MPIプログラムのための ランダム遅延によるUnit Testing Framework *

阿部 真也† 西谷 泰昭‡

岩手大学大学院 工学研究科† 岩手大学工学部 情報システム工学科‡

1 はじめに

並列プログラムは，逐次プログラムよりスループットを向上できる反面，実行の非決定性のためテストが難しい [1]。ここでいう実行の非決定性とは，各々のプロセスが非同期に動作し，実行ごとに異なるイベント系列が発生することである。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，すなわちテスト時に発生するイベント系列に偏りがあり，バグ検出率が低いことにある。テストでは多様なイベント系列を発生させる仕組みが必要である。

ConTest[2] は並列 Java プログラムにおいて，多様なイベント系列を発生させるツールである。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（たとえば，共有変数へのアクセスやノンスレッドセーフ関数の呼び出しなど）の前にコンテキストスイッチをランダムに発生させ，多様なイベント系列を発生させる。文献 [2] でバグ検出率が向上することが実証されているが，スレッドによる並列化だけを考慮しているため，プロセスを扱う並列プログラムからは利用することができない。

本稿では，プロセスを扱う並列プログラムのテスト支援を目的として，イベント処理の前にランダム遅延を挿入し，イベント系列に多様性をもたせる仕組みを提供する。並列プログラムの多くはMPI（Message Passing Interfame）によって開発されていることを踏まえ，対象とするプログラムはMPIプログラムとする。

さらに，ランダム遅延を組み込んだMPIプログラムのための Unit Testing Framework である MPIUnit を提供する。Unit Testing Framework は，ユニットテストの作成／実行を支援し，リファクタリングやパフォーマンス・チューニングの変更リスクを下げる効果がある。MPIUnit によって MPI プログラムの変更リスクを下げるとともに，積極的なパフォーマンス・チューニングを可能にする。

2 ランダム遅延

MPI プログラムの式，文，関数において，相対順序がプログラムの振る舞いを変化させるのは MPI 関数である。本稿では MPI 関数の実行だけをイベントとして扱い，MPI 関数にランダム遅延を挿入する。

図 1 にランダム遅延機能の構成を示す。MPI プログラムと MPI ライブラリとは別にランダム遅延ライブラリを置く。MPI 関数を呼び出す際にランダム遅延をフックし，呼び出しのタイミングをずらす。

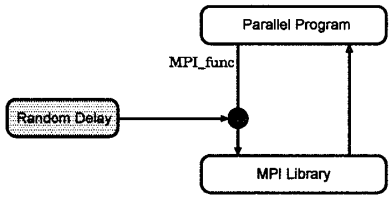

図 1 ランダム遅延機能の構成

ランダム遅延はGCCの拡張機能である GCC Instrumentation Call を使って実装する。GCC Instrumentation Call は，関数の入り口（enter）と出口（exit）にフック関数の呼出しを生成する。フック関数はランダム時間 sleep するように定義する。

3 MPIUnit

MPIUnit は MPI プログラムのための Unit Testing Framework である。MPIUnit は，MPI によって記述された関数に対して，ブラックボックステストをおこなう機能を提供する。ブラックボックステストは関数入出力のみテストし，内部状態まで踏み込まない。

図 2 に MPIUnit の構成を示す。テストコードは，テスト時に MPIUnit の機能とテスト対象関数を呼び出す。MPIUnit は C の Unit Testing Framework である

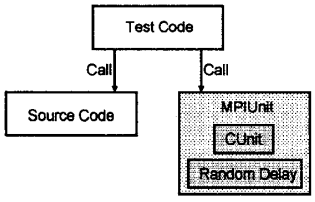

図 2 MPIUnit の構成

CUnit をベースとし，内部に 2 節で述べたランダム遅延機能を組み込む。MPIUnit は次に示す機能を提供する。

- テストの並列実行
- 簡潔なテスト記述
- プロセスごとの { 表明，結果表示 }
- ランダム遅延
- 実行の再現

図 3 に MPIUnit を利用して記述された `cpi` 関数のテストコードを示す。`cpi` 関数は積分をもちいて π の

*A Unit Testing Framework with Random Delay for MPI Programming

†Shinya Abe, Graduate School of Engineering, Iwate University

‡Yasuaki Nishitani, Department of Computer and Information Sciences, Faculty of Engineering, Iwate University

値を計算するMPIプログラムである。ランク0ではcpi()の戻り値が3.1416（誤差0.001以内）であることを表明している。それ以外のプロセスはcpi()を実行するだけである。

test_cpi

```
MPIUT_DEF(test_cpi)
{
  if(MPIUT_RANK == 0) {
    MPIUT_ASSERT_DOUBLE_EQUAL(cpi(),
                              3.1416,
                              0.001);
  }
  else {
    cpi();
  }
}
```

図3　cpi関数のテストコード

図4にプロセス0のテスト結果を示す。表明数1，パス1，失敗0でテストをパスしている。

result

```
rank  0:
  Test: test_cpi() ... passed

Summary: Type       Total  Pass  Fail
         tests          1     1     0
         asserts        1     1     0
```

図4　プロセス0のテスト結果

4　評価

ランダム遅延を挿入することでバグ検出率が向上するかを評価する。実験に使っ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[3]である。姫野ベンチはヤコビ法の主要ループを1分間で何回実行できるかを測定し，並列計算機の性能を評価する。姫野ベンチは正しいプログラムなので，実験をするにあたって姫野ベンチの書き換えをおこなう。まず，通信の一部をブロッキング通信からノンブロッキング通信に変換し，さらに同期機構を取り払う。すると，同期がうまくとれなくなり，異常終了または期待した出力にならないなどの不具合を起こすと予想される。このプログラムの主要ループ回数を1に固定して実行したとき，100に固定して実行したとき，ループ回数1においてランダム遅延を挿入して実行したときのバグ検出率を表1に示す。実験はすべて20回ずつおこなう。

正常終了は期待した出力が得られたことを意味し，実行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。これらの結果から，ループ回数1回程度ではバグを検出できないが，ループ回数を100として大規模な実行をおこなうと検出されるバグであるといえる。一方，ランダム遅延を挿入することでループ回数1という小規模な実行でもバグを検出できる。

表1　ベンチマークのバグ検出率

動作＼ループ回数	1	100	1（遅延挿入）
正常終了	20	4	13
実行エラー	0	16	7
バグ検出率	0.0	0.8	0.35

MPIUnitによるテストコードは図3のように簡潔に記述できる。MPIUnitによるパフォーマンス・チューニングは，以下のようなステップでおこなう。

1. テストの記述
2. パフォーマンス向上のためソースコードを変更
3. テストの実行
4. 2に戻る

ソースコードを少しでも変更したら，すぐにテストを実行することで，問題を早期に発見することができる。このことは，パフォーマンス・チューニングの変更リスクを下げるとともに，積極的なパフォーマンス・チューニングを可能にする。

5　おわりに

本稿では，MPI関数呼び出しにランダム遅延を挿入するランダム遅延機能と，それを組み込んだUnit Testing FrameworkであるMPIUnitを開発した。

ランダム遅延によってバグ検出率が向上し，小規模な実行でもバグを検出できることがわかった。MPIUnitはテスト記述が簡潔かつ実行は自動化されており，テスト作業コストが低い。また，MPIプログラムの変更リスクを下げると同時に，積極的なパフォーマンス・チューニングを可能にする。

今後の課題として，遅延時間の最大値／バグ検出率／実行時間の相関を求めることや，Fortranへの対応があげられる。

なお，MPIUnitはオープンソースソフトウェアとして公開しており，自由に利用，改変，再配布ができる[1]。

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] Ryutaro Himeno, Himeno Benchmark, http://accc.riken.jp/hpc/HimenoBMT/index.html, 2001.

[1] 入手先: https://sourceforge.jp/projects/mpiunit/