プログラミング 8－25
（1996. 8. 29）

並列オブジェクト指向言語A-NETLのワークステーションクラスタへの実装

古田 貴寛† 齋藤 宜人† 月川 淳 吉永 努† 馬場 敬信†

本研究は，ネットワークで結合されたワークステーション等のヘテロなネットワークコンピュータを，並列オブジェクト指向概念のもとに統合的に扱うことによって，高性能な並列・分散コンピューティングシステムを使いやすい形でユーザに提供することを目的とする．このため，標準メッセージパッシングライブラリPVMを利用してA-NETLをC言語に変換するトランスレータを試作した．メッセージによる同期，動的型付けをサポートするために，トランスレータでは構造体を用いてタグつきデータ構造を実現しているが，多くの変数は単純なデータ型として利用されることが多く，実行時の型判定がオーバヘッドとなっている．予備実験の結果は，変数型の特定によってプログラムの実行時間が短縮されることを示している．本稿ではトランスレータの実装方法と変数型を特定するための型推論の方法について述べる．

The implementation of a parallel object-oriented language A-NETL on workstation clusters

Takahiro Furuta† Norihito Saitoh† Atsushi Tsukikawa† Tsutomu Yoshinaga† Takanobu Baba†

The objective of this research is to provide a useful parallel/distributed computing system on workstation clusters and high performance parallel machines based on a parallel object-oriented concept. In order to attain this objective, a compiler has been developed to translate an A-NETL, parallel object-oriented program to a C language program with PVM function calls. Preliminary results show the effectiveness of type inference. We are now therefore designing a type inference system for the A-NETL.

1 はじめに

近年，WS/PCの性能の向上は著しく，安価で高い処理性能を得ることができるようになっている．さらに，ネットワークで結合されたそれらのWSにより構成されるWSクラスタ上で，メッセージパッシングによる並列処理を行うためのインターフェイス仕様が提案されている[1, 11]．これらの仕様は，商用並列計算機上にも実装され[6, 5]，これらの並列・分散環境におけるプログラムの可搬性が増している．

我々の研究プロジェクトにおいて，過去10年に渡り並列オブジェクト指向トータルアーキテクチャA-NETの研究開発を進めてきた[2]．この中で，大規模並列プログラムの記述が容易に行え，かつ，マルチコンピュータ上で効率的な実行ができることを目的として，設計されたのが，並列オブジェクト指向言語A-NETL[3]であり，静的多数オブジェクトの定義，宣言型同期機構[4]などの特徴を持つ．ユーザは，問題のもつ並列性を自然な形で抽出し，プログラムを記述することができる．また，このA-NETLを高速に実行するためのA-NETマルチコンピュータが設計・開発されており，現在は16ノードプロトタイプが実動している[9]．

本稿では，アーキテクチャ独立でユーザフレンドリーな並列オブジェクト指向プログラミング環境の提供を目的として，並列・分散環境におけるPVMを用いたA-NETLの実現について述べる．同時に，A-NETLプログラムの実行性能を純粋なC言語プログラムに近付けるために，型推論[8]などによる性能向上の可能性を検討する．

2 A-NETLの実装

2.1 実装の方針と問題点

A-NETLをワークステーションクラスタや商用並列計算機に実装する手段として，C言語に変換するトランスレータを作成した．メッセージパッシングの実現には，PVM，MPIなどの標準メッセージパッシングライブラリを利用する．これによって，これらのライブラリが実装されている多くの並列・

†宇都宮大学工学部情報工学科
Faculty of Engineering, Utsunomiya University

分散環境において，変換したプログラムの実行が可能となる．本実装では現在のところPVMに対応している．

変換に当たっては，A-NETLとC言語との次のようなセマンティックギャップをどう解消するかが問題となる．

(i) データ: A-NETLでは静的にデータ型を固定せず，動的にデータ型に応じた処理を行う．これに対して，C言語では静的に型が固定される．

(ii) メソッド: A-NETLのメソッドは実行を一時中断し，条件成立後，中断した直後の場所から実行を再開できなければならない．

(iii) メッセージ: A-NETLではメッセージ受信処理は明示的に記述されないが，C言語ではユーザプログラム中に記述しなければならない．

以下，PVM上の実装においてこれらの問題にどのように対処したかを，A-NETL専用マルチコンピュータ(以下，専用機)での実装[2, 3]と対比しつつ述べる．

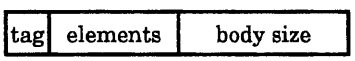
2.2 データの扱い

A-NETLで使用するデータの型は静的には宣言されず，動的型変換も可能である．データ型としては，整数型，浮動小数点数型，文字列型などの基本的なものの他，配列型，リスト型，辞書型が用意されている．配列型，リスト型，辞書型ではこれら全てのデータ型のデータを要素とすることができる．

このようなデータ型を実現するために，専用機の実装では，タグ付きアーキテクチャを採用しており，1ワード40ビットの上位8ビットをタグとして使用している．

本実装では，これをC言語の上に実装するため，タグとデータの2ワードの構造体によって定義している．

図1にタグの構造を示す．



tag: データの種類(4bits)
elements: 配列などの要素の大きさ(12bits)
body size: データの大きさ(16bits)

図1. タグの構造

このタグには，データの種類以外に，データの大きさや，配列，リスト型などではその要素の数も含む．

基本的に全てのデータ型は，このタグとデータ部から成っており，整数，浮動小数点数型，真偽型などでは，データ部に実際のデータが格納される．文字列，配列，リスト，辞書型などの構造をもつデータ型ではデータ部は実データへのポインタとなっている．

なお，専用機実装のようにアーキテクチャの支援がないところで，すべて実行時に型を決定して処理することは大幅に性能を低下させる可能性がある．このため，静的に推論により型を決定し，高速化を図ることが考えられるが，これについては後述する．

2.3 オブジェクトの実装

専用機におけるオブジェクトの実装では，メッセージによる交信を制御するために各ノード上にローカルOSを置き，メッセージの送受信，データのパック・アンパック，コンテクストチェンジなどの処理を行わせている．

本実装では，ローカルOSの行っている処理をユーザオブジェクト自身に付加する．これは，PVMによって実現される仮想並列計算機(以下バーチャルマシンと呼ぶ)において，並列に動作する単位であるタスクが，デーモンを介して直接メッセージのやり取りを行うため，タスク内においてデータのパック・アンパック，コンテクストチェンジなどを行う必要があることによる．

2.3.1 インデックストオブジェクト

インデックストオブジェクトは，インデックス番号を持っていることを除けば通常のオブジェクトと同様に扱える．インデックス番号は，HOSTから渡された各オブジェクトのTIDの情報から算出する．

2.3.2 クラスオブジェクト

専用機の実装では全ノード上に予めクラスオブジェクトがロードされ，ユーザが指定したノード上のクラスオブジェクトに対してnewメッセージを送ることで新たにオブジェクトが生成される．

本実装では，newメッセージを送る代わりにタスクを直接起動することでオブジェクトの動的な生成を実現している．従って，実際にはnewメッセージの送信は行わない．

2.4 メソッド

先に述べたように，メソッド実行の一時停止と再開ができなければならない．専用機実装ではOS以下のシステムがメソッド実行の一時停止時に，実行

のイメージをコンテクストとして保存し，再開時にデータを戻す．

本実装においては，メソッド起動時に予めコンテクストを作成し，それに基づいてメソッド実行を行うという方法を採っている．一時停止時には，リストやキューなどのデータ構造にコンテクストを保存し，実行再開の条件が満たされたならば，コンテクストを取り出してメソッドの実行を再開する．

また，メソッドの途中の位置から実行を再開することを可能とするために，switch 文と goto 文とを用いている．具体的には，現在型のメッセージ送信や未来変数の参照などの，メソッドが一時停止する可能性のある場所にラベルを付け，実行が一時停止した場合は，コンテクストに停止位置の情報を保存する．再開時には，メソッドの冒頭に配置した switch 文によって適切な位置からの実行再開が可能となる．メソッド起動までの手順は [7] と同様である．

2.5 同期機構

基本的な機構は，専用機実装も本実装も同様である [4]．同期機構に関係するメソッドの先頭には，同期のための各機能に応じた条件文が加えられ，条件不成立時にはコンテクストをキューに保存し実行を停止する．

条件不成立の際，専用機ではユーザモードからのトラップにより，ローカル OS に制御が移されて以降の処理が行われるのに対して，本実装ではユーザプログラム内で処理を進める点が異なる．

2.6 メッセージ通信

専用機の実装では機械語レベルで各メッセージ送受信命令が用意されている．

本実装では，メッセージの送受信には PVM の提供するメッセージ送受信関数 (`pvm_send()`, `pvm_recv()`) を利用する．

各型におけるマルチキャストの実装については，現在のところループを用いた単一メッセージの送信の繰り返しによって実現している．

3 タグ付きデータのオーバヘッド

構造体によるタグ付きデータによって動的に型変換が可能になった．しかし，これによって，ユーザが明らかにある特定の型としてのみ変数を使っている場合でも，C 言語においてタグ付きデータになってしまうために，四則演算，メッセージ引数のパック/アンパック，メッセージ通信等においてオーバヘッドが生じる．実際には，A-NETL の変数は特定の型の変数として使われるものが殆んどであるため，トランスレート時に型を何らかの方法により特定できれば，実行性能の向上を図ることができると考える．

型の特定による性能への影響を調べるため，以下に挙げる 3 つの方式に従って，簡単な A-NETL プログラムを C 言語にハンドトランスレートし実行した．

(a) 全ての変数を構造体を用いて定義したもの

(b) 簡単な型推測のルールを作り，それに基づいて可能な範囲で型を特定したもの

(c) メッセージ引数などの推測不可能な変数についても型を特定したもの

(b) の方法で用いた型の推測は，代入文の右辺の型の評価，及びプリミティブメソッドの使用に基づいて特定している．

実験には以下のようなプログラムを用いた．

N-Queens : N×N のチェス盤に N 個の Queen を互いにとることができないように配置するパズル．ここでは N = 8 で実行した．

Life game : 次元格子点上にある生命体が，誕生と死滅を繰り返す，ある種の生体系シミュレーション．

Road map : あるマップに幾つかの目的地が存在して，出発地点から右折禁止で進み，到達可能な目的地を見つける問題．

表 1 がその実行結果である．

表 1. 各方式の平均実行時間 (単位 sec)

	N-Queens	Life game	Road map
(a)	8.24	5.90	2.03
(b)	7.79	4.71	1.97
(c)	7.01	4.32	1.90

表 1 の結果，N-queens では 450$msec$, Life game では 1190msec，Road map では 360msec の速度向上が達成された．

また，並列マシンに対する A-NETL トランスレータにおいて，型推論ができるという仮定のもとに，トランスレータの生成した C プログラムを最適化した結果，実行時間の大幅な改善が期待できるという実験結果も得られている [7]．

4 型推論による最適化

4.1 型推論の概略とデータ構造

実験の結果を踏まえ，型推論を実現することとし，その方法を検討した．

A-NETLには変数として，オブジェクト内でローカルな状態変数と，メソッド内でローカルな一時変数，メッセージ引数の3種類がある．型推論を行うために各変数について図2に示すデータ構造を用意する．型情報にはその変数のとり得る型を記録する．例えばX = 1という代入文があれば，変数Xの型情報として整数が記録される．関係リストはX = Yのような代入文等によってできる変数の依存関係を表すものであり，この場合はYがXの型に影響を与えるため，Yの関係リストにXが加えられる．また，メッセージ送信によるデータの流れを明らかにするために，図3のようなデータ構造をメソッド毎に用意し，型推論に必要なデータを記録する．

型推論は大きく3つの段階に分かれる．

(1) オブジェクト内のデータの流れを明らかにする．

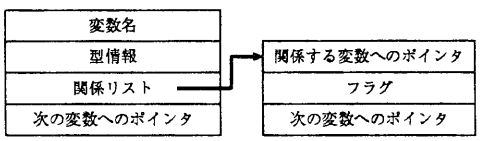
ここでは，定数の代入などによって簡単に型が分かる変数の型情報の記録及び，オブジェクト内の変数の関係リスト作成を行う．また，メッセージ送信に対しては，図3のメッセージ送信情報リストに，送信先オブジェクト名，メッセージセレクタ名，引数として送られる変数，メッセージの返値が格納される変数を記録する．

(2) オブジェクト間のデータの流れを明らかにする．

ここでは，メッセージ送信による引数と返値の受渡しから，異なるオブジェクトの変数間の関係リストを作成する．

(3) (1)と(2)に基づいて変数の型を決定する．

ここでは，型情報として記録されている変数の型を，関係リストに入っている他の変数の型情報に反映させる．



型情報：その変数がとる型を表す
関係リスト：その変数が影響を与える他の変数のリスト

図2. 変数情報

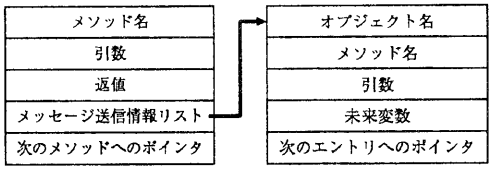


図3. メソッド情報

どのオブジェクトからもメッセージが送られないメソッドはユーザからのが送信が予想されるが，そのようなメッセージの引数の型を推論することは困難である．このような引数はデフォルトではタグ付きのデータ構造をもつ変数とし，ユーザが型を指定した場合にはその型の変数として扱う．

4.2 型推論

ここでは，前述した型推論の3つの処理について具体的に説明する．

(1) 変数情報とメソッド情報の作成

先に述べたように，図2のデータ構造を用いて各オブジェクト内の変数に対して型情報と関係リストの作成を行う．またメッセージ送信に対して図3のデータ構造を用いて引数や返値を記録する．

メソッドの記述としてX = Yのような代入文からXはYの型をとることが分かる．このとき，Yとして記述されている内容に応じ以下のような処理を行う．

(a) Yが定数であればその定数の型をXの型情報に記録する．

(b) Yが変数であればYの関係リストにXを追加する．

(c) Yが他の変数のプリミティブメソッド呼び出しであるとき，それが常に同じ型を返すのならば，Xの型情報に記録する．変数の型に依存して返値の型が違う場合は，その変数の関係リストにXを追加する．直接の代入である(b)とはフラグによって区別する．

(d) Yがメッセージである場合，Xはその返値の型となる．メッセージの返値がどのような型をとるかはこの時点では明らかでないため，オブジェクト名，メッセージセレクタ名，引数，未来変数Xを図3のメッセージ送信情報リストとして記録する．

変数の代入文の他に，リターンメッセージと過去・現在型のメッセージ送信および，プリミティブメソッドの実行に対しても以下のような処理を行う．

(e) メソッド内にリターンメッセージが記述されている場合，メソッドの返値として新たに変数を生

成する．この変数の型情報に，返値としてメッセージの送信側に渡されるデータの型を記録し，この変数を図 3 の返値として記録する．

(d) と (e) では，未来型のメッセージ送信のメッセージの返値としての受信側から送信側へのデータの流れを考慮している．送信側から受信側へのデータの流れとしてメッセージの引数がある．過去型・現在型のメッセージ送信の場合は返値がないため，メッセージ引数がある場合のみ考慮すればよい．

(f) 他のオブジェクトへのメッセージ送信において引数があるとき，オブジェクト名とメッセージセレクタ名，引数として渡される変数を図 3 のメッセージ送信情報リストとして記録する．

(g) 配列型やリスト型変数には幾つかのプリミティブメソッドが定義されている．これによって，代入文と同様に変数の型が変わる可能性があるため，実行されるプリミティブメソッドに応じて変数の型情報を記録する．

(2) オブジェクト間のデータの流れ

ここではオブジェクト間のメッセージ送受信によるデータの流れを明らかにする．

既に，(1) の (d)(e)(f) によって，プログラム内に記述された全てのメッセージ送信において，

- どのような変数が引数として渡されるのか
- どのような変数が返値として渡されるのか
- 未来型メッセージでは上記の 2 つに加え，どの変数に返値が代入されるのか

が図 3 に記録されている．ここでは，これらのデータの受渡し関係によって影響を与える変数を関係リストによって結合する．

まず，メッセージ引数として渡される送信側の変数の関係リストに，受信側の対応する引数を加える．次に，未来型メッセージの場合は，受信側の返値の関係リストに送信側の未来変数を加える．これによって，オブジェクト間のデータの流れが変数の関係リストとして表される．

(3) 変数型の決定

(1)，(2) によってプログラム全体でのデータの流れが明らかにされ，それは関係リストによって表されている．各変数の型情報を関係リストに加えられている変数の型情報に反映させていけば，全ての変数についてその型が型情報として明らかになる．

例えば，図 4 のようなプログラムについて上記 (1)(2) の処理を行うと，各変数の型情報と関係リストは表 2 のようになる．

```
object  A
state   s1 = 1.                          (a)
methods {
    start : arg1 { | tmp1 tmp2 |
        tmp1 = s1.                       (b)
        tmp2 get: 2.                     (g)
        tmp2 at: 1 put: 0.               (g)
        tmp2 at: 2 put: 1.               (g)
        B mes1: tmp2.                    (f)
    }
    end {
        !'END'.                          (e)
    }
}

object B
methods {
    mes1: arg2 { | tmp3 |
        tmp3 = arg2 at: 1.               (c)
    }
    mes2 { | tmp4 |
        tmp4 = A end.                    (d)
    }
}
```

注: 括弧内は (1) で適用する処理を示す

図 4. サンプルプログラム

表 2. サンプルプログラムの型情報と関係リスト

変数名	型情報	関係リストに入る変数
A		
s1	整数	A の tmp1
arg1	—	—
tmp1	—	—
tmp2	整数配列	B の arg2
end の返値	文字列	B の tmp4
B		
arg2	—	B の tmp3*
tmp3	—	—
tmp4	—	—

* tmp3 には arg2 の型ではなく arg2 が配列として使われるときの要素の型が影響する

これらの型情報と関係リストから次のように型を推論できる．s1 が整数，よって tmp1 も整数となる．tmp2 は整数配列であることから，arg2 が整数

配列となり，tmp3 は arg2 の要素であるため，整数となる．A end の返値が文字列であるため，tmp4 は文字列となる．この結果，残りの arg1 の型が不明のままであるためタグ付きデータ型となる．

4.3 変数とメソッドの分離

4.2 によって，プログラム全体について一意的に型が決まらない場合でも，次のような方法によって，実行時の型チェックや型変換を取り除ける可能性がある．

(1) ある変数がプログラムの全体で見れば，整数・配列 2 つの型をとるとしても，ある範囲では整数，別の範囲では配列，というように分かれているならば，これは 2 つの変数に分けることが可能であり，そうすることによって変数の型を 1 種類に限定することが可能となる．

(2) あるメソッドについて，送信側によってメッセージ引数の型が異なるような場合，引数の型に応じて別々のメソッドを用意する．送信側では引数の型に合ったメソッドを呼び出すことによって，引数の型が 1 種類に限定できる．

現在，これらの具体的な実現方法について検討を行なっている．

5 おわりに

本稿では，PVM メッセージパッシングライブラリを用いた，並列オブジェクト指向言語 A-NETL の実装とその最適化の方法について述べた．

今後の課題として，型推論の実現とトランスレータへの反映が挙げられる．また，処理系の MPI への対応，トランスレートされたプログラムの実行性能の評価も行う必要がある．

本言語処理系によって，ユーザは並列オブジェクト指向によるメッセージパッシング型プログラムを容易に記述できる．また，標準メッセージパッシングライブラリを利用しているため，様々な WS クラスタや商用並列マシン上で実行が可能である．しかし，本言語処理系の有用性は実行性能次第であり，最適化による改善によって，純粋な C 言語プログラムの実行性能にどれだけ近付けることができるかが最も重要な課題である．

謝辞

本研究は，一部文部省科学研究費 (基盤 (C) 課題番号 08680346)，電気通信普及財団，並列・分散処理研究推進機構の援助による．

参考文献

[1] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderan : PVM 3 USER'S GUIDE AND REFERENCE MANUAL, ORNL/TM-12187, Sep 1994.

[2] 馬場敬信, 吉永努 : 並列オブジェクト指向トータルアーキテクチャA-NET における言語とアーキテクチャの統合, 信学会論文誌 Vol.J75-D-I No.8, pp563-574(1992).

[3] T. Baba and T. Yoshinaga : A-NETL: A Language for Massively ParallelObject-Oriented Computing, Proc. Massively Parallel Programming Models(MPPM'95), pp.98-105(1995).

[4] T. Baba, N. Saitoh, T. Furuta, H. Taguchi and T. Yoshinaga : A Declarative Synchronization Mechanism for Parallel Object-Oriented Computation,IEICE Trans. E78-D, 8, pp.969-981(1995).

[5] 岩下茂信, 國貞勝弘, 村上和彰 : PVM/AP1000 の実現および通信性能評価, HPC-50-13pp.97-104(1994).

[6] 小西弘一, 神舘淳, 加納健, Christopher Howson, 高野陽介 : MPI/DE — 並列計算機 Cenju-3 上の MPI ライブラリ—の性能評価, HPC-55-13, pp.97-96(1995).

[7] S. Numprasertchai, T. Yoshinaga and T. Baba: The Implementation of A-NETL on a Highlly Parallel Computer AP1000, 1996 "Akita" Summer United Workshops on Parallel, Distributed, and Cooperative Processing(SWoPP 秋田'96), 発表予定 (1996).

[8] J.Plevyak and Andrew.A.Chien:Precise Concrete Type Inference for Object-Oriented Languages,pp.324-340(1994).

[9] 吉永努, 馬場敬信 : 並列オブジェクト指向トータルアーキテクチャA-NET, -マルチコンピュータ開発と言語実装の現状-, 情報処理学会第 52 回全国大会, pp.6-97-6-98(1996).

[10] A. Yonezawa : ABCL: An Object-Oriented Concurrent System, The MIT Press, Cambridge, Mass(1990).

[11] MPI : A Message-Passing Interface Standard, Message Passing Interface Forum, May 5, 1994.