システムソフトウェアと
オペレーティング・システム 64－8
マルチメディア通信と分散処理 65－8
（1994．5．19）

# プログラムスライシングの分散プログラムへの適用

太田 剛，渡辺 尚，水野忠則

静岡大学 工学部 情報知識工学科
〒432 静岡県浜松市城北 3-5-1

分散プログラムのデバッグにおいては，従来の逐次プログラムにはなかった，実行の非決定性，デバッグするための情報の増大，すべてのプロセスが共通に使うことのできる絶対時間の欠如といった難しさが存在する．一方，逐次プログラムのデバッグ技術として，バグの存在範囲を特定するためのプログラムスライシング技術が研究されてきた．本稿では，プログラムスライシングを分散プログラムに適用することによって上記の難しさを克服しようとしたときに生じる組み合わせ爆発問題を明らかにし，その一解決法を述べる．

# An Application of a Program Slicing Technique to Distributed Programs

Tsuyoshi Ohta, Takashi Watanabe, and Tadanori Mizuno

Department of Computer Science, Faculty of Engineering, Shizuoka University
3-5-1 Johoku, Hamamatsu, Shizuoka, 432 Japan

In the case of debugging distributed programs, there are some difficulties caused by a non-determination of execution, a mass of information for debugging, and a lack of a global clock for every processes. Meanwhile, a program slicing technique has been studied for debugging sequential programs to decide a set of statements that contains a bug. In this paper, we describe (1) a combination explosion problem in the case we apply a program slicing technique to distributed programs and (2) one of the solutions to the problem.

## 1 まえがき

近年，ネットワークによって接続されたワークステーション群やマルチプロセッサ型ワークステーションが身近に利用できるようになり，それにつれて分散処理環境が手軽に利用できるようになってきた．しかし，そういった環境のもとで 稼働する分散プログラムのデバッグには，従来のシングルプロセッサ上の逐次型プログラム開発にはなかった難しさが存在する．それは，

(1) 実行のたびにプログラムの挙動が変化する非決定性の存在

(2) プロセス間で受け渡される情報や共有資源の使用状況等の，デバッグするために必要な情報の増大

(3) 分散プログラム内の全プロセスが共通に使用できる絶対時間の欠如

等に起因する [5, 17]．このような難しさに対して，これまでに以下のような手法が提案されている．

再現法 [4, 6, 13]
プログラム実行中の記録を収集・保存し，後にそれを用いて分散プログラムの動作を再現することによってデバッグを行なう．これにより，非決定性のあるプログラムの動作を決定的動作の上でデバッグできることになる．ただし，実行記録を取る行為自体がプログラムの実行に余分な遅延をもたらすことになり，プログラムの挙動を変えてしまう恐れがある．これをプローブ効果と呼ぶ [5]．

視覚化法 [9, 2]
大量のデバッグ情報を単純なテキスト形態だけで扱うことは人間にとって大きな負担となる．そこで，これらの情報を図的表現を活用して視覚化することにより，プログラムの挙動を把握し易くする．

イベントベース法 [1, 7, 3]
逐次プログラムにおけるイベントは全順序関係によって時間順に整列させることができるため，因果関係の把握は比較的容易である．しかし，分散プログラムにおいてはプロセスが同時並行的に動作するため，異なるプロセスにおける 2 つのイベント間の順序関係は，プロセス間通信における発信と受信との関係に基づいた半順序関係にしか成り得ない．そこで，半順序関係に基づいた方式を用いなくてはならない．

一方，逐次型プログラムにおけるデバッグ技術として，プログラムスライシングが研究されてきた [10, 11]．プログラムスライシングとは，プログラム中のある文の実行に影響を与える全ての文を抽出するための技術であり，抽出された文の集合をスライスと呼ぶ．スライスは，プログラム中の変数に対する値の定義および値の使用に基づくデータ依存関係，さらに，if 文や while 文の条件節の真偽に依存して文が実行され得ることを示す制御依存関係の 2 種類の依存関係をたどることによって得られる．

ある変数の値が間違っていることが判明した場合には，その変数に値を定義した最後の文において，その変数に関するスライスを求める．このスライスには，その変数の値に影響を与える可能性のある文すべてが含まれているので，この中に誤りがあることがわかる．このことを利用したバグ同定法の提案 [14, 8, 16] や，デバッグシステムの提案 [12] がなされている．

本稿では，まず 2 節においてプログラムスライシングを紹介する．これを分散プログラムに適用する場合の問題点を 3 節において明らかにし，その一解決法を 4 節に述べる．最後に 5 節において今後の課題をまとめる．

なお，分散プログラムには逐次プログラムには含まれない要素として，プロセス間通信の概念がある．本稿では，プロセス間通信の手段として同期式メッセージパッシング方式を用いる．同期式メッセージパッ

シング方式による情報の受け渡しは，プロセス間にまたがる代入文と考えることができ，逐次プログラムにおけるプログラムスライシングとの親和性が良く，特殊な解釈を必要としないためである．

## 2 プログラムスライシング

プログラムスライシングとは，プログラム中のある文の実行に影響を与える全ての文を抽出するための技術であり，抽出された文の集合をスライスと呼ぶ．スライスには静的スライスと動的スライスとがある．静的スライスはプログラムテキスト上の情報だけを用いて構成するスライスであり，動的スライスは実行時情報を用いて構成するスライスである [15]．ここでは静的スライスに関して説明する．

スライスを構成するためには，次に示す2種類の依存関係を用いる．

データ依存関係 (data dependency)
変数への値の代入によって定義される依存関係である．すなわち，代入の左辺に現れる変数の値は右辺に現れる変数の値に影響を受ける．

制御依存関係 (control dependency)
if や while 等の制御文における条件節の真偽によって定義される依存関係である．すなわち，条件節が真であれば本体 (に含まれる代入文) が実行されるが，偽であれば実行されないことによって，変数の値に影響を及ぼす．

図1に示すプログラム例を用いてプログラムスライシングを説明する．今17行めの出力文において，total の値が意図していたものと異なっていたものとする．このとき，17行めの total におけるスライスは図2のようになる．

すなわち，データ依存関係を用いることによって，変数名 total を左辺に持つ代入文が2つ (10行め，15行め) 抽出され，さらに，15行めの右辺に現れる変数 x と y に値を設定している9行めの文が抽出さ

```
 1 main()
 2 {
 3   int  x;
 4   int  y;
 5   int  z;
 6   int  total;
 7   int  sum;
 8
 9   scanf("%d %d", &x, &y);
10   total = 0;
11   sum = 0;
12   if (x <= 1) sum = y;
13   else {
14     scanf("%d", &z);
15     total = x * y;
16   }
17   printf("%d %d", total, sum);
18 }
```

図 1: プログラム例

```
 1 main()
 2 {
 3   int  x;
 4   int  y;
 5
 6   int  total;
 7
 8
 9   scanf("%d %d", &x, &y);
10   total = 0;
11
12   if (x <= 1)
13   else {
14
15     total = x * y;
16   }
17   printf("%d %d", total, sum);
18 }
```

図 2: 17行め total における静的スライス

れる．また，15行めの代入文は12行めの if 文における else 節内にあるため，制御依存関係によって12行めの条件節が抽出される．このようにして抽出された文はすべて，17行めの total の値に影響を与える可能性のある文である．

## 3 問題点

分散プログラムは，複数のプロセスがプロセス間通信を用いてお互いの情報を伝達しながら動作する．分散プログラムにプログラムスライシングを適用するために，この動作を変数の定義と使用の観点から見ることにする．すると，送信動作は情報を保持している変数を使用することであり，受信動作は変数に値を定義することである．すなわち，送信動作はプロセスをまたがって行なわれる代入の右辺，受信動作は左辺に相当すると解釈できる (図 3)．

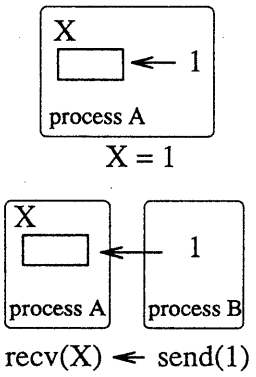


図 3: プロセス間通信と代入文

しかしながら，プロセスをまたがる代入は逐次プログラムにおける代入文とは異なり，左辺と右辺とが固定的に一対一に対応しているとは限らない．これは，分散プログラムのデバッグにプログラムスライシング技術を用いる場合に，以下に述べる組み合わせ爆発問題を招く原因となる．

異なる送信動作から送信したデータを同一の受信動作で受け取る場合を考える (図 4)．この場合，受信データを保持する変数 x が送信データを保持する変数 a および b のどちらに依存しているかについては，テキスト上で一意には決定できない．このような場合にスライスを構成するためには，2つの送信動作の両方に関して別々に解析をしなくてはならない．さらに，この送信データが以前の受信データを基に計算され，その受信動作が複数の送信動作と対応するといったことが繰り返されると，これは容易に組み合わせ爆発を起こすことになる．

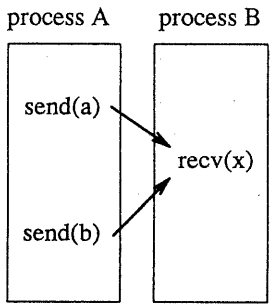


図 4: プロセス間通信例 (1)

また，ひとつのサーバが共有リソースを管理し，これを複数のクライアントプログラムが獲得しようと競合している状況を考える (図 5)．この場合，サーバがどのクライアントの要求を受け付けるかに関して非決定性が存在する．すなわち，要求受け付けの受信動作が，複数ある送信動作のどれに実際対応するのかは，実行してみなくてはわからない．このような場合にスライスを構成するためには，前述の例と同様に複数の送信動作のすべてに関して別々の解析を必要とし，これも容易に組み合わせ爆発を起こす．

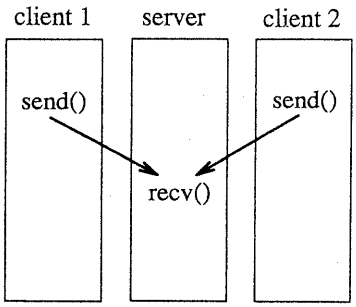


図 5: プロセス間通信例 (2)

## 4 解決法

以上の考察から，分散プログラムに対して単純にプログラムスライシングを適用することは，組み合わせ爆発を招くことがわかる．この問題は，プログラムテキスト上では受信動作に対応する送信動作を一意に決定することができないことに起因している．しかし，実動プログラム上ではこの対応が一意に決まっているので (そうでなければプログラムは動作しない)，この対応を実行中に記録し，スライスを求める時にこの記録を利用することを考える．すなわち，動的スライスを用いて組み合わせ爆発を抑える．

そのためには，次の 2 機能が必要となる．

(1) プログラム中の送信動作すべてに識別子を与え，データ送信の際にその識別子を付加して送る．

(2) プログラム中の受信動作のすべてにおいて，データに付加された上記の識別子を記録することによって，そのデータがどの送信動作から来たものであるかを記録する．

これらの機能は，プリプロセッサならびに送受信ライブラリによって吸収できるので，ソースプログラム自体は変更の必要がない．しかしながら，この機能を付加することによってプローブ効果が生じる恐れがある．そのため，実際のデバッグシステム構築にあたっては，これを十分に注意する必要がある．

## 5 まとめ

本稿では，逐次プログラムのデバッグ技術として提案されたプログラムスライシングを分散プログラムに適用するときの問題点を明らかにし，その一解決法について述べた．

同期式メッセージパッシング方式をプロセス間通信の手段として用いる場合，これはプロセスをまたがる代入と考えることができるためプログラムスライシングとは親和性が良い．しかし，プロセスをまたがる代入の左辺 (受信動作) と右辺 (送信動作) とがプログラムテキスト上で一対一に対応していないことに起因する組み合わせ爆発の問題が，スライス構成時に生じる．これを解決するためには，プログラム上のすべての送信動作に一意の識別子を与えてデータ送信時にこれを付加して送り，受信動作ではこの識別子を記録することによって，プログラム実行時に一対一の対応を把握する．この記録を用いてプログラムスライシングを行なうことによって組み合わせ爆発の問題に対処し，分散プログラムにプログラムスライシング技術を適用できるようになる．ただし，このような記録を取ることによるプローブ効果には十分な注意を要する．

本稿ではプロセス間通信の手段として同期式メッセージパッシングを取り上げたが，非同期通信や共有メモリによるプロセス間通信方式は，単純にプロセスをまたがる代入と解釈することはできず，今後の検討が必要である．

## 参考文献

[1] Bates, P. and Wileden, J.C.: "An Approach to High-Level Debugging

of Distributed Systems," *ACM SIGSOFT/SIGPLAN Proc. of Software Engineering Symposium on High-Level Debugging*, pp.107–111 (1983).

[2] Hough,A.A. and Cuny,J.E.: "Initial Experiences with a Pattern-Oriented Parallel Debugger," *ACM SIGPLAN/SIGOPS Proc. of Workshop on Parallel and Distributed Debugging*, pp. 195–205 (1988).

[3] Fidge,C.J.: "Partial Orders for Parallel Debugging," *ACM SIGPLAN/ SIGOPS Proc. of Workshop on Parallel and Distributed Debugging*, pp. 183–194 (1988).

[4] LeBlanc,T.J. and Mellor-Crummey, J.M.: "Debugging Parallel Programs with Instant Replay," *IEEE Trans. Comput.*, Vol. C–36, No. 4, pp. 471–482 (1987).

[5] McDowell,C.E. and Helmbold,D.P.: "Debugging Concurrent Programs," *ACM Computing Surveys*, Vol. 21, No. 4, pp. 593–622 (1989).

[6] Miller,B.P. and Choi,J.-D.: "A Mechanism for Efficient Debugging of Parallel Programs," *ACM SIGPLAN/ SIGOPS Proc. of Workshop on Parallel and Distributed Debugging*, pp. 141–150 (1988).

[7] Rosenblum,D.S.: "Specifiying Concurrent Systems with TSL," *IEEE Software*, Vol. 8, No. 3, pp. 52–61 (1991).

[8] Shimomura,T.: "Critical Slice-Based Fault Localization for Any Type of Error," *IEICE Trans. on Inf. & Syst.*, Vol.E-76-D, No.6, pp.656–667 (1993).

[9] Socha,D.,Bailey,M.L., and Notkin,D.: "Voyeur: Graphical Views of Parallel Programs," *ACM SIGPLAN/ SIGOPS Proc. of Workshop on Parallel and Distributed Debugging*, pp. 206–215 (1988).

[10] Weiser,M.: "Programmers Use Slices When Debugging," *Comm. of the ACM*, Vol. 25, No. 7, pp. 446–452 (1982).

[11] Weiser,M.: "Program Slicing," *IEEE Trans. on Softw. Eng.*, Vol. SE–10, No. 4, pp. 352–357 (1984).

[12] Weiser,M. and Lyle,L.: "Experiments on Slicing-Based Debugging Aids," *Empirical Studies of Programmers*, Ablex Publishing Corp., pp. 187–197 (1986).

[13] Wittie,L.D.: "Debugging Distributed C Programs by Real Time Replay," *ACM SIGPLAN/ SIGOPS Proc. of Workshop on Parallel and Distributed Debugging*, pp. 57–67 (1989).

[14] 下村隆夫: "変数値エラーにおける Critical Slice に基づくバグ究明戦略," 情報処理学会論文誌, Vol. 33, No. 4, pp. 501–511 (1992).

[15] 下村隆夫: "Program Slicing 技術とテスト, デバッグ, 保守への応用," 情報処理, Vol. 33, No. 9, pp. 1078–1086 (1992).

[16] 下村隆夫: "Critical Slice の拡張と分割検証の定式化," 情報処理学会論文誌, Vol. 34, No. 11, pp. 2401–2411 (1993).

[17] 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," 情報処理, Vol. 34, No. 9, pp. 1170–1178 (1993).