スクラッチパッドメモリ搭載組込みシステムのソフトエラー耐性を向上するメモリオブジェクト配置手法

森本 喬[†] 小林 良太郎[†] 杉原 真[††,†††]

近年の集積回路の微細化に伴い，ソフトエラーと呼ばれる現象の発生が増加している．特にSRAMであるキャッシュはソフトエラーに対して脆弱であり，ソフトエラー耐性の向上が必要となる．メモリ回路のソフトエラー耐性向上手法として，誤り訂正符号（ECC）技術がしばしば用いられる．ECCの使用はメモリ回路においてアクセスレイテンシの増加を伴う．アクセスレイテンシの増加は，キャッシュを搭載した高い性能が要求される組込みシステムでは許容できない．本研究では，キャッシュのアクセスレイテンシを増加せずに，ECC技術を適用したスクラッチパッドメモリ（SPM）を用いるソフトエラー耐性向上手法を提案する．計算機実験により，SPM未実装時と比較して最大72%のソフトエラー耐性向上を確認した．

A Memory Objects Allocation Scheme to Improve Soft Errors Tolerance on Embedded Systems with a Scratch-pad Memory

Takashi Morimoto,[†] Ryotaro Kobayashi [†]
and Makoto Sugihara[††,†††]

Recent advances in shrinking integrated circuits increase occurrences of soft errors. In particular, since SRAM-based caches are vulnerable to soft errors, it is necessary to improve soft errors tolerance. As soft errors tolerance improvement methods for memory circuits, error correcting code (ECC) techniques often are used. Using ECC involves the increase of access latencies in memory circuits. The increase of access latencies cannot be permitted for embedded systems, which require high performance, with caches. In this paper, we propose a method to improve soft errors tolerance by using a scratch-pad memory (SPM) with ECC techniques, without increasing cache access latencies. Our computer experiments show soft errors tolerance improvement by up to 72%, compared to a system without SPM.

1. はじめに

近年の集積回路の微細化に伴い，半導体RAMで生じるソフトエラーの数が増加している[1)2)]．ソフトエラーは，メモリのビット値が反転する現象である．ソフトエラーの原因は主に宇宙から飛来する宇宙線（主に中性子線やアルファ線）の半導体RAMへの衝突である．ソフトエラーが発生したデータをプロセッサがRead（読み込み）すると，システムが誤動作を引き起こす可能性がある．ソフトエラーの単位ビット当たりの発生率はRAMの種類により異なり，90nmのテクノロジーノードにおいてSRAM（Static RAM）の方がDRAM（Dynamic RAM）より約10,000〜100,000倍程度高い[3)]．SRAMで構成されるキャッシュメモリ（以下キャッシュ）において，システムの誤動作を招くソフトエラーに対する耐性（以下ソフトエラー耐性）の向上が特に必要となる．

ソフトエラー耐性を向上させる手法として，誤り訂正符号（Error Correcting Code：以下ECC）がよく用いられる． ECCはメモリのビット値を基に冗長なビットを生成し利用することでビット値の誤り検出や誤り訂正を可能にする．ECCをキャッシュへ実装すると，冗長ビットを生成するための符号化回路や冗長ビットを用いて誤り検出訂正するための復号化回路による遅延時間が生じる．キャッシュへのECCの実装は，キャッシュのアクセスレイテンシを増加し，性能の低下を招く問題を発生する．Sadlerらは16kbyteのキャッシュの場合，アクセスレイテンシはECCをワー

† 豊橋技術科学大学大学院 工学研究科 情報・知能工学専攻
Department of Computer Science and Engineering, Toyohashi University of Technology
†† 九州大学システムLSI研究センター
System LSI Research Center, Kyushu University
††† 独立行政法人科学技術振興機構, CREST
Japan Science and Technology Agency, CREST

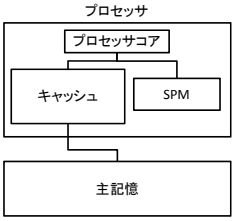

図 1 SPM とキャッシュを混載したシステム構成

ド単位で実装すると 7%増加し，ブロック単位で実装すると 362%増加すると報告している[4]．キャッシュを搭載した高い性能が要求される組込みシステムでは，キャッシュへの ECC の実装による性能低下は許容できない．

本研究では，組込みシステムのキャッシュのアクセスレイテンシを増加させずに，ソフトエラー耐性を向上することを目的とする．目的を果たすために，ECC を実装した小容量のスクラッチパッドメモリ（Scratch-Pad Memory：以下 SPM）と呼ばれるメモリをキャッシュと併用する手法を提案する．高信頼化 SPM の容量をキャッシュの容量以下にすることにより，高信頼化 SPM のアクセスレイテンシをキャッシュのアクセスレイテンシ以下にすることができると考えられる．高信頼化 SPM の併用により，システム全体としての性能が低下しないでソフトエラー耐性の向上が期待できる．高信頼化 SPM の併用に加え，ソフトエラーの発生がシステムの誤動作を引き起こす可能性が高いメモリオブジェクト（プログラムやデータ）を高信頼化 SPM へ配置する手法を提案する．高信頼化 SPM へのメモリオブジェクト配置手法により，さらなるソフトエラー耐性の向上が期待できる．

次節以降の論文構成は次の通りである．2 節では本研究の関連研究について述べる．3 節では提案手法について述べ，4 節において提案手法の評価環境と評価結果について述べる．5 節では本研究のまとめを行う．

2. 関 連 研 究

2.1 スクラッチパッドメモリに関する関連研究

SPM は**図 1** のようにキャッシュと同じメモリ階層に位置し，主記憶と同様なアドレスによるメモリアクセスを行うプロセッサ上の RAM である．SPM はキャッシュと同様に SRAM によって構成されることが一般的であるが，仕組みがキャッシュとは異なる．**図 2**(b) に示す SPM は，図 2(a) に示すキャッシュのようにタグ領域やタグ比較器が存在しない．SPM は，キャッシュのようにプログラム実行中にデータのキャッシュへの Insert（挿入）やキャッシュからの Evict（追い出し）を行わないため，キャッシュよりアクセスレイテンシを抑制できる．組込みシステムは汎用システムのように様々なプログラムを実行することが少なく，処理特化したプログラムを繰り返し実行することが多いため，SPM は組込みシステムで用いられることが多い．メモリオブジェクトの SPM への配置決定はコンパイラやユーザが明示的に行う必要があるが，SPM の利用は組込みシステムの性能要求を満たすことを可能にする．

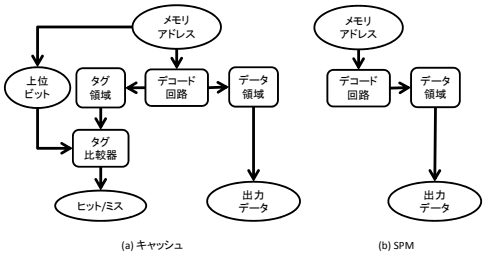

図 2 キャッシュと SPM の各メモリの構成

SPM を用いた関連研究は既に広く行われている．Banakar らはキャッシュの代わりに SPM を用いることを提案し，キャッシュを用いた場合と比較して平均 40%の消費電力を削減した[5]．SPM を用いることにより消費エネルギー削減も可能となる．Steinke らはプログラム領域内の関数や静的領域内のグローバル変数を SPM へ静的配置する手法を提案し，キャッシュを用いた場合と比較して平均 12%〜43%の消費電力を削減した[6]．Avissar らはスタック領域のメモリオブジェクトを SPM へ静的に配置する手法を提案し，スタックを全て主記憶に配置する手法と比較して 44%のプログラム実行時間を削減した[7]．SPM を用いた関連研究の目的は消費電力や実行時間削減であり本研究とは目的が異なる．

2.2 キャッシュのソフトエラー耐性向上に関する関連研究

キャッシュのソフトエラー耐性を向上する本研究と同様な関連研究が既に行われている．Lee らは部分保護キャッシュ（Partially Protected Caches：以下 PPC）と呼ばれるキャッシュを提案した．PPC は，ECC を実装し高信頼化したキャッシュと通常のキャッシュの 2 種類のキャッシュから構成される．PPC の提案に加えて，PPC の各キャッシュへ最適にマッピングするデータを決定する手法も提案した[8]．PPC では，ECC の実装に伴うアクセスレイテンシの増加を抑制するために，ソフトエラー耐性を向上する方のキャッシュを小容量にして 2 種類のキャッシュのアクセスレイテンシ

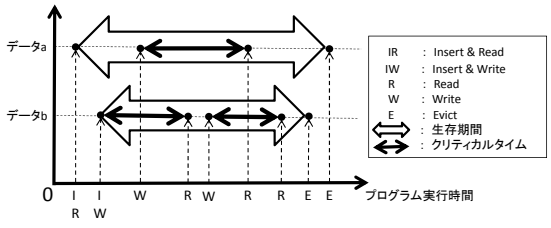

図 3 キャッシュにおけるクリティカルタイムと生存期間のイメージ

を同等にしている． PPC を評価した結果，各種ベンチマークにおいて通常のキャッシュのみを用いた場合と比較して平均で約 50%のソフトエラー耐性を向上した．この関連研究と本研究は使用するメモリが異なるがキャッシュのソフトエラー耐性を向上されるために新たな別の高信頼なメモリを併用する手法を提案している点について関連していると言える．

Lee らは 2 種類のキャッシュのいずれかにマッピングするデータを決定するために，脆弱性尺度と呼ばれる信頼性指標を用いている．脆弱性尺度はさらにクリティカルタイムとサイズの積で表される．クリティカルタイムとは，データがメモリに存在する期間（生存期間）内で，プロセッサによってメモリに Write（書き込み）されてからメモリから Read されるまでの時間のことである[9]．キャッシュにおける各データのクリティカルタイムと生存期間のイメージを**図 3** に示す．図 3 はデータ a に Insert，Read，Write，Read，Evict の順でキャッシュイベントが発生し，またデータ b に Insert，Write，Read，Write，Read，Evict の順でキャッシュイベントが発生した時の様子を表している．ここでは Evict の後，データ a，b は 2 度とキャッシュに Insert されないものとする．図 3 の場合，データ a は Write された後に Read が行われる 1ヶ所，データ b は 2ヶ所がそれぞれクリティカルタイムとなる．以降ではデータ（メモリオブジェクト）においてクリティカルタイムが 2ヶ所以上存在する場合は各場所のクリティカルタイムの合計値をそのデータのクリティカルタイムとする．今後クリティカルタイム，生存期間及び脆弱性尺度について言及する場合，キャッシュにおけるクリティカルタイム，生存期間及び脆弱性尺度について言及するものとする．

クリティカルタイム内でソフトエラーが発生した場合，発生以降に誤ったデータが必ず Read されることになるため，システムが誤動作を引き起こす可能性が高い．一方でクリティカルタイム外でソフトエラーが発生した場合，発生以降に誤ったデータが Write により上書きされたり Evict されて 2 度と使用されなくなるため，システムが誤動作を引き起こす可能性はない．クリティカルタイムが長いデータ程，ソフトエラーの発生によりシステムが誤動作を引き起こす可能性が高くなる．

各データのクリティカルタイムが等しい場合でも各データのサイズの大小によってシステムが誤動作を引き起こす可能性が変化する．データサイズが大きくなる程，データを格納するためのメモリの面積が増加することになるため，ソフトエラーの原因となる宇宙線が衝突する時間当たりの総数が増加する．以上から脆弱性尺度はクリティカルタイムとサイズの積となる．

Lee らは様々なサイズのキャッシュにおいて脆弱性尺度とソフトエラー耐性の相関関係を調査する実験を行った．結果，脆弱性尺度を基にして予測したソフトエラー耐性と実際のソフトエラー耐性が平均で 5 %以内の誤差になったことを確認している[8]．以上から，脆弱性尺度とソフトエラー耐性の間には相関関係が存在し，脆弱性尺度の大きいデータ程ソフトエラー耐性が低いことが言える．本研究でもこのキャッシュにおける脆弱性尺度を利用する．

3. 提 案 手 法

3.1 高信頼化 SPM の併用

本研究では，キャッシュを搭載した組込みシステムにおいてキャッシュのアクセスレイテンシを増加せずに，ソフトエラー耐性の向上を図る．目的を果たすために，ECC を実装し高信頼化した SPM をキャッシュと併用する手法を提案する．ここで重要なことは，高信頼化 SPM のアクセスレイテンシをキャッシュのアクセスレイテンシ以下にすることである．高信頼化 SPM のアクセスレイテンシは，アクセスするデータを指定するアドレスのデコード（アドレスデコード），ECC の符号化・復号化に要する時間により決定される．一方でキャッシュのアクセスレイテンシは，アドレスデコード，タグ比較に要する時間により決定される．アドレスデコードに要する時間はメモリ容量により増減するが，ECC の符号化・復号化に要する時間はメモリ容量に関わらず一定となる．以上から高信頼化 SPM のアクセスレイテンシをキャッシュのアクセスレイテンシ以下にするためには，高信頼化 SPM の容量を少なくし，アドレスデコードに要する時間を削減する必要がある．キャッシュのアクセスレイテンシ以下の高信頼化 SPM のアクセスレイテンシを実現するために，高信頼化 SPM 容量をキャッシュ容量に対して小容量にする．小容量の高信頼化 SPM の併用により，キャッシュのアクセスレイテンシを増加させる

ことなく，ソフトエラー耐性の向上が実現できる．

3.2 ソフトエラー耐性を向上させるメモリオブジェクト配置手法

高信頼化 SPM を小容量にするため，高信頼化 SPM に配置可能なメモリオブジェクトの数に制約が生じる．記憶容量制約を満たし，かつ，よりソフトエラー耐性を向上するために，高信頼化 SPM へのメモリオブジェクトの配置手法も提案する．高信頼化 SPM に配置するメモリオブジェクトを決定するために，キャッシュにおける脆弱性尺度を用いる．具体的には以下の手順でメモリオブジェクトの配置決定を行うことを提案する．

(1) キャッシュのみを実装した環境（高信頼化 SPM 未実装の環境）においてプログラムを実行した時の各メモリオブジェクトの脆弱性尺度のプロファイルを取る．
(2) 高信頼化 SPM へ配置対象の全メモリオブジェクトを単位サイズ（バイト）当たりの脆弱性尺度が大きい順にソートする．
(3) ソート結果の先頭から順に高信頼化 SPM への配置を考える．この時，配置を考えるメモリオブジェクトのサイズと既に配置が決定しているメモリオブジェクトのサイズを合計し，高信頼化 SPM 容量以下であるかを確認する．
    - 高信頼化 SPM 容量以下の場合：高信頼化 SPM に配置することが決定される．
    - 高信頼化 SPM 容量を超えている場合：高信頼化 SPM に配置しないことが決定される（主記憶に配置される）．

上記の配置手法により，結果的に脆弱性尺度が大きい，つまりソフトエラー耐性の低いメモリオブジェクトが高信頼 SPM に配置される．高信頼化 SPM に配置したメモリオブジェクトはキャッシュを使用しないため，脆弱性尺度は 0 となる．以上から各メモリオブジェクトの脆弱性尺度の合計は小さくなり，全体としてソフトエラー耐性は向上すると考えられる．上記の配置手法において特に注意しておくことは，主記憶に配置する各メモリオブジェクトの脆弱性尺度がキャッシュのみ実装した環境と高信頼化 SPM を併用した環境で等しいと仮定していることである．つまり，高信頼化 SPM を併用しても高信頼化 SPM に配置しないで主記憶に配置している各メモリオブジェクトの脆弱性尺度は変化しないと仮定している．実際の環境では高信頼化 SPM の併用により主記憶に配置しているメモリオブジェクトの脆弱性尺度が変化し，高信頼化 SPM 併用前と誤差が生じることは十分考えられる．次節の評価において，提案手法より決定した配置から求めた時の脆弱性尺度と高信頼化 SPM を併用した環境でシミュレーションして求めた時の脆弱性尺度の誤差を検証する．

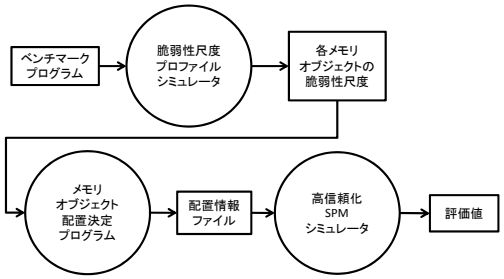

図 4 提案手法評価のフレームワーク

本稿で提案する手法は，高信頼 SPM を併用した環境においてプログラムを実行させる前に各メモリオブジェクトの脆弱性尺度をプロファイリングして配置を決定するため静的配置手法となる．同様にプログラム実行中にメモリオブジェクト配置を変更する動的配置手法も考えられるが， プログラム実行中に SPM と主記憶間にデータ転送が必要となり，ハードウェアコストが生じたり転送によるオーバヘッドが生じる．本稿においては，メモリオブジェクトを動的に配置することは対象外とする．

4. 評　　価

本節では提案手法の適用によりソフトエラー耐性がどの程度向上するかの評価を行う．高信頼化 SPM を実装した環境におけるベンチマークプログラム実行時の各メモリオブジェクトの脆弱性尺度の合計をシミュレータを用いて評価する．キャッシュのみを実装した環境においてベンチマークプログラムを実行した時との比較を行う．以降の評価で表している脆弱性尺度は特別な指定がない限り各メモリオブジェクトの脆弱性尺度の合計を表すものとする．同様にベンチマークプログラムの実行サイクル数の評価も行う．

4.1 評 価 環 境

本研究の提案手法を評価するためのフレームワーク（データフロー図）を**図 4** に示す．図 4 のフレームワークでは，主に脆弱性尺度プロファイルシミュレータ，メモリオブジェクト配置決定プログラム，及び高信頼化 SPM シミュレータの 3 つを用いて評価を行う．

最初の脆弱性尺度プロファイルシミュレータは，評価対象のベンチマークを入力とし，シミュレータ内でキャッシュをシミュレーションしながら脆弱性尺度を計算し，プロファイル結果を出力する．本研究では，

評価環境のキャッシュを搭載したプロセッサのシミュレータとして SimpleScalar 3.0[10] を使用し，このシミュレータ内に各メモリオブジェクトの脆弱性尺度を計算する機構を実装した．

次のメモリオブジェクト配置決定プログラムは，各メモリオブジェクトの脆弱性尺度を入力とし，SPM への配置決定を行い，決定されたメモリオブジェクト配置の情報が記録される配置情報ファイルを出力する．本研究では，このプログラムに 3.2 節で述べた配置手法を C 言語で実装した．

最後の高信頼化 SPM シミュレータは，配置情報ファイルを入力とし，配置情報に従ったメモリオブジェクト配置で対象ベンチマーク実行し，評価値（脆弱性尺度，実行サイクル数）を出力する．従って，最初に使用した脆弱性尺度プロファイルシミュレータに改良を加え，高信頼化 SPM のシミュレーションも可能にしたシミュレータを高信頼化 SPM シミュレータとして用いた．

本研究において，高信頼化 SPM への配置対象はプログラム領域及び静的領域のメモリオブジェクトとした．また，配置対象のメモリオブジェクトの粒度はプログラム領域では関数単位，静的領域では変数単位で考えることとした．

本研究の評価環境における各種パラメータは以下のように設定した．

- キャッシュ構成:L1 命令/データキャッシュ（32K バイト）
  - 命令キャッシュ：セット数 512，ウェイ数 1，ラインサイズ 32 バイト（16K バイト）
  - データキャッシュ：セット数 128，ウェイ数 4，ラインサイズ 32 バイト（16K バイト）
- キャッシュライン置換方式：LRU
- キャッシュのアクセスサイクル数：1 サイクル
- 主記憶のアクセスサイクル数：100 サイクル
- 高信頼化 SPM 容量：128，256，512，1K，2K，4K，8K バイト
- 高信頼化 SPM のアクセスサイクル数:1 サイクル

ここでキャッシュのアクセスレイテンシを増加せずに，高信頼化 SPM を併用することが可能である妥当性を示す．**表 1** に 32K バイトのキャッシュ及び各高信頼化 SPM のアクセスレイテンシを示す．キャッシュと SPM のアクセスレイテンシは CACTI 6.5[12] を用いて求めた．SPM の高信頼化に用いた ECC のレイテンシは Sadler ら[4] の実験結果（16K バイトのキャッシュに ECC をワード単位で実装すると約 7%のアクセスレイテンシが増加）を元に求めた．表 1 よりどの高信頼化 SPM もキャッシュ以下のアクセスレイテンシになることから，キャッシュのアクセスレイテンシを増加せずに，高信頼化 SPM を併用することが可能となる．以上のことから本研究では，キャッシュのアクセスサイクル数と高信頼化 SPM のアクセスサイクル数を等しくする．

メモリの種類	アクセスレイテンシ [ns]
キャッシュ32K バイト	0.292
高信頼化 SPM 1K バイト	0.180
高信頼化 SPM 2K バイト	0.190
高信頼化 SPM 4K バイト	0.230
高信頼化 SPM 8K バイト	0.287

表 1 キャッシュと高信頼化 SPM のアクセスレイテンシ

評価対象のベンチマークプログラムとして，組込みシステム向けベンチマークスイートである Mibench[11] の中から比較的小規模なベンチマークである *bitcount*，*string_search*，*FFT* の 3 種類を用いた．ベンチマークプログラムの特徴を**表 2** に示す．表 2 において脆弱性尺度と実行サイクル数は，キャッシュのみ実装された環境においてベンチマークプログラムを実行した時の脆弱性尺度と実行サイクル数をそれぞれ示している．表 2 のメモリオブジェクトサイズに注目すると，いずれも本研究で用いる最大の高信頼化 SPM 容量である 8K バイトより大きく，高信頼化 SPM に全てのメモリオブジェクトを配置できないことから，配置するメモリオブジェクトを決定する必要があることがわかる．

4.2 評価結果

4.2.1 提案手法の妥当性の検証

この節では実際の評価を行う前に，提案したメモリオブジェクト配置手法の妥当性を検証する．妥当性検証のために，提案手法を適用して決定した配置時の脆弱性尺度と脆弱性尺度の下限値の比較を行う．

本研究で対象としているメモリオブジェクト配置問題は，ナップサック問題と等価な最適化問題である．ナップサック問題とは大きさ M のナップサックに i 個のオブジェクト（各オブジェクトの大きさ m_i，価値 w_i）の内のいくつかをオブジェクトの価値の合計が最大になるように詰め込む方法を考える問題である．ナップサック問題では各オブジェクトのサイズが全て等しい場合，各オブジェクトの価値の大きい順に配置すれば価値の合計が最大となることが知られており，その価値の合計を上限値とし，提案したオブジェクトの詰め込み方法時の価値の合計と比較し，誤差が小さい程提案した詰め込み方法が優れていると言える．

ナップサック問題におけるナップサックの大きさ，オブジェクトの大きさ，及びオブジェクトの価値は，

ベンチマーク名	メモリ オブジェクト数	メモリオブジェクト サイズ [バイト]	脆弱性尺度 [サイクル・バイト]	実行サイクル数 [サイクル]
bitcount	111	62784	2.68E+10	4.74E+5
string_search	75	33724	5.18E+10	8.08E+5
FFT	120	36228	1.92E+11	2.51E+7

表 2 ベンチマークプログラムの特徴

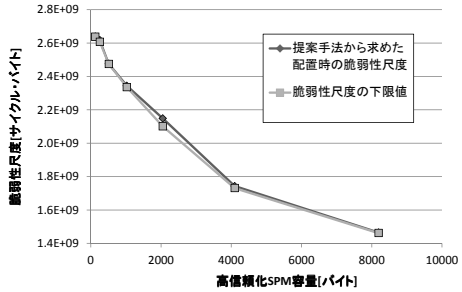

図 5 bitcount ベンチマークプログラムを用いた場合の提案手法を適用して決定した配置時の脆弱性尺度と脆弱性尺度の下限値

メモリオブジェクト配置問題における高信頼化 SPM 容量，メモリオブジェクトのサイズ，メモリオブジェクトの脆弱性尺度にそれぞれ対応する．本研究をナップサック問題に適用すると，各メモリオブジェクトの脆弱性尺度の合計を最小化するため上限値でなく下限値を求めて妥当性を検証することになる．メモリオブジェクトのサイズは異なっており，通常の方法では脆弱性尺度の下限値を求められない．そこで今回は全てのメモリオブジェクトを単位バイト毎に分割することが可能であると仮定して下限値を求めることを考える．その際分割したメモリオブジェクトの脆弱性尺度も元のメモリオブジェクトの脆弱性尺度を分割した数で割った値とする．この方法で求めた脆弱性尺度の下限値と提案手法を適用して求めた配置時の脆弱性尺度の比較を行う．**図 5** に *bitcount* ベンチマークプログラムを用いた場合の提案手法を適用して求めた配置時の脆弱性尺度と脆弱性尺度の下限値を示す．

図 5 から提案手法を適用して決定した配置時の脆弱性尺度と脆弱性尺度の下限値の間に最大で 2.3%の誤差が存在することがわかる．*bitcount* ベンチマークのような結果の表示は割愛するが，*string_search* 及び *FFT* ベンチマークでも同様な比較を行い，それぞれ最大で 0.82%，0.80%の誤差が存在することを確認した．以上より提案手法により決定される配置時の脆弱性尺度は下限値に十分近い値が得られることから，妥当な提案手法であることが言えると考えられる．

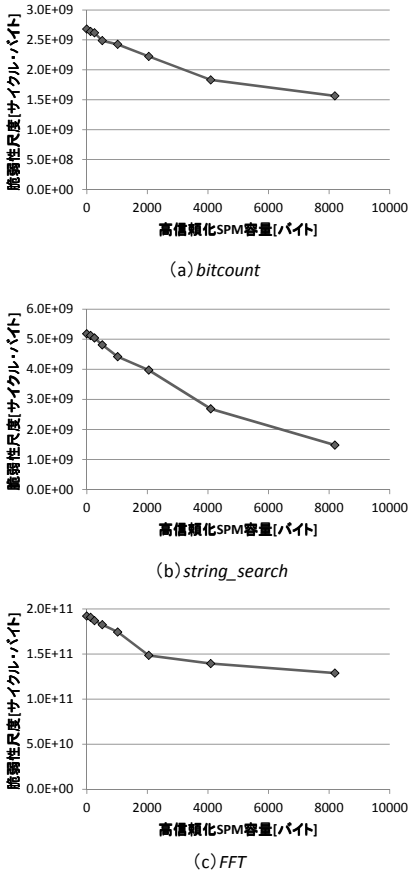

図 6 各ベンチマークプログラムにおける各高信頼化 SPM 容量時の高信頼化 SPM シミュレータから求めた脆弱性尺度

4.2.2 各高信頼化 SPM 容量時の脆弱性尺度と実行サイクル数

この節では提案手法を適用して決定したメモリオブジェクト配置でベンチマークプログラム実行時の脆弱性尺度と実行サイクル数を高信頼化 SPM シミュレータを用いて評価する．**図 6** に各ベンチマークプログラムにおける各高信頼化 SPM 容量時の高信頼化 SPM シミュレータから求めた脆弱性尺度を示す．図 6 から，高信頼化 SPM 容量を増加させると脆弱性尺度が減少することがわかる．また，*bitcount*，*string_search*，及び *FFT* の各ベンチマークにおいて，キャッシュのみを実装した環境と比較してそれぞれ最大約 42%，72%，

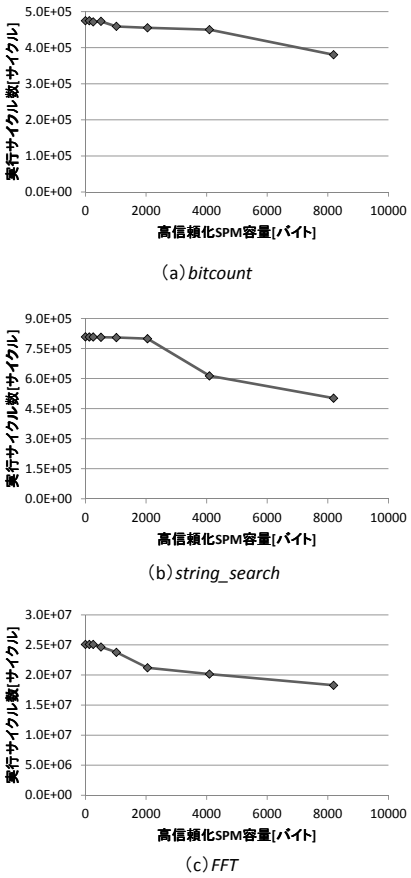

図 7 各ベンチマークプログラムにおける各高信頼化 SPM 容量時の高信頼化 SPM シミュレータから求めた実行サイクル数

及び 33%の脆弱性尺度が減少し，ソフトエラー耐性が向上していることがわかる．高信頼化 SPM 容量の増加に伴う脆弱性尺度の変化の度合いは図 6 からわかるようにベンチマーク毎に異なるため，アプリケーション依存であることが言える．

同様に**図 7** に各ベンチマークプログラムにおける各高信頼化 SPM 容量時の高信頼化 SPM シミュレータから求めた実行サイクル数を示す．図 7 から，高信頼 SPM 併用による実行サイクル数の増加は全く見られないことがわかる．それどころか SPM 容量を増加すると実行サイクル数が減り，*bitcount*，*string_search*，及び *FFT* の各ベンチマークにおいて最大で約 13%，38%，及び 27%の実行サイクル数がそれぞれ削減されていることがわかる．削減された理由は，キャッシュミスアクセスの減少であると考えられる．キャッシュのみの環境時にキャッシュミスが起こっていたメモリオブジェクトのキャッシュミスによる主記憶アクセスが高信頼化 SPM への配置により減少し，実行サイクル数が削減されたと考えられる．この副次的効果は本来の組込みシステムにおける SPM 利用のメリットによるものであると考えられる．

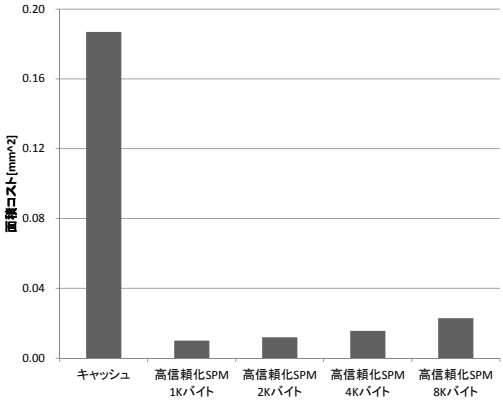

図 8 評価環境のキャッシュと各容量の高信頼化 SPM の面積コスト

4.2.3 高信頼化 SPM 併用時の面積オーバヘッド

組込みシステムは限られた資源で様々な機能を実現するためにチップ面積のオーバヘッドの削減が重要視される場合がある．本研究の提案手法は高信頼化 SPM をキャッシュと併用する手法であり，面積オーバヘッドを許容しなければならない．この節ではどれくらいの面積オーバヘッドがあるか調査するため，評価環境で使用したキャッシュと高信頼化 SPM の面積コストの比較評価を行った．今回キャッシュ及び SPM の面積コストを求めるために CACTI 6.5[12] を使用した．メモリのテクノロジーノードは 32nm とした．ECC を実装した高信頼化 SPM の面積コストについては以下の 3 点を仮定して評価した．

- SPM の面積コストはメモリセル数（ビット数）に比例する．
- ECC の実装は 2^n ビットのデータ毎に $n+2$ ビットの冗長ビットが必要となる．
- ECC の実装はワード単位で行う．

評価環境で使用したキャッシュ（32K バイト）と各容量の高信頼化 SPM（1K，2K，4K，8K バイト）の面積コストを**図 8** に示す．図 8 から，高信頼化 SPM 容量が増加すると面積コストも増加することがわかる．また，1K，2K，4K，及び 8K バイトの各高信頼化 SPM の併用でそれぞれキャッシュの約 5.4%，6.4%，8.3%，及び 12%の面積オーバヘッドがあることがわかる．前節の評価結果も考慮すると，以上から面積オーバヘッドとソフトエラー耐性（及び実行サイクル数）の間にはトレード関係が存在することが考えられる．また，面積オーバヘッドの割合が前節のソフトエ

ラー耐性向上や実行サイクル数削減の割合と比較すると小さいことから，本提案手法は割合的に少ない面積オーバヘッドでソフトエラー耐性向上と実行サイクル数削減を達成可能な有効な手法であると言えると考えられる．

4.2.4 高信頼化 SPM を併用する手法と ECC を実装したキャッシュを使用する手法の比較

1 節においてキャッシュのソフトエラー耐性向上手法としてキャッシュに ECC を実装して使用する手法を述べた．この節では提案した高信頼化 SPM を併用する手法と ECC を実装したキャッシュを使用する手法の比較を行った．比較はベンチマークプログラムの実行時間及び面積コストに関して行った．ベンチマークプログラムの実行時間はキャッシュのアクセスレイテンシと SimpleScalar 3.0 を用いて得た実行サイクル数の積で求めた．キャッシュ及び SPM のアクセスレイテンシは CACTI 6.5[12] を使用して評価した．ECC を実装したキャッシュのアクセスレイテンシについては CACTI 6.5 による評価に加えて，Sadler ら[4] の論文を参考にして評価した．SimpleScalar 3.0 におけるシミュレーションでは，キャッシュのアクセスサイクル数は ECC を実装しても変更せず 1 サイクルとし，キャッシュ以外のアクセスサイクル数を変更してシミュレーションを行った．ECC を実装したキャッシュの面積コストは前節の高信頼化 SPM の面積コストの計算時と同様な計算を行った．キャッシュではデータ領域だけでなくタグ領域にも ECC を実装すると仮定した．キャッシュ1 ラインのタグサイズは命令キャッシュ及びデータキャッシュでそれぞれ 19 ビット及び 21 ビットと仮定した．

図 9 に高信頼化 SPM 併用時と ECC を実装したキャッシュ使用時のベンチマークプログラムの実行時間を示す．図 9 にはキャッシュのみを使用する環境，ECC を実装したキャッシュを使用する環境，キャッシュと 4K バイトの高信頼化 SPM を併用する環境，キャッシュと 8 バイトの高信頼化 SPM を併用する環境における実行時間を示している．図 9 からキャッシュのみの環境と比較して ECC を実装したキャッシュを使用する環境ではどのベンチマークプログラムを用いても実行時間が増加してしまうことがわかる．一方で，高信頼化 SPM を併用した場合はキャッシュのみの環境と比較して，どのベンチマークプログラムを用いても実行時間が減少していることがわかる．

図 10 に高信頼化 SPM 併用時と ECC を実装したキャッシュ使用時の面積コストを示す．図 10 にはキャッシュ，キャッシュ+ECC，キャッシュ+4K バイト高信頼化 SPM，及びキャッシュ+8K バイト高信頼化 SPM の面積コストを示している．図 10 からキャッシュのみ場合と比較すると，ECC を実装したキャッシュの使用と高信頼化 SPM の併用のいずれも面積オーバヘッドが生じることがわかるが，高信頼化 SPM の併用の方が少ない面積オーバヘッドになることがわかる．

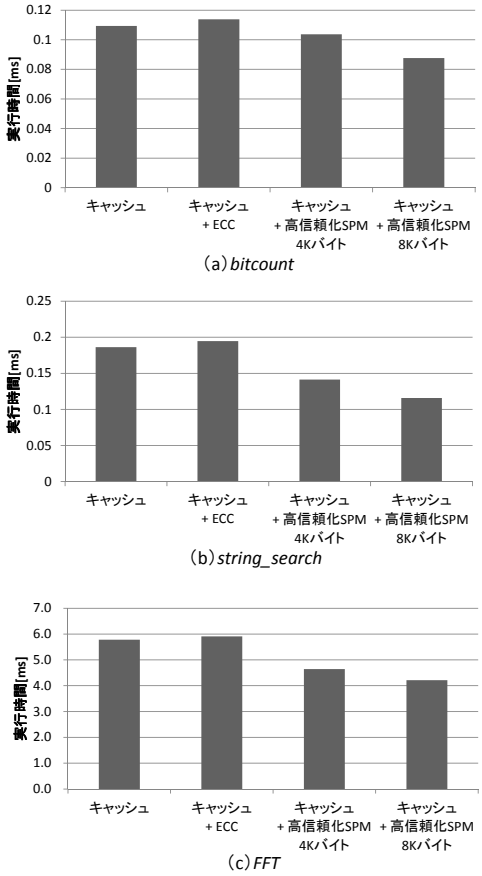

図 9 高信頼化 SPM 併用時と ECC を実装したキャッシュ使用時のベンチマークプログラムの実行時間

ECC を実装したキャッシュを使用する環境では，全てのメモリオブジェクトが高信頼なキャッシュで使用されることになるためソフトエラー耐性は非常に高くなるが，プログラム実行時間増加と面積オーバヘッドは回避できない．一方で提案した高信頼化 SPM を併用する手法は，ECC を実装したキャッシュよりソフトエラー耐性が低いと考えられるが，プログラムの実行時間増加なしで，かつ，少ない面積オーバヘッドで，ある程度のソフトエラー耐性向上が実現できる手法であると言える．

4.2.5 脆弱性尺度の誤差の検証

この節では高信頼化 SPM シミュレータを用いて評

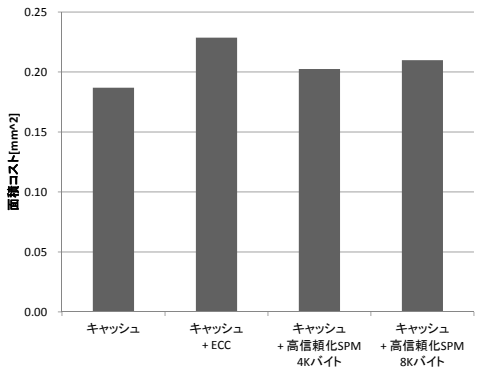

図 10 高信頼化 SPM 併用時と ECC を実装したキャッシュ使用時の面積コスト

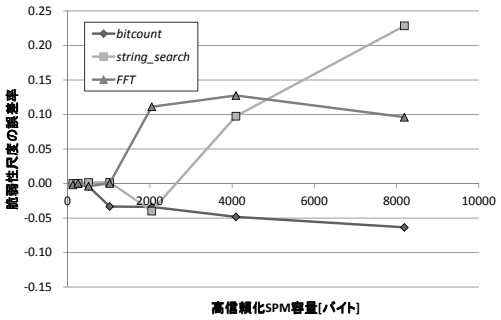

図 11 各ベンチマークプログラムにおける脆弱性尺度の誤差率

価し求めた脆弱性尺度（以下 VF_{sim}）と，本研究で提案した配置手法より決定した配置の脆弱性尺度（以下 VF_{our}）の比較を行う．**図 11** に各ベンチマークプログラムにおける脆弱性尺度の誤差率を示す．ここで脆弱性尺度の誤差率とは，VF_{sim} を基準にして VF_{our} と誤差がどれくらいあるのかを表している（誤差率 $= VF_{our}/VF_{sim} - 1$）．脆弱性尺度の誤差率が正の数ならば $VF_{sim} < VF_{our}$ であり，負の数ならば $VF_{sim} > VF_{our}$ をそれぞれ表している．

図 11 から，*bitcount*, *string_search*, 及び *FFT* の場合においてそれぞれ最大約 6.3%, 13%, 及び 23%の誤差が生じていることがわかる．誤差が生じる原因は 2 つ考えられる．1 つはメモリオブジェクトの高信頼化 SPM への配置に伴う他のメモリオブジェクトの脆弱性尺度の増加である．例えば，メモリオブジェクト x が高信頼化 SPM に配置される場合，メモリオブジェクト x はキャッシュを使用しなくなる．メモリオブジェクト x のキャッシュ不使用に伴い，メモリオブジェクト x と同じキャッシュラインを使用していたメモリオブジェクト y のクリティカルタイムがメモリオブジェクト x の生存期間分だけ増加すると考えられる．実際に調査した結果，脆弱性尺度が増加している全てのメモリオブジェクトが高信頼化 SPM へ配置したメモリオブジェクトと同じキャッシュラインを SPM 未実装時に使用していた．もう 1 つの原因はキャッシュミスの減少である．SPM の使用によりキャッシュミス及び主記憶アクセスが減少し，実行サイクル数が削減される．これに伴いクリティカルタイムも短くなり脆弱性尺度が減少したと考えられる．1 つ目の原因の影響を強く受けた結果が $VF_{sim} < VF_{our}$ であり，2 つ目の原因の影響を強く受けた結果が $VF_{sim} > VF_{our}$ であると考えられる．以上のように VF_{sim} と VF_{our} にはある程度の誤差が生じることから，提案手法より求めた配置方法よりソフトエラー耐性が向上する配置方法の存在する可能性があると考えられる．

5. ま と め

本研究では組込みシステムのキャッシュのアクセスレイテンシを増加してプログラムの実行時間増加せずに，ソフトエラー耐性を向上することを目的とした．この目的を果たすために，小容量の高信頼化した SPM をキャッシュと併用する手法，及び高信頼化 SPM へのメモリオブジェクト配置手法を提案した．各種ベンチマークプログラムを用いて計算機実験を実行した結果，本提案手法により高信頼化 SPM 未実装時と比較して，最大 12%の面積オーバヘッドで最大 72%のソフトエラー耐性を向上し，副次的効果として最大 38%のプログラムの実行サイクル数を削減した．

本研究の高信頼化 SPM へのメモリオブジェクト配置手法の適用によりある程度のソフトエラー耐性向上は見込める．一方で提案した配置手法により求まった配置方法が全ての配置方法の中で最もソフトエラー耐性を向上させる最適な配置方法であると断言はできない．今後の課題として，ソフトエラー耐性が最も向上するメモリオブジェクト配置方法を探し出す手法を提案することである．

謝辞 本研究の一部は，科学技術振興機構 (JST) の戦略的創造研究推進事業 (CREST) の研究領域「ディペンダブル VLSI システムの基盤技術」の支援の下に推進されました．

参 考 文 献

1) R. C. Baumann, “Radiation-Induced Soft Errors in Advanced Semiconductor Technologies,” *IEEE Transactions on Device and Mate-*

rials Reliability, Vol.5, No.3, pp.305-316, Sept. 2005.
2) P. Shivakumar, M. Kistler, S. W. Leckler, D. Burger, and L. Alvisi, "Modeling the Effect of Technology Trebd on the Soft Error Rate of Combinational Logic," In *Proc. International Conference on Dependable Systems and Networks*, pp.389-398, June. 2002.
3) C. W. Slyman, "Cache and Memory Error Detection, Correction, and Reduction Techniques for Terrestrial Servers and Workstations," *IEEE Transactions on Device and Materials Reliability*, Vol.5, No.3, pp.397-404, Sept. 2005.
4) N. N. Sadler and D. J. Sorin, "Choosing an Error Protection Scheme for a Microprocessor's L1 Data Cache," In *Proc. International Conference on Computer Design*, pp.499-505, Oct. 2006.
5) R. Banakar, S. Steinke, Bo-Sik. Lee, M. Bakakrishnan, and P. Marwedel, "Scratchpad Memory: A Design Alternative for Cache On-chip Memory in Embedded Systems," In *Proc. 10th International Symposium on Hardware/Software Codesign*, pp.73-78, May. 2002.
6) S. Steinke, L. Wehmeyer, Bo-Sik. Lee, and P. Marwedel, "Assigning Program and Data Objects to Scratchpad for Energy Reduction," In *Proc. Design, Automation and Test in Europe Conference and Exhibition*, pp.409-415, March. 2002.
7) O. Avissar, R. Barua, and D. Stewart, "An Optimal Memory Allocation Scheme for Scratch-Pad-Based Embedded Systems," *ACM Transactions on Embedded Computing Systems*. Vol.1, No.1, pp.6-26, Nov 2002.
8) K. Lee, A. Shrivasta, N. Dutt, and N. Venkatasubramanian, "Partitioning Techniques for Partially Protected Caches in Resource-Constrained Embedded Systems," *ACM Transactions on Design Automation of Electronic Systems*, Vol.15, No.4, Sept. 2010.
9) G.-H. Asadi, V. S. Mehdi, B. Tahoori, and D. Kaeli, "Balancing Performance and Reliability in the Memory Hierarchy," In *Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software*, pp.269-279, Mar. 2005.
10) T. M. Austin, "The Simplescalar Tool Set as an Instructional Tool: Experimences and Future Directions," In *Proc. 1998 Workshop on Computer Architecture Education*, 1998.
11) M. R. Guthaus, J. S. Ringenberg, D. Emst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," In *Proc. IEEE 4th Annual Workshop on Workload Characterization*, pp.3-14, Dec 2001.
12) N. P. Jouppi and S. J. Wilton. "An Enhanced Access and Cycle Time Model for On-Chip Caches," DEC WRL Research Report 93/5, July. 1994.