情報処理学会論文誌　コンピューティングシステム　Vol.16　No.1　14–28 (July 2023)

DRAMの設計余裕を活用した低レイテンシ化・低消費電力化手法とその制御法の研究動向

穐山 空道[1,2,a)]　山田 淳二[3]　塩谷 亮太[2]

受付日 2022年8月26日, 採録日 2023年4月11日

概要：コンピュータのメインメモリを構成するDRAMのランダムアクセスレイテンシと消費電力は大きな課題である．前者は20年以上にわたりほぼ改善しておらず，後者は近年のメモリ搭載量の増大からコンピュータ全体の消費電力の大きな割合を占めるに至る．この問題に対し，DRAMの設計余裕を活用した低レイテンシ化，低消費電力化が着目されている．DRAM内の電気的操作は最悪ケースを想定し余裕を持ったタイミングで行うよう設計されている．そのため，たとえば規定の待機時間を待たず操作しても多くの場合正常に動作し，これを利用するとレイテンシと消費電力が削減できる．これを本論文では「DRAMの設計余裕活用技術」と呼ぶ．しかし本技術からアプリケーションが恩恵を得るには，正常動作せずデータが壊れる場合に対処する必要があり，多くの研究が行われている．本論文では，(1) DRAMおよびDRAMの設計余裕活用技術の動作原理，(2) 設計余裕活用技術を適用してもデータが破壊されることを防止する研究，(3) 設計余裕活用技術の適用時にデータが壊れることを仮定したうえでアプリケーションから意味のある計算結果を得る研究，(4) 残された技術的課題，のそれぞれを論じ本問題に対する研究動向をまとめる．

キーワード：メインメモリ，DRAM，ランダムアクセスレイテンシ，消費電力，設計余裕

Research Trend of Low-latency and Low-power DRAM Technologies that Exploit Design Margins

Soramichi Akiyama[1,2,a)]　Junji Yamada[3]　Ryota Shioya[2]

Received: August 26, 2022, Accepted: April 11, 2023

Abstract: The random-access latency and energy consumption of DRAM are challenging issues. The latency has not improved for decades, and the energy accounts for a significant part in modern computers. To mitigate this issue, exploiting the design margin of DRAM is proposed because DRAM can usually operate correctly with timing violations. However, these exploitation techniques can cause data losses, and considering them is essential. In this paper, we discuss and survey (1) the operations of DRAM and margin exploitation techniques, (2) prevention of data losses, (3) accepting data losses in applications, and (4) remaining problems.

Keywords: main memory, DRAM, random access latency, energy consumption, design margins

1. はじめに

コンピュータのメインメモリは2つの側面で性能のボトルネックになっている．第1に，ランダムアクセスのレイテンシは20年以上にわたり改善していない．ここでランダムアクセスレイテンシとは，**ランダム**なアドレスに対し次々とデータを要求するときあるデータが到着してから次のデータが到着するまでの時間と本論文では定義する．な

[1] 立命館大学情報理工学部
College of Information Science and Engineering, Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
[2] 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[3] キオクシア株式会社第二メモリ設計部メモリ設計第三担当
Memory Design Group 3, Memory Design Department 2, KIOXIA Corporation, Minato, Tokyo 108–0023, Japan
[a)] s-akym@fc.ritsumei.ac.jp

お連続するアドレスにアクセスする場合のレイテンシついてはCPUのプリフェッチ機構により大きく隠蔽されるため本論文の対象外である．一方CPUのコアあたりの性能は向上し続けており [1]，メインメモリのランダムアクセスレイテンシとの差は拡大し続けている．第2に，メモリによる消費電力はコンピュータ全体の消費電力の無視できない割合を占めるようになっている．これは機械学習やビッグデータ処理のような大量のデータを扱う計算要求の高まりによる搭載メモリ量増大が原因である．

このボトルネックを解消するため，メモリを構成するデバイスであるDRAM（Dynamic Random Access Memory）内に存在する設計余裕の活用による低レイテンシ化，低消費電力化が研究されている．このような技術を本論文ではDRAMの「設計余裕活用技術」と呼ぶ．設計余裕は幅広い温度・湿度などの環境下で確実な読み書きを保証するために存在する．設計余裕の一例として，保持データを表現するキャパシタへの電荷溜め直し間隔の伸長があげられる．このキャパシタは微小であり定期的に電荷の溜め直しが必要だが，溜め直し間隔を仕様より長くしてもほとんどの場合でデータは失われないことが知られている．この性質を利用し意図的に溜め直し感覚を伸ばすことで溜め直しに必要な電力が減り，また溜め直し操作の代わりに読み書き操作ができるため平均レイテンシが減る [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13].

DRAMの設計余裕活用技術を有効に機能させるためには，その適切な制御が必要である．適切な制御とは，設計余裕活用技術を適用したDRAMのうえで実行されるアプリケーションが意味のある計算結果を出力できるよう，どの程度設計余裕を活用するかを調整することである．たとえばキャパシタの電荷溜め直し間隔を無制限に伸ばすと記録されたデータを保持できないため，適切な溜め直し間隔を選ぶ制御が必要である．しかし適切な制御には様々な困難が存在する．たとえば待ち時間をどの程度まで削減しても正常に動作するのかはDRAMのベンダや個体により大きく異なることが知られている [14], [15].

DRAMの設計余裕活用技術の適切な制御のため，多くの研究がなされている．本論文では以下の流れに沿ってこれらの研究の動向を明らかにする．2章ではDRAMのランダムアクセスレイテンシと消費電力がどのようにコンピュータ性能のボトルネックになっているかを述べる．3章ではDRAMの動作原理を説明する．4章ではDRAM設計余裕活用の動作原理と，そのデメリットであるビット反転の発生について説明する．5章ではデメリットに対処するための研究群を分類する．6章ではこのうちビット反転を防止する研究について，また7章ではビット反転を限定的に受容する研究についてそれぞれ詳細に説明する．8章で残された技術的課題を述べる．

2. 背景：DRAMを取り巻く課題

本章ではDRAMのランダムアクセスレイテンシと消費電力がいかにコンピュータ性能のボトルネックとなっているかを説明する．

2.1 ランダムアクセスレイテンシ

DRAMのランダムアクセスレイテンシは長きにわたり改善していない．本論文では以降DRAMのランダムアクセスレイテンシを縮めてDRAMのレイテンシと呼ぶ場合がある．文献 [14] はDRAMの仕様策定団体であるJEDEC [16] による各仕様 [17], [18], [19] からDRAMのレイテンシの構成要素の年代変化を纏めている（文献 [14], Figure 1）．それによればDRAMのレイテンシは1999年から2015年までの間に合計で数%しか変化していない．また2020年に公開されたより新しい仕様であるDDR5 [20] でもこれらの値はほぼ一定である．DRAMのレイテンシが年代を経ても改善しない理由は3章で述べるが，大まかには半導体の微細化により本来は悪化するレイテンシを回路設計の工夫により一定程度にとどめているためである．

DRAMのレイテンシはアプリケーション性能に大きな影響を与える．特にCPUのキャッシュやプリフェッチによるレイテンシ隠蔽が有効でないケース（アクセスされるアドレスが不規則な場合）にこの影響が顕著である．たとえばCPU性能を計測するベンチマークであるSPEC CPU 2006 [21] に含まれる組み合わせ最適化アプリケーションであるmcfを4コアのアウトオブオーダCPUとDDR4メモリで実行すると，CPI（Cycles Per Instruction）のうち約半分がDRAMのレイテンシによるストールである [22].

DRAMのレイテンシが改善しないことでCPUが無駄に消費する電力も増大している．第1に，CPUがDRAMからのデータを待ちストールしている間でも消費電力は0にならない．これはストール中にもクロック信号やリーク電流によって電力を消費するためである．したがってDRAMのレイテンシが改善しなければCPUの高速化に応じストールの割合が増え無駄な消費電力の割合も増大する．第2に，DRAMのレイテンシ隠蔽のためにCPUは大きなキャッシュ（近年では数MBから数10 MB）を搭載しキャッシュの消費電力が増大している．文献 [23] によれば8コアのスーパースカラCPUの消費電力のうち25%以上がL3キャッシュとL2キャッシュに消費される．またキャッシュの消費電力削減の研究 [24], [25] がさかんなこともキャッシュの消費電力が課題であることを示す．

2.2 消費電力

DRAMによる消費電力はシステム全体の消費電力のうち無視できない割合を占める．文献 [26] は2ソケット構成のサーバに32 GBのDDR3メモリを搭載するとDRAM

表 1　近年のスーパコンピュータの 1 ノードあたりのメモリ搭載量
Table 1　Amount of memory per node in recent supercomputers.

名称	設置年	ノードあたりメモリ量
富岳	2021	32 GiB (HBM2) [28]
Summit	2018	512 GB (DDR4), 96 GB (HBM2) [29]
Sierra	2018	256 GB (DDR4) [30]
Sunway TaihuLight	2016	32 GB (DDR3) [31]
Perlmutter	2021	512 GB (DDR4) [32]

の消費電力はシステム全体の約 19%を占めると報告している．また文献 [27] によると 2007 年の Google 社のデータセンタのコンピュータの消費電力のうち 30%が DRAM により消費された．

また近年はマシンあたりの DRAM の搭載容量が非常に多い．これは人工知能やビッグデータ解析などの多量のメモリを必要とするアプリケーションによる要請が原因である．**表 1** はスーパーコンピュータ性能の世界ランキング「Top 500」の 2021 年 11 月のランキング [33] に掲載された上位 5 件のシステムの 1 ノードあたりのメモリ容量であり，最大でノードあたり 512 GB ものメモリが搭載されている．なお容量に付随した括弧内にはメモリデバイスの種類を記した．DRAM による消費電力は同一規格の DIMM を使用する限りは DIMM 枚数に比例し増加するため，ノードあたり容量が大きなシステムでは DRAM による消費電力も大きくなる．たとえば表 1 の Summit のスペックである 512 GB の容量実現には 32 枚の 16 GB DIMM が使用されている [34].

3. DRAM の動作原理

本章では本論文の理解に必要な DRAM の動作原理について説明する．なおここでの説明は本論文に必要な部分のみに着目するため，より詳細な情報を知るには文献 [35] や文献 [36] などを参照のこと．

3.1 DRAM の構造と読み書き動作

図 1 は 1 枚の DRAM モジュールの内部構造の概略図である．1 枚の DRAM モジュールは *DIMM*（*Dual Inline Memory Module*）とも呼ばれる．DIMM には *DRAM chip* が複数搭載されており，1 個の DRAM chip の内部は互いに独立に読み書き可能な *bank* と呼ばれる構造に分かれている．1 つの bank 内には *cell* が行列上に並び，横 1 行を *row*，縦 1 列を *column* と呼ぶ．cell 内のキャパシタの電荷の有無が 1 または 0 を表し[*1]，*sense amplifier* と呼ばれる回路で電荷の有無の判断やチャージを行いデータを読み書きする．

[*1] 電荷ありが 1 を表す通常のセル（*true-cell*）と，電荷なしが 1 を表す反セル（*anti-cell*）が半数ずつ存在する．

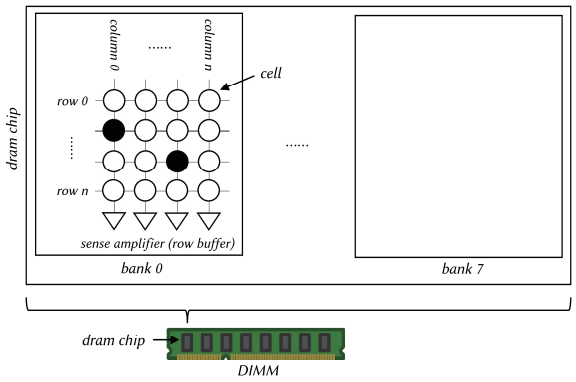

図 1　DRAM 構造の概略：1 つの DRAM chip は複数の bank で構成され，各 bank 内にはデータを保持する cell が行列上に並ぶ
Fig. 1　Overview of DRAM internals.

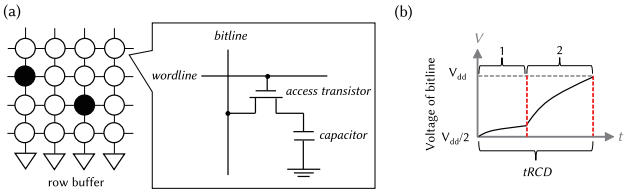

図 2　(a) cell 周辺の詳細な構造：キャパシタにデータを表す電荷が蓄えられ，トランジスタを介し bitline とつながっている．(b) cell の電荷がある場合の activation 時の bitline 電圧変化：cell の電荷により bitline の電圧が上がり，その電圧上昇を sense amplifier で増幅する
Fig. 2　Detailed structure of a cell.

図 2 を用い DRAM の読み出し動作を説明する．図 2 (a) は cell を拡大した図である [37]．cell のキャパシタは *access transistor* を介して *wordline* と *bitline* に接続されており，次の手順で値が読み出される．

(1) **Activation**：まず対象 cell が属する row を *enable* する．電気的には当該 row の wordline に電圧をかけ，cell と bitline を隔てる access transistor を導通させる．これにより cell のキャパシタに蓄えられた電荷の有無に応じ bitline の電圧が変化する．図 2 (b) はキャパシタに電荷がある場合の電圧変化の概念図である．bitline は電源電圧 V_{dd} の半分にリセットされており，電荷がキャパシタから流入することで電圧が上昇する（図 2 (b) の領域 1）．この電圧変化を sense amplifier と呼ばれる回路で増幅（図 2 (b) の領域 2）し cell の電荷の有無を検知する．電圧変化が十分大きくなるまでの時間は *tRCD* と呼ばれるパラメータで定義される．この動作全体を *activation* と呼ぶ．

(2) **Restore**：キャパシタの電荷を activation 前の状態に戻す．この操作を *restore* と呼ぶ．cell のキャパシタは微小であり容量が bitline の寄生容量よりも小さい．そのため activation 後には保持していたデータが消失した状態になっておりこれを元に戻す．

(3) **Column read**：sense amplifier から読み出し対象の

column を選択し，マルチプレクサを通して CPU 側に転送する．この操作を *column read* と呼ぶ．本操作は restore と同時に実行可能である．このとき sense amplifier が読み出し対象 row のバッファとして働くことから，sense amplifier を *row buffer* とも呼ぶ．

(4) **Precharge**：次の読み出しに備えるため，sense amplifier，bitline，wordline の電圧をリセットする．sense amplifier と bitline は $\frac{V_{dd}}{2}$ に，wordline は V_{ss} と呼ばれる電圧にリセットされる．V_{ss} は access transistor の導通を完全に切る目的から $\frac{V_{dd}}{2}$ よりも低く設定することが普通である．この操作を *precharge* と呼び，ある row の activation 開始後に precharge が開始できるまでの時間は *tRAS* と呼ばれるパラメータで定義される．また precharge を開始してから完了するまでの時間は *tRP* と呼ばれるパラメータで定義される．

書き込み：DRAM への書き込みは次の手順で行われる．(1) 書き込み対象のデータが属する row を activation する．これにより読み出し時と同様に sense amplifier が当該 row のバッファとして働き，バッファに蓄えられたデータへの上書きが可能になる．(2) CPU 側から送られたデータで row buffer を上書きする．row のサイズは CPU から送られくるデータサイズ（64 バイト）よりも大きいことが普通であり，row buffer のデータの一部のみが上書きされる．(3) row buffer から row へ restore と同様に電荷を戻す．この時電荷が十分な量たまるまでに待機すべき時間は *tWR* で定義される．(4) precharge を行い次の activation に備える．

refresh：cell のキャパシタは微小でありわずかな時間で電荷量が減少するため，データ保持のために定期的な電荷の溜め直しが必要である．この溜め直しを refresh 操作と呼ぶ．各 row が refresh されるべき間隔は *tREF* と呼ばれ，一般に 64 ms である．たとえば Micron 社の MT40A1G8 および MT40A51M16 という製品のデータシート [38] には，動作温度が −40°C から +85°C の条件では refresh time が 64 ms とある．また refresh 操作をメモリコントローラが要求すべき平均間隔を *tREFI* と呼ぶ．tREFI は DDR4 で動作温度 85°C 以下では 7.8 us である [19]．これは 64 ms の間に 8,192 回の refresh 操作要求を送ることに等しい（$\frac{64\times10^{-3}}{8192} \approx 7.8\times10^{-6}$）．なお DRAM 内の全 row 数が 8,192 であるとは限らず，全 row 数に応じて一度の refresh 要求で refresh される row 数が変わる [39]．

3.2　DRAM のレイテンシの構成要素

図 3 に DRAM のランダムアクセスレイテンシの構成要素を示す．図の横軸は時刻を表し，横軸の下の矢印はそれぞれの操作を開始してから完了するまでの時間を表す．CPU が DRAM にデータを要求してから当該データが返るまでの時間は，DRAM の内部状態によって図の (a)，(b)，(c)，(d) の 4 通りになりうる．

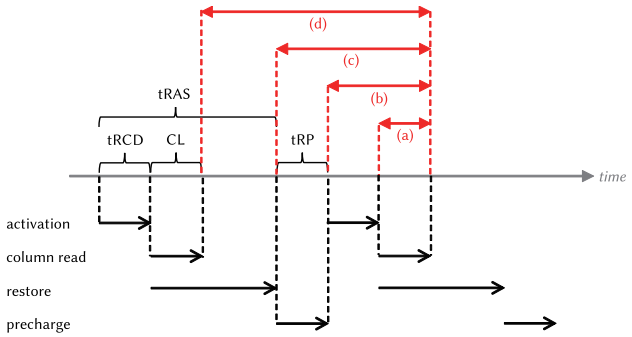

図 3　DRAM のランダムアクセスレイテンシの構成要素．(a)，(b)，(c)，(d) は CPU がデータを要求してから当該データが返るまでの時間のありうるパターンである

Fig. 3　Breakdown of DRAM access latency.

(a)　読み出し対象データの属する row が activation 済みで row buffer に格納されている状態であり，レイテンシが最も短い．これを *row buffer hit* とも呼ぶ．この場合のレイテンシは CL で決まる [40]．

(b)　Row buffer が precharge され空の状態であり，読み出し対象データの属する row を activation する必要がある．この場合のレイテンシは tRCD + CL で決まる．

(c)　読み出し対象データの属する row とは異なる row が row buffer に格納されている状態であり，precharge を行い読み出し対象データの属する row を activation する必要がある．この場合のレイテンシは tRP + tRCD + CL で決まる．

(d)　row buffer miss に加えさらに 1 回前の activation にともなう restore が完了していない状態であり，レイテンシが最も長い．この場合のレイテンシは tRP + tRAS で決まり，この和は *tRC* とも呼ばれる．

上記のうち (b) と (c) はともに row buffer に読み出したい値が入っていないという意味で *row buffer miss* と呼ばれる．Row buffer が precharge されているかいないかは DRAM のスケジューリング方針により決まる．なるべく precharge を行い異なる row の読み出しに備える方針を *closed-page policy* と呼び，なるべく precharge を行わず同一の row の読み出しに備える方針を *open-page policy* と呼ぶ．

具体的にたとえば DDR4-3200 W [19] では tRCD，tRP は 12.5 ns，tRAS は 32 ns，CL はデバイスによって複数の値を取れるがおよそ tRCD と同程度である（文献 [19]，Table 113）．したがって DDR4-3200 W のレイテンシは最良ケースでは 12.5 ns 程度，最悪ケースでは 44.5 ns である．なおソフトウェアから見たメモリシステム全体のレイテンシはキャッシュを参照する時間などを追加で含みこれより長い．

3.3　レイテンシが改善しない理由

2.1 節で述べたように，DRAM のレイテンシは長年の間ほとんど改善していない．この理由は以下の 2 点に分けて説明できる．

(1) DRAM のレイテンシは主に配線遅延によって決定されること
(2) 集積回路の配線遅延は製造技術が進み微細化しても改善しないこと

3.2 節で示したように，DRAM のレイテンシはキャパシタや電源からの電荷の移動にかかる時間で構成される．たとえば tRCD はキャパシタから bitline に電荷が移動し bitline の電圧が変化するまでの時間と，さらに電源から bitline に電荷が移動しその電圧変化が増幅されるまでの時間から成る．これらは回路の配線遅延によって決定される．

配線遅延は半導体が微細化しても改善せず，むしろ悪化することもある．一般に回路の配線遅延はその配線の抵抗と寄生容量によって定まる．このうち抵抗に関しては微細化が進むと配線が細くなることで増加し，配線遅延は悪化する．DRAM ではこれに対抗するため bank 内を複数の回路単位に分割することで配線長を短くし配線遅延を増加させない努力がなされている [36]．このように DRAM のレイテンシは微細化により悪化する傾向を回路設計の工夫により一定程度にとどめているといえる．

4.　設計余裕活用技術の共通原理

本論文では DRAM 内に存在する設計余裕を活用することで低レイテンシ化，低消費電力化を実現する研究をサーベイする．この技術を本論文では DRAM の「設計余裕活用技術」と呼ぶ．

本章では DRAM の設計余裕活用技術に共通するレイテンシ削減と消費電力削減の原理および本技術のデメリットについて述べる．

4.1　待機時間の削減

3 章で示したように，DRAM の各操作の後には電気的に安定な状態になるまでの待ち時間（例：tRCD）が定められている．DRAM の設計余裕活用技術の 1 つ目は，この待ち時間を仕様を逸脱する範囲まで削減することである [14], [15], [41], [42], [43], [44], [45], [46], [47]．これには次の 2 点の効果がある．

(1) レイテンシ削減：たとえば tRCD を削減する場合を考える．tRCD を削減すると，3.2 節で示したパターンのうち (b) と (c) でレイテンシが削減される．具体例として，DDR4-3200 W の仕様では tRCD は 12.5 ns であり，この仕様を逸脱し tRCD を 7.5 ns に削減するとする．パターン (b) では tRP + tRCD + CL が 37.5 ns から 32.5 ns に約 13%削減され，パターン (c) では tRC が 44.5 ns から 39.5 ns に約 11%削減される．
(2) 消費エネルギー削減：この効果はさらに 2 つに分けられる．第 1 に待ち時間を削減すると電気的な操作にかかる時間が短縮され当該操作自体の消費エネルギーが削減される．たとえば activation 時に流れる電流は，メモリコントローラからのアクセス要求のデコード時に流れるものと定常的に流れるものに分けられる [48]．tRCD を短縮し activation を早く終了すれば，定常的に流れる電流が消費するエネルギーは tRCD の短縮率に比例して削減される．第 2 に DRAM のレイテンシ削減によりアプリケーション実行が高速化されるため，アプリケーションの完了までに必要なエネルギーが削減される．メモリアクセスが高速化されることにより CPU のストール時間が減り，その間に消費される無駄なエネルギーが削減できる．

4.2　Refresh 間隔の伸長

DRAM の設計余裕活用技術の 2 つ目は，tREF すなわち各 row が refresh される間隔を大きくすることである [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [49]．これには次の 2 点の効果がある．

(1) レイテンシ削減：ある bank の refresh 中には同じ bank に対しその他の操作が行えないため tREF を大きくするとレイテンシが減少する．refresh 操作の開始後にその他の操作を実行できない時間は *tRFC* というパラメータで規定される．ある row を refresh するには当該 row に属する各 cell の電荷の有無に応じ電荷の溜め直しを行う．これはその row を activation し restore することと同じであり，refresh 操作中は sense amplifier が専有され他の操作ができない．
(2) 消費電力削減：refresh 操作を間引くと消費電力を大きく削減できる．refresh 操作では電源から bitline を通りキャパシタに電荷が移動し大きな電力を消費する．たとえば Micron 社による試算 [48] では 8 Gb の DDR4 2666 モジュールの消費電力 408.3 mW のうち，refresh 操作に消費する電力は 22.5 mW と全体の約 5.5%である．また文献 [50] によれば 32 Gb の DRAM モジュールでは refresh 操作がモジュールの消費電力の 20%以上を占めた．

4.3　課題：ビット反転率の向上

待機時間の削減と refresh 間隔の伸長に共通の課題は，保持データへのビット反転混入確率の向上である．概念的には，待機時間や refresh 間隔は守るべき仕様であるためそれを破ると正常な動作が保証されずビット反転が発生することがある．

図 4 に refresh 間隔を伸長する場合のビット反転発生原理の概念図を示す．図は上から 2 番めの row の refresh 間隔を 64 ms より大きくした場合の例である．当該 row の一

表 2　設計余裕活用技術の研究群の分類
Table 2　Categorization of research on exploiting design margins.

		活用する設計余裕	
		待機期間（4.1 節）	refresh 頻度（4.2 節）
ビット反転への対処法 防止	cell 内電荷量の時間的違いの利用（6.1 節）	文献 [41], [43], [45]	-
	製造ばらつきの利用（6.2 節）	文献 [14], [15], [42], [46]	文献 [2], [4], [5], [6], [7], [8], [9]
ビット反転への対処法 受容	OS のメモリ管理との協調（7.1 節）	-	文献 [3], [10]
	DNN の特性の利用（7.2 節）	文献 [47]	文献 [11], [12], [13]

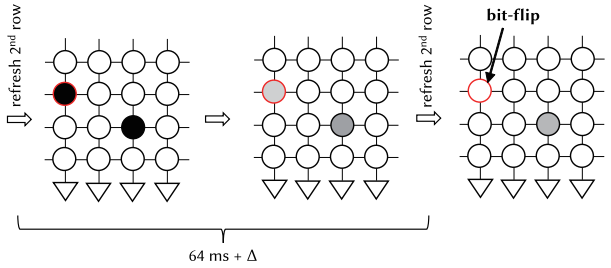

図 4　refresh 間隔が規定の 64 ms より大きくなった場合のビット反転発生原理の概念図：refresh 間隔を伸ばしたことで二行目一番左の cell の電荷が抜けすぎ，refresh が電荷を抜く方向に働く
Fig. 4　How bits can flip when refresh interval is longer than 64 ms.

番左の cell のキャパシタには電荷があり，refresh 操作では電荷が溜め直されるべきである．しかしこの例では refresh 間隔が伸びたため電荷が抜ける時間が長くなり，refresh 開始時の電荷がごく少ない．この状態で refresh 操作を行うと当該 cell に接続される bitline の電圧が「電荷あり」と判断する基準まで達しない．したがって refresh 操作は電荷を抜く方向に働き，refresh 後にはこの cell の電荷はなくなりデータビットが反転する．

また待機時間を削減する場合には，それぞれの待機時間パラメータによって以下のような原理でビット反転が発生する．

tRCD　bitline の電圧変化を早期に打ち切るため，cell の電荷の有無を判定し間違える．

tRP　bitline や row buffer の電圧が基準電圧からずれた値にリセットされるため，その後の activation 操作や refresh 操作で cell の電荷の有無を判定し間違える．

tRAS　restore 操作を早期に打ち切るため，cell の電荷量が正常な値からはずれその後の activation 操作や refresh 操作で cell の電荷の有無を判定し間違える．

tWR　書き込み操作を早期に打ち切るため，tRAS 削減時と同様に原理でビット反転が発生する．

5.　設計余裕活用技術の分類

表 2 は本論文でサーベイする設計余裕活用技術の研究群を分類したものである．行方向は設計余裕活用技術のデメリットであるビット反転への対処方法の違いを，列方向は活用する設計余裕の違いを表す．

ビット反転への対処方法は大きく以下の 2 通りが存在する．これらは根本的な考え方が異なる．

(1) **ビット反転の防止**：DRAM の電気的な特性を考慮し**設計余裕の活用を一定程度に抑える**ことで，ビット反転の発生を**完全に防ぐ**．

(2) **ビット反転の限定的受容**：ビット反転が起きる位置や頻度を**一定の範囲に制限する**ことで，ビット反転が起きても意味のある計算を継続可能にする．

本論文では以降ビット反転への対処方法の違いに沿い各研究を詳しく説明する．この対処を行わないとアプリケーションから見れば予測不可能な時点でデータの予測不可能な位置が壊れる．したがってビット反転への対処は設計余裕活用技術から恩恵を得るうえで最も重要であり，各研究の注力する点である．6 章ではビット反転の防止に関する研究を，7 章ではビット反転の限定的受容に関する研究をそれぞれ詳細に説明する．また表 2 の各行にそれぞれの分類に沿う研究を説明する節番号（例：6.1）を記した．

活用する設計余裕の違いには，4.1 節で原理を説明した待機時間削減と 4.2 節で原理を説明した refresh 頻度の削減がある．**表 3** は待機時間を削減する各研究のそれぞれが削減するパラメータの具体名を示す．

表 3　各研究が削減する待機時間パラメータ
Table 3　Timing parameters shortened in each work.

文献	発表年	tRCD	tRAS	tRP	tWR
[41]	2014	✓	✓		
[42]	2015	✓	✓	✓	✓
[14]	2016	✓		✓	
[15]	2016	✓			
[43]	2016		✓	✓	
[44]	2017		✓	✓	
[45]	2018	✓	✓		✓
[46]	2018	✓			
[47]	2019	✓			

6.　ビット反転の防止

本章ではビット反転の防止により設計余裕活用技術のデ

メリットに対処する研究群の詳細を説明する．ビット反転の防止方法は以下の 2 種類に分類できる．

(1) cell 内電荷量の時間的違いの利用

(2) 製造ばらつきの利用

以下ではそれぞれの分類に沿って各研究を説明する．なお本章で示す各文献の定量評価結果は単純に相互比較できない．各文献でベースラインやアプリケーションなどの実験詳細が異なるが，本論文では過度な煩雑さを避けるためこれらを記していないからである．

また本章ではこれらに加え関連する話題として，DRAM 内に元来存在する設計余裕を活用するのではなく DRAM 内部動作自体の改変により待機時間削減を実現する研究群についても紹介する．なおこれらの研究は設計余裕の活用やその際のビット反転防止を目的としたものではないため，表 2 および表 3 には含めていない．

6.1　cell 内電荷量の時間的違いの利用

この種類の研究は DRAM の内部操作の電気的な特性を分析することで，ビット反転が起きない程度に設計余裕を活用する．具体的には，cell 内のキャパシタの電荷量の時間的な違いを利用し，電荷が多いときには事前にシミュレーションによって決定した削減率に応じ待機時間パラメータを調整する．

cell 内のキャパシタ電荷量の時間的な違いは，cell が最後に refresh されてからの経過時間の違いにより生じる．4.2 節で示したように，cell 内のキャパシタは微小でわずかな時間で電荷が抜けるため定期的な refresh が必要である．これは refresh 直後の row に属する cell と refresh 直前の row に属する cell では電荷量が大きく違うことを意味する．

文献 [41] は電荷量の時間的違いを利用し tRCD を短縮する．Activation 操作では cell 内のキャパシタから bitline に電荷が移動し bitline 電圧が上昇し，上昇速度はキャパシタの電荷が多いほど速い．したがって refresh 直後の row を activation する場合 tRCD を短縮しても bitline の電圧は十分な値まで上昇し誤動作しない．本文献はこのアイディアを元に DRAM アクセスをレイテンシが短い順に優先度付けし，FR-FCFS（First Ready-First Come First Served）スケジューラ [51] の open，close モードに対しそれぞれ 8.1%，7.3%の性能向上を得た．

文献 [43] は tRAS と tWR を，当該 row の次回の refresh が時間的に近い場合に短縮する．tRAS，tWR は restore や書き込みにおいてキャパシタの電荷が溜まるのに十分な時間を元に規定される．しかし次回の refresh が時間的に近い場合はその refresh により電荷が溜められるため，それまでの間データを保持できる程度に電荷が残っていればよい．したがってこの場合は tRAS，tWR を短縮してもビット反転は発生しない．本文献はこのアイディアを実装したメモリコントローラをシミュレーションし，通常のタイミングを用いた DRAM に対し平均 15%の性能向上，平均 17%の電力削減を得た．

文献 [45] は文献 [43] を発展させ tRAS をさらに削減する．第 1 に，tRAS は当該 row への次の activation が時間的に近い場合にも削減できる．これは次回の activation で再度 restore が起こるためである．第 2 に，このときビット反転が起きない最小の長さに tRAS を削減すると全体の性能は必ずしも最大化されない．前述のように activation 時にキャパシタの電化量が多ければ tRCD を削減でき，tRAS を最小にするとこの効果を打ち消すからである．そこで本文献では，ある row へのアクセス時にその row への次回のアクセスと refresh までの時間を予測し，全体の性能を最大化するように tRAS を削減する．本文献はこのアイディアを実装したメモリコントローラをシミュレーションし，文献 [43] を上回る性能向上と消費電力削減を得た．

6.2　製造ばらつきの利用

この種類の研究では，DRAM 内部の回路部品の製造ばらつきにより生じるビット反転の局所性に応じどの程度設計余裕を活用するかを決める．製造ばらつきとは，半導体製造工程における排除できない要因によって容量や抵抗値に違いが出ることである [52]．またビット反転の局所性とは，同じ待機時間や refresh 頻度を適用してもビット反転が起こる場合と起こらない場合があることをいう．たとえば製造ばらつきにより容量が通常より大きい cell では，通常より多くの電荷が抜けても正常動作するため refresh 頻度を下げられる．これらの研究では事前に製造ばらつきによるビット反転の起きやすさの違いを計測し，実行時にその結果を利用する．

6.2.1　製造ばらつきによるビット反転の局所性

製造ばらつきによるビット反転局所性は，待機時間削減時と refresh 間隔伸長時のどちらにも発生する．以下ではそれぞれについて説明する．

待機時間削減時の局所性：文献 [14], [15] によれば，この局所性はさらに (i) DRAM チップごと (ii) 同一 DRAM チップ内の 2 種類がある．(i) の例として，文献 [14] では DDR3-1666 の tRCD を 7.5 ns に短縮しビット反転発生率を調査した．結果として同一ベンダ・同一モデルのチップでも個体ごとに 100 倍以上発生率が異なることを発見した．また (ii) の例として，文献 [15] では DDR3-533/667/800 の tRCD を 8.75 ns に短縮しビット反転発生率を調査した．結果として同一チップ内でも bank ごとにビット反転発生率が大きく異なることを発見した．

Refresh 間隔伸長時の局所性：これに関する初期の研究 [2] では，16 MB の DDR チップの全 row に対しビット反転が発生しない最長の被 refresh 間隔（retention time）を調査した．結果として retention time は実際には 500 ms から 50 秒と大きくばらつくと報告している．またより最

近の研究 [5] では 512 MB の LPDDR チップでは 45°C と 1,024 ms の被 refresh 間隔の条件で全ビットの 4.65×10^{-8}% しか反転しなかった．

6.2.2　ビット反転の局所性を使う研究群

文献 [42], [46] は待機時間削減時のビット反転局所性を利用する．文献 [42] は cell の製造ばらつきにより待機時間削減時に生じるビット反転局所性を利用する．たとえば平均より内部抵抗が低く容量が大きい cell は refresh や restore で溜まる電荷が多い．したがってこの cell は activation 時の bitline 電圧上昇が速く，restore や書き込み時の電荷チャージ速度も速い．この性質を利用し，本文献は row ごとにビット反転を起こさない最小の tRCD，tRAS，tWR，tRP を事前にプロファイルし実行時に利用する．短縮したパラメータを実機のコンピュータ上で設定することで，通常の DRAM を用いる場合に対し平均 10.5%の性能向上を得た．文献 [46] は tRCD を削減する際のビット反転発生率が bitline ごとに大きく異なることを発見した．具体的には，3 つの主要ベンダの LPDDR4 チップに対し tRCD を削減した場合 column のうち平均 3.7%，2.5%，2.2%のみにビット反転が発生した．またアプリケーションのメモリアクセスを分析し，row 内の最初の 64 バイトにアクセスされる確率が高いことを発見した．したがって row 内の一部 column のみ tRCD を削減すれば全体として大きな効果が得られる．本文献はこれらのアイディアを実装したメモリコントローラをシミュレーションし，通常の LPDDR4 を使用する場合に対し平均 4.97%の性能向上を得た．

文献 [2], [4], [9] は refresh 間隔伸長時のビット反転局所性を利用する．文献 [2] ではビット反転が起きない row の合計サイズがアプリケーションの必要メモリ量を上回るような単一の refresh 間隔を適用する．このために retention time が長い row から優先的にメモリ領域を確保する．また文献 [4] では row ごとの retention time を事前に計測し，ビット反転が起きないよう row ごとに異なる refresh 間隔を適用する．refresh 間隔を row ごとに異なる値にするためにメモリコントローラを改変する．別の文献 [9] では retention time が短い cell に格納されるべきデータを ECP（Error-Correcting Pointer）[53] を用い別の cell に格納することで refresh 間隔を伸長する．

6.2.3　Refresh 間隔伸長の実装上の課題

refresh 間隔伸長時のビット反転局所性の利用には各 row に異なる refresh 間隔が必要であり，その実装方法についても研究がある．これは容量の多い DRAM chip では *Auto-refresh*（*AR*）と呼ばれる方式で refresh を行う必要があるためである．

AR 方式では従来方式の refresh（対象 row を activation し precharge する）と異なり対象 row を指定できない．これは refresh 対象 row をメモリコントローラが指定するのではなく AR 要求を受けた DRAM chip が自律的に選ぶためである．この動作は容量増加にともない row 数が増えてもメモリコントローラからの refresh 操作要求間隔を一定に保つために必須である．たとえば Samsung 社のある DRAM chip [54] では row 数は容量に応じ 32 K から 128 K まで様々だが，すべて同じ DDR4 なのでメモリコントローラが refresh 操作を要求すべき間隔は一定である．

従来方式の refresh は AR 方式よりもオーバヘッドが大きいため，refresh 間隔伸長のために単に AR 方式から従来方式に変更することは困難である．これは bank 内は並列動作可能な単位（*sub-array* という）にさらに分割されており [55]，AR 方式ではこの並列性を活かせるためである．文献 [7] では容量が 16 Gb を超えると性能・消費電力の両面で AR 方式の方がオーバヘッドが少ないことを，文献 [8] は容量が 32 Gb の場合には従来方式で refresh 回数を 70% 省いても AR 方式の方が性能オーバヘッドが少ないことを指摘した．

そこで AR 方式使用時も row ごとに refresh 間隔を変更するため，文献 [6], [8] では AR 方式と従来方式の refresh を組み合わせる．文献 [6] は AR 方式では伸長した refresh 間隔を用い，DRAM chip が追加の refresh が必要と判断すると activation 要求をメモリコントローラに送出する．メモリコントローラは受け取った activation 要求を他の要求と同様にスケジューリングし再び DRAM chip へ送出する．これにより DRAM chip が内部的に追加の refresh を行う場合に比べ柔軟なスケジューリングが可能である．また文献 [8] は DRAM chip の内部状態レジスタを読み出すインタフェースが存在することを利用し，AR 方式使用時の refresh 情報をメモリコントローラが読み出す．読み出した情報から次に refresh される row を判断し，その row の retention time が長ければ新たに定義したダミー操作を要求する．ダミー操作を受け取った DRAM chip は refresh に関する情報の更新だけを行い実際には refresh しないことで特定の row の refresh を間引く．

6.3　関連技術：DRAM 内部動作自体の改変

ここでは関連技術として，DRAM の内部動作を変更することで待機時間削減を可能にする研究を紹介する．6.1 節と 6.2 節でレビューした研究群は，DRAM の回路がもともと備えている動作タイミングの余裕を発見し利用することで待機時間削減を実現する．一方ここでレビューする研究群は DRAM の内部構造や動作そのものの変更をともない本サーベイの主題である設計余裕の活用とは直接的には呼べないが，DRAM の低レイテンシ化・低消費電力化のために待機時間を削減する点で関連する．

6.3.1　複数キャパシタの同時利用

待機時間削減を可能にする 1 つ目の手法は，同一のデータを保持するキャパシタを 2 つ（以上）持つすることである．これらを同時に bitline に接続することにより，sense

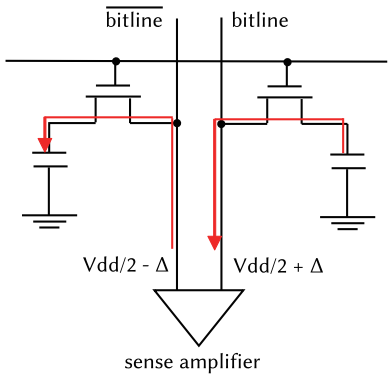

図 5　Twin-cell 構造のセル：1 bit のデータを相反する 2 つのキャパシタで表現する．bitline と $\overline{\text{bitline}}$ の電圧が反対方向に変化し sense amplifier はその差を増幅する

Fig. 5　Structure of a twin-cell.

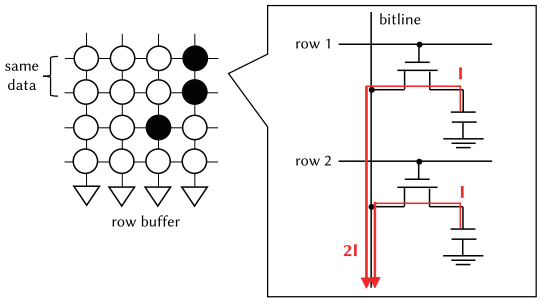

図 6　2 つの同じデータを持つ row を同時に activation する様子：通常の 2 倍の電流が bitline に流れる

Fig. 6　Two rows with the same data are activated simultaneously.

amplifier の検知する電圧変化を高速化したり，少ない電荷でも十分な電圧変化を得られる効果がある．

文献 [56] は Twin-cell と呼ばれる DRAM セルを提案する．本技術は 1 bit のデータを，1 つの sense amplifier に接続される相反する電荷状態を持つ 2 つのキャパシタで表現する．通常の電荷状態のセルを接続する bitline に対し，反対の電荷状態のセルを接続する配線を $\overline{\text{bitline}}$ と呼ぶ．**図 5** にこの様子を示す．Bitline と $\overline{\text{bitline}}$ の電圧の差分を増幅することで，通常の DRAM よりも高速な activation および高い refresh 頻度削減への耐性を持つ．本文献では wordline 電圧が 2.2 V，bitline の基準電圧が 0.4 V の条件において，activation 開始から sense amplifier が電圧を目標値の 90%に増幅するまでの時間を 3 ns 短縮した．また wordline 電圧が 2.3 V，bitline の基準電圧が 0.5 V の条件において，データを正しく保持できる時間が通常の DRAM に対し 20%増加した．

文献 [57] では，通常の DRAM において同一のデータを持つ K 個の row（K = 2 or 4）を同時に activation することで tRCD，tRAS，tRFC を削減する．この様子を**図 6** に示す．Bitline には通常の K 倍の電荷が移動し電圧変化が高速化され，結果として上記パラメータの値を削減してもデータは正常に読み出せる．実装では K 個の物理的な row（K = 2 or 4）を 1 つの論理 row と見なし，論理 row を activation する際にはそれに属する K 個の物理 row を同時に activation する．電気的には 2 つの row の access transistor をそれぞれ独立に導通させればよく，実装は容易である．これを DRAM 全体に適用すると DRAM 容量が $\frac{1}{K}$ になるが，本文献では OS との協調により頻繁にアクセスするデータのみに手法を適用しこれを軽減する．本文献は 4 コアのシステムをシミュレーションする評価で平均 10.2%の性能改善と 23.2%の EDP（Energy Delay Product）改善を得た．

文献 [58] では文献 [57] の発展形として，activation 時に対象 row の持つデータを別の特別な row にコピーする．これにより次回の activation 時に 2 つの同じデータを持つ row を activation し tRCD を削減できる．Row のコピーには RowClone [59] に類似のメカニズムを用いる．これはコピー元 row の activation 後に precharge を行わず続けてコピー先 row を activation する手法である．Precharge を行わないことで row buffer の電圧がコピー元の値を保持し，続く restore でコピー先 row に書き戻されコピーが実現される．本文献では 4 コアのシステムをシミュレーションする評価で平均 20.0%の性能改善と 22.3%の DRAM 消費電力削減を得た．

6.3.2　不要なデータを保持しない

待機時間削減を可能にする 2 つ目の手法は，不要なデータの restore や refresh を完全にやめることである．文献 [60] では，上述の RowClone を使いある row の activation 時にその row をコピーする．同じ row への次のアクセスではコピー先の row を使い restore を省略する．コピー先 row のデータは破壊されるが，コピー元 row は保持されるため CPU にはデータ破壊は観測されない．

また文献 [49] は値がすべて 0 である row の refresh を完全にやめる．なるべく多くの row を 0 で埋めるため，本文献は次の 2 つの手法を用いる．第 1 に，OS がアプリケーションからメモリページを返還された際に 0 埋めするよう改変する．第 2 に，使用中のページにも 0 を増やすため，データに BDI（Base-Delta-Immediate）圧縮 [61] に類似の手法を適用する．BDI 圧縮は複数の類似データを 1 つのデータ（Base）とそれからの差分（Delta）で表現する手法であり，差分はほとんどが 0 である．さらに本文献では複数の Delta の同一ビット位置を同じ row に格納することで，値がすべて 0 である row を増やす．

7.　ビット反転の限定的受容

本章ではビット反転の限定的受容により設計余裕を活用する研究の詳細を述べる．これらの研究ではビット反転を特定のメモリ領域や確率に抑え込むことで，アプリケーションが意味のある計算を実行できるようにする．ビット

反転が起きる可能性のあるメモリは Approximate Memory と呼ばれる．

Approximate Memory 研究の基本アイデアは，アプリケーションのビット反転耐性を利用することである．ビット反転耐性とは，データの一部が変わっても最終的な計算結果がまったくあるいは少ししか変わらない性質を指す．たとえば何らかの評価値の比較に基づく探索アルゴリズム（例：A* アルゴリズム）では，評価値の値が変わっても評価値どうしの大小が変わらなければ結果は不変である．

Approximate Memory 研究は，(i) **OS のメモリ管理との協調**，(ii) **DNN（Deep Neural Network）の特性の利用**に分類できる．以下ではこの分類に沿って各研究を説明する．なお本章でも各研究の定量的な性能改善率は単純に相互比較できないことに注意が必要である．

7.1　OS のメモリ管理との協調

これらの研究群では，メモリ領域をビット反転率の異なる複数の領域に分割し OS のメモリ管理と協調する．アプリケーションプログラマがメモリ領域の確保時に通常領域とデータ一貫性が保証されない領域を使い分けることでビット反転に対処する．

文献 [3] は bank 内を refresh 間隔が短くビット反転が起きない領域（row の集合）と，refresh 間隔が長くビット反転が起きる領域に分割する．アプリケーションは各データのビット反転耐性に応じどちらかの領域を使用する．本文献では OS のページテーブルエントリに新たなビットを追加しこれをサポートする．このビットは当該ページが含むデータにビット耐性があるかないかを示し，真偽に応じ物理アドレスを割り当てることで 2 つの領域の使い分けを実現する．提案手法を用い，ヒープ中のビット反転耐性があるデータとグローバル変数をビット反転が起きる領域に置く条件で最大 25%の DRAM 消費電力削減を得た．

文献 [10] は refresh 間隔を伸長した際のビット反転混入度合いの低い順に OS が row を割り当てる．単一の refresh 間隔を使用しメモリコントローラの変更が不要なため，文献 [3] よりも実装が容易で類似の効果が得られる．割り当て順を決めるビット反転混入度合いのメトリクスには，(1)：ビット反転発生数，(2)：少なくとも 1 ビットが反転したワード数，(3)：ビット反転発生数にワード内の位置により重み付けしたもの（例：MSB の反転はより重大），(4)：メトリクス (2) において 1 から 0 への反転と 0 から 1 への反転を区別し重み付けしたもの，がある．アプリケーションに応じ，たとえばデータのほとんどが 0 ならばメトリクス (4) を用いるなどの使い分けができる．提案手法により文献 [3] を上回る refresh 消費電力の削減率を得た．

7.2　DNN の特性の利用

DNN は多くの応用で高い精度を出す一方，巨大なネットワークでは大容量のメモリを消費したりメモリに対し大きな重み行列を読み書きする必要性からメモリの性能や消費電力がボトルネックとなっている．そこで DNN を Approximate Memory 上で実行し計算時間や消費電力を削減する研究が多く行われている [11], [12], [13], [47]．これらの研究の基本アイディアは，DNN に特有の性質を利用することである．

文献 [47] は DNN がパターンを学習する特性を利用し tRCD を削減する．具体的には，学習中の入力，重み，中間層出力にビット反転を挿入することでビット反転も含めたデータに対して DNN を訓練する．ビット反転挿入では既知のビット反転発生原理（例：文献 [46]）を考慮する．また学習が進むにつれてビット反転発生率を上げ初期段階で学習が発散することを防ぐ．提案手法により，シミュレートされた GPU において 1%以内の精度低下で平均 2.7%の性能向上と 21%の DRAM 消費電力削減を得た．

文献 [13] は DNN の学習中に得られる loss 値を利用する．loss 値はたとえば学習データとの MSE（Mean Squared Error）などで定義され，学習中のネットワークの推論結果の良さを表す．本文献は loss が十分下がっている場合には refresh 間隔を伸長し，そうでない場合には短縮することで refresh 間隔を動的に調整する．loss が十分下がっているかの判断には広く使われるネットワークをビット反転のない DRAM 上で事前に学習し得た傾向を用いる．提案手法により 24.7%の DRAM 消費電力削減を得た．

文献 [11], [12] は DNN の扱うデータのビットごとの重要度の違いを利用し refresh 間隔を伸長する．IEEE 754 浮動小数点数形式では，最上位ビットが符号部，続く数ビットが指数部，残りが仮数部に割り当てられる．たとえば 32 ビット浮動小数点数では符号部が 1 ビット，指数部が 8 ビット，仮数部が 23 ビットである．このうち符号部と指数部はビット反転による数値的影響が仮数部より大きく，また仮数部の中でも上位ビットはビット反転の数値的影響が下位ビットより大きい．文献 [11] では下位ビットを上位ビットのパリティとして利用する．広く使われる AlexNet [62]，GoogleNet [63]，ResNet [64] を分析し，重みの下位 3 ビットはほぼ一定であることを発見した．本文献はこの 3 ビットをパリティとして利用し，浮動小数点数の上位 4 ビットが反転しても復元可能にする．提案手法を適用した DRAM を搭載した GPU をシミュレーションし，各 row の被 refresh 間隔が 512 ms のとき 23%の DRAM 消費電力削減を得た．文献 [12] では複数のデータの同一ビットを同一 row に格納し異なるビット位置に異なる refresh 間隔を適用可能にする．通常の DRAM では 1 回のアクセスで 64 バイトのデータを取り出すため連続する 64 バイトは同一の row に格納される．これを連続するデータを同一の column に格納するよう変更することでビット位置ごとに refresh 間隔を変更できる．**図 7** にこの様子を示す．提案

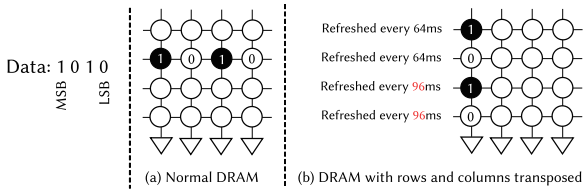

図 7　ビット列 1010 の DRAM 上の物理的配置．(a) 通常の DRAM：同一の row に格納 (b) 文献 [12] の提案：同一の column に格納され，ビットごとに異なる refresh 間隔を適用可能

Fig. 7　Physical placements of a bit-pattern“1010”.

手法により 26.3%の DRAM 消費電力削減を得た．

DNN 以外のアプリケーションでもビット反転耐性の違いを利用する試みは行われているが，これらの研究は発展途上の段階である．文献 [65] は SPEC CPU 2006，SPEC CPU 2017 の中から DRAM アクセスが多いアプリケーションを分析し，同じアプリケーション内にビット反転耐性の異なるデータが存在することを明らかにした．しかし一般のアプリケーションでは文献 [12] のように連続するデータを同一 column に格納することはオーバヘッドの観点から難しい．3 章で示したように DRAM からのデータ読み出しは row ごとに行われる．したがって連続するデータを同一 column に格納すると，単一の値の読み出しに多くの activation（32 ビット読むなら 32 回）が必要になる．複雑な DNN では重みのサイズが大きくもともと複数の row にまたがっているためこれは問題にならないが，少数の値をメモリから読み出し計算するアプリケーションでは大きな性能低下につながる．文献 [44] はデータビットの重要度に応じビット列を並べ替え，同一 row 内のビット反転率の異なる column を割り当てる（ビット反転の局所性の利用）．本文献は一般のアプリケーションが対象だが，ビット列の並べ替えパターンはあらかじめ決められている．並べ替えは DRAM 読み書き命令の前後に必ず必要であるため，任意のパターンによる並べ替えを無視できる性能低下で実現することは難しい．

8. 残された技術的課題

本章では DRAM の設計余裕活用技術の研究に残された技術的な課題を述べる．

8.1 計算結果の誤差保証の難しさ

残された課題の 1 つは，ビット反転の限定的受容においてアプリケーションの計算誤差の保証が難しいことである．ここで計算誤差の保証とはアプリケーションの計算結果の正しい値からのずれを与えられた範囲に収めることをいう．

計算誤差の保証に向け，ビット反転を意図的に挿入しアプリケーションのビット反転耐性を調査する研究がさかんである．文献 [66] はアプリケーション実行中に 1 ビットだけが反転するモデルにおいて，レジスタ上のビット反転位置とアプリケーションの計算誤差の関係を調査する．既存の知見 [67] を用い，同様の計算誤差を生むビット反転位置を特定し調査を高速化する．文献 [68] はビット反転発生原理を考慮した軽量なビット反転挿入手法を提案する．実機の DRAM 上で特定のメモリ領域に発生する activation，restore，precharge 回数を計測し，計測結果に応じてビット反転を挿入する．文献 [69] はビット反転発生をエミュレートするメモリシステム上でアプリケーション実行を可能にする．FPGA（Field Programmable Gate Array）上に DRAM エミュレータとソフトコアプロセッサを構成し，後者で実行するアプリケーションが前者にデータを保存する．なお本文献は 3 章とは異なる DRAM 方式をモデル化するが，モデルの変更により第 3 章の DRAM 方式にも適用可能と考えられる．

計算誤差の保証に向けた研究は，(1) サンプリングに依存し何らかの保証がないこと，(2) 計算結果の誤差の許容範囲から tRCD などのパラメータの逆算が難しいことが問題である．たとえば文献 [69] は実際の DRAM チップのビット反転特性をエミュレートしアプリケーションを複数回実行することで計算誤差を測定する．しかし本手法では実験設定とは異なる DRAM チップや入力データに対し同様の計算誤差になる保証がなく（問題点 (1)），また与えられた計算誤差を満たすパラメータを算出するには様々なパラメータを試すほかない（問題点 (2)）．これらの問題点に対し，著者らの知る限り初期段階の研究しか行われていない．文献 [70]，文献 [71] ではそれぞれ BNN（Binarized Neural Network），集合の類似度比較に限りビット反転混入と計算結果の誤差の関係を数理的に解析した．しかしこれらの解析は個別のアプリケーション強く依存し，一般のアプリケーションへの適用は難しい．

8.2 セキュリティレベルの低下

残された課題の 2 つ目は，セキュリティレベルの低下である．この課題はさらに (i) **電気的不安定さの増大**，(ii) **攻撃可能箇所の増加**に分類できる．

8.2.1 電気的不安定さの増大

refresh 間隔の伸長や restore，書き込み時の待ち時間削減を行う手法では，RowHammer [72] や RAMBleed [73] などの電磁気的相互作用による脆弱性に弱くなると予想される．

RowHammer は DRAM の保持データをそのデータが入った cell にアクセスせずに書き換える手法である．ある row の wordline に電圧をかけ access transistor を導通させると，電磁気的相互作用により隣の row の access transistor もわずかに導通する．この現象により，攻撃対象の cell が属す row の両隣の row を高速に何度も activation すると

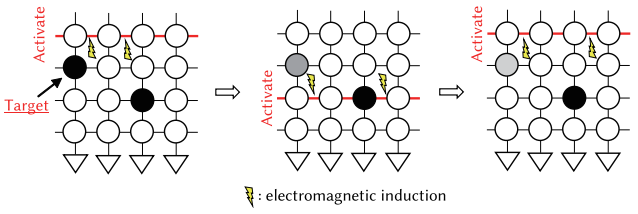

図 8　RowHammer 攻撃の動作原理：攻撃対象 row の両隣の row を高速に何度も activation すると電磁気的相互作用により電荷量が変化する

Fig. 8　Principle of RowHammer attack.

cell のキャパシタの電荷量が変化しデータが書き換わる．この様子を図 8 に示す．ただし cell が refresh されると電荷量は元に戻るため，攻撃者は refresh 間隔の間に十分な回数 activation する必要がある．また RAMBleed は RowHammer の応用でアクセス権限のないデータを読み出す手法である．

RowHammer とその応用に弱くなると予想される理由は以下である．第 1 に，refresh 間隔を伸長すると十分な回数 activation するために与えられる時間が長くなる．第 2 に，restore や書き込み時の待ち時間を削減すると cell のキャパシタの電荷量が通常の DRAM より減り，データ書き換えに必要な activation 回数が減る．これらのついて詳細な調査をした文献は著者らの知る限り存在せず，現象の定量的な理解から解決方法まで様々な研究が今後必要である．

8.2.2　攻撃可能箇所の増加

動作タイミングをソフトウェアと協調し制御する手法では，攻撃可能箇所（*attack surface*）の増加によるセキュリティレベルの低下が懸念される．7.1 節の研究群ではビット反転が混入する領域としない領域を OS のメモリ管理機構が割り当てる．したがってメモリ管理機構が信頼できない場合には意図せず重要なデータの破壊が起こる可能性がある．文献 [74] はこの可能性を指摘したが，可能性の言及にとどまり詳細な分析や対処法の提案は行っていない．

OS やそのメモリ管理機構が信頼できない可能性は十分にありうる．メモリ管理の重要な情報（たとえば読み書き権限やビット反転混入を許すかどうか）が格納されたページテーブルはそれ自体がメモリに格納されるデータである．したがってページテーブルを RowHammer 攻撃で書き換えればメモリ管理の動作を変更できる．またクラウドなどの遠隔環境では悪意のある管理者による OS の動作改変も指摘されている [75]．

9.　おわりに

本論文では DRAM のランダムアクセスレイテンシと消費電力の問題を論じ，それを解決するための設計余裕活用技術とその制御方の研究動向をまとめた．特にデメリットを限定的に受容する技術である Approximate Memory は様々な研究が進む途上であり今後の発展が期待される．

謝辞　本研究は，JST，ACT-I，JPMJPR18U1 および JST，さきがけ，JPMJPR22P1 の支援を受けたものである．

参考文献

[1] Karl Rupp: 42 Years of Microprocessor Trend Data (2018), available from ⟨https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/⟩.
[2] Venkatesan, R., Herr, S. and Rotenberg, E.: Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM, *International Symposium on High-Performance Computer Architecture* (*HPCA*), pp.155–165 (2006).
[3] Liu, S., Pattabiraman, K., Moscibroda, T. and Zorn, B.G.: Flikker: Saving DRAM Refresh-Power through Critical Data Partitioning, *International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS*), pp.213–224 (2011).
[4] Liu, J., Jaiyen, B., Veras, R. and Mutlu, O.: RAIDR: Retention-aware intelligent DRAM refresh, *International Symposium on Computer Architecture* (*ISCA*), pp.1–12 (2012).
[5] Baek, S., Cho, S. and Melhem, R.: Refresh Now and Then, *IEEE Trans. Computers*, Vol.63, No.12, pp.3114–3126 (2014).
[6] Wang, J., Dong, X. and Xie, Y.: ProactiveDRAM: A DRAM-initiated retention management scheme, *International Conference on Computer Design* (*ICCD*), pp.22–27 (2014).
[7] Cui, Z., McKee, S.A., Zha, Z., Bao, Y. and Chen, M.: DTail: A Flexible Approach to DRAM Refresh Management, *International Conference on Supercomputing* (*ICS*), pp.43–52 (2014).
[8] Bhati, I., Chishti, Z., Lu, S.-L. and Jacob, B.: Flexible Auto-Refresh: Enabling Scalable and Energy-Efficient DRAM Refresh Reductions, *International Symposium on Computer Architecture* (*ISCA*), pp.235–246 (2015).
[9] Lin, C.-H., Shen, D.-Y., Chen, Y.-J., Yang, C.-L. and Wang, C.-Y.M.: SECRET: A Selective Error Correction Framework for Refresh Energy Reduction in DRAMs, *ACM Trans. Architecture and Code Optimization*, Vol.12, No.2, pp.1–24 (2015).
[10] Raha, A., Sutar, S., Jayakumar, H. and Raghunathan, V.: Quality Configurable Approximate DRAM, *IEEE Trans. Computers*, Vol.66, No.7, pp.1172–1187 (2017).
[11] Nguyen, D.-T., Ho, N.-M. and Chang, I.-J.: St-DRC: Stretchable DRAM Refresh Controller with No Parity-Overhead Error Correction Scheme for Energy-Efficient DNNs, *Design Automation Conference* (*DAC*), pp.1–6 (2019).
[12] Nguyen, D.T., Hung, N.H., Kim, H. and Lee, H.-J.: An Approximate Memory Architecture for Energy Saving in Deep Learning Applications, *IEEE Trans. Circuits and Systems I: Regular Papers*, pp.1–14 (2020).
[13] Lin, X., Sun, L., Tu, F., Liu, L., Li, X., Wei, S. and Yin, S.: ADROIT: An Adaptive Dynamic Refresh Optimization Framework for DRAM Energy Saving In DNN Training, *Design Automation Conference* (*DAC*), pp.751–756 (2021).
[14] Chang, K.K., Kashyap, A., Hassan, H., Ghose, S., Hsieh, K., Lee, D., Li, T., Pekhimenko, G., Khan, S. and Mutlu, O.: Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization, *International Conference on Measurement and Modeling of Computer Science*

(*SIGMETRICS*), pp.323–336 (2016).
[15] Taassori, M., Shafiee, A. and Balasubramonian, R.: Understanding and alleviating intra-die and intra-DIMM parameter variation in the memory system, *International Conference on Computer Design* (*ICCD*), pp.217–224 (2016).
[16] JEDEC: Global Standards for the Microelectronics Industry, available from ⟨https://www.jedec.org/⟩.
[17] JEDEC: DDR2 SDRAM Standard, JESD79-2F (2009).
[18] JEDEC: DDR3 SDRAM Standard, JESD79-3F (2010).
[19] JEDEC: DDR4 SDRAM Standard, JESD79-4B (2017).
[20] JEDEC: DDR5 SDRAM Standard, JESD79-5 (2020).
[21] Standard Performance Evaluation Corporation: SPEC CPU 2006, available from ⟨https://www.spec.org/cpu2006/⟩.
[22] Li, S., Reddy, D. and Jacob, B.: A Performance & Power Comparison of Modern High-Speed DRAM Architectures, *International Symposium on Memory Systems* (*MEMSYS*), pp.341–353 (2018).
[23] Horowitz, M.: Computing's Energy Problem (and what we can do about it), *International Solid-State Circuits Conference Digest of Technical Papers* (*ISSCC*), pp.10–14 (2014).
[24] Mittal, S., Zhang, Z. and Vetter, J.S.: FlexiWay: A cache energy saving technique using fine-grained cache reconfiguration, *International Conference on Computer Design* (*ICCD*), pp.100–107 (2013).
[25] Arima, E., Noguchi, H., Nakada, T., Miwa, S., Takeda, S., Fujita, S. and Nakamura, H.: Immediate sleep: Reducing energy impact of peripheral circuits in STT-MRAM caches, *International Conference on Computer Design* (*ICCD*), pp.149–156 (2015).
[26] Malladi, K.T., Nothaft, F.A., Periyathambi, K., Lee, B.C., Kozyrakis, C. and Horowitz, M.: Towards energy-proportional datacenter memory with mobile DRAM, *International Symposium on Computer Architecture* (*ISCA*), pp.37–48 (2012).
[27] Barroso, L. and Hoelzle, U.: *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, Morgan & Claypool (2009).
[28] 石川　裕，佐藤三久，新庄直樹，清水俊幸：「京」の後の時代を支えるスパコン：2. 次期フラッグシップスーパーコンピュータの概要—スーパーコンピュータ「富岳」，情報処理，Vol.60, No.12, pp.1182–1188 (2019).
[29] Oak Ridge National Laboratory: Summit Oak Ridge National Laboratory's 200 petaflop supercomputer, available from ⟨https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/⟩.
[30] HPC@LLNL: Using LC's Sierra Systems, available from ⟨https://hpc.llnl.gov/documentation/tutorials/using-lc-s-sierra-systems⟩.
[31] Fu, H., Liao, J., Yang, J., Wang, L., Song, Z., Huang, X., Yang, C., Xue, W., Liu, F., Qiao, F., Zhao, W., Yin, X., Hou, C., Zhang, C., Ge, W., Zhang, J., Wang, Y., Zhou, C. and Yang, G.: The Sunway TaihuLight supercomputer: System and applications, *Science China Information Sciences*, Vol.59, pp.1–16 (2016).
[32] NERSC: Perlmutter, available from ⟨https://www.nersc.gov/systems/perlmutter/⟩.
[33] TOP500: List - November 2021 (2021), available from ⟨https://www.top500.org/lists/top500/list/2021/11/⟩.
[34] Oak Ridge National Laboratory: Summit System Overview, available from ⟨https://www.olcf.ornl.gov/wp-content/uploads/2018/05/Intro_Summit_System_Overview.pdf⟩.
[35] Jacob, B., Ng, S. and Wang, D.: *Memory Systems: Cache, DRAM, Disk*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2007).
[36] 伊藤清男：超 LSI メモリ，培風館 (1994).
[37] Dennard, R.H.: Field-effect transistor memory, U.S. Patent No.3387286 (1968).
[38] Micron: 8Gb: x8, x16 Automotive DDR4 SDRAM, MT40A1G8, MT40A512M16 (2016).
[39] Mathew, D.M., Zulian, E.F., Jung, M., Kraft, K., Weis, C., Jacob, B. and Wehn, N.: Using Run-Time Reverse-Engineering to Optimize DRAM Refresh, *International Symposium on Memory Systems* (*MEMSYS*), pp.115–124 (2017).
[40] Kim, C., Lee, H.-W. and Song, J.: *High-Bandwidth Memory Interface*, Springer (2014).
[41] Shin, W., Yang, J., Choi, J. and Kim, L.-S.: NUAT: A non-uniform access time memory controller, *International Symposium on High Performance Computer Architecture* (*HPCA*), pp.464–475 (2014).
[42] Lee, D., Kim, Y., Pekhimenko, G., Khan, S., Seshadri, V., Chang, K. and Mutlu, O.: Adaptive-latency DRAM: Optimizing DRAM timing for the common-case, *International Symposium on High Performance Computer Architecture* (*HPCA*), pp.489–501 (2015).
[43] Zhang, X., Zhang, Y., Childers, B.R. and Yang, J.: Restore truncation for performance improvement in future DRAM systems, *International Symposium on High Performance Computer Architecture* (*HPCA*), pp.543–554 (2016).
[44] Zhang, X., Zhang, Y., Childers, B.R. and Yang, J.: DrMP: Mixed Precision-Aware DRAM for High Performance Approximate and Precise Computing, *International Conference on Parallel Architectures and Compilation Techniques* (*PACT*), pp.53–63 (2017).
[45] Wang, Y., Tavakkol, A., Orosa, L., Ghose, S., Mansouri Ghiasi, N., Patel, M., Kim, J.S., Hassan, H., Sadrosadati, M. and Mutlu, O.: Reducing DRAM Latency via Charge-Level-Aware Look-Ahead Partial Restoration, *International Symposium on Microarchitecture* (*MICRO*), pp.298–311 (2018).
[46] Kim, J., Patel, M., Hassan, H. and Mutlu, O.: Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines, *International Conference on Computer Design* (*ICCD*), pp.282–291 (2018).
[47] Koppula, S., Orosa, L., Yağlıkçı, A.G., Azizi, R., Shahroodi, T., Kanellopoulos, K. and Mutlu, O.: EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM, *International Symposium on Microarchitecture* (*MICRO*), pp.166–181 (2019).
[48] Micron: Calculating Memory Power for DDR4 SDRAM, TN-40-07 (2017).
[49] Kim, S., Kwak, W., Kim, C., Baek, D. and Huh, J.: Charge-Aware DRAM Refresh Reduction with Value Transformation, *International Symposium on High Performance Computer Architecture* (*HPCA*), pp.663–676 (2020).
[50] Bhati, I., Chang, M.-T., Chishti, Z., Lu, S.-L. and Jacob, B.: DRAM Refresh Mechanisms, Penalties, and Trade-Offs, *IEEE Trans. Computers*, Vol.65, No.1, pp.108–121 (2016).
[51] Rixner, S., Dally, W.J., Kapasi, U.J., Mattson, P. and Owens, J.D.: Memory Access Scheduling, *International Symposium on Computer Architecture* (*ISCA*), pp.128–138 (2000).

[52] Kuhn, K.J., Giles, M.D., Becher, D., Kolar, P., Kornfeld, A., Kotlyar, R., Ma, S.T., Maheshwari, A. and Mudanai, S.: Process Technology Variation, *IEEE Trans. Electron Devices*, Vol.58, No.8, pp.2197–2208 (2011).
[53] Schechter, S., Loh, G.H., Strauss, K. and Burger, D.: Use ECP, Not ECC, for Hard Failures in Resistive Memories, *International Symposium on Computer Architecture* (*ISCA*), pp.141–152 (2010).
[54] Samsung Electronics Co., Ltd.: 8Gb C-die DDR4 SDRAM x16 (2017), available from ⟨https://www.samsung.com/semiconductor/global.semi/file/resource/2017/12/x16%20only_8G_C_DDR4_Samsung_Spec_Rev1.5_Apr.17.pdf⟩.
[55] Kim, Y., Seshadri, V., Lee, D., Liu, J. and Mutlu, O.: A case for exploiting subarray-level parallelism (SALP) in DRAM, *International Symposium on Computer Architecture* (*ISCA*), pp.368–379 (2012).
[56] Takemura, R., Itoh, K., Sekiguchi, T., Akiyama, S., Hanazawa, S., Kajigaya, K. and Kawahara, T.: Long-Retention-Time, High-Speed DRAM Array with 12-F^2 Twin Cell for Sub 1-V Operation, *IEICE Trans. Electronics*, Vol.E90.C, No.4, pp.758–764 (2007).
[57] Choi, J., Shin, W., Jang, J., Suh, J., Kwon, Y., Moon, Y. and Kim, L.-S.: Multiple Clone Row DRAM: A Low Latency and Area Optimized DRAM, *International Symposium on Computer Architecture* (*ISCA*), pp.223–234 (2015).
[58] Hassan, H., Patel, M., Kim, J.S., Yaglikci, A.G., Vijaykumar, N., Ghiasi, N.M., Ghose, S. and Mutlu, O.: CROW: A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability, *International Symposium on Computer Architecture* (*ISCA*), pp.129–142 (2019).
[59] Seshadri, V., Kim, Y., Fallin, C., Lee, D., Ausavarungnirun, R., Pekhimenko, G., Luo, Y., Mutlu, O., Gibbons, P.B., Kozuch, M.A. and Mowry, T.C.: RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization, *International Symposium on Microarchitecture* (*MICRO*), pp.185–197 (2013).
[60] Xin, X., Zhang, Y. and Yang, J.: Reducing DRAM Access Latency via Helper Rows, *Design Automation Conference* (*DAC*), pp.1–6 (2020).
[61] Pekhimenko, G., Seshadri, V., Mutlu, O., Kozuch, M.A., Gibbons, P.B. and Mowry, T.C.: Base-delta-immediate compression: Practical data compression for on-chip caches, *International Conference on Parallel Architectures and Compilation Techniques* (*PACT*), pp.377–388 (2012).
[62] Krizhevsky, A., Sutskever, I. and Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks, *Comm. ACM*, Vol.60, No.6, p.84–90 (2017).
[63] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A.: Going deeper with convolutions, *Conference on Computer Vision and Pattern Recognition* (*CVPR*), pp.1–9 (2015).
[64] He, K., Zhang, X., Ren, S. and Sun, J.: Deep Residual Learning for Image Recognition, *Conference on Computer Vision and Pattern Recognition* (*CVPR*), pp.770–778 (2016).
[65] Akiyama, S. and Shioya, R.: The Granularity Gap Problem: A Hurdle for Applying Approximate Memory to Complex Data Layout, *International Conference on Performance Engineering* (*ICPE*), pp.125–132 (2021).
[66] Venkatagiri, R., Mahmoud, A., Hari, S.K.S. and Adve, S.V.: Approxilyzer: Towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency, *International Symposium on Microarchitecture* (*MICRO*), pp.1–14 (2016).
[67] Hari, S.K.S., Adve, S.V., Naeimi, H. and Ramachandran, P.: Relyzer: Exploiting Application-Level Fault Equivalence to Analyze Application Resiliency to Transient Faults, *International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS*), pp.123–134 (2012).
[68] Akiyama, S.: A Lightweight Method to Evaluate Effect of Approximate Memory with Hardware Performance Monitors, *IEICE Trans. Inf. & Syst.*, Vol.E102.D, No.12, pp.2354–2365 (2019).
[69] Widmer, M., Bonetti, A. and Burg, A.: FPGA-Based Emulation of Embedded DRAMs for Statistical Error Resilience Evaluation of Approximate Computing Systems, *Design Automation Conference* (*DAC*), pp.1–6 (2019).
[70] Buschjager, S., Chen, J.-J., Chen, K.-H., Gunzel, M., Hakert, C., Morik, K., Novkin, R., Pfahler, L. and Yayla, M.: Margin-Maximization in Binarized Neural Networks for Optimizing Bit Error Tolerance, *Design, Automation & Test in Europe Conference* (*DATE*), pp.673–678 (2021).
[71] Reviriego, P., Liu, S., Ertl, O., Niknia, F. and Lombardi, F.: Computing the Similarity Estimate Using Approximate Memory, *IEEE Trans. Emerging Topics in Computing*, pp.1–12 (2021).
[72] Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J.H., Lee, D., Wilkerson, C., Lai, K. and Mutlu, O.: Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors, *International Symposium on Computer Architecuture* (*ISCA*), pp.361–372 (2014).
[73] Kwong, A., Genkin, D., Gruss, D. and Yarom, Y.: RAMBleed: Reading Bits in Memory Without Accessing Them, *Symposium on Security and Privacy* (*S&P*), pp.695–711 (2020).
[74] Yellu, P., Boskov, N., Kinsy, M.A. and Yu, Q.: Security Threats in Approximate Computing Systems, *Great Lakes Symposium on VLSI* (*GLSVLSI*), pp.387–392 (2019).
[75] Duncan, A.J., Creese, S. and Goldsmith, M.: Insider Attacks in Cloud Computing, *International Conference on Trust, Security and Privacy in Computing and Communications* (*TrustCom*), pp.857–862 (2012).

穐山 空道　（正会員）

2010 年京都大学工学部情報学科卒業．2015 年東京大学大学院情報理工学系研究科創造情報学専攻修了．博士（情報理工学）．日本電信電話株式会社，産業技術総合研究所，東京大学を経て，2022 年 4 月より立命館大学情報理工学部セキュリティ・ネットワークコース准教授．メモリシステム，性能分析，仮想化技術等の研究に従事．

山田 淳二　（正会員）

2010 年信州大学工学部情報工学科卒業．2017 年東京大学大学院情報理工学系研究科電子情報学専攻修了．博士（情報理工学）．2004〜2015 年まで，エルピーダメモリ株式会社（現マイクロンメモリジャパン合同会社）で DRAM の開発に従事．2017 年より東芝メモリ株式会社（現キオクシア株式会社）で NAND フラッシュメモリの開発に従事．

塩谷 亮太　（正会員）

1981 年生．2006 年東京大学工学部電子工学科卒業．2011 年東京大学大学院情報理工学系研究科電子情報学専攻博士課程修了．博士（情報理工学）．2011 年名古屋大学大学院工学研究科助教．2016 年同大学院同研究科准教授．2018 年東京大学大学院情報理工学系研究科創造情報学専攻准教授，現在に至る．コンピュータ・アーキテクチャや基盤ソフトウェア等の研究に従事．IPSJ/IEEE-CS Young Computer Researcher Award (2020)，情報処理学会山下記念研究賞 (2019) 等受賞．電子情報通信学会，IEEE，ACM 各会員．