Vol.2021-ARC-245 No.15
2021/7/21

Approximate Memory 制御手法の評価のためのベンチマーク開発

穐山 空道[1,a)]　松宮 遼[2]　吉藤 尚生[2]　梶 信也[2]

概要：Approximate Memory とは格納されたデータにエラーが入ることを許す代わりに低いレイテンシでアクセス可能な新しいメモリ技術である。本技術はデバイスレベルでは広く研究された様々な知見が蓄積されているものの、実際のアプリケーションに適用する最適な方法は未だ確立していない。Approximate Memory を実際のアプリケーションに適用する手法を評価するためには、ある手法で得られる高速化とアプリケーションの計算結果の誤差のトレードオフをベンチマークによって評価する必要がある。このためのベンチマークには、SPEC CPU Benchmarks などの広く使われるベンチマークと比較して次のような要求がある。(1) 計算結果の正解値のみならず、正解値からの誤差を評価するメトリックが与えられること。(2) 計算結果の誤差は一般に途中結果から外挿できないため、サイクルベースシミュレーション上でも現実的な時間で終了すること。本稿では (1)、(2) の要件を満たすベンチマーク開発の経験に基づき、誤差の評価に用いるメトリックや入力データの選定に関する議論を行う。

1. 序論

Approximate Memory は DRAM の電気的操作を意図的に仕様から逸脱させアクセスレイテンシを削減する技術である。CPU 性能が近年まで指数的に向上していたのに対し DRAM のアクセスレイテンシは 20 年以上ほぼ一定である [1,2]。このような性能向上の差異による DRAM アクセスレイテンシへの性能への影響増大を抑えるために Approximate Memory は有望な技術である。

Approximate Memory ではアクセスレイテンシが削減される代わりにビット化け確率が増大する。これは Approximate Memory が DRAM の電気的操作のタイミング制約を破り電気的に不安定な状態で動作するためである。具体的には、activation や precharge と呼ばれる操作に定められた待ち時間を短縮する手法 [2] や、DRAM のメモリセルを構成するキャパシタのリフレッシュ間隔を伸長する手法 [3,4] がある。

Approximate Memory の実用化には、その制御手法を評価する枠組みが必要である。Approximate Memory の制御とは、その上で実行するアプリケーションが意味のある結果を出力するよう適切な制御パラメータを選択することである。ある制御手法を評価するには実際にアプリケーションを実行し計算結果が実用的であるかを評価する必要がある。しかしコンピュータアーキテクチャ研究で広く用いられる SPEC CPU Benchmarks などのベンチマークでは計算結果の正解値のみが与えられ計算結果の誤差を評価する指標は与えられていない。単に正解値との誤差を計算することは可能であるが、その誤差がどの程度であれば実用的に許容できるのかはアプリケーションごとに異なり客観的な指標が必要である。

そこで本稿では、Approximate Memory 制御手法を評価するためのベンチマークセットを提案する。提案するベンチマークの特徴は (1) アプリケーションの計算結果の誤差を実用上の観点から評価可能であること、(2) 実行時間への DRAM アクセスの影響が小さくないこと、(3) データ領域の確保が明示的に行われる C、C++ で記述され Approximate Memory 上にデータを選択的に配置可能なことである。本稿では本ベンチマーク開発の経験から、特に特徴 (1) の実現について詳しく議論し Approximate Memory 制御手法の評価の一助となることを目指す。

2. Approximate Memory 制御手法の評価

2.1 Approximate Memory の制御

Approximate Memory 上でアプリケーションを実行する際のビット化け発生と最終的な計算結果のずれの関係はアプリケーションごとに異なる。**図 1** は様々なアプリケー

[1] 東京大学 情報理工学系研究科 創造情報学専攻
Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo
[2] 株式会社フィックスターズ
Fixstars Corporation
[a)] akiyama@ci.i.u-tokyo.ac.jp

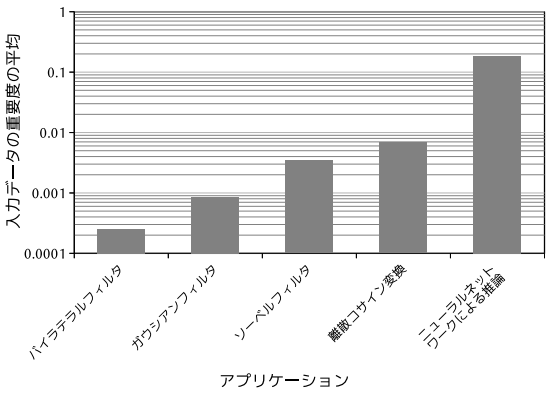

図 1 アプリケーションによる入力データ重要度の違い

ションの入力データの「重要度」を代表的な入力データに関し平均したものである。横軸は各アプリケーション、縦軸は重要度を対数スケールで表す。入力データの重要度とは、入力データが微少量ずれたときどの程度アプリケーションの最終結果がずれるかを表す指標である。重要度の正確な定義、算出方法および各アプリケーションの詳細は文献 [5] を参照のこと。図よりアプリケーションごとに入力データの重要度が大きく異なることが分かる。

Approximate Memory を実用化するためには、その適切な制御が必要である。Approximate Memory では制御できるパラメータは 2 つ存在する。1 つめはタイミング制約をどの程度積極的に破るかである。これは DRAM 内の電気的操作の待ち時間の設定値により制御できる。2 つめはどのデータを Approximate Memory に格納しどのデータを通常のメモリに格納するかである。これはメモリ空間上のデータ配置の変更により制御できる。

Approximate Memory の「適切な制御」とは、ユーザの許容する計算結果のずれをある程度保証するように制御パラメータを設定することとする。前述のように入力データの重要度はアプリケーションごとに大きく異なる。よって単一の制御パラメータでは一部のアプリケーションでは意味のある結果が得られても他のアプリケーションではずれが大きすぎ意味のない結果になる可能性がある。従って適切な制御にはアプリケーションに合わせ制御パラメータを決定する必要があり、これは自体が大きな研究課題である。

2.2 制御手法の評価における現状と課題

既存の Approximate Memory 制御手法研究の評価は、提案手法を特定のアプリケーションに適用し計算結果の誤差を定量化する。文献 [6] ではクラスタリングアプリケーションに対し、通常実行と Approximate Memory 上での実行でのクラスタ中心間の距離やテストデータに対するクラスタリング正解率を定量化する。文献 [7] では画像圧縮、画像分類、エッジ検出、動画圧縮、画像セグメンテーションなどのアプリケーションに対し計算結果の誤差を定量化するメトリクスを定義し、通常実行と Approximate Memory 上での実行でのメトリクスを比較する。また Approximate Memory 上の深層学習に特化した手法 [8,9] では様々な既存のネットワークに提案手法を適用し、通常の計算機上での実行と推論精度を比較する。

これらの Approximate Memory 制御手法の評価には、結果の解釈と公平性の観点で以下ような課題がある。

(1) 計算結果の誤差を定量化するに留まり、誤差の実用性が評価されていない。
(2) 同一のアプリケーションやデータセットで評価されておらず複数の制御手法の比較が困難である。

そこで本稿では、Approximate Memory 制御手法による計算結果の誤差の実用性を評価するベンチマークセットを提案しこれらの課題解決を目指す。

3. ベンチマークへの要求

本章では提案するベンチマークに求められる要求を述べる。広く使われる既存のベンチマークが要求を満たさない理由は第 4 章で議論する。

本稿で作成するベンチマークには次の 3 点の要件がある。

(1) ベンチマークを Approximate Memory 上で実行した際に発生する計算結果と正しい結果との誤差を、実用上の観点から評価する指標が与えられること。
(2) ベンチマークの実行がサイクルレベルシミュレーションにおいても現実的な時間で終了すること。
(3) C や C++ などのメモリ確保が明示的に行われるプログラミング言語で記述されること。

要件 (1) は Approximate Memory 制御手法の評価は、その手法を用いた際の結果の実用性に基づくべきだからである。基礎的な現象を解明する Approximate Memory のデバイスレベル研究と異なり、その制御手法研究ではアプリケーションの実行結果が実用的でなければ意味がない。しかし第 4 章で述べるように、既存のベンチマークでは正しい計算結果やそれとの誤差を与えるのみで誤差の実用性評価はできない。従って計算結果の実用性評価の枠組みを議論することが必須である。

要件 (2) の理由の一点目は、Approximate Memory 制御手法の評価はシミュレータで行える必要があることである。実機の DRAM のタイミングパラメータを変更する研究は数多く行われている [2,10,11] が、これらの研究での実機の使用は DRAM チップ（を搭載した DIMM）とメモリコントローラに限られる。実際にアプリケーションを実機の Approximate Memory 上で動かすには、改変を含むメモリコントローラを通常の CPU から使う方法、OS の使用するデータなどの保護すべきデータを通常のメモリ上に分離配置する方法、DRAM チップの温度のようなエラー率に影響する物理パラメータを制御する方法などが必要であり、非常に手間がかかる。これらを実際に実装し Approximate

Memory の実機の上で実際にアプリケーションを実行する研究はごく少なく、我々の知る限り文献 [12] のみである。また本文献では Approximate Memory 上で実行する仮想マシンがクラッシュすることを認めるため、どのデータを保護すべきかは細かく制御していない。

要件 (2) の理由の二点目は、アプリケーションの最終出力は途中結果からの推定が難しいことである。アプリケーションをサイクルレベルシミュレーション上で実行するには非常に長い時間がかかり、我々の調査では SPEC CPU 2006 ベンチマークで平均して実機の約 9,000 倍低速であった（文献 [13] の Fig. 3)。従ってサイクルレベルシミュレーションでは一般にアプリケーションの一部のシミュレーション結果から全体の結果を推定する。例えば消費電力ならばアプリケーションの主な処理を一定の命令数のみシミュレートし、得た消費電力を全体の命令数でスケールする。しかしアプリケーションの最終出力と途中結果の関係は複雑であり、途中結果から最終出力の予測は困難である。従って作成するベンチマークではアプリケーション全体をサイクルレベルシミュレーションで実行する必要がある。

要件 (3) は Approximate Memory 上のアプリケーション実行ではエラー混入を許すデータと許さないデータを区別し別々のメモリ領域に配置する必要があるためである。データにはポインタのように 1 ビットのエラーが重大な影響を及ぼすものと、数値計算に用いる値のように多少のエラーが混入しても最終結果が大きく変化しないものがある。前者は通常のメモリに配置し保護し、後者のみを Approximate Memory 上に配置してアクセスを高速化すべきである。これらの配置を明示的に制御するために C や C++ のようにヒープ上のデータ確保を明示的に行うプログラミング言語での記述が望ましい。一方 Python や Javascript のようなスクリプト言語では全てのデータはオブジェクトとして暗黙にメモリ領域確保が行われ、データの保護レベルに合わせた配置変更は難しい[*1]。

4. 既存のベンチマーク

本章では既存の広く使われるベンチマークについて述べ、それらがなぜ第 3 章の要求を満たさないのかを議論する。

4.1 SPEC CPU Benchmarks

SPEC CPU Benchmarks [14] はコンピュータアーキテクチャ研究の評価に広く使われる。アプリケーションは整数演算を主に扱うものと浮動小数点演算を主に使うものに分類され、前者は動画処理、離散最適化、イベントシミュレータ、コンパイラなどを、後者は波動シミュレーション、量子計算シミュレーション、画像処理などを含む。

SPEC CPU Benchmarks は出力結果の正解のみを与えるため本研究の目的には利用できない。本ベンチマークの目的は Approximate でない通常の計算機の性能測定であり、出力結果の正解は測定の公平性を担保するために利用される。従って出力結果は正解と完全一致するか浮動小数点演算の実装の差異による誤差内に収まることが要求され、結果の誤差を定量評価する枠組みは与えられない。

4.2 PARSEC

PARSEC [15] は SPEC CPU Benchmarks と同じくコンピュータアーキテクチャ研究の評価に広く使われるベンチマークである。例えば HPCA 2010 に発表された論文中の共有メモリ型マルチプロセッサの評価のうち 36% に PARSEC が使われた（文献 [16] の Figure 1.1)。PARSEC は動画エンコード、レイトレーシング、株価予想などのアプリケーションを含む。

PARSEC も SPEC CPU Benchmark と同様に出力結果の正解のみを与えるため本研究の目的には利用できない。本ベンチマークの主な目的はマルチコアプロセッサの性能評価である。従って出力結果の正解は測定の公平性を担保するために利用され、結果の誤差を定量評価する枠組みは与えられない。

4.3 AxBench

AxBench [17] は Approximate Computing によるアプリケーション実行結果を定量化するベンチマークであり、画像処理、株価予想、機械学習、数値計算などを含む。また関数をニューラルネットワークで置き換える Approximate Computing の一手法 [18] を簡単に適用可能にするためのアノテーションが付いている。

AxBench の特徴は Approximate Computing で得られた計算結果を正しい結果と比較するメトリクスを定義する点である。具体的には数値計算アプリケーションに対し平均相対誤差、画像処理アプリケーションに対し画像の差異、真理値を返すアプリケーションに対し正解率を計算する。しかし AxBench は計算したメトリクスが「実用上許容可能であるか」を判断する基準は与えておらず、適用した Approximate Computing が実用上有用かどうかを客観的に判断することはできない。

5. ベンチマークの構成

5.1 構成の方針と追加の要求

要求を満たすベンチマークの構成には既存のアプリケーションを選定する方法と新たなアプリケーションを開発する方法があるが、本稿では前者を採用する。これはベンチマークの最も重要な要求が計算誤差の実用性評価をすることだからである。理論的背景や使用実績が豊富な既存アプリケーションを選定し、計算誤差の実用性が議論可能にな

[*1] 言語処理系自体を改変すれば可能だが、これは今後の課題である。

表 1 選定したアプリケーションの概要

名前	ドメイン	アルゴリズム
HPCG	数値計算	CG 法
姫野ベンチ	数値計算	ヤコビ法
k-shortest-path	グラフ探索	Yen's algorithm の改良版 [19]
mfem-nv	数値計算	有限要素法

表 2 使用するマシンのハードウェアおよびソフトウェア仕様

OS	Ubuntu 18.04 LTS (Linux kernel 4.15)
CPU	Intel Core i7-4790 (4 cores, 8 threads)
LLC	8 MB (2 MB per core)
Memory	8 GB
Compiler	gcc 7.5
Options	-mno-avx -mno-avx2 -mno-bmi -mno-bmi2

ることを目指す。

要求を満たすベンチマークを構成するため、4 個のアプリケーションを選定した。選定にあたっては第 3 章に記した基本的な要求の他に、追加の要件として以下の 2 点を考慮した。

(1) 動画像処理やニューラルネットワーク以外のアプリケーションであること。
(2) アプリケーション性能に与える DRAM アクセスの影響が低すぎないこと。

追加の要求 (1) は、動画像処理やニューラルネットワークでは計算誤差の実用性評価が比較的簡単に行えるためである。動画像処理は SN 比による出力結果の定量評価が一般的で、計算誤差の実用性評価は SN 比の閾値を決めればよい。またニューラルネットワークで幅広く行われている推論タスクでは出力結果は推定精度で定量評価され、実用性評価は対象領域に合わせ推定精度の目標値を決めればよい。一方その他のドメインのアプリケーションでは計算誤差の実用性評価方法が確立されておらず、実用性の定義の議論から始めることが必要かつ有意義である。

追加の要求 (2) は、性能への DRAM アクセスの影響が低すぎるアプリケーションでは Approximate Memory を使う意味がないからである。Approximate Memory は DRAM のアクセスレイテンシを削減する手法であり、メモリにほとんどアクセスしないあるいはプリフェッチが有効に働くアプリケーションには恩恵が少ない。性能への DRAM アクセスの影響が低すぎないアプリケーションを選定し、ベンチマークの有用性を高める。

5.2 選定したアプリケーション

表 1 に選定したアプリケーションの概要を示す。それぞれの詳細は以下の通り。なお「リファレンス実装」とは各アプリケーションの出典が提供する実装のことで、具体的なコードへの参照は第 5.4 章に記す。

HPCG 3 次元格子内で自由度 1 の熱拡散方程式を前処理付き CG 法（Conjudate Gradient Method、共役勾配法）で解くアプリケーションである [20]。具体的には与えられた行列 A とベクトル b に対し $Ax = b$ を解く。A は粗行列であり主な計算は粗行列ベクトル積である。A, b はリファレンス実装と同様に与え、格子の 1 辺のサイズは 32 とする。

姫野ベンチ 非圧縮流体の運動を記述する偏微分方程式の圧力に関する項をヤコビ法で解くアプリケーションである[*2]。非圧縮流体とは圧力で密度が変化しないと仮定できる流体（例：水）である。ヤコビ法は A を行列、p, w をベクトルとして $Ap = w$ を反復解法で解くアルゴリズムであり、姫野ベンチでは偏微分方程式を離散化し $Ap = 0$ の形にしてヤコビ法を適用する。主な計算はステンシル計算である。入力データはリファレンス実装の S を用いる。

k-shortest-path グラフ上で与えられた始点から終点までの経路を、距離が短い順に k 本算出するアプリケーションである。算出には Yen's algorithm [21] の改良版 [19] を用いる。用いるグラフは第 6.3 章で説明する。

mfem-nv 非圧縮流体の運動を記述する偏微分方程式を、有限要素法による時間発展により解くアプリケーションである。具体的には mfem と呼ばれる有限要素法ライブラリの例題[*3]から Navier Miniapp を選定する。有限要素法には Tomboulides の射影法 [22] を用いる。偏微分方程式はリファレンス実装と同様に与える。

5.3 DRAM アクセスの実行時間への影響

アプリケーションの実行時間に DRAM アクセスが与える影響を Intel 製プロファイラである VTune Profiler で測定する。測定には VTune が提供するメトリクスのうち `DRAM Bound` という値を用いる。VTune のメトリクスは階層構造になっており、`DRAM Bound` は `Memory Bound` カテゴリに属する。`Memory Bound` はアプリケーションの全実行時間のうちメモリ関連の事象が消費する割合、`DRAM Bound` はさらにそのうちキャッシュヒット等を除き DRAM 関連の事象が消費する割合である。

アプリケーションの実行は**表 2** の環境で行う。Options 行は、選定したアプリケーションのリファレンス実装のコンパイルオプションに追加で指定するオプションである。これはコンピュータアーキテクチャ研究で広く使われるサイクルレベルシミュレータ gem5 がサポートしない拡張命令を除外するためである。gem5 が拡張命令をサポートしないことの懸念点については第 7.1 章で議論する。

表 3 は各アプリケーションの実行時間への DRAM アク

[*2] https://i.riken.jp/supercom/documents/himenobmt/
[*3] https://mfem.org/examples/

表 3 各アプリケーションの実行時間への DRAM アクセスの影響

アプリケーション	Memory Bound	DRAM Bound
HPCG	53.6 %	18.4 %
姫野ベンチ	57.5 %	32.6 %
k-shortest-path	25.1 %	12.6 %
mfem-nv	18.8 %	8.9 %

セスの影響である。入力データや反復回数は第 6 章で各アプリケーションについて示すものを使用した。なお DRAM Bound の値は Memory Bound 全体を 1 とした時の割合であり、実行時間全体に占める割合は Memory Bound と DRAM Bound をかけた値である。例えば HPCG では実行時間全体に占める DRAM アクセスの影響は $0.536 \times 0.184 \approx 0.098$ (9.8 %) である。

HPCG と姫野ベンチは元々メモリアクセスの性能を計測することが目的の 1 つであるため、実行時間への DRAM アクセスの影響が高い。mfem-nv は実行時間への DRAM アクセスの影響が $0.188 \times 0.089 \approx 0.016$ (1.6 %) と高くないが、他に検討し不採用となったアプリケーションより高い値だったため採用した。

5.4 各アプリケーションの実装

各アプリケーションの実装は以下に示す既存の実装を用い、必要に応じて改変を行う。

HPCG Dongarra らによるリファレンス実装[*4]を用いる。

姫野ベンチ 理化学研究所によるリファレンス実装[*5]のうち、“C, dynamic allocate version” を用いる。Fortran 版ではなく C 版を使う理由は、Approximate Memory 上で実行するためにデータ領域が明示的に確保されるコードが適しているからである。

k-shortest-path Yen’s algorthim の改良版にはその OSS 実装[*6]を用いる。また第 6.3 章で参照する svp-c には文献 [23] の著者らによる実装[*7]を用いる。

mfem-nv mfem ライブラリに含まれる例題 Navier Miniapp のリファレンス実装[*8]を用いる。

姫野ベンチではリファレンス実装からステンシルの結合数を削減した。リファレンス実装では 3 次元 19 点ステンシル計算を行うが、本稿の実装では自分自身と各軸方向の隣のみを使い 3 次元 7 点ステンシル計算を行う。これは行列 A の作り方から残り 12 点は必ず 0 だからである [24]。不要なアクセスがあると実行時間に対する DRAM アクセスの影響が増大し Approximate Memory 制御手法による改善を不当によく評価する可能性がある。

mfem-nv ではリファレンス実装の MPI 関連関数を何もしないスタブで置き換えた。リファレンス実装は並列実行を想定し MPI を使用するが、本稿では簡単のためシングルプロセス実行を仮定する。シングルプロセスで MPI 関連関数を実行しても機能的には問題なく動作するはずである。しかしシミュレーション時間削減のため不要な操作は削除すべきであり、また gem5 上での MPI 関連関数の動作確認は特に通信 API 部分に手間がかかる、あるいは追加実装が必要になる可能性がある。シングルプロセスでの MPI 関連関数は実質的に何もせず、スタブで置き換えてもアプリケーションの本質には関わらない。

5.5 ベンチマークの実行時間

表 4 各アプリケーションの実機での実行時間

アプリケーション	実行時間（秒）
HPCG	10.59
姫野ベンチ	10.93
k-shortest-path	79.51
mfem-nv	2.33

表 4 に、各アプリケーションを表 2 に記した環境で実行した際の実行時間を示す。なおこの環境は通常のマシンの実機でありサイクルレベルシミュレータではない。

HPCG と姫野ベンチは反復法によるアルゴリズムであり実行時間に任意の値を設定できる。本稿では実行時間を 10 秒に設定した。ただし 10 秒で打ち切るのではなく実行時間が 10 秒に近くなるよう反復数を設定するため誤差が生じている。値として 10 秒を選んだ理由は、実機で 10 秒程度ならばサイクルレベルシミュレータ上でも現実的な時間で終了すると考えられるからである。前述のように gem5 上での実行は平均で実機の約 9,000 倍低速だが、10 秒の 9,000 倍は約 1 日でありサイクルレベルシミュレーションとしては現実的な時間である。

k-shortest-path は実機で約 80 秒かかっており、サイクルレベルシミュレータでは 1 週間程度かかると予想される。ただし本アプリケーションでは経路を求める始点・終点の数を変更することで実行時間を調整できる。第 6.3 章に示すように本稿では始点・終点のペア数が 100 であり、これを例えば 10 にすれば 1 日以内で実行が完了する。

mfem-nv の実行時間は十分短くサイクルレベルシミュレータ上の実行でも問題ないと考えられる。

6. 計算誤差の実用性評価

各アプリケーションの計算誤差を定量化するメトリクスと、計算誤差の実用性を判断する基準を与える。全てのアプリケーションで計算誤差は通常のマシン上での実行結果と Approximate Memory を搭載したマシン上での実行結

*4 http://www.hpcg-benchmark.org/
*5 https://i.riken.jp/supercom/documents/himenobmt/download/win-mac/
*6 https://github.com/yan-qi/k-shortest-paths-cpp-version
*7 https://github.com/tchond/kspwlo
*8 https://github.com/mfem/mfem/tree/master/miniapps/navier

果を比較し定量化する。なお k-shortest-path アプリケーションのみ 2 つのマシンで異なるアルゴリズムを実行するため注意が必要である。

本章の基準は計算誤差の実用性を評価するものであり、Approximate Memory 上での実行が通常の計算機上での実行より良いかを判断するのはこれに加え実行時間の比較が必要である。すなわち、ある計算を Approximate Memory 上で実行したほうが「良い」とは、

(1) Approximate Memory 上での計算結果が本章の実用性基準を満たす
(2) Approximate Memory 上での計算結果が通常の計算機上での実行より実行時間が短い

の両方が成り立つことである。以下では (2) は共通に仮定されるとし記述を省略する。

また以降ではベクトル v の絶対値 $|v|$ は v の L2 ノルム（「全要素の二乗和」の正の平方根）を表す。

6.1 HPCG

計算誤差には k 回目のイテレーションでの解 x_k から計算した Ax_k と目標値 b との差を用いる。ただし粗行列ベクトル積 Ax_k の計算に時間がかかるため、$Ax_k - b$ の値として計算途中の変数から得られる近似値を用いることがある。以後これを「残差ベクトル」と呼び、k 回目のイテレーションでの残差ベクトルを r_k と書く。また区別のため通常のマシン上と Approximate Memory を搭載したマシン上で計算した x をそれぞれ x^r, x^t と書く。

まず通常のマシン上で 50 回イテレーションを実行し、$t_1 = |r_{50}|$ と $t_2 = |Ax^r_{50} - b|$ を得る。次に Approximate Memory を搭載したマシン上で以下を計算する。

(1) 残差ベクトルの L2 ノルムが t_1 以下になるまたはイテレーション数が 500 になるまで繰り返した時の、最終イテレーションでの x^t の値 x^t_a（$a \leq 500$）
(2) (1) をシミュレーション空間の時間で 10 秒間実行した時の、それぞれの実行で得られる最終的な x^t の値

これらを用い、以下の条件が共に満たされれば Approximate Memory を搭載したマシン上での計算結果は実用的であると判断する。

解の収束性 $|Ax^t_a - b|$ が t_2 以下である。すなわち Approximate Memory 上で通常マシン上の 10 倍のイテレーション数（前者は 500 回、後者は 50 回）繰り返しても解が改善しなければ実用的ではないとする。

解の再現性 (2) の各実行で得られる x^t から計算される $|Ax^t - b|$ の分散が 10^{-6} 未満である。すなわち Approximate Memory 上での複数の実行で得られる解のぶれが大きすぎれば実用的ではないとする。

これらの基準はスーパーコンピュータの性能を測るベンチマークである元々の HPCG が計算の正しさを検査する基準と同じである。HPCG のような反復法では通常の計算機上でも反復回数を変えれば実行時間と解の良さのトレードオフを調整できる。つまり Approximate Memory の「高速化する代わりに計算誤差が大きくなる」という特性をアルゴリズム自身がイテレーション数というパラメータを通じ実現している。従って Approximate Memory 上の実行の実用性を判断するには解の良さを直接比較することが必要である。

6.2 姫野ベンチ

計算誤差にはヤコビ法の各イテレーションで得られる計算結果と目標値との「誤差」を用いる。k 回目のイテレーションでの誤差 G_k は以下のように定義する[*9]。まず k 回目のイテレーション終了時の p を p_k とし、$r_k(i) = -\frac{1}{A(i,i)}A(i)p_k$ を計算する。$A(i,i)$ は A の (i,i) 要素を、$A(i)$ は A の i 行目のベクトルを表す。次に r_k を第 i 要素が $r_k(i)$ であるベクトルとする。これは Ap_k を A の対角成分で正規化したものである。最後に $G_k = |r_k|$ を得る。また区別のため Approximate Memory を搭載したマシンと通常のマシンで得られる G_k をそれぞれ GT_k, GR_k と書く。

これらの値を用い、以下の条件を満たせば GT_k は実用的と判断する。なお以下の NT, NR は通常のマシンと Approximate Memory を搭載したマシンでそれぞれ姫野ベンチを 10 秒間実行し反復できた回数である。時間はそれぞれの計算機上で計測する。

解の収束性 $2 \leq 2^l \leq NT$ を満たす任意の l に対し、$GT_{2^l} \leq GT_{2^{(l-1)}}$ が成立する。すなわち、Approximate Memory 上での実行で反復回数を増やしても目標値との誤差が減少しなければ実用的でないとする。G が理論的には反復ごとに減少することを根拠とする。ただし実際には浮動小数点計算による（Approximate Memory とは独立の）計算誤差で G が増加することがあるため、2 の冪ごとにサンプルする。通常の計算機上での誤差 GR_{2^l} は l に関し単調減少することを経験的に確かめた。

性能の実用性 $GT_{NT} \leq GR_{NR} + 0.1(GR_{NR-1} - GR_{NR})$ が成立する。すなわち、同一の実行時間で GT が GR と類似の値に達しなければ Approximate Memory による高速化は実用的ではない。

6.3 k-shortest-path

計算誤差の評価は Approximate Memory を搭載したマシンでの厳密アルゴリズムの結果を通常のマシンでの近似アルゴリズムの結果と比較し行う。**厳密アルゴリズム**は出力する k 本の経路が厳密に条件（可能な全経路のうちで距離の短い順に k 本であること）を満たす。一方で**近似アルゴリズム**は出力する経路が厳密に上記条件を満たさない可

[*9] これはリファレンス実装で GOSA と呼ばれているものである。

能性がある。Approximate Memory を搭載したマシンでの厳密アルゴリズムの解が通常のマシン上での近似アルゴリズムの解よりも悪ければ厳密アルゴリズムを使う意味がないという観点から実用性を評価する。

比較する近似アルゴリズムには svp-c アルゴリズム [23] を用いる。これは (1) 2020 年に発表された新しいものであり、(2) 必ず k 本の解を出力し厳密アルゴリズムと比較が容易であるため選定した。

入力データはユタ大学による Real Datasets for Spatial Databases: Road Networks and Points of Interest[*10]からカリフォルニア州道路データ [25] を使用する。選定理由は実世界の道路情報に基づくこと、Yen's algorithm の改良版を提案する文献 [19] で使われるグラフと比較しキャッシュミス率が高く Approximate Memory の恩恵が大きいと予想されることである。k は 3 とし、始点と終点は 100 ペアを乱数で事前生成する。

Approximate Memory を搭載したマシン上の厳密アルゴリズムと通常のマシン上の近似アルゴリズムで 100 ペアの始点・終点について上位 3 経路を出力する。得た経路をそれぞれ $PT_{i,j}$, $PR_{i,j}$ と書く。i は始点・終点のペアに関するインデックス（$i = \{1, 2, ..., 100\}$）で、j はある始点・終点ペアに対する解に関するインデックス（$j = \{1, 2, 3\}$）である。例えば $PT_{50,2}$ は、Approximate Memory を搭載したマシン上で計算した 50 番目の始点・終点ペアに対する解の 2 番目のものである。

これらの値を用い、以下の条件が共に満たされれば Approximate Memory を搭載したマシン上での実行は実用的であるとする。

解の妥当性　全ての (i, j) に関し $PT_{i,j}$ を通り始点から終点に到達できる。すなわち意味のない経路を出力すれば実用的でないとする。

解の良さ　$PT_{i,j}$ の平均経路長が $PR_{i,j}$ の平均経路長以下である、すなわち $\sum_{i,j}(|PT_{i,j}|) \leq \sum_{i,j}(|PR_{i,j}|)$ を満たす。ただし $|P|$ は経路 P の長さである。Approximate Memory 上での厳密アルゴリズムによる解が通常のマシン上での近似アルゴリズムによる解よりも悪ければ実用的でないとする。

6.4 mfem-nv

計算誤差には時間発展の各ステップでの流速と圧力を用いる。まず Approximate Memory を搭載したマシンの上で mfem-nv を実行し、各ステップでの観測点における流速と圧力を記録する。これを近似解と呼ぶ。観測点はシミュレーション空間を 2×4 に 8 分割した各メッシュの中心点である。次に mfem-nv を通常のマシン上で実行し、時間発展の各ステップで観測点における流速と圧力を記録する。これを厳密解と呼ぶ。最後に厳密解と近似解を比較し、以下の条件が共に満たされれば Approximate Memory を搭載したマシン上での実行は実用的であるとする。

解の妥当性　各時間ステップでの近似解と厳密解の差の L2 ノルムが、流速では 10^{-6} 以下、圧力では 10^{-5} 以下である。これらの閾値はリファレンス実装において解の妥当性を検証するために使われている値と同じである。なおリファレンス実装を通常のマシン上で実行した場合の流速の L2 ノルムは 10^{-7} 程度であり、同じ解法の別実装による実験結果 [26] でも同様の値が報告されている。従って誤差の L2 ノルムが 10^{-6} を越えることは絶対値が厳密解の 10 倍以上違うことを意味し、実用的でないと判断できる。

動作の妥当性　メッシュの頂点において、CFL 条件 [27] が満たされる。これを満たさない場合シミュレーションは発散することが知られている。

7. 議論と今後の課題

7.1 gem5 の拡張命令対応

コンピュータアーキテクチャ研究で広く用いられるサイクルレベルシミュレータである gem5 は一部の拡張命令をサポートしない。具体的には、バージョン 20.0.0.0（2020 年 5 月時点での最新版）で少なくともベクトル演算命令である AVX、AVX2 とビット演算高速化命令である BMI、BMI2 をサポートしない。また本稿執筆時点の最新版は 21.0.0.0 であるが、そのリリースノートにこれらをサポートしたという記述はない。

拡張命令がないことはシミュレーションによる DRAM アクセスの影響見積もりの正しさに影響する。これは拡張命令により計算速度が向上し実行時間に占める DRAM アクセスの影響が増大するためである。例えば AVX は複数の浮動小数点演算をまとめて実行するため実行時間のうち浮動小数点演算の割合が減り、DRAM アクセスの影響が相対的に増える。従って Approximate Memory 制御手法の評価での gem5 の利用は適切とは言えない可能性がある。しかし gem5 は多くのコンピュータアーキテクチャ研究で利用されており、一概に対象外にすることは難しい。

7.2 本稿の経験に基づく統一的な評価基準の導出

本稿で議論した計算結果の誤差の評価基準は、アプリケーションごとに様々である。これは各アプリケーションに応じ実用性の観点を議論し導出した結果である。例えば HPCG における解の再現性の評価は、スーパーコンピュータの性能を計測する元々の HPCG が要求する基準と同一である。また厳密アルゴリズムと近似アルゴリズムによる結果を比較し Approximate Memory を使用する価値を判断する基準は k-shortest-path のみに存在する。

今後の課題の 1 つは各アプリケーションごとに定めた

[*10] https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm

評価基準からより統一的で有用な基準を導出することである。例えば解の再現性は HPCG 以外のベンチマークでも有用だと考えられる。ビット化けの位置が 1 ビット異なるだけでアプリケーションの計算結果は大きく違いうるため、1 度の実行でたまたま計算誤差が小さかったからといって必ずしも実用性があるとは判断できない。また厳密アルゴリズムと近似アルゴリズムを比較する方法も別のアプリケーションにも適用できる可能性がある。通常のマシンで精度が悪いが速いアルゴリズムを実行することと、Approximate Memory 上で精度が良いが遅いアルゴリズムを高速化することは、実行速度と精度を交換するという点で類似する。従ってこれらをある特定のメトリクスで比較し良い方を選ぶことには意味がある。

7.3 アプリケーションの多様性

本稿で選定したアプリケーションは類似のものが多い。HPCG と姫野ベンチマークはともに線型方程式の解を求めるアプリケーションという観点で類似し、姫野ベンチマークと mfem-nv はともに非圧縮流体のシミュレーションという観点で類似する。

アプリケーションに類似のものが多い理由は、これらのアプリケーションは計算誤差の実用性を比較的議論しやすいからである。大規模線型方程式の解を求めるアプリケーションは大規模計算機の性能評価に広く用いられ、計算の正しさの検証やベンチマークの公平性担保のために許容できる計算誤差が定義されている。またマクロな物理現象のシミュレーションでは現実の現象と大きく乖離しないという条件から計算結果の実用性を議論できる。本稿は画像処理以外のアプリケーションの計算結果の実用性を議論する点ではある程度目標を達成しているが、今後はより多くの種類のアプリケーションに関して同様の議論が必要である。

8. 結論

Approximate Memory 制御手法の評価のため、計算結果の誤差の実用性を評価できるベンチマークが必要である。本稿ではそのようなベンチマークに求められる要求を定義し、それを満たすベンチマークを構成するためのアプリケーション選定とその計算結果の誤差を評価する基準の議論を行った。今後の課題は得られた知見に基づきより統一的な評価基準を提案すること、より多様なアプリケーションを選定すること、並列処理などより現実的な設定での動作に対応することである。

謝辞 本研究は、JST、ACT-I、JPMJPR18U1 の支援を受けたものである。

参考文献

[1] Hennessy, J. L. and Patterson, D. A.: *Computer Architecture, Fourth Edition: A Quantitative Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2006).

[2] 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).

[3] Das, A., Hassan, H. and Mutlu, O.: VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency, *Design Automation Conference (DAC)*, pp. 1–6 (2018).

[4] 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).

[5] 穐山空道，塩谷亮太：Approximate Memory におけるエラー混入対象データの重要度の事前推定に関する検討，情報処理学会研究報告システム・アーキテクチャ，No. 2021-ARC-245 (2021).

[6] Tian, Y., Zhang, Q., Wang, T., Yuan, F. and Xu, Q.: ApproxMA: Approximate Memory Access for Dynamic Precision Scaling, *Great Lakes Symposium on VLSI (GLSVLSI)*, p. 337–342 (2015).

[7] Raha, A., Sutar, S., Jayakumar, H. and Raghunathan, V.: Quality Configurable Approximate DRAM, *IEEE Transactions on Computers*, Vol. 66, No. 7, pp. 1172–1187 (2017).

[8] 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)*, p. 166–181 (2019).

[9] Nguyen, D. T., Hung, N. H., Kim, H. and Lee, H.-J.: An Approximate Memory Architecture for Energy Saving in Deep Learning Applications, *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14 (2020).

[10] Kim, J., Patel, M., Hassan, H. and Mutlu, O.: Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines, *IEEE International Conference on Computer Design (ICCD)*, pp. 282–291 (2018).

[11] Lee, D., Khan, S., Subramanian, L., Ghose, S., Ausavarungnirun, R., Pekhimenko, G., Seshadri, V. and Mutlu, O.: Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms, *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, pp. 1 – 36 (2017).

[12] Tovletoglou, K., Mukhanov, L., Nikolopoulos, D. S. and Karakonstantis, G.: HaRMony: Heterogeneous-Reliability Memory and QoS-Aware Energy Management on Virtualized Servers, *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, p. 575–590 (2020).

[13] Akiyama, S.: A Lightweight Method to Evaluate Effect of Approximate Memory with Hardware Performance Monitors, *IEICE Transactions on Information and Sys-*

tems, Vol. E102-D, No. 12, pp. 2354–2365 (2019).
[14] Standard Performance Evaluation Corporation: SPEC CPU 2017 Overview, `https://www.spec.org/cpu2017/Docs/overview.html` (2020).
[15] Princeton University: The PARSEC Bennchmark Suite, `https://parsec.cs.princeton.edu/` (2010).
[16] Bienia, C.: Benchmarking Modern Multiprocessors, PhD Thesis, Princeton University (2011).
[17] Yazdanbakhsh, A., Mahajan, D., Esmaeilzadeh, H. and Lotfi-Kamran, P.: AxBench: A Multiplatform Benchmark Suite for Approximate Computing, *IEEE Design Test*, Vol. 34, No. 2, pp. 60–68 (2017).
[18] Esmaeilzadeh, H., Sampson, A., Ceze, L. and Burger, D.: Neural Acceleration for General-Purpose Approximate Programs, *IEEE Micro*, Vol. 33, No. 3, pp. 16–27 (2013).
[19] Martins, E. Q. and Pascoal, M. M.: A new implementation of Yen's ranking loopless paths algorithm, *Quarterly Journal of Operations Research (4OR)*, Vol. 1, pp. 121 – 133 (2003).
[20] Dongarra, J., Heroux, M. A. and Luszczek, P.: HPCG Benchmark: a New Metric for Ranking High Performance Computing Systems, Technical Report UT-EECS-15-736, The University of Tennessee, Knoxville (2015).
[21] Yen, J. Y.: An algorithm for finding shortest routes from all source nodes to a given destination in general networks, *Quart. Appl. Math.*, Vol. 27, pp. 526 – 530 (1970).
[22] Tomboulides, A. G., Lee, J. C. Y. and Orszag, S. A.: Numerical Simulation of Low Mach Number Reactive Flows, *Journal of Scientific Computing*, Vol. 12, pp. 139 – 167 (1997).
[23] Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U. and Blumenthal, D. B.: Finding k-shortest paths with limited overlap, *The VLDB Journal*, Vol. 29, No. 5, pp. 1023–1047 (2020).
[24] 渡部善隆，南里豪志，藤野清次：Himeno BMT によるハイパフォーマンスコンピュータの性能評価，情報処理学会研究報告ハイパフォーマンスコンピューティング，No. 2003-HPC-095, pp. 137 – 142 (2003).
[25] Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G. and Teng, S.: On Trip Planning Queries in Spatial Databases, *International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, pp. 273–290 (2005).
[26] Franco, M., Camier, J.-S., Andrej, J. and Pazner, W.: High-order matrix-free incompressible flow solvers with GPU acceleration and low-order refined preconditioners, *arXiv:1910.03032*, pp. 1 – 20 (2019).
[27] Courant, R., Friedrichs, K. and Lewy, H.: On the Partial Difference Equations of Mathematical Physics, *IBM Journal of Research and Development*, Vol. 11, No. 2, pp. 215–234 (1967).