TM-1173

分散メモリ型並列計算機におけるストリーム通信実装方式の評価

小西　弘一、丸山　勉、小長谷　明彦（日電）、
吉田　かおる、近山　隆

April, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

分散メモリ型並列計算機におけるストリーム通信実装方式の評価 (extended abstract)

小西 弘一　丸山 勉　小長谷 明彦　†吉田 かおる　†近山 隆

日本電気（株）C&C システム研究所
216 川崎市宮前区宮崎 4-1-1
{konishi, maruyama, konagaya}@csl.cl.nec.co.jp

†（財）新世代コンピュータ技術開発機構
108 港区三田 1-4-28 三田国際ビル 21 階
{yoshida, chikayama}@icot.or.jp

1992 年 2 月 10 日

概要

分散メモリ型並列計算機上でのストリーム通信の実装方式の評価結果について述べる。筆者らはストリームに基づく並列オブジェクト指向言語処理系 A'UM-90 の開発を行っている。これまで、ストリーム通信の実装方式として、通信経路に流れるメッセージあたりの処理量を減らし、また各通信経路を最初に流れるメッセージの遅延を小さくすることをを特徴とする方式を提案し、これを用いて A'UM-90 のストリーム通信を実装した。本稿では簡単なプログラムを A'UM-90 で実行して上記実装方式を評価した結果を述べる。評価には、共有メモリ上に実現した仮想的なメッセージ通信環境を用いた。通信コストの差が最も現れにくい環境下であるにも関わらず、筆者らの方式は、メッセージ当たりの処理量を最適化していない方式と比べ、全処理時間の比で 1 割高速であった。

1　はじめに

本稿では、筆者らが提案した、分散メモリ型並列計算機におけるストリーム通信の実装方式の評価結果について述べる。筆者らはストリームに基づく並列オブジェクト指向言語 A'UM[2] の処理系 A'UM-90[3] の開発を行っている。ストリーム通信モデルはプロセス間通信のモデルであり、主に並列論理型言語などで用いられている [1]。ストリームによる通信は非同期で non-strict であり、またメッセージの到着順序を容易に制御することができる、メッセージ送信後に通信の当時者以外の第三者が受信者を定めることができる、などの特徴を持つ。

従来の処理系は、必ずしも最高の効率でストリーム通信を実現していない。これは、ストリーム通信がプログラミング上の技法として実現されることが多かったことによる。この場合、言語処理系はストリーム通信の存在を認識できず、ストリーム通信の処理効率に配慮して最適化などを行うことができなかった。

A'UM はストリーム通信モデルをベースとした並列オブジェクト指向言語であり、ストリームを言語モデル中に採り入れている。このため、A'UM に対してはストリーム通信を効率良く実行する処理系を作成することができる。筆者らは、共有メモリを持たない並列計算機を対象とする効率のよいストリーム通信実装方式を提案し [4]、これを A'UM-90 の開発に採り入れて評価を行った。この方式は通信経路に流れるメッセージあたりの処理量を減らし、また各通信経路を最初に流れるメッセージの遅延を小さくすることを特徴とする。

評価は共有メモリ上に実現した仮想的なメッセージ通信環境で行った。このため、実際のメッセージ通信型計算機に比べ通信速度が非常に速い、という条件下であったにも関わらず、筆者らの方式は、メッセージ当たりの処理量を最適化していない方式と比べ、全処理時間の比で 1 割高速であった。

以下、次節では A'UM 言語の特徴をのべ、さらに 3 節で A'UM のストリーム通信モデルについて説明する。4 節では筆者らのストリーム通信の実装方針を述べ、それに基づいた実装方式の詳細は 5 節に示す。そして 6 節に評価方法および結果を示す。

2　A'UM

A'UM の主な特徴は以下の 3 つである。

- 並列オブジェクト指向モデル
- 並列性を基本とした記述
- ストリームによる通信

A'UM 言語モデルの構成要素は、オブジェクトとストリームとメッセージである。オブジェクトはメッセージを受けとって、そのメッセージによって選択されるメソッドを実行する。オブジェクトの振舞いは、そのオブジェクトのクラスに定義されている。各オブジェクトは原則として並行して動作する。

A'UM のプログラムはクラス定義の集まりである。クラス定義には、継承するクラスとスロット名の宣言、およびメソッド定義を含む。メソッド定義は、メソッドセレクタとアクションの記述の集まりである。アクションの実行順序を記述する構文はなく、基本的にはすべての処理が並行動作する。処理の実行順序は、値の依存関係、メッセージの発信と受信の前後関係、およびメッセージの到着順序によって決まる。

全ての通信はストリームを介して行われる。ストリーム通信は A'UM の最も重要な特徴なので、更に次節で説明する。

3 A'UM のストリーム通信

ストリームはメッセージの追い越しがないことが保証された通信経路であり、メッセージの順序集合である。複数のストリームを接続、合流させてストリームのネットワークを作ることによって、メッセージの半順序集合を表現することができる。ネットワークに送られたメッセージは、ネットワークの先頭のストリームがオブジェクトに接続されるまでオブジェクトに届かない。そして接続が行われると、ネットワークが表す半順序にしたがってオブジェクトに流れ込む。

ストリームへの参照は、メッセージの引数としてオブジェクト間で受け渡すことができる。また、ストリームの接続や合流は、メッセージの発信者受信者に限らず任意のオブジェクトが行うことができる。

あるストリームに対して、以後メッセージ送信を行わないことを宣言する操作を閉鎖と呼ぶ。閉鎖は、ストリームへの参照がなくなる時に自動的に行われる。ストリームの閉鎖は、そのストリームの接続先に伝えられる。接続先がオブジェクトであれば、閉鎖はメッセージ同様にオブジェクトのメソッドを起動する。また、ストリームの合流地点に伝達された閉鎖が、メッセージの流れを変えることもある。

ストリーム通信の主な特徴は以下の二つである。

- 多対一の通信において複数の送信者が非同期に発信するメッセージの到着順序を簡潔に表現できる。

  ストリームはメッセージ間の順序を表現する。さらにストリーム同士を接続することによって、ストリーム間の順序を表すことができる。そこで、各送信者が個別のストリームにメッセージを流す一方で、それらのストリームを適切に接続することにより、メッセージの到着順序を指定することができる。

- 接続先が未定のストリームを引数に持つメッセージを送信できる。

  ストリーム通信では、接続先が未定のストリームを引数に持つメッセージを送信することができる。これにより、その引数であるストリームの接続先を求める計算と、そのメッセージによって起動されるメソッドの処理の間の並列性を表現できる。

以上、A'UM の概要と、A'UM のストリーム通信の特徴について述べた。次節では、A'UM-90 におけるストリーム通信の実装方針について述べる。

4 実装の方針

A'UM のプログラムでは、規模が大きくなるほど、多くの PE 間に渡って伸びるストリームが数多く生成されることが予想される。そこで、このようなプログラムにおける処理時間の増大を抑えることを狙って、次の方針を定めた。

1. PE 間に渡るメッセージの配送は最終的な受信者の存在する PE への直接転送で行う。

2. ストリームを最初に流れるメッセージだけは、配送の遅延を防ぐ。

以下、これらの方針について説明する。

4.1 最終受信者への直接転送

一般的に言って、PE 間のメッセージ通信は重い処理であり、また通信速度自体も処理速度に比べてかなり遅い。したがって PE 間通信はできるだけ避けなければならない。

メッセージを常にストリームの連鎖に沿って流すとすると、メッセージの到着順序は用意に保証できる。また、この方式では直接隣接するストリームが判明した時点で直ちにメッセージを下流に流せるため、ストリームの連鎖が短かければ通信遅延が最も小さくなる。しかし、ストリームの連鎖が長いと、多くの PE 間でメッセージの転送が繰り返されることによって処理コストが著しく増す恐れがある。

そこで、プロセッサ間に渡るストリームの連鎖がある場合には、この連鎖を制御メッセージによりデリファレンスし、各 PE に最終受信者のアドレスを知らせる。最終受信者のアドレスが判明するまでは、各 PE はメッセージを下流に流さず、デリファレンスが終ってから直接最終受信者の PE にメッセージを転送する。この方式では通常のメッセージ一つにつき、PE 間転送は高々 1 回しか生じない。

4.2 第一メッセージの遅延の削減

最終受信者への直接転送を行う方式では、メッセージをストリームの連鎖に沿って流す方式に比べ、個々のメッセージが到着する時刻はデリファレンスにかかる時間だけ遅くなる。この遅延は、メッセージの最終受信者であるオブジェクトが行う様々な処理の開始を遅らせ、ひいては処理全体の速度を大幅に落す恐れがある。

そこで、ストリームを流れる最初のメッセージだけは、必ずしも最終受信者の判明をまたずに下流の PE に転送する方針をとった。あるメッセージに先行して別のメッセージが同じストリームを流れている場合には、後ろのメッセージは多少の到着が遅れても構わない。前のメッセージによって開始された処理をオブジェクトが行っている間は、後ろのメッセージの到着の有無によらず次の処理は開始されないからである。[1]逆に、ストリームを最初に流れるメッセージの遅延は全体の処理速度に大きな影響を与える可能性がある。

以上をまとめると、基本的にはスループットを重視し、先頭のメッセージについてのみ遅延時間に配慮する方針を取ったと言える。次節では、この方針に従った実装方式の詳細について述べる。

5 実装方式の詳細

5.1 データ構造

個々のストリームは転送先を示すポインタ、リファレンスカウンタ、およびメッセージキューの 3 つによって表現する。以下、これら 3 つを備えた構造体を M node と呼ぶ。

5.2 PE 内処理

PE 内では、ストリームの接続は上流から下流に向けて転送先ポインタを張る操作で表す。転送先ポインタは、アクセスがある度に常にデリファレンスを行って更新する。メッセージの送信はメッセージをキューに入れる操作で表す。ストリームの閉鎖はリファレンスカウントを減らす操作になる。

5.3 アドレスの輸入

ストリームのアドレスは、メッセージの引数としてプロセッサ間を渡る。他 PE にあるストリームへのポインタを輸入した時には、新しい M node を自 PE 内に作り、その転送先ポインタとして輸入先へのア

[1] A'UM モデルでは一つのオブジェクトの異なる世代がメソッドを並行に処理するが、A'UM-90 では各世代を逐次処理する。

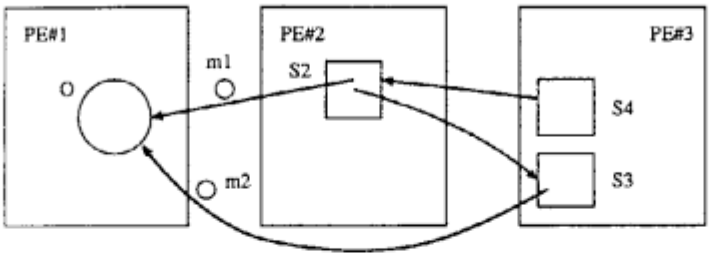

図 1: アドレスの再輸出

ドレスを当座持たせておく。

プロセッサ間に渡るリファンレスカウントは、公知の Weighted Reference Count 方式で管理している。この方式では一本のポインタを輸出する際、このポインタのカウントを 1 ではなく $n > 1$ 増やし、n を輸出先の PE に告げる。これにより、輸出先では、n の範囲内ならば輸入したアドレスをさらに別のプロセッサに再輸出することができる。

ただし、A'UM の通信モデルが保証するメッセージ到着順序を実現するためには、再輸出をしてはならない場合もある。輸入アドレスを再輸出してもよいのは、そのアドレスに宛ててメッセージがまだ一つも送られていない場合に限る。メッセージが既に送られている場合には、輸入アドレスを保持している M node のアドレスを輸出しなければならない。

図 1において、PE#1 のオブジェクト O のアドレスが PE#2 に輸出されて M node S2 になっている。これを PE#3 に輸出して S3 とした場合、A'UM では、S2 から流されたメッセージ m1 が S3 から流されたメッセージ m2 よりも先に最終受信者に届くことを要求するプログラムを書くことができる。これを保証するためには、O のアドレスの再輸出をしてはならず、代わりに S2 のアドレスを PE#3 に輸出して S4 とし、O 宛のメッセージを S4 から流させることにより S2 を経由するようにしなければならない。

以上述べたように、A'UM では、通信モデルがメッセージの到着順序の制御を要求するため、明示的にはストリームを接続しないプログラムでもしばしば M node の連鎖が形成される。そこで、次節では連鎖のデリファレンスについて述べる。

5.4 PE 間でのデリファレンス

PE 間に渡る通信では、デリファレンスを行うために、下流のストリームに最終的な受信者のアドレスを問う制御メッセージを流す。この制御メッセージにはこれを発信したストリームのアドレスを持たせておき、最終的な受信者からの返事を受けとることができるようにする。以下、下流に流す問い合わせのメッセージを WRU(Where are you?) と呼び、これに対する返事を HERE とよぶ。このデリファレンスによって判明した受信者は確定したものであり、以降デリファレンスを繰り返す必要はない。HERE 到着後は最終的な受信者へのポインタをストリームに持たせておき、上流からのメッセージは即座に転送する。

デリファレンスの間、つまり WRU を流して HERE が返ってくるまでの間は、上流からのメッセージはデリファレンス中のストリームのキューに溜めておく。このことから、PE 間での WRU の転送回数は、一つの WRU については高々 2 回となる。長いストリームの連鎖の中のあるストリームが流した WRU は下流側の直接隣接するストリームより先には流れず、そのストリームのデリファレンスが終ってから最終受信者に直接流れるからである。

5.5 デリファレンスとメッセージ順序制御

メッセージを連鎖に沿って流す方式を用いる場合には確実に到着順序を保証できる。一方、デリファレンスを行う方式では、5.3節で述べたような、到着順序を制御するために生じたストリームの連鎖に対して

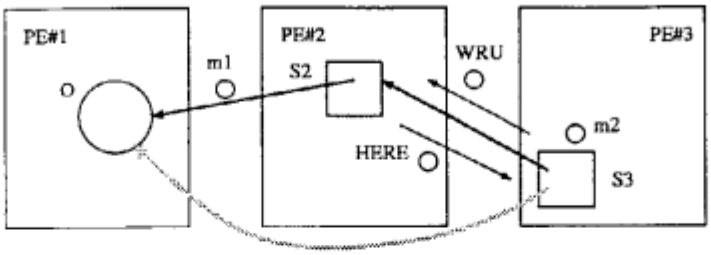

図 2: デリファレンスでのメッセージ順序保証

特に注意を払う必要がある。

上流から流れてきた WRU を受けとったストリームが、既に最終受信者を知っている場合、その WRU の発信者に直接 HERE を返して良いだろうか？図 2において、S2 に流されたメッセージ m1 が S3 に流されたメッセージ m2 よりも先に届くことをプログラムが要求しているとする。S2 が S3 からの WRU を受けとったとき、S2 が S3 に HERE を送って S3 から O への直接ポインタができたならば、この経路によって流される m2 が m1 より先に着かないことは保証できない。したがって、S2 は S3 に HERE を返してはならず、S3 からの WRU をそのまま O に流さなければならない。なお、プログラムが m1 と m2 の順序を指定していなければ、当然 S2 は S3 に HERE を返してよい。

5.6 第一メッセージと WRU との融合

すでにキューの中にメッセージを持っているストリームをデリファレンスする場合、先頭のメッセージを WRU と共に下流に流す。これにより、各ストリームに送られた最初のメッセージは、デリファレンス処理による遅延を受けずに配送される。また、WRU と共に流すことによって、先頭のメッセージの PE 間転送のコストを WRU と共有することができる。

6 評価

本論文で述べた方式を用いる処理系について、簡単なプログラムの処理時間を測定した。比較のため、PE 間デリファレンスを行わず、隣接するストリームが判明した時点で直ちにメッセージを下流に流す処理系も作成し、これについても同じ測定を行った。

測定には Sequent 社の Symmetry を用いた。Symmetry は共有メモリ型並列計算機であり、測定に用いた処理系は、共有メモリを用いて仮想的に実現されたメッセージ通信機構を用いている。このため、一回のメッセージ通信に要する時間は実際のメッセージ通信型計算機の場合に比べてかなり短い。したがってこの測定は通信コストの差が最も現れにくい環境で行ったことになる。

まず、ストリームの連鎖にメッセージを流したときに、そのメッセージが流れ終るまでの時間を測定した。測定に用いた処理の構成を図 3 に示す。各 PE[2] 間に 2 本ずつ違う向きにストリームを張り、全 PE を環状につなぐ。淡い色のストリームは各 PE に指示を送るために用いる制御用ストリームであり、濃い色の方が測定対象の連鎖を構成するストリームである。

淡い色の制御用ストリームにメッセージ connect を流すと、これを受けとった各 PE は濃い色の測定対象のストリームを接続する。図 3 は connect が PE#1 を経由して PE#2 まで到達したところを示しており、PE#1 と PE#2 では、濃い色のストリームが接続されている。

以上の構成において、PE#0 からメッセージ connect を制御用ストリームに流すと同時に、濃い色のストリームにメッセージ hello を何個か流す。そして、最後に流した hello が再び PE#0 に帰ってくるまでの時間を測定した。また、hello を送らずに濃い色のストリームをいきなり閉鎖した場合に閉鎖が PE#0 に伝わるまでの時間も測定した。実際の測定では、10 台の PE を用い、測定対象のストリームを毎

[2]実際には各 PE に配置したオブジェクト間に張られているが、各 PE にオブジェクトを一つしか置かないので、PE とオブジェクトを同一視している。

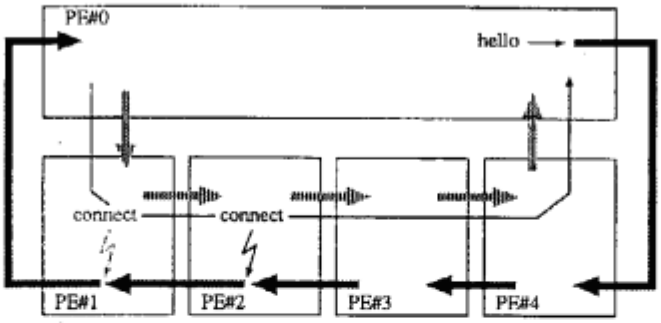

図 3: ストリームの連鎖の形成

回新しく生成しながら繰り返した 100 回の時間の和を求めた。

結果を図 4に示す。縦軸は時間を、横軸は送った hello の数を表している。WRU を使ってデリファレンスを行う処理系では、最初のメッセージの到着が 2 番目以降のメッセージより大幅に早い。また、メッセージを 2 個以上送った場合、WRU によってデリファレンスを行う処理系は、デリファンレスを行わない処理系に比べ、約 200msec 速かった。

二つの処理系の処理時間の違いは、まさにデリファレンスに起因している。図3の場合で言えば、connect が PE#4 に届くまではどちらの処理系でも hello は PE#4 に留まっている。connect が届いた時、WRU を用いる処理系では PE#3 と PE#4 の間の短いやりとりの後、PE#4 のストリームから直接 PE#0 に hello が送られるのに対し、デリファレンスしない処理系では、hello はすべての PE をめぐってから PE#0 に届く。これが 200msec の違いとなって現れている。

デリファレンスを行わない処理系では、connect が流れている間には何も処理を行わない。これに対し、WRU を用いる処理系においては、connect が流れている間に connect が通過した PE において次々にデリファレンスが行われる。このため、PE#4 に connect が着いたころには、PE#3 のデリファレンスはほとんど終っている。そのため PE#3 は PE#4 からの WRU に対して即座に HERE を返すことができ、これを受けとった PE#4 は PE#0 に hello を送ることができる。

次に簡単ながら意味のある処理を行うプログラムについて、両方の処理系による処理時間を比較した。測定に用いたプログラムは generate and test により指定された上限までの範囲の素数を求めるものである。このプログラムは上限までの整数の生成器、ある素数の倍数を除外するフィルタ、および結果の出力を行うもの、の 3 種のオブジェクトで構成される。

測定結果を図 5に示す。WRU を用いる処理系は、用いないものに比べて 1 割処理時間が短い。共有メ

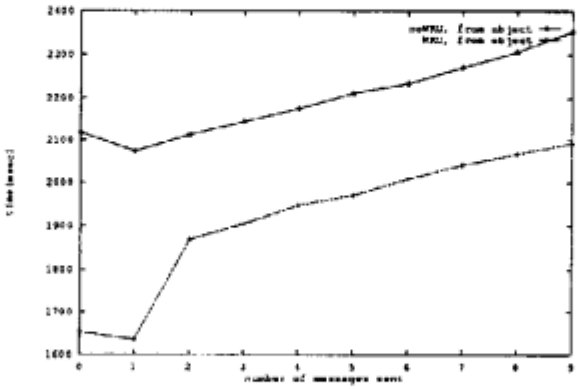

図 4: ストリームの連鎖による通信の処理時間

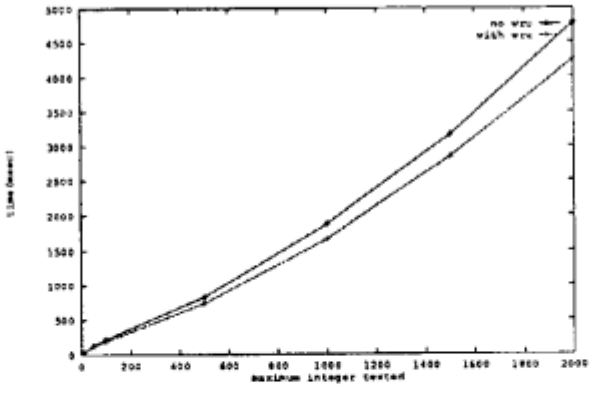

図 5: 素数生成プログラムの処理時間

モリ上の仮想メッセージ通信機構の通信が高速であることを考慮すると、全処理時間の 1 割の改善は大きいと言える。これは主に発見された素数をフィルタから出力オブジェクトに送る通信経路での通信処理時間が短縮された結果である。この経路は常に最も新しいフィルタが保持しており、新しいフィルタができる度にそのフィルタに渡される。このため、この経路は各フィルタの置かれた PE をめぐるストリームの長い連鎖になっている。処理時間の違いは、WRU を用いない処理系ではメッセージがストリームの連鎖に沿って何度も転送されるのに対し、WRU を用いる処理系ではメッセージがフィルタの PE から出力オブジェクトの PE に直接転送されることによる。

7 おわりに

ストリーム通信の実装方式の評価について述べ、WRU によるデリファレンスがストリームの長い連鎖に対して有効であり、処理時間を短縮できることを示した。

ストリーム通信の形態には、ストリームの連鎖の長さ、および連鎖の形成順序、通信経路当たりのメッセージの流量などによって様々なものがありうる。したがって、多様な形態における WRU 方式の特性を調べると共に、実用的プログラムではどのような形のストリーム通信がどの位の割合で用いられているか、その割合の元で WRU 方式が有効かどうかを更に明らかにする必要がある。

謝辞

日本電気技術情報システム開発（株）の柳田伸二氏と丹下利雄氏の、A'UM-90 の開発への尽力に深く感謝します。

参考文献

[1] E. Shapiro, A. Takeuchi, *Object-oriented Programming in Concurrent Prolog*, New Generation Computing, 1, 1983.

[2] K. Yoshida and T. Chikayama, "A'UM – A Stream-Based Object-Oriented Language –," Proc. Int'l Conf. on Fifth Generation Computer Systems (FGCS '88), pp. 638-649, ICOT, November 1988.

[3] 小西、丸山、小長谷、吉田、近山、「並列オブジェクト指向言語 A'UM-90 」, JSPP'90 論文集, 1990 年 5 月

[4] 小西、丸山、小長谷、吉田、近山、「A'UM-90 のストリーム通信の分散実装方式」, JSPP'91 論文集, 1991 年 5 月