3B-8

並列推論マシンＰＩＭ
－　ストリームの効率的実現方式　－

久野　英治　　伊藤　徳義　　佐藤　正俊
(沖　電　気)　　(ＩＣＯＴ)

概要

ストリーム並列型言語である　ＧＨＣ (Guarded Horn Clauses) [1] における Packed Stream を用いた効率的な処理方式及び、ソフトウェアシミュレーションによる評価結果について述べる。これによるとリスト表現によるストリームに比して効率的な処理が実現できることが確かめられた。

１．はじめに

第五世代コンピュータの並列型核言語ＫＬ１のベース言語であるＧＨＣはストリーム並列処理を実現する論理型言語である。筆者等は、ＧＨＣをサポートする並列推論マシンの研究を行っており [2]、ＧＨＣの基本機能であるストリーム処理の効率的な実現法を検討してきた [3]。この基本的な考え方は、論理型言語の枠組みを保持しつつ CDRコーディング [4] の手法を利用するストリーム処理プリミティブを提供することである。本稿では、ストリームの共有制御方式、及び、データフロー方式並列推論マシンのシミュレータによる評価について述べる。

２．ストリームの構造

ストリームはその生産者プロセスと消費者プロセス間のインタラクティブな通信手段を提供する。以下にストリーム通信を利用したＧＨＣプログラム例を示す。

```
?- p(X), ···, q(X).                    ··· ゴール
p(X) :- true | X=[A|X1], ···, p(X1).   ··· 生産者
q(X) :- X=[A|X1] | ···, q(S).          ··· 消費者
```

起動された二つのサブゴール p 及び q は変数Ｘを共有する。 p から起動された節(生産者)は、変数Ｘにリストセル [A|X1] を具体化する。この具体化に伴い q から起動される節(消費者)はこのリストセルを受け取る。これらの節はリストの CDR 部を引数に持つ再帰的なゴール呼出しを行うプロセスとして実現されており、結果的に生産者側から消費者側へ一次元リストから成るストリームを介した通信が行われる。

このようなリスト表現のストリームの処理効率改善のために CDRコーディング の手法を用いた処理方式を提案した [3]。即ち、リストの CDR 部を省略し、一次元リストをメモリの連続領域に格納する構造(これを Packed Stream と呼ぶ)を導入した。図1に Packed Stream の構造を示す。 Packed Stream はストリーム記述子 ＳＤ (Stream Descriptor) 及びストリーム要素列を格納するバッファ ＳＢ(Stream Buffer)から構成される。ＳＤにはＳＢ要素へのポインタが格納される。各プロセスはＳＤへのポインタを持っており、ストリーム要素をアクセスする度にストリーム要素へのポインタの内容をインクリメントする(これによってＳＤは次要素のアドレスを指す)。

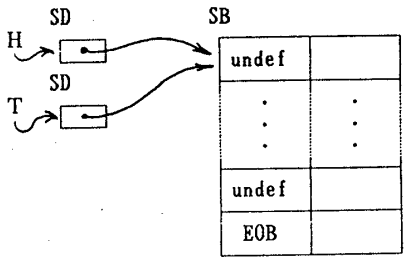

図１．Ｐａｃｋｅｄ　Ｓｔｒｅａｍの構造

前述のプログラムを Packed Stream 操作プリミティブを用いて書き直すと以下のようになる。

```
?- create_stream(H, T) & p(T), q(H).
p(T) :- true | put_stream(T, E, T1), ···, p(T1).
q(H) :- get_stream(H, E, H1) | ···, q(H1).
```

ここで、 create_stream は図1に示したように二つのＳＤ及びプロセス間の共有バッファＳＢを生成し、二つのＳＤを変数Ｈ及びＴに具体化する。このとき、ＳＤの内容はＳＢの先頭アドレスに初期化され、ＳＢの内容は全て未定義変数(undef)に初期化される。なお、バッファの最後には特殊記号 ＥＯＢ(End Of Buffer) が書かれる。

各プロセスは以下のようなストリーム操作プリミティブを用いてストリーム要素の読み書き制御を行う。 put_stream(T, E, T1) プリミティブは、Ｔで示されるＳＤを用いて要素アドレスに要素Ｅを書込む(実際にはＥとバッファ内の値 undef とのユニフィケーションが行われる)、変数Ｔ1にはＴで示していたＳＤが具体化される。 get_stream(H, E, H1)は、Ｈで示されるＳＤを用いて、要素の内容を読み出し、変数Ｅとユニファイする(このプリミティブはバッファ要素が未定義であれば待ち状態となる)。なお、これらのプリミティブにおいて、ＥＯＢを検出したとき(即ち、バッファの最後に到達したとき)、新しいＳＢが割り当てられ、旧ＳＢにチェインされる。

３．ストリームの共有制御

文献［３］で示したように複数の生産者プロセス間で生成されるストリームを非決定的に併合(merge)する処理は、生産者プロセス間で一つのＳＤを共有させることによって簡単に実現できる。

これに対して、論理的に共有されるストリームではＳＤをコピーしなければならない。例えば、以下のような二つの生産者プロセスを起動する節が与えられたとする。

```
p(X) :- true | q(X), r(X).
```

この節の入力変数Ｘにストリームが具体化されているとき、二つのゴール q 及び r で参照されるストリームの内容は

Parallel Inference Machine : PIM - an Efficient Stream Implementation -

KUNO Eiji　　ITO Noriyoshi　　SATO Masatoshi
Oki Electric Industry Co., Ltd.　　ICOT

論理的に同一であることが要求される。即ち、ゴールから起動される両プロセスで n(n=1, 2, ・・・) 番目に参照される要素は同一でなければならない。このため、このようなストリーム共有を行う場合、ＳＤのコピーを生成し、両プロセスに渡す制御を行う。この処理を行うプリミティブを用いると、与えられた節は以下のように書きなおせる。

```
p(X) :- true | share_stream(X, X1, X2) & q(X1),
                                        r(X2).
```

変数ＸにＳＤが具体化されているとき、share_stream(X, X1, X2) はＸで示されるＳＤを新たにコピーし、それぞれ X1及び X2に具体化する。

しかし、これは次のような問題を生む。ストリームを含む構造データが変数Ｘに具体化されてプロセス間で共有される場合、ストリームのＳＤのコピーを生成するためには、構造データをコピーしなければならない。例えば、先に述べた節で、変数Ｘがストリーム Ｓを含む構造体データ f(・・・, S, ・・・) に具体化しているとき shared_stream は図2のようにＳＤの二つのコピー、S1 及びS2 を生成して f(・・・, S1, ・・・) 及び f(・・・, S2, ・・・) を作成し、述語 q 及び r の引数として渡す処理を行う必要がある。このようなコピーを share_stream 実行時に eagerに生成するのはオーバヘッドが大きい。即ち、分散環境における構造データコピー生成の問題を伴うし、また、コピーしようとする構造データ中に一つでも未定義変数が存在すれば、後に変数がストリームに具体化されるかもしれないために具体化まで share_stream 実行を中断しなければならない。

この回避策の一つとして上述のようなストリームを含む構造データの共有を許さない方式がある。 即ち、Packed Stream を構造の要素としてユニファイできないようにすることである。しかし、この方式はあまりに制約が大きい。

もう一つは、変数に共有フラグを用意し、プロセス間で共有される変数とそうでない変数を区別する方法である。これは、前述の share_stream に、Ｘが未定義変数であれば、その共有フラグをセットして共有変数に変えて、X1 及び X2とユニファイする機能を持たせる。これによって、ゴール q 及び r はその引数として共有変数を受取る。同様に、構造データである場合はその共有フラグをオンにし、共有構造データとする。なお、Ｘがアトム定数の場合は X1及び X2にそのまま定数がユニファイされる。このような共有フラグはユニフィケーション時にその部分構造に継承される。即ち、構造データを分解するユニフィケーションを行うとき、その要素として現われる変数や構造データの共有フラグはオンに設定され、それぞれ、共有変数及び共有構造データに更新される。ＳＤのコピーは共有変数とＳＤとのユニフィケーションが行われたとき始めて行われる。このような方式をＳＤの遅延コピー方式と呼ぶ。

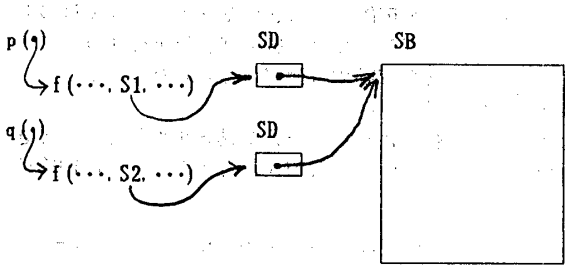

図２．構造体データの eager なコピー

４．評価

上述のようなストリーム操作用プリミティブをデータフローマシンの機械語として実現し、ソフトウェアシミュレーションを行った。 本シミュレーションにおいては、Packed Stream のＳＤやＳＢを処理要素ＰＥ(Processing Element)の局所メモリＬＭ(Local Memory)に格納するモデルを採用した。

シミュレーション対象プログラムとして 7queensを選び、表1のように、①最適化しない版、②ＴＲＯによる最適化版、及び③ＴＲＯと Packed Stream による最適化版の三つのコードについて評価した。

ここで、ＴＲＯ(Tail Recursive Optimization)とは、末尾再帰呼出しを手続き呼出しとしてではなくループとして実現することによる最適化である。表1にこれらの結果を示す(但し、同表はＰＥ１台のときの結果である)。

表１　実行サイクル数と実行命令数

	①	②	③
サイクル数	4835015	1887866	1543482
命令総数	625597	285807	258958

ＴＲＯを行うことによって、手続き呼出しのオーバヘッドが大幅に減少し、性能が約2.5倍向上した。 さらに、Packed Stream を導入することによる最適化を行うと性能は約 10%向上する。

５．終りに

ＧＨＣにおけるストリーム処理を効率化するために Packed Stream を導入し、その基本操作プリミティブを示した。シミュレーションによると、これを導入することにより性能向上が確かめられた。

現在、この操作プリミティブの実験機[3]上のインプリメンテーションを進めており、実験機による評価も行う予定である。この Packed Stream 導入による効果は命令の間の独立性が保証されているデータフローマシンよりも、むしろ現在、検討を進めている中期並列推論マシン[2]で大きいと考えられ、このマシン上での実現も検討している。

最後に、日頃御指導をいただくＩＣＯＴ内田室長、後藤研究員、及び、御討論いただくＰＩＭグループ関係各位に深謝する。

[1] Ueda, K., "Guarded Horn Clauses," ICOT TR-103, June, 1985

[2] 後藤、他、『並列推論マシン ― 中期構想 ―』、本予稿集 3 Ｂ ― 5

[3] Ito, N., et al., "The Dataflow-based Parallel Inference Machine To support Two Basic Language in KL1," IFIP-10 Working Conf. on FGCA, Manchester, July, 1985.

[4] 黒川、井田、『リスト処理とアーキテクチャ』、情報処理、23巻、8号、1982