イベント系列の並べ替えによる並列プログラムテスティング*

阿部 真也†

† 地方独立行政法人 東京都立産業技術研究センター

1　はじめに

並列プログラムのテストは，実行の非決定性により逐次プログラムと比較して難しいことが知られている[1]。実行の非決定性とは，各々のプロセスあるいはスレッドが非同期に動作し，実行ごとに異なるイベント系列が発生することをいう。テストの難しさは，発生する可能性のあるすべてのイベント系列をテスト時に発生できないことにある。

テスト時に多様なイベント系列を発生させる仕組みとして，ConTest[2] が提案されている。ConTest は並列 Java プログラムにおいて，スレッド間の相対順序が実行結果に影響を与える可能性のあるイベントの前時点でコンテキストスイッチをランダムに発生させる。文献 [2] でバグ検出率が向上することが実証されている。だが，Java VM 上で実行可能なプログラムでなければならず，なおかつプロセスによる並列化はサポートされていない。

より汎用的な手法として，ランダム遅延 [3] が提案されている。ランダム遅延は，イベント処理の前にランダムに発生する遅延処理を挿入することで多様なイベント系列を発生させる仕組みである。遅延の実装に特別な制限はなく，様々な並列処理モデルに利用可能である。たとえば文献 [3] では，メッセージパッシングモデルにおける実装例が示されている。ランダム遅延は，ConTest と同様に個々のイベントに着目したアプローチをとっている。

本稿では，個々のイベントではなくイベント系列に着目した並列プログラムのテスト法を提案する。イベント系列を並べ替え，それに従って再実行することで多様なイベント系列を発生させる。対象とする並列プログラムは，実行の非決定性が特に問題視される共有メモリ型並列プログラムとする。

2　構成

図 1 に本手法の構成を示す。イベント順序を制御するためのイベントスケジューラをプロセスと共有メモリの間に置く。イベントスケジューラは，共有メモリへのアクセスをプロセスに代わっておこない，同時にイベント系列の取得や決められた系列での実行などテストに必要な機能を提供する。

テストは，次の 3 ステップによっておこなわれる。

1. イベント系列の取得
2. イベント系列の並べ替え
3. 再実行

以下，それぞれについて述べる。

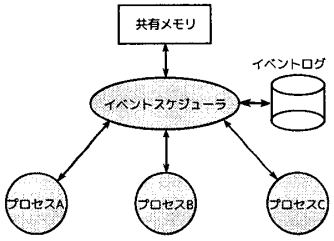

図 1　構成

Step1. イベント系列の取得　このステップでは，テスト対象となる並列プログラムを通常どおり実行し，そのイベント系列を取得する。取得したイベント系列はイベントログに保存する（図 2）。図中の a_i, b_i, c_i は，プロセス A, B, C がそれぞれ実行した非同期イベント，Syn は同期イベントを表している。イベント系列の取得と保存はイベントスケジューラがおこなう。

c0	a0	b0	a1	Syn	a2	c1	Syn	b1	b2	c2	Syn	a3	Syn

図 2　取得したイベント系列

Step2. イベント系列の並べ替え　このステップは，イベントログに保存されたイベント系列をいくつかの部分系列に切り分ける “切り分け操作” と，部分系列内のイベントの順序を並べ替える “並べ替え操作” からなる。

まず，切り分け操作からおこなう。図 2 に示すとおり，イベントには非同期イベントと同期イベントがある。切り分け操作では，取得したイベント系列を同期イベントを区切りとして，いくつかの部分系列に切り分ける（図 3）。

S0	c0	a0	b0	a1	Syn

S1	a2	c1	Syn

S2	b1	b2	c2	Syn

S3	a3	Syn

図 3　切り分けして得られた部分系列

次に，各々の部分系列に対して並べ替え操作をおこなう。並べ替えのルールを以下に示す。

- 別プロセスが実行するイベント（たとえば a_i と b_j）は順序を並べ替えてよい。

*Parallel Program Testing Based on the Permutation of Event Sequence

†Shinya Abe, Tokyo Metropolitan Industrial Technology Research Institute

- 同じプロセスが実行するイベント（たとえば a_i と a_j）は順序を並べ替えてはいけない。

このルールは，別プロセスがそれぞれ実行するイベントは順序不定だが，同じプロセスが実行するイベントの順序はプログラムで決定される場合があるので，不用意に並べ替えるべきではないという考えに基づいている。図 4 に部分系列 S_0 に対して並べ替え操作をおこなった S_0' を示す。

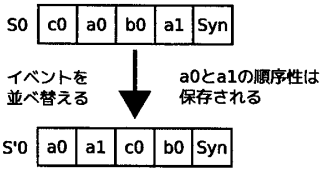

図 4　並べ替えた部分系列

同様にして，全ての部分系列について並べ替えをおこない，得られた系列を再結合する（図 5）。

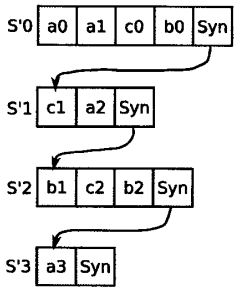

図 5　再結合した系列

Step3．再実行　このステップでは，再結合した系列（図 5）に従って，並列プログラムの再実行をおこなう。イベントスケジューラは，系列どおりにプログラムが実行されるよう，wait イベントなどを用いて順序制御をおこなう。

3　評価

本手法によって，バグ検出率がどの程度向上するかを評価する。実験に使用したベンチマークプログラムは，並列計算機の性能を評価するのにしばしば使われる姫野ベンチ [4] である。実験を進めるにあたって，姫野ベンチを以下のように変更する。

- Java を用いてプログラムを書き換える。
- 通信をメッセージパッシング方式から共有メモリ方式に変更する。
- 同期機構の一部を取り除く（エンバグ）。

ConTest との比較をするために，姫野ベンチを Java プログラムに書き換え，Java VM 上で実行可能にする。また，通信は MPI（Message Passing Interface）を用いたメッセージパッシング方式で実装されているので，これを共有メモリを介した通信に変更する。さらに，同期機構の一部を取り除くことでエンバグする。このエンバグによって，実行の非決定性に起因するエラーが発生するようになる。

上記のプログラムをテスト対象プログラムとして，通常実行，ConTest，ランダム遅延，本手法によるテストをそれぞれ 100 回ずつ実行し，バグ検出率を測定する。それをまとめたものを表 1 に示す。表 1 の正常は期待した出力が得られたことを意味し，エラーは異常終了または期待した出力が得られなかったことを意味する。

バグが存在するにもかかわらず，通常実行ではまったく検出できていない。ConTest，ランダム遅延，本手法はバグを検出できており，その中でも本手法はバグ検出率が高い。したがって，本手法は，実行の非決定性に起因するバグを検出するための方法として，より有用であるといえる。

表 1　ベンチマークのバグ検出率

動作	通常実行	ConTest	ランダム遅延	本手法
正常	100	62	71	8
エラー	0	38	29	92
検出率	0.0	0.38	0.29	0.92

4　おわりに

実行の非決定性に起因するバグを検出する方法として，イベント系列の並べ替えによるテスト法を提案した。本手法は，イベント系列を並べ替え，それに従って再実行することで多様なイベント系列を発生させる。ベンチマークプログラムの同期機構を取り除いてエンバグし，そのプログラムをテスト対象として実験した結果，既存手法と比較してバグ検出率が高く，より有用なテスト法であることが分かった。

今後の課題として，再実行時におけるイベント増減への対応が挙げられる。イベント順序が変化したことで新たに発生するイベントがある場合や，逆に発生しなくなるイベントがある場合は，本稿で述べた方法では再実行できない。まずはこの問題を解決する必要がある。

参考文献

[1] Michael Donat, Silicon Chalk, Debugging in an Asynchronous World, *Developer Tools*, Vol.22, No.6, 2003.

[2] Orit Edelstein, Eitan Farchi, Evgeny Goldin, Yarden Nir, Gil Ratsaby, Shmuel Ur, Framework for testing multi-threaded Java programs, *Concurrency Computat.: Pract. Expr.*, 15:485–489, 2003.

[3] 阿部 真也，西谷 泰昭，MPI プログラムのためのランダム遅延による Unit Testing Framework，情報処理学会 全国大会講演論文集，第 70 回平成 20 年 (1)，pp."1-147"-"1-148"，2008 年 3 月.

[4] Ryutaro Himeno, Himeno Benchmark, http://accc.riken.jp/hpc/HimenoBMT/index.html, 2001 年 12 月.