5J-9

メモリエラーに起因するデータ破損を検出するソフトウェアの提案

若林大晃† 片山吉章‡ 出口昌弘‡ 毛利公一‡†

† 立命館大学大学院 理工学研究科 ‡ 三菱電機株式会社 情報技術総合研究所 ‡† 立命館大学 情報理工学部

1 はじめに

1年間に何らかのメモリエラーが発生する確率は，常時稼働しているサーバ（1–4GB の DIMM を複数搭載）の場合，計算機ごとに約 32.2%であるという報告がされている [1]．ECC メモリでは，メモリエラーの検出・訂正が可能であるが，一般向けに広く普及している Non-ECC メモリでは，メモリエラーの検出は困難である．そのため，メモリエラーの発生に気付かずに処理を続けると，重要なデータを失ったことに気付かない可能性がある．

以上の背景から，Non-ECC メモリのエラーに起因する重要なデータの破損を防ぐ研究を行っている．具体的には，アプリケーションのコンパイル時に，メモリエラーを検出するコードを加えることで，データの破損を検出し，訂正する．本論文では，メモリエラーに対処するための手法と，適用した場合にかかるオーバーヘッドの見積りについて述べる．

2 メモリエラーと対策

メモリエラーが発生すると，メモリが記憶していた値に破損が生じてしまう．メモリエラーに対処するためには，エラーを検出することが重要である．メモリエラーには，ソフトエラーとハードエラーの 2 種類がある（表 1）．ソフトエラーは，一時的に発生するエラーであり，正しい値を書き直すことで修復が可能であるため，エラーの訂正が有効である．一方，ハードエラーは，ハードウェアの故障がエラーの原因であるため，同じアドレスで何度もエラーが発生する．したがって，ユーザにエラーの発生を通知する機構が必要である．本論文では，検出・訂正・通知のうち，検出と訂正手法について述べる．

メモリエラーから復帰するためには，メモリエラーを検出し，正しい値に復元する必要がある．これは，メモリエラーを検出・訂正するための機能（コード）と，正しい値に復元するための情報（データ）が必要であること意味する．そのため，検出・訂正用のコードとデータを作成することが大きな課題である．

表 1: メモリエラーの種類

種類	持続時間	原因
ソフトエラー	一時的	α 線や中性子線
ハードエラー	永続的	ハードウェアの故障

表 2: エラー検出・訂正アルゴリズムの例

種類	検出	訂正	データ 8bit に必要なデータ量
データの複製	○	×	8bit
データの複製（複数）	○	○	16bit 以上
ハミング符号	2bit	1bit	4bit

3 メモリエラーの訂正における課題点

メモリエラーを修復するためには，以下の二つの課題を解決する必要がある．

1. エラー検出・訂正アルゴリズム（コードとデータ）
2. エラー検出・訂正コードの付加手法

課題点 1 については，実現した場合にかかる計算量を考慮して，適切なアルゴリズムを選択する必要がある．例えば，以下のようなアルゴリズムが考えられる（表 2）．

- データの複製: 保護データの複製を作ることで，エラーの検出が可能
- データの複製（複数）: 保護データの複製を複数作ることで，エラーの訂正が可能
- ハミング符号: ECC メモリでも用いられているエラー訂正符号の一つであり，2 ビットまでのエラー検出もしくは 1 ビットのエラー訂正が可能

計算量の多いアルゴリズムは，オーバーヘッドによる実行効率の低下が懸念される．そのため，実行効率と機能のバランスを考慮して選択しなければならない．また，アルゴリズムごとに要するデータ量が異なるため，おのおのに適したメモリレイアウトが必要である．エラー検出・訂正データは，アプリケーションが利用する領域と分離して確保し，保護するデータと一対一に対応付ける必要がある．

課題点 2 については，適切なエラー検出・訂正コードの生成とデータ領域の確保が可能である必要がある．

Proposal of error check software for memory errors

Hiroaki WAKABAYASHI†, Yoshiaki KATAYAMA‡, Masahiro DEGUCHI‡and Koichi MOURI‡†

†Graduate School of Science and Engineering, Ritsumeikan University

‡Information Technology R&D Center, Mitsubishi Electric Corporation

‡†College of Information Science and Engineering, Ritsumeikan University

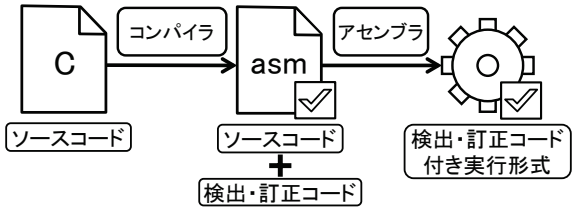

図 1: メモリエラー検出・訂正コードの生成機構

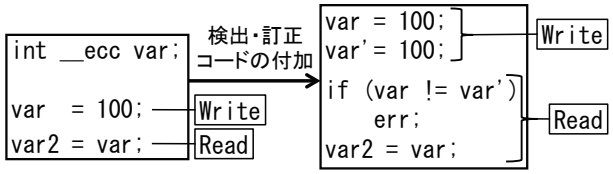

図 2: メモリエラーの検出コードを付加した例

エラー検出・訂正データはメモリに格納されるため，メモリエラーが発生する可能性がある．そのため，一度メモリに格納した値は，値をレジスタにロードしエラーがないことを確認した後，レジスタから用いなければならない．したがって，レジスタを適切に用いるように，アプリケーションのコードとエラー検出・訂正コードを組み合わせる必要がある．

4 コード挿入によるメモリエラーの検出・訂正手法

本研究では，アプリケーションにメモリエラーを検出・訂正する機能を加えることで，データの保護を実現する．検出・訂正コードは，アプリケーションのコンパイル時に自動的に付加する（図 1）．具体的には，ソースコードに記述されている変数に対して，重要であるという印（図 2 左，__ecc）を付け，その変数を扱う命令にエラー検出・訂正用のコードを加える（図 2 右）．これにより，アプリケーションの開発者は，保護したい変数に印を付けることで，変数をメモリエラーから保護することができる．

エラー検出・訂正コードは，変数を READ/WRITE する命令に付加する．実際に付加するコードは，アルゴリズムによって異なるが，図 2 の例では，WRITE 時にデータの複製をとり，READ 時に比較することでメモリエラーの検出を行っている．

5 オーバーヘッドの見積り

本手法を適用した場合にかかる実行時のオーバーヘッドを見積もるために，エラー検出・訂正アルゴリズムを実装したプログラムを作成し，メモリエラーが発生しない場合の READ/WRITE アクセスにかかるオーバーヘッドを計測した．作成した計測プログラムは，検出に「データの複製」，訂正に「ハミング符号」を用いるもので，1.5GB のメモリを下位アドレスからリニアに READ/WRITE アクセスを行う．READ アクセスは，文字定数'c' で初期化したメモリを順番に読み出す．WRITE アクセスは，あらかじめ確保しておいたメモリに，1 から順番に値を書き込む．双方とも，バイト単位で行った．WRITE アクセス時はエラー検出・訂正用データ（「データの複製」と「ハミング符号」）の作成を行い，READ アクセス時はエラーの検出のみ（「データの複製」）を行う．検出と訂正に別のアルゴリズムを用いることで，READ アクセス時のオーバーヘッドを低減することができる．

表 3: オーバーヘッド計測環境

項目	内容
CPU	Intel Core-i7 920 2.67GHz
メモリ	6GB
OS	Debian GNU/Linux 6.0 Squeeze
カーネル	Linux 2.6.32-5

表 4: オーバーヘッドの見積り

READ	データの複製	約 1.6 倍
WRITE	データの複製とハミング符号	約 26 倍

表 3 の環境で計測を行った結果，検出・訂正アルゴリズムを実装してないプログラムと比較して，WRITE アクセスのオーバーヘッドは約 26 倍，READ アクセスのオーバーヘッドは約 1.6 倍であった（表 4）．このことから，WRITE アクセスが少なく，READ アクセスが多いアプリケーションでは，本手法は有用であると考えられる．

6 おわりに

本論文では，メモリエラーによる重要なデータの破損の検出と訂正手法について述べた．データの保護は，アプリケーションにメモリエラーの検出・訂正を行う機能を付加することで実現する．このエラー検出・訂正機能は，アプリケーションのコンパイル時に自動的に付加するため，開発者は重要な変数に印をつけることで，変数をメモリエラーから保護することができる．

本手法を適用した場合のオーバーヘッドは，変数への WRITE アクセス時に約 26 倍，READ アクセス時に約 1.6 倍であることがわかった．今後は，コンパイラを用いてエラー検出・訂正コードを付加する機構の実装を行う．

参考文献

[1] Bianca Schroeder, Eduardo Pinheiro and Wolf-Dietrich Weber: DRAM errors in the wild: a large-scale field study, In Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems, pp. 193–204 (2009).