TR-643

A'UM-90 のストリーム通信の分散実装方式

小西　弘一、丸山　勉、
小長谷　明彦（日電）、
吉田　かおる、近山　隆

May, 1991

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

A'UM-90 のストリーム通信の分散実装方式

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
†(財) 新世代コンピュータ技術開発機構

1991 年 4 月 11 日

概要

A'UM は、ストリーム通信に基づく並列オブジェクト指向言語である。ストリームは、メッセージ間の順序を保つ通信経路であり、これを用いると多対一通信におけるメッセージ到着順序を表現することができる。本稿では、A'UM-90 言語処理系における、ストリーム通信機能の分散環境への実装方法を述べ、小さなオーバーヘッドでプロセッサ間の通信を最短経路で行えることを示す。本稿にはノウハウに関する記述は含まれていない。

A'UM-90 のストリーム通信の分散実装方式

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　†(財) 新世代コンピュータ技術開発機構

概要

A'UM は、ストリーム通信に基づく並列オブジェクト指向言語である。ストリームは、メッセージ間の順序を保つ通信経路であり、これを用いると多対一通信におけるメッセージ到着順序を表現することができる。本稿では、A'UM-90 言語処理系における、ストリーム通信機能の分散環境への実装方法を述べ、小さなオーバーヘッドでプロセッサ間の通信を最短経路で行えることを示す。

1　はじめに

複数の独立性の高いモジュールが他のモジュールと通信しながら並列に処理を進めるモデルは、現実の多くの社会組織やシステムに当てはまる、普遍的なモデルである。並列処理の可能性と需要が増大するにつれ、このようなモデルに対する関心は高まり、現在、この分野は最も盛んな研究が行われる領域の一つとなっている。並列オブジェクト指向モデルは、この種のモデルの、かなり広い部分に対する名称である。

多くの並列オブジェクト指向言語のモデルでは、原則としてオブジェクトの処理はメッセージによって選択され、起動される。この機構は、処理の進行を直接左右するため、ここに、より柔軟な制御を持ち込もうとする試みが盛んに行なわれている。その中の目だった流れの一つとして、オブジェクトの外部に特別な機構を設けて、その機構にそのオブジェクトに送られたメッセージが到着する順序を制御させるものがある [2][5]。

メッセージ到着順序を制御する機構の一つと見なし得るものに、KL1 (GHC), Fleng などの Commited-choice 並列論理型言語でプロセス間通信に用いられるストリームがある。ストリームは、通信の経路のモデルであり、同一経路で送られるメッセージ間の順序を表すものである。これを利用して、経路を明示して通信を記述することにより、様々な到着順序制御を簡潔に分かりやすくプログラムできる。

そこで、ストリーム通信をオブジェクト指向モデルに取り入れることにより、メッセージ到着順序の制御の簡潔な表現を可能にしようとするプログラミング言語が A'UM[1, 3] である。この言語はまた、並列処理に重点を置いた構文を持っており、素直な記述によってアルゴリズム中の全ての並列性を表現できる。

プログラム中にメッセージ到着順序を簡潔に表現する手段を与えることが言語の責任であるのに対して、表現されたメッセージ到着順序制御を正しく、そして効率良く実現することは、言語処理系の責任である。これを目標として、我々は、MIMD 型並列計算機や LAN で結ばれた複数の WS などの実行環境を想定した A'UM 処理系 A'UM-90 を開発している。本稿では、A'UM-90 におけるストリーム通信機能の実装方法について述べ、ストリーム通信として表されたメッセージの順序を小さなオーバーヘッドで実現できることを示す。

小さなオーバーヘッドでプロセッサ間に渡るストリーム通信を最短経路で行えることを示す。まず 2 節で、A'UM の概要を、3 節でストリーム通信を説明し、その後 4 節で、ストリーム通信の実装方式について説明する。

2　A'UM 概要

A'UM の特徴は以下の 3 つに要約できる。

- 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

Distributed Implementation of Stream Communication in A'UM-90
Koichi Konishi, Tsutomu Maruyama, Akihiko Konagaya, Kaoru Yoshida†, Takashi Chikayama†
C&C Systems Research Laboratories, NEC Corporation
†Institute for New Generation Computer Technology

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。しかし、アクションの実行順序を記述する構文はない。したがって、基本的にはすべての処理が並列に動作し、必要な処理の実行順序は、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、次節で詳しく説明する。

3 ストリーム通信の特徴

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や、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。A'UM では、あるストリームへの参照がなくなる時に、自動的にそのストリームを閉鎖するので、参照の消失自体を閉鎖と見なすことができる。

ストリームの閉鎖は、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。また後述する、ストリームの順序つきの合流に閉鎖が伝わると、合流先に流れ込むストリームが切替えられる。

ストリーム通信の主な特徴は以下の二つである。

- 多対一の通信におけるメッセージ間の順序を表現できる。
- 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。

以下、この二つについて順に解説する。

3.1 メッセージ間の順序

低レベルの通信では、後から送ったメッセージが先に送ったメッセージを追い越して先に到着することがあり得る。このモデルによる並列処理の記述は困難なので、通常、並列オブジェクト指向モデルでは、一対一の通信において、メッセージの追い越しがないことを仮定している。

しかし、多対一の通信についてもメッセージの到着順序を指定できると便利な場合がある。例として、「振込人が 50 ドルを口座に振り込み、払出人に口座番号を通知する。払出人は通知を受けると、口座から 50 ドルを引き出す」という処理を考える（図 1）。

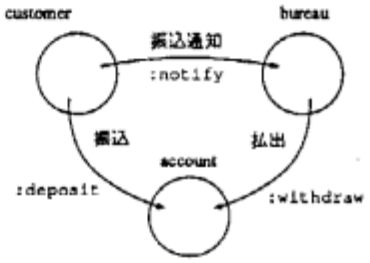

図 1: 口座振込の例

この処理を、通常の非同期通信でそのまま書くと、口座への振込メッセージと、払出メッセージの到着順序が保証できないため、振込前に引き出しが行われてしまう恐れがある。正しい順序を保証するためには、振込人が口座への振込が行なわれたことを確認してから払出人に通知する（図 2）か、口座に払出人への通知を依頼する（図 3）必要がある。現実の世界でもこのどちらかの手段が取られているが、そのことは必ずしも意識されていない。また、通常の非同期通信による順序制御では、払出人と口座のそれぞれが行う処理の間の並列性が表現されていない。例えば、払出人は通知を受けたとき、払出以外に例えば商品の手配を始めるものとすると、図 2では、口座から振込人に確認が帰るまで、図 3でも口座に振込メッセージが届くまでは絶対に手配が始まらない。

A'UM のストリーム通信では、図 1の考え方に沿って、振込人が口座につながるストリームに振込メッセージを送った後、そのストリームを通知メッ

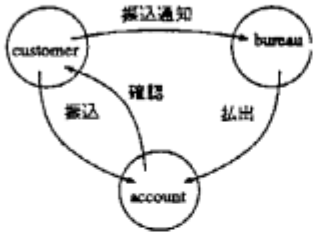

図 2: 振込確認後の通知

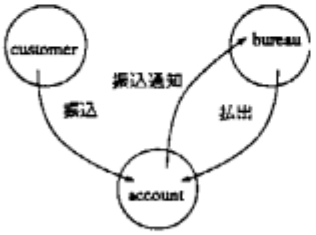

図 3: 口座からの通知

セージによって払出人に渡し、そこに払出人が払出メッセージを送る、という、以下のような記述ができる。

```
class customer.
:transfer(^Bureau, ^Account) ->
    Bureau :notify(Acc),
    Account :deposit(50) = ^Acc.
end.
```

これは振込人のクラス customer の定義であり、メソッド transfer が定義されている。このメソッドは、引数として、それぞれ払出人と口座につながるストリーム Bureau と Account を受けとり、Bureau には通知メッセージ:notify を、 Account には振込メッセージ :deposit を送る。:notify の引数 Acc もストリームであり、これは、:deposit を受けとった Account の後ろに接続されている。払出人は、:notify を受けると、その引数である Acc に対して払出メッセージを送るので、その結果ストリーム Account における順序として、:withdraw と :deposit の到着順序を表現できる。

また、振込人から払出人への通知メッセージは、振込メッセージの到着も、それに対する確認も待たずに送られるので、通知を受けた払出人が商品の手配をする場合、払出人は口座が行う振込処理と並行して手配を行うことができる。

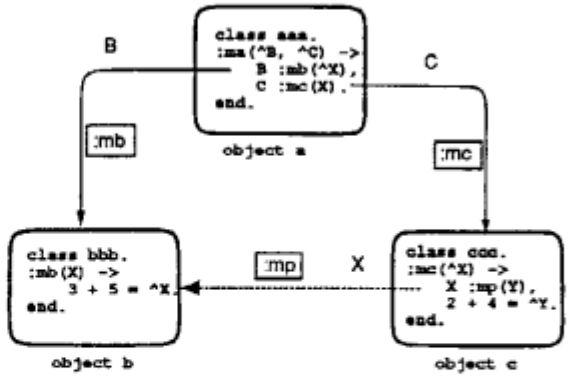

図 4: 未接続ストリームの送信

なお、付録には、メッセージ順序を表現する構文の一覧を示した。

3.2 接続先が未定の引数を持つメッセージの送信

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。

図4は、クラス aaa のオブジェクト a, bbb のオブジェクト b、そして ccc のオブジェクト c の間でストリーム通信が行われる様子を示している。a にメッセージ :ma を送って、b につながるストリーム B と、c につながるストリーム C を渡すと、a はストリーム X を作り、B にメッセージ:mb(^X) を、C にメッセージ :mc(X) を送る。

オブジェクト b はメソッド :mb において、値 p (図4では 3 + 5) を求め、これに X を接続する。一方、オブジェクト c はメソッド :mc において、ストリーム X にメッセージ:mp(Y) を送り、また、値 q (図4では 2 + 4) を計算して、これに Y を接続する。

オブジェクト b における値 p の計算と、オブジェクト c における値 q の計算は並行して行われ、p が求められてこれに X が接続されると、メッセージ:mp(Y) が p に届き、メソッド :mp が起動される (図4では 8 :mp(6) という処理が行われる)。

上述の処理における p と q の計算の並列性を、通常の非同期通信で記述しようとすると、オブジェクト b から c に値 p をメッセージで送ったり、このメッセージが起動するメソッドが値 q にアクセス

できるように、qをcの内部状態に置いたりする処理の明示的な記述が必要になる。このため、記述量は増え、pとqの関係は分かりにくいものになる。

future オブジェクトを返す非同期通信では、future オブジェクトへのアクセスがそれを作成したオブジェクトに限られていなければ、ストリーム通信の場合とほぼ同じ形で先の並列性を表現できる。そうでなければ、値pとqの両方をaに作った future オブジェクトに受けとるようにする、などの処理の形の変更が必要になったり、あるいは上述の非同期通信の場合と同様に、別のメッセージと、内部状態を経由する値渡しの記述が必要になったりする。

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、メッセージで通信する複数のプロセッサから構成されるシステムへのストリーム通信の実装方式について説明する。

4 ストリーム通信の実装方式

ストリームの実装における目標は、次の二つである。

- ストリームとしての機能の保証
- 上記保証範囲内での、最高の処理効率の実現

実装の対象とするシステムが、メッセージによって通信を行うマルチプロセッサシステムである場合、機能の保証に関して特に問題になるのは、異なるプロセッサから送られるメッセージの到着順序の制御である。また、その場合の処理効率は、主にプロセッサ間通信の回数によって計られることになる。

実現するべきストリーム通信の機能は大きく二つに分けられる。一つは、最終目的地が定まるまでの間、ストリームに送られたメッセージと、それらの間の順序関係を保持しておくことである。もう一つは、最終目的地が定まってから、そこにメッセージを保持していた順序にしたがって届けることである。以下では、最初にメッセージ順序の保持に必要なデータ構造を、次に、メッセージの配送手順について説明する。

4.1 メッセージおよび順序情報の保持

ストリームに送られる複数のメッセージとそれらの順序を保持するためには、データ構造の要素として、以下に挙げるものが必要である。

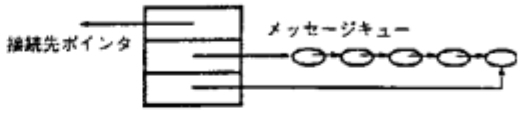

図 5: ストリームを表すノード（M ノード）

[メッセージキュー] メッセージの保持と、局所的な順序の保持に用いる。

[接続先ポインタ] キュー間の順序の保持に用いる。

[合流順序ポインタ] 合流するストリーム間の順序の保持に用いる。

[参照数] ストリームの閉鎖の判定に用いる。

接続先ポインタや、合流順序ポインタは、順序を示すためのものであって、メッセージの配送経路を示すものではない。

上記の構造要素を用いると、ストリーム通信における様々な順序が以下の構造で表現できる。

まず、ストリームを表す最小構造は、メッセージキューと接続先ポインタからなるノード（図5）である。これは、後述する理由により、Merge ノード、略して M ノードと呼ぶことにする。接続先が決まっていないストリームに送られたメッセージは、送られた順にメッセージキューにつながれて溜っていく。

ストリームの接続は、接続先ポインタを後ろのノードから前のノードに張ることによって表せる。この際、同一プロセッサ内での接続であれば、二つのメッセージキューを一つにまとめておくこともできよう。こうして、一般には合流のない一本のストリームは、接続先ポインタでリンクされた M ノードのリストとして表される。

順序のない合流は、多重参照された M ノード（図6）によって表せばよい。ただし、合流先のストリームの閉鎖、すなわちストリームへの参照がなくなることを調べるには、参照数を M ノードに持たせる必要がある。参照数は動的に増減することがあるので、結局、すべての M ノードに参照数を持たせる。Merge ノードをこの名前で呼ぶことにしたのは、このノードが一般には順序のない合流を表すことからである。

順序のある合流の表現は、もう少し複雑になる。合流する複数のストリームを表す M ノードに合流順序ポインタを持たせて、順序の早いものが遅いものを指すようにする（図7）。この合流順序ポインタつき M ノードは Append ノード、略して A ノードと呼んでいる。

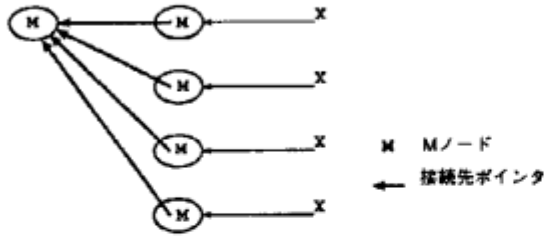

図 6: 順序なしで合流するストリームを表す構造

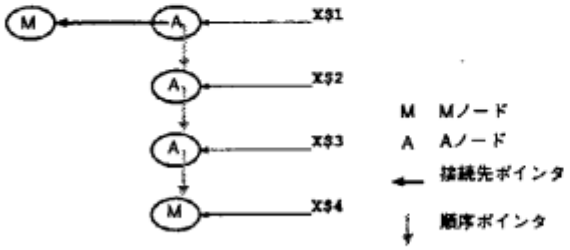

図 7: 順序つきで合流するストリームを表す構造

合流順序が最も早いAノードには、接続先ポインタを持たせるが、それより遅いAノードの接続先ポインタは未定にしておく。先頭のAノードが閉鎖されたならば、そのAノードが合流順序ポインタで指しているAノードに、それまでの先頭が持っていた接続先ポインタの値を与える。こうして、順序のある合流は、合流順序ポインタによってリンクされたノードのリストと、1本の接続先ポインタによって表すことができる。

以上、ストリーム、すなわちメッセージの順序の実装上の表現について述べた。次節では、上記の構造がオブジェクトに接続された時に、そのオブジェクトにメッセージを配送する手順について説明する。

4.2 メッセージの配送

メッセージのプロセッサ間転送回数を減らすためには、通常メッセージは、各ノードから最終目的地に1回のプロセス間転送で直接送りたい。このため、二つの制御メッセージを使って、接続先のデリファレンスを行う。また、このとき使われる制御メッセージを使って、ストリームの参照数管理も同時に行う。さらに、順序つき合流がある場合には、Aノードを最終目的地が判明し次第、最終目的地のあるプロセッサに移住させ、そこに宛ててメッセージを送らせる。

以下、デリファレンス、参照数管理、Aノードの移住の順で説明する。なお、以下の説明において、通常メッセージはA'UMプログラム上に現れる、ストリーム通信によって送られるメッセージを意味する。また、制御メッセージは、通常メッセージの到着順序を制御するためにA'UM処理系が用いるメッセージを意味する。処理系が用いるプロセッサ間通信機能に関しては、一対一通信においてメッセージは送信された順序で到着することを仮定する。また、すべてのノードは異なるプロセッサにあるものとする。

4.2.1 デリファレンス

デリファレンスには、以下の制御メッセージを用いる。

[Where_are_you] Mノードが最終目的地のアドレスを尋ねるメッセージ。

[I_am_here] 最終目的地またはAノードがアドレスを通知するメッセージ。各ノードからのメッセージの転送許可の通知も兼ねている。

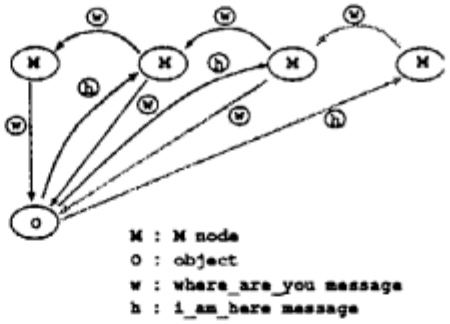

図 8: デリファレンス

プロセス間に渡るストリームのデリファレンスは、以下の手順によって行う。

1. ノードの接続の際に、後ろのノードのアドレスを持つ Where_are_you を前のノードに送る。このメッセージは前のノードが最終目的地を知っていればそこに転送され、知らなければ前のノードのキューに入れられる。
2. Where_are_you を受けとった最終目的地では、Where_are_you が持っているアドレスに対して I_am_here メッセージを送り、最終目的地のアドレスを通知する。
3. I_am_here を受けとったノードは、最終目的地に向けて接続先ポインタを張り、キューに

ある全てのメッセージを最終目的地に送る。このとき、一つ後ろのMノードから送られたWhere_are_youなどの制御メッセージも、一緒に最終目的地に送られる。

メッセージの到着順序は以下のようにして保証される。あるノードに送られたメッセージは、そのノードから接続先に送られたWhere_are_youが最終目的地に着いてI_am_hereが帰ってくるまでは最終目的地には送られない。Where_are_youは接続先のキューの最後尾に入れられるから、それ以前に接続先のキューにあった、すべてのメッセージが最終目的地に着くまで、その後ろのノードのメッセージの転送は起きない。

この手順では、異なるプロセッサにあるノードごとに3回の制御メッセージ（Where_are_youが2回、I_am_hereが1回）のプロセッサ間転送が行われる（図8）。通常メッセージは、一つにつき1回のプロセッサ間転送で配送される。

4.2.2 参照数管理

A'UMでは、あるストリームへの参照がなくなるという事象が、閉鎖としてモデルに組み込まれており、メソッドを起動したり、順序つき合流において合流先に流れ込むストリームの切替えのきっかけになったりする。したがって、A'UM処理系において参照数管理は必須の処理である。

プロセッサ内部で閉じたポインタに関する参照数管理は容易なので、ここではプロセッサ間に渡るポインタの参照数管理について述べる。この場合の参照数管理は次の手順によって行う。

1. 新たな参照が必要な時には、必要とする場所のアドレスを持つWhere_are_youメッセージを使って要求する。
2. Where_are_youを受けとった最終目的地は、参照数を増やしてから、そのアドレスをI_am_hereによって渡す。
3. 参照数の減少はCloseメッセージによって通知する。

参照数管理で問題になるのは、複数のプロセッサ間で参照の輸出入を行う場合である。別のプロセッサのノードへの参照を勝手に増やして他人に渡してから、ノードがあるプロセッサに+1を通知すると、別のプロセッサから送られる−1との間で競合が起こり、場合によっては参照が残っているにも関わらず、参照数が0になってしまう（図9）。

メッセージの競合を防ぐためには、

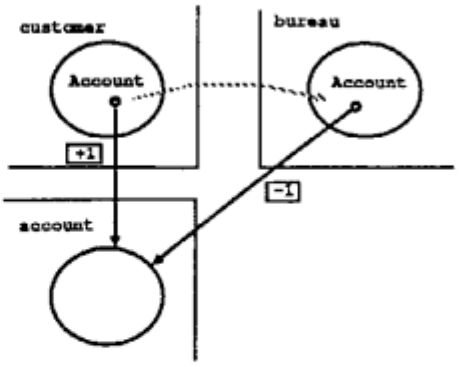

図9: メッセージの競合

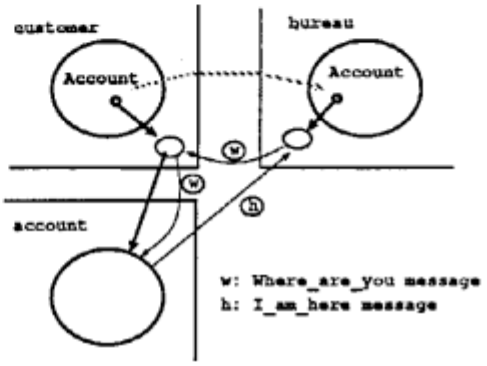

図10: 安全な参照数制御

Where_are_youによって参照の増加を要求し、それに応じて、被参照ノードが自ら参照数を増やした後にI_am_hereで参照を渡す、という手順（図10）を取らなければならない。

実際には、A'UM-90では良く知られた重みつき参照カウントの手法を用いている。これは、あらかじめ複数の参照カウントをまとめて渡しておくことによって、それがなくなるまでは被参照ノードに参照数の増加を通知する必要がないようにする手法である。本節で述べた手順は、参照数の貯蓄を使い切った場合にのみ必要になる。

4.2.3 Aノードの移住

合流するストリームが閉鎖されたときには、順番を待っている次のストリームを接続する必要がある。そのためには、ストリームの閉鎖がAノードに通知されなければならないが、デリファレンスを行うと、ストリームの閉鎖を通知するメッセージCloseが最終目的地に直接到達するので、このままではAノードには閉鎖が通知されない。

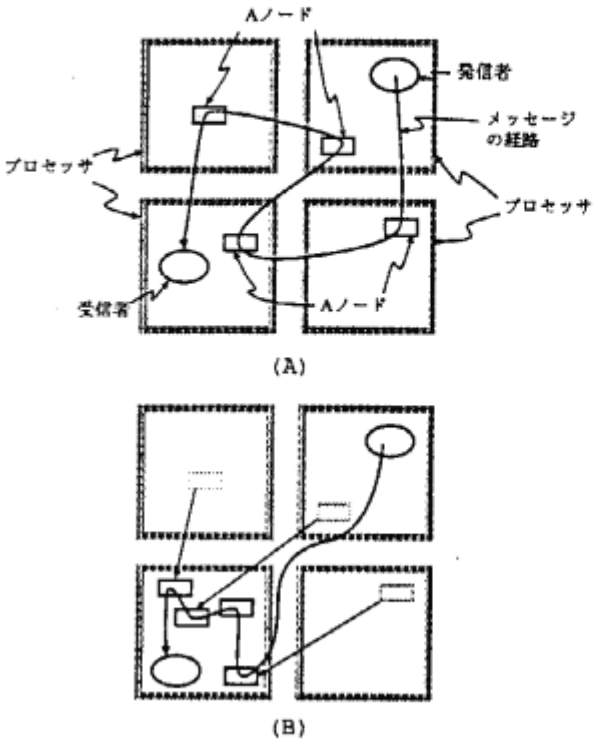

図 11: A ノードの移住

Close が A ノードを通るようにするため、デリファレンスの際に A ノードをスキップしない、すなわち A ノードに Where_are_you が送られたときには A ノードのアドレスを I_am_here で返すことにすると、他のすべてのメッセージも A ノードに送られる（図 11(A)）。これでは、通常メッセージが最終目的地に届くまでに複数回のプロセッサ間転送が起きる恐れがある。

順序つきの合流を効率よく実現するためには、最終目的地が判明してから、そこにつながったストリーム中のすべての A ノードを最終目的地のプロセッサに移住させた上で（図 11(B)）、A ノードをスキップしないデリファレンスを行う必要がある。

A ノードを移住させるため、ここでは、次に示す制御メッセージを用いる。

[Where_are_you_from_Anode] A ノードが最終目的地のアドレスを尋ねるメッセージ。A ノードのアドレスを持つ。

[Close_from_Anode] A ノードの閉鎖を接続先に知らせるメッセージ。閉鎖された A ノードの合流順序ポインタを持つ。

これらを用いると、A ノードを移住させる手順は次のようになる。

1. A ノードを他のノードに接続する際に、Where_are_you_from_Anode を接続先に送る。
2. 最終目的地が Where_are_you_from_Anode を受けとったならば、同じプロセッサ内に、その最終目的地を接続先とする A ノードを作り、I_am_here によって、最終目的地のアドレスの代わりに新しい A ノードのアドレスを元の A ノードに通知する。
3. 元の A ノードが I_am_here を受けとったら、その接続先ポインタを新しい A ノードに張る。
4. 元の A ノードが閉鎖されたら、新しい A ノードに Close_from_Anode を送る。
5. 新しい A ノードが Close_from_Anode を受けとったならば、その中のアドレスを自分の合流順序ポインタとして持つ。
6. 新しい A ノードが閉鎖されたら、その A ノードのアドレスを、合流順序ポインタが指すノードに I_am_here で通知する。

なお、新しい A ノードは制御メッセージに対しては、最終目的地として振舞う。すなわち、Where_are_you に対しては自分のアドレスを I_am_here で通知し、また Where_are_you_from_Anode に対しては、新しい A ノードを作る。

この手順では、合流順序が先頭の A ノードについて Where_are_you_from_Anode, I_am_here の 2 個のメッセージが必要になり、このうち Where_are_you_from_Anode は 2 回の転送を必要とする場合がある。それに続くノードは、一つについて Close_from_Anode, I_am_here が 1 回ずつ転送される。したがって、n 本のストリームの順序つき合流では、最大 2n +1 回制御メッセージのプロセッサ間転送が行われる。通常メッセージは、すべて 1 回のプロセッサ間転送で最終目的地に到着する。

5 まとめ

以上、並列オブジェクト指向言語 A'UM のストリーム通信を、複数のプロセッサ間に渡る環境に実装する方法を述べ、少数の制御メッセージにより、ストリームが表すメッセージ間の順序通りに、通常メッセージを最適な経路で配送できることを示した。

現在、本稿で述べた方法を用いる A'UM-90 処理系の開発を進めており、Where_are_you, I_am_here, Close の 3 つのメッセージについては、基本動作を確認している。引続き A ノードの移住に関する部分についても開発を進める予定である。

今後の課題としては、必要な制御メッセージの個数の最適化、そのためのコンパイラの役割の解明、およびある程度の規模のプログラムでの性能評価がある。

謝辞

A'UM-90 処理系の開発に参加してくださっている、日本電気技術情報システム開発（株）の柳田氏、丹下氏に感謝致します。

参考文献

[1] 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.

[2] N. Carriero and D. Gelernter, "Linda in Context," pp444-458, CACM, Vol.32, No.4, 1989.

[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] 丸山、小西、小長谷、吉田、近山、「ストリームに基づく並列オブジェクト指向言語 A'UM-90 - ストリーム分散実装方式」、情処学会第 4 1 回全国大会講演論文集、2E-2, 1990 年 9 月

[5] S. Watanabe, Y. Harada, K. Mitani, E. Miyamoto, "Kamui88: A Parallel Computation Model with Fields and Events," pp153-175, Advances in Software Science and Technology, Vol.2, 1990, JSSST.

付録: メッセージ順序の表現

[送信] ストリーム Foo に メッセージ :abc と:def をこの順序で送信する、という記述は次の式になる。

```
Foo :abc :def
```

この式は、(Foo :abc) :def と解釈され、式 Foo :abc の値は、:abc を受けとったあとの Foo である。

[接続] ストリーム Foo と Bar の接続は、次の式で表される。

```
Foo = ^Bar
```

^ はストリームの先頭にアクセスしていることを示す。次に示す式では、Foo と Bar は一本のストリームになって、3 つのメッセージの間には、 :abc :def :ghi という順序ができることを表している。

```
Foo :abc :def,
Bar :ghi,
Foo = ^Bar
```

[順序のない合流] 順序をつけずに合流する場合、メッセージはそれぞれのストリーム内での順序を保ったまま、他のストリームと非決定的な順序で合流する。ストリーム Bar と Gnu が順序のない合流をして、Foo に流れ込むことは、次の式で表せる。

```
Foo = ^Bar = ^Gnu
```

この式は、(Foo = ^Bar) = ^Gnu と解釈され、式 Foo = ^Bar の値は、Bar と合流して Foo に流れ込むストリームである。メッセージを Bar と Gnu に送ると、それらは Foo に非決定的な順序で流れ込む。これと同じことは、次の式でも表せる。

```
Foo :abc, Foo :def
```

つまり、同じストリームを参照する名前が二つ以上現れるとき、それらは順序なしに合流する複数のストリームを表す。

[順序のある合流] ストリームを順序をつけて合流させると、先になったストリームが閉鎖されるまで、後のストリームのメッセージは合流先に流れない。ストリーム Bar と Gnu がこの順序で合流して Foo に流れ込むことは、次の式で表せる。

```
Foo \ ^Bar = ^Gnu
```

この式は (Foo \ ^Bar) = ^Gnu と解釈され、式 Foo \ ^Bar の値は、Bar と合流する、Bar より順序が後のストリームである。Bar と Gnu にそれぞれ:abc と:def を送ると:abc は必ず:def より先に Foo に流れ込む。これと同じことは、次の式でも表せる。

```
Foo$1 :abc, Foo$2 :def
```