TR-309

並列推論マシンPIM-Dにおける ストリーム処理方式の評価

久野英治，伊藤徳義，大原輝彦
（沖電気）

October, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列推論マシンＰＩＭ－Ｄにおける ストリーム処理方式の評価

An Evaluation of Stream processing on a Prallel Inference Machine PIM-D

久野　英治　　伊藤　徳義　　大原輝彦
KUNO Eiji　　ITO Noriyoshi　　OOHARA Teruhiko
沖電気工業株式会社
Oki Electric Industry Co., Ltd.

あらまし

ＰＩＭ－Ｄ (Parallel Inference Machinebased Data-flow model) [1] [6] [9] は並列論理型言語ＫＬ１ (Kernel Language 1st version) [5] を目標言語とする並列推論マシンである。筆者等は、１６台の処理要素および１５台の構造メモリモジュールから成るＰＩＭ－Ｄ実験機を試作し、その評価を行ってきた[3] [4]。本稿ではＫＬ１のベース言語として選択されたＧＨＣ (Guarded Horn Clauses) [10] における効率的なストリーム処理方式[2] [7]、インプリメンテーション、及びＰＩＭ－Ｄ上におけるそれらの評価結果について述べる。評価結果によれば、Packed-stream の導入による実行性能の向上が見られ、共有メモリに対するアクセス数も半減することから、本方式の有効性が確認できた。

１．はじめに

ＧＨＣはプロセス間の通信をストリームとして実現することを基本とする言語であり、このストリーム処理を効率良く行うことによってシステム性能が大幅に改善される可能性がある。ＧＨＣにおけるストリームはリスト構造として表現され、その要素の書込みや読出しはリスト間のユニフィケーションによって実現される。また典型的な探索問題において頻繁に使用されるストリームの併合処理は merge プロセスとして実現される。このプロセスは複数の生産者プロセスから生成されるストリーム要素列を受取り、消費者プロセスに対するストリームにこれらを非決定的に書込む処理を行う。このようなストリーム処理の実現上の問題点としては、メモリセルの割り当て・解放の負担、或いは mergeプロセス管理のオーバヘッドなどが挙げられていた。

本稿で提案するストリーム処理方式は、ＧＨＣにおける論理的枠組みを保持しつつ、効率良いデータ構造、およびその操作プリミティブを提供するものである。ストリームは、ＣＤＲcoding[8] の手法を用いてストリーム要素列を圧縮格納したバッファとそのポインタによって表現される。このようなストリームを Packed-stream と呼び、Packed-stream に対する読み書きは、ポインタの指すバッファ要素とのユニフィケーションを行うプリミティブとして実現される。　同様に、非決定的な併合処理も、複数の生産者プロセス間でバッファポインタを共有させることによって、プリミティブレベルで実現できるので、プロセスによって実現する場合に比べて効率が良い。　本稿では、上記のストリーム処理のためのプリミティブのＰＩＭ－Ｄ上における試験的なインプリメント手法、及びその評価結果について報告する。

以下、２章では Packed-stream の処理方式、３章では実験機における実現法、４章では評価結果について述べ、５章はそれらの考察である。

2．ストリーム

2.1 Packed-stream

並列論理型言語ＧＨＣは、並列性を陽に記述する言語であり、ストリームによる並列プロセス間の通信手段を提供する。この言語の並列操作の最小単位をプロセスと呼び、ゴールによって起動される述語に相当する。述語呼出しによって生成されるプロセス間で共有される論理変数を介してメッセージ通信を行う。このとき、起動されたプロセス間で通信されるメッセージ系列はストリームとして実現される。

ストリームは任意の要素系列からなる一次元の構造データとして表現され、要素系列を生成する生産者プロセスとそれらを消費する消費者プロセスの間のインタラクティブな通信手段を提供する。これらプロセス間のメッセージデータ送受のために、各々のプロセスはストリーム内の要素とのユニフィケーションを行い、ストリーム通信を実現する。

ストリームを構成するリストセルは、生産者プロセスによって割り当てられ、消費者プロセスによって解放される。そのため、ストリームを多用する生産者・消費者問題では、ストリーム処理におけるリストセルの割り当て・解放処理の割合が実際の処理に対して大きな比重を占めるため、システム性能を低下させる。

この性能低下の原因となるストリームの割り当て・解放の負担を減少させるために、ストリームをＣＤＲcodingによる手法を用いて適当な大きさのバッファとして設定する。これはリストセルのCDR部を省略して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）に変換してストリームを表現する手法である。Packed-streamはストリーム要素列を格納するストリームバッファで構成される。各プロセスはバッファの任意の場所を指定するアドレスを持ち、ストリーム要素を読み書きするたびにそのアドレスを１語進めてバッファ内の次要素を指定するアドレスに変更する。

2.2 基本操作と構造

Packed-streamの基本操作は、ストリームバッファの割り当て、ストリームへのデータ書き込み、およびストリームからのデータ読出しである。この一連の操作がPacked-streamを用いたプロセス間通信を実現している。ストリームバッファは、バッファを参照するプロセスの数を格納する領域とストリーム要素を格納する領域からなる。

ストリームを割り当てるプリミティブは明示的に表現され、このプリミティブが実行されるとバッファの割り当てと初期化が行われる。例えば、生産者プロセスqと消費者プロセスrが以下のプログラムで与えられているとする。

```
?- q(Y),r(Y).
```

Packed-streamを割り当てるプリミティブをcreate_streamとして表現すると、次のように書換えることができる。

```
?- create_stream(H,T),q(H),r(T).
```

導入した述語の実行で得られたバッファの先頭アドレスは、HとTに具体化されて生産者プロセスと消費者プロセスに渡され、同時にバッファの参照数が"２"に設定される。

データの書き込みにおいても同様な表現が可能となる。例えば、生産者プロセスqにおけるストリームの生成を示すプログラムを、

```
q(Y) :- true | Y=[Y1|Yr],q(Yr).
```

と記述する。このとき、Y1はなんらかのデータが具体化されているものとする。ストリームに書き込みを行うPacked-streamのプリミティブをput_streamとすれば、生産者プロセスのプログラムは、

```
q(H) :- true | put_stream(E,H,H1),q(H1).
```

と書換えられる。具体化されたデータを表わす変数Eは、Hに具体化されたバッファアドレスの内容とユニフィケーションが行われる。Hのバッファアドレスを１語進めた値をＨ１に具体化する。

データの読出しでも、例えば、消費者プロセスrのプログラムが以下に与えられている場合、

```
r(Y) :- Y=[Y1|Yr] | r(Yr).
```

実際のプログラムではＹ１はボディ部でなんらかの操作を行うが、その処理は省略する。データの読出しを行うプリミティブをget_streamとして表現すれば、

```
r(T) :- get_stream(E,T,T1) | r(T1).
```

と書換えられる。get_streamはTに具体化したバッファアドレスの内容とEのユニフィケーションを行い、結果をEに具体化する。また、Ｔ１にはTのバッファアドレスを１語進めた値が具体化される。

バッファの参照数を減らす処理は、ストリームの終了を示すデータを書き込んだ時、そのデータを読出した時、または新バッファを指定するデータを読出した時に行われる。以上がPacked-streamの基本操作である。

2.3 併合処理

典型的な探索問題で頻繁に使用されるストリームの非決定的な併合処理は、merge述語のプロセスを起動することによって実現されていた。以下にそのプログラム例を示す。

```
?- merge(S,T,U),q1(S),q2(T),r(U).
```

q1、およびq2は生産者プロセスを示し、rは消費者プロセスを示す。この場合のmerge述語は次のように書ける。

```
merge(S,T,U) :- S=[E|S1] | U=[E|U1],
                merge(T,S1,U1).
merge(S,T,U) :- T=[E|T1] | U=[E|U1],
                merge(T1,S,U1).
merge(S,T,U) :- S=[] | U=T.
merge(S,T,U) :- T=[] | U=S.
```

従って、併合処理における mergeプロセス管理のオーバヘッドのほうが併合処理自体よりも大きくなる。しかし、この処理もストリームバッファを指定するアドレスを複数の生産者プロセスによって共有することにより、プリミティブレベルでの実現が可能となる。すなわち、ストリームバッファを指定するアドレスを格納するためのメモリセル（記述子）を確保し、その記述子へのアドレスを複数の生産者プロセスに対して渡すプリミティブを用意すればよい。このプリミティブをmerge_streamとして上記のプログラムを置換えると、

```
?- create_stream(H,T).
     merge_stream(H,H1,H2).
      q1(H1),q2(H2),r(T).
```

となり、merge述語は不要となる。Hに具体化されたバッファアドレスは、merge_streamによって、割り当てられた記述子に格納され、H1、H2に記述子を指定するアドレスが具体化される。このプリミティブは記述子に対する操作のみを行い、実際の併合処理はストリームバッファへの書き込み要求をメモリセルに対して発することにより実行される。従って、併合処理プロセスの管理が不要となり、プロセス管理のオーバヘッドが解消できる。

また複数の生産者プロセス、或いは複数の消費者プロセスによるデータの共有は、ポインタとして渡されるデータのタグ部に1ビットのフラグを用意することによって実現できる。なぜなら、ポインタを共有したとしても実際には、データの型のみを必要とする処理も多いので、ポインタが示す構造データをコピーするのは無駄である。また、未定義変数を含む構造データのコピーの場合には、未定義変数に成るまでのデータをコピーできるが、以降、未定義変数になんらかのデータが具体化されるまでコピーを待たなければならない。そこで、必要なときにコピー出来るように共有フラグを設け、部分構造の分解によるユニフィケーション時に未定義変数に共有フラグを設定することによって構造データの部分構造に共有フラグを継承すればよい。

Packed-streamにおけるストリームバッファには、バッファを参照するプロセス数が"零"になったところで回収される。初期状態のストリームバッファは、未定義変数とバッファの終端を示すデータが書き込まれる。バッファの終端に達して新たなデータが格納できなくなった場合には、新しいバッファを割り当て、旧バッファの終端に新バッファの先頭アドレスを格納し、バッファのチェインを設ける。

ストリームの併合処理に用いる記述子に対しても参照数管理を行い、その参照数が"零"になると、記述子が回収され、ストリームバッファの参照数を減らし、バッファ内の未定義変数とストリームの終端を示すデータとのユニフィケーションを行って結果をバッファに格納する。

このようなインプリメンテーションによって、プログラム上は論理型言語の枠組みを壊すことなく、ストリーム処理が効率よく実行できる。即ち、Packed-streamのプリミティブを組込み述語として導入することにより、従来ストリーム要素の書込みや読出しの度にリストセルの割当てや解放が行われていたものが、バッファを一度設定しバッファ内の読み書きでバッファ境界に達したときのみ、バッファや記述子を管理すればよいことになる。

3．インプリメンテーション

3.1 ＰＩＭ－Ｄ実験機

ＰＩＭ－Ｄはデータフローモデルをベースとした密結合型の多重プロセッサシステムである。

実験機を構成する処理要素と構造メモリモジュールに用いる制御プロセッサは、同一の構造を持ち、それぞれのマイクロプログラムにより機能を分担している。さらにプロセッサには、この機能を補佐する役割として、データの型を高速に判定するためのタグ判定ハードウェア機構も内蔵している。また、処理要素には、データフロー発火制御を行うハードウェアを用意している。

ネットワークでは、　共通バスとネットワークインタフェースによって低遅延通信を実現している。最下層のバスは４台の処理要素と４台の構造メモリモジュールを持ち、これによって１クラスタを構成する。実験機では，４クラスタを共通バスで接続した階層型のネットワーク構成をとっている。

3.2 プリミティブ

本方式では、 Packed-stream　を実現するために、ストリームバッファを構造メモリ上に設定する。この場合、 Packed-streamに対する処理は、データフローグラフのストリーム操作プリミティブ実行における副作用として実現されるために、プログラム実行のなかに隠蔽されることになる。以下、　インプリメントしたプリミティブの機能と、ストリームバッファ操作指令を示す。

. create_stream

このプリミティブは、ストリームバッファを設定し、その先頭アドレスを生産者プロセス、および消費者プロセスに渡す機能を有している。

予め幾つかのストリームバッファ先頭アドレスを要素プロセッサの局所メモリ（ＬＭ）内に用意しておき、上記プリミティブが実行されると１つのバッファアドレスが取りだされる。取りだされた値はトークンとしてプロセスに割り当てられる。一方、そのバッファが存在する構造メモリモジュールに対してバッファ割り当て指令パケットを送出する。そしてパケットを受け取った構造メモリで、バッファを初期化する。

. put_packed_stream

ストリームバッファを指定するアドレス、または記述子によって間接指定されるバッファアドレスの内容と、他方の入力データとのユニフィケーションを行い、その結果を指定アドレスに格納する。

データの型がストリームバッファを指定する場合は、そのアドレスを１語進めてトークンとし、他の場合は間接指定していたアドレスを１語進めて記述子に格納し、記述子へのアドレスをトークンとして次のプロセスに渡す。構造メモリモジュールに対しては、ストリームバッファ、または記述子を指定するアドレスにデータを書き込むための指令パケットを送る。通常、ストリームバッファを指定するアドレスであれば、そのアドレスを１語進めた値をトークンとするが、それがバッファ境界の場合、次のバッファを得るために構造メモリに指令パケットを送る必要があるので、境界で得た値をトークンとする。

. put_end_of_stream

ストリームの終端を示すデータを書き込むことを除いては、前述の put_packed_stream プリミティブとほぼ同一の機能を有している。

構造メモリにおけるストリームバッファに対する操作は、 put_packed_streamと同様の指令パケットを送るのみであるが、パケットを受け取るとバッファの参照数を "１" だけ減じなければならない。記述子に対しても参照数を減じ、参照数が "零" になったところで終端を示すデータをバッファに書き込み、バッファの参照数を "１" 減じる。

. get_packed_stream

ストリームバッファ内を指定するアドレスの内容を獲得すると同時に、アドレスを１語進めた値を次の消費者プロセスに渡す機能を有する。

バッファ内の要素を得るために指令パケットを該当する構造メモリに送出し、得た値をトークンとする。また、ストリームバッファを指示する値が、バッファ境界でなければ、アドレスを１語進めてトークンとし、 境界であれば、 次なるバッファへのアドレスを得るための指令パケットを送り、得た値をトークンとして、次のプロセスに渡す。　ストリームバッファの要素がストリームの終端を示すデータであれば、バッファの参照数を "１" 減じる。

. test_packed_stream

ストリームバッファを指示するアドレスの内容を獲得する機能を有する。

この機能はバッファ内に格納されているデータの型を得ることを目的としており、ストリームの終端を示すデータを得ても参照数の操作は行わない。

. merge_packed_stream

ストリームバッファを指定するアドレスを、このプリミティブ実行によって割り当てられた記述子に格納し、そのアドレスを他のプロセスに渡す機能を有する。

入力データの型がストリームバッファを指示するアドレスであれば、予め局所メモリに格納されていた記述子アドレスを割り当て、入力データをその記述子に格納するための指令パケットを送る。記述子を示すデータの型であれば、必要な参照数を加えるための指令パケットを送出する。

. share_packed_stream

入力データの型が構造データ、或いは変数の場合、データのタグ部に共有フラグを設定する機能を有する。

入力データが構造データか変数の場合、共有フラグを設定した値をトークンとし、他の場合は入力データをトークンとする。データが構造メモリ上にあれば、参照数を増やすか、または減らすための指令パケットを送る。

他に制約条件として、チェイン用の新バッファ割り当てや記述子割り当ては、同一の構造メモリモジュール内に設定する。またバッファを回収する際には、さらに格納していたデータについても参照数操作が必要なデータか否かの判定を行い、参照数を減ずるための操作、またはパケットの送りださなければならない。

構造メモリ上には、従来のリストセル処理とPacked-stream処理を共存させる。特に通常のデータ処理はリスト処理とし、ストリーム処理のみを Packed-stream とすることもできる。

4. 評価

4.1 評価プログラム

評価プログラムとして素数生成問題と Queens 問題を選定した。素数生成問題は、典型的な generate_and_test であり、全てのプロセス間通信はストリーム処理によって実現される。また、候補列ストリームの生成、新たな候補列ストリームの試験と新たなストリームの生成は、いずれも再帰的な述語呼出しによって実現されるので、再帰呼出しの最適化も施せる。そのためプログラムとしては次の処理を施したものを用いる。

(1) リストセルのストリーム
(2) (1)に Packed-stream を施したもの
(3) (1)に末尾再帰呼出し最適化を施したもの
(4) (3)に Packed-stream を施したもの

一方、Queens 問題は、並列性が高く、非決定的なストリームの併合処理を行っているので適当である。比較対象となるプログラムとしては、次の処理を施したものを用いる。

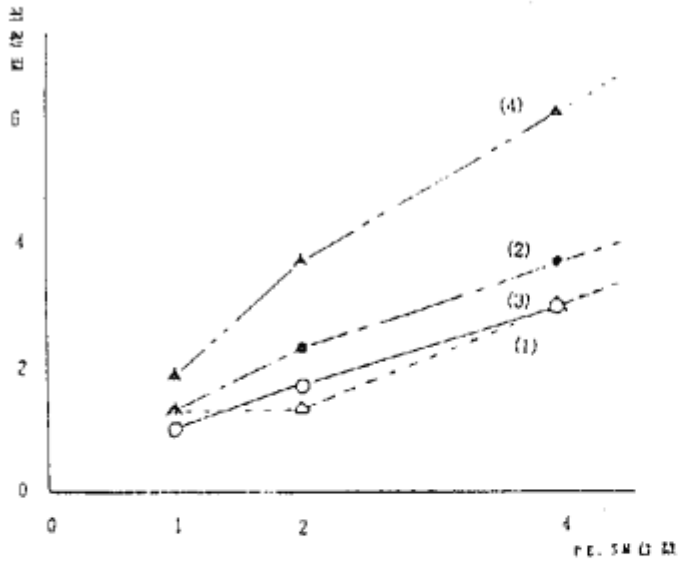
(5) リストセルのストリーム
(6) 併合処理のみの Packed-strem
(7) 全ての構造データの Packed-stream

評価のポイントは、ストリーム処理プリミティブ導入による実行効率の向上、及び共有された構造メモリに対するアクセス頻度（つまり、パケット数）の測定である。リストセルストリームとPacked-streamの比較は、ストリームバッファの大きさを"8"に設定したものを代表値として用いる。Packed-stream導入の効果を測定する尺度として実行性能による比較と構造メモリ受信パケット数によるアクセス数を測定した。その理由は、マルチプロセッサシステムにおける共有メモリに対するアクセス競合の問題を扱うためである。さらに、ストリームバッファの大きさを変更して、実行性能に対する影響と構造メモリのアクセス数も測定を行った。

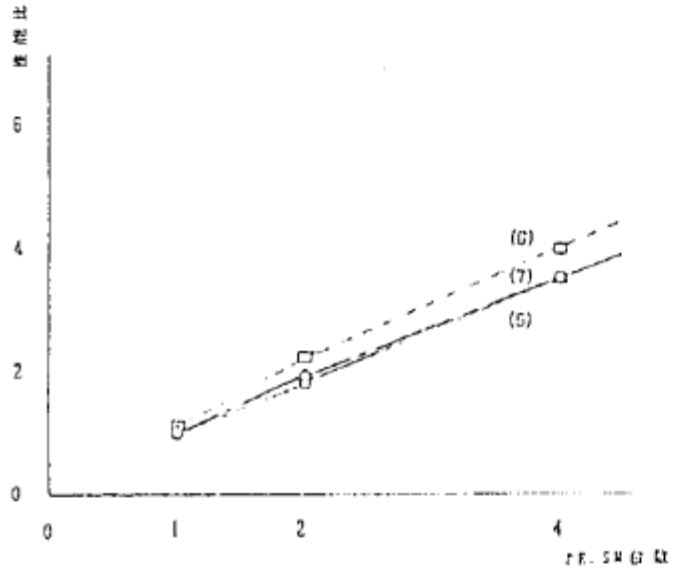
4.2 実行性能と台数効果

第1図は 200までの素数生成問題における台数効果と実行性能を示したものである。いずれも Packed-stream を導入したプログラム (2)、(4)の方が、通常のリストセルのストリーム (1)、(3)と較べてプログラム実行における性能が30%ないし40%向上しており、効果が大きいことがわかる。

第2図は 6queens問題における実行性能と台数効果を示したものである。ここでは、(5)のプログラムに対して、mergeプロセスを merge-packed-



第１図 素数問題における実行性能と台数効果



第２図 Queens 問題における性能と台数効果

stream に置換えたプログラム(6)の方が１５％程度の性能向上が得られている。全ての構造データをストリームとして扱った(7)は(5)と殆ど変らなかった。

4.3 構造メモリアクセス

前節で用いた評価結果を構造メモリモジュールの視点から見る。 いずれも１台の処理要素と構造メモリモジュールにおける受信パケット数を示す。他の場合はそれにプラスアルファすれば見当がつくので省略することとする。

表３は素数生成問題実行時の構造メモリモジュールのパケット受信数である。 Packed-streamを用いると、構造メモリのアクセス数がほぼ３０％に減少した。

表３ 素数生成問題実行時のパケット受信数

プログラム	受信パケット数
(1)	23750
(2)	3179
(3)	20111
(4)	3179

表４ Queens 問題実行時のパケット受信数

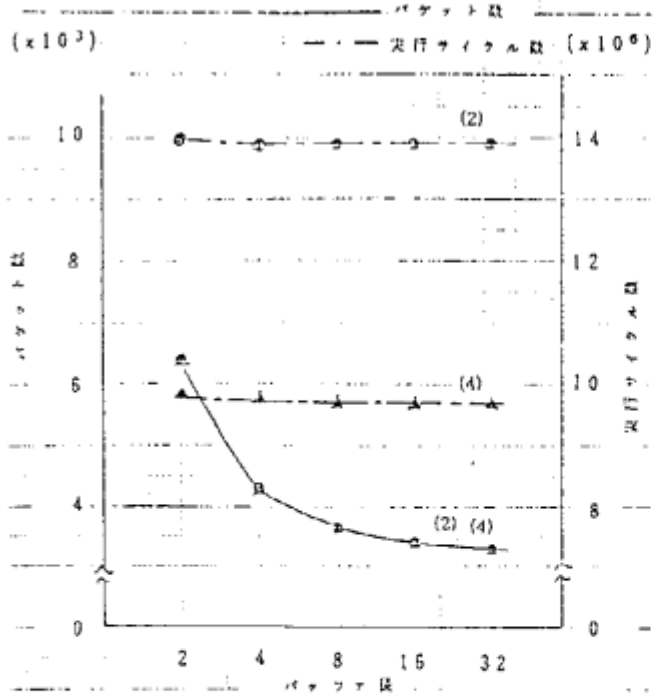
プログラム	受信パケット数
(5)	25641
(6)	21782
(7)	11385

同様にして、 6queens問題でのパケット数を示したのが、表４である。 merge部分を換えるだけで２０％弱のアクセスの減少、全てをストリームに置換えると約６０％構造メモリアクセスが減少する。

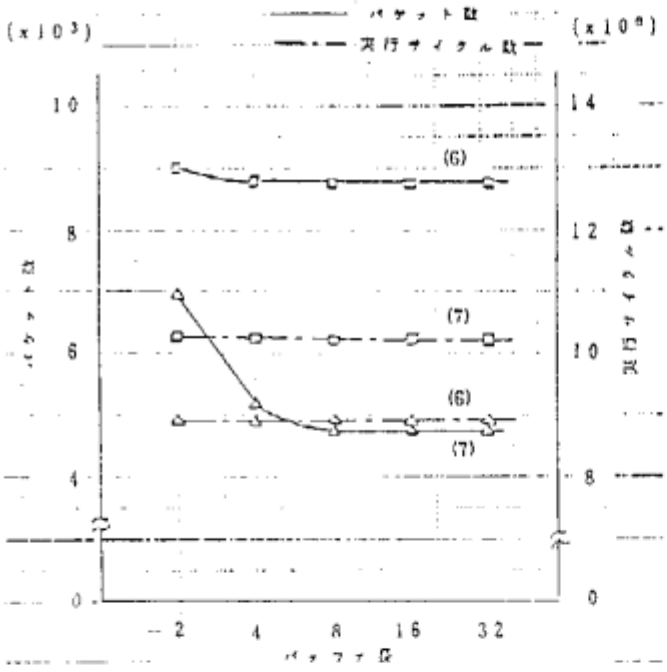
4.4 バッファ長

ストリームバッファの大きさを変化させて実行した結果を以下に示す。ここでは、１０００迄の素数生成問題と、 7queens問題に対して、ストリームのバッファ長を、２、４、８、１６、及び、３２と変えてプログラムを実行させ、そのときの実行サイクル数と構造メモリに対するアクセス数を測定した。

第５図は、素数生成問題である。実線がメモリアクセス数であり、一点鎖線がプログラム実行サイクル数である。メモリアクセスが変化するにも拘わらず、実行サイクル数には殆ど変化がない。しかし、末尾再帰最適化処理を施すことによって約３０％の性能向上を得た。



第5図　バッファ長変更時の実行性能と
パケット受信数（素数生成問題）



第6図　バッファ長変更時の実行性能と
パケット受信数（Queens 問題）

第6図のグラフは 7queens 問題である。本来 merge のみを Packed-stream としたものはバッファ長を変えてもアクセス数に大きな変化は見られないし、実行サイクル数にも影響しない。全ての構造データを Packed-stream に変えたプログラムはバッファ長を変えることによってアクセス数に変化が生じるがやはり実行サイクル数には大きな影響を及ぼさない。

5．考察

4.2 節の結果からいずれも Packed-stream を導入する効果は大きい。特にストリーム操作だけを行っている素数生成問題では、大きな効果が得られる。ところが Queens 問題では、併合処理用のストリームを Packed-stream に変えた(6)のほうが、すべてデータを変更した(7)と比べて性能が高い。(7)のプログラムでは併合処理呼び出し毎にストリームバッファを割り当てるので、解を得るために一方のバッファが他方のバッファの内容を複写する必要がある。そのため、バッファ複写用の手続き呼び出しを設定することとした。この呼び出しによるオーバヘッドが、丁度(5)における併合処理オーバヘッドに相当する。ところが、(6)では1つのストリームバッファに対して併合処理を行っているので、上記の複写のためのオーバヘッドをまったく無視できる。従って、論理的に一つのストリームとみなせるものは、(6)と同様に、統一的に扱かったほうが Packed-stream の効果を期待できる。

4.3 節によれば、構造メモリアクセスが幾段に減少しているにも拘わらず、その効果が実行性能に反映されないことは、共有メモリに対するアクセスが本実験機の性能のボトルネックとなっていないことを示している。

4.4 節では、規模の大きなプログラムをバッファ長を変えて実行させても性能に影響がない。バッファ長の変更によって、処理要素の命令実行数が変わらないが、構造メモリへのパケット数（メモリアクセス数）が変わるだけである。即ち、メモリアクセスは、データフローにおける副作用として実現されていることにより、構造メモリにおける指令パケット処理がプリミティブ実行のなかに隠蔽される。また Packed-stream 導入により、隠蔽できる程度の処理量になっている。また Packed-stream におけるバッファの大きさ「2」の場合とリストセルは同じ構造である。しかも、本方式を用いることにより、従来の方式と比較してメモリ操作処理を減らすのに有効である。つまり、GHC で共有メモリに対するアクセス数を減らすためには、通信のリストデータとストリームデータを統一的に扱い、併合処理用の記述子を設けて本方式を適用すればよい。

4.3節と4.4節で用いた素数生成問題(2)、(4)は、メモリアクセス数がいずれも同じであった。その

性能の差は、ちょうど末尾再帰最適化を施したことによるプロセス管理のオーバヘッド分であるといえる。

以上の考察によると、ストリーム処理にPacked-streamを導入することは、処理効率を高めるのに非常に有効である。また、本方式は共有メモリアクセスを減らすのに効果があるため、マルチプロセッサシステム上での大規模プログラム実行時における共有メモリアクセスの競合を、従来の処理方式と比較して半減できる可能性がある。そして、ストリームと構造データを統一的に扱えれば、さらに効率的な処理も期待できる。しかし、その実現のためにはこれまで述べたプリミティブだけでは不足している。例えば、append述語でストリーム操作を行うためには、ストリームどうしの連結を行わなければならないが、それに相当するプリミティブのインプリメントも必要である。

6．おわりに

本稿では、GHCにおける効率的なストリーム処理方式、PIM－D上でのインプリメンテーション、及びその評価について述べた。その結果、packed-streamはストリームの非決定的な併合処理に対して有効であり、また、共有メモリに対するアクセス数を格段に減らすことができることが明らかとなった。従って、本方式は、マルチプロセッサシステムを構築するさいの共有メモリに対するメモリアクセス競合にも有効である。

謝辞

最後に、日頃御指導賜く ICOT 第4研究室の内田室長、御討論賜く後藤研究員、PIMグループ、及び沖電気のPIM－D関係者各位に深謝する。

（参考文献）

[1] Ito.N. and Masuda.K. "Parallel Inference Machine Based on the Data Flow Model." Proceedings of International Workshop on High Level Computer Architecture 84. Los Angeles.California. May 1984.

[2] Ito.N., Kishi.M., Kuno.E., and Rokusawa.K., "The Dataflow-based Parallel Inferrence Machine To Support Two Basic Languages in KL1." Proceedings of IFIP Working Conferrence on Fifth Generation Computer Architecture.UMIST(Manchester), July. 1985.

[3] Ito.N., Sato.M., Kuno.E., and Rokusawa.K., "The Arhchitecture and Preliminary Evaluation Results of The Experimental Parallel Inference Machine PIM-D." Proceedings of ISCA Tokyo.Japan. June 2-5.1986

[4] 伊藤、他、「並列推論マシンＰＩＭ－Ｄの評価」 データフローワークショップ予稿集 5月 １９８６。

[5] Kimura.Y. and T.Chikayama. "An Abstract KL1 Machine and Its Instruction Set." Proceedings of 4th Symposium on Logic Programming.Aug. 1987.

[6] 来住、他、「データフロー方式並列推論マシン実験機のアーキテクチャ」 第３０回情報処理学会全国大会予稿集、１９８５。

[7] 久野、他、「並列推論マシンＰＩＭ －ストリームの効率的実現方式－」 第３３回情報処理学会全国大会予稿集、１９８６。

[8] 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

[9] 大原、他、「並列推論マシンＰＩＭ－Ｄ」 データフローワークショップ予稿集、5月 １９８６。

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