PVMプログラムのための再演型デバッガの実現と評価

三栄 武[†] 高橋直久[†]

我々はこれまでに，メモリ共有型並列プログラムを対象とした再演法を用いた並列プログラムデバッガdbxRを提案し，その実現法を示した．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の停止位置設定手段の提供，実行動作上に設定し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プロセス間通信命令に閉じ込めてプログラム言語・デバッガコマンドからの独立性を高めるなどの特長を持つ．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に生じる課題について議論し，これら課題を解決したデバッガモデルを提案，その実現法について述べる．また，Oak Ridge National Laboratory (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の実現法を示し，その基本的特性の評価についても報告する．

Implementation and Evaluation of a Replay-Based Debugger for PVM Programs

TAKESHI MIEI[†] and NAOHISA TAKAHASHI[†]

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 Oak Ridge National Laboratory (ORNL)'s PVM – a widely used message passing library – and shows evaluations of program debugging with the prototype debugger.

1. はじめに

メッセージや共有データを介して互いに通信する複数プロセスからなる多くの並列プログラムでは，メッセージの受信順序や共有データへの排他制御操作の振舞いが非決定的となる．このため，プログラムに同じ入力を与えて再実行させてもプログラムの振舞いが必ずしも同じにならない．逐次型プログラムに対して多用されているサイクリックなデバッグ手法[1]は，同じ実行動作が繰り返し得られることを前提としているので，非決定的な並列プログラムに対しては正しく適用できない．このため，非決定的な並列プログラムのデバッギングは一般に困難である．この問題に対して，Instant Replay法[2]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が，これらの提案は主に再演制御について議論しており，再演法を用いたデバッガの実現法については十分な議論がなされていない[3]．また，再演のためのオーバヘッドなど，再演法がプログラムの実行時間に与える影響について定量的な評価はほとんどなされていない．

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プログラムデバッガdbxRを提案し，その実現法を示した[4]．本稿では，dbxRと同様のデバッガモデルに基づき，メッセージパッシング型並列プログラムを対象とした並列プログラムデバッガdbxR-II[5]について議論し，その実現法を示す．dbxR-IIは，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[6]を使用した並列プログラムをデバッグ対象とする．PVMのプログラムは，共有メモリ型並列プログラムと異なり，(1) 被デバッグプロセスのアドレス空間が独立であり，(2) 受信バッファによるメッセージの蓄積を行う．このため，dbxRのデバッガモデルを適用するには，共有メモリを用いずに再演のための制御用データベースを各プロセスに

† NTT ソフトウェア研究所
NTT Software Laboratories

分散させるとともに，メッセージパッシングによる再演制御機構を実現する必要がある．また，バッファからメッセージを選択する際に生じる非決定的動作を記録して再現する機構が必要になる．本稿では，これらの要求に応える再演機構を提案する．さらに，再演のために記録すべきデータの最小化，および再演時の実行動作の並列化の，2つの要求にそれぞれ応える逐次再演モデルと並列再演モデルと呼ぶ2つの再演法を提示する．

本稿では，まず，再演法を用いたデバッガを実現する際に問題となる，最小限の再演実行法，実行停止位置の設定法および再演停止法について議論すべき点を示す．次に，dbxRのデバッガモデルをPVM上のプログラムに適用する際の問題点を示し，これら問題点を解決した並列プログラムデバッガdbxR-IIの実現法を，逐次再演モデルと並列再演モデルそれぞれについて述べる．さらに，作成したプロトタイプを用いて，プログラムの実行監視と再演制御がプログラムの実行時間に与える影響を考察し，並列再演モデルに基づくdbxR-IIの有効性を示す．

2. 再演型並列プログラムデバッガ実現上の課題

2.1 最小限の再演実行

これまで提案された再演法[2),7)]では，論理的に順序関係がある事象について，実行監視時に先に発生した事象を再演時でも先に発生させることを保証している．しかし，順序関係のない事象に対しては再演時に実行順序制御を行っていない．このため，ある命令を再演する場合には，先に実行しておくべき命令がすべて再演済みであれば，その命令を実行する．すなわち，これらの手法を用いたデバッガでは，**図1**のように，あるプロセスがブレークポイント（以下，BPと略す）で停止しても，他のプロセスは可能な限り実行を進めてしまう．このため，プログラマが新たにBPを設定しようとした命令がすでに実行済みである可能性が高く，この場合にはプログラムを始めから実行しなおさなければならない．

すべてのプロセスの適切な位置にBPを設定することで，この問題は回避できるが，プロセス数が増大した場合には，繁雑な作業となる．よって，あるプロセスがBPで停止した場合には，その停止点までの再演に必要最小限の命令だけを再演するよう，デバッガが他プロセスの実行をおさえることが望ましい．

図1は，3個のプロセスからなる並列プログラムの実行動作を表している．ノードは，再演性を保証するために記録すべき順序関係を生じさせるメッセージ送信命令（send）および受信命令（receive）の実行を表している．ノードを結ぶ細線矢印はメッセージを表しており，メッセージに付加する数字はユーザが定義したメッセージの識別子（メッセージタグと呼ぶ）である．太線矢印は各プロセスの実行順序を表している．ここでは，プロセス1からのメッセージを受信後にプロセス2がBPで停止した場合に再演するノードを網掛けで示している．

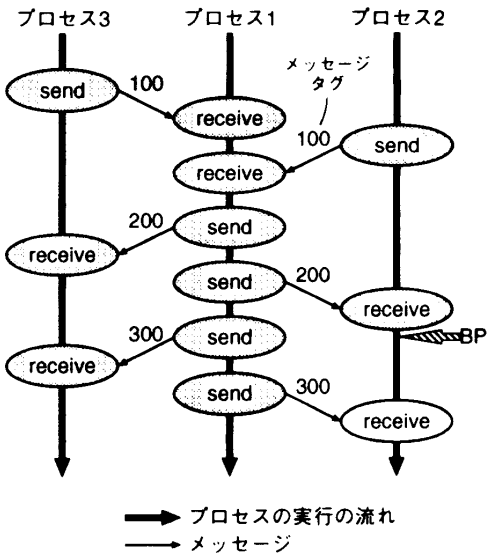

図1 従来の再演法による再演実行
Fig. 1 Re-execution with usual replay method.

2.2 実行記録上での停止位置設定

デバッグにおいてプログラマは，プログラムの実行停止位置を設定し，その実行動作の途中経過を観察する．通常のデバッガでは，静的なソースコード上の実行停止位置BPを設定する手段のみを与え，動的な実行動作上で停止位置を設定する手段は提供していない．このため，プログラマが実行動作上に停止位置を考えた場合には，その位置からソースコード上の位置を推察して，BPとして設定しなければならない．しかし，設定されたBPに再演時のプロセスが到達するとは限らず，また，1つのBPにプロセスが複数回到達する場合も生じる．一方，実行動作上に設定する停止位置では，その実行動作上に直接的に設定でき，再演時にプロセスが，ただ一度だけ到達することが保証される．通常のデバッガではプログラムの実行動作を保存していないため，実行動作上の停止位置の設定手段が課題となるが，再演法を用いたデバッガでは，実行動作の保存を行うので，これを利用して実行動作上に対しても直接的に停止位置の設定手段を提供することが望まれる．

2.3 再演停止法

デバッグでは，すべてのプロセスを停止させた後に，プログラマはプログラムの状態の観察や変更を行う．このため，停止したプログラムの状態は，つねに決定的であることが望まれるが，あるプロセスがBPに到達して停止した時点で単純に他のプロセスを停止させると，それらプロセスの停止位置が非決定的となる問題が生じる．一方，あるプロセスがBPで停止後，十分長い時間が経過すれば，全プロセスがデータ待ち状態となり，実行が進められなくなる．その時点で全プロセスを停止させれば，停止位置が決定的となるが，十分長い時間待機することは，プログラマにとって許容できない．よって，デバッガは，BPで停止したプロセスの発生後，他プロセスがもはや実行を進められない状態に推移したことを，できる限り早い時期に検出し，全プロセスを停止させることが望ましい．

3. メッセージパッシング型並列プログラムデバッガ dbxR-II

dbxR-IIは，標準的なメッセージパッシング型並列プログラミング環境であるPVM[6]上で動作する並列プログラムをデバッグ対象とする．またdbxR-IIは，dbxR[4]と同様に，先に提案した要求駆動型再演法[8]を使用し，プログラムの実行動作を記録する実行監視モードと，その記録に従ってプログラムの実行動作を再現する再演実行モードを持つ．このとき，プログラムの実行動作を記録したデータを実行記録と呼ぶ．

本章ではまず，Oak Ridge National Laboratory (ORNL) が開発したPVMの特徴をデバッガ作成の観点からまとめ，次に，dbxRのデバッガモデルをPVMプログラムに適用する際に新たに生じる実現上の課題について述べる．さらに，これら課題を解決した並列プログラムデバッガdbxR-IIの実現法と構成について述べる．

3.1 デバッグ対象プログラム

PVM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築するソフトウェアである．**図2**に示したように，各計算機上に常駐するPVMのデーモン（pvmd）が互いに通信して，アプリケーションプロセスの実行と通信を制御することにより仮想並列計算機を構成する．ここで，PVMのアプリケーションプロセスをPVMプロセスと呼ぶ．dbxR-IIがデバッグ対象とするプログラムはPVMプロセスからなり，次のような特徴を持つ．

- アプリケーションプログラムは複数のプロセスからなり，独立したアドレス空間を持つ．

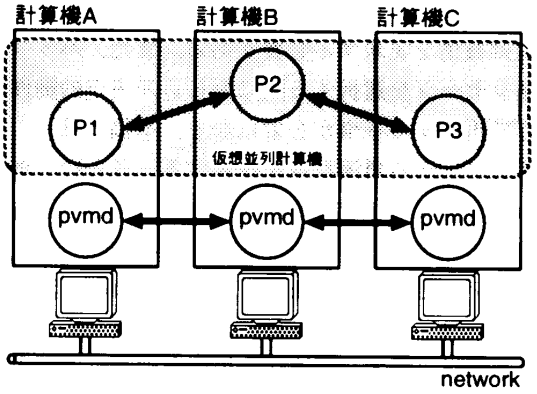

図2 PVMによる仮想並列計算機
Fig. 2 Parallel Virtual Machine.

- プロセスは，PVMのライブラリにより提供されるメッセージ通信命令によりデータ授受を行う．
- 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．
- あるプロセスが，1つのプロセスに対して複数のメッセージ送信を行った場合，送信と同一順序でメッセージが受信プロセスに到着する．
- 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．このとき，同じ送信プロセス識別子とメッセージタグを保持するメッセージが受信バッファ内に複数蓄えられている場合には，先に到着したメッセージを取り出す．

PVMのライブラリが提供するメッセージ通信命令には，プログラムの実行動作の順序関係に非決定性を生じる可能性がある命令として，1対1のメッセージ通信命令（pvm_send(), pvm_recv()），1対多のメッセージ通信命令（pvm_mcast()），同期命令（pvm_barrier()），受信バッファの状態検査命令（pvm_probe(), pvm_nrecv()）などがある．再演型デバッガは，これらの命令すべてについて再演実行制御を実現する必要がある．本稿では，その第一段階として，次のようなプログラムを対象として作成したデバッガについて議論を進める．

- PVMプロセスは，メッセージ通信命令にpvm_send(), pvm_recv()を用いる．
- プロセスの非決定的な実行動作は，受信したメッセージの内容によってのみ決定する．

上記の制約を課すことは，たとえば，ポーリングによって投機的な計算を中断するなど，受信バッファに蓄積されたメッセージによって非決定的な実行動作するプログラムなどは許さないことを意味する．このような制約を取り除く手法，すなわち，本稿で対象外としたメッセージ通信命令の再演実行制御の実現法については，5.4節で述べる．

3.2 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型デバッガの実現課題

dbxR-IIでは，メッセージの送受信順序による並列プログラムの非決定的な実行動作を再演制御の対象とする．すなわち，実行監視モードと再演実行モードにおいて，メッセージの送信および受信が同一順序で実行されるよう制御することで並列プログラムの実行動作を再現する．これは，受信バッファへのアクセス順序を実行記録に保存し，実行記録に基づいてアクセス順序を再現すればよい．このとき，PVMプログラムは各プロセスのアドレス空間が独立であり，また，受信バッファにメッセージを蓄積することから，デバッガの実現では，以下のような問題を考慮する必要がある．

(1) プロセスごとに独立したアドレス空間

PVMプログラムでは各プロセスのアドレス空間が独立であるために，送信プロセスと受信プロセスは実行記録を共有できない．すなわち，受信バッファへのアクセス履歴である実行記録を，受信プロセスのアドレス空間内に構築すると，送信プロセスは実行記録への書き込みができない問題がある．また，再演実行を制御するためには，メッセージの送信をうながすための実行要求信号と，要求したメッセージの送信終了を示す実行終了信号を管理するデータベースが必要である．PVMでは共有メモリが存在しないので，この制御用データベースを各プロセスに分散させて構築するとともに，制御用の信号をメッセージを用いて実現する必要がある．

(2) 受信バッファによるメッセージの蓄積

PVMプログラムでは，到着したメッセージは各プロセスの受信バッファ内に蓄積され，プロセスは送信プロセスの識別子とメッセージタグを指定して，受信するメッセージを能動的に選択する．このため，受信バッファへの書き込み順（すなわちメッセージの到着順）に読み出されるのではないので，プログラムの実行を再現するには，実際に読み出されたメッセージの順序を実行記録に保存する必要がある．このとき，送信プロセスとメッセージタグが同一のメッセージが複数送信された場合には，先に送信されたメッセージが先に受信されることをPVMが保証しているので，実行記録を受信プロセスごとに用意すれば，実行記録のデータは，送信プロセスの識別子とメッセージタグの対を読み出されたメッセージの順序で並べたものとなる．

3.3 再演モデル

再演型デバッガでは，定常的に実行記録を作成することを許容できるようにするために，記録すべきデータ量および記録回数をできる限り少なくする必要がある．一方，効率的に再演を行うためには，再演制御によってプログラムが持つ並列性を損なわないようにする必要がある．これら2つの要求について，それぞれをできる限り満たすようにした，逐次再演モデルと並列再演モデルと呼ぶ2つの再演法について述べる．

3.3.1 逐次再演モデル

逐次再演モデルでは，実行記録のデータ量を最小限におさえ，実行記録の作成に必要なオーバヘッドを削減する．しかし，再演を停止すべき地点を実行記録かあらかじめ得ることができないので，再演時にプロセスの実行を逐次化して，必要最小限の命令だけが実行されるように調べながら再演を進める．

3.3.1.1 実行監視モード

実行監視モードにおいて，dbxR-IIでは，受信バッファへのアクセス履歴である実行記録を受信プロセスのアドレス空間内に作成する．送信プロセスは受信プロセス内の実行記録へのアクセスはできないが，3.2節で述べたように，メッセージの受信順序を実行記録に保存すれば，プログラムの再演実行が可能となる．よって，dbxR-IIでは，受信プロセスが受信命令を実行するごとに，受信バッファから取り出したメッセージが持つ送信プロセス識別子とメッセージタグの対を，バッファから取り出した順序に従って並べた実行記録を作成する．これにより，受信プロセスが送信プロセスと競合することなく，独立して実行記録の作成が可能となり，並列プログラムが持つ並列性を損なわずに実行記録の作成が可能となる．並列プログラムが実行監視モードで，図1のように動作した場合に作成される実行記録の例を**表1**に示す．

3.3.1.2 再演実行モード

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

dbxR-IIはdbxRと同様に，ソースコード上と実行記録上の2種類の実行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で提供されるブレークポイント（BP）であり，後者はデマンドポイント（以下，DPと略す）と呼ぶ．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

表1 作成する実行記録の例
Table 1 Examples of execution record.

プロセス1の実行記録		
送信プロセス	3	2
メッセージタグ	100	100
プロセス2の実行記録		
送信プロセス	1	1
メッセージタグ	200	300
プロセス3の実行記録		
送信プロセス	1	1
メッセージタグ	200	300

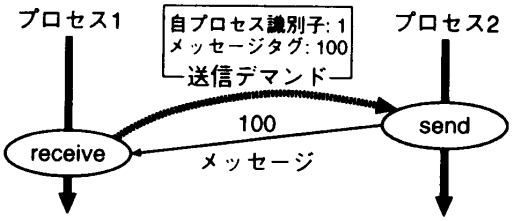

図3 要求駆動型再演法
Fig. 3 Demand-driven based replay method.

て述べる．本手法では，まずプログラマが実行記録から，送信プロセスとメッセージタグの対で表される受信メッセージの1つを選択し，DPとして指定する．デバッガは指定されたメッセージに対応する受信命令の直後の実行位置をDPと解釈し，先に提案した要求駆動型再演法を用いて，DPへの到達に必要最小限の命令のみを再演実行する．

PVMでは，各プロセスのアドレス空間が独立であるので，dbxRでの実行要求信号に対応する送信デマンドと呼ぶメッセージを用いて，要求駆動型再演法を実現する．要求駆動型再演法は次のように動作する．プロセスは受信命令を再演するごとに，受信命令に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**のように，自プロセス識別子と要求するメッセージのメッセージタグを内容とする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送信命令まで再演を進め，メッセージを送信する．待機していた受信プロセスは，到着したメッセージを受信バッファから取り出して受信命令を終了する．

このように，必要最小限の命令だけが実行されるよう調べながら実行を進めるために，DPを与えられたプロセスは，受信命令を再演するごとに新たな送信デマンドを送出して待機する．また，この送信デマンドを受信したプロセスは，送信デマンドによって指定された送信命令に到達する以前に，受信命令を再演する必要がある場合には，さらに送信デマンドを他プロセスに伝播させて，待機する．このため，逐次再演モデルでの再演実行では，ある時点で再演を進めているプロセスはただ1つとなり，プログラムの実行が逐次化される．

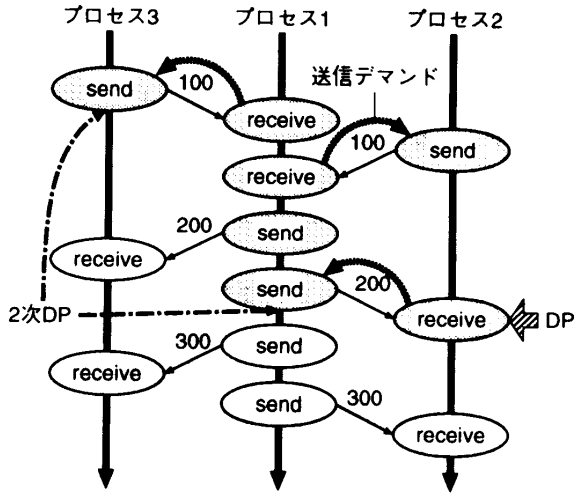

図4 要求駆動を用いた再演実行
Fig. 4 Re-execution with demand-driven based replay method.

このとき，プロセスは送信デマンド表と呼ぶデータベースを用いて，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，メッセージの到着遅延などで送信デマンドと再演したメッセージが行き違う場合にも，正しく再演できる．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とnの大小比較により次のことが分かる．

$m > n$：実行を要求されたが未再演の送信命令が存在するので再演を進める必要がある

$m < n$：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

$m = n$：送信デマンドを与えられた送信命令はすべて再演済みであり，かつ再演済の送信命令に対する送信デマンドはすべて受信している

ここで，DPを持つプロセスをDPプロセスと呼び，それ以外を非DPプロセスと呼ぶ．また，非DPプロセスにおいて，上記$m = n$が成立する送信命令の直後の実行位置をそのプロセスの2次DPと呼ぶ．2次DPは他プロセスからの送信デマンドの到着により，動的に位置を変更する．

図4は，図1において，プロセス2の1回目の受信命令にDPが設定された場合に，再演が行われる部分を網掛けで示し，送出される送信デマンドを破線矢印で示している．再演開始時には全プロセスは停止しているが，プログラマがDPを設定後，DPプロセス

であるプロセス2が再演を開始する．プロセス2は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．非DPプロセスであるプロセス1は送信デマンドを受けると再演を開始し，最初の受信命令を実行する直前にプロセス3へ送信デマンドを送出すると，プロセス3が再演を開始する．その後，プロセス3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2へのメッセージ送信までを実行して停止する．プロセス1からのメッセージを受信したプロセス2は，DPまでの再演を終了し，停止する．このように，本デバッガでは，送信デマンドを用いて各プロセスがDP，2次DPまでの再演を行うことにより，DPまでの必要最小限の命令の再演実行を実現する．

再演停止法

dbxR-IIはdbxRと同様に，DPとBPを単独あるいは併用して実行停止位置を設定でき，これら停止位置にプロセスが到達した場合には，決定的状態でのプログラム停止を保証する．

あるプロセスがDPに到達した場合には，要求駆動再演法の性質より，全プロセスが実行を進められない実行位置にあることが保証できる．また，あるプロセスがBPに到達した場合には，十分長い時間が経つと全プロセスがデータ待ちの状態となるので，その時点で停止させれば，全プロセスの停止位置は決定的となる．このような，すべてのプロセスが実行を進められない状態を安定状態と呼び，安定状態での各プロセスの実行位置をそのプロセスのSPと呼ぶ．また，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を2次BPと呼ぶ．実際のデバッグでは，あるプロセスがBPで停止後，十分長い時間待機することは許容できないため，できる限り早い時期に全プロセスのSP到達を検出し，安定状態で実行を停止させる手段が必要となる．

DP，BP，2次BPは定義より明らかにSPである．2次DPは送信デマンドを受信するたびに位置が変化するので，新たな送信デマンドを受信しないことが保証された場合にのみSPとなる．よって，プロセスがDP，BP，2次BP，または，新たな送信デマンドを受信する可能性のない2次DPのいずれかに到達したとき，そのプロセスはSPに到達したと判定できる．以下にSPへの到達判定の実現法について述べる．

再演実行中にあるプロセスがDPに到達したときには，要求駆動再演法の性質から，他のすべてのプロセスが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降送出されないことが保証できる．

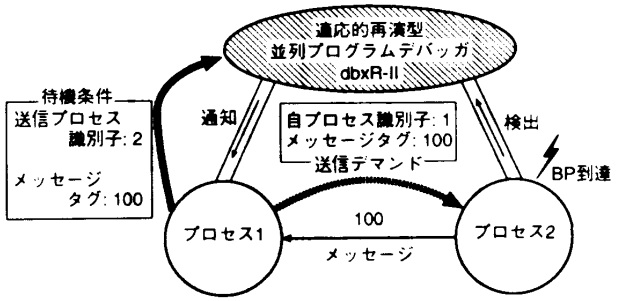

図5 2次BP到達検出法
Fig. 5 Detection of 2nd BP.

よって，すべてのプロセスが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よい．

あるプロセスがBPに到達したときには，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．プロセスの制御を行うデバッガはプロセスのBP到達を検出できるので，以下の手法を用いて，比較的少ないメッセージ数でプロセスの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，**図5**に示すように，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6は，図4と同様にプロセス2の受信命令にDPを設定したが，再演途中にプロセス1がBPに到達した場合に再演するノード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3は送信デマンドを与えられた送信命令をすべて再演した2次DPで停止する．プロセス2はプロセス1からのメッセージを待機する実行位置，すなわち受信命令の直前で2次BP到達となって停止し，DPが指定する受信命令は再演されない．

3.3.2 並列再演モデル

逐次再演モデルでは，送信デマンドの送出を受信命令の実行直前まで遅らせているために，逐次的な再演となっている．これは，1つのプロセスが実行した送信命令と受信命令の実行順序を，受信命令の履歴である実行記録から求められないので，非DPプロセスは，DPまでの最小限再演となるよう調べながら再演を進

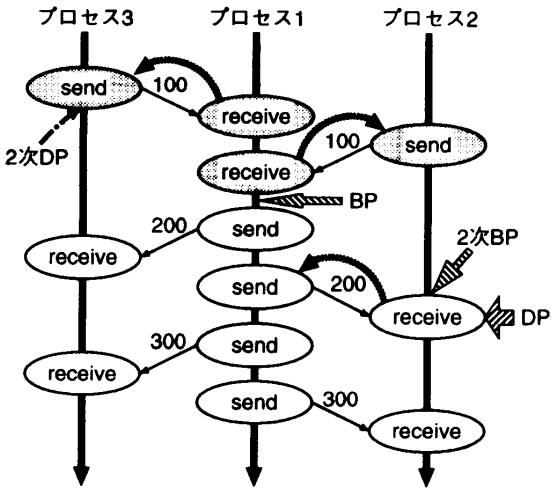

図6 DP と BP を併用した再演実行
Fig. 6 Halting the program using demand-driven replay method with DP and BP.

めるためである．これに対して，並列再演モデルでは，再演開始時に，DP に到達するまでに送出する送信デマンドをすべて送出し，各プロセスが再演すべき送受信命令を明らかにすることにより，再演実行の並列化を行う．このためには，DP に到達するまでにプログラムが再演すべき送信命令と受信命令をすべて再演開始時に求める必要がある．この場合，各プロセスにおいて，実行すべき送信命令より前に実行すべき送信・受信命令を実行記録より求められなければならない．よって，各プロセスは，実行した受信命令の順序だけでなく，送信命令の実行順序も実行記録に保存する必要がある．

3.3.2.1 実行監視モード

実行監視モードでは，プロセスは送信命令と受信命令を実行するごとに相手プロセスの識別子，メッセージタグおよび命令の種別（送信または受信）を保存する．並列プログラムが，本モデルの実行監視モードで，図1のように動作した場合に，プロセス1が作成する実行記録を，**表2** に例示する．

3.3.2.2 再演実行モード

再演実行モードでの DP プロセスは，再演開始時に，DP までに再演すべき受信命令を実行記録よりすべて求め，これら受信命令に対応する送信デマンドをすべて送出する．非 DP プロセスは，送信デマンドにより指定された送信命令を再演するまでに実行すべき受信命令を実行記録より求めて，これら受信命令に対応する送信デマンドをすべて送出した後に再演を開始する．これにより，各プロセスは再演開始時に，DP への到達に必要最小限の命令すべてに送信デマンドが与えられるため，並列的に再演実行が可能となる．

表2 並列化再演実行で作成する実行記録の例
Table 2 Examples of execution record in parallel replay.

プロセス1の実行記録						
相手プロセス	3	2	3	2	3	2
メッセージタグ	100	100	200	200	300	300
送受信種別	受信	受信	送信	送信	送信	送信

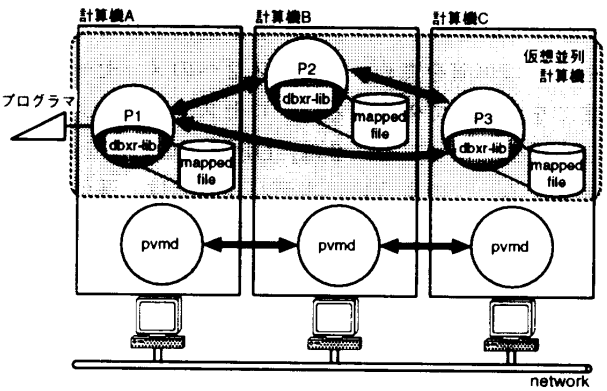

図7 実行監視モードでの被デバッグプロセスの構成
Fig. 7 Process construction of dbxR-II in monitor mode.

3.4 dbxR-II の構成

dbxR-II のプロトタイプの実現を，逐次再演モデルと再演実行モデルの両方について進めている．逐次再演モデルについては，現在，実行記録作成機能と要求駆動による最小限再演機能の実現を終え，BP を併用した場合の再演停止機能の実装を進めている段階である．並列再演モデルについては，実行記録作成機能のみを実現した段階である．また，ユーザインタフェースについては，XPVM[9),10)] を利用した GUI を実現している[11),12)]．

dbxR-II を用いてデバッグする場合，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）を加えて実行する．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う．dbxr-lib は PVM が提供するメッセージ通信と子プロセス生成のライブラリと同様のインタフェースを持つ．

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い，メッセージを受信するごとに，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．また，プロセス生成時には子プロセスの識別子をプロセステーブルに書き込む．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

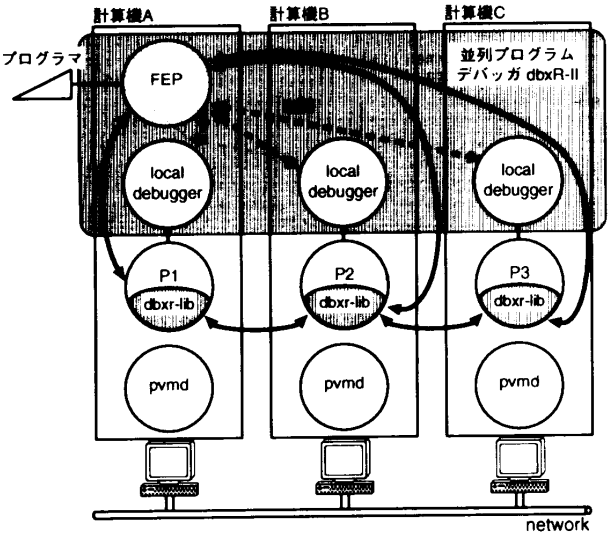

図8 再演実行モードでのdbxR-IIのプロセス構成
Fig. 8 Process construction dbxR-II in replay mode.

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う.再演実行モードでのdbxr-libは互いにメッセージを授受し,3.3.1項で述べた要求駆動再演を実現する.

4. dbxR-IIの評価実験

4.1 実験方法

dbxR-IIの基本特性を明らかにするために,2種類のテストプログラムを用いて,dbxR-IIを用いない通常の実行の場合,実行監視モードで実行させた場合,再演実行モードで実行させた場合の3種類に分けて実行時間を測定し,分析した.実行監視モードでは,さらに,逐次再演モデルの場合と再演実行モデルに分けて測定を行った.この実験では,**図9**のように2種類のネットワークに接続された最大6台のUNIX計算機を用いて,PVMの仮想並列計算機を構築した.これらの計算機は端末として日常的に使用されており,様々な負荷がかかっている.テストプログラム1は,引数で与えられた個数のプロセッサを一周するメッセージ通信を50回繰り返す.**図10**はプロセッサ数が3の場合のテストプログラム1の動作を示している.テストプログラム2は,NASAで開発された並列ベンチマークプログラム[13]であり,引数で与えられた個数のプロセッサを用いて65536個の整数を整列させる.

テストプログラム1を用いた実験では,プログラム

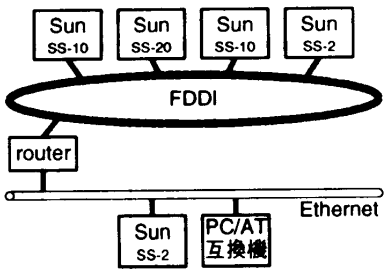

図9 評価実験環境
Fig. 9 Processors construction in examination.

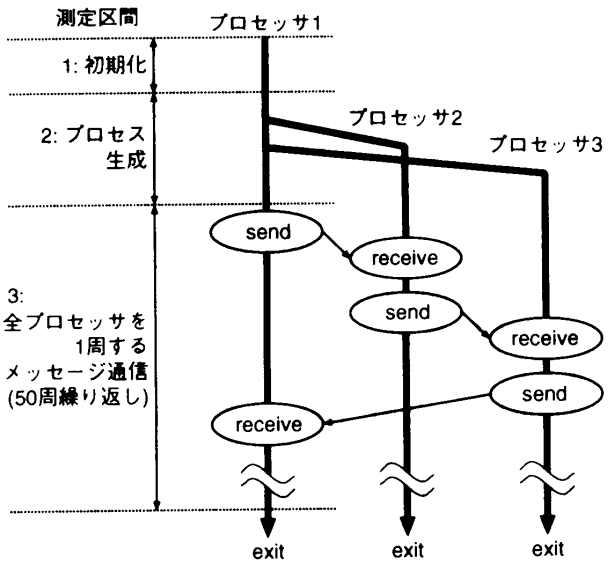

図10 テストプログラムの動作
Fig. 10 Execution behavior of test program.

を以下のnormal, monitor1, replayの3種類のモードで動作させ,図10のように,実行時間を3区間に分けて測定した.テストプログラム2を用いた実験では,プログラムを以下のnormal, monitor1, monitor2の3種類のモードで動作させ,ベンチマーク結果を測定した.

- **normal:** dbxR-IIシステムを組み込まず,PVMライブラリが提供するメッセージ送受信命令を使用する.実行記録の作成は行わない.
- **monitor1:** 逐次再演モデルのdbxR-IIシステムが提供するメッセージ送受信命令を実行監視モードで使用し,受信命令の実行順序を実行記録に保存する.
- **monitor2:** 並列再演モデルのdbxR-IIシステムが提供するメッセージ送受信命令を実行監視モードで使用し,送信命令と受信命令の実行順序を実行記録に保存する.
- **replay:** 逐次再演モデルのdbxR-IIシステムが提供するメッセージ送受信命令を再演実行モード

表3 テストプログラムの実行時間の平均値（ミリ秒）
Table 3 Execution time of test program.

プロセッサ台数	実行モード	測定区間 1	測定区間 2	測定区間 3
2	normal	115.8	20.8	186.2
	monitor	129.9	20.8	190.1
	replay	132.8	1115.4	367.7
3	normal	112.9	79.6	420.9
	monitor	125.6	61.9	422.6
	replay	131.1	12321.8	2358.2
4	normal	138.1	86.8	538.8
	monitor	118.7	119.3	535.8
	replay	133.0	15594.5	3909.0
5	normal	113.0	141.0	795.7
	monitor	129.0	129.6	804.9
	replay	134.3	20833.1	5991.0
6	normal	126.2	174.5	953.1
	monitor	118.9	149.2	1002.3
	replay	124.6	23340.3	8177.2

で使用し，monitor1 モードで作成した実行記録に従って再演実行を行う．

4.2 実験結果

4.2.1 測 定 1

normal, monitor1, replay の各モードについて，テストプログラム 1 をプロセッサ数が 2～6 それぞれの場合について，各測定区間の実行時間を 100 回ずつ測定した．100 回の測定結果の平均値（ミリ秒）を**表 3** に示す．測定区間 1（初期化）では，各モードともほぼ同じ値となっている．測定区間 2（プロセス生成）では，normal と monitor1 はほぼ同じ値で 20～200 ミリ秒程度であるが，replay では 1～20 秒程度要している．replay でのプロセス生成時間が大きいのは，プロセッサと同数のローカルデバッガを生成することが原因である．測定区間 3（50 周のメッセージ通信）では，normal と monitor1 はほぼ同じ値である．replay の値は normal や monitor1 に比べて 2～9 倍程度大きな値となり，プロセッサ数の増大とともに差は大きくなっている．monitor1, replay の平均値と normal の平均値の比較を**図 11** に示した．横軸はプロセッサ数を示し，縦軸は normal の平均値を 1 とした場合の monitor1, replay の平均値の倍数を示している．プロセッサ数 2～4 では replay の平均値はプロセッサ数に比例して大きくなるが，プロセッサ数 5～6 では比較的なだらかに増大している．

また，逐次再演モデルでは実行記録作成のために，表 1 に示したように，メッセージの受信ごとに送信プロセスの識別子とメッセージタグを保存する．並列再演モデルでは，表 2 で示したように，メッセージ送信時と受信時に相手プロセス識別子，メッセージタグ，

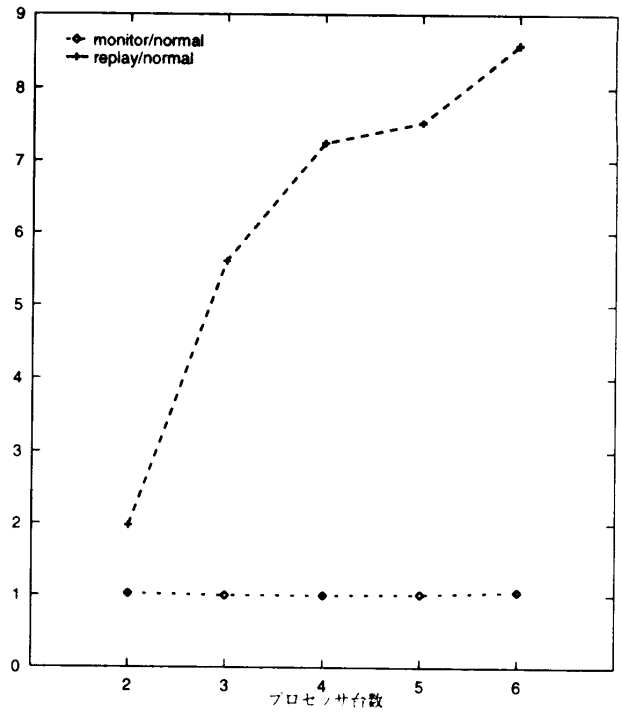

図 11 monitor, replay と normal の平均値の比較
Fig. 11 Comparison of execution time.

および送受信種別を保存する．このとき，dbxR-II では 1 回の送信または受信に対して，プロセス識別子の保存に 4 byte，メッセージタグの保存に 4 byte，送受信種別の保存に 1 byte のデータ量をそれぞれ必要とする．テストプログラム 1 の実行開始から終了まででは，各プロセッサは，メッセージ送信と受信を各々51 回行うので，実行記録として monitor1 では約 400 byte, monitor2 では約 1 Kbyte のメモリが必要となる．

4.2.2 測 定 2

逐次再演モデルの実行記録作成に必要なオーバヘッド時間をより詳しく測定するため，プロセッサ数が 6 の場合の normal と monitor1 について，テストプログラム 1 の測定区間 3 の実行時間を，100 回を 1 セットとして 5 セット測定した．測定結果を**図 12** に示す．図の横軸はセット数を表し，縦軸はメッセージが 50 周するのに要した時間（ミリ秒）を表している．また，各モードについて 1 つの点は 1 回の測定結果を示し，100 回の平均値を横棒で示している．図 12 では，normal, monitor1 の値はともに，ばらつきが大きく，その平均値はほぼ同じとなっている．

4.2.3 測 定 3

逐次再演モデルと並列再演モデルの実行記録作成に必要なオーバヘッド時間を比較するため，プロセッサ数が 4 の場合の normal, monitor1 と monitor2 について，テストプログラム 2 をそれぞれ 400 回ずつ実

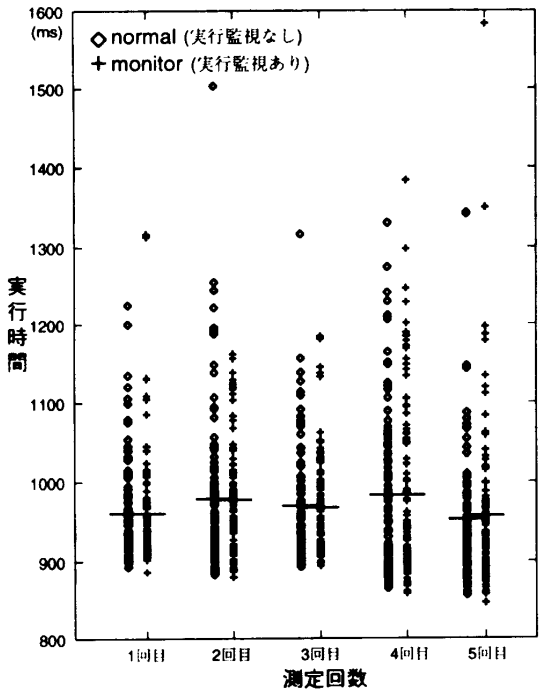

図12 プロセス数6の場合の測定区間3の実行時間
Fig. 12 Spread of execution time.

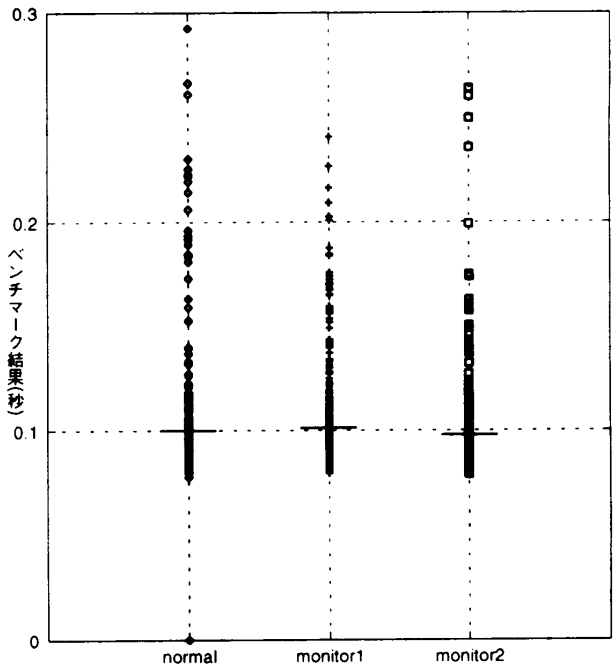

図13 ベンチマーク結果
Fig. 13 Results of the sort benchmark programs.

行し，測定した．測定結果を**図13**に示す．図の横軸はモードを示し，縦軸はベンチマーク結果（秒）を表している．また，各モードについて1つの点は1回の測定結果を示し，400回の平均値を横棒で示している．図13では，normal，monitor1，monitor2の値はともに，ばらつきが大きく，その平均値はほぼ同じとなっている．

また，各プロセッサはメッセージ送信と受信を各々11回行うので，monitor1では約90 byte，monitor2では約200 byteのメモリを実行記録の作成に使用する．

5. 考　　察

5.1 実行監視モードでのオーバヘッド

逐次再演モデルの実行監視モードでは，受信命令を実行する際，PVMのメッセージ受信に比べて，受信したメッセージから送信プロセスの識別子とメッセージタグの値を取り出して実行記録に保存する操作が加えられている．また，並列再演モデルの実行監視モードではさらに，送信命令を実行する際に，送信先プロセスの識別子とメッセージタグの値を実行記録に保存する操作が加えられている．これらの実行記録作成に必要な操作によるオーバヘッドは極力小さくおさえる必要性がある．図12および図13では，normal，monitor1，monitor2のいずれの場合でも，測定値のばらつきはほぼ同様である．このため，実行記録作成のオーバヘッドは，実行時の環境の違いにより生じるプログラム実行時間のばらつきに比べて十分小さいので，定常的に監視モードでプログラムを実行させても，実行時間の点で実用上問題ないといえる．

5.2 再演実行モードでのオーバヘッド

再演実行モードでは，通常実行および監視実行に比べて，次のような要因により実行時間が増大する可能性がある．

(a) メッセージ数の増大によるオーバヘッド
(b) 再演制御用メッセージの送受信順序の逐次化
(c) 再演制御によるデバッグ対象プログラムの逐次化

以下に各要因について，テストプログラム1の再演実行を例に考察を加える．

(a) 逐次再演モデルの再演実行モードでは，実行監視モードでの1回のメッセージ送受信を再演するのに，送信デマンドを1つ送出することから，メッセージ数が2倍になっている．これは，図11でプロセス数が2の場合に，replayの平均値がnormalの平均値に比べ約2倍になっていることからも確認できる．

(b) 逐次再演モデルでは，送信デマンドの送出を受信命令の実行直前まで遅らせているために，送信デマンドの送出が逐次的となる．このため，送信デマンドの伝播遅延が加算されていくことになるので，並列に送信デマンドを送出してよい場合に比べて，プログラム実行時間が増大する．プロセッサ台数が増大すると，送信デマンドが伝播する段数が増えるため，伝播遅延が増大し，問題が深刻になる．

(c) テストプログラム1では送信命令と受信命令を繰り返す．実行監視モードにおいて，各プロセッサは，受信命令の実行時にメッセージ到着の待ち合わせを行い，送信命令の実行時には待ち合わせを行わない．たとえば図10のプロセッサ1とプロセッサ2のような，メッセージ授受を行う隣あったプロセッサでは，プロセッサ1の送信命令から次の受信命令の区間と，プロセッサ2の受信命令から次の受信命令までの区間は並列に実行可能である．これに対して，逐次再演モデルの再演実行モードでは，図10のプロセッサ1は送信命令を再演後，新たな送信デマンドが到着するまで実行を進められない．プロセッサ2はプロセッサ1からのメッセージを受信後，再演を継続し，新たな受信命令を再演する際にプロセッサ1へ送信デマンドを送る．プロセッサ1は新たな送信命令までの再演を再開するが，このとき，プロセッサ2はプロセッサ1からのメッセージが到着するまで，実行を進められない．このように，逐次再演モデルの再演実行モードでは実行が逐次化される．このため，プロセス数が多いほど失われる並列性が顕著となる

図11において，監視モードでは通常実行と実行時間がほぼ同程度であるのに対して，再演実行ではプロセッサ台数が増えると実行時間が急激に増えている．この原因は上記 (b), (c) によるものと考えられる．

テストプログラム1は逐次的な部分が多いので，要因 (b), (c) の影響を詳細に分析するためには，今後，並列性の高いプログラムについても評価を進める必要がある．

5.3 逐次再演モデルと並列再演モデルの比較

逐次再演モデルと並列再演モデルでは，実行記録のデータ量が2倍になっているが，図13ではmonitor1, monitor2のいずれの場合でも測定値のばらつきは，ほぼ同様となり，両モデルでの実行記録作成のオーバヘッドの差は，実行時の環境の違いにより生じるプログラム実行時間のばらつきに比べて十分小さいといえる．また，図11から，逐次再演モデルではプロセス数が増大するとともに，再演実行時間が増大することが明らかとなった．並列再演モデルでは，実行記録のデータ量増大による実行記録作成のオーバヘッド時間への影響は実用上問題なく，プログラムが持つ並列性を引き出すことによる再演実行時間の短縮効果が期待できることから，dbxR-IIの実現においては，並列再演モデルが有効であるといえる．並列化再演による再演実行時間の短縮効果については定量的な評価が必要であるが，これについては今後の課題とする．

5.4 PVMにおける非決定的な命令

本稿では，再演時の実行を制御すべき，プログラムの実行動作に非決定性を生じさせる命令として，メッセージ通信命令pvm_send(), pvm_recv()だけを対象に再演法を議論した．しかし，PVMでは次のような命令でも実行動作の順序関係に非決定性が生じる可能性がある．

(1) 1対多のメッセージ通信命令

pvm_mcast()： 指定した複数プロセスにメッセージの同報を行う

pvm_bcast()： プロセスグループを指定し，属する全プロセスにメッセージの同報を行う

(2) 同期命令

pvm_barrier(): プロセスグループを指定し，属する全プロセスがpvm_barrier()を実行するまで待機する

(3) 受信バッファの状態検査命令

pvm_probe(): 指定した送信プロセスの識別子とメッセージタグを持つメッセージが，受信バッファに到着しているか検査し，到着している場合には真を返し，到着していない場合には偽を返す

pvm_nrecv(): 指定した送信プロセスの識別子とメッセージタグを持つメッセージが，受信バッファに到着しているか検査し，到着している場合にはメッセージを受信し，到着していない場合には偽を返す

以下に，これらの命令について再演実行制御の実現法を考察する．

(1) は，1対1の送信命令の繰返しとして，pvm_send(), pvm_recv()と同様の実行制御を繰り返せばよい．

(2) は，グループサーバと呼ぶPVMプロセスを用いて実現されている．pvm_barrier()を実行するプロセスはグループサーバにメッセージを送り，グループサーバからのメッセージを待機する．グループサーバは指定されたプロセスグループに属するプロセス数と同数のメッセージを受信した後に，pvm_mcast()を用いてプロセスグループに属するプロセスにメッセージの同報を行っている．よって，被デバッグプロセスとグループサーバ・プロセスの両方に対して，pvm_send(), pvm_recv()と同様の実行制御方法を行えばよい．

(3) は，実際のメッセージ受信をともなわない．実行監視モードでは，受信バッファの検査結果の返り値を実行記録に保存し，再演実行モードでは，実行監視時と同一の検査結果を返すことにより，再演実行が実現できる．

6. おわりに

本稿では，共有メモリ型並列プログラムを対象とした再演法を用いた並列プログラムデバッガdbxRを，メッセージパッシング型並列プログラムであるPVMプログラムに適用する際の課題を述べ，さらに，実行記録のデータ量と再演実行時の並列性の観点から，逐次再演モデルと並列再演モデルの2種類のデバッガモデルを提案した．そして，実現したプロトタイプを用いて，デバッグ時のプログラムの実行時間を測定し，逐次再演モデルと並列再演モデルはともに，実行記録の作成が比較的小さなオーバヘッドで実現され，定常的に実行監視モードでプログラムを実行させても実行時間の点で実用上問題ないことを示した．特に，逐次再演モデルと並列再演モデルにおいて，実行記録作成がプログラム実行時間に与える影響の差が十分小さいので，必要最小限の命令だけを再演することを保証しながら再演時の並列性を向上させるという点で，並列再演モデルが有効性であることを示した．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と，並列再演モデルのプロトタイプの実現を進め，再演実行の並列化による再演実行時間の短縮効果について定量的評価を進めていく予定である．

謝辞 日頃，ご指導ご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) Fairley, R.: *Software Enginnering Concepts*, McGraw-Hill (1985).
2) LeBlanc, T.J. and Mellor-Crummey, J.M.: Debugging Parallel Programs with Instant Replay, *IEEE Trans. Comp.*, Vol.C-36, No.4, pp.471–482 (1987).
3) 山田 剛：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, pp.1170–1178 (1993).
4) 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，情報処理学会論文誌，Vol.36, No.7, pp.1589–1599 (1995).
5) 三栄 武，高橋直久：適応的再演型並列プログラムデバッガのPVM上での実現，信学技報，Vol.94, No.383, CPSY94-81, pp.73–80，情処研報，Vol.94, No.106, 94-OS-67-10, pp.73–80 (1994).
6) Geist, G.A. Beguelin, A., et al.: PVM3 User's Guide and Refernce Manual, ORNL/TM-12187 (1993).
7) Carver, R. and Tai, K.: Reproducible Testing of Concurrent Programs Based on Shared Variable, *Proc. 6th Int. Conf. Dist. Comp. Sys.*, pp.428–433 (1986).
8) 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).
9) Kohl, J.A. and Geist, G.A.: XPVM 1.0 User's Guide, ORNL/TM-12981 (1995).
10) Kohl, J.A., Geist, G.A., et al.: The PVM 3.4 Tracing Facility and XPVM 1.1, http://www.epm.ornl.gov/pvm/trace.ps (1995).
11) 三栄 武，高橋直久：要求駆動型再演機能を持つPVMプログラムデバッガXdbxR/PVM，信学技報，Vol.95, CPSY95-45, pp.25–32 (1995).
12) Miei, T. and Takahashi, N.: XdbxR/pvm: A PVM Program Debugger with Demand-Driven Replay, *EuroPVM '95*, pp.185–190 (1995).
13) Sukup, F.: *Efficiency Evaluation of Some Parallelization Tools on a Workstation Cluster Using the NAS Parallel Benchmarks*, ACPC/TR 94-2 Computing Center, Vienna University of Technology (1994).

(平成7年9月4日受付)
(平成8年3月12日採録)

三栄 武（正会員）

昭和40年生．昭和62年名古屋工業大学工学部情報工学科卒業．平成元年同大学院工学研究科電気情報工学専攻博士前期課程修了．現在NTTソフトウェア研究所勤務．並列プロセススケジューリング，並列仮想計算機，並列プログラム実行環境，広域ネットワークコンピューティングなどの研究に従事．

高橋 直久（正会員）

昭和26年生．昭和49年電気通信大学応用電子卒業．昭和51年同大学院修士課程修了．同年日本電信電話公社武蔵野電気通信研究所入所．以来，機能分散型並列計算機，データフロー型計算システム，関数型プログラミング，ソフトウェア・リエンジニアリング，広域ネットワーク・コンピューティングなどの研究に従事．現在，NTTソフトウェア研究所超並列プログラミング研究グループリーダ．工学博士（東京工業大学）．電子情報通信学会，日本ソフトウェア科学会，IEEE-CS，ACM各会員．