

TM-1140

A'UM-90 並列版処理系 概要

大西 弘一、柳田 伸二、丹下 利雄、
丸山 勉、近山 隆 (日電)

December, 1991

© 1991, ICOT

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

並列版 A'UM-90 処理系概要

第 1 章

はじめに

オブジェクト指向言語では、オブジェクトの処理はメッセージを受信することによって選択され起動される。従って、オブジェクトの実行制御を行なうためには、メッセージの到着順の制御を行なうことが必要となる。逐次的な環境下では、メッセージの到着順の制御は容易であるが、並列環境下では各オブジェクトが並列に動作するために、複数のオブジェクトからあるオブジェクトにメッセージが送信された場合等には、それらのメッセージの到着順を制御するために何らかの機構が必要となる。この機構は、言語の記述性及び処理性能に直接影響するため、様々な試みが行なわれている。

A'Um[1][2] は、オブジェクト間の通信路をストリームとして捉え、ストリームの接続を制御することによって、この機構を実現している。ストリームは自由にストリームおよびオブジェクトと接続することが可能である。接続されるとストリーム中のメッセージは順序を保ったまま接続先に転送される。ストリームを動的に生成、接続することによってオブジェクト間のストリームネットワークを容易に構築することができ、これを用いてオブジェクトの実行制御を効率的に行なうことができる。

これまでに A'Um の並列版処理系 (PAS) が完成した。実装においては、高並列な処理を実現するために共有メモリの存在は仮定しない方式をとっている。A'Um 並列版処理系は、現在市販の共有メモリ型並列マシン上で動作しており、通信の局所性等を全く利用せずに全ての通信がプロセッサ間通信となるような負荷分散をとった場合においても、通信の局所性を利用できる 1 プロセッサの場合と較べて、15 プロセッサで約 8~10 倍の性能向上が見られストリーム通信の有効性が確認された [5]。

以下、A'Um の概要および A'Um 並列版処理系 (PAS) の概要について述べる。

第 2 章

A'Um 概要

A'Um の主な特徴は以下の 3 点である。

- 並列オブジェクト指向
- 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。各オブジェクトは原則として並行に動作する。

更に、A'Um にはメソッド中の各アクションに実行順序を記述する構文はなく、各アクションは並行に実行することが可能である。各アクション間での実行順序は、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のみによって決まる (但し、後で述べるように PAS においては、各アクションは実際には並列に実行されない)。

以下、A'Um の最も重要な特徴であるストリームの基本操作について述べる。

2.1 ストリーム通信

ストリームはメッセージキューとしての機能を持つ。ストリームは生成時にはどこにも接続されていない。複数のストリームを接続、合流させることにより、オブジェクト間のネットワークを自由に作ることが可能である。このようにしてネットワークを動的に作ることによって、各オブジェクトへのメッセージの到着順を容易に制御することができる。

以下ストリームの基本機能について述べる。

ストリームの生成

ストリームは動的に生成することができる。ストリームには、頭部 (inlet) と尾部 (outlet) がある。ストリームに対するメッセージ送信は、outlet に対して行なわれる。ストリームが生成された時点ではそのストリームはどこにも接続されていない。このような未接続のストリームに対してもメッセージ送信を行なうことが可能である。未接続のストリームに対して送信されたメッセージは、ストリーム中に留まる。

ストリームの接続

ストリームの inlet は、他のストリームの outlet およびオブジェクトと接続することができる。ストリームが接続されると、そのストリーム中に蓄えられていたメッセージは接続先に送信される。このとき、各ストリーム内でのメッセージ順序が保証されるばかりでなく、ストリーム

```

class class_name.      % class 定義
  int inlet_slots.... % inlet slot の宣言
  out outlet_slots... % outlet slot の宣言
:method(args...) ->   % method 名と引数
  actions...          % method 内処理
...
end.                  % class 定義の終り

```

図 2.1: A'Um プログラムの概略

間での順序も保証される。例えば、ストリーム A の outlet とストリーム B の inlet が接続された場合には、ストリーム B 中のメッセージは、ストリーム A 中のメッセージがオブジェクトに到着した後に到着することが保証される。

ストリームの閉鎖

それ以上参照されないストリームは閉鎖することができる (言語処理系ではそれ以上参照されない場合に自動的に閉鎖される)。ストリームの閉鎖は、ストリームの接続先に伝搬される。ストリームの接続先がオブジェクトで、その入力ストリームが全て閉鎖された場合にはオブジェクトは消滅する。

2.2 A'Um プログラム

A'Um プログラムはおおよそ図 2.1 のような形をとる。以下、A'Um プログラムにおける代表的な処理について説明する。

ストリーム

ストリームはプログラム中で、大文字で始まる変数で表される。ストリームの inlet は `^Stream` のように 変数の頭に `^` をつけて表され、outlet は何もつけずに単に `Stream` のように表される。ストリームは自由に生成することができ、プログラム中に新たな変数を書けば、自動的にストリームが生成される。

ストリーム変数のスコープは、メソッド内部のみである。従って、メソッドの実行が終了すると不必要となったストリームは自動的に回収される。プログラム上で特にストリームの回収に関して記述する必要はない。

スロット

スロット変数は、オブジェクトに固有の変数である。各メソッドからアクセスすることが可能であるため、メソッド間での共有変数として用いることができる。スロット変数には、inlet スロットと outlet スロットの 2 種類がある。inlet スロット変数は、ストリームの inlet を保持するのに用いられ、outlet スロット変数は、ストリームの outlet を保持するのに用いられる。

ストリームの接続とスロットへの代入

A'Um プログラムにおいては、演算子 '=' は 2 つの意味を持ち、以下のルールによって、使い分けられている。

1. ストリームの接続

$X = ^Y$ のように、'=' の左辺が outlet であり、右辺が inlet である場合は、ストリームの接続が行なわれる。このとき、メッセージは右辺のストリームから、左辺のストリームへと流れ、左辺のストリーム中のメッセージが、右辺のストリーム中のメッセージより先に到着することが保証される。

2. スロット変数への代入

スロット変数のスコープは、クラス定義内である。従って、メソッド間に跨るデータの共有はスロット変数を用いて行なうことができる。inlet slot は頭に @ をつけて用いられ、outlet slot は ! をつけて用いられる。

(a) '=' の左辺が inlet slot である場合

@inlet_slot = ^X のように '=' の左辺が inlet slot である場合には、右辺の inlet が inlet slot に代入される。

(b) '=' の右辺が outlet slot である場合

X = !outlet_slot のように '=' の右辺が outlet slot である場合には、左辺の outlet が outlet slot に代入される。

以上のストリームの接続と代入においては、以下のような記述も可能である。

$X = ^Y = ^Z$

これは、

$X = ^Y, X = ^Z$

と等価である。このとき、ストリーム Y と Z に蓄えられていたメッセージは、ストリーム X 中のメッセージより後に到着するが、Y と Z 中のメッセージの到着順は保証されない。

オブジェクトの生成と消去

オブジェクトの生成は、#class_name と書くことによって実行される。このとき、#class_name の返却値は生成されたオブジェクトへのストリームである。一般には、

#class_name = ^X

のように用いられ (生成されたオブジェクトとストリーム X の inlet が接続される)、X に送信されたメッセージは、生成されたオブジェクトに到着する。

どのストリームからも参照されなくなったオブジェクトは自動的に回収される。このとき、オブジェクトのスロット変数によって参照されているストリームも必要に応じて自動的に回収される。

メッセージ送信

メッセージ送信は、

Stream :message(args..)

のように書くことによって実現される。送信先は outlet slot であることもある。このメッセージ送信は式の値を持ち、ストリームの outlet を返す。従って、

X :message(args) = ^Stream,

のように書くこともできる。この場合 :message は Stream に対して送られたメッセージより先

に到着することが保証される。また、

```
X :m(args) :n(args) = ^Stream,
```

のような記述も可能である。この時は、メッセージ :m が先に到着し、:n はその後で到着することが保証される。さらに、オブジェクトを生成した場合にも、その返却値はオブジェクトへのストリームとなるので、以下のような記述も可能である。

```
#class_name :message(args...) = ^Stream
```

宛先ストリームを指定しないメッセージ送信は自分自身へのメッセージ送信となる。このメッセージは、他のオブジェクトからのメッセージより優先的に実行される。

条件判断

条件判断は、基本的にはオブジェクトに値を問うメッセージを送信し、その返答を受けるという形をとる。以下に示すのがその記法である。ストリーム X に 値が1であるかを問うメッセージを送信し、その結果がメッセージ名 :true もしくは :false というメッセージによって返答される。

```
(X == 1)? (  
  :true ->  
    ....  
  :false ->  
    ....  
)
```

第 3 章

並列版処理系 (PAS)

並列版処理系 (PAS: Parallel A'Um System) は現在、市販の共有メモリ並列マシン Symmetry 上に実装されている。Symmetry は共有メモリを持つマシンではあるが、PAS の実装においては、非共有メモリ型のより多くのプロセッサを持つマシンでも動作するように、共有メモリを仮定しない実装方式をとっている。現在、メッセージの送受信部にソケットを用いた分散処理系についてもデバッグ中である。以下、PAS について述べる。

3.1 概要

A'Um プログラムはまず、A'Um コンパイラによって抽象機械語にコンパイルされ、次いで並列版処理系 PAS によって実行される。抽象機械語のレベルはかなり高く、A'Um の基本機能であるメッセージ送信、ストリームの接続等とほぼ 1 対 1 に対応している。

A'Um の言語モデルでは、各アクションは並列に実行可能である。従って、あるメソッドの実行中に、他のメソッドの実行を開始することも可能である。しかし、汎用プロセッサベースの並列マシンへの実装を考えた場合には、各アクションのような細粒度の処理を並列に実行しても処理速度の向上は期待できない。このため、並列版処理系 (PAS) における並列処理の単位はオブジェクトであり、各メソッド内の処理は逐次的に行なわれる。また、あるメソッドを実行中に他のメソッドの処理を開始することもない。

オブジェクト内の処理はこのように逐次的ではあるが、A'Um コンパイラは、各メソッド内のアクションの実行順序は任意であることを利用して、処理の最適化を行なっている。

3.2 データ構造

A'Um の基本データ構造は、オブジェクトとメッセージとストリームである。

オブジェクトは、メッセージキュー、スロット変数領域、クラステンプレートへのポインタ、スケジューラ待ち用のポインタ等から構成される。オブジェクトに送られたメッセージは、このキューにつながる。メッセージを有するオブジェクトは、スケジューラの待ち行列につながれ、順番が来るとオブジェクトに接続されていたメッセージが実行される。

メッセージは、オブジェクトのメッセージキューに繋ぐためのポインタ、メッセージ ID および引数領域からなる。

ストリームは、ジョイントとして実現される。ジョイントには 2 種類がある。一つは、ストリームの順序を指定しない併合を表すものであり、マージジョイントと呼ばれる。他の一つは、併合の順番を陽に指定する場合に用いられるものであり、アペンドジョイントと呼ばれる。各ジョイントは、接続先が未確定の場合には、メッセージを蓄えるキューとして実現され、接続先が確定した場合には、単なるポインタとして機能する。マージジョイントの構造を図 3.1 に示す。

Tag	RC
Destination/ first message	
Message counter / last message	

図 3.1: ジョイントの構造

3.3 アドレス管理

A'Um 並列版処理系 (PAS) では、プロセッサ間の共有メモリを仮定していない。アドレス空間はプロセッサ間で異なるため、プロセッサ間でのアドレス管理が必要となる。このために、アドレス輸入表とアドレス輸出表を用いている。

アドレスには、グローバルアドレスとローカルアドレスの2種類がある。ローカルアドレスは、各プロセッサ内のアドレス空間そのものである。ローカルアドレスが他プロセッサに輸出される場合には、そのローカルアドレスは、アドレス輸出表に登録され、グローバルアドレスに変換される。グローバルアドレスは、PE 番号と PE 内でのユニークな番号から構成され、システム内で一意に定まるものである。アドレス輸出表への登録にはハッシュを用いており、同一のローカルアドレスには、必ず同一のグローバルアドレスが割り振られる。

輸入されたグローバルアドレスは、輸入表に登録され、ローカルアドレスに変換される。輸入されたグローバルアドレスを保持するために用いられるデータ構造は、マージジョイントそのものである。輸入表への登録においては、ハッシュを用いており同一のアドレスは、必ず同一のエントリに登録される。

上記のように、輸出表と輸入表を用いて、ローカルアドレスとグローバルアドレスの変換を行なうことによって、プロセッサ間での参照カウント操作のためのメッセージ送信回数を最低限に抑えている。

3.4 ガーベジコレクション

並列環境下でのリアルタイムガーベジコレクションを実現するために、重みつき参照カウント方式を用いている。一般に、参照カウント操作命令としては、参照カウントの加算と減算があるが、加算はストリームの分散実装を行なうための制御メッセージと重複が可能なため、PAS では、減算メッセージのみがある。

3.5 制御メッセージ

既に述べたようにストリームはジョイントと呼ばれるデータ構造を用いて実現されている。プロセッサ間でのストリームの接続等の処理は、相手のストリームまたはオブジェクトのアドレスが即座にわかり、それらを直接操作することが可能であるが、プロセッサ間の処理ではこれらの操作に制御メッセージが必要となる。

このような制御メッセージには、原則として以下のものがある。

- Create
オブジェクト生成要求メッセージ

- WhereAreYou
オブジェクトの位置問い合わせメッセージ
- IamHere
WhereAreYou メッセージに対する返答
- Close
参照カウント減算メッセージ

以下、制御メッセージが用いられる処理について説明する。

オブジェクトの生成

他プロセッサ中にオブジェクトを生成は以下のように行なわれる。

1. 自 PE 中に未接続のジョイントをつくる。
2. 他 PE にオブジェクト生成要求を送る。返答先は上記のジョイントとする。
3. オブジェクト生成要求を受信した PE 中はオブジェクトを作る。
4. オブジェクトのアドレスを引数として IamHere メッセージを返答する。
5. IamHere を受信することによって、接続先が確定する。この間に、このジョイントにメッセージが送信されていれば (このジョイント中に蓄えられている)、これらをオブジェクトに対して送信する。

上記のようにジョイントを一旦生成することによって、オブジェクト生成要求に対する返答を待たずに、即座にそれ以降の処理を行なうことを可能としている。

ストリームの接続

同一 PE 内で、ストリームの接続が行なわれた場合には、ジョイントの連鎖が作られ、ストリームに対してメッセージ送信が行なわれた時などに、必要に応じてこの連鎖のデレファレンスが行なわれる。

しかし、並列環境下でのストリームの実装では、次のような問題点が生じる。異なる PE 上にある複数のジョイントが次々と接続された場合には、接続を行なわれた回数分だけ、PE 間でのメッセージの転送が行なわれてしまい (異なる PE 間でのポインタのデレファレンスはメッセージの到着順等の保証を考えると非常に非効率である)、メッセージの転送回数を不必要に増大させる。また並列環境下ではストリームに対する操作が、異なる PE 間で全く非同期に行なわれるためメッセージの到着順を正しく保証するために何らかの手段が必要となる。

従って、分散環境下でのストリームの実装においては、以下の 2 点を実現することが重要である。

- PE 間でのメッセージ送信回数の最小化
- (参照カウント操作メッセージを含む) 効率的なメッセージ到着順の保証

上記の問題点を解決するために、PE 間にまたがるストリーム操作に対しては、以下の基本手順をとることとする。

1. ストリーム同士の接続が行なわれた場合には、接続先ストリームに WhereAreYou メッセージを送信する。
2. ストリームの接続先は未確定にする。従って、それ以降このストリームに送信されたメッセージはこのストリーム内に保持され、接続先のストリームには送られない。

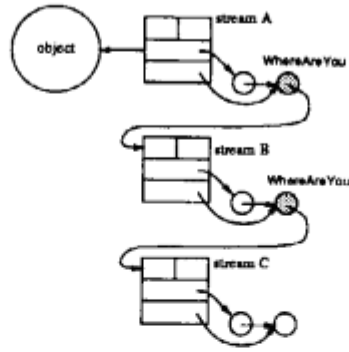


図 3.2: ストリームの接続

3. WhereAreYou メッセージに対する返答である
IamHere メッセージの受信を待つ。
4. WhereAreYou を受信したオブジェクトは自分自身の位置を引数として IamHere を返送する。
5. IamHere を受信したストリームは、接続先を
IamHere の引数とする。
6. 新たな接続先に対してメッセージ送信等を行なう。

この手順の特徴は、一つ先のストリームのみに対して WhereAreYou を送信し、WhereAreYou がそれより先のストリームに転送されないことにある。一つ先のストリーム経由で WhereAreYou を送信することによって、一つ先のストリーム中のメッセージがオブジェクトに受理された後で WhereAreYou がオブジェクトに到着し IamHere が返送されるため、メッセージの正しい到着順を保証できると同時に、一つ先のストリームしか経由しないため、メッセージ転送回数を最小化することができる。

図 3.2 を例にとり、この手順を具体的に説明する。図 3.2 において、まずストリーム B とストリーム C が接続されたとする。この時、C 中のメッセージを B に転送せずに、B に対して WhereAreYou を送信する。この WhereAreYou の引数は C の位置である。次いで、ストリーム A とストリーム B の接続が行なわれると、上記と同様にして WhereAreYou が A に対して送信される。最後に、オブジェクトとストリーム A の接続が行なわれると、A 中のメッセージがオブジェクトに対して送信される。その結果、B から A に送られていた WhereAreYou がオブジェクトに到着し、その返答である IamHere(引数はオブジェクトの位置) が B に対して返信される。IamHere を受信することによって、B の宛先は、直接オブジェクトとなり B 中のメッセージがオブジェクトに対して送信される。以下、同様にして C の宛先もオブジェクトとなり、C 中のメッセージがオブジェクトに対して送信される。

上記の手順は基本的なものであり、それ以降そのストリームに対してメッセージが送信されないことが明確である場合、及びストリームが直接オブジェクトに接続された場合には、メッセージの直接転送等によるメッセージ送信回数の低減を行なうことが可能である。また、メッセージの複合化によるメッセージ送信回数の低減も行なっている。

第 4 章

おわりに

以上、A'Um の並列版処理系の概要とその評価について述べた。ストリームを用いることによって、並列処理の記述が容易になること、またストリームの並列実装方式について述べた。

処理系は、コンパイラを含めて全て C++ で記述されている。共有メモリの存在を仮定しない実装方式をとっており、マシン依存の部分は最小限に抑えられている。今後、処理系の効率化、より高度な負荷分散機能の処理系への組み込み等を行なう予定である。

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] 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
- [3] 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」、情報学会第 41 回全国大会講演論文集、2E-2, 1990 年 9 月
- [4] 小西、丸山、小長谷、吉田、近山、「A'UM-90 のストリームの分散実装方式」, JSPP'91 論文集, 1991 年 5 月
- [5] 丸山、柳田、小西、小長谷、近山、「並列オブジェクト指向言語 A'UM-90 - 並列分散処理方式の評価 -」, SWOPP'91 1991 年 7 月

付録 A

翻訳系プログラム