システムソフトウェアと
オペレーティング・システム 67－10
(1994. 12. 8)

適応的再演型並列プログラムデバッガのPVM上での実現

三栄 武　　高橋 直久

NTT ソフトウェア研究所

〒180 東京都武蔵野市緑町 3-9-11

あらまし

我々はメモリ共有型並列プログラムを対象に、再演法を用いた並列プログラムデバッガdbxRについて議論を行なってきた。dbx Rは、静的なソースコード上と動的な実行動作上の両方に対しての停止位置設定手段の提供、実行動作上に設定された停止位置までの必要最小限な再演実行、決定的な状態での全プロセスの停止、再演に必要な機能を閉じ込めた適応的再演型ロック命令によるプログラム言語・デバッガコマンドからの高い独立性などの特長を持つ。本稿では、dbxRをメッセージパッシング型並列プログラムに適用する際の問題点について議論し、これら問題点を解決するためのデバッガモデルを提案する。また、ORNLで開発された標準的なメッセージパッシング型並列プログラムライブラリPVM上で作成した、本デバッガのプロトタイプについて、その実現法と動作例を示す。

和文キーワード　並列プログラム、メッセージパッシング、デバッガ、再演、要求駆動、ワークステーション・クラスタ

An Implementation of Adaptively Replayable Parallel Debugger on PVM.

Takeshi MIEI　Naohisa TAKAHASHI

NTT Software Laboratories

9-11, Midori-Cho 3-Chome Musashino-Shi, Tokyo 180 Japan

Abstract

The dbxR which we previously proposed was a replay-based debugger for shared-memory parallel programs. It enabled us to specify break points both on static source code and on dynamic execution sequences of parallel programs, to reproduce the non-deterministic execution behavior of parallel programs by using a demand-driven replay method, and to halt the execution of the program at a determined state. This paper proposes the dbxR debugger model should be applied to parallel programs used for message passing communications. It also describes the prototype implementation of the parallel debugger on ORNL's PVM - a widely used message passing library - and shows examples of program debugging with the prototype debugger.

英文 key words　parallel program, message passing, debugger, replay, demand driven, workstation cluster

1 はじめに

同じ入力に対しても実行毎に動作が変化する可能性がある並列プログラムに対して、Instant Replay 法[1]など並列プログラムの実行動作を記録し、その記録をもとに同じ実行動作を得る再演法が提案されている。しかし、これら再演法を用いたデバッガの実現法に関しては十分な議論がなされていない[2]。

我々はこれまで、共有メモリ型並列プログラムを対象に、再演法を用いた並列デバッガ dbxR[3]を提案し、プロトタイプを作成した。dbxR は、静的なソースコード上と動的な実行動作上の両方に対して停止位置を設定可能である。実行動作上に設定された停止位置までの必要最小限な実行を再演する。決定的状態で全プロセスを停止させる。再演に必要な機能を閉じ込めた適応的再演型ロック命令によりプログラム言語・デバッガコマンドからの高い独立性を達成している等の特長を持つ。

本稿では、dbxR をメッセージパッシング型並列プログラムに適用する際の問題点について議論し、これら問題点を解決するためのデバッガモデルを提案する。さらに、このモデルに基づき、標準的なメッセージパッシング型並列プログラムライブラリ PVM[5]上で作成したプロトタイプ dbxR-II の実現法と、その動作例についても述べる。

2 メッセージパッシング型並列プログラム

本稿は次のようなメッセージパッシング型並列プログラムを対象に議論する。

- アプリケーションプログラムは複数のプロセスからなる
- 各プロセスは独立したアドレス空間を持ち、非同期メッセージ通信によりデータ授受を行なう
- メッセージには送信元プロセスの識別子、送信先プロセスの識別子およびメッセージタグが付与される。メッセージタグとはユーザが定義し与えるメッセージ種別である
- メッセージは有限時間の遅延の後、受信プロセスのアドレス空間内にある無限長バッファに到着することが保証される
- 受信プロセスは送信プロセス識別子とメッセージタグを指定し、一致するメッセージをバッファから取り出す

3 共有メモリ型並列プログラムの適応的再演型デバッガ

先に提案した適応的再演型デバッガ dbxR[3]では、共有データを介してプロセス間通信を行なう共有メモリ型並列プログラムを対象とし、プログラムの実行動作を記録する実行監視モードと、その記録(実行記録と呼ぶ)に従ってプログラムの動作を再現する再演実行モードを持つ。実行監視モードでは、共有データへのアクセス順に、プロセス識別子とアクセス種別(read/write を表す 1bit)を共有メモリ上の実行記録に保存する。プロセス P1 が共有データへ書き込み、その後プロセス P2 が読み出した場合の実行記録の例を図 1に示す。

再演実行モードでは、次のような特長を持つ。

1. 静的なソースコード上と動的な実行記録上の両方に対して停止位置を設定する手段を提供する。
2. ソースコード上の停止位置が指定されていない、あるいは、指定された停止位置にどのプロセスも到達しない場合には、要求駆動再演法[4]を用いて、実行記録上で指定された停止位置に到達するために必要最小限な命令のみを再演する。
3. ソースコード上の停止位置に到達したプロセスが発生した場合には、指定された実行記録上の停止位置までの最小限の命令の再演を進め、全てのプロセスが実行を進められない状態に到達したことを検出して、全てのプロセスが決定的な状態で停止することを保証する。
4. 適応的再演型ロック命令と呼ぶ排他制御命令の機能の中に、再演に必要な実行制御機能を閉じ込めて実現することにより、プログラム言語やデバッガコマンドからの独立性を高めている。

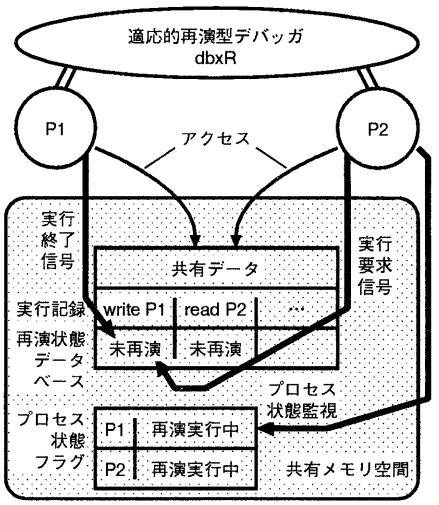


図 1: 適応的再演型デバッガ dbxR

要求駆動による再演は次のように動作する。プロセスは共有データへのアクセスを再演する際に、先に実行すべきアクセス命令に実行要求信号を送り、実行終了信号を待機する。実行要求信号を与えられたプロセスはアクセス終了後、実行終了信号を送り返す。この時、実行記録の各アクセス命令への実行要求ならびに実行終了は、共有メモリ上のデータベース(再演状態データベースと呼ぶ)で管理する。図 1に示したように、P2 が共有データからの読み出し(read P2)を再演する際、先に P1 の書き込み(write P1)

が必要なことから、P2は実行要求信号として、"write P1"の再演状態データベースを書き換える。P1は、共有データへの書き込み終了後、実行終了信号として"write P1"の再演状態データベースを再び書き換えている。

決定的な状態での再演停止では、ソースコード上の停止位置に到達して停止したプロセスを待ち合わせて、もはや実行を進められないことを各プロセスが個別に判定することにより、実現している。このため、プロセスが再演実行中か停止しているかを表すフラグ(プロセス状態フラグと呼ぶ)を共有データ上に管理する。プロセスは実行終了信号を待つ際に、上記フラグを監視することにより、待ち合わせるプロセスの停止を検出する。図1では、P2は実行要求信号を送った後、実行終了信号を待ち合わせると同時に、P1のプロセス状態フラグの監視を行なう。P1が停止した場合には、P1のプロセス状態フラグが変化し、P2がもはや実行を進められないことを検出する。

4 メッセージパッシング型並列プログラムへの適用上の問題

メッセージパッシング型並列プログラムにdbxRを適用し、再演実行を実現するには、メッセージの送受信順序が実行監視時と同一となるよう制御する必要がある。この場合、プロセスのアドレス空間が独立しているため、dbxRにおいて共有データ上に構築していた実行記録、再演状態データベースおよびプロセス状態フラグの実現法が課題となる。

(1) 実行記録 メッセージ通信は、送信プロセスが受信プロセス内のバッファにメッセージを書き込み、受信プロセスがそのバッファから読み出していると見做せる。よって、受信プロセスがバッファへのアクセス順序を実行記録として保存すれば、再演実行モードでのメッセージ通信の再現が可能となる。この時、実行記録は受信プロセスのアドレス空間に置かれることになるため、プロセスのアドレス空間の分離により、送信プロセスが実行記録へのアクセスを直接行なえないという問題がある。

(2) 再演状態データベース 再演実行モードにおいて受信命令を再演するプロセスは、対応する送信プロセスへ実行要求信号を送り、送信命令の再演を促すが、アドレス空間の分離により、実行要求信号もメッセージを用いて実現する必要がある。このため、再演状態データベースを実行要求信号の送信側と受信側の両側で管理しなければならない。この時、メッセージの到着遅延により、各プロセスの再演状態データベースに不整合が生じる場合がある。このため、その一貫性を保つ機構が必要となる。

(3) プロセス状態フラグ 決定的な状態での再演停止の実現ではプロセス状態フラグが必要となるが、共有メモリ上にプロセス状態フラグを実現できないために、他プロセスの停止を直接監視することはできない。他プロセスの状態を検査するメッセージを送ることで、他プロセスの停止を検出可能であるが、実行終了信号を待機する間に繰り返しメッセージを送付するためにオーバヘッドが大きくなる問題がある。

5 メッセージパッシング型並列プログラムの適応的再演型デバッガ

本章では、4章で述べた問題点を解決し、3章で述べたdbxRをメッセージパッシング型並列プログラムに適用したデバッガモデルについて述べる。

5.1 実行監視モード

無限長の受信バッファを持つメッセージパッシング型並列プログラムでは、メッセージ受信後のプロセスの実行動作はメッセージの受信順序にのみ依存する。すなわち受信動作は、送信プロセスの識別子とメッセージタグを指定し、一致するメッセージを受信バッファから取り出すことから、メッセージの送信順序と受信バッファへの到着順序は、受信プロセスの実行動作に影響を与えない。このため、受信バッファから取り出したメッセージの順序が実行監視モードと再演実行モードで同一となることを保証すれば、並列プログラムの実行動作を再演可能となる。

よって本デバッガでは、受信プロセスが受信命令を実行する毎に、受信バッファから取り出した順序に従って、メッセージの送信プロセス識別子とメッセージタグの対を並べてできる実行記録を作成する。これにより、4章の(1)で述べた問題点を回避できる。また、実行記録へアクセスする際に送信プロセスとの競合が発生しないことから、受信プロセスの並列性を損なわずに実行記録の作成が可能となる。

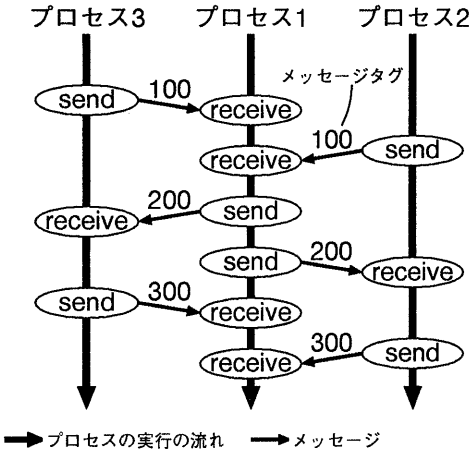
3つのプロセスからなる並列プログラムの実行動作と作成される実行記録の例を図2に示す。図2において、ノードはプロセスが送信/受信命令を実行したことを示し、細線矢印はメッセージを示している。各メッセージに付随した数字は、使用されたメッセージタグを表す。太線矢印は各プロセスの実行の流れを示している。

5.2 再演実行モード

5.2.1 実行記録上での停止位置設定と要求駆動による最小限再演

再演実行モードにおいて本デバッガは、dbxRと同様に、静的なソースコード上と動的な実行動作上の2種類の停止位置の設定手段をプログラマに提供する。前者は通常のデバッガにおいて提供されるブレークポイント(BPと略す)である。後者は5.1節で作成した実行記録上に設定する停止位置であり、デマンドポイント(DP)と呼ぶ。DPを与えられたプロセス(DPプロセスと呼ぶ)は再演を開始し、先に提案した要求駆動型並列実行制御に基づく再演法[4]を用いて、DPに到達するのに必要最小限な命令のみを再演実行する。本節では、DPのみを設定し、BPを設定しない場合の再演実行について述べる。

本デバッガでは、実行要求信号をメッセージで実現し、送信デマンドと呼ぶ。受信命令を再演するプロセスは、対応する送信プロセス識別子とメッセージタグを実行記録から読み出し、図3に示したように、自プロセス識別子とメッセージタグを内容とした送信デマンドを、送信プロセスへ送出し、メッセージの到着を待機する。送信デマンドを受けたプロセスは、送信デマンドの内容によって指定される



プロセス1の実行記録				
送信プロセス	3	2	3	2
メッセージタグ	100	100	300	300

プロセス2の実行記録		
送信プロセス	1	
メッセージタグ	200	

プロセス3の実行記録		
送信プロセス	1	
メッセージタグ	200	

図 2: 作成する実行記録の例

送信命令まで再演を進め、メッセージ送信を行なう。受信命令を再演するプロセスは、到着したメッセージをもって実行終了信号と見做し、受信命令の再演を終了する。

この時、プロセスは送信デマンド表と呼ぶデータベースで受け取った送信デマンドの総数、および再演した送信命令の総数を個別に管理する。これにより、4章の (2) で述べたような、メッセージの到着遅延による再演状態データベースの不整合を回避する。すなわち、受け取った送信デマンドの総数を m、再演した送信命令の総数を n とすると、$m > n$ の場合には、実行を要求されたが未再演の送信命令が存在することから、再演を進める必要あることを示し、$m < n$ の場合には、再演済の送信命令に対して送信デマンドの到着が遅延していることを示している。$m = n$ の場合には、送信デマンドを与えられた送信命令は全て再演済であり、かつ再演済の送信命令に対する送信デマンドは全て受信したことを示している。

図4は、図2においてプロセス3の受信命令に DP が設定された場合に、再演が行なわれる部分を網掛けで示しており、送出される送信デマンドを破線矢印で示している。DP プロセスであるプロセス3は再演を開始するが、プロセス1と2は、DP プロセスではない (非 DP プロセスと呼ぶ) ため、再演開始時には停止している。プロセス3は

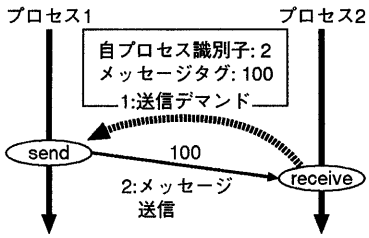


図 3: 要求駆動型再演法

DP が示す受信命令を実行する直前に、プロセス1へ送信デマンドを送出する。プロセス1は再演を開始し、プロセス2へ送信デマンドを送出すると、プロセス2が再演を開始する。その後、プロセス2からプロセス1へメッセージが送信されると、プロセス1は再演を進め、プロセス3へのメッセージ送信を行なって停止する。プロセス1からのメッセージを受信したプロセス3は、DP までの再演を終了し、停止する。

ここで、非 DP プロセスが送信デマンドにより実行を要求された送信命令のうち、各プロセスが最後に実行する送信命令を2次 DP と呼ぶ。2次 DP の位置は、他プロセスからの送信デマンドの到着により、変化する可能性がある。このように、本デバッガシステムでは、送信デマンドを用いて各プロセスが DP、2次 DP までの再演を行なうことにより、DP が示す受信命令までの必要最小限な命令の再演実行を実現する。

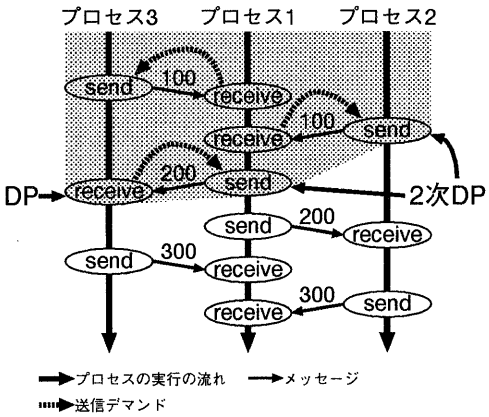


図 4: 要求駆動を用いた再演実行

5.2.2 再演停止法

本デバッガでは、DP と BP を併用し、再演中にあるプロセスが BP に到達した場合には、決定的状態でのプログラム停止を保証する。この時、再演する命令は、DP に到達するのに必要最小限の命令の一部分であることを保証する。プロセスがもはや実行を進められない実行位置を、そのプロセスの SP と呼び、全てのプロセスが SP に到達

した状態を安定状態と呼ぶ。本デバッガでは、安定状態を検出して再演を停止することで、決定的状態でのプログラムの停止を実現する。

プロセスは次の4種類の実行位置のいずれかに到達した場合に、SP到達が保証される。

- DP
- 新たな送信デマンドが到着しない2次DP
- BP
- 2次BP(BPまたは2次BPで停止したプロセスからのメッセージを待機する実行位置)

再演実行中にあるプロセスがDPに到達した時には、要求駆動再演法[4]の性質から、他の全てのプロセスが2次DPに到達しており、かつ新たな送信デマンドがその時点以後送出されないことが保証できる。よって全てのプロセスがSPに到達しており、安定状態が保証されるため、その時点で再演を停止すれば良い。

あるプロセスがBPに到達した時には、どのプロセスもDPに到達しないので、他のプロセスのSP到達を個々に検出する必要がある。しかし、4章の(3)の問題により、2次BP到達の検出が容易ではない。プロセスの制御を行なうデバッガは、プロセスのBP到達を検出できることから、本デバッガでは以下の手法を用いて各プロセスの2次BP到達を検出する。

受信命令を再演するプロセスは、送信デマンドを送出すると同時に、待ち合わせるプロセスの識別子をデバッガに通知する。BPで停止したプロセスを検出したデバッガは、先の通知に従って2次BPへ到達したプロセスを求め、2次BPへ到達したことをメッセージで通知する。メッセージを待機中のプロセスは、要求したメッセージが到着すれば再演を継続でき、デバッガからメッセージが到着すれば2次BPに到達したと判定できる。これにより、受信命令を再演するプロセスは、メッセージを1つデバッガへ送信するだけで2次BP到達の検出が可能となる。

図5は、プロセス3の受信命令にDPを設定し、再演途中にプロセス1がBPに到達した場合の再演状況を網掛けで示している。プロセス1はBPで停止し、プロセス2は送信デマンドを与えられた送信命令を再演後、2次DPで停止する。プロセス3はプロセス1からのメッセージを待機し、受信命令の直前で2次BP到達となり停止する。

6 PVM上での適応的再演型デバッガdbxR-IIの実現

前章までに述べたメッセージパッシング型並列プログラムの適応的再演型デバッガのプロトタイプ(dbxR-IIと呼ぶ)を、PVM[5]上で作成した。現在、dbxR-IIは5.1節、5.2.1節で述べた実行記録作成機能とDPを用いた要求駆動による最小限再演機能の実現を終え、BPを用いた場合の再演停止機能の実装を進めている段階である。

6.1 PVM

PVM[5]はネットワークに接続された複数の異機種計算機を統合して仮想的な並列計算機を構成するライブラリ群

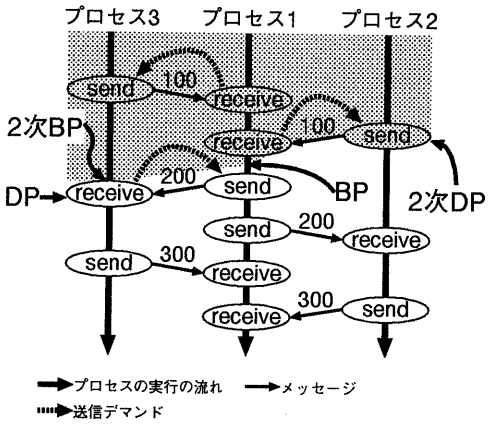


図5: DPとBPを併用した再演実行

である。図6に示したように各計算機上に常駐するPVMのデーモン(pvmd)が、互いに通信し合うことにより仮想並列計算機を構築する。この並列仮想計算機上で動作するプロセス(PVMプロセスと呼ぶ)は、PVMが提供するライブラリを用いることにより、非同期メッセージ通信を行なう。

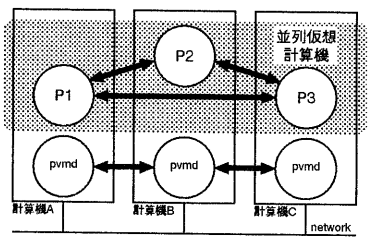


図6: PVMによる並列仮想計算機

6.2 システム構成

dbxR-IIを用いてデバッグする場合に、実行監視モードでは図7に示すように、PVMプロセスである被デバッグプロセス(P1～P3)をPVMのデーモン(pvmd)の下で実行させる。再演実行モードでは、図8のように、これらのプロセスにdbxR-IIのプロセス(FEP、ローカルデバッガ)を加えて実行させる。両モードにおいて、被デバッグプロセスは本システムが提供するライブラリ(dbxr-lib)を用いてメッセージ通信と子プロセスの生成を行なう。dbxr-libは、PVMが提供するメッセージ通信命令pvm_send(), pvm_recv()および子プロセス生成命令pvm_spawn()と同様のインタフェースを持つdbxr_pvm_send(), dbxr_pvm_recv(), dbxr_pvm_spawn()を提供する。

dbxr-libは図7のように、実行監視モードで実行記録とプロセステーブルを保存するため、ファイルのマッピングを行なう。dbxr_pvm_recv()は、メッセージを受信する毎に、送信プロセスの識別子とメッセージタグを実行記録に

書き込む。dbxr_pvm_spawn() はプロセス生成時に、子プロセスの識別子をプロセステーブルにに書き込む。これらのデータは、OS のファイルマッピング機能により、ファイルに書き込まれ保存される。

dbxR-II は被デバッグプロセスを個別に管理するローカルデバッガと、これらを司るプロセス (FEP) からなる。FEP も被デバッグプロセスと同様に PVM プロセスであり、図8のように、ユーザからのコマンドを受け付け、ローカルデバッガの制御ならびに被デバッグプロセスの制御を行なう。

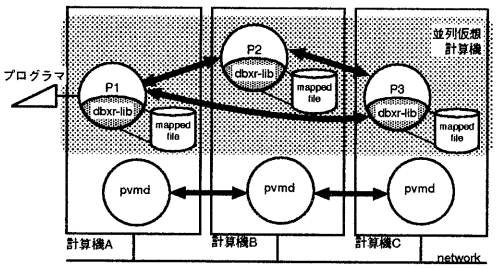


図 7: 実行監視モードでの被デバッグプロセスの構成

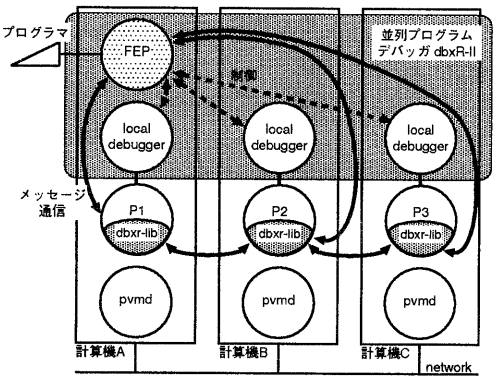


図 8: 再演実行モードでの dbxR-II プロセス構成

6.3 データ構造

本デバッガシステムでは、図9に示す実行記録、プロセステーブル、送信デマンド表、プロセス対応表の 4 種類のデータ構造を使用し、これらのデータ構造は被デバッグプロセスが 1 つずつ保持する。また、dbxR-II は全プロセスのプロセステーブルと実行記録のコピーを保持する。各データ構造の各フィールドの意味と取り得る値を表1〜4に示す。ここで、各データ構造が用いるプロセス識別子は、全て PVM により付与される値である。

実行記録 (表 1) の num_record は、以下に続く受信履歴の総数を表す。実行記録の (from, tag, demand) の 3 つ組のうち、(from, tag) の対は、実行監視時における 1 回のメッセージ受信の履歴を表し、そのプロセスが受信バッファ

実行記録

num_record		
from	···	from
tag	···	tag
demand	···	demand

プロセステーブル

m_parent		
r_parent		
m_tid		
r_tid		
dbxr_tid		
dp_flag		
num_child		
m_child	···	m_child
spawn_demand	···	spawn_demand

送信デマンド表

from	tag	demand	done
:	:	:	:

プロセス対応表

m_tid	r_tid
:	:

図 9: dbxR-II が使用するデータ構造

num_record :	実行記録の長さ
from :	PVM が管理する送信プロセスの識別子
tag :	メッセージタグ
demand :	送信デマンドの発生 / 完了状態
	= NONE (送信デマンドがない時)
	= DO
	(送信デマンドがあり、未再演の時)
	= DONE (再演実行済の時)

表 1: 実行記録の各フィールドの意味

からメッセージを取り出した順序で並んでいる。また、(from, tag, demand) の demand の値により受信命令の再演状況を表している。再演開始時に demand は全て NONE に初期化され、(from , tag) で表される受信命令の再演を終了する毎に、付随する demand を DONE にして、その受信命令が再演済みであることを表す。プログラマが DP を設定すると、対応するメッセージに対する demand の値を DO にする。DP を設定されたプロセスは、demand が DO の値を持つ受信履歴までの命令を再演する。

プロセステーブル (表 2) の m_parent, r_parent, m_tid, r_tid は、各々、実行監視時と再演実行時の親プロセスの識別子および自プロセスの識別子であり、m_parent と m_tid は実行監視時に、r_parent と r_tid は再演実行時に値を代入する。dbxr_tid は FEP のプロセス識別子であり、被デバッグプロセスと FEP のメッセージ通信に用いる。dp_flag は、自プロセスが DP プロセスであるか否かを示す。num_child は子プロセスの総数を表し、実行監視時の子プ

m_parent :	実行監視時の親プロセスの識別子
r_parent :	再演実行時の親プロセスの識別子
m_tid :	実行監視時の自プロセス識別子
r_tid :	再演実行時の自プロセスの識別子
dbxr_tid :	FEP のプロセス識別子
dp_flag :	DP フラグ = 1 (DP プロセス) = 0 (非 DP プロセス)
num_child :	子プロセスの数
m_child :	実行監視時の子プロセスの識別子
spawn_demand :	子プロセス生成の要求 / 完了状態 = NONE (子プロセス生成の要求がない時) = DO (子プロセス生成の要求があり、未生成の時) = DONE (子プロセス生成済の時)

表 2: プロセステーブルの各フィールドの意味

ロセスの識別子を生成順に m_child に格納する。(m_child, spawn_demand) の対は、再演実行時における子プロセス生成の状況を示している。子プロセス未生成の時、spawn_demand の値は NONE である。dbxr_pvm_spwan() は、spawn_demand の値が DO になると子プロセスを生成し、生成後に値を DONE にする。

送信デマンド表 (表 3) は、5.2.1で述べた送信デマンドと再演済送信命令の総数を管理するデータベースであり、(from, tag, demand, done) の 4 つ組で 1 つのエントリを形成する。from は送信先プロセスの再演実行時の識別子であり、tag は送信の際にメッセージに付与するメッセージタグである。送信デマンドを受信する毎に、送信デマンドの内容と (from, tag) が一致するエントリの demand の値を 1 増加させ、また、送信命令を再演する毎に対応する done の値を 1 増加させる。

from	送信先プロセスの識別子
tag	メッセージタグ
demand	与えられた送信デマンドの総数
done	送信済のメッセージの総数

表 3: 送信デマンド表の各フィールドの意味

m_tid	実行監視時のプロセス識別子
r_tid	再演実行時のプロセス識別子

表 4: プロセス対応表の各フィールドの意味

プロセス対応表 (表 4) は、実行監視時と再演実行時のプロセス識別子の対応表であり、(m_tid, r_tid) で 1 つのプロセスを表現する。受信命令を再演するプロセスは、実行記録より実行監視時の送信プロセスの識別子を読み出した後、本対応表を用いて、再演実行時のプロセス識別子へ変換する。

6.4 dbxR-II における被デバッグプロセスの動作

本節では、dbxr-lib が提供するメッセージ送受信命令 dbxr_pvm_send()、dbxr_pvm_recv() の実行監視モードおよび再演実行モードにおける動作について述べる。

6.4.1 実行監視モード

dbxr_pvm_send() メッセージの送信を行なう。

dbxr_pvm_recv() メッセージを受信バッファより取り出す。取り出したメッセージの送信プロセス識別子、メッセージタグを実行記録の最後の (from, tag) に書き加えて、num_record の値を 1 増加させる。

6.4.2 再演実行モード

dbxr_pvm_send()

1. 送信先プロセスの識別子を tid, 使用するメッセージタグを msgtag とし、送信デマンド表の (from, tag) が (tid, msgtag) と一致するエントリを検索し、対応する done フィールドの値を 1 増加させる。
2. tid で示されるプロセスへメッセージを送信する。

dbxr_pvm_recv()

1. 実行記録を調べ、demand フィールドが NONE あるいは DO である最初の (from, tag, demand) の 3 つ組を読み出す。
2. 実行監視時においてプロセス識別子が from であったプロセスの、再演実行時のプロセス識別子をプロセス対応表から調べ、r_tid とする。プロセス対応表で r_tid が求まった場合には 4へ行く。プロセス対応表で求まらなかった場合には 3へ行く。
3. dbxR-II へ from の値を内容とする変換依頼メッセージを送り、dbxR-II より再演実行時のプロセス識別子を受け取る。これを r_tid として、(from, r_tid) の対を新たなエントリとして、プロセス対応表へ登録する。
4. 1で読み出した実行記録の (from, tag, demand) の 3 つ組の demand フィールドを DO にする。自プロセス識別子と tag を内容とする送信デマンドを r_tid が示すプロセスへ送信する。
5. r_tid が示すプロセスからメッセージタグの値が tag であるメッセージの到着を待機する
6. 1で読み出した実行記録の (from, tag, demand) の 3 つ組の demand フィールドを DONE にする。

6.5 PVM プログラムの再演例

図 10に示したテストプログラムを、dbxR-II を用いて再演させた場合の動作例を図 11に示す。図 10のテストプログラムは、test1、test2 の 2 つのプログラムで構成さ

れ、test1 が test2 を子プロセスとして生成する。テストプログラムは、test2 → test1 → test2 の順でメッセージ通信を行ない、終了する。メッセージタグには全て 1 を用いる。

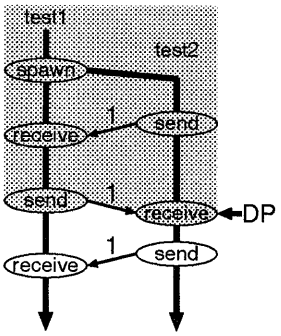


図 10: テストプログラム

図 11では、test2 プロセスの受信命令に DP を設定した場合の動作例を示している。図 11の 1 行目で dbxR-II を起動しており、引数の"test1" で被デバッグプログラムを指定している。2 行目の文字列"(dbxR)"は、dbxR-II のプロンプトであり、プログラマは run コマンドを用いて被デバッグプログラムを起動している。起動時の dbxR-II は実行監視モードで動作しており、3 ～ 10 行目では、実行監視モードでの被デバッグプロセスからの出力を表示している。

11 行目の collect コマンドで、先の実行で作成された実行記録の内容を dbxR-II に収集する。12 ～ 15 行目では、record コマンドを用いて収集した実行記録の内容表示している。14 行目では、実行監視モードにおいてプロセス識別子 40024 を持つプロセス 0 が、プロセス識別子 80011 のプロセスからメッセージタグ 1 のメッセージを 2 回受信していることを示している。

16 行目では replay コマンドを用いて再演実行モードに切替え、17 行目の run コマンドで被デバッグプログラムを起動しているが、この時点では、まだ DP が設定されていないので、再演は開始されない。18 行目で demand コマンドを用いて、プロセス 1 の 0 回目の受信命令に DP を設定を行ない、再演を開始する。19 ～ 25 行目は被デバッグプログラムからの出力である。DP までの再演終了後、被デバッグプロセスは自動的に停止させられる (27,29 行目)。31 行目の record コマンドで表示した実行記録では、再演した受信命令を"*" を付加して表示し、2 つの被デバッグプロセスは、ともに受信命令を 1 回ずつ再演したことを示している。

7 おわりに

本稿では、メッセージパッシング型並列プログラムを対象とした、再演法を用いた並列デバッガモデル dbxR-II を提案し、PVM 上での実現について述べた。現在、プロトタイプでは実行記録作成機能および実行記録上の停止位置までの最小限再演機能が稼働している。今後は、ブレークポイントを併用した際の再演停止機能の実現およびユーザインタフェースの拡充を行なう予定である。また、作成したプロトタイプを用いて、再演法を用いた並列プログラムデバッガの評価実験を進める予定である。

```
 1 % dbxr test1
 2 (dbxR) run
 3 Starting program: /home1/take/pvm3.3.1/bin/
 : [中略]
10 Program exited with code 01.
11 (dbxR) collect
12 (dbxR) record
13 Proc(m_tid)      From(Tag)
14   00(40024)  0: 80011(  1) 1: 80011(  1)
15   01(80011)  0: 40024(  1)
16 (dbxR) replay
17 (dbxR) run
18 (dbxR) demand message 1 0
19 Starting program: /home1/take/pvm3.3.1/bin/
   SUN4/test1
20 Starting program: /tmp_mnt/home1/take/
 : [中略]
25 parent:send:peanuts:nttspe
26
27 Program received signal SIGINT, Interrupt.
28 0x1cf88 in gettimeofday ()
29 Program received signal SIGINT, Interrupt.
30 0x1cbb0 in select ()
31 (dbxR) record
32 Proc(m_tid)      From(Tag)
33   00(40024)  0: 80011(  1)*  1: 80011(  1)
34   01(80011)  0: 40024(  1)*
35 (dbxR) quit
```

図 11: dbxR-II によるテストプログラムのデバッグ例

謝辞

日頃、御指導御討論いただく後藤滋樹部長をはじめ広域コンピューティング研究部の皆様に深く感謝します。

参 考 文 献

1) LeBlanc T.J. et al., "Debugging Parallel Programs with Instant Replay," IEEE Trans. Comp., C-36, 4, pp.471-482, 1987.

2) 山田, "並列処理システムにおけるプログラムデバッギング", 情処学会誌 VOL.34 NO.9,1993.

3) 三栄武, 高橋直久, "適応的再演型ロック命令を用いた並列プログラムデバッガの実現", JSPP'94, pp.241-248, 1994.

4) 高橋直久, "データ共有型並列プログラムの要求駆動型再演システムの実現と評価", JSPP '90, pp.361-368, 1990.

5) A1 G., et al., "PVM3 User's guide and refernce manual", ORNL/TM-12187, May 1993.