信は単一参照の時と同じコストで行なうことができる。これは %unify と %release をカウントするカウンタを外部参照ポインタと輸出表エントリに設けることによって実現可能になる。到着順序に依存せず %unify と %release の両方を受信して初めて輸出表エントリが解放されるからである。

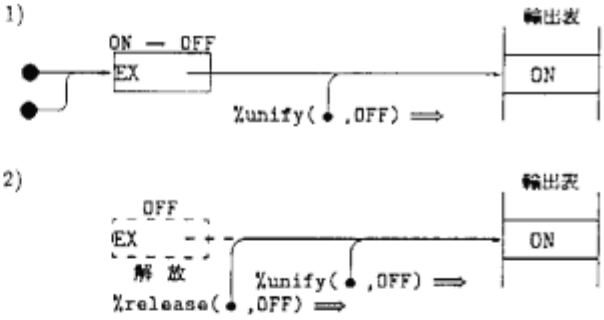
カウントが必要なのは最初の %unify 送信だけである。またカウント値ゼロは解放されている状態であることからとりうる値は 1 と 2 しかない。従ってカウンタは必要なくビットでよい。ビット ON/OFF がカウント値 2/1 に対応する。多重参照時の処理を以下に示す[1]。

読み出し： ビット OFF の %read を送信する[2]。外部参照ポインタは操作しない。

書き込み： ビットが ON ならば OFF にセットし、ビット OFF の %unify を送信する (図 4)。コストは単一参照の時と同じである。ビットが OFF ならば *Ack* を用いる方式を行なう。この状態になって初めてコストが高くなる。

解放： 外部参照ポインタのその時のビットを %release で送る。

この方式による書き込み (unify) 処理の例を図 4 に示す。%unify 送信のコストは単一参照の時と同じである。また %release が %unify を追い越して先に到着しても輸出表エントリが解放されてしまうことはない。



%unify と %release を受信すると輸出表エントリは解放される。

図 4: 1 ビット情報を用いる方式

5 おわりに

メッセージの追い越しがある分散環境における低コストな外部参照管理について述べた。本方式は 1 ビット情報を用いることによって最初の %unify 送信を単一参照の時と同じコストで処理することができた。

参考文献

[1] N. Ichiyoshi et al. "A New External Reference Management and Distributed Unification for KL1," New Generation Computing, Vol.7 Nos.2,3, 1990.

[1] 単一参照の時はいつもビット ON のメッセージを送る。

[2] %read のビット OFF はカウント値ゼロに対応する