Vol.2017-HPC-160 No.8
2017/7/26

メモリアクセスパターン依存故障の注入のための QEMU ベース故障注入器

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：並列計算機の大規模化で, Silent Data Corruption（SDC）による信頼性低下が懸念されている. SDC は検出が困難な障害で, 対応にはコストがかかる. 適切な方法を構築・選択するには, 故障注入によるオーバーヘッドや耐故障性の評価が重要になる. しかし, これまでの故障注入器はランダムなビットフリップを行うものが多く, ハードウェア特有の故障パターンを再現できない。
本研究では実故障の注入を目的として, 仮想マシンエミュレータ QEMU を拡張し, 故障注入器 MH-QEMU を作成した. MH-QEMU では, メモリ状態の変更のみならず, 仮想マシンのメモリへのアクセスを検知・処理できるメモリアクセスハンドラ機能を実現した. これによりメモリアクセスパターン依存故障や永続的故障を注入できる. これらの機能のオーバーヘッドは仮想マシン上のワークロードごとに異なり, NAS Parallel Benchmarks（NPB）を用いた場合には, もっともよい場合で実行時間が約 20 倍で抑えられることを確認した. さらに, NPB の CG カーネルに対し, シングルビットフリップの注入では約 100% の割合で計算が正常終了したが, Row-Hammer の注入では, 約 40% の割合で異常終了が起き, 3% の割合で SDC が発生することを確認した.

キーワード：耐故障, フォールトトレランス, 故障注入, QEMU, Silent Data Corruption, Row-Hammer

1. はじめに

近年, 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の利用が拡大している. GPU やディープラーニング専用プロセッサを用いた複雑かつ大規模なスーパーコンピュータやデータセンタが開発され, GPU を搭載した HPC 向けのクラウドコンピュータの提供も広がっている.

その過程で, より高い計算性能の提供のために生じたノード内の部品の増加・高密度や全ノード数の増加により, システム全体での故障率が増加している [6]. その中でも, Silent Data Corruption（SDC）と呼ばれる, アプリケーションが異常終了せず計算を完了するが誤った結果を出力する障害が懸念されている. SDC の検出には, クラッシュ障害と異なり, 以下のような特別な手法が必要になる.

ハードウェアレベル耐故障手法（HWFT）　HWFT はハードウェア実装された耐故障手法であり, ビット単位のエラーの検出・訂正に用いられる. この技術は高速であることが多いが, ユーザが耐故障強度を調整できず, かつ, 追加の部品が必要になり, 部品の面積の制約が厳しくなる. メインメモリでよく採用される技術には ECC や Chipkill [8] がある. ECC はハミング符号によりビットエラーの検出・訂正を行う手法である. 今日では Single-Error-Correction and Double-Error-Correction（SEC-DED）ECC が広く普及し, 64 ビットあたりに 2 ビットのエラーまでが検出され, 1 ビットのエラーまでが訂正される.

複製ベース手法　複製ベース手法は同じ計算を複数行い, 冗長性を利用する耐故障手法である [13]. この手法は他の手法と異なり, SDC とクラッシュ障害の両方への耐性を持つ. 例えば, 三回同じ計算を行う Triple Modular Redundancy（TMR）では, 二回分の計算が異常終了しても, 残りの一回分の計算が正常終了することで計算結果を取得できる. また, SDC により一回の計算が異常な結果を出力しようとも, 残りの二回の計算が正常な結果を出力すれば, 多数決により正常な計算結果を取得できる. 複製ベース手法は汎用性があるが, 比較的多くの計算資源を消費する. 計算資源の消費を抑えるために, 複製する部分を限定する研究も行われている [17]. そうした研究と MTBF の縮小により, 複製ベース手法がチェックポイント・リスター

[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

ト手法にとって代わると言われている [9].

アルゴリズムベース耐故障手法（ABFT） ABFT は各アルゴリズムの特徴を利用する耐故障手法である. そのため汎用性が低いが, 高い性能を実現することがある. 例として, 行列のチェックサムがある [11]. これは, 行列に対しエラー検知・訂正用の行や列を追加することで, 行列を保護する手法である. この手法は, エラー検知・訂正用の行や列を含めて一つの行列として和算や積算をすることで, 特別な前処理や後処理をせずとも, 行列が保護され続ける性質を持つ. また, チェックサム用の行や列の個数を変更することで, 耐故障強度を調整できる [19].

このように多くの耐故障手法が開発されてきたが, それぞれの耐故障性やコストが異なり, 適切な手法の選択方法が確立されていない. 適切な選択方法を開発する際には, 各耐故障手法の評価が重要である. 各手法を評価する手法として, 実行中のアプリケーションに対する故障注入がある. 故障注入に用いられる故障モデルは, 単一のランダム・ビットフリップや縮退故障が主であった.

しかし, 耐故障性評価のためには, より現実的な故障を注入しなくてはならない. なぜなら, 現実には Row-Hammer のようなメモリアクセスパターン依存故障が確認されており, かつ, 注入する故障によりアプリケーションへの影響が異なるためである [2].

本研究では, メモリへの現実的故障注入を行うための故障注入器 MH-QEMU を, 仮想マシンエミュレータ QEMU [3] を拡張して実装した. MH-QEMU ではハードウェアをエミュレートするので, ユーザーレベルからハードウェアレベルまでの広範な耐故障技術に対する故障注入が可能である. 故障注入のための機能として, MH-QEMU にはメモリファイルマッピング機能とメモリアクセスハンドラ機能を実装した. メモリファイルマッピング機能は, 仮想マシンの物理メモリをホスト OS 上のファイルにマッピングする機能である. これにより, ランダム・ビットフリップのような, マシンの動作に依存しない一時的故障の注入が可能になる. 一方, メモリアクセスハンドラ機能は, ユーザー定義の関数をメモリアクセスのたびに呼び出す機能である. これにより, メモリアクセスの情報の取得・変更が可能になり, Row-Hammer のようなメモリアクセスパターン依存故障の注入や ECC のようなハードウェアレベルのメモリ保護を実装できる.

これらの機能を評価したところ, 仮想マシン上のワークロードによってオーバーヘッドが異なることが確認された. また, MH-QEMU による故障注入により, SDC の際の計算結果の偏りや故障パターンごとの影響の違いを確認した.

2. 関連研究

これまで様々な故障注入器が開発されており, ソースコードを改変するものからハードウェアに直接故障を注入するものまである.

LLVM を用いた故障注入器もある [5,18]. これは, LLVM における中間コードに故障を再現するコードを挿入し, 故障注入をする. LLVM ではコードの解析が行われるため, ソースコードレベルの故障注入よりも故障の精度がよい. コンパイル時に故障注入のためのコード挿入が行われるため, ランタイムソフトウェアによる故障注入よりもオーバーヘッドが低いことが多い. また, Intel Pin [14] や Dynamorio [1] などの Instrumentation Tools を用いることで, アセンブラレベルでの故障注入も行える. ただしこれらは, 実行時にアセンブラを処理するため, 無視できない性能オーバーヘッドが生じる. また, これらの方法ではシステムレベルの耐故障手法への故障注入評価ができず, ハードウェア構成を考慮した故障注入も困難である.

また, FPGA を用いた故障注入器もある [15]. これには, FPGA 上のビットや部品が再構成されることで故障を注入するものと, 専用の部品を追加して故障を注入するものがある. ハードウェアを実装できるため, 正確な故障注入が可能になる. また, ハードウェアで実装されているため, 実行時間オーバーヘッドが小さいことが多い. しかし, 専用のハードウェアを用意する必要があり, 既存の計算機リソースの利用も限られるため, 反復実験には向かない.

Xception [7] は CPU のデバッグやモニタリング機能を用いたベアメタルマシンベースの故障注入器である. これを使う際は, カーネルへの Xception のモジュールのリンクと, アプリケーションからの Xception 関数の呼び出しが必要であり透過的ではないが, トレースモードを使わないため性能低下がほとんどない. 故障はメモリ管理ユニットや汎用レジスタなどの CPU の各機能ユニットやメモリに注入される. しかし, Xception はハードウェアの動作や状態に依存した故障の注入はできない.

また, 仮想マシンベースの故障注入実験向けツールとして D-Cloud [10] がある. これは QEMU を基にした故障注入器 FaultVM をクラウドコンピュータ上に展開し, 故障注入実験を並列かつ大規模に行えるようにした. FaultVM では, ユーザがデバイスモデルを実装し, 故障注入実験に利用できるようになっている. また, シナリオファイルを用いることで故障注入を自動化しつつ, 再現性を担保している. 加えて, システムを解析するツールも搭載している. しかし, 故障モデルは限定されており, ハードウェアに特有の故障パターンの注入には向かない.

これまでは, メモリアクセスパターンに依存する故障を考慮した故障注入器がなかった. 本研究では, 実ハードウェア構成を考慮してメモリアクセスパターンを収集するために, 仮想マシンベースの故障注入器 MH-QEMU を開発した.

3. DRAM 故障パターン

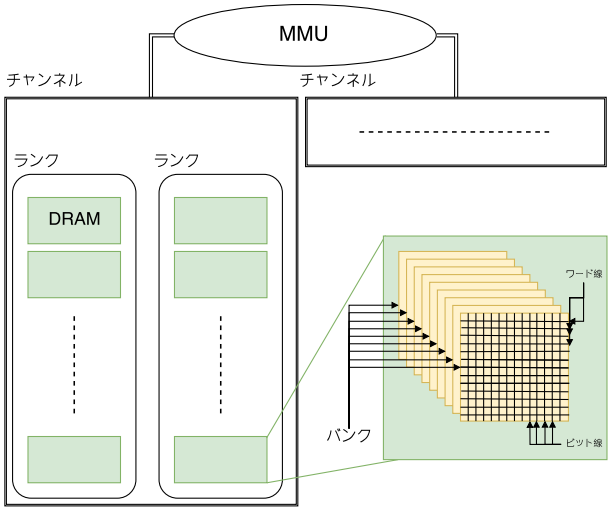


図 1: メモリシステム

3.1 DRAM の構成

近年の DRAM を用いたメモリシステムを図 1 に示す. DRAM チップは DIMM 上にまとめられ, DIMM 単位でシステムに搭載される. CPU が DRAM 上のビットにアクセスするためには, メモリコントローラが物理アドレスから対応するチャンネル・ランク・バンク・行・列のインデックスを計算する. これらのインデックスは階層を成している. チャンネルはメモリコントローラが一度に制御できる DIMM の集合であり, ランクは同じロックステップで動作する DRAM チップの集合である. よって, チャンネルとランクを指定することで DRAM チップを指定できる. バンクは DRAM チップ内に複数個存在するメモリセル列それぞれに対応する. 一つのメモリセル列ではメモリセルが格子状に並んでいる. 縦の線をビット線, 横の線をワード線と呼び, それらの交点にメモリセルがある. 行・列のインデックスはこのビット線・ワード線を指定する. よって, バンク・行・列のインデックスを指定することで, DRAM チップ内でアクセスするメモリセルを指定できる.

メモリコントローラが各インデックスを計算したら, それをメモリモジュールに送る. すると DRAM チップ内では, 指定された DRAM チップのメモリセル列の対応するビット線とワード線が印可し, 対象のビットがアクセスされる.

3.2 DRAM 故障パターン

本研究では以下の故障を対象とする.

3.2.1 ランダムビット・フリップ

ランダムビット・フリップは, 無作為に選ばれたビットが反転する一時的故障である. 故障注入では発生時間も無作為に選択されることが多い. ランダムビット・フリップは故障注入では一般的で, 現実のシステムでも最も多く観測される故障である [16]. この原因としては宇宙船や α 線, 高熱がある.

3.2.2 メモリアクセスパターン依存故障

DRAM はハードウェアの構造に基づき、メモリアクセスに依存する故障パターンが存在する. その一例として Row-Hammer がある. Row-Hammer とは, 一本のワード線が短時間に何度もアクセスされたとき, その周囲のメモリセルのビットが消失する一時的故障である. トランジスタの微細化に伴う DRAM の高密度化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, この問題が近年になって報告された. 2014 年には Kim らにより Row-Hammer を調査する研究 [12] 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, アクセスが非連続であると, 論理電圧が一度 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彼らは予想している. さらに, メモリセルごとに Row-Hammer の影響を受ける可能性が異なり, 一度ビットの消失を経験したメモリセルでは再発の可能性が高いことが述べられている. 本研究ではこのメモリアクセスに応じて発生する故障パターンの一つとして Row-Hammer を取り上げていく.

3.2.3 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永続的故障がある. 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.

4. MH-QEMU の設計

本研究では, 故障発生器 MH-QEMU を仮想マシンベースで作成した. メモリアクセスパターン依存故障はメモリモデルに依存するため, MH-QEMU がハードウェア構成を認識できなくてはならず, そのため, 故障注入器は仮想マシンか実ハードウェアをベースとすることとした. ここで, 実ハードウェアへの故障注入は反復実験になることの多い耐故障評価に向かないため, 採用しない. なぜなら, まず, 意図した通りに故障注入し, 意図しない故障が発生しないことを保証することが難しい. 次に, 故障の直接注入でハード

ウェア自体が故障しやすくなり, 故障検知・修理が必要になる. さらに, 異なるハードウェアでの耐故障性評価を行う場合, その都度調達が必要になる. よって, MH-QEMU では仮想マシンエミュレータ QEMU により, ハードウェアをソフトウェアエミュレートすることとした. なお, これにより, MH-QEMU は QEMU が動作する環境では動作するようになった.

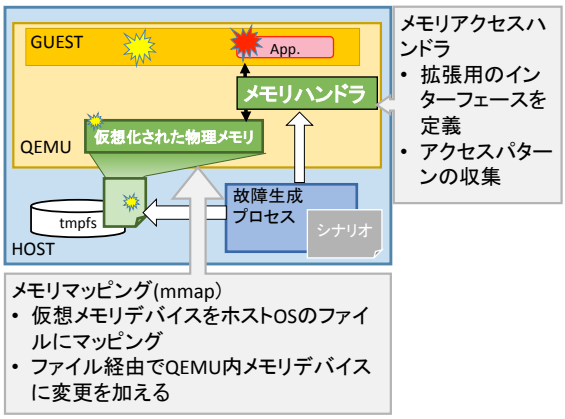


図 2: MH-QEMU のアーキテクチャ

MH-QEMU を用いて節 3.2 の故障を注入するためには, 以下の要件を満たす必要がある.

一時的データ変更 ランダムビットフリップ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 ワード線縮退故障はランダムな永続的故障であるため, アプリケーション実行中にデータを変更でき, 上書きによるデータ変更ができないようにする必要がある.

メモリアクセスパターンに応じたデータ変更 メモリアクセスパターンに依存する故障のためには, メモリアクセスパターンを収集し, 一時的・永続的データを変更ができる必要がある.

これらを実現するために, 以下の機能を QEMU に追加する（図 2）.

メモリファイルマッピング 仮想マシンの物理メモリを, ホスト OS のファイルにマッピングする機能である. これにより, 両者の内容が同じになる. ここではアプリケーションのメモリ領域だけでなく, OS のカーネル領域を含む仮想マシンの物理メモリ全体がマップされる. このファイルを通して, ユーザが物理メモリの内容を読み書きできる. これを用いてユーザがファイルの値を変更することで, 「一時的データ変更」を再現できる.

メモリアクセスハンドラ ユーザ定義のメモリアクセスハンドラを, 仮想マシンのメモリアクセスごとに呼び出す機能である. 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. さらに, QEMU の QMP/HMP 機能を用いてメモリアクセスハンドラ機能の有効・無効の切り替えも実装する. なお, メモリアクセスハンドラの定義は故障発生器では行わず, ユーザが指定した外部の共有オブジェクトファイルから読み込む. これにより, 特定のアドレスへのアクセスの際に読まれる値を変更することで「永続的データ変更」も再現できる. また, 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・永続的データ変更」を再現できる.

これらの機能は, ユーザに仮想マシンにアクセスする手段を提供しているが, 故障発生器自体に故障を注入する機能はない. 故障を注入するには, 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, プラグインを作成する必要がある. これにより実装のコストはかかるが, 柔軟な故障注入を行える. 例えば, SRAM や今後現れる次世代メモリそれぞれに特有なメモリアクセスパターン依存故障の注入も可能となっている.

5. MH-QEMU の実装

MH-QEMU の基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, ホストマシンは x86_64 アーキテクチャと仮定する.

5.1 メモリファイルマッピング

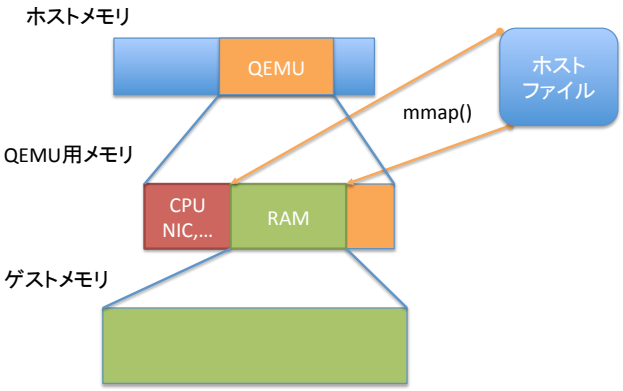


図 3: メモリファイルマッピング

QEMU の-mem-path オプションを変更して実装した. 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.

(0) マッピングされるファイルを用意する. このとき, ファイルサイズが仮想マシンのメモリサイズより大きくなくてはならない. ここで, マッピングされるファイル

表 1: memacc_hdlr_info 構造体のメンバ変数
`target_ulong` はゲストマシンでの `unsinged long` に対応する型

変数名	型	説明
val	uint64_t	ロードされた値・ストアされる値
dsize	size_t	アクセスされるデータサイズ
gvAddr	target_ulong	ゲストマシン上の論理アドレス
gpAddr	target_ulong	ゲストマシン上の物理アドレス
hvAddr	uintptr_t	ホストマシン上の物理アドレス
isLoad	bool	メモリアクセスがロードなら true, ストアなら false
isBigEndian	bool	ゲストマシンのバイトオーダがビックエンディアンなら true, リトルエンディアンなら false

を tmpfs などの高速なファイルシステム上に置くと, 性能低下を軽減できる.

(1) 仮想マシン起動時に`-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際に, 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.

5.2 メモリアクセスハンドラ

メモリアクセスハンドラを実装するために, QEMU の Tiny Code Generator（TCG）を拡張した. TCG はゲストマシンのゲストコードをホストマシンのホストコードへと変換する機構である. TCG があるために, QEMU はホストマシンとは異なる ISA のゲストマシンを動作させることができる. TCG では, 一度ゲストコードを RISC である中間コードに変換し, その後中間コードをホストコードへと変換している. QEMU ではメモリアクセスが発生する中間コードの命令は, ロードとストア命令のみであるため, それらの中間コードをホストコードへ変換する部分を拡張し, メモリアクセス時にメモリアクセスハンドラを呼び出すホストコードを生成するように拡張した. なお, TCG は KVM を有効にすると使用されないため, メモリアクセスハンドラ機能を利用するときは KVM を利用できない.

メモリアクセスハンドラ関数は `memacc_hdlr_info` 構造体のポインタの引数から, メモリアクセス情報を取得できる（表 1）. なお, ここで `val` メンバ変数の値を変更することで, ロード・ストアされる値を変更できる. また, ユーザは, ストアの際のメモリアクセスハンドラ関数に 0 を返させることで対応するストア命令を実行を回避できる. 一方,

表 2: メモリアクセスハンドラモード

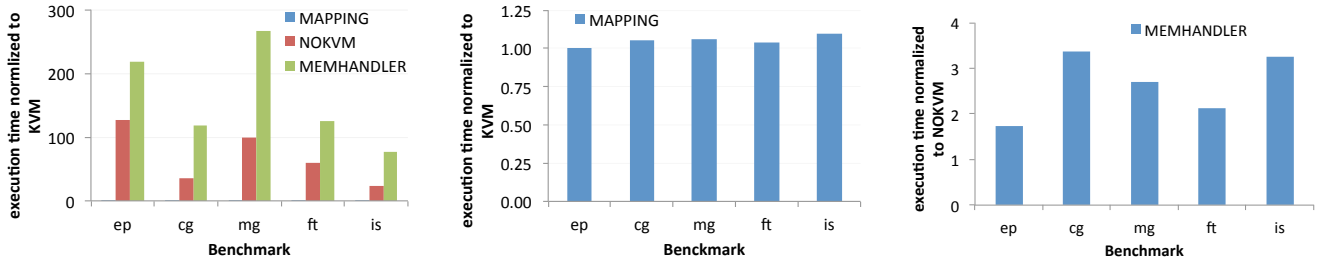
モード	説明
none	メモリアクセスハンドラが呼ばれない
ld	ロードの後にメモリアクセスハンドラを呼び出す
st	ストアの前にメモリアクセスハンドラを呼び出す

```
# command line option
./qemu-system-x86_64 --memacc-handler-mode st,ld
    --memacc-handler-path ./memhandler.so --
    memacc-handler-func handler_func

# QMP
{
        "execute": "memhandler",
        "arguments": {
                "mode": "st,ld",
                "path": "memhandler.so",
                "func": "handler_func"
                 }
}

# HMP
(qemu) memhandler st,ld memhandler.so handler_func
```

図 4: メモリアクセスハンドラ機能の操作コマンド例. st/ld モードを有効にし, memhandler.so 内の handler_func 関数をロード・ストアごとに呼び出す.

そのストア命令を実行したい場合には, メモリアクセスハンドラ関数に 1 を返させなくてはならない.

ユーザが自身のメモリアクセスハンドラ機能を利用する際は, メモリアクセスハンドラ関数をエクスポートする動的共有ライブラリを作成し, QEMU にそのファイルパス・関数名・メモリアクセスハンドラモードを指示する. メモリアクセスハンドラモードはメモリアクセスハンドラが呼ばれるタイミングを示すものである（表 2）. 指示の方法にはコマンドライン引数と QMP/HMP がある. コマンドライン引数による指示は, 仮想マシンの起動時からメモリアクセスハンドラ機能を有効にしたい場合に使用され, メモリアクセスハンドラモードのための`--memacc-handler-mode`, ファイルパスのための`--memacc-handler-path`, 関数名のための`--memacc-handler-func` のオプションを用いる. 一方, QMP/HMP による指示は, 仮想マシンの稼働中にメモリアクセスハンドラ機能の有効・無効やメモリアクセスハンドラ関数を変更する際に使用される. QMP（QEMU Machine Protocol）は JSON 形式のメッセージを QEMU と送受信することで, QEMU や仮想マシンの情報取得・操作を可能にする機能である. HMP（Human Monitor Interface）は QMP をユーザが使いやすいようにした, ワンラインコマンドを用いる対話型のモニターである. 各使用方法は図 4 に示す.



(a) KVM で正規化した実行時間

(b) KVM で正規化した MAPPING での実行時間

(c) NOKVM で正規化した MEMHANDLER の実行時間

図 5: MH-QEMU における各ベンチマークの実行時間

6. 評価

6.1 MH-QEMU のオーバーヘッド

6.1.1 設定

ここでは, MH-QEMU の各機能によるオーバーヘッドを評価する. ここでは, NAS Parallel Benchmarks（NPB）の IS,CG,EP,MG,FT の 5 つのカーネルの問題クラス A を 8 ノードで並列に実行し, 実行時間を計測する. MH-QEMU の設定は, KVM 有効（KVM）·KVM 有効かつメモリファイルマッピング有効（MAPPING）·KVM 無効（NOKVM）·KVM 無効かつメモリアクセスハンドラ有効（MAH）の 4 つを用いる. 前二者はメモリファイルマッピング機能の, 残りはメモリアクセスハンドラ機能のオーバーヘッドを調査するためである. MAPPING でマップされるファイルは tmpfs 上に置く. MAH で呼ばれるハンドラには, 何もしない空のハンドラを用いる. なお, 各仮想マシンでは x86_64 CPU のコア数を 1 つ, メモリサイズを 512MB とした.

6.1.2 結果

各設定での計算結果を図 5 に示す.

図 5a によると, メモリファイルマッピング機能ではメモリアクセスハンドラ機能よりもオーバーヘッドが小さい. メモリファイルマッピング機能によるオーバーヘッドを示したものが図 5b であり, 最高で 10% となっている. これは, MH-QEMU のメモリ空間と tmpfs 上のファイルの同期のために生じるオーバーヘッドによると思われる.

一方, NOKVM のオーバーヘッドはより大きく, 実行時間が最低で 20 倍, 最高で 120 倍と変動がある. この変動原因は, 各アプリごとの特徴にあると思われる. TCG ではゲストマシンの複雑な命令は中間コードにおいて複数の単純な命令に変換されるため, より複雑な命令で構成されるアプリケーションほど性能低下が大きくなると思われる. また, KVM でも NOKVM でも I/O デバイスは QEMU がエミュレートするため, ノード間通信が相対的に多いアプリケーションではオーバーヘッドが小さくなると思われる. さらに, メモリアクセスがランダムな場合は, TLB ミスの発生確率が高く, TLB ルックアップが発生する頻度が高く, それをソフトウェアで実現している TCG ではよりオーバーヘッドが大きくなると考えられる.

また, メモリアクセスハンドラ機能によるオーバーヘッドを計測したところ（5c）, 実行時間が最大で 3.4 倍, 最小で 1.8 倍と変動があった. この原因としては, メモリアクセスが多いほどオーバーヘッドが大きくなることが考えられる.

今後, オーバーヘッドの原因を特定する必要がある.

6.2 耐故障性評価

6.2.1 設定

ここでは, NPB の MPI バージョンの CG カーネルをに対して MH-QEMU を用いて耐故障性評価を行う. 問題クラスは B とし, プロセス数とノード数は 8 とする. CG カーネルは逆冪乗法を共役勾配法を用いて解く問題であり, 入力行列の絶対値最小固有値を出力する. これらアルゴリズムはともに反復法であり, 故障により計算途中の解が破壊されようとも, 最後には十分に収束した値を出力し, 強い耐故障性を実現することが期待できる. なお, CG カーネルを変更し, 規定の反復回数の後でなく, 十分に収束したときに終了するようにした.

注入する故障は節 3.2 より, Row-Hammer とシングル・ビットフリップを選択する. なお, それぞれの故障について, １００回ずつ故障注入を行った.

シングル・ビットフリップ アプリケーションの実行中に一度だけ, 一様ランダムに選択された物理メモリ上のビット一つを反転させる. この故障を注入する時刻もアプリケーションの時間中で一様ランダムに選択する. ビット反転にはメモリファイルマッピング機能を用いる.

Row-Hammer アプリケーションの実行中に一度だけ, 頻繁にアクセスされるワード線の周辺のビットを 0 に

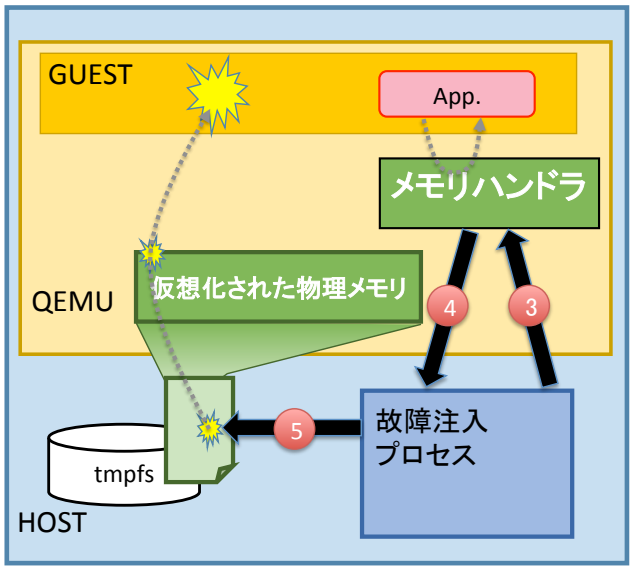


図 6: MH-QEMU を用いた Row-Hammer の注入

する．ここでは，MH-QEMU のメモリファイルマッピング機能とメモリアクセスハンドラ機能を利用する．なお，ここでは仮想マシンの物理アドレスからワード線へのアドレスマッピング方式として，Intel 82955X MCH の dual channel symmetric mode（rank capacity は 512MB）を用いた．故障注入の手順は以下の通りである（図 6）．

(0) メモリアクセスパターンを監視する時刻を，あらかじめ測定したアプリケーションの実行時間中で一様ランダムに決める．これは，本来幾度も発生する Row-Hammer のうちから，ランダムに一つを選びとる操作である．

(1) MH-QEMU により仮想マシンを起動し，アプリケーションを実行する．

(2) アプリケーション実行から第 0 ステップで選択した時間だけ経過したら，HMP によりメモリアクセスハンドラ機能を有効にし，メモリアクセスを監視する．

(3) メモリアクセスハンドラにより，監視を始めてからアクセスが 1000 回を超えたワード線に対し，アクセスのたびに $1/(2^{32}-1)$ の確率で故障を発生させるようにランダムに判定し，発生させる際は故障注入プログラムに通知を行う．ここでアクセスが 1000 回を超えてから故障発生判定をする理由は，アクセスの少ないワード線に対する故障発生を避けるためである．

(4) 故障注入プログラムは，通知に含まれる故障発生が起きるワード線の位置から，データ破壊を受ける可能性のあるメモリセルを特定し，それぞれに対して 50% の確率でビットを 0 にする．ここで可能性のあるすべてのメモリセルに対し故障注入を行わないのは，現実の Row-Hammer でも同様のことが起きないためである．メモリセルの一時的データ変更は，メモリファイルマッピング機能を通じて行う．

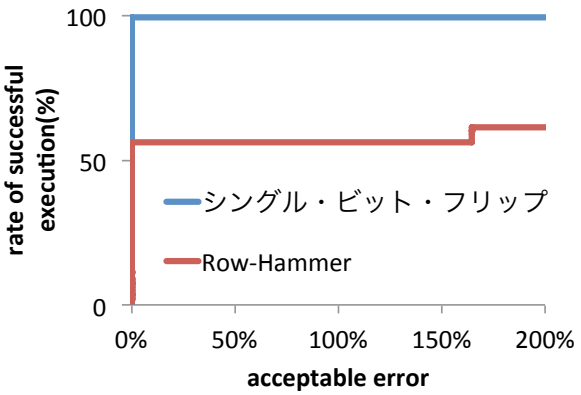


図 7: NPB CG の出力結果の解析解からの誤差

6.2.2 結果

故障注入による耐故障性評価の結果を図 7 に示す．横軸は，アプリケーションユーザが許容できる誤差を示しており，縦軸は出力結果が解析解からその誤差の範囲内に収まったアプリケーションの実行の割合を示している．なお，アプリケーションが異常終了した場合は，計算結果の誤差が無限大であるとしてグラフを作成した．図によると，シングルビットフリップでは約 100% の割合でアプリケーションの実行が正常終了しているが，Row-Hammer では 40% ほどの実行は異常終了している．また Row-Hammer では，誤差が約 160% となる計算結果を出力した実行があるが，他に誤差のある計算結果を出力する実行はなかった．この原因として，1）反復法による局所解への収束，2）入力行列やその他静的変数の破壊，3）MPI プロセス間の一貫性の崩壊，などが考えられる．しかし，反復法による局所解への収束については，CG カーネルで用いられる逆冪乗法は局所解に収束することがないため，棄却される．また入力行列は，各実行の終了時にファイルに出力して元の行列と比較したところ，破壊されていないことを確認した．この偏った誤差の傾向の原因に関しては，より詳細な解析が必要になる．

7. まとめ

本研究では，メモリアクセスハンドラ機能やメモリファイルマッピング機能を仮想マシンエミュレータ QEMU に実装し，ユーザが様々なメモリ故障を注入するための故障注入器 MH-QEMU を作成した．これにより，ランダムビットフリップのみでなく，メモリアクセスパターン依存故障の仮想物理メモリへの注入が可能になった．ただし，MH-QEMU の各機能によるオーバーヘッドが，仮想マシン

上で実行するワークロードごとに異なり, NPB を用いた際には最大で約 260 倍の性能低下を招くことを確認した. また, 故障注入により, NPB の CG カーネルでは, 故障によるエラーに偏りが生じることを確認した.

今後は, MH-QEMU による耐故障性評価を容易にするために, メモリアクセスハンドラ機能の拡張やトレーサの追加などを行うべきである. また, 今回の評価で確認した誤差の偏りや MH-QEMU のオーバーヘッドの原因などを詳細解析する必要がある. さらに, MH-QEMU を用いて様々なアプリケーションや耐故障技術の耐故障性評価を行う必要がある.

謝辞

本研究の一部は JST CREST(JPMJCR1303), JSPS 科研費 JP23220003 及び産総研・東工大 実社会ビッグデータ活用オープンイノベーションラボラトリの支援による.

参考文献

[1] Dynamorio dynamic instrumentation tool platform. `http://www.dynamorio.org/home.html`.

[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] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

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

[5] Jon Calhoun, Luke Olson, and Marc Snir. Flipit: An llvm based fault injector for hpc. In *European Conference on Parallel Processing*, pages 547–558. Springer, 2014.

[6] Franck Cappello, Al Geist, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1):5–28, 2014.

[7] Joao Carreira, Henrique Madeira, João Gabriel Silva, et al. Xception: Software fault injection and monitoring in processor functional units. *Dependable Computing and Fault Tolerant Systems*, 10:245–266, 1998.

[8] Timothy J Dell. A white paper on the benefits of chipkill-correct ecc for pc server main memory. *IBM Microelectronics Division*, 11, 1997.

[9] Kurt Ferreira, Jon Stearley, James H Laros, 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 *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12. IEEE, 2011.

[10] Toshihiro Hanawa, Hitoshi Koizumi, Takayuki Banzai, Mitsuhisa Sato, Shin'ichi Miura, Tadatoshi Ishii, and Hidehisa Takamizawa. Customizing virtual machine with fault injector by integrating with specc device model for a software testing environment d-cloud. In *Dependable Computing (PRDC), 2010 IEEE 16th Pacific Rim International Symposium on*, pages 47–54. IEEE, 2010.

[11] Kuang-Hua Huang et al. Algorithm-based fault tolerance for matrix operations. *IEEE transactions on computers*, 100(6):518–528, 1984.

[12] 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.

[13] Arnaud Lefray, Thomas Ropars, and André Schiper. Replication for send-deterministic mpi hpc applications. In *Proceedings of the 3rd Workshop on Fault-tolerance for HPC at Extreme Scale*, FTXS '13, pages 33–40, New York, NY, USA, 2013. ACM.

[14] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Acm sigplan notices*, volume 40, pages 190–200. ACM, 2005.

[15] S. Sau, M. Kooli, G. Di Natale, A. Bosio, and A. Chakrabarti. Schifi: Scalable and flexible high performance fpga-based fault injector. In *2016 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–4, Nov 2016.

[16] 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.

[17] Omer Subasi, Gulay Yalcin, Ferad Zyulkyarov, Osman Unsal, and Jesus Labarta. Designing and Modelling Selective Replication for Fault-tolerant HPC Applications. pages 452–457, 2017.

[18] Anna Thomas and Karthik Pattabiraman. Llfi: An intermediate code level fault injector for soft computing applications. In *Workshop on Silicon Errors in Logic System Effects (SELSE)*, 2013.

[19] Panruo Wu, Qiang Guan, Nathan DeBardeleben, Sean Blanchard, Dingwen Tao, Xin Liang, Jieyang Chen, and Zizhong Chen. Towards practical algorithm based fault tolerance in dense linear algebra. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '16, pages 31–42, New York, NY, USA, 2016. ACM.