

共有メモリマルチプロセッサにおける効率的な KL1 ストリームマージ処理方式

2E-10

今井 明

稲村 雄

後藤 厚宏

新世代コンピュータ技術開発機構 (ICOT)

はじめに

我々は、並列論理型言語 KL1 を高速に実行する並列推論マシン PIM[1] を開発中である。KL1 では、プロセス間をストリームと呼ぶメッセージ通信の経路で接続することで並行プログラムを記述してゆくため、複数のストリームをマージ(併合)する処理が頻繁に行われる。PIM 上のオペレーティングシステムである PIMOS も KL1 で書かれており、このようなプログラミング手法が随所に用いられているため、ストリームのマージ処理の効率がシステム全体の性能に大きく影響を与える。

本稿では、PIM のクラスタのような共有メモリマルチプロセッサにおいて、このストリームのマージ処理を効率的に行う方式について述べる。

1 マージ処理のためのデータ構造

KL1 では 2 本のストリームのマージを行うプロセスを

```
merge([], In2, Out) :- true | Out=In2.
merge(In1, [], Out) :- true | Out=In1.
merge([X|In1], In2, Out) :- true |
    Out=[X|NewOut], merge(In1, In2, NewOut).
merge(In1, [X|In2], Out) :- true |
    Out=[X|NewOut], merge(In1, In2, NewOut).
```

のように記述できるが、この場合、

- 入力ストリームに 1 要素が入力される度に、merge/3 述語の中断 / 再開という重い処理が行われる。
- 入力ストリームの数が増える毎に、マージ処理のコストも増える

という問題がある。

そこで、任意の入力ストリーム本数に対して入力ストリーム本数に依存しないで、コンスタントオーダで実現する方式が提案されており [2]、我々は、この方式を非共有メモリマルチプロセッサであるマルチ PSI に実装することで、その有効性を確認している [3]。この方式では、図 1 のような構造でマージプロセスを表現する。ここで、マージに対する入力ストリームが MHV(マージフック変数)であり、これが指すマージレコード MR は、出力ストリームへのポイントと、入力ストリーム数(RC)を保持する。

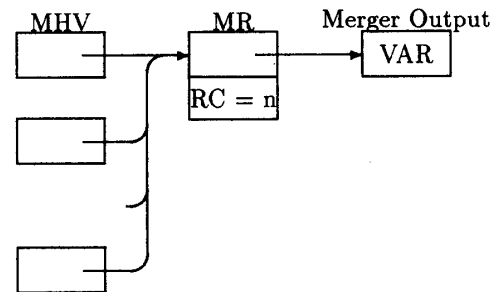


図 1: n 入力マージプロセスを表現するための構造

この処理を、共有メモリマルチプロセッサで、複数のプロセッサが MHV セルや MR セルに同時にアクセスできるような状況でも適用できるように、排他制御を施した処理方式を設計した。これは、図 1 における構造自体は特に変更せず、次章に述べるアルゴリズムの変更のみによって対応している。

2 マージ処理の排他制御方式

2.1 設計方針

共有メモリ結合マルチプロセッサでは、同時に複数の入力ストリームに対して入力される可能性だけでなく、1 本の入力ストリームに対して同時に 2 つ以上のプロセッサが入力を行う可能性もある。このため排他制御が必要となるが、例えば「入力操作中は MR セルをロックし続ける」というような排他制御方式は、並列性を著しく損なってしまふ。そこで、並列性をできる限り損なわないような排他制御方式を設計した。なお、この排他制御方式はデッドロックの防止のため、全ての排他制御処理を Compare & Swap (CS) や、Fetch & Add (FA) という「リード・モディファイ・ライト」を不可分操作とする基本操作で実現した。

2.2 ストリームマージ処理の詳細

前述の構造を用いたマージ処理の詳細について述べる。実際のマージ処理は、MHV セルと各種タイプとのユニフィケーションによって行われる¹。

リストとのユニフィケーション

入力ストリームに 1 メッセージが入力されたことを意味し、入力されたリストセルを出力につなぐ処理を行

¹MHV への参照が唯一であることが保証されている場合は、処理手順がかなり単純化できるが、ここでは省略する。

```

procedure UnifyMHVandList (*MHV, *LIST)
begin
    NewL := リストセルを割り付る
    NewL→CAR := LIST→CAR
    NewMHV := 同じ MR を指す新しい MHV セルを割り付ける
    if (CS (ptr = MHV, old = 以前読んだ MHV の値, new = LIST)) ..... (1)
    then
        NewOut := 新しい出力先変数を割り付ける
        NewL→CDR := NewOut
        repeat
            until (CS (ptr = MR, old = Output, new = NewOut)) ..... (2)
            Unify (Output, NewL) ..... (3)
            Unify (LIST→CDR, NewMHV) ..... (4)
        else
            NewMHV と NewL を回収し, Unify(MHV, LIST) (5)
        endif
    end

```

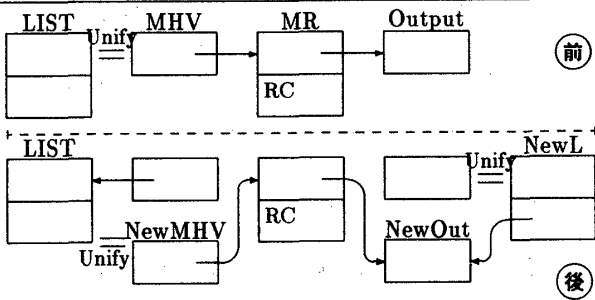


図 2: MHV とリストとのユニフィケーション

ら。具体的な処理手順を、図 2 に示す。

ベクタとのユニフィケーション

入力ストリーム数の増加(ゼロ要素ベクタの場合は減少)を意味する。入力ベクタの各要素が追加された入力ストリームとなる。

MHV セルを CS でベクタに具体化し、MR セル中の RC を FA で(ベクタサイズ-1) だけ増加させる。(ベクタサイズ) 個の MR セルを指す MHV セルを新たに割り付け、それぞれベクタの要素とユニファイする。

NIL(□) とのユニフィケーション

入力ストリームの一本が閉じられたことを意味する。

MHV セルを CS で NIL に具体化し、MR セル中の RC を FA で 1 だけ減じる。減じた結果が 0 になった場合は、全ての入力ストリームが閉じられてマージプロセスが終了したことを意味するので、MR セルを回収し、出力を NIL とユニファイする。

3 排他制御のコスト

マージャに対する 1 入力処理で、共有メモリ結合に対応するために追加した処理のコストについて考察する。

図 2 のアルゴリズムにおいて、異なるプロセッサが複数のストリームから同時にリストを入力した場合を想定する。この場合でも、直列化すべき「際どい区間」は、出力ストリームへリストを繋ぐ操作を行う間だけであり、これを (2) の CS といった単純な操作で実現している。CS に失敗した場合であっても、その処理は、読み直したポインタの値を元に CS を再試行するだけである。これは、マージャでは、同一ストリームでの順序だけを保証すればよいので²、言い換えれば、異なる入力ストリーム間での入力順序は必ずしも出力に反映させる必要がないからである。

これ以外の排他制御 (1),(3),(4) でも CS を用いた排他制御を行っているが、KL1 のプログラムでは、ある変数に対して複数の「書き手」が存在することが稀であることや、CS によって「際どい区間」に入っている時間を短く抑えているので、(1) の失敗による (5) の処理増加はほとんど無視できると考えられる。

また同様に、入力ストリームを閉じたり、増やしたりする処理でも単純な処理で直列化しているため、排他制御のコストも小さいと言える。

おわりに

共有メモリマルチプロセッサ上でも、並列性を抑えることなく効率の良いストリームマージ処理が実現できることを示した。本稿で述べた以外にも、MHV とのユニフィケーションにはさまざまな組み合わせがあり、それぞれ排他制御が必要となるが、紙面の関係上割愛した。

本処理方式は、実際に共有メモリマルチプロセッサで並列に動作するシミュレータ上に実装して動作を確認しており、今後、PIM の KL1 言語処理系に実装してその有効性の評価を行いたい。

日頃貴重なご意見、ご討論を頂く ICOT の PIM/Multi-PSI グループの方々に、この場を借りて感謝したい。

参考文献

- [1] A. Goto, et.al., "Overview of the Parallel Inference Machine Architecture (PIM)" In *Proceedings of FGCS'88*, pp.208-229, 1988.
- [2] K. Ueda and T. Chikayama, "Efficient Stream/Array Processing in Logic Programming Language" In *Proceedings of FGCS'84*, pp.317-326, 1984.
- [3] Y. Inamura, et.al., "Optimization Techniques Using the MRB and Their Evaluation on the Multi-PSI/V2" In *Proceedings of NAELP'89*, pp.907-921, 1989.

²同一ストリームでの順序は、(4)まで NewMHV を他のプロセッサに見せないことで保証している。