並列オブジェクト指向言語 mosaic のランタイムシステム

屋鋪 正史* 石原 義勝* 松川 力* 小西 健三* 瀧 和男**

*神戸大学大学院自然科学研究科
**神戸大学工学部情報知能工学科

並列オブジェクト指向言語 mosaic のマルチワークステーションシステム上における実行系の実装について報告する．mosaic の特徴の１つである，未生成オブジェクトへのメッセージ送信の機能を実現するために，ストリームオブジェクトと，その連結の操作の実装を行なった．さらに，連結の操作によって生じる連鎖の短縮について実装方法を検討し，実装を行なった．合わせてストリームオブジェクトの処理効率についても考察した．また，メモリ管理に関して，オブジェクトの回収にマーク・アンド・スイープ方式のガーベジコレクションを適用するため，マーキングルートとなるオブジェクトの ID を検出するための機構として，オブジェクトに参照リストを持たせる方法を提案した．

Run-Time System of Concurrent Object-Oriented Language "mosaic"

Masafumi YASHIKI* Yoshikatsu ISHIHARA* Chikara MATSUKAWA*
Kenzo KONISHI* Kazuo TAKI**

*Graduate School of Science and Technology, Kobe University
**Department of Computer and Systems Engineering,
Faculty of Engineering, Kobe University

We report an implementation of concurrent object-oriented language mosaic on the "Multi-Workstations". "Stream-object" and its implementation are proposed, which is used to realize the "un-created-object" function, a special feature of mosaic. An algorithm is designed to reduce the length of stream-objects chain, which grows at the "concatenate" operation. Execution efficiency of the stream-object implementation is also considered. For the garbage collection of objects, a new mechanism to find marking roots is proposed.

1 はじめに

非データ並列問題を扱う効率の良い並列処理は，一般に難しいとされているが，このような問題領域に対して，第五世代コンピュータプロジェクトで開発された並列論理型言語 KL1 [9] による，並列オブジェクトモデルに基づくプログラミングが有効であると指摘されている [5,7]．しかし，KL1 の特徴の1つである暗黙の通信・同期を実現するための不要な通信・同期コードが挿入されるための実行オーバヘッドが問題となることもあった．

そこで我々は，非データ並列領域に対して親和性が高いと思われる高並列オブジェクトモデル [7] を基本とする並列オブジェクト指向言語 mosaic を提案し，言語仕様や処理系の概要についてすでに報告してきた [4,8]．

本研究では mosaic 言語の実行系の効率の良い実装を目指している．実行系は小粒度・多数のオブジェクトをできるだけ軽い処理で効率良く扱えること，オブジェクト間のメッセージ通信が応答性に優れていることに留意して設計した．本報告では，実行系の実現方式の中から特に，言語が提供する未生成オブジェクトへのメッセージ送信の機能を実現するための方法について述べる．また，プログラム中に，C 言語に組み込みのデータ型と新しく導入したデータ型の混在を許すための，メモリ管理の方法について述べる．

本稿の構成は次の通りである，2節で mosaic の言語，実行系の設計方針の概要を述べる．3節では，実行系の実装の中でメッセージに関係する事項を述べる．さらに，4節では mosaic の特徴の1つである未生成オブジェクトへのメッセージ送出の機能の実現に関して提案したストリームオブジェクトの実装，連結・短縮について述べる．5節では，ガーベジコレクションに関する，オブジェクト ID の検出方法について提案する．6節では実装したストリームオブジェクトの処理効率について考察する．

2 mosaic の概要

2.1 言語の特徴

mosaic 初版の設計では，KL1 言語で並列オブジェクトモデルに基づくプログラムを記述する場合と同等の記述力となるような並列オブジェクト指向言語を目指し，KL1 の長所を採り入れた．基本となる並列オブジェクトモデルは，アクタモデル [3] に近い単純なもので，メッセージ通信は非同期メッセージ送信のみを許す．これは，ABCL/1 [1] の用語では，過去型メッセージである．

以下に，本稿で述べる実装内容に関係のある mosaic の特徴を紹介する．

未生成オブジェクトへのメッセージ送出機能　将来決まるはずのオブジェクトにメッセージを送り付けられる機能である．これによって，メッセージの経路を下流（メッセージを受けるオブジェクト側）から決める従来の方法に対して，上流（メッセージを送るオブジェクト側）からメッセージの経路を決定することができる．オブジェクトの生成とメッセージ送出の間の不要な逐次性の減少が期待できる．

C 言語からの移行性　C プログラマが移行し易いように，以下の方針をとる．

- メソッド記述には C 言語がそのまま使用できる．
- C 言語にあるデータ型とオブジェクト指向システムとして新しく追加するデータ型を自然なかたちで混在して使用できるようにする．

オブジェクト型　新しく追加したデータ型としてオブジェクト型がある．これはオブジェクトの ID を保持するためのもので，スロット変数，メソッド内の auto 変数として宣言できる．この変数に対しては，以下の操作が可能である．

- メッセージの送出先として使用
- メッセージの引数にセット
- 他のオブジェクト型変数が保持する ID の代入
- 他のオブジェクト型変数との連結 （後述）

2.2 実行系の設計方針

異なるプロセッサ間での通信時にできるだけ高い応答性を確保するために，徹底して OS を呼ばない通信方式をとる．プロセッサ内においては，オブジェクトの実体，及びそれらを管理する実行系全てを 1 つの UNIX プロセス中に実現し，オブジェクトのスケジューリングをメッセージドリブンなものとすることでオブジェクトのスイッチングをできるだけ軽いものとする．これは，メソッドの実行の途中では中断しない言語実行モデルとしたため可能となっている．プロセッサ内で起動されるオブジェクトは一度に 1 つであるが，各々のオブジェクトが一度に行なう計算が比較的小さい場合，マルチプロセ

スやマルチスレッドを用いてオブジェクトを複数起動し，オブジェクト間の通信にプロセス（スレッド）間通信を用いる方式より有利であると判断したからである．

3 mosaic 実行系の実装

マルチワークステーション上 [8] にシングルユーザ用システムとして実装している．

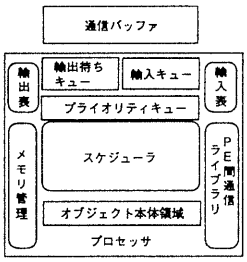

図 1: 実行系の概要

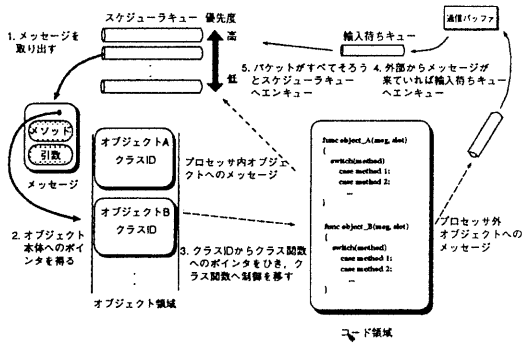

図 2: スケジューラの動作

3.1 メッセージの一括管理

非同期メッセージ送信のみを許すモデルを採用したことで，オブジェクト側にメッセージキューを持たせずにシステムでメッセージを一括管理することが可能になった．従って，オブジェクトの起動は，オブジェクトへ送られてくるメッセージをプライオリティごとの FIFO キューで管理し，メッセージをスケジューリングすることによって行なうことができる．つまり，スケジューラは最高プライオリティのキューから先頭のメッセージを取り出し，対応するオブジェクトのコードを呼び出すことによってオブジェクトの起動を行なう．

3.2 別プロセッサ上のオブジェクト ID の表現

メッセージ送出の際宛先として使用されるオブジェクト ID は，以下のような構成となる．

- プロセッサ ID
- プロセッサ内のオブジェクトテーブルのインデックス

オブジェクトの GC でオブジェクトのアドレスが変化しない GC 方式を採用したのでこれが可能となっている．オブジェクト ID はプロセッサ間を通して一意に決定できる．そして，実行系側では，オブジェクト ID に含まれるプロセッサ ID からオブジェクトが存在する場所がわかり，送出されたメッセージがプロセッサ渡りかどうかの判断を行なうことができる．このようにして，暗黙のプロセッサ間通信を実現している．

3.3 プロセッサ渡りのメッセージ送受信

マルチワークステーション上の実装では，プロセッサ間通信には LAN ではなく，外付け VME バス上の共有メモリを通信バッファに用いる．ユーザレベルの非特権プロセスからこれを直接操作し，メッセージの到着はポーリングによって検出している．

メッセージ送出 異なるプロセッサ上のオブジェクトへのメッセージは，直接該当するプロセッサ用の通信バッファに書き込まず，一旦宛先プロセッサごとに存在する輸出待ちキューに登録する．これは，通信バッファの容量が通信量に対して十分であるとはいえず，バッファフルになる可能性があるために設けたものである．バッファフルの状態でメッセージを書き込もうとした時，領域が空くのを待って書き込むよりは，一旦輸出待ちキューに登録し，次の処理を行なった方が効率がいいと思われるからである．スケジューラはこのキューに対する登録操作を行なうのと同じタイミングで，各プロセッサの通信バッファへの書き込みが可能であるかどうかチェックし，可能であればキューの先頭から通信バッファへの書き込みを行なう．

メッセージの受信 スケジューラは自分のプロセッサの通信バッファをポーリングする際，メッセージが書き込まれていればそれを取り出し一旦輸入キューに登録する．システムが扱うメッセージは可変長で

あり，複数のパケットに分割されている場合，一回のポーリングで全てのパケットが到着するとは限らない．よって，そのまま直接スケジューラのプライオリティキューに登録して処理することができないことからこのようなキューを設けている．このキューの上で全てのパケットが到着したことが判明したメッセージは直ちに対応するプライオリティキューに登録される．

4 未生成オブジェクトへのメッセージ送信の実現

4.1 未生成オブジェクトへのメッセージ送信とは

mosaic では，オブジェクト型変数に対して前述の操作が定義されている．オブジェクト型変数が宣言されると，その変数が実際に生成されたユーザオブジェクトの ID を保持していなくても，ID を保持しているかのように扱うことができ，その変数に対して，メッセージを送出したり，メッセージで他のオブジェクトへ渡すことができる．この変数を受けたオブジェクトも，変数に対して同様の操作が可能である．オブジェクト型変数の間には「連結」と呼ぶ操作が定義されている．オブジェクト型変数は，最初メッセージの流し口を持っていて，連結の操作でメッセージの流し先が決まり，連結されたオブジェクトは，メッセージを受けることができる．オブジェクトの生成に先だって送られたメッセージは，連結が行なわれるまでオブジェクト変数中にバッファリングされる．詳細は文献 [4].

4.2 ストリームオブジェクト

メッセージのバッファリングを実現する特別のオブジェクトで，プログラマからは見えない．ユーザオブジェクトと連結されると，メッセージをフォワードする機能を持つ．

ストリームオブジェクトには連結の操作が定義されている．連結の操作は，論理変数のユニフィケーションに近い操作で，ストリームオブジェクトが他のオブジェクトへのポインタ（ID）を 1 つ内部状態として持つことにより実現する．
連結の操作は以下の通りである．

- ストリームオブジェクト同士の連結
- ストリームオブジェクトとユーザオブジェクトの連結
- 異なるユーザオブジェクト同士の連結はエラーとする

宛先となるオブジェクトが生成される前にメッセージが送出された時は，ストリームオブジェクトが代りにバッファリングし，その後ユーザオブジェクトと連結され，メッセージがユーザオブジェクトにフォワードされることで未生成オブジェクトへのメッセージ送出の機能を実現している．

4.3 連結のアルゴリズム

連結の操作によって生じるストリームオブジェクトの連鎖が，環状にならないようにするため，連鎖の先頭のオブジェクト同士で一意に決まる向きに連結する．
連結の向きは，

- ストリームオブジェクト同士の時は，オブジェクト ID の小さい方から，大きい方
- ユーザオブジェクトとストリームオブジェクトの時は，ストリームオブジェクトからユーザオブジェクト

と決め，
連鎖の先頭は，

- 連結されていないストリームオブジェクト
  または，
- ユーザオブジェクト

と定義する．

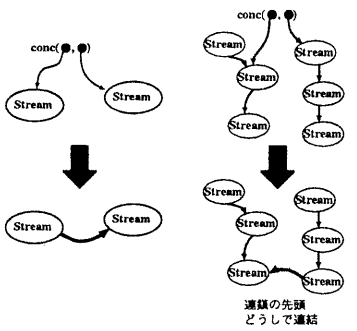

図 3: オブジェクトの連結

連結の手順

1. 連鎖の先頭を手繰り，先頭の ID を得る．

2. 先頭のIDを比較して上記のルールに従って連結する.

連鎖の先頭を手繰る際，プロセッサ渡りの連結が行なわれている部分は，メッセージ通信を用いなければならない．よって，まず，プロセッサ単位で先頭を手繰り，得られたオブジェクトのIDのうち，小さい方のオブジェクトのあるプロセッサに，実際の先頭を手繰るためのメッセージを送る．この時，メッセージには，もう一方のプロセッサ単位の先頭のオブジェクトのIDが含まれる．

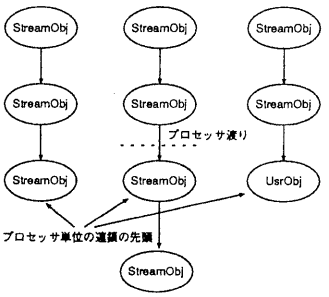

図 4: プロセッサ単位の連鎖の先頭

一方の実際の先頭のIDが得られると，もう一方へ実際の先頭を手繰るメッセージを送り，先頭のIDが2つ得られると，ルールに従って連結する．IDの大小関係が逆の場合は，IDの小さい方へメッセージを送り返し，ルールに従って連結する．

4.4 連鎖の短縮

ストリームオブジェクトの連鎖が生成され，連鎖の先頭にユーザオブジェクトが連結された後，連鎖上の全てのストリームオブジェクトを経由して，メッセージが目的オブジェクトに送られることは効率が悪い．さらに，プロセッサ渡りの連結が行なわれている部分では，メッセージが複数のプロセッサを渡って目的のオブジェクトへ送られる可能性があり，このような場合には，不要なプロセッサ渡りのメッセージが発生してしまう．従って，ユーザオブジェクトと連結された後は，連鎖を短縮する必要がある．

4.5 短縮のアルゴリズム

短縮とは，次の操作のことである．

1. ストリームオブジェクトの連鎖の先頭にユーザオブジェクトが連結された時点で，各々のストリームオブジェクトが直接ユーザオブジェクトを指すようにする（図5(a)）.
2. さらに，メッセージ送出があった時点で，メッセージ送出元のユーザオブジェクトが，直接送出先ユーザオブジェクトを指すようにする（図5(b)）.

連結の操作では，短縮を行なうためのメッセージを，連結先のストリームオブジェクトに送る．このメッセージには，連鎖を構成するストリームオブジェクトのIDが含まれている．以下，このメッセージをストリームメッセージと呼ぶ．

ストリームメッセージを受ける連鎖の先頭のストリームオブジェクトは，連鎖に含まれる全てのストリームオブジェクトのIDを知ることができ，このIDを用いて短縮を行なうことができる．

短縮の手順

i 連結時にストリームメッセージを送る.

ii ユーザオブジェクトと連結されると，ストリームメッセージによって得たIDを基に連鎖に含まれるストリームオブジェクトにユーザオブジェクトのIDを知らせる.（短縮1に対応）

iii その後，ストリームオブジェクト宛にメッセージが送られる時に，ユーザオブジェクトの中のストリームオブジェクトのIDを指している変数の中身を，直接ユーザオブジェクトのIDに書き換える.（短縮2に対応）

手順iの時に，プロセッサ渡りの連結が生じる部分では，短縮時に発生するプロセッサ渡りのメッセージを削減するために以下の工夫をする．

- プロセッサ渡りのストリームメッセージには自分のIDだけを含め，プロセッサ内の連鎖に含まれるストリームオブジェクトのIDは保持し続ける．

このようにすることで，プロセッサ渡りのストリームメッセージを小さくすることができ，ユーザオブジェクトのIDを知らせるプロセッサ渡りのメッセージを削減することができる．

メッセージのフォワード　短縮を行なうことで全てのストリームオブジェクトは，最終的には直接ユー

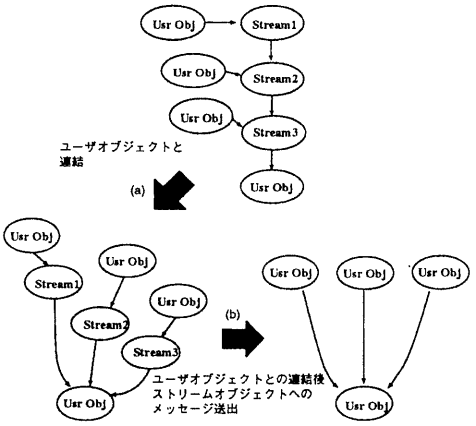

図 5: 連鎖の短縮

ザオブジェクトと連結されることが期待できる．そこで，連結先がユーザオブジェクトでない場合，バッファリングされているメッセージは一切フォワード動作をしない．このときストリームメッセージのみを送る．そして，短縮によりユーザオブジェクトのIDを得るとメッセージのフォワードを行なう．このようにすることで，フォワードによる不要なメッセージ通信も削減される．

5 メモリ管理

プログラムの実行に従って動的に使用状況が変化するメモリ領域は，オブジェクト本体，スロット変数領域，メッセージパケットである．この中で，メッセージパケットは，領域確保と解放のタイミングが明確であり，サイズを固定長にしているので，フリーリストで管理し実時間で回収する．オブジェクト本体はフリーリストで管理し，マーク・アンド・スイープ方式のガーベジコレクションを行なう．スロット変数領域はコピーイング GC を行なう．現在は実装途中である．

5.1 オブジェクトの参照管理

オブジェクト領域の GC は，他から参照されなくなったオブジェクトに対して行なうので，オブジェクトの参照管理が必要である．

外部参照管理 外部参照されているオブジェクトの管理は，オブジェクト ID のプロセッサ渡りの輸出入管理により行なう．各プロセッサの実行系は，オブジェクトの ID の輸出入表を持っている．

- 輸出表の管理：オブジェクトの ID の輸出は，メッセージの引数に載せて他のプロセッサへ送る時に発生する．このタイミングで，輸出するオブジェクト ID を輸出表に登録することでオブジェクトが外部参照されていることが分かる．輸出表の管理には重み付き参照カウント（WRC）[6] 方式を用いて行なう．エントリの解放は，WRC が 0 になった時に行なう．
- 輸入表の管理：輸入表へのエントリは，オブジェクトがメッセージを受けて起動され，引数を取り出す時に行なう．輸入表のエントリの解放は，GC の時に行なう．プロセッサ内のオブジェクトが輸入表にエントリされているオブジェクトを参照していない時に，WRC を輸出元に返し，エントリを解放する．

5.2 スロット変数領域にあるオブジェクト ID の検出

オブジェクトは，スロット変数領域にあるオブジェクト型変数で他のオブジェクトを参照する．マーキングルートとなるオブジェクト型変数がスロット変数の構造の任意の場所に現れるので，実行系はオブジェクト型変数を検出することができない．そこで，これを検出するための機構としてオブジェクト本体の構造に参照リストを設ける．このリストは，オブジェクトが生成される時にスロット変数として宣言されているオブジェクト型変数の数用意される．また，GC の起こるタイミングを以下のように限定したため，auto 変数で参照するオブジェクトを管理する必要はない．参照リストに，オブジェクトがスロット変数で保持しているオブジェクトの ID が含まれるので，このリンクをたどることにより参照しているオブジェクト ID を検出することを可能としている．

このような参照リストを設けた理由は，Full Reference Count では，他のオブジェクトからは参照されておらず，かつ，相互に参照しあっているオブジェクトを回収できない．このようなオブジェクトを回収するため，マーク・アンド・スイープ方式を選択し，マーキングルートとなるオブジェクトの ID を得るためである．参照リストの領域確保は，オブジェクトが生成される時の 1 回のみであり，解放される時は，GC によって回収される時である．したがって，オブジェクトの生成が実行開始時に集中し

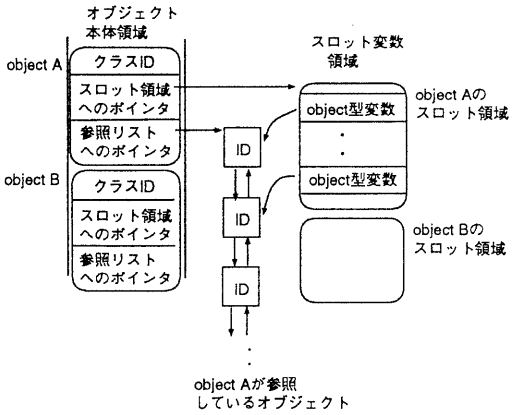

図 6: 参照リスト

て行なわれるモデルでは，影響は小さいと思われるが，実行が進むにつれて生成回収が繰り返される場合は，その影響は無視できないものとなる可能性がある．

5.3 プロセッサ内ローカル GC

GC を行なうタイミングは，オブジェクトからスケジューラへ制御が戻った段階で，オブジェクト本体用の領域の残りが一定値を下回っていればプロセッサ単位でローカル GC を行なう．GC では，プライオリティキューにあるメッセージの宛先となっているオブジェクト，そのメッセージの引数部から参照されているオブジェクト（メッセージの引数となっているオブジェクト ID の場所は，その場所の情報がメッセージに付加されているので ID を得ることができる），外部参照されているオブジェクトをルートとして，そのオブジェクトの参照リストにあるオブジェクトを再帰的に検索して，検索されたオブジェクトにマークをつける．全ての検索が終了した時点でマークのついていないオブジェクト領域に対してフリーリストへの回収を行なう．

6 ストリームオブジェクトに関する考察

今回提案した，ストリームオブジェクトの処理効率を調べるために最短経路問題を用いて実験を行なった．

最短経路問題　最短経路問題とは，重要なグラフ問題の 1 つで，出発点から目的点に至るあらゆる経路の中から最短の経路を捜し出す問題である．ここで言う最短の経路とは，点と点の間には辺が存在しその辺にはコストがあり．経路に含まれる辺のコストの和が最小となる経路のことである．

実験は，SPARCstation2 互換機 4 台を用いて，160×160=25600 点からなる正方格子のグラフを例題として行なった．辺のコストは乱数で与え，解を求めるアルゴリズムは文献 [10] の分散アルゴリズムを用いる．このアルゴリズムでは，探索の終了条件をグラフ上からメッセージが消滅した時としている．したがって，メッセージ消滅の検出を 2 通りの方法で行なった．

- システムが提供するメッセージ消滅の検出機構 ··· 方法 1
- ストリームオブジェクトを用いた方法 ··· 方法 2

ストリームオブジェクトを用いた方法　この方法は，KL1 などの論理型言語で用いられる「ショートサーキット法」[2] である．解探索の枝分かれの度にストリームオブジェクトを枝の数生成し，1 つの枝の探索が終了する（メッセージを消滅させる）度にストリームオブジェクトが連結される．最終的に全てのストリームオブジェクトが 1 つに連結されると，最上流に流し込んでおいた，探索終了のメッセージが到着し，終了の判定を行なうことができる．

実行時間を比較してみると，方法 2 では 3.4 倍の増加が認められた．両者とも最短経路の計算を行なう点オブジェクトの動作にかかった時間はほぼ同じであったので，この時間の増加はストリームオブジェクトの連結の手続きの時間と言える．

表 1: 実行時間

	実行時間 [sec]
方法 1	1.542
方法 2	5.287

生成されたオブジェクトの数は以下の通りである．

- ユーザオブジェクト数 ··· 25600
- ストリームオブジェクト数 ··· 約 13300/processor
- プロセッサ渡りの連結となるストリームオブジェクト数 ··· 約 660/processor

ここで，ストリームオブジェクトの連結の手続きについて考察する．通信時間を含まない連結に要した時間はストリームオブジェクト 1 つあたり 113.8μsec であった．この中には，連結において先頭を手繰って連結先を決定する手続きと，短縮にために上流の情報を伝える 2 つの手続きが含まれる．現在実装されている実行系において，自分自身にメッセージを送出し，スケジューラが 1 回まわってオブジェクトが起動されるまでの時間が，75.4μsec であることから，プロセッサ内の連結操作に要する時間は大きくはないと言える．

プロセッサ渡りの連結では，連結の依頼のメッセージと連結先を知らせるメッセージが発生する．この 2 つの通信時間はそれぞれ 382.2μsec であるが，一回の連結に 3020μsec かかっている．これは，先頭を手繰るためのメッセージがプロセッサ間で相互に送られ，また連結先を知らせるメッセージも相互に送られる．さらに，連結後もう一度先頭を手繰り，連鎖の確認をしているため，冗長なメッセージが発生していることが原因と考えられる．

この冗長なメッセージの内，連結後もう一度先頭を手繰り，連鎖の確認をするためのメッセージは削減することができるので，その分連結に要する時間を短くできると思われる．

以上のように，ストリームオブジェクトの連結について，プロセッサ内においては比較的効率良く実行されているが，プロセッサ渡りの連結では，メッセージ量の削減が必要であることが分かった．

7 おわりに

並列オブジェクトモデルを基にした並列オブジェクト指向言語 mosaic の実行系の実装を行ない，言語の特徴の 1 つである，未生成オブジェクトへのメッセージ送出の機能の実現するための，ストリームオブジェクトの実装について述べ，ストリームオブジェクトに対して定義された連結の操作のアルゴリズム，さらに，連鎖を短縮するためのアルゴリズムについて述べた．また．メモリ管理に関しては，言語仕様として任意の場所にオブジェクト型変数の記述を許しているため，マーク・アンド・スイープ方式の GC を行なう際のマーキングルートとなるオブジェクト ID の検出方法について報告した．

ストリームオブジェクトの連結操作の実装では，プロセッサ渡りの連結が効率良く行なわれていないことが判明し，メッセージ量の削減を検討する必要があることが分かった．

GC に関しては，未だ実装途中であり，その効率に対しての評価を行なうことが課題となっている．

スケジューラやメッセージ送受信の処理効率の向上，他の並列計算機への移植等を計画している．

謝辞

実装に関して，御協力頂いた，笹木歩氏に感謝する．

参考文献

[1] A.Yonezawa. *ABCL:An Object-Oriented Concurrent System.* MIT Press, 1990.

[2] 古川康一, 溝口文雄編. 並列論理型言語 GHC とその応用. 共立出版, 1987.

[3] 石川裕, 所真理雄. オブジェクト指向並行プログラミング言語. 情報処理学会学会誌, Vol. 29, No. 4, 1988.

[4] 小倉毅, 瀧和男. 並列オブジェクト指向言語とマルチワークステーション上 の実装. JSPP'94 論文集, pp. 97–104, May 1994.

[5] Kazuo TAKI. Parallel Inference Machine PIM. *Proc. of the Intl.Conf. on FGCS 1992, Tokyo*, pp. 50–72, June 1-5 1992.

[6] 瀧和男. 第五世代コンピュータの並列処理. bit 別冊. 共立出版, 1993.

[7] 瀧和男, 一吉伸行. マルチ PSI における並列処理とその評価 — 小粒度高並列オブジェクトモデルに基づくパラダイムについて —. 信学論, Vol. J75-D-I, No. 8, pp. 723–739, 1992.

[8] 瀧和男, 小倉毅, 小西健三. ワークステーション複合体による並列処理システム— 中・小粒度オブジェクト指向並列処理の実現 —. 情報処理学会 PRG13-7 研究報告, Vol. 93, No. 73, August 1993.

[9] K. Ueda and T. Chikayama. Design of the kernel language for the parallel inference machine. *The Computer Journal*, Vol. 33, No. 6, pp. 494–500, 1990.

[10] 和田久美子, 一吉伸行. マルチ PSI 上の最短経路問題の実現と評価. 情処計算機アーキテクチャ研資, Vol. 83, No. 79, pp. 83–91, November 1990.