Vol.2016-ARC-221 No.10
2016/8/8

単調性を利用した計算再利用の近似計算への応用

藤井 政圭[1]　佐藤 裕貴[1]　津邑 公暁[1]　中島 康彦[2]

概要：計算を近似化することで高速化・省電力化を図る Approximate Computing というパラダイムが注目を集め，盛んに研究されている．Approximate Computing は信号処理や機械学習など，出力にある程度の誤差が許容されるアプリケーションに対して大きな効果が期待される．しかし，近似的な計算が出力に与える影響および出力における許容誤差範囲がともにアプリケーション毎に異なることや，様々なアプリケーションに対して統一的に Approximate Computing を適用する方法が確立されていないことから，Approximate Computing は生産性や可用性の低さに問題を抱えている．こうした背景から，我々は可用性の高い Approximate Computing 計算基盤の実現を目指している．この計算基盤では，Approximate Computing と親和性の高い計算再利用の考え方を用いることで，様々なアプリケーションに対して統一的に Approximate Computing を適用する，本稿では，計算を近似化する際に関数の単調性を利用することで，関数の出力に求められる計算精度を保ちつつ，最適な計算近似度に調整する手法を検討する．MediaBench の mesa に含まれる log 関数に対してシミュレーションによる評価を行った結果，通常の計算再利用を用いた場合の再利用率が 8.7%であったのに対し，出力許容誤差を 0.01 とした場合で，再利用率は 90.4%まで向上，また計算再利用に必要な記憶容量を 7.3%まで削減し，提案手法の有効性を確認した．

1. はじめに

近年，メディアデータやセンサデータ，ソーシャルメディアを介して発信されるデータなどコンピュータで扱うべきデータが爆発的に増大している．このような大規模データを処理する必要があるアプリケーションでは，必ずしも計算結果に高信頼性と高精度を必要としない場合が多く，必要最低限以上の精度で，より早く結果を出力する，あるいはより多くの計算を行うことが重要となる．このような背景から，計算の近似化によって実行時間や消費電力を削減する，**Approximate Computing（Approx. Comp.）**というパラダイムが注目を集め，プログラミング言語 [1] からトランジスタレベル [2] に至るまで様々な分野で盛んに研究されている．Approx. Comp. とは，出力における誤差が許容できる範囲であれば，完全に正確な計算ではなく近似的な計算を行うことを許容する，という考え方であり，信号処理や機械学習など様々な分野での活用が期待されている．

しかし，様々なアプリケーションに対して統一的に Approx. Comp. を適用する方法については検討すらされていないのが現状である．そのため Approx. Comp. を適用するためには，プログラマはアプリケーション毎に試行錯誤しながら，近似的な計算が出力に与える影響の検証，および出力における許容誤差範囲に合わせた計算近似度の調整を行う必要があり，生産性や可用性の低さに問題を抱えている．

こうした背景から，様々なアプリケーションに対する統一的な Approx. Comp. の適用方法として，我々は Approx. Comp. と親和性の高い**計算再利用（Computation Reuse）**をベースとした，可用性の高い計算基盤の実現を目指している [3]．計算再利用は，関数の過去の入力と現在の入力が一致した場合に，過去の実行結果を再利用することで関数の実行自体を省略し高速化する手法である．この計算基盤では，入力の一部の不一致を許容する計算再利用を適用するという方法で，様々な処理に Approx. Comp. を統一的に適用可能とする．また，不一致を許容するビット幅を変更することで容易に計算近似度を調整できる．本稿では，計算を近似化する際に関数の単調性を利用することで，関数の出力に求められる計算精度を保ちつつ，最適な計算近似度に調整する手法について検討する．

2. Approximate Computing

本章では，Approx. Comp. の概要と Approx. Comp. を統一的に適用可能な計算基盤の実現に向けた展望について述べる．

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

2.1 Approximate Computing の概要と応用例

前章で述べたように，Approx. Comp. とは出力における誤差が許容できる範囲であれば，完全に正確な計算ではなく近似的な計算を行うことを許容する，という考え方である．Approx. Comp. を用いる場合，プログラマはまず対象とするアプリケーションの出力に許容できる誤差の範囲を決定する．そして各処理の出力に与える影響と許容誤差範囲とを考慮して，近似計算を適用する処理の選択および計算近似度の調整を行う．近似計算の例として，出力に与える影響が小さい処理における有効数字の削減や，ループ処理における途中結果の利用などが挙げられる．このような近似計算は，全ての計算を高精度に行う場合に比べ，処理量や必要とする計算資源が少なく済むため，高速化・省電力化を実現することができる．

Approx. Comp. が適用可能なアプリケーションとして，メディアデータなどの人間の知覚に関わるデータを扱うものや統計・確率的な計算，入力に冗長性が存在するものなどが挙げられる．例えば，画像処理や音声処理の場合，人間が知覚できない程度であれば出力に誤差が生じても問題ないとされている．そのため，知覚できない程度の誤差の範囲内で処理を近似的に行うことで，体感的な画質や音質を維持しながら，処理の高速化・省電力化が達成できる．また統計処理や機械学習など，大量のデータを処理するアプリケーションにおいても Approx. Comp. は有効である．このようなアプリケーションでは，各データの処理結果の正確さ以上に，短時間でより多くのデータを処理することが重要視されるためである．

Approx. Comp. に関する研究は，プログラミング言語やアーキテクチャ，論理合成など様々なレイヤーで行われており，Approx. Comp. の適用方法もそれぞれ異なる．例えば Raha ら [4] は，アプリケーションの実行中に計算の近似度を動的に調節し，いくつかの計算をスキップすることにより，消費電力を削減する手法を提案している．また，Sasa ら [5] が提案するシステムでは，まずプログラマがコード内で，重要度が低く近似計算を行っても良い部分を指定する．そしてその情報を利用し，信頼度の低く，消費電力の少ないハードウェアに重要度の低い命令を自動的に割り当てることで消費電力を削減している．他にも，Shoushtari ら [6] は，SRAM 内の保護帯域の設定を緩和し，消費電力を削減している．このように，Approx. Comp. は近似計算の実現方法から応用先まで非常に多岐に渡り，盛んに研究されている一方で，様々なアプリケーションに対して統一的に Approx. Comp. を適用する方法については十分に検討されていない．

2.2 Approx. Comp. 基盤の実現にむけて

Approx. Comp. を適用することで，大きな効果が期待できるアプリケーションは多く存在するが，出力における許容誤差範囲や，近似計算が出力に与える影響は各アプリケーションに強く依存する．そのためプログラマは，出力における許容誤差を超過しない計算近似度や，どの計算がどのくらい出力へ影響を与えるか，アプリケーション毎に試行錯誤を繰り返しながらチューニングを行う必要がある．

こうした中で，我々はこれまで，過去の入出力を記憶することで実行自体を省略する，計算再利用に基づく高速化手法についての研究を行っており，現在の入力と過去の入力の一致比較に曖昧性を持たせることで Approx. Comp. を実現しうることに着目している [7]．また，一致比較の際の曖昧さを変動させることで，計算量とアプリケーションの出力精度を調整可能であることも確認している [3]．この知見を生かし，計算再利用と近似計算とを組み合わせた**近似計算再利用**により，様々なアプリケーションに対して統一的に Approx. Comp. を適用可能な計算基盤の実現を我々は目指している．Approx. Comp. を効果的に適用するためには，アプリケーション毎に要求される出力誤差の範囲内で最大限に計算を近似することが求められる．そこで，単調性を持つ関数に着目し，関数および入力区間毎の出力許容誤差に合わせて計算近似度を調整することで，近似計算再利用を効果的に適用する方法について検討する．続く 3 章では，計算再利用および近似計算再利用について述べ，4 章では，単調性を持つ関数の特徴，および単調性を利用した計算近似度の調整方法について述べる．

3. 近似計算再利用

本章では，計算再利用および近似計算再利用の概要について述べる．

3.1 計算再利用

計算再利用とは，プログラム内の関数実行時にその入力の組と出力の組を記憶しておき，再び同じ入力によりその関数が実行されようとした場合に，記憶された過去の出力を利用することで関数の実行自体を省略し，高速化を図る手法である．この計算再利用を行うためには，過去の入出力を記憶しておくための表が必要となる．本稿ではこれを再利用表と呼ぶ．ここで，再利用対象の関数が呼び出される場合を考える．再利用対象の関数が呼び出される際，まず再利用表を参照し，現在の入力と再利用表に記憶されている過去の入力とを比較する．もし，現在の入力が再利用表上のいずれかの入力と一致する場合，その入力に対応する出力を再利用表からレジスタやキャッシュに書き戻すことで実行を省略する．一方，現在の入力が再利用表のいずれの入力とも一致しない場合，その関数を通常通り実行する．そして，その関数の実行を終える際，実行中に出現した入出力を再利用表へと登録し，将来の再利用に備える．

ここで**図 1** に示すサンプルプログラム内の関数 gray に計算再利用を適用する例を説明する．このプログラムは，

```
1  int gray(int r, int g, int b) {
2    int x;
3    x = (r + g + b) / 3;
4    return(x);
5  }
6
7  void main() {
8    gray(0x11, 0x25, 0x40);
9    gray(0x12, 0x23, 0x43);
10   gray(0x11, 0x25, 0x40);
11   exit(0);
12 }
```

図 1　サンプルプログラム

再利用表

関数	入力			出力
gray	0x11	0x25	0x40	0x25
gray	0x12	0x23	0x43	0x26

図 2　関数 gray の入出力の登録

3 つの入力の平均を出力として返す関数 gray を 3 回呼び出すプログラムである．初期状態では，再利用表には何も登録されていないものとする．まず，8 行目の関数 gray が呼び出される際，再利用表を参照し，現在の入力と一致する過去の入力を検索する．しかし，再利用表には過去の入力は登録されておらず，一致する入力は存在しないため，8 行目関数 gray は通常通り実行され，return 命令の実行とともに再利用表に関数名および入出力を登録する．続いて，9 行目の関数 gray が呼び出される際も同様に，再利用表を検索する．この時，再利用表には 8 行目の関数 gray の呼び出しに対応する入出力が登録されているが，9 行目の呼び出しで用いられている入力とは一致せず計算再利用を適用できない．そのため，9 行目の関数 gray も同様に通常通り実行され，return 命令の実行とともに再利用表に関数名および入出力を登録する．9 行目の関数 gray の実行を終えた時点での再利用表の状態を**図 2** に示す．その後，10 行目の関数 gray が呼び出される際も同様に，再利用表を検索する．この時，再利用表には現在の入力と一致する，8 行目での呼び出しに対応する入出力が登録済みであるため，その出力を再利用することで実行が省略される．

3.2　計算再利用を用いた近似計算

前節で述べた計算再利用は，入力が完全に一致する場合にのみ適用できる．そのため，同じ入力が出現しないような処理では，計算再利用による高速化の恩恵を得ることができない．特に，センサデータなどを対象とした処理では，値の近い入力が多く出現するにもかかわらず，入力が完全

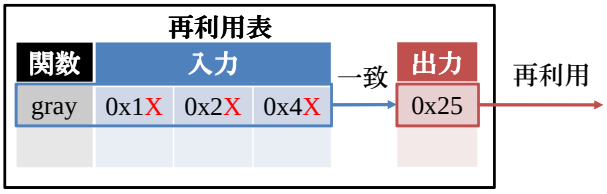


図 3　入力値にマスクを用いた近似計算再利用

に一致することが少ないため，計算再利用を適用できる機会は僅かである．

しかしこのような処理においても，計算再利用における入力一致比較の際，入力にある程度の不一致を許容する，近似計算再利用を用いることで計算再利用の恩恵を得ることができる．これにより，計算再利用を適用できる機会が増加するだけでなく，プログラムを変更することなく Approx. Comp. を適用することが可能となる．それでは，図 1 のサンプルプログラムにおいて関数 gray に近似計算再利用を適用する例を**図 3** を用いて説明する．なお，図 3 中の X はドントケア値を表している．この例では，再利用表への入力の登録時に入力 8bit のうち下位 4bit をマスクし，ドントケア値として登録することで，一致比較の際に下位 4bit の不一致を許容する．前節で見たように，通常の計算再利用では，9 行目の関数 gray は入力の不一致により計算再利用を適用できなかった．一方，近似計算再利用を用いる場合，9 行目での呼び出しに対応する入力の上位 4bit が，再利用表に記憶された 8 行目での呼び出しに対応する入力の上位 4bit と一致しているため近似計算再利用を適用することができる．このように，計算再利用と近似計算を組み合わせることで，計算再利用を適用する機会を増加させることができる．また，ドントケア値を用いることで複数の入力値と一致する表現となるため，従来，再利用表の複数行を使用していた登録情報が 1 行に統合され，再利用表に必要な記憶容量も削減することができる．さらに，一致比較における不一致を許容するビット幅を調整することで，容易に計算近似度を調整することができる．しかし，僅かな入力の変化に対して出力が大きく変化する関数に近似計算再利用を適用する場合，計算精度を大きく損う恐れがあるため，不一致を許容するビット幅の調整は関数の特徴を考慮して適切に行わなければならない．

4.　単調性を利用した近似計算再利用

関数が単調性を持つ場合，単調性を持つ区間の特徴を考慮して不一致を許容するビット幅を調整することにより，出力の許容誤差を超過しない最適な計算近似度で，関数に近似計算再利用を適用することができる．本章では，単調性を持つ区間の特徴と，その区間における近似計算再利用の適用方法について述べる．

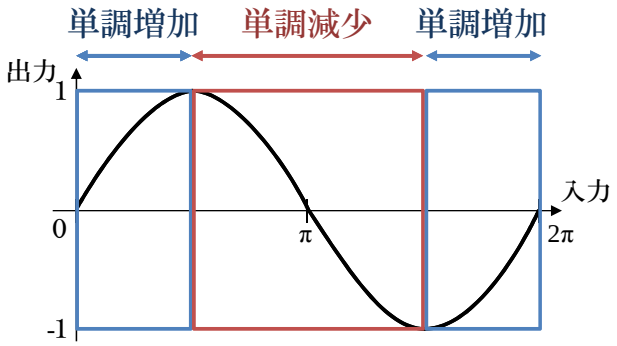


図 4　sin 関数と各区間の極値

4.1　単調性を持つ区間の特徴

近似計算において，その適用対象となる処理が取り得る入力に対し，出力の極値を知ることは重要である．それは，入力に対してある区間を定義した場合，その区間内の極小値と極大値の差が，その区間の入力を全て同一とみなした際に発生しうる出力誤差の最大値に相当するためである．この発生しうる誤差とその関数の出力に許容される誤差を比較することで，その区間に対する最適な計算近似度を求めることができる．

例えば，関数 $f(x)$ が単調増加であるとすると，$x_1 < x_2$ ならば必ず $f(x_1) < f(x_2)$ となるため，$f(x)$ に対して定義した任意の入力区間 $[x_1, x_2]$ において，区間内の入力の最小値 x_1 に対応する出力が当該区間の極小値，最大値 x_2 に対応する出力が極大値となることが，それぞれ保証される．単調減少の場合も同様に，入力区間の最小値に対応する出力がその区間の極大値，最大値に対応する出力がその区間の極小値となることが保証される．この例を**図 4** を用いて説明する．図 4 は定義域 $[0, 2\pi]$ における sin 関数のグラフを表している．sin 関数は一定周期で単調増加と単調減少を繰り返しており，グラフに示すように $[0, \frac{1}{2}\pi]$ および $[\frac{3}{2}\pi, 2\pi]$ が単調増加区間，$[\frac{1}{2}\pi, \frac{3}{2}\pi]$ が単調減少区間である．ここで $[0, \frac{1}{2}\pi]$ に着目すると，入力区間の最小値である入力 0 に対する出力 0 が極小値，最大値に入力 $\frac{1}{2}\pi$ に対する出力 1 が極大値となる．残る $[\frac{3}{2}\pi, 2\pi]$，$[\frac{1}{2}\pi, \frac{3}{2}\pi]$ においても，入力区間の両端が極値となる．このように，単調性を持つ区間では，その入力区間の両端に対応した出力がその区間の極値であるため，両端に対応した出力の差が，その区間の入力を全て同一であると見なした場合に発生しうる出力誤差の最大値に相当する．

4.2　単調性を利用した入力近似区間の調整

近似計算再利用を適用する際，出力誤差の許容範囲に合わせ，不一致を許容するビット幅を適切に調整する必要がある．ここで，計算再利用の入力の一致比較において一部の不一致を許容することは，関数のある入力区間の出力を同一とみなす，と言い替えることができる．つまり近似計算再利用を適用するために，出力誤差の許容範囲に合わせ，

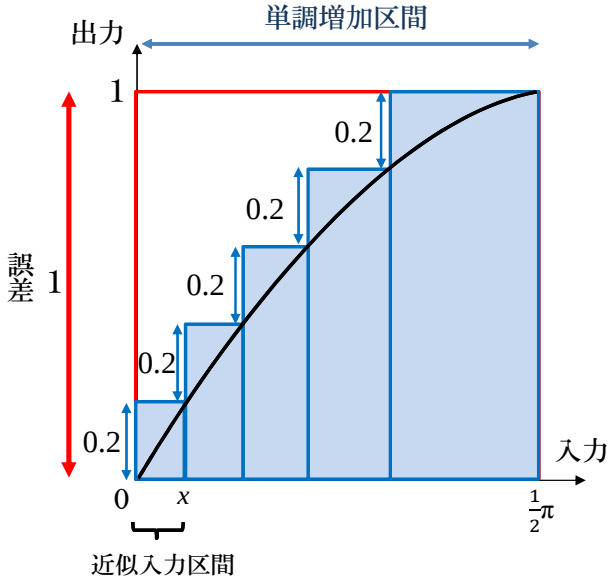


図 5　sin 関数における入力近似区間調整

出力を同一とみなす入力の区間を適切に調整する必要がある．なぜなら，最適な区間の大きさに対して，出力を同一とみなす入力の区間が大きすぎる場合，その区間の近似計算再利用によって発生する誤差が許容できる範囲を越えてしまう恐れがあるためである．一方，その区間が小さすぎる場合，近似計算再利用を適用する機会を十分に得られず，高速化・省電力化を実現できない．

そこで単調性を持つ区間の特徴を利用し，その関数の出力において許容される誤差の範囲を超えないようにしつつ，出力を同一とみなす入力の区間が最大となるように各区間の大きさを調整する．本稿では，出力を同一とみなす入力の区間を**入力近似区間**と呼ぶ．この入力近似区間調整の例を**図 5** に示す．図 5 は，sin 関数において単調増加する $[0, \frac{1}{2}\pi]$ を拡大した図であり，出力の許容誤差を 0.2 としたときの入力近似区間調整の様子を表している．もし，単調増加の $[0, \frac{1}{2}\pi]$ の区間全体をひとつの入力近似区間として近似計算再利用を適用したとすると，この区間の極小値は 0（入力 0），極大値は 1（入力 $\frac{1}{2}\pi$）となるため，発生しうる誤差は最大 1 となり，許容される誤差を越えてしまう．そこで，区間を分割し，発生しうる誤差が許容される誤差よりも小さくなる区間を求める．このとき，$[0, \frac{1}{2}\pi]$ は単調増加区間であり，極小値は左端，極大値は右端の値となる．つまり，図 5 に示すように，区間の左端を入力 0 とする場合，$\sin x = 0.2$ となる x を右端とする区間 $[0, x]$ が，入力近似区間となる．またこれ以上区間を大きくすると，発生しうる誤差が許容誤差の 0.2 を越えてしまうため，この区間が，左端を 0 とした場合に許容誤差を超過しない最も大きな入力近似区間となる．残りの $[x, \frac{1}{2}\pi]$ も同様に，左端と右端の値の差が許容誤差を超えないように区間を分割していくことで，定義域を適切な大きさの入力近似区間に分割できる．以上で得られた入力近似区間に従い，不一致を許

容するビット幅を変更することで，関数の出力誤差を許容範囲内に保ちつつ，最大限の計算近似度で近似計算再利用を適用することができる．また，再利用表に入力近似区間を記憶する際，各入力近似区間はドントケア値を用いて，それぞれ 1 行で表現されるため，関数毎に必要となる再利用表の行数は，入力近似区間と同数となる．そのため，定義域を最小個数で分割することで，必要な再利用表の行数，つまり記憶容量を最小限に抑えることができる．

5. 評価

以上で述べた，近似計算再利用における単調性利用の有効性を確かめるため，シミュレーションによる評価を行った．

5.1 評価環境

評価には，メディア系ベンチマークである MediaBench から，3D グラフィクスを生成する mesa に含まれる mipmap を用い，log 関数に近似計算再利用を適用した際の再利用率および再利用表への登録パターン数を計測した．再利用率については，理想性能を求めるために無限大のサイズの再利用表を仮定した場合の再利用率と，現実的なサイズである 128KBytes（4K 行）の再利用表を仮定した場合の再利用率の両方を計測した．なお計算基盤の一部として，対象の命令区間に対し自動的に計算再利用を行う自動メモ化プロセッサ（Auto-Memoization Processor）[8] を用いることを想定し，シミュレーションを行った．

なお本評価では，単調性を活用した近似計算再利用の有効性を確認するという目的で，図 4 で示したような単調性に関する情報を実行時システムが利用可能であると仮定してシミュレーションを行った．この単調性に関する情報は，比較的単純な処理に対しては，プログラムの静的構文解析等を用いることで自動的に抽出可能である．また，単調性の自動検出が難しいような比較的複雑な処理・関数や，関数の定義域のうち一部の入力区間にのみ単調性が成立するような関数においては，プログラマにヒント情報を記述させることで，この単調性に関する情報を取得することを想定している．これらの具体的な方法については，現在検討を進めている段階である．

計算基盤の一部として想定した自動メモ化プロセッサは，単命令発行の SPARC V8 アーキテクチャをベースとしており，計算再利用のための機構として再利用表（MemoTbl）と，それに対する書き込みバッファとして再利用バッファ（MemoBuf）を持つ．評価では再利用バッファのサイズは 64KBytes とした．なお，自動メモ化プロセッサは再利用表の一部に CAM を用いることを想定しており，サイズは 32Bytes 幅 ×4K 行の 128KBytes とした．

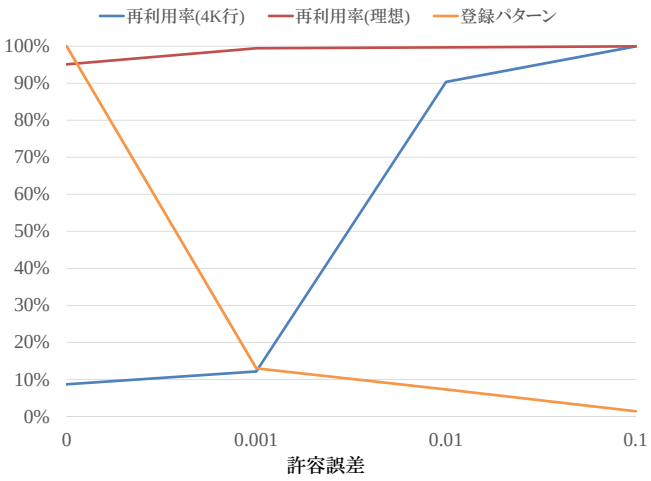


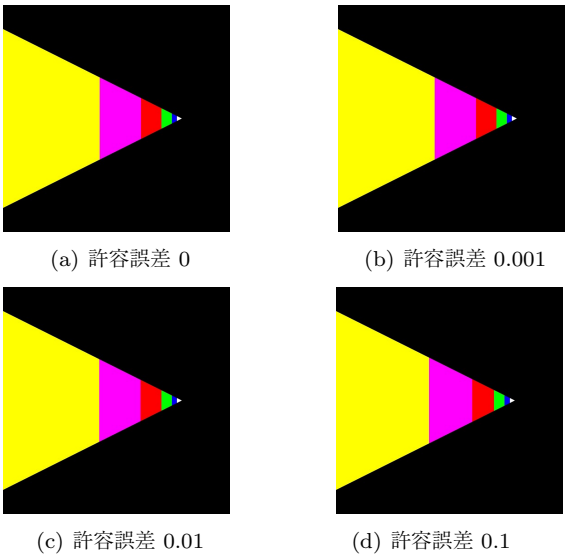
図 6 評価結果

5.2 評価結果

評価結果を**図 6** に示す．図 6 は，今回対象とした mipmap に含まれる log 関数の，出力値に対する許容誤差の絶対値を 0 から 0.1 に変化させた際の，再利用率と再利用表への登録パターン数の変化を表している．なお，各再利用率は log 関数の総呼び出し回数に対する，近似計算再利用が適用できた回数の割合とした．また，再利用表への登録パターン数は，log 関数の許容誤差を 0 とした時の登録パターン数を 1 として正規化した値で示している．

まず，再利用表への登録パターン数に着目すると，許容誤差が 0 の時の値を 100%とした場合，許容誤差 0.001 では 13.0%，許容誤差 0.01 では 7.3%まで減少しており，許容誤差の拡大とともに登録パターン数が大きく削減できていることがわかる．それに伴い，再利用率も向上している．再利用表サイズを無限大と仮定した場合の再利用率は，許容誤差を 0 とした場合で 95.1%であったのに対し，許容誤差 0.001 では 99.5%，許容誤差 0.01 では 99.7%までそれぞれ向上しており，許容誤差の拡大とともに再利用の可能性も向上していることがわかる．また，現実的なサイズである 4K 行の再利用表を仮定した場合の再利用率も，許容誤差を 0 とした際の再利用率がわずか 8.7%であったのに対し，許容誤差 0.001 では 12.2%，許容誤差 0.01 では 90.4%と著しく向上している．このことから，許容誤差の拡大とともに，少ない入力パターンで全入力が表現できるようになったことにより，再利用表に必要な記憶容量が減少し，全入力パターンの大部分に対してマッチするような入出力情報のセットを再利用表上に保持し続けることができるようになったことが伺える．これらの結果から，近似計算再利用では許容誤差を増加させるにつれ，Approx. Comp. による高速化・省電力化の効果をより得られることを確認できた．

次に，近似計算再利用が出力に与える影響を確認するにあたり，許容誤差毎の mipmap の出力結果を**図 7** に示す．許容誤差を 0 とした場合の出力結果である**図 7(a)** と比較して，許容誤差 0.001 の**図 7(b)**，許容誤差 0.01 の**図 7(c)**



(a) 許容誤差 0　(b) 許容誤差 0.001

(c) 許容誤差 0.01　(d) 許容誤差 0.1

図 7　mipmap の出力結果

ではほとんど変化は見られないことがわかる．また，許容誤差 0.1 の**図 7(d)** では，図 7(a) と比較してわずかに変化がみられるが，図 6 から，許容誤差 0.1 と 0.01 の間には再利用率等に大きな差はないため，今回評価に用いたプログラムでは，許容誤差を 0.01 と設定すれば十分であると考えられる．以上の結果から，関数の単調性を考慮して入力近似区間を適切に調整することで，関数の許容誤差に合わせた近似計算再利用を適用可能であることを確認できた．

6. おわりに

本稿では，Approximate Computing を統一的に適用可能な計算基盤の実現に向け，近似計算と計算再利用を組み合わせた近似計算再利用において，関数の単調性を利用することで，最適な計算近似度で Approximate Computing を適用する方法を検討した．MediaBench から，3D グラフィクスを生成する mesa に含まれる mipmap を用いて評価を行った結果，mipmap 内の log 関数において通常の計算再利用を用いた場合の再利用率が 8.7%であったのに対し，log 関数の出力許容誤差を 0.01 とした場合で再利用率が 90.4%まで向上すること，また計算再利用に必要な記憶容量が 7.3%まで低減できることを確認した．今後の課題として，関数毎に単調性を持つ区間を自動検出する機構の検討や，関数毎の誤差がアプリケーションの出力に及ぼす影響の調査が挙げられる．

参考文献

[1] Hadi, E., Adrian, S., Luis, C. and Doug, B.: Architecture Support for Disciplined Approximate Programming, *Proc. 17th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12)*, pp. 301–312 (2012).

[2] Vaibhav, G., Debabrata, M., Anand, R. and Kaushik, R.: Low-Power Digital Signal Processing Using Approximate Adders, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 32, No. 1, pp. 124–137 (2013).

[3] Sato, Y., Tsumura, T., Tsumura, T. and Nakashima, Y.: An Approximate Computing Stack based on Computation Reuse, *Proc. 3rd Int'l Workshop on Computer Systems and Architectures (CSA'15)*, pp. 378–384 (online), DOI: 10.1109/CANDAR.2015.35 (2015).

[4] Raha, A., Venkataramani, S., Raghunathan, V. and Raghunathan, A.: Quality Configurable Reduce-and-Rank for Energy Efficient Approximate Computing, *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pp. 665–670 (2015).

[5] Sasa, M., Michael, C., Sara, A., Qi, Z. and C., R. M.: Chisel: Reliability- and Accuracy-Aware Optimization of Approximate Computational Kernels, *Proc. ACM Int'l Conf. on Object Oriented Programming Systems Languages & Applications (OOPSLA'14)*, pp. 309–328 (2014).

[6] Shoushtari, M., BanaiyanMofrad, A. and Dutt, N.: Exploiting Partially-Forgetful Memories for Approximate Computing, *IEEE Embedded Systems Letters*, Vol. 7, No. 1, pp. 19–22 (2015).

[7] 津邑公暁，清水雄歩，中島康彦，五島正裕，森眞一郎，北村俊明，富田眞治：ステレオ画像処理を用いた曖昧再利用の評価，情報処理学会論文誌コンピューティングシステム，Vol. 44, No. SIG 11(ACS 3), pp. 246–256 (2003).

[8] Tsumura, T., Suzuki, I., Ikeuchi, Y., Matsuo, H., Nakashima, H. and Nakashima, Y.: Design and Evaluation of an Auto-Memoization Processor, *Proc. Parallel and Distributed Computing and Networks*, pp. 245–250 (2007).