Vol.2021-ARC-244 No.45
Vol.2021-SLDM-194 No.45
Vol.2021-EMB-56 No.45
2021/3/26

ハードウェア抽象化記述SHIMにおけるLLVM命令実行時間計測手法

井ノ川 誠[1,a)]　枝廣 正人[1]

概要：組込みシステム開発において，開発初期の実機が存在しない状況等でも実機の特徴を考慮した見積を行うために，SHIM 性能見積手法が研究されている．SHIM を作成する際に行う SHIM 性能値計測に関して，本研究では，既存の回帰分析を用いた手法に改善を施し，新しい定式化の提案を行った．また，本手法によって得られた SHIM 性能値を用いて見積精度の評価を行い，実施した改善の効果を考察するとともに，本手法の妥当性を検証した．

1. はじめに

近年，組込みシステムの大規模化，複雑化により，消費電力などの厳しい制約下では，シングルコアプロセッサにおける性能限界が問題となっている．そこで，マルチ・メニーコアプロセッサによる性能向上が期待されている．また，制御設計の効率化を図るために，制御を抽象的な「モデル」を用いて表し，モデル上で設計を行うモデルベース開発の研究が進んでいる．

モデルベース開発を用いたマルチ・メニーコアプロセッサへの並列制御設計の手法の 1 つに，モデルから生成された逐次コードを並列化する，コードベースの並列化手法がある [1]．この手法では，モデルの設計時に並列性を考慮していないため，効果的な並列化が難しいという課題がある．この課題への解決策として，我々は，モデルから並列性の高いソフトウェアを開発するモデルベース並列化ツール [2][3] の提案を行っている．

モデルベース並列化における性能見積では，実機や命令セットシミュレーター (ISS) を必要とせずに実機の特徴を考慮して見積ることが必要である．これは，組込みシステムでは，ソフトウェアとハードウェアの開発が同時に進められることがあり，実機が存在しない開発初期段階でもソフトウェア設計を行う場合があるためである．

そこで，ハードウェア抽象化記述である SHIM[4] を用いた性能見積である，SHIM 性能見積の研究が行われている [5][6]．SHIM とはハードウェアの特徴をソフトウェア向けに表現したデータ構造である．SHIM を利用した性能見積では，実機を用いずに実機の特徴を考慮した性能見積が可能となる．

SHIM にはプロセッサで実行可能な命令として，LLVM-IR 命令セットの命令が用いられている．SHIM が LLVM-IR 命令セットを採用している理由は，多種多様なプログラミング言語に対応することができる LLVM-IR を利用することで，SHIM の汎用性を高めるためである．

SHIM を作成する際に行う SHIM 性能値計測では，LLVM-IR の特徴を考慮する必要がある．例えば，LLVM-IR には 1 つの命令から複数種類のターゲットコードが生成されるという特徴がある．図 1 は LLVM-IR 命令の add 命令から，ルネサスエレクトロニクス社の RH850/E1M-S2[7] のアセンブラプログラムへの変換の例である．変換後のアセンブラプログラムには，レジスタ上のデータ加算を行うアセンブリ命令である add 命令に加え，場合に応じてオペランドレジスタの退避処理やメモリからの読込処理が出現している．このように，1 つの LLVM-IR 命令から異なるターゲットコードが生成されるため，正確な LLVM-IR のレイテンシを算出するためには，ターゲットコードごとの出現確率を考慮しなくてはならない．しかし，ハードウェアのアーキテクチャが複雑になるにつれて，この出現確率を考えることが困難になる．また計測には，対象の命令ごと，ハードウェアごとに計測プログラムを作成する必要があり，作業コストが大きいという課題がある．

この課題に対し鳥越ら [8] は，回帰分析を用いた計測手法の提案を行い，異なるハードウェアに対して同一の計測プログラムから，統計的に SHIM 性能値を計測することを検証した．しかし，この方法には回帰分析に制約条件を加えることが困難であることや，実機実行で発生する事象を

[1] 名古屋大学大学院情報学研究科
Nagoya University, Graduate School of Informatics
[a)] m_inokawa@ertl.jp

Vol.2021-ARC-244 No.45
Vol.2021-SLDM-194 No.45
Vol.2021-EMB-56 No.45
2021/3/26

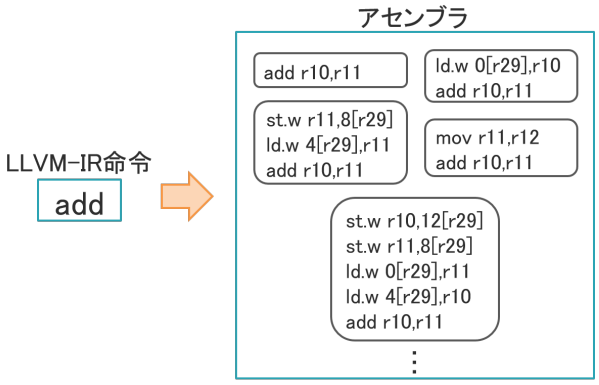

図 1　LLVM-IR 命令からアセンブラへの変換の例

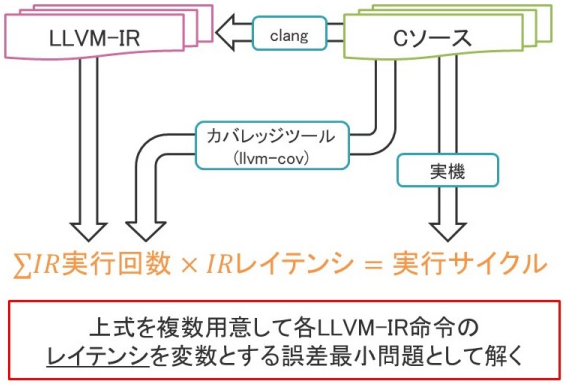

図 2　従来手法 [8] における計測の流れ

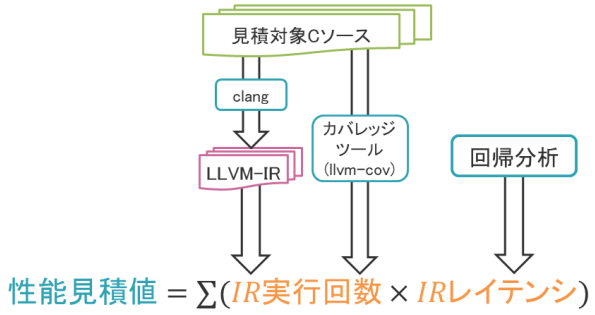

図 3　従来手法 [8] における性能見積方法

考慮してないことなどの課題がある．

本研究の目的は，SHIM の作成コストを小さくし，SHIM 性能見積の適用を容易にすることと，見積サイクル数の誤差率を SHIM の目標値である ±20%に収めることである．これらを達成するため，[8] で提案された回帰分析による SHIM 性能値計測手法の改善を実施し，新しい定式化の提案を行った．

2. 従来手法

2.1 概要

SHIM 性能値計測では，LLVM-IR 命令のレイテンシ (実行サイクル数) を求めることが重要である．鳥越ら [8] は，回帰分析によって統計的に SHIM 性能値を計測する手法を提案した．この手法により，同一の計測プログラムセットを用いて異なるハードウェアの計測を行うことが可能となった．同研究では，対象ハードウェアを Raspberry Pi3 Model B+[9] としている．

図 2 に計測の流れを示す．まず，C 言語の計測プログラム (C ソース) を複数作成する．次に，この計測プログラムから 2 種類の計測値を取得する．LLVM-IR の各命令の実行回数と，実機での実行サイクル数である．この 2 種類の計測値と求めたい LLVM-IR 命令のレイテンシを用いて式 (1) のように表す．このような式を計測プログラムの数だけ用意し，レイテンシを変数とする回帰分析を実施することで，LLVM-IR の各命令のレイテンシを予測する．

$$\sum_i (\text{IR 命令}_i\text{実行回数} \times \text{IR レイテンシ}_i) = \text{実行サイクル数} \quad (1)$$

性能見積をする際には，図 3 のように，見積対象となる C 言語プログラムの LLVM-IR 命令実行回数と，回帰分析によって求めた命令のレイテンシから性能見積値を算出する．

表 1　命令の種類による分類

命令変数	命令		
arithmetic	add	sub	mul
	sdiv	srem	urem
float	fadd	fsub	fmul
	fdiv	fcmp	
load	load		
store	store		
others	その他		

2.2 準備

この手法を適用するにあたり，いくつかの準備が必要である．

まず，計測プログラムの用意である．効果的な回帰分析を実施するためには十分な数の計測プログラムを用意しなくてはならない．また，処理の傾向や出現する LLVM-IR 命令が偏らないようにすることに注意をしなくてはならない．[8] では 25 種類の C 言語プログラムを実装し，計測プログラムとして利用している．

次に，変数の設定である．同手法では LLVM-IR 命令のレイテンシを回帰分析の変数に割り当てるが，単純に 1 対 1 で割り当てると変数の数が多くなりすぎてしまい，必要な計測プログラムの数が多くなってしまう．しかし，大まかすぎる割り当ては精度の低下を招いてしまう．したがって変数への命令の割り当てについては，変数の数が増えすぎないようにしながらも，同じ変数内でのレイテンシの差

Vol.2021-ARC-244 No.45
Vol.2021-SLDM-194 No.45
Vol.2021-EMB-56 No.45
2021/3/26

ができるだけ小さくなるようにしなくてはならない．同研究では命令を詳細に分類した方が回帰分析の見積精度向上を実現できると示されている．これに加えて，命令分類を詳細にするにつれて，見積精度の確保に必要な計測プログラムの数が増えることも指摘している．

[8] における命令の分類方法では，LLVM-IR 命令が行っている処理に着目し，処理が似ている命令ごとに分類をしている．LLVM-IR 命令の処理については，LLVM Language Reference Manual[11] を参考にしている．変数と命令の分類は表 1 の通りである．

また，回帰分析の方法については以下の通りである．ここでは回帰分析の中でも，式 (2) のような多変量の最小 2 乗法を利用している．

$$\boldsymbol{b} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y} \tag{2}$$

式 (2) の $\boldsymbol{X}$ は各計測プログラムの LLVM-IR 命令実行回数の行列，$\boldsymbol{Y}$ は各計測プログラムの実機での実行サイクル数の行列，$\boldsymbol{b}$ は求める各命令のレイテンシの行列を表している．

2.3 評価

計測プログラム 25 種類を用いて同手法を適用し得られた LLVM-IR 命令のレイテンシは表 2 の通りである．レイテンシの単位はクロックサイクルである．変数 arithmetic と float のレイテンシが負の値となった．

評価のために別にプログラムを用意し，結果を評価した．評価用プログラムは 5 種類あり，その概要を表 3 に示す．なお，CoreMark.c[10] を除く 4 種は，筆者の自作である．

表 2 の結果から式 (3) と式 (4) を用いて評価用プログラムの誤差率を算出し，精度評価を実施した．誤差率は図 4 のようになった．グラフの縦軸の単位は%で，横軸の数字は表 3 の番号と対応している．図 4 より，評価用プログラム 5 種類のうち 2 種類で誤差率が ±20%以内に収まっていることが分かる．しかし，ここで算出している見積サイクル数には，負の値になったレイテンシの結果も含んでいることを考慮しなくてはならない．レイテンシが負の値にならないことは明らかであるので，現実的な見積結果であるとは言い難い．

$$\text{見積サイクル数} = \sum_i (\text{IR 命令}_i\text{実行回数} \times \text{IR レイテンシ}_i) \tag{3}$$

$$\text{誤差率 (\%)} = \frac{\text{実機実行サイクル数} - \text{見積サイクル数}}{\text{実機実行サイクル数}} \times 100(\%) \tag{4}$$

表 2　従来手法 [8] における回帰分析の結果

命令変数	レイテンシ
arithmetic	−2.31
float	−0.91
load	2.39
store	4.20
others	1.57

表 3　5 種類の評価用プログラム

番号	プログラム名	概要
1	CoreMark.c	組込みシステム向けベンチマークEEMBCの1つ
2	Selectionsort.c	要素数100の整数型配列に対する選択ソート
3	Functions3.c	再帰関数の呼び出しとループ処理を行う
4	Functions4.c	再帰関数で整数演算と2数の大小比較を行う
5	Functions11.c	再帰関数で除算を含む浮動小数点演算と2数の大小比較を行う

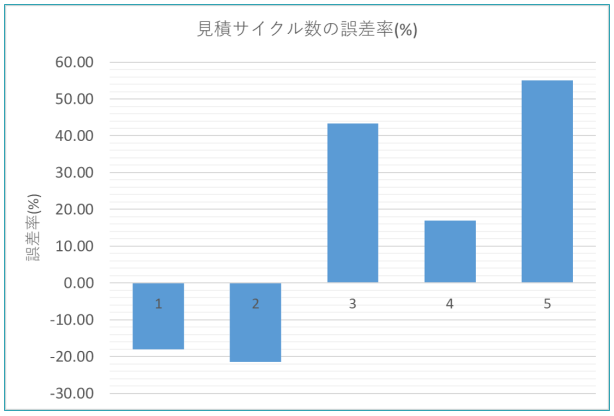

図 4　従来手法 [8] における見積サイクル数の誤差率

2.4 課題点

従来手法の課題点として，以下のようなことが挙げられる．

- 回帰分析の結果レイテンシが負の値になってしまい，現実的な結果とは言い難い
- キャッシュやメモリアクセス，実機計測時にかかるオーバヘッドなどの実機実行で発生する事象について考慮していない
- LLVM-IR 命令をより詳細な分類にする必要性
- 命令分類に応じた十分な計測プログラムの用意

3. 提案手法

本章では，従来手法 [8] の回帰分析を用いた SHIM 性能値計測手法の, 実機計測，シミュレーション，回帰分析の各工程に対して，それぞれ改善提案を行う．また，計測プログラムについては既存の 25 種類に加えて，新たなプログラムを用意した．本研究では，表 4 のように改善の実施順に 8 段階に分けた．本章では実施した改善を段階ごとに紹介する．

Vol.2021-ARC-244 No.45
Vol.2021-SLDM-194 No.45
Vol.2021-EMB-56 No.45
2021/3/26

表 4　実施した改善の概要

段階	実施した改善
0	先行研究の段階
1	キャッシュとメモリに関わる変数の追加
2	PMUの値取得に関する調査
3	外れ値の除外
4	実行サイクル数の下限の調整
5	オペランド依存関係を持つ命令の分類と変数othersの再設定
6	div命令の分離
7	計測オーバヘッドに関わる変数の追加
8	新しいプログラムの追加

3.1　段階 1 キャッシュとメモリアクセスに関わる変数

LLVM-IR では無限の仮想レジスタが存在することを想定している．しかし，実機においてはレジスタ数が有限であるため，レジスタスピルによってレジスタに配置されたデータが無くなってしまうということが起こり得る．その場合，LLVM-IR では必要がなかったメモリアクセスが発生する．

本論文では，このような LLVM-IR 上でのシミュレーションでは計測できないキャッシュアクセスやメモリアクセスなどの実機上で起こる事象を考慮することを提案する．具体的には，回帰分析で値を予測する変数 (レイテンシ) として，キャッシュのアクセスレイテンシとメモリアクセスレイテンシ (読込と書込をそれぞれ)，パイプラインハザードのペナルティを追加することにした．また，性能見積のときには，レジスタスピル率やキャッシュミス率を外から与え，見積もることを考える．

3.2　段階 2 PMU の値取得に関する調査

キャッシュやメモリについて回帰分析内で考慮することとなったため，計測プログラムを実機実行する際にキャッシュミス率などの取得を行う必要がある．そこで，Raspberry Pi3 の Cortex-A53[12] プロセッサに備わっている PMU(Performance Monitor Unit) を利用して，キャッシュのアクセス回数とリフィル回数を取得する関数を用意し，計測プログラムに組込んだ．この関数では，PMU を利用して各イベントカウンタの値を取得し，その値をあらかじめ用意していた配列に格納するという処理を行うことになっている．ここでは，この関数を実行する際に必要なキャッシュアクセス回数やリフィル回数を調査した．

調査の結果，PMU の値取得を行う関数を 1 回実行すると，キャッシュアクセスが約 200 回，キャッシュリフィルが約 2 回発生することが分かった．この調査結果を用いて，キャッシュミス率の計算式は式 (5) のようになった．

$$\text{L1 キャッシュミス率} = \frac{\text{L1 キャッシュリフィル回数} - 2}{\text{L1 キャッシュアクセス回数} - 200} \tag{5}$$

3.3　段階 3 外れ値の除外

実機上での計測では，計測プログラムを 100 回反復実行して計測値の取得を行う．また，回帰分析へはその 100 回の計測値の 1 回あたりの平均値を利用している．その 100 回の反復実行の中で，キャッシュアクセス回数やリフィル回数に外れ値が生じることがある．ここでは，この外れ値の除外について実施した．

外れ値を除外するためには，まずどの値が外れ値であるかを考える必要がある．今回は，100 個ある計測値の中央値を基準とすることにした．ここで平均値を用いなかったのは，外れ値の影響によって平均値が大きくなったり小さくなったりする可能性があり，本来利用したい計測値を効率よく抽出することが難しいと考えたためである．中央値を基準に，どれだけ離れた値までをその後の計算に用いるかの許容範囲を設定し，その許容範囲の外側にある計測値を外れ値として除外することにした．

3.4　段階 4 実行サイクル数の下限の調整

実機における計測では，予想していないオーバヘッドがかかることがある．実行サイクル数が小さい計測プログラムでは，このオーバヘッドの影響が大きくなってしまうと考えられる．

そこで，実行サイクル数の下限を調整し，実行サイクル数が小さくなりすぎないようにした．ここでは下限を 10000 サイクルに設定した．実行サイクル数が 10000 未満の計測プログラムに対して，main 関数内での処理を for 文などでループさせることで，実行サイクル数を下限以上にした．なお，このループ回数は計測プログラムによって 10 回，100 回，1000 回のいずれかにした．

3.5　段階 5 オペランド依存関係を持つ命令の分類と変数 others の再設定

まず，オペランド依存関係を持つ命令の分類について紹介を行う．本研究における，”ある命令 A がオペランド依存関係を持つ”とは，LLVM-IR において，命令 A の 1 つ前の命令の結果のレジスタを命令 A がオペランドに使用しているということを意味している．この場合，命令 A は 1 つ前の命令の結果を待たないと実行ができないため，命令間に依存関係があるといえる．

このように，同じ命令でも依存関係を持つ命令と持たない命令に分類することができる．本研究では，変数 arithmetic と float と load に対して，依存関係の有無で命令を分類し，それぞれを回帰分析の変数に割り当てた．

同時に，段階 1 で追加したパイプラインハザードの変数

Vol.2021-ARC-244 No.45
Vol.2021-SLDM-194 No.45
Vol.2021-EMB-56 No.45
2021/3/26

を削除した．これは，フォワーディングの影響で，オペランド依存関係を持つ全ての命令でパイプラインハザードが発生するとは限らないと考えたためである．このため，オペランド依存関係を持つ命令を回帰分析の変数として設け，パイプラインハザードのペナルティを含んだ，命令実行の際のレイテンシをより詳細に予測することにした．

次に，変数 others についてである．変数 others 内でも，なるべく処理量が似た命令をまとめて割り当てられるように，新しい変数 others1 と others2 を追加した．具体的には，変数 others1 へは sext 命令や sitofp 命令などのオペランドの型変換を行う命令を割り当て，変数 others2 へはそれ以外のどこにも属さない getelementptr 命令や alloca 命令などの命令を割り当てることにした．

3.6 段階 6 div 命令の分離

ここでは除算演算を行う sdiv 命令を分離させ，独立した変数に割り当てることを行った．段階 6 の前までは，sdiv 命令は変数 arithmetic に割り当てていた．

sdiv 命令に加え，浮動小数点演算の除算を行う fdiv 命令も変数 float から分離させた．また，依存関係を持つ命令の変数として，変数 div_d と fdiv_d も新たに追加した．

3.7 段階 7 計測オーバヘッドに関わる変数の追加

より実機実行を意識した回帰式を立てるために，ここでは計測オーバヘッドのレイテンシも回帰分析で予測することにした．つまり，計測オーバヘッドのレイテンシを新たな変数として回帰式に組込むということである．計測オーバヘッドの発生回数は 1 回であると定義した．

3.8 段階 8 新しいプログラムの追加

回帰式の変数が増えたことに伴い，計測プログラムの数を増やす必要性が高まった．そこで，新たな計測プログラムの作成を行った．

今回は新たに 11 種類の計測プログラムを作成した．プログラミング言語は従来手法と同様に C 言語である．その内訳は，ソートアルゴリズムを実行するプログラムが 2 種類，関数呼び出しをしながら数値計算を行うプログラムが 9 種類である．

3.9 回帰分析の計算方法

8 段階の改善とは別に，回帰分析の計算方法の変更を行った．

[8] では，回帰分析の計算を式 (2) のように，行列表現を用いた多変量の最小 2 乗法を用いて行っていた．しかし，この手法では制約条件の設定が困難であるという課題があった．そのため従来手法では，回帰分析の結果，命令のレイテンシが負の値になってしまっていた．

これらの課題を解決するために，本研究では，制約条件

表 5 提案手法の命令分類

命令変数	命令				
arithmetic	add	sub	mul	srem	urem
div	sdiv				
arithmetic_d	add_d	sub_d	mul_d	srem_d	urem_d
div_d	sdiv_d				
float	fadd	fsub	fmul	fcmp	
fdiv	fdiv				
float_d	fadd_d	fsub_d	fmul_d	fcmp_d	
fdiv_d	fdiv_d				
load	load				
load_d	load_d				
store	store				
callret	call	ret			
others1	sext	zext	sitofp	fptosi	
others2	getelementptr	alloca	phi	br	icmp

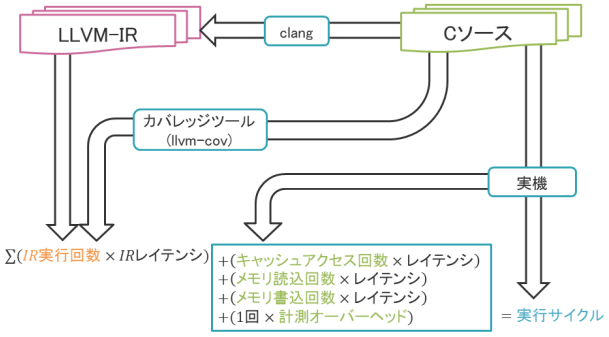

図 5 提案手法による計測の流れ

つき誤差最小 2 乗法を利用することにした．ここでの誤差というのは，実機上での実行サイクル数と見積サイクル数の差のことを指している．計測プログラムごとの誤差の 2 乗を算出し，それぞれを足すことで，誤差の 2 乗和を求める．この誤差の 2 乗和が最小となるレイテンシの組み合わせを回帰分析で予測する．今回の制約条件は以下の 2 点である．

- レイテンシの値が非負である
- 見積サイクル数の誤差率が ±20%以内である

3.10 命令分類と回帰式

8 段階の改善を実施し，LLVM-IR 命令の分類と計測の流れは，それぞれ表 5 と図 5 のようになった．

命令分類については段階 5 と段階 6 の改善により，[8] よりも詳細な分類となり，処理の似た命令が各変数に振り分けられた．

回帰式については，段階 1 と段階 7 の改善により，キャッシュアクセスやメモリアクセス，計測オーバヘッドなどの，実機実行で発生する事象を考慮した形式となった．

3.11 性能見積

提案手法における性能見積方法は，図 6 の通りである．従来手法と比較して，対象 C 言語プログラムの LLVM-IR 命令に関する部分は変わっていないが，レジスタスピル率やキャッシュミス率を外部から与える部分が新しく追加さ

Vol.2021-ARC-244 No.45
Vol.2021-SLDM-194 No.45
Vol.2021-EMB-56 No.45
2021/3/26

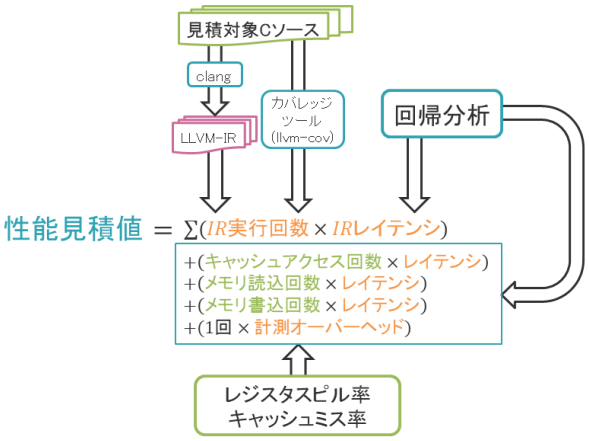

図 6　提案手法における性能見積方法

れた．

本研究では，Raspberry Pi3 Model B+という既存の実機を用いたため，レジスタスピル率やキャッシュミス率を実際に取得することができ，性能見積に組込むことができた．しかし，SHIM 性能見積は実機が完成前などで存在していない状況でも行えることを想定している．このような場合には，今回のように実機上で計測することができないため，何らかの形でこれらの値を与える必要である．その際には，完成前の実機にアーキテクチャが似ている既存の実機における計測値を用いたり，完成前の実機の性能を想定して算出した値を用いたりするといった方法が考えられる．

4. 見積精度評価実験

4.1 概要

前節で紹介した 8 段階の改善の各段階における回帰分析で求まったレイテンシと，そのレイテンシを用いて算出した評価用プログラムと計測プログラムの見積サイクル数の誤差率の評価をそれぞれ実施した．評価用プログラムとは，[8] の評価でも用いた 5 種類のプログラム (表 3) のことであり，計測プログラムとは，回帰分析の対象となったプログラムのことである (段階 7 までは 25 種類，段階 8 では 36 種類)．なお，[8] の結果は改善の 0 段階目であるとする．また，回帰分析の計算方法については，段階 0 では式 (2)，段階 1 から段階 8 では 3.9 節で説明した手法を用いた．

4.2 評価用プログラムの誤差率

図 7 と図 8 は段階ごとの評価用プログラムの誤差率の変化を示したグラフである．縦軸は見積サイクル数の誤差率であり単位は%である．横軸は改善の段階を表している．図 7 は，評価用プログラムの 1 種類ごとの誤差率の変化である．見やすさを考慮して，グラフの範囲は-100%から 100%とした．図 8 は誤差率の絶対値に関するグラフである．グラフの範囲は 0%から 200%とした．橙色のプロット

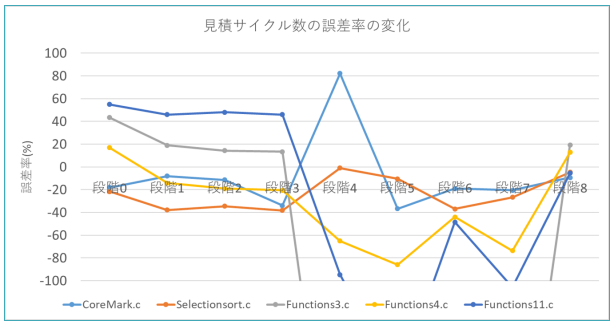

図 7　評価用プログラムごとの見積サイクル数の誤差率の変化

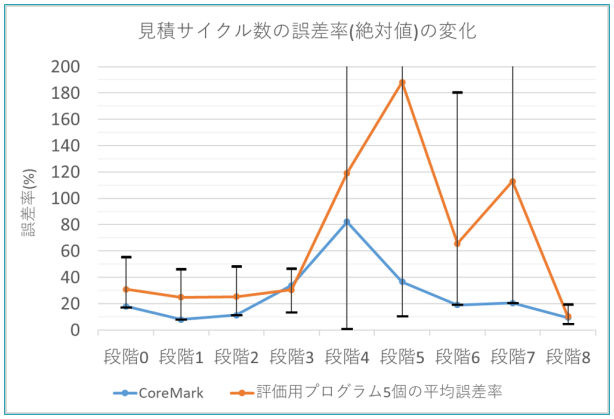

図 8　評価用プログラムの見積サイクル数の誤差率 (絶対値) の変化

は評価用プログラム 5 種類の誤差率の絶対値の平均値である．青色のプロットは CoreMark の誤差率絶対値を表している．また，評価用プログラム 5 種の誤差率 (絶対値) の最大値と最小値を表示し，それらの差を表現した．CoreMark のみを抽出して評価した理由は，CoreMark は組込みシステムのベンチマークとして著名のものであり，評価用プログラムの中で最も実用的なプログラムであると言えるからである．

図 7 と図 8 から，段階 0 から段階 3 までは誤差率の変動が小さく，段階 4 から段階 7 では大きな変動が見られ，段階 8 で ±20%以内に収まったということが分かった．

段階 4 から段階 7 では，段階 3 以前では 0 であったメモリアクセスのレイテンシが大きな値 (約 400 サイクル～約 5000 サイクル) に予測された．評価用プログラム 5 種のうち，CoreMark を除く 4 種類のプログラムは実機における実行サイクル数が CoreMark のそれと比べてとても小さい (約 1 万サイクル～約 16 万サイクル，CoreMark は約 1600 億サイクル)．そのため，4 種類のプログラムでは見積サイクル数に占めるメモリアクセスのサイクルの割合が大きくなってしまい，誤差率の変動に繋がってしまったと考えられる．

4.3 計測プログラムの誤差率

図 9 と図 10 は段階ごとの計測プログラムの見積サイクル数の誤差率の変化を箱ひげ図で表したものである．縦軸

Vol.2021-ARC-244 No.45
Vol.2021-SLDM-194 No.45
Vol.2021-EMB-56 No.45
2021/3/26

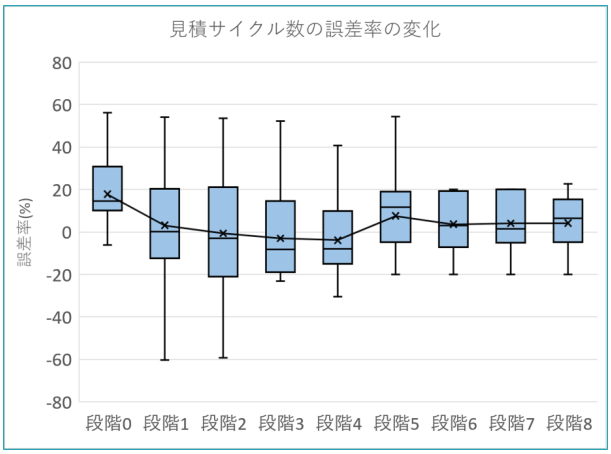

図 9 計測プログラムの見積サイクル数の誤差率の変化

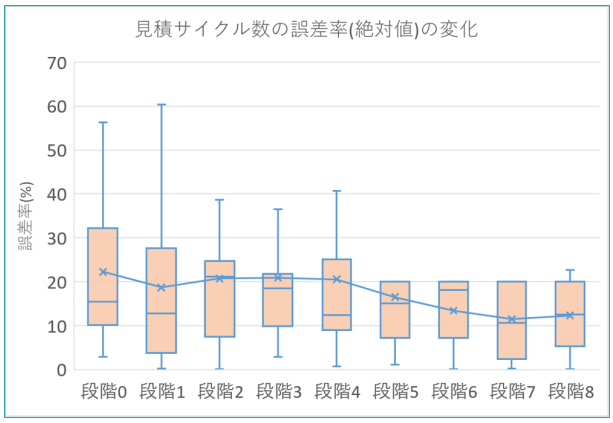

図 10 計測プログラムの見積サイクル数の誤差率 (絶対値) の変化

は見積サイクル数の誤差率であり単位は%である．横軸は改善の段階を表している．図 9 は実際の誤差率を用いたものであり，図 10 は誤差率の絶対値を用いたものである．箱ひげ図の×印は各段階の誤差率の平均値を表しており，折れ線グラフは平均値の変化を示している．両者とも，段階 0 から段階 7 までは計測プログラムが 25 種類で，段階 8 では 36 種類である．

図 9 と図 10 から，段階が進むごとに誤差率のばらつきが小さくなるとともに，誤差率の平均値や最大値も小さくなる傾向があることが分かる．このことから，回帰分析を行う上では，表 5 の命令分類と図 5 の回帰式が現時点で最良であるといえる．

しかし，段階 4 から段階 7 における評価用プログラムの誤差率の変化を考慮すると，計測プログラムの誤差率が向上したときに，必ずしも評価用プログラムの誤差率が向上するわけではなかった．これは，計測プログラムと評価用プログラムの命令実行回数やキャッシュ，メモリアクセス回数の傾向が異なっていることが理由であると考えられる．段階 8 では，評価用プログラムの自作の 4 種類 (ソートプログラムと数値計算プログラム) と似た処理を行うプログラムを計測プログラムに追加した．その結果，評価用プログラムの見積サイクル数の誤差率は向上した．以上のことから，様々な処理を行うプログラムを小さい誤差率で見積るためには，計測プログラムにも様々な種類のプログラムがあると良いと考えられる．しかし，様々なプログラムを用意し，計測プログラムとして用いることには難しさがある．

5. おわりに

5.1 まとめ

本研究では，モデルベース並列化などで用いる SHIM の作成手法として，従来手法 [8] の回帰分析を利用する手法の課題点を考え，解決策の提案を行った．解決策として，同手法の実機計測，シミュレーション，回帰分析の各工程に対する改善の提案と，新たな計測プログラムの実装を行った．また，Raspberry Pi3 Model B+をターゲットハードウェアとして，今回提案した改善を加えた手法を用いて実行サイクル数を見積もり，その見積精度の評価を実施した．

その結果，改善を進めていく段階で見積精度が悪化する場合があったが，提案した全ての改善を実施した際には見積精度が向上し，従来手法よりも現実的な見積に近づいたと言える．

5.2 今後の課題

回帰分析を用いた SHIM 性能値計測手法に関して，大きく 2 つの課題があると考えている．

1 つ目は様々な計測プログラムを追加しなくてはならないということである．見積精度評価実験において，評価用プログラムと似た傾向を持つプログラムを計測プログラムに追加した結果，そのプログラムの見積サイクル数の誤差率が向上するということが分かった．このことから，SHIM 性能見積において様々なプログラムを扱えるようにするためには，それに応じた様々な処理を行う計測プログラムが必要であると考えられる．このことを実現するため，現在の手法では対応できていない，計測プログラムのシステムコールやライブラリ関数への対応を行うことが望ましい．

2 つ目は異なるハードウェアに対して本手法を適用すべきだということである．この研究の大きな目的として，異なるハードウェアに対して容易に SHIM 性能値計測を行うことのできる手法を提案するということがある．しかしながら，現時点では Raspberry Pi3 Model B+のみをターゲットハードウェアとしている．そのため，他のハードウェアにおいても本手法を適用できるかを試し，さらなる改善を行う必要があると考えられる．

謝辞 本研究を進めるにあたりご議論いただいた三菱電機株式会社情報技術総合研究所の皆様に深く感謝いたします．

参考文献

[1] 梅田弾，金羽木洋平，見神広紀，林明宏，谷充弘，森裕司，木村啓二，笠原博徳．”MATLAB/Simulink で設計されたエンジン制御 C コードのマルチコア用自動並列化”，情報処理学会論文誌，Vol 55，No. 8，pp.1817-1829，2014.
[2] 山口滉平，竹松慎弥，池田良裕，李瑞徳，鍾兆前，近藤真己，枝廣正人．”Simulink モデルからのブロックレベル並列化”，組込みシステムシンポジウム 2015 論文集，Vol.2015，pp.123-124，2015.
[3] 生沼正博，山本椋太，竹内成樹，権藤正樹，本田晋也，近藤真己，枝廣正人．”モデルベース並列化ツールを用いたマルチコアシステム開発フローの提案”，情報処理学会 DA シンポジウム，pp.73-80，2020.
[4] M.Gondo，F.Arakawa，and M.Edahiro．”Establishing a standard interface between multi-manycore and software tools - SHIM”，COOL Chips XVII，VI-1，2014.
[5] 西村裕，中村陸，荒川文男，枝廣正人．”ソフトウェア向けハードウェア性能記述を用いたマルチコアにおける性能見積り”，ETNET2014，2014.
[6] 佐合惇，枝廣正人．”ハードウェア抽象化記述 SHIM と SHIMulator によるソフトウェア動的性能見積手法”，ETNET2019，2019.
[7] Renesas electronics RH850/E1M-S2，(参照 2021-01-22). https://www.renesas.com/jp/ja/products/microcontrollers-microprocessors/rh850/rh850e1x/rh850e1ms2.html.
[8] 鳥越敬，枝廣正人．”ハードウェア抽象化記述 SHIM による性能見積のための LLVM-IR 命令実行時間計測手法”，ETNET2020，2020.
[9] IODATA raspberry pi 3 model b+ (UD-RP3BP) 仕様，(参照 2021-01-22). https://www.iodata.jp/product/pc/raspberrypi/ud-rp3bp/spec.htm.
[10] CoreMark – EEMBC Embedded Microprocessor Benchmark Consortium, (参照 2021-01-26). https://www.eembc.org/coremark/.
[11] LLVM Language Reference Manual，(参照 2021-01-22). https://llvm.org/docs/LangRef.html.
[12] Cortex-A53 – Arm Developer，(参照 2021-01-22). https://developer.arm.com/ip-products/processors/cortex-a/cortex-a53.