Vol.2019-ARC-240 No.31
Vol.2019-SLDM-191 No.31
Vol.2019-EMB-53 No.31
2019/2/28

近似レベルを動的制御可能なアーキテクチャの提案

道上 和馬[1,a)]　中村 朋生[2]　小泉 透[2]　入江 英嗣[2,b)]　坂井 修一[2]

概要：Approximate Computing は，計算精度と引きかえに実行時間と消費電力の双方を削減する技術である．この技術の適用範囲を広げる上での課題のひとつは，誤差を許容範囲内に収めることである．本論文では，この許容範囲がしばしばユーザの主観によって動的に変化することに着目し，計算精度を動的制御可能なアーキテクチャと計算精度の段階的制御が可能なループ近似手法「Loop Body Switching」を提案する．近似の積極度合いを指示する近似レベルを，Control and Status Register（CSR）に保持し，その値で Loop Body Switching の計算精度を制御する．提案するアーキテクチャをシミュレータ上に実装し，4 つのベンチマークを用いて評価をおこなう．近似レベルの増加に対して実行サイクル数は段階的に減少し，専用の分岐命令とハードウェア装置によりさらに実行サイクル数が減少した．

1. はじめに

情報化社会の発展により，処理しなければならない情報量は増加し続けている．これに対処するためには計算資源の増強または計算の効率化が必要である．Approximate Computing は，近似により計算の効率化を図る技術である．この技術は，多少の誤差を許容するアプリケーションに対して適用可能である．その例としてマルチメディア情報処理と機械学習が挙げられる．これらのアプリケーションでは入力データ自体のノイズや，人間の知覚能力の限界から高い計算精度を必要としないことがある [1]．このような場合に，Approximate Computing は誤差を犠牲にすることで計算の高速化や低消費電力化を達成することができる．

この技術の適用範囲を広げる上での課題のひとつは，誤差を許容範囲内に収めることである．この許容範囲はしばしばユーザの主観によって動的に変化することがある [2]．例えば，近似によって出力画像の品質が変わる場合，どの程度の品質までを許容することができるかはユーザの感覚による．比較的許容範囲の狭いユーザを基準にして静的に許容範囲を設定してしまうと，比較的許容範囲の広いユーザに対して近似の効果を十分に引き出せなくなる．このことに着目し，本論文ではハードウェアによって計算精度の動的制御をサポートするアーキテクチャを提案する．我々は，近似レベルを「ハードウェアがどれだけ近似を積極的におこなうか指示する値」と定義した．この近似レベルを CSR として保持し，外部入力やハードウェアの検出によって動的に設定することで，適切な計算精度での実行を可能にする．また，本論文では提案するアーキテクチャ上で動作する近似手法のひとつとして，計算精度の段階的制御が可能なループ近似手法「Loop Body Switching」を提案する．この手法は近似レベルによって決まる確率で，元の正確なループボディの代わりに近似ループボディを実行する．専用の分岐命令である「確率的分岐命令」とループボディを選択するためのハードウェア装置である「分岐決定器」を追加することで，ループボディ選択と分岐にかかるオーバーヘッドを小さくする．提案手法は，RISC-V[3] を命令セットとするアーキテクチャに，確率的分岐命令と分岐決定器を追加することで実現できる．

提案手法の評価にあたり，提案するアーキテクチャをシミュレータに実装し，4 つのベンチマークに Loop Body Switching を適用する．32 段階の近似レベルごとの実行サイクル数と誤差を計測し，段階的な近似が可能であることを示す．また，分岐決定器がある場合とない場合の比較をおこない，分岐決定器の有効性を評価する．本論文の貢献は次の通りである．

- 近似レベルを動的制御可能なアーキテクチャの提案
- 計算精度の段階的制御が可能な Loop Body Switching の提案
- Loop Body Switching をサポートする確率的分岐命令と分岐決定器の提案

[1] 東京大学 工学部
The University of Tokyo
[2] 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, University of Tokyo
[a)] michigami@mtl.t.u-tokyo.ac.jp
[b)] irie@mtl.t.u-tokyo.ac.jp

ⓒ 2020 Information Processing Society of Japan

- 4 つのベンチマークを用いた近似レベルに対する誤差と実行サイクル数の変化の調査
- 分岐決定器の有無による実行サイクル数の変化の調査

以降に本論文の構成を示す．2 章で関連研究について述べ，3 章で近似レベルを動的制御可能なアーキテクチャ，Loop Body Switching とそれをサポートする確率的分岐命令と分岐決定器の詳細を説明する．4 章では評価に用いるベンチマークとシミュレーション環境について述べる．5 章では近似レベルに対する誤差と実行サイクル数の変化と，分岐決定器の有無による実行サイクル数の変化の結果を示し，6 章でその結果に対する考察をおこなう．最後に 7 章で結論を述べる．

2. 関連研究

Approximate Computing の近似手法はアルゴリズムレベルから回路レベルまでと多岐にわたる分野で研究されている．具体的には，ビット長を動的に削減して演算をおこなう手法 [4] やキャッシュミス時にロード値を確率的に近似する手法 [5]，精度が可能な近似演算器 [6] などが提案されている．佐藤ら [7] と Liu ら [8] はハードウェアと協調したメモ化技術を提案している．Loop Perforation[9] はループ中のイテレーションを完全にスキップすることで誤差対性能のトレードオフをおこなう手法である．一方，Approximate Loop Unrolling[10] はアンローリングされたループボディを補間処理に近似するコンパイラ最適化手法である．この手法は Loop Perforation と異なりイテレーションを完全にスキップせずに補間処理を行うため，比較的精度を高く保つことができる．補間処理は配列への代入処理を対象としており，配列の 1 つ前の値を補間値として使う Nearest Neighbor(NN) 法と 1 つ前と 1 つ先の値の平均値を補間値として使う Linear Interpolation(LI) 法の二つがある．

近似手法そのもの以外の研究も提案されている。近似的可能な部分を自動的に発見する手法 [11] や Approximate Computing をサポートするプログラミング言語 [12] などが挙げられる．また, 近似によって生じる誤差を制御する手法として，事前に与えられる入力データのサンプルをもとに近似を行う手法 [13] や， 実行時に出力値を監視する手法 [14], [15] が提案されている．Miguel ら [2] は出力品質の保証が可能な Anytime Algotirhm に基づいた計算モデルを提案している．

3. 提案手法

3.1 モチベーション

Approxmimate Computing の課題のひとつは，誤差を許容範囲内に収めつつ，相応の実行時間・消費電力を削減することである．画像処理などのアプリケーションにこの技術を適用する場合，人間の知覚が個人や状況によって異なることにより，誤差の許容範囲はしばしば動的に変化する．許容範囲が狭い場合に合わせてしまうと十分に近似の効果を得られないが，許容範囲を緩めてしまうとアプリケーションの利用が難しくなる．もしユーザが実際に必要とする品質に応じた計算精度での実行が可能になれば，Approximate Computing によって得られる利得はより大きくなる．これを達成するための課題は大きくわけて次の二点である．

- 同じバイナリを，どのようにして段階的に異なる計算精度で実行するか
- ユーザが必要としている品質をどのように推定するか

もし一番目の課題が解決すれば，二番目の課題は最も単純にはユーザから直接フィードバックを取ることで解決することができる．一番目の課題は実行時に書きかえ可能な計算精度を示す内部レジスタとその値により計算精度が変更可能な近似手法によって実現できる．例えば，ループのイテレーションごとに元の正確なループボディと近似的なループボディのどちらかを実行することにし、近似的なループボディを選択する確率を内部レジスタによって決めれば動的に計算精度を変更できる．

3.2 概要

本論文の提案手法の概要を説明する．提案手法は大きく分けて次の三つから構成される．

- 近似レベルを動的に制御できるアーキテクチャ（3.3 節参照）
- 提案するアーキテクチャ上で実行されるループ近似手法「Loop Body Switching」（3.4 節参照）
- Loop Body Switching をサポートする「確率的分岐命令」（3.5 節参照）と「分岐決定器」（3.6 節参照）

一つ目は，近似レベルの値を CSR として持つアーキテクチャである．近似レベルはハードウェアがどれだけ近似を積極的におこなうか指示する値である．二つ目は確率的に近似ループボディを実行する計算精度の変更が可能なループ近似手法である．三つ目はそのループ近似手法の確率的分岐をサポートすることで性能を向上させるハードウェア機構である．ハードウェア機構は，近似レベルに応じた確率で分岐するための確率的分岐命令とその命令の分岐方向を決定する分岐決定器によって構成される．

3.3 近似レベルの制御

近似レベルの値を保持する CSR は 32 ビットのレジスタであり最大で 2^{32} 段階のレベルを持つことができる．近似レベルが N 段階であるとき近似レベルの範囲を 0 から N-1 とする．近似レベルはプログラム中の命令や外部割り込み，ハードウェアの検出によって書き換える．近似レベルを利用する近似手法は，近似レベルが大きくすると計算精度が低下するかわりに実行サイクル数と消費電力が削減

Vol.2019-ARC-240 No.31
Vol.2019-SLDM-191 No.31
Vol.2019-EMB-53 No.31
2019/2/28

されるように設計する．

次に本論文での実装を述べる．近似レベルは 5 ビットで表され，0 から 31 の 32 段階となる．近似レベル 0 は全く近似しないことを意味する．近似レベルはプログラム中の命令によって CSR 値を書き換えることで変更される．

3.4 Loop Body Switching

この手法は，元の正確なループボディから近似ループボディを生成し，一部のイテレーションにおいて近似ループボディを実行することにより近似をおこなう．イテレーションごとにどちらかのループボディを選択できるようにループ構造の変形と分岐命令の挿入をおこなう．Loop Body Switching の動作を示すために， 近似対象のループの例をソースコード 1 に，近似により変形されたあとのループをソースコード 2 に示す．

ソースコード 1: 近似対象のループの例

```
for(int i=0; i<b; i++)
{
    a[i] = func(2*i);
}
```

ソースコード 2: 変形されたループの例

```
for(int i=0; i<b; i++)
{
    double p = approx_level/(max_approx_level+1);
    bool taken = Bernoulli(p);
    if(taken == false)
    {
        // Exact Body
        a[i] = func(2*i);
    }
    else
    {
        // Approximate Body
        a[i] = a[i-1];
    }
}
```

変形後のループはループボディ選択部と元の正確なループボディ，近似ループボディの三つから構成される．ループボディ選択部ではどちらのループボディを実行するかを確率的に決定し条件分岐をおこなう．変数 *approx_level* に依存するベルヌーイ分布から真偽値をサンプリングすることでループボディを選択する．近似ループボディでは元の正確なループボディを近似した処理をおこなう．この例では NN 法による補間をおこなっている．

本論文では，ループボディ選択部の処理をハードウェアがおこなう．まず，近似ループボディの選択確率はプログラム中の変数ではなく，ハードウェアの CSR によって決定する．これにより動的な外部からの書き換えやハードウェアからの参照が容易になる．ループボディ選択の分岐は，通常の条件分岐命令ではなく専用の確率的分岐命令と分岐決定器によっておこなう．確率分岐命令はフェッチ時に分岐方向が分岐決定器によって決定される命令であり，分岐予測ミスの発生を完全に防ぐことができる．

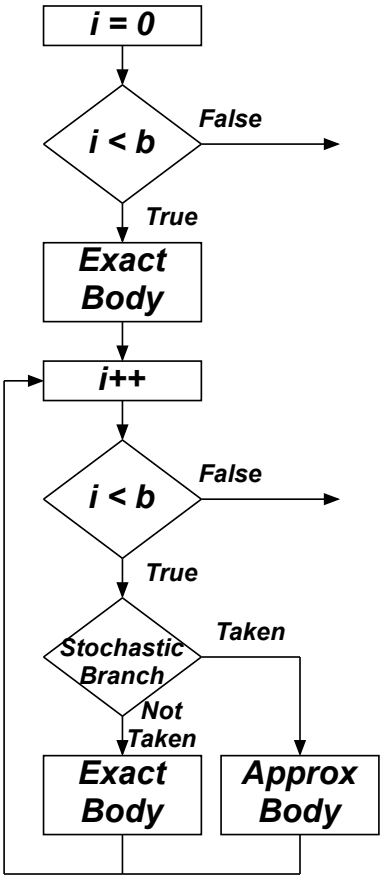


図 1: 近似されたループのフローチャート

近似ループボディの生成，ループ構造の変形，分岐命令の挿入は手作業でアセンブリを編集することでおこなう．ただしこの手作業での編集は，コンパイラにより自動処理可能と考えているものに限定している．近似されたループのフローチャートを**図 1** に示す．近似ループボディでは補間処理をおこなうか何もしないかのどちらかである．補間処理は NN 法による配列の値の補間をおこなう．NN 法では補間値としてひとつ前のイテレーションの値を使用するため，最初のイテレーションでは適切な補間をおこなうことができない．このため最初のイテレーションでは必ず元の正確なループボディが実行されるようにループ構造を変形する．

ハードウェアサポートなしで，Loop Body Switching をおこなう場合，ループボディ選択部のオーバーヘッドが大きい．まず，近似レベルに応じたベルヌーイ分布から分岐方向をサンプリングするためのオーバーヘッドが生じる．さらに分岐予測器は近似レベルの情報を知らない状態で確率的な分岐を当てなければならない．この場合，分岐予測

ⓒ 2020 Information Processing Society of Japan

Vol.2019-ARC-240 No.31
Vol.2019-SLDM-191 No.31
Vol.2019-EMB-53 No.31
2019/2/28

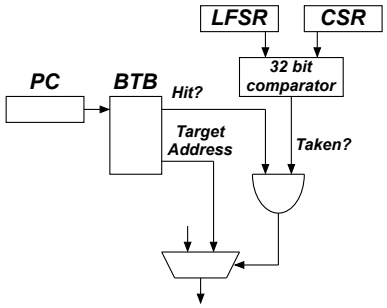


図 2: 分岐決定器

ミスの発生によりさらにオーバーヘッドが大きくなる．

3.5 確率的分岐命令

確率的分岐命令はハードウェアによって次のプログラムカウンタ（PC）の値が実行時に決定される分岐命令である．分岐先アドレスはオペランドの即値に PC の値を加算した値である．フェッチ時に分岐方向が決まるため分岐予測ミスが発生しない．

次に本論文での実装を述べる．実行ステージでは分岐先アドレスが計算される．分岐方向にかかわらず Branch Target Buffer（BTB）への書き込みを必ずおこなう．必要に応じて分岐予測器に分岐方向の情報を与える．レイテンシは 1 である．

3.6 分岐決定器

分岐決定器は確率的分岐命令の次の PC を決定する装置である．分岐方向は 1（分岐する）または 0（分岐しない）を出力する分岐方向決定器分岐によって決定される．分岐をおこなう場合，BTB から得られた Target Address が次の PC となる．BTB は分岐予測器と共有される．本論文では次に説明するベルヌーイ分岐決定器を使用する．ベルヌーイ分岐決定器を使用する場合の PC 決定の様子を**図 2** に示す．この分岐決定器はベルヌーイ分布からの疑似的なサンプリングをおこなう．内部にハードウェア擬似乱数生成器である線形帰還シフトレジスタ（LFSR）を備えている．近似レベルが *nbit*，LFSR が *mbit* であるとき出力値を式（1）で決定する．

$$approx\_level \cdot 2^{m-n} \geq random\_value \tag{1}$$

ただし $1 \leq n \leq m$ とし，$random\_value$ は LFSR の値，$approx\_level$ は近似レベルを意味する．ここで，LFSR が 0 を除く $2^m - 1$ 通りの値を出力し各値の出現確率が一様であるとすると式（1）が真となる確率は式（2）で表される．

$$P(true) = \frac{approx\_level \cdot 2^{m-n}}{2^m - 1} \tag{2}$$

$$\approx \frac{approx\_level}{2^n}$$

$$= \frac{approx\_level}{max\_approx\_level + 1} \tag{3}$$

表 1: アーキテクチャの主なパラメータ．

ISA	RV64G
Fetch Width	4
Issue Width	int: 2，fp: 2，mem: 2
Instruction Window	int: 32，fp: 16，mem: 16
Branch Predictor	g-share，BHR: 10bit，PHT: 8KiB
Branch Target Buffer	2K-entry，4-way
Load Store Queue	load: 48-entry，store: 48-entry
L1 Instracution Cache	32KiB，8-way，4-cycle hit latency
L1 Data Cache	32KiB，8-way，4-cycle hit latency
L2 Data Cache	256iKB，4-way，12-cycle hit latency
L3 Data Cache	2MiB，16-way，44-cycle hit latency
Main Memory	200-cycle hit latency

式（3）は $2^m \gg 1$ を仮定して得られる近似式である．

4. 評価方法

4.1 シミュレーション環境

サイクルアキュレートなシミュレータである鬼斬弐 [16] に提案するアーキテクチャを実装し評価をおこなった．評価するプロセッサの構成を**表 1** に示す．この構成を基本に, 3 章で述べた手法を実装したプロセッサをシミュレートする．

4.2 ベンチマーク

表 2 に評価に使用するベンチマークの概要を示す．PERFECT[17] は様々な分野のアプリケーションによって構成される組み込み計算向けのベンチマークスイートである．AxBench[18] は近似に適したアプリケーションから構成されるベンチマークスイートである．PERFECT から histeq，AxBench から jpeg，kmeans，sobel の計 4 つのベンチマークを選出した．これらのベンチマークの出力は画像であり，視覚的に出力品質を確認することができる．

ベンチマークは GCC version8.1.0 でコンパイルした．一度アセンブリにコンパイルし，手作業でループ構造の変形と確率的分岐命令，近似レベル変更命令の挿入をおこない，バイナリを生成する．測定区間では近似レベルを固定して各近似レベルに対する実行サイクル数と誤差を測定する．

実行サイクル数の測定の開始位置と終了位置は，近似対象のループの前後またはそれを含む最外ループの前後とする．近似対象のループを含む最外ループを測定区間とする例をソースコード 3 に示す．この例に示すように近似対象のループ以外の処理が測定区間に含まれる場合もある．

ソースコード 3: 測定区間の例

```
// 測定開始
// 最外ループ
for(int i=0; i<N; i++)
{
```

ⓒ 2020 Information Processing Society of Japan

表 2: 評価に使用するベンチマーク．入力データはシミュレーション時に用いた入力データである．

ベンチマーク名	分野	説明	入力データ
histeq	画像処理	ヒストグラム均等化	480x640 ピクセルのグレイスケール画像
jpeg	圧縮	JPEG 圧縮	512x512 ピクセルのカラー画像
kmeans	機械学習	K-means クラスリング	512x512 ピクセルのカラー画像
sobel	画像処理	ソーベルフィルタによるエッジ検出	512x512 ピクセルのカラー画像

```
    // 近似対象のループ
    for(int j=0; j<M; j++){...}
    // 近似対象外の処理
    func();
}
// 測定終了
```

4.3 誤差評価

誤差の指標として正規化平均平方二乗誤差（NRMSE）を用いる．画像の画素数を n， 原画像と近似画像の i 番目の画素値をそれぞれ p_i，p'_i，画素値の上限値と下限値をそれぞれ p_{max}，p_{min} とすると誤差は次の式で表される．

$$NRMSE = \frac{\sqrt{\frac{1}{n}\sum_i^n (p_i - {p'}_i)^2}}{p_{max} - p_{min}} \quad (4)$$

5. 結果

図 3（histeq），**図 4**（jpeg），**図 5**（kmeans），**図 6**（sobel）に実行サイクル数と誤差の結果を示す．横軸は近似レベル，左の縦軸は実行サイクル数を近似レベル −1 の実行サイクル数で正規化した正規化実行サイクル数，右の縦軸は NRMSE である．近似レベル −1 は，近似のためのアセンブリの編集をおこなわずに生成されたバイナリの実行結果を意味する．したがって，近似レベル −1 をベースラインとする．また，**図 7**（histeq），**図 8**（jpeg），**図 9**（kmeans），**図 10**（sobel）に近似レベル −1，6，12，18，24，30 で実行した場合の出力画像を示す．

誤差対実行サイクル数のトレードオフ　図に示された結果から，分岐決定器を用いる場合，近似レベル 0 から 31 の範囲では近似レベルが大きくなると誤差は増加し，実行サイクル数は減少することがわかる．これは近似レベルの制御によって誤差対実行サイクル数のトレードオフ制御が可能であること示している．動的に近似レベルを変更する手法と組み合わせることで実際に必要とされる出力品質での実行が可能となる．

実行サイクル数の変化　分岐決定器ありの場合の実行サイクル数に注目する．図 3（histeq）の近似レベル 26 から 27 の変化を除いて，近似レベル 0 から 31 の範囲では近似レベルが大きくなると正規化実行サイクル数は単調減少する．ベンチマークごとに減少度合いは異なり，近似レベル 31 での正規化実行サイクル数は histeq が 0.71，jpeg が 0.76，kmeans が 0.051，sobel が 0.16 である．式（2）により，近似レベル 31 のときに近似ループボディを実行する確率は約 0.97 であり，かなり高い確率で処理が少ない近似ループボディを選択するはずだが，histeq と jpeg の実行サイクル数は減少量が小さい．histeq の実行サイクル数の減少量が小さいのは，元の正確なループボディの処理が小さく近似ループボディを実行してもあまり実行サイクル数が減少しないからである．jpeg の実行サイクル数の減少量が小さいのは，測定区間に近似対象のループ以外の処理を含むからである．これはソースコード 3 のように近似とは無関係な func() 処理を含むことに相当する．

図 7（histeq）を除いて，近似レベル 0 などの小さい近似レベルでは近似手法を適用しない場合よりも実行サイクル数が増加する．これは近似による実行サイクル数の減少よりも，近似のためのループ構造の変形や確率的分岐命令の挿入による実行サイクル数の増加が大きいためである．

分岐決定器の効果　次に分岐決定器なしと分岐決定器ありの場合を比較する．図 3（histeq）と図 4（jpeg）は正規化実行サイクル数に大きな差がある一方で，図 5（kmeans）と図 6（sobel）ではほとんど差がないことがわかる．分岐決定器なしの場合は，分岐決定器ありの場合に比べて実行サイクル数の最大で histeq は 44.5%，jpeg は 4.6%，kmeans は 0.9%，sobel は 5%増加する．この増加量の違いは，元の正確なループボディの実行にかかる実行サイクル数の大きさの違いによって生じる．元の正確なループボディの実行にかかる実行サイクル数が大きい場合，分岐予測ミスペナルティの影響が相対的に小さなくなる．このため，分岐決定器の使用により分岐予測ミスが減少してもその影響が見えにくい．また，全てのベンチマークおける，全ての近似レベルにおいて分岐決定器ありの場合の正規化実行サイクル数は分岐決定器なしの場合の実行サイクル数以下である．これは今回用いた分岐予測器を使用する場合，分岐決定器の使用により性能が劣化することがないことを示している．また，図 4（jpeg）と図 5（kmeans）を見ると，はじめは近似レベルが大きくなると分岐決定器なしとありの場合の差が大きくなっていくが，途中からその差が小さくなっていくことがわかる．これは近似レベル 0 から近似レベル 16 への変化では確率的分岐命令の分岐方向の偏りが減少し，近似レベル 16 から近似レベル 31 への変化では分岐方向の偏りが増加することに起因する．擬似乱数によって分岐方向を決定する場合，一般的な分岐予測器は分岐方

ⓒ 2020 Information Processing Society of Japan

向に偏りがあるほうが当てやすい．このため近似レベルの変化に対する分岐決定器なしとありの差は，最初は大きくなっていくが中央付近の近似レベルを境に小さくなっていく．

誤差の変化　近似レベルが大きくなると誤差が単調増加するという傾向はどのベンチマークにも共通している．また，近似レベル最小付近と最大付近では大きな変化がみられ，それ以外の部分ではおおむね直線的な傾向が見られるという傾向も共通している．出力画像を見るとどのベンチマークでも近似レベルが大きくなるほど劣化していることがわかる．しかし，品質の劣化度合いは異なっている．図 3（histeq）は近似レベルが大きくなってもそれほど劣化していない．一方，図 4（jpeg）は近似レベル 30 では画像に何が写っているか判別できないほど劣化している．

6. 考察

6.1 実行サイクル数の例外的な変化

近似レベルに対する実行サイクル数の変化の結果において，予想に反する結果となっている部分がある．それは図 3（histeq）の分岐決定器ありの場合の，近似レベル −1 から 0 へ変化する部分と，近似レベル 26 から 27 へ変化する部分である．近似レベル −1 から 0 へ変化する部分は，5 章で述べたように実行サイクル数は増加するはずであり，近似レベル 26 から 27 へ変化する部分は近似ループボディの選択確率の増加により実行サイクル数が減少するはずである．分岐決定器を使用する場合でも，確率的分岐命令の分岐方向を g-share 分岐予測器のグローバル分岐履歴に反映しているため，他の条件分岐命令の分岐予測結果の変化によるもんではないと考えている．実際，全体の実行を通しての条件分岐命令（確率的分岐命令は条件分岐命令として扱わない）の分岐決定器を使用する場合と使用しない場合の分岐予測ミス回数の差は 1 回以下である．現状ではキャッシュの状態変化やフェッチグループの変化が原因ではないかと考えている．

6.2 分岐決定器の必要性

図 3（histeq）の結果を見ると分岐決定器が実行サイクル数の削減に大きく寄与していることがわかる．ここで，分岐予測ミスが生じる原因が 32 ビット LFSR の擬似乱数による分岐方向決定であることに注目する．擬似乱数の周期が十分に短い場合，既存の分岐予測器によって十分学習可能なため，分岐決定器は不要に思える．

しかし，他の問題が生じる．擬似乱数で近似するかしないかを決める手法では，近似レベルの最大の段階数は擬似乱数の周期によって決まる．例えば周期が 32 の場合，32 回の確率程分岐命令のうち何回近似するかで近似レベルが決まるため，近似レベルの段階数は最大 32 に制限される．近似レベルの段階数はトレードオフ制御の粒度に依存する．分岐決定器を使用しない場合トレードオフ制御の粒度が分岐予測器が学習可能かどうかに依存してしまう．

また，仮に学習可能な擬似乱数系列であっても近似レベルが変更されるたびに学習する必要があり，実用的ではない．これは人間が手動で調整する場合はまり問題にならないかもしれない．しかし，自動で近似レベルを調整するような場合，高頻度で近似レベルが切り替わることにより分岐予測ミスが増大してしまう可能性がある．

7. おわりに

本論文では近似レベルを動的制御可能なアーキテクチャとそのアーキテクチャ上でのループ近似手法「Loop Body Switching」を提案し，評価をおこなった．4 つのベンチマークを用いた評価の結果，32 段階の近似レベル制御により誤差対実行サイクル数のトレードオフ制御が可能であることと，分岐決定器が Loop Body Swiching の効果を高めることが示された．今後は必要とされる出力品質に応じた近似レベルの制御方法の検討，FPGA 上での実装，近似レベルに対する消費電力変化の調査に取り組む．

謝辞　本研究の一部は，株式会社富士通研究所の支援を受けて実施されました．

参考文献

[1] Xu, Q., Mytkowicz, T. and Kim, N. S.: Approximate Computing: A Survey, *IEEE Design Test* (2016).
[2] Miguel, J. S. and Jerger, N. E.: The Anytime Automaton, *ACM/IEEE International Symposium on Computer Architecture* (2016).
[3] RISC-V-Foundation: RISC-V Foundation — Instruction Set Architecture(ISA), RISC-V Foundation (online), available from ⟨https://riscv.org⟩ (accessed 2020-01-27).
[4] Yeh, T., Faloutsos, P., Ercegovac, M., Patel, S. and Reinman, G.: The art of deception: Adaptive precision reduction for area efficient physics acceleration (2007).
[5] Thwaites, B., Pekhimenko, G., Esmaeilzadeh, H., Yazdanbakhsh, A., Park, J., Mururu, G., Mutlu, O. and Mowry, T.: Rollback-free value prediction with approximate loads (2014).
[6] Kahng, A. B. and Kang, S.: Accuracy-configurable adder for approximate arithmetic designs, *Proceedings of the 49th Annual Design Automation Conference* (2012).
[7] 佐藤裕貴，津村高範，津邑公暁：自動メモ化プロセッサにおける近似的入力一致比較手法，電子情報通信学会技術研究報告= IEICE technical report : 信学技報 (2015).
[8] Liu, Z., Yazdanbakhsh, A., Wang, D. K., Esmaeilzadeh, H. and Kim, N. S.: AxMemo: hardware-compiler co-design for approximate code memoization, *ACM/IEEE International Symposium on Computer Architecture* (2019).
[9] Sidiroglou-Douskos, S., Misailovic, S., Hoffmann, H. and Rinard, M.: Managing Performance vs. Accuracy Trade-Offs with Loop Perforation, *ACM SIGSOFT International Symposium on the Foundations of Software Engineering* (2011).
[10] Rodriguez-Cancio, M., Combemale, B. and Baudry, B.:

ⓒ 2020 Information Processing Society of Japan

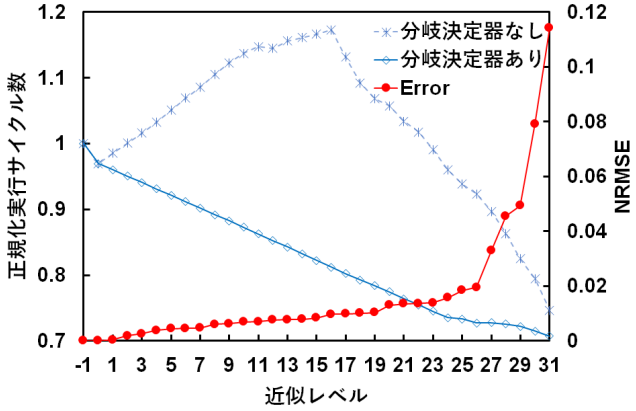


図 3: histeq

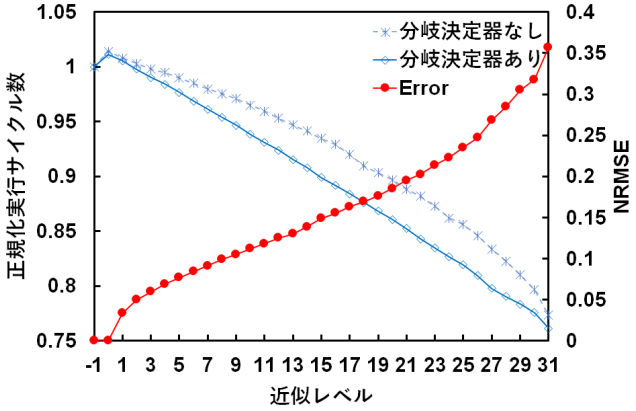


図 4: jpeg

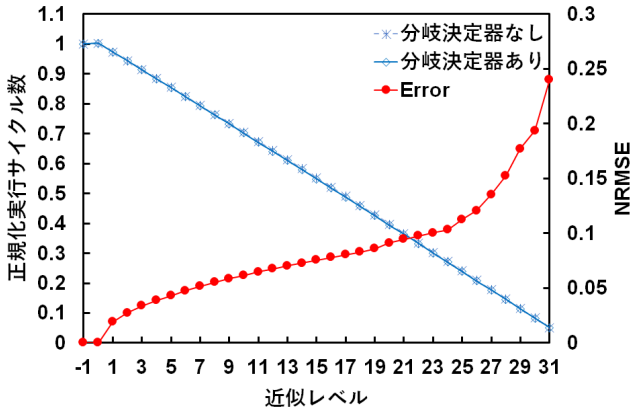


図 5: kmeans

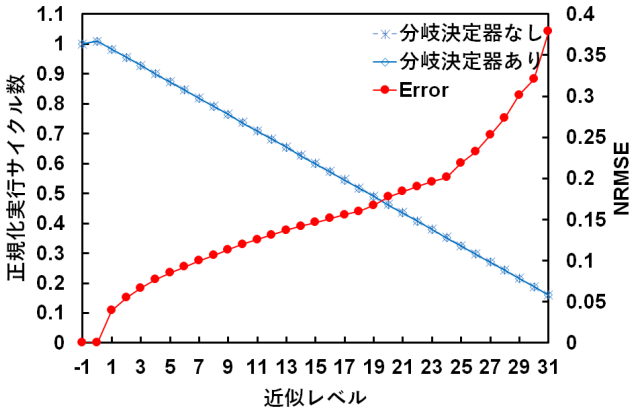


図 6: sobel

Approximate Loop Unrolling, *ACM International Conference on Computing Frontiers* (2019).

[11] Roy, P., Ray, R., Wang, C. and Wong, W. F.: Asac: Automatic sensitivity analysis for approximate computing, *SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems* (2014).

[12] Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L. and Grossman, D.: EnerJ: approximate data types for safe and general low-power computation (2011).

[13] Esmaeilzadeh, H., Sampson, A., Ceze, L. and Burger, D.: Neural Acceleration for General-Purpose Approximate Programs, *IEEE/ACM International Symposium on Microarchitecture* (2012).

[14] Samadi, M., Lee, J., Jamshidi, D., Hormati, A. and Mahlke, S.: SAGE: Self-tuning approximation for graphics engines, *IEEE/ACM International Symposium on Microarchitecture* (2013).

[15] Xu, S. and Schafer, B. C.: Toward Self-Tunable Approximate Computing, *IEEE Transactions on Very Large Scale Integration Systems* (2019).

[16] 塩谷亮太，五島正裕，坂井修一：プロセッサ・シミュレータ「鬼斬弐」の設計と実装，先進的計算基盤システムシンポジウム SACSIS2009 (2009).

[17] Barker, K., Benson, T., Campbell, D., Ediger, D., Gioiosa, R., Hoisie, A., Kerbyson, D., Manzano, J., Marquez, A., Song, L., Tallent, N. and Tumeo, A.: *PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual*, Pacific Northwest National Laboratory and Georgia Tech Research Institute (2013).

[18] Yazdanbakhsh, A., Mahajan, D., Esmaeilzadeh, H. and Lotfi-Kamran, P.: AxBench: A Multiplatform Benchmark Suite for Approximate Computing, *IEEE Design Test* (2017).

ⓒ 2020 Information Processing Society of Japan

Vol.2019-ARC-240 No.31
Vol.2019-SLDM-191 No.31
Vol.2019-EMB-53 No.31
2019/2/28

(a) ベースライン（近似レベル：-1）

(b) 近似レベル：6

(c) 近似レベル：12

(d) 近似レベル：18

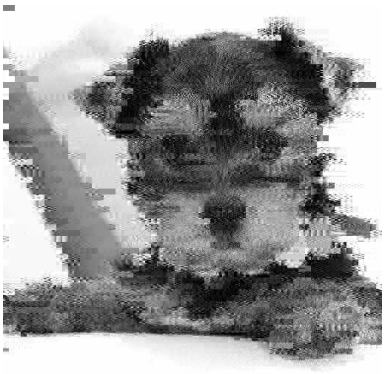
(e) 近似レベル：24

(f) 近似レベル：30

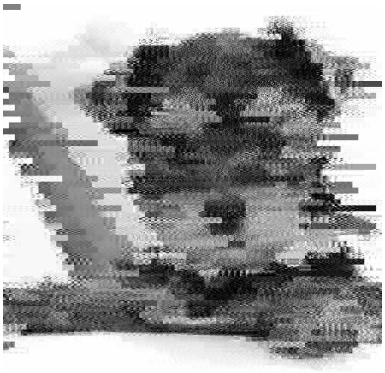
図 7: histeq の出力画像

(a) ベースライン（近似レベル：-1）

(b) 近似レベル：6



(c) 近似レベル：12



(d) 近似レベル：18

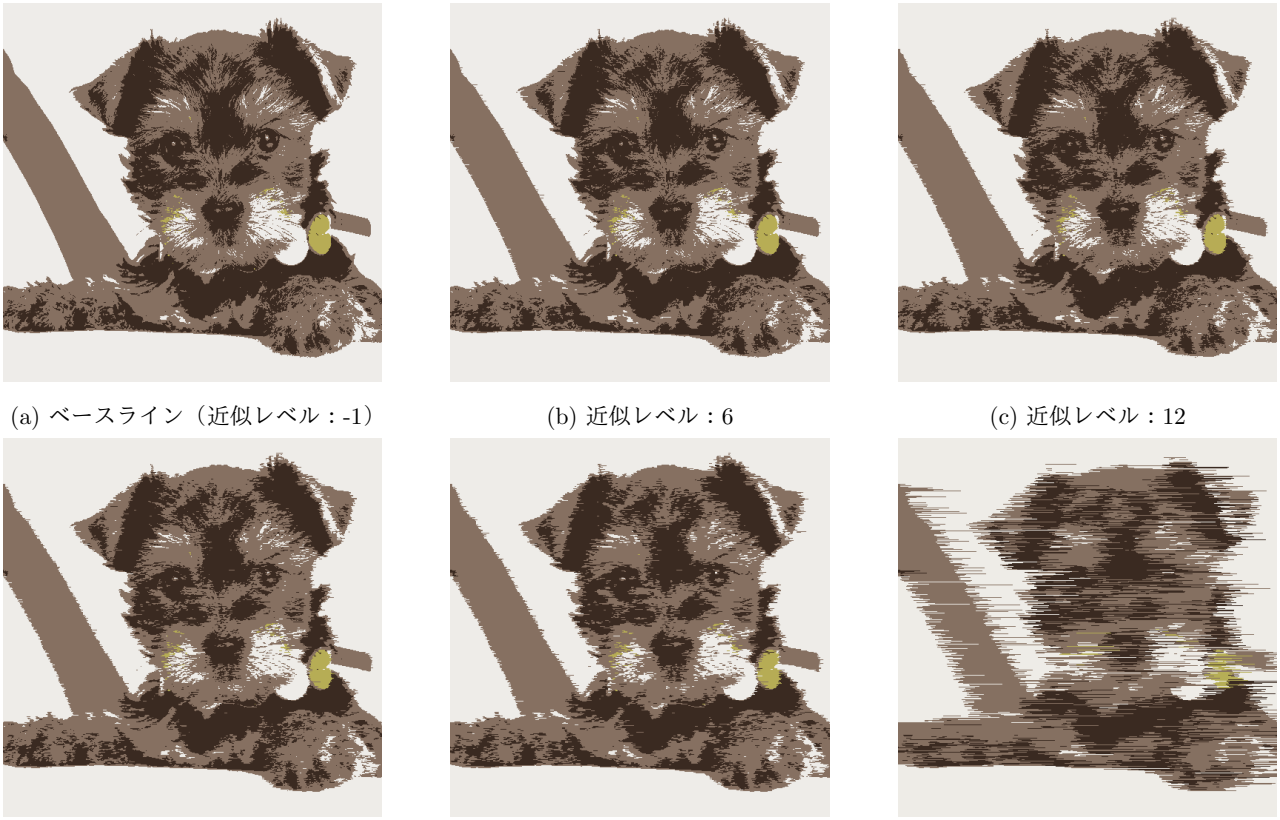
(e) 近似レベル：24

(f) 近似レベル：30

図 8: jpeg の出力画像

ⓒ 2020 Information Processing Society of Japan

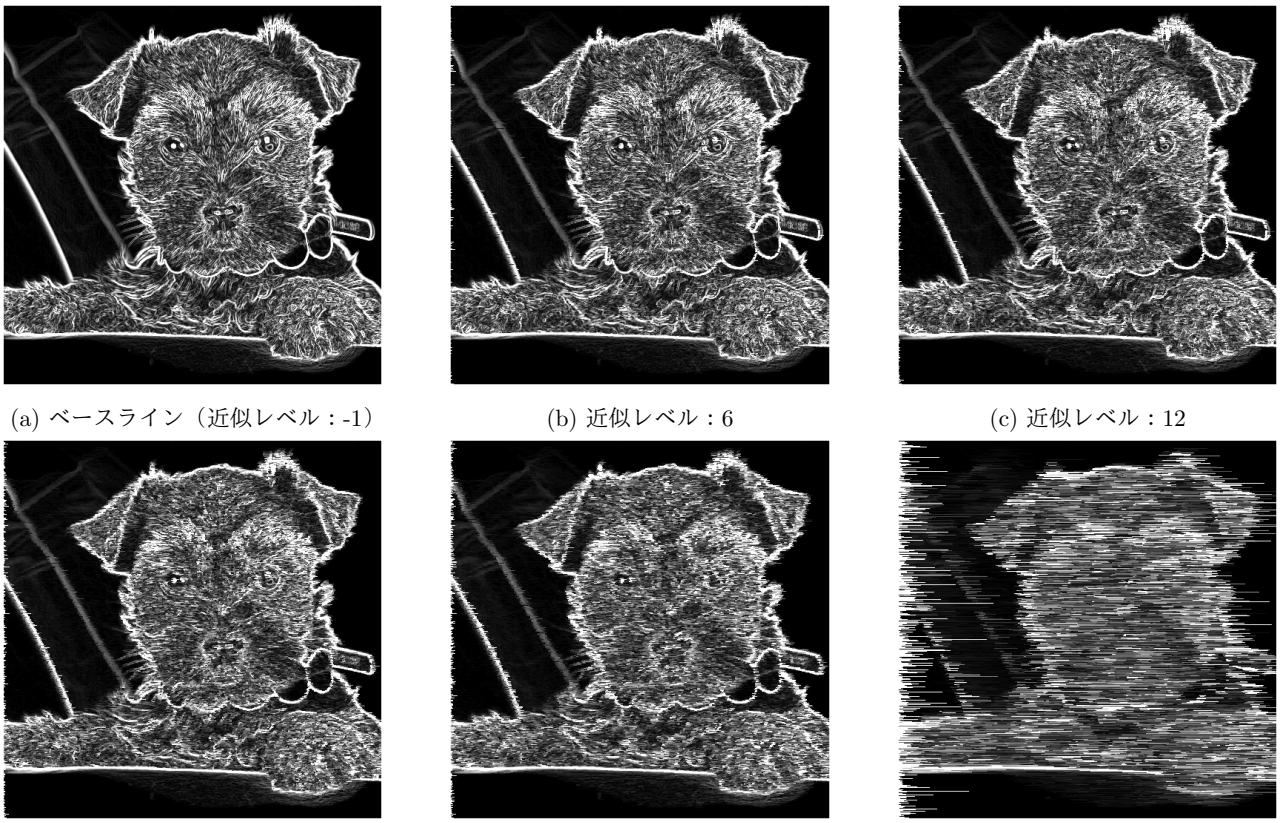
Vol.2019-ARC-240 No.31
Vol.2019-SLDM-191 No.31
Vol.2019-EMB-53 No.31
2019/2/28



(a) ベースライン（近似レベル：-1）　(b) 近似レベル：6　(c) 近似レベル：12

(d) 近似レベル：18　(e) 近似レベル：24　(f) 近似レベル：30

図 9: kmeans の出力画像



(a) ベースライン（近似レベル：-1）　(b) 近似レベル：6　(c) 近似レベル：12

(d) 近似レベル：18　(e) 近似レベル：24　(f) 近似レベル：30

図 10: sobel の出力画像

ⓒ 2020 Information Processing Society of Japan