公開

（事後評価・研究者用）

研　究　報　告　書

「次世代メモリデバイスによるアプリケーションの自動高速化」

研究期間：　2018 年 10 月～2020 年 3 月
研究者番号：　50161
研 究 者：　穐山　空道

1.　研究のねらい

コンピュータの構成要素は CPU、メモリ、ディスク、ネットワークと様々であるが、その性能の伸び率は一定ではない。特にメモリには近年のビッグデータや AI アプリケーションの重要性の高まりに応じてより高い性能が求められている。しかし CPU の性能（単位時間あたりに実行できる浮動小数点演算の数）が「ムーアの法則」に従って比例的に伸びている一方、メモリを構成する DRAM のアクセスレイテンシは 10 年以上に亘り向上していない。そこで本研究では、近年注目されている Approximate memory を用いてアプリケーションを自動的に高速化する手法を研究する。

Approximate memory とはアクセスレイテンシを高速化する代わりにデータに小さな確率でエラー（ビット化け）が入るメモリである。具体的に DRAM では、メモリコントローラから DRAM に送られるコマンド間に定められた待ち時間を短縮しても多くの DRAM チップにおいてほぼ正常に動作することが知られている。

本研究では、Approximate memory のエラー率と高速化のトレードオフをアプリケーションに合わせて動的に設定することで、アプリケーションの最終結果に対してユーザが指定する誤差率を保証した上で最大の高速化を得ることを目指す。Approximate memory の既存研究ではデバイスレベルで実機を使った様々な計測実験が行われる一方、アプリケーションレベルではランダムなエラーモデルに基づいたエラー耐性評価しか行われていない。これらのギャップを埋め実際のアプリケーションから Approximate memory を有効に利用できるようになることが本研究の最終目標である。

2.　研究成果

（1）概要

本研究の 3 点の具体的成果を概説する。

(a) Approximate memory によるエラーがアプリケーションの最終結果に与える影響を軽量に見積もる手法を提案した。本研究の狙いであるエラー率の自動設定を実現するには、既存研究で調べられている DRAM の内部動作単位のエラー率から実際のアプリケーションの最終結果への影響へ変換する必要がある。 既存研究ではこの変換をハードウェアのシミュレータを用いて行っているが、 ハードウェアシミュレータは平均で実機の 900 倍と非常に低速であり機械学習などの時間のかかるワークロードに適用できない。そこで本成果では、エラーが起こりうる DRAM の内部動作の回数をパフォーマンスカウンタと呼ばれる性能測定機能を用いてアプリケーション実行中に測定し、発生した内部動作の回数に応じてアプリケーションのデータにエラーを挿入することで、実機の数倍程度の実行速度でエラーによるアプリケーションの

最終結果の変化を推定することを可能にした。

(b) エラーが混入してもよい混入してはいけないデータが混在している場合に発生する課題の提言および初期的な解決手法を提案した。Approximate memory を用いて適切な結果を得るためには、ポインタなどの管理構造やプログラムを構成する命令列などのエラーが混入してはいけないデータと、エラー耐性のある数値計算に用いるデータなどのエラーが混入してはいけないデータに異なるエラー率を設定する必要がある。一方でメモリは性能要求から各ビットを独立に駆動することは現実的ではないため、エラー率の設定は 4KB ごとなどの大きな単位でしかできない。従って 1 つのデータ構造にエラー混入してよいデータとしてはいけないデータが混在している場合(例えばグラフのノードを表すデータ構造で、次のノードを指すポインタはエラー混入してはいけないがノードのスコアにはエラー混入してもよいなど)、データ構造を分割してメモリ上に配置する必要がありアプリケーションの性能が大きく下がりうる。本研究ではまずこの問題による影響を明らかにするため、広く使われているベンチマークのメモリアクセス情報とソースコードを解析し、多くのアプリケーションにおいてエラー混入してよいデータとしてはいけないデータが混在している可能性が高いことを突き止めた。また分割配置されたデータ構造のうち片一方のデータのアクセスした際にもう一方のデータを同時に読み込むことで性能低下をある程度抑えられることを示した。

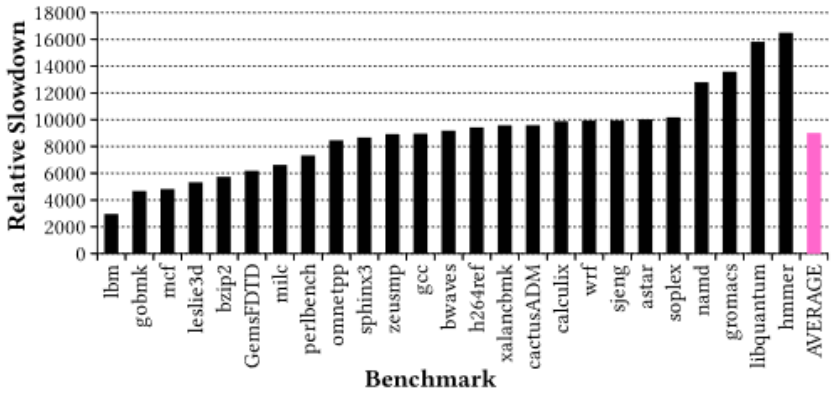
(c) 実機の Approximate memory 上で実際にアプリケーションコードを動作させる end-to-end なシステムを提案した。

(2)詳細

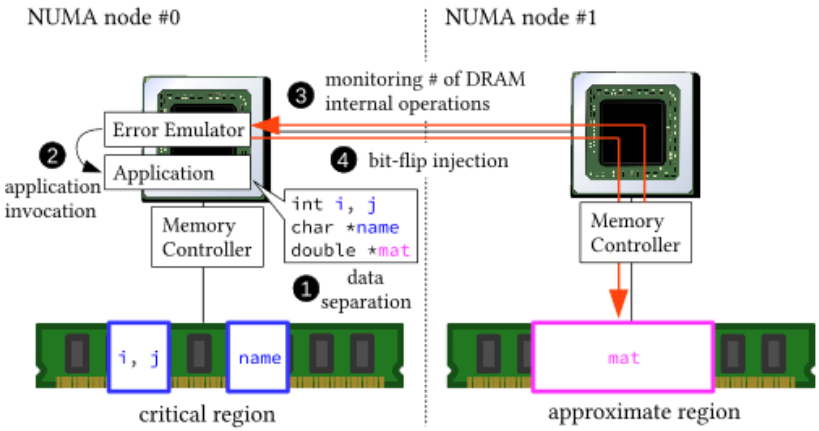
研究成果の詳細

(a) Approximate memory によるエラーがアプリケーションの最終結果に与える影響を軽量に見積もる手法

本研究の狙いである Approximate memory のエラー率の自動制御を達成するためには、発生するエラー(ビット化け)がアプリケーションの最終結果にどのような影響を与えるかを知る必要がある。既存のデバイスレベルの研究では 1 回のメモリ操作あたりのビットエラー率を計測しており、アプリケーションへの影響自体は議論していない。また逆に既存のエラー耐性に関する研究では多くが大規模なデータセンタでのランダムなビット化けを想定しており、発生するエラーの分布はランダムとしている。しかし実際の Approximate memory においてはエラーがメモリ操作ごとに発生するためランダムではなく、実際のエラー発生方式を考慮したエラーモデルによってアプリケーションへの影響を議論する必要がある。実際のエラー発生方式を考慮するためにハードウェアのシミュレータを用いることが考えられるが、一般にハードウェアのシミュレータは非常に複雑で低速である。下図はコンピュータアーキテクチャの研究で非常によく用いられる gem5 というシミュレータでアプリケーションを実行した際の、実機に対する相対実行時間である。実機に対して平均で 9000 倍の時間がかかることが示されている。



そこで本研究では、実機上のメモリコントローラに搭載された計測機能を用いてメモリ操作の回数を計測し、エラーの発生回数を見積もる手法を提案する。ここでメモリ操作とは、メモリコントローラから DRAM に送られるコマンドのうちエラーを発生させると既存研究で明らかになっているものを言う。提案手法ではまず影響を推定したいアプリケーションのソースコードを変更し、エラーが入ってもよい（エラーを入れたい）データを専用のメモリアロケータによって確保する。アプリケーションは CPU を 2 個搭載したマシンの上で実行され、当該メモリアロケータによって確保されたデータは 2 個目の CPU 上が管理するメモリ上に確保される。メモリコントローラに搭載された計測機能ではメモリチャネルごとにメモリ操作の回数を計測可能であり、2 個目の CPU 上のメモリチャネルの値のみを計上することで「エラーを入れたいデータに発生したメモリ操作の回数」を知ることができる。下図はシステムの概念図であり、NUMA node #1 が 2 個目の CPU に該当し、紫色のデータがエラーを入れたいデータである。



提案手法を用いて SPEC CPU Benchmark に含まれる mcf（離散最適化）および milc（量子色力学計算）アプリケーションのエラー耐性を評価したところ、実機の数倍程度のオーバーヘッドで収まることが示された。これはハードウェアのシミュレータを使う場合と比較して 3 桁の性能向上である。本成果は IEICE Transactions on Information and Systems に採択された（主な研究成果の 1）。

(b) エラーが混入してもよい混入してはいけないデータが混在している場合に発生する課題の提言および初期的な解決手法

本研究の狙いを実現するためには、アプリケーションのデータのうちエラー混入してはいけない部分は保護しつつエラー混入してもよい部分へのアクセスを高速化する必要がある。このためにはそれぞれの部分が保存されたメモリ領域に異なるエラー率を適用する必要があるが、Approximate memory の実現方式による制限からエラー率の制御は 4KB ごとなどの大きな単位でしか行えない。Approximate memory は例えば DRAM のコマンド間の待ち時間を仕様よりも短くすることで実現できるが、性能への要求から 1 つのコマンドは多くのメモリセルを同時に駆動する必要がある。従って待ち時間の設定は同時に駆動されるメモリセル全てに一律に適用されるため、エラー率の制御もこの単位でしか行えない。アプリケーションのデータ構造内にエラー混入してもよいデータとエラー混入してはいけないデータが混在している場合、それぞれを 4KB ごとのメモリ領域に分割して配置せねばならずアクセス局所性低下により性能が大きく悪化することが予想される。

本研究ではまずこの問題が重要な課題であるかを調査した。この課題はいままで低減されていなかったものであり、重要性の評価自体も大きな貢献である。調査は以下の手順で行った。まず複数のベンチマークに対して、プログラム中でもっとも多くのキャッシュミスを発生する命令を特定した。次にプログラムのバイナリとソースコードを突き合わせ、もっとも多くのキャッシュミスを発生する命令がアクセスしているデータ構造（例えば struct point 等）を特定した。最後に特定したデータ構造に対してエラーを混入してもよいデータと混入してはいけないデータが混在しているかどうかを予測する基準を 3 つ適用した。基準は以下の 3 点である。C1: データ構造が C の構造体あるいは C++のクラスであるか？ C2: C1 が Yes のとき、その構造体またはクラス内にポインタと非ポインタが混在しているか？ C3: C1 が Yes のとき、その構造体またはクラス内に浮動小数点数が入っているか？調査結果を下の表に示す。Target Data Type は最も多くのキャッシュミスを発生する命令がアクセスしているデータ構造である。結果から、調査したベンチマークのうち多くのもので提言された課題が実際に問題である可能性が高いことが分かる。ここまでの成果（課題の提言およびベンチマークの分析）は The 10th Workshop on Systems for Post-Moore Architectures (SPMA) に査読付き論文として採択された（主な研究成果の 3）。

Name	Target Data Type	C1	C2	C3
milc	struct complex	Y	N	Y
sjeng	struct QTType	Y	N	N
libquantum	struct quantum_reg_node_struct	Y	N	Y
lbm	double	N	-	-
omnetpp	class cChannel	Y	N	N
soplex	struct Element	Y	N	N
gobmk	Hashnode (struct hashnode_t)	Y	Y	N
gcc	struct rtx_def	Y	N	N
mcf	arc_t (struct arc)	Y	Y	N
dealII	double	N	-	-
namd	struct CompAtom	Y	N	Y
Graph 500	int64_t	N	-	-
GraphMat	float	N	-	-

さらに本研究ではこの課題を解決するための初期的な手法を提案した。基本的アイディアは、もともと同一の構造体内にあったデータを同時にメモリからキャッシュにフェッチすることで

ある。メモリアクセスは例えば DRAM のバンクごとなどに並列に行えるため、この手法により 1 回のアクセスと同じレイテンシで 2 か所に分散したデータをキャッシュにフェッチできる。これを実現するため提案手法では元々の構造体のあるメンバにアクセスする際に用いる特別な関数を実装した。本関数を用いてアクセスを行うと分割された構造体に入っていた他のメンバにも同時にアクセスが発生する。提案手法をマイクロベンチマークで評価し、アプリケーションによってはアクセス局所性低下による性能低下を完全に取り戻せることが明らかになった。一方でアプリケーションのメモリアクセスパターンやデータサイズによっては性能低下が改善できないケースもあり、アーキテクチャレベルの詳細な調査およびさらなる研究が必要である。本手法は組込み技術とネットワークに関するワークショップ（ETNET 2019)において電子情報通信学会 コンピュータシステム研究専門委員会の研究会優秀若手発表賞を受賞した（主な研究成果の 2, 4）。

研究目的の達成状況

本研究の目標は Approximate memory の利用に関する本質的な問いを設定しており、ACT-I 期間で完全には達成できていない。しかし一番の課題であるエラー率からアプリケーションの出力への影響をどう見積もるかについてエラーモデルを用いた手法（研究成果の (a)）と実機を用いた手法（研究成果の(c)）の両方面からアプローチし大きく進展していると言える。また Approximate memory の利用に関して当初想定しておらず関連研究も存在しない課題を発見（研究成果の(b)）し重要性の評価および予備的な手法を与えており、Approximate memory を適切に利用するという観点から大きく学術を進歩させていると言える。

3. 今後の展開

今後の研究の展開として、元々の目標であったエラー率の自動制御を達成するためにアプリケーション実行中に動的にエラー率を決める手法を研究する。ACT-I 期間ではこのためにエラーモデル・実機の両方面からエラーが入った際にアプリケーションの結果がどう変わるかを実測する手法を研究してきたが、今後はより視野を広げ数値計算における数値誤差の分析を用いたアプローチ、確率的に値が変わる様子を確率過程としてとらえるアプローチなどを検討する。これまで Approximate memory を含むコンピュータアーキテクチャなどのシステム系分野ではこのような他分野（特に抽象的な数学に近い分野）との協業が少なく、例えばパラメータは多くの実験から経験的に決めるなどのアプローチが取られてきた。今後の研究ではこれにとらわれず様々な分野の研究者と協業することで本研究の目標を達成すると同時にコンピュータアーキテクチャ分野での研究方法においても影響を与えていきたい。

4. 自己評価

研究目的の達成状況

本研究の目標は Approximate memory の利用に関する本質的な問いを設定しており、完全には達成されていない。しかしこれは研究途中で新たな課題（研究成果の(b)）を発見するなどの深掘りを行ったためであり、研究の進捗状況としては全く問題がない。

研究の進め方（研究実施体制及び研究費執行状況）

研究実施体制及び研究費執行状況はほぼ予定通りであり問題がない。第 2 年次の研究費において一部予定していた学会出張を ETH Zurich（スイス）滞在中の国内会議の旅費に振り替えたが、これは当該滞在が本研究を進めるうえで大変有意義なものであったという点で必要な措置であったと言える。

研究成果の科学技術及び学術・産業・社会・文化への波及効果

今回の研究成果はいずれも類似の手法・提案がほぼないものであり、学術への波及効果は高いと考える。特にこれまで重要視されていなかった新たな課題を提言した点、これまで行われていなかった実機の Approximate memory でのアプリケーション実行はインパクトが大きい。また産業においては Approximate memory の導入は未だ進んでいないが、本研究の推進によって利用可能性が拡がればメモリで律速されている多くのアプリケーションが高速化できることが期待される。

研究課題の独創性・挑戦性

本研究課題の独創性・挑戦性は非常に高いと考えている。国内では Approximate memory の研究を行っている研究者は私の知る限り他にいない。また海外でもデバイスレベルの計測研究やアプリケーションレベルのエラー耐性研究は行われているもののそれらを繋いで総合的に Approximate memory の利用可能性を促進する研究はほとんど行われていない。

5. 主な研究成果リスト

（1）論文（原著論文）発表

1. Soramichi Akiyama. A Lightweight Method to Evaluate Effect of Approximate Memory with Hardware Performance Monitors. IEICE Transactions on Information and Systems. 2019, Vol. E102-D, No. 12, pp. 2354 - 2365.

（2）特許出願

なし

（3）その他の成果（主要な学会発表、受賞、著作物、プレスリリース等）

主な学会発表

2. 穐山 空道, 塩谷 亮太. Approximate Memory のデータ分離に起因する性能低下を抑制するプリフェッチ手法. 組込み技術とネットワークに関するワークショップ (ETNET 2019), March 2019.
3. Soramichi Akiyama. Assessing Impact of Data Partitioning for Approximate Memory in C/C++ Code. The 10th Workshop on Systems for Post-Moore Architectures (SPMA), pp. 1 - 7, April 2020 (to appear).

受賞

4. 電子情報通信学会 コンピュータシステム研究専門委員会（CPSY）研究会優秀若手発表賞, 2019 年 7 月.