Vol.2016-HPC-155 No.10
2016/8/8

仮想マシンエミュレータを用いた特定故障パターン発生時におけるアプリケーションの誤差の評価

小林 佑矢[1,a)]　實本 英之[1,b)]　野村 哲弘[1,c)]　松岡 聡[1,d)]

概要：高性能計算機の規模は年々大きくなっている．大規模化に伴う故障率の増加により, Silent Data Corruption（SDC）と呼ばれる問題が深刻になると予想されている．SDC はアプリケーションが異常な結果を出力するが, 停止には至らないため計算結果の誤りを検知できない障害である．SDC に対処するため多くの研究が行われたが, 計算機の変化とともに故障の種類や発生傾向も変化しており, 新たな耐故障手法が求められている．
本研究は連続したメモリデータの破壊または複数のビットエラーが DRAM 上に発生した際に, NAS Parallel Benchmark の CG カーネルに現れる SDC の調査を目的とする．またそのために, DRAM に特定の故障パターンを注入する故障発生器を, 仮想マシンエミュレータである QEMU を拡張して作成した．
これにより, SDC が発生しうること, アプリケーションの特性により SDC の発生割合が約 5%減少することを確認した．また連続したデータ破壊が発生したとき, アプリケーションは約 80%の割合で正常な結果を返すが, 同じビット数のビットエラーを注入した際には約 90%の割合で異常終了することを確認した．

キーワード：フォールトトレランス, 故障注入

1. はじめに

スーパーコンピュータをはじめとする大規模計算機システムは様々な分野で広く使われている．近年ではスーパーコンピュータに限らず, Amazon Web Service, Inc. や Microsoft Corporation などが, インターネット上にコンピュータを柔軟に構築できるサービス Infrastructure as a Service（IaaS）を展開し, その利用は拡大している．

大規模計算機システムの規模は年々拡大しており, 以下の要因による故障率の増加が予想されている [13] [5].

部品数の増加　故障しうる部品が増え, システム全体での故障率が増加する．

厳しい電力制約　メモリ等の上でデータを表現するための電荷が減少し, 1 ビットあたりの故障率が増加する．

トランジスタの微細化　データを表現するための電荷が減少するとともに, 部品の高密度により部品同士がより近接し, 相互干渉がより頻繁に発生する．

故障が引き起こす問題の一つに Silent Data Corruption（SDC） がある．SDC とは, エラーによりアプリケーションが異常な結果を出力するが, ユーザがその異常を認知せず, 結果が正常であると判断する障害である．クラッシュやハングアップなどと異なり, SDC ではアプリケーションは正常実行時と同様に振る舞うため, 検出が困難であるという特性がある．

SDC に対処するための研究は以前より行われており, 以下のような手法が提案されている．

ハードウェアレベル検出
故障検出をハードウェアで実装する手法であり, ソフトウェアに依存せず, 性能への影響が小さい．一方で, 追加の論理回路や記憶領域が必要となり, 部品自体が高価になる．今日では Single-bit Error Correction Double-bit Error Detection（SEC-DED） ECC が広く採用され, 64 ビットあたりに 2 ビットのエラーまで検出でき, 1 ビットのエラーまで訂正できる．ECC より強力な技術である Chipkill が採用されることもあり, 単一のメモリチップ上のマルチビットエラーまで検出できる．

Algorithm-Based Fault Tolerance（ABFT）
アルゴリズムの特性を利用した耐故障技術である．性能低下が比較的小さいことが多いが, 汎用性に欠けることも多い．行列計算におけるチェックサム [9] など, 計算の前後で満たされるべき不変式をエラーの検知・

[1] 東京工業大学
[a)] kobayashi.y.bk@m.titech.ac.jp
[b)] jitumoto@gsic.titech.ac.jp
[c)] nomura.a.ac@m.titech.ac.jp
[d)] matsu@is.titech.ac.jp

訂正に用いる手法が知られている.

プロセス複製

一つのプロセスを複製し複数実行を行う耐故障手法である. SDC を検知するため複数の複製プロセスの結果を比較し, 全て結果が同じならば SDC が発生していないと判断し, そうではなければ SDC が発生したと判断する. これはすべてのアプリケーションに使用できるため汎用性は高い一方で, 計算資源を多く消費する. この手法は SDC のみでなくクラッシュなどの障害への対処手法として注目され, チェックポイント・リスタートに代わるとも言われている [6]. 近年ではプロセス全体でなく特定の部分のみを複製しオーバーヘッドを減らす研究が行われている [7].

いずれの手法も性能と汎用性とコスト, 故障検知能力の間のトレードオフを持つ. 今後故障率の増加によりさらに深刻になると予想される SDC のために, 新たな手法が求められている. そのためには, SDC へ至る故障の発生割合や SDC における結果の誤りの傾向など, SDC の性質に関する深い理解が必要になる.

故障がアプリケーションに与える影響の研究には [2, 3, 11, 12] が挙げられる. しかし, これらの研究では実際にハードウェア上で発生する故障パターンが考慮されてない. 実際の故障パターンを考慮した評価により, より現実に則した耐故障性の評価が期待される.

本研究では, DRAM に発生する故障に基づく故障パターンを NAS Parallel Benchmark の CG カーネルに注入し, SDC の調査を行う. また故障注入のために仮想マシンエミュレータである QEMU を拡張して故障発生器を設計・実装した.

これにより, SDC が発生しうること, アプリケーションの特性により SDC を軽減できることを確認した. また故障の種類により計算結果に及ぼす影響が異なることも確認した.

2. DRAM の構成と故障

2.1 DRAM の構成

DRAM はデータをまとめて保持するため, メモリセルが格子状に並んだメモリセル列を複数持つ（**図 1**）. 格子における行と列に対応する信号線があり, それぞれワード線・ビット線と呼ばれる. メモリを操作する際はメモリアドレスから指定される行と列をそれぞれ行デコーダと列デコーダで計算し, それらに対応する線に印加することで交点にあるメモリセルのデータが読み書きされる.

2.2 DRAM における故障

SDC の調査のため以下の故障モデルを参考にし, 再現する故障を決定した.

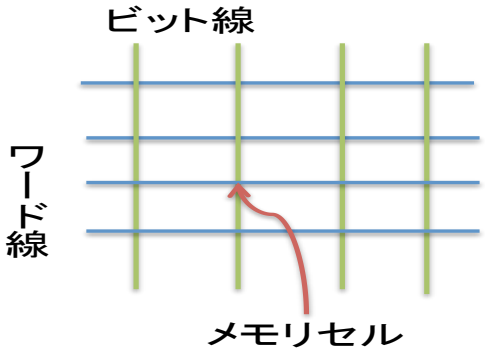


図 1: DRAM のメモリセル列

2.2.1 ワード線縮退故障

Functional Memory Fault Model [4] における故障モデルに縮退故障という, メモリ部品の論理電圧が一つの値に固定される故障がある. その一種であるワード線縮退故障ではワード線の論理値が変化しなくなる. 論理値が 1 に固定された場合にメモリがどのように動作するかは定かでなく, メモリの物理特性により変わる. 一方論理値が 0 に固定された場合にはそのワード線上のメモリセルにアクセスできなくなり, それらのメモリセルの論理電圧が 0 であるように振る舞う.

2.2.2 Retention Fault

Retention Fault とは漏電などによりメモリセルで電荷が徐々に失われ, 意図せず論理電圧が 0 になってしまう故障である. これは一時的故障であり, データの上書きにより隠蔽されうる. DRAM では電荷が時間の経過とともに失われやすいため, 定期的に行われるリフレッシュによりメモリセルに充電が行われる. リフレッシュはメモリセル列の行単位に行われるため, この故障がおきたとき特定の行上のメモリセルが複数侵されることが多い. また, 一度 Retention Fault を経験したメモリセルは再度 Retention Fault を起こす可能性が高いということがわかっている.

2.2.3 Row-Hammer

Row-Hammer とは, 特定のワード線が短い時間で何度もアクセスされたときその周囲のメモリセルのデータが破壊される故障である. トランジスタの微細化に伴う DRAM の密度の増加により, メモリセル同士が互いに干渉しやすくなったことでこの問題が以前より頻発するようになり [14], 近年話題になっている. 2014 年には Kim らにより Row-Hammer を調査する研究 [10] が行われた. Row-Hammer では同じワード線への短時間アクセスは非連続でなくてはならない. 同じワード線に連続してアクセスするときはそのワード線の論理電圧は 1 のままであるが, それらのアクセスの間に他のワード線へのアクセスがあると, 一度論理電圧が 0 になってから再び 1 になる. この電圧の変化が周囲のメモリセルに干渉していると考えてられている. その他に, メモリセルごとに Row-Hammer の影響を受ける可

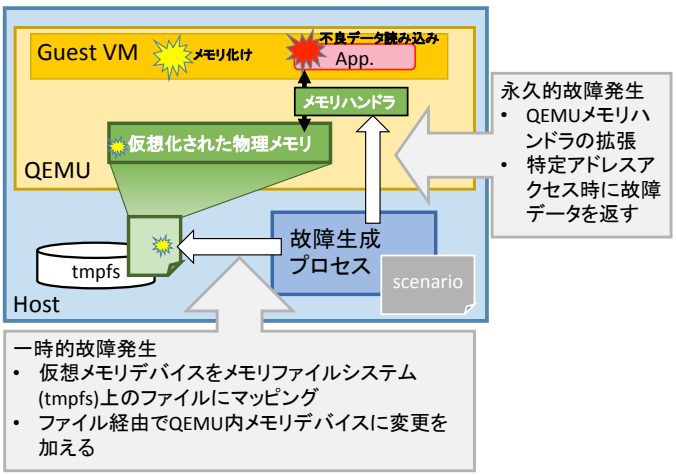


図 2: 故障発生器

能性が異なり, 一度これにより電荷を失ったメモリセルには同じことが再び起こる可能性が高いことが述べられている.

3. 故障発生器の設計

故障注入では, 仮想マシン上でアプリケーションを実行した上で, メインメモリのデータを変更し故障を再現する. 実ハードウェアを用いると, 実験で対象とする特性をもつハードウェアの用意および故障した部品の交換のため, 大きなコストが生じると予想される. 故障を意図した通りに起こし, かつ副作用による意図しない故障の発生を避けることも困難であると考えられる. このため実ハードウェアを用いた故障注入は繰り返し実験に適切でないと思われる. 一方で仮想ハードウェアを用いると, ハードウェア特性はソフトウェアにより記述され, 仮想マシンの生成・破棄も容易である. 故障の発生もユーザが制御でき, 意図しない故障は避けられると考えられる. そのため, 仮想ハードウェアの方が繰り返しの実験およびパラメータやハードウェア特性を変更した実験に適していると考えられる.

故障注入のために仮想マシンエミュレータを拡張し故障発生器を作成する. 今回は節 2.2 の故障を再現するため, 以下の 3 つ要件が必要である.

一時的データ変更 Retention Fault はランダムな一時的故障であるため, アプリケーション実行中にデータを変更でき, データの上書きがその変更を無効化できる必要がある.

永続的データ変更 値が 0 に固定されるワード線縮退故障はランダムな永続的故障であるため, アプリケーションの実行を通して無効にされないデータ変更を行う必要がある.

メモリアクセスパターンに応じた一時的データ変更
Row-Hammer は頻繁にアクセスされるワード線の周囲のメモリセルのデータが破壊される故障であるため, メモリアクセスパターンに応じて関係するデータを変更する必要がある.

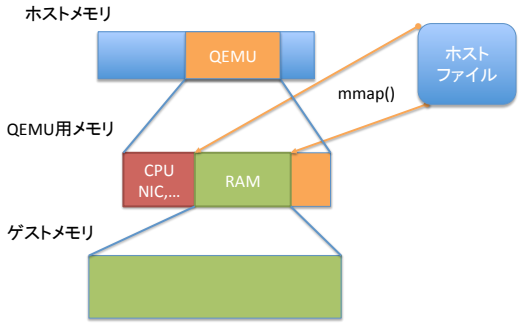


図 3: ファイルマッピング

これらを実現するため故障発生器に以下の機能を加える（図 2）.

メモリファイルマッピング
仮想マシンのメインメモリをホスト OS 上のファイルにマップし, 両者が同じ内容を持つようにする. ここではアプリケーションのメモリ領域だけでなく, 仮想マシンの物理メモリ全体が含まれる. ユーザはファイルを書き換え, メインメモリの値を変更することで, 「一時的データ変更」を行える.

メモリハンドラ
仮想マシンがメインメモリにアクセスする際, 故障発生器がメモリハンドラを呼ぶ. ハンドラでは, メモリで読み書きされる値やアクセスされるメモリアドレスの取得・変更ができ, メモリの内容も操作できる. 故障発生器自体はメモリハンドラの動作を定義せず, 起動時にその定義を含む共有オブジェクトファイルを読み込む. メモリハンドラでメモリアクセスパターンを収集し, メインメモリの内容を書き換えることで「アクセスパターンに応じた一時的データ変更」を行い, 特定のアドレスへのアクセスの際に読まれる値を特定の値に変更することで「永続的データ変更」を実現する.

前述のように故障発生器自体に故障を引き起こす機能はない. ユーザが意図する故障に合わせてデータ変更パターンを決定し, 故障を再現するプラグインを作成する必要がある. 実験を始めるためのコストはかかるが, 柔軟な故障発生を実現できる.

4. 故障発生器の実装

故障発生器のもとになる仮想マシンエミュレータは QEMU バージョン 2.3.1 とする. 対象とするモードは Full System Emulation であり, ホストマシンとゲストマシンはともに x86_64 アーキテクチャを仮定する. この QEMU に「メモリマッピング」を実装して故障発生器を作成した. 「メモリハンドラ」の実装は今後の課題である.

メモリマッピング

```
Algorithm CG(A)
  b = {1,1,...,1};
  DO iter = 1, 75; //Power Method loop
    Solve Ax = b with CG method;
    ζ_iter = 1/(b*x);
    b = normalize(x);
  ENDDO
  return ζ_iter ;
End-Algorithm
```

図 4: NPB の CG 問題クラス B のアルゴリズム

QEMU の-mem-path オプションの機能を変更し実装した. この機能を有効にした故障発生器の動作は以下の通りである.

(1) 故障発生器起動時に-mem-path $map_file と, マッピングされるファイル名が指定される.
(2) オプション解析により, -mem-path オプションで指定されたファイル名を取得する.
(3) 仮想マシンのハードウェア初期化のなかのメモリ作成の際に, 通常のメモリ初期化に代わり file_ram_alloc() をよび, mmap() により指定ファイルを自身のメモリ空間にマップをする（**図 3**）.
(4) その領域を仮想マシンのメインメモリとして登録する.
(5) 仮想マシンを稼働させる.

5. 評価

5.1 対象アプリケーション

NAS Parallel Benchmarks（NPB）[1] の MPI バージョン 3.3.1 における CG カーネルの問題クラス B を対象にして, SDC の調査をする. NPB の CG は逆冪乗法 [15] を用いて疎行列の絶対値最小な固有値の計算を行っており, その各反復の中で共役勾配法 [8] により連立一次方程式を解いている（**図 4**）. ともに反復法であるため, 故障が引き起こす誤りは反復計算により軽減されると期待できる. またそれぞれの方法は局所解へは収束せず, 一意な解析解に収束することが知られている.

今回は NPB の CG を拡張し 3 種の CG アプリケーションを用意した.

オリジナル CG 拡張してない元のアプリケーションである. 逆冪乗法は 75 回反復計算を行い, 共役勾配法は 25 回反復計算を行う.

動的冪乗法 CG 逆冪乗法の反復回数が動的に変化するアプリケーションである. オリジナル CG から逆冪乗法での反復の終了条件を変更した. 連続した反復における近似解同士の差分が 10^{-10} 未満になった時点で, それ以降の反復を実行せずに逆冪乗法が終了する. ただし逆冪乗法は反復回数が 300 回を超えた時点で終了する.

表 1: メモリ使用率

ノード並列数	2	4	8
ノードあたりメモリ使用率（%）	44.2	25.4	13.8

表 2: 仮想マシン環境

メモリサイズ	512MB
OS	Scientific Linux 7.1
仮想化支援（VT-d）	有効

動的反復 CG 逆冪乗法と共役勾配法の反復回数が動的に変化するアプリケーションである. 逆冪乗法の反復の終了条件は動的冪乗法 CG と同じである. オリジナル CG から共役勾配法での反復の終了条件を変更した. $|Ax - b| < 10^{-10}$（A は連立一次方程式の係数行列, b は定数ベクトル, x は共役勾配法により導かれる近似解）となった時点で, それ以降の反復を実行せずに共役勾配法を終了する. ただし各反復法は反復回数が 300 回を超えた時点で終了する.

5.2 故障注入

アプリケーションの耐故障性を確認するため 2 種類の一時的故障を対象にする.

Retention_Faults 仮想マシンのメインメモリを 512 バイトごとにブロック分割し, ランダムに選んだ 1 つのブロック内のビットを全て 0 にする故障. Retention Fault に基づく.

Single_Bit_Faults 仮想マシンのメインメモリ内のランダムな 512 バイト分のビットを 0 にする故障. シングルビットエラーに基づく.

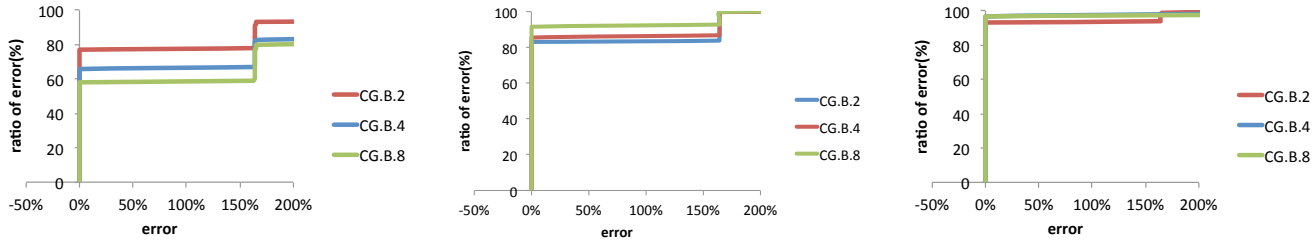
また故障の発生パターンも 2 種類採用する.

複数発生パターン 故障は 1 ノードあたり平均 30 秒に 1 回起き, 発生場所も一様ランダムに決まる.

単一発生パターン 故障はアプリケーションの開始から 5 秒後に 1 度だけ起きる. 発生場所は一様ランダムに決まる.

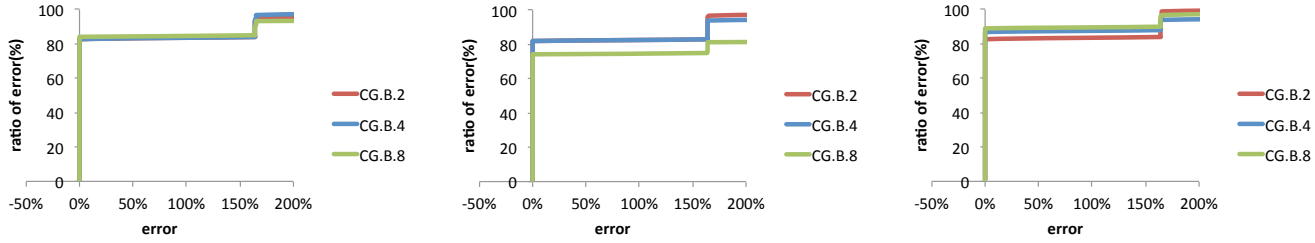
故障注入による SDC の評価のための 1 回の試行は以下のように行う.

(1) 「メモリマッピング」機能でマッピングファイルを指定し, 故障発生器を実行して仮想マシンを起動する.
(2) ゲスト OS に SSH ログインができるまで待ち, SSH コマンドで実行コマンドを送り, アプリケーションを実行する. 実行開始時点を基準時間として故障を発生させる. ssh プロセスはアプリケーションの出力を標準出力するため, これをファイルにリダイレクトして計算結果を集める.
   - 60 秒おきに各ノードに “ssh $node_name hostname” でハングアップやシステムクラッシュの発生を確認



(a) オリジナル CG　(b) 動的冪乗法 CG　(c) 動的 CG

図 5: Retention_Faults・複数発生パターンによる NPB CG の結果の解析解との誤差の累積分布



(a) オリジナル CG　(b) 動的冪乗法 CG　(c) 動的 CG

図 6: Retention_Faults・単一発生パターンによる NPB CG の結果の解析解との誤差の累積分布

表 3: 故障がない時の実行時間（sec）

ノード数	2	4	8
オリジナル CG	70	67	50
動的冪乗法 CG	30	27	21
動的 CG	8	8	8

する．ホスト名を返さないノードがあればシステムクラッシュやハングアップが発生したとし，試行を終了する．

- アプリケーションのハングアップを検出するために実行時間に上限を設けた．あらかじめ故障を発生させずアプリケーションの実行時間を測定し，その 2 倍の時間を超えてなお 1 回の試行が終了しなければ，アプリケーションがハングアップしたとみなし試行を終了する．

(3) アプリケーションが終了し次第，試行の終了とする．

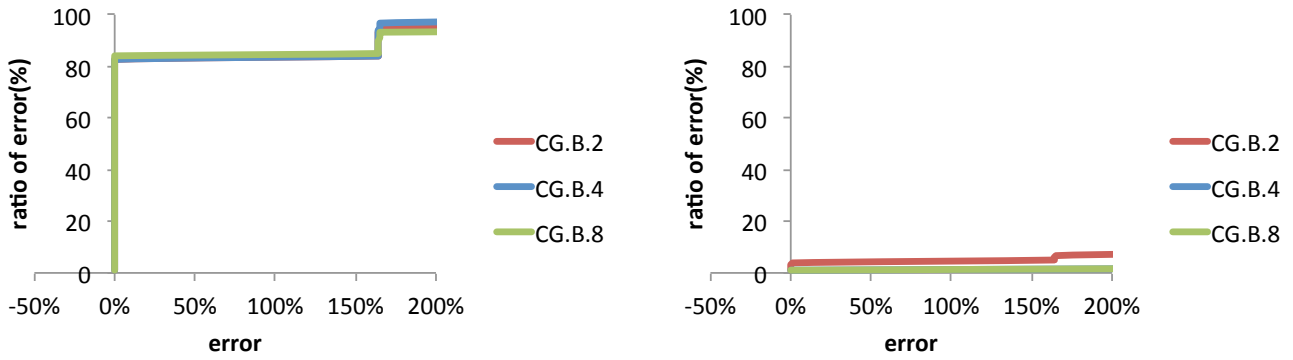
以上のようにして NPB の CG を 2,4,8 ノードで実行したときの実行結果を各アプリケーション・故障パターンに対してそれぞれ 100 だけ収集した．仮想マシンの環境を**表 2** に，NPB の CG カーネルがノードあたりに占有するメモリの割合を**表 1** にそれぞれ示す．

5.3 結果

Retention_Faults を複数発生パターンに従い発生させた際の計算結果の誤差の累積分布を**図 5** に示す．なおアプリケーションの異常終了のために結果が出力されないときや，NaN や Inf など明らかに結果が異常なときは，エラーを無限大として図を作成した．この図より，SDC が発生し，その際はエラーが 160% 付近に集中することが確認できる．またオリジナル CG・動的冪乗法 CG・動的 CG の順に故障率が減少している．これはアプリケーションの実行時間（**表 3**）が短くなり，オリジナル CG 故障に注入される故障が動的 CG には注入されないためと考えられる．

Retention_Faults を単一発生パターンに従い発生させた場合の計算結果の誤差の累積分布を**図 6** に示す．複数発生パターンと異なり実行時間が短くとも同じ量の故障が注入される．図によると 4 ノードで実行した動的 CG では正常終了の割合が約 5%だけ大きくなった．共役勾配法が十分に収束しないまま終了すると，逆冪乗法で正しい値を返す保証がなくなるが，動的 CG では共役勾配法が収束するまで計算されたことで正常な解に収束したためであると考えられる．ただし統計的に，サンプル数を設定し，SDC の発生割合の違いをより詳細に解析する必要がある．

Retention_Faults と Single_Bit_Faults を，それぞれ単一発生パターンに従い注入した際の，オリジナル CG の計算結果の誤差の累積分布を**図 7** に示す．Single_Bit_Faults では異常終了の割合が大きくなっている．これは Retention_Faults における連続したデータ破壊がメモリ全体に分散したために，OS カーネルの領域に故障が発生する確率が高くなり，ノードのハングアップやクラッシュへ至った



(a) Retention_Faults　　**(b)** Single_Bit_Faults

図 7: 単一発生パターンによるオリジナル CG の結果の解析解との誤差の累積分布

ためと思われる．より適切な比較のために, 故障の数や頻度などのパラメータをより広範にし検証することで, 特性の異なる故障発生の影響と比較解析する必要がある．

6. 関連研究

故障とアプリケーションの関係を観察する研究は多く行われてきた．Bronevetsky らの研究では [3] は, 線形代数における反復法とそのための ABFT の耐故障性を調査するため, そのアルゴリズムのデータを表す変数にビットフリップを起こし, アプリケーションの実行結果がどのように変化するかを確認した．Charng-da Lu らの研究 [11] では MPI アプリケーションがどの程度故障の影響を受けるのかを評価している．この研究ではメモリのみならずレジスタや通信路における故障も考慮しており, それぞれがアプリケーションに与える異なる影響を与えることが確認された．しかしこれらの研究で対象にされているシングルビットフリップは ECC により訂正されるため, 現在のシステムでは問題にならない．

マルチビットフリップによる評価を行っている研究もある．Yixin Luo らの研究 [12] では, データセンターで用いられるアプリケーションの実行結果に対し, ハードエラーやソフトエラーが与える影響が解析されている．彼らはその影響の種類や度合いがアプリケーション・故障が注入されるメモリ領域・故障の種類により変化することを確認し, それを元に耐故障に重きをおいたヘテロジーニアスなメモリシステムを提案した．Adamu-Fika らの研究 [2] では, ダブルビットフリップだけでなく新しい故障モデルであるダブルシングルビットフリップを定義し, それらを再現することでアプリケーションの耐故障性を評価した．しかしこれらの研究では実際にハードウェアに起きうる故障パターンを考慮していない．

7. まとめ

本研究では, ユーザが柔軟に故障を再現するための故障発生器を作成した．さらにシングルビットエラーと Retention Faults に基づく故障パターンを NPB の CG カーネルを拡張した 3 つのアプリケーションに注入し, SDC を調査した．SDC の発生が確認され, アプリケーションの特性により SDC の発生割合が減少することも確認された．また注入される故障により, 発生する障害の傾向が異なることが確認できた．

今後の研究として, ディープラーニングやグラフプロセッシングなどの実アプリケーションにおける SDC を調査することがあげられる．また, 異なる故障の種類や発生パターンにより引き起こされる SDC の性質の調査をする必要がある．さらに, 統計的手法や故障発生アドレスの情報などを用いたより詳細な解析をしなくてはならない．他にも, 故障発生器の実装と拡張も今後の課題となる．

謝辞

本研究の一部は科学研究費補助金基盤研究 (S)23220003「10 億並列・エクサスケールスーパーコンピュータの耐故障性基盤」及び科学技術振興機構戦略的創造研究推進事業「EBD:次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」による．

参考文献

[1] Nas parallel benchmarks. `https://www.nas.nasa.gov/publications/npb.html`, March 1994.

[2] Fatimah Adamu-Fika and Arshad Jhumka. *An Investigation of the Impact of Double Bit-Flip Error Variants on Program Execution*, pages 799–813. Springer International Publishing, Cham, 2015.

[3] Greg Bronevetsky and Bronis de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Pro-*

ceedings of the 22Nd Annual International Conference on Supercomputing, ICS '08, pages 155–164, New York, NY, USA, 2008. ACM.

[4] Michael Bushnell. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Kluwer Academic, New York, 2002.

[5] Franck Cappello, Al Geist, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1), 2014.

[6] Kurt Ferreira, Jon Stearley, James H. Laros, III, Ron Oldfield, Kevin Pedretti, Ron Brightwell, Rolf Riesen, Patrick G. Bridges, and Dorian Arnold. Evaluating the viability of process replication reliability for exascale systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 44:1–44:12, New York, NY, USA, 2011. ACM.

[7] C. George and S. Vadhiyar. Fault tolerance on large scale systems using adaptive process replication. *IEEE Transactions on Computers*, 64(8):2213–2225, Aug 2015.

[8] Magnus Rudolph Hestenes and Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS, 1952.

[9] Kuang-Hua Huang and J.A. Abraham. Algorithm-based fault tolerance for matrix operations. *Computers, IEEE Transactions on*, C-33(6):518–528, June 1984.

[10] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *SIGARCH Comput. Archit. News*, 42(3):361–372, June 2014.

[11] Charng-da Lu and Daniel A Reed. Assessing fault sensitivity in mpi applications. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 37. IEEE Computer Society, 2004.

[12] Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu. Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 467–478. IEEE, 2014.

[13] Bianca Schroeder and Garth A Gibson. Understanding failures in petascale computers. In *Journal of Physics: Conference Series*, volume 78, page 012022. IOP Publishing, 2007.

[14] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B. Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. Memory errors in modern systems: The good, the bad, and the ugly. *SIGARCH Comput. Archit. News*, 43(1):297–310, March 2015.

[15] 正武 森. **数値解析**. 共立出版, 東京, Japan, 第 2 版 edition, 2002.2 2002.