Vol.2019-ARC-235 No.14
Vol.2019-SLDM-187 No.14
Vol.2019-EMB-50 No.14
2019/3/17

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

佐合　悖[1,a)]　枝廣　正人[1,b)]

概要：組込みシステムの大規模・複雑化によりモデルベースで並列度の高いソフトウェアを開発する手法の需要が高まっている．モデル設計時に各ブロックの実行サイクル数を見積り，それをもとにモデルを設計することで，高い並列度のソフトウェアを開発することが可能である．そこで，本研究では SHIM と呼ばれるハードウェアの特徴を記述するデータ構造を用いて，モデル設計段階でソフトウェアの性能を見積る手法を提案する．本手法では，SHIM を入力としてハードウェアの動的シミュレーションを行う SHIMulator を提案する．サンプルプログラムとエンジン制御モデルを用いた見積精度評価実験では誤差 20%以内を達成し，システム設計者が SHIM を用いて，モデル設計段階でモデルから生成した逐次コードの SHIM 性能見積を行うことで，実機や ISS が存在しない状況でもソフトウェアの実行サイクル数を簡単に予測できることが確認できた．

Software dynamic performance estimation with hardware abstract description SHIM and SHIMulator

1. はじめに

近年，組込みシステム開発では大規模・複雑化が進んでおり，効率的な開発プロセスとしてモデルベース開発の需要が高まっている．また，組込みシステムは時間や消費電力などの制約が厳しいことから，性能要求を満たすため組込みシステムのマルチコア化が進んでいる．そのような背景から，マルチ・メニーコアシステムのモデルベース開発手法が必要とされている．

モデルベース開発でマルチ・メニーコア向け並列ソフトウェア開発する手法として，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がある．しかし，コードベースの並列化では，生成される逐次コードは負荷分散など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存在する．コードベースの並列化手法の課題の解決策として，枝廣研究室ではモデルベース並列化 (MBP:Model-Base Parallelization) を提案している．モデルベース並列化では並列度の高いソフトウェアをモデルベース開発で開発するために，モデル設計段階で各コアの負荷分散を考慮し，並列性の高いモデルを設計する．また，モデルから逐次コードを生成するのではなく，並列コードを自動生成する．

モデルベース並列化において，並列性の高いモデルを設計するには，モデルの各ブロックの実行時間を把握し，コアごとの負荷を均等にすることが重要である．また，そのためにはモデル設計段階でソフトウェアの実行サイクル数を予測する性能見積手法が必要である．この性能見積手法には，システム開発の初期段階において，実機や命令セットシミュレータ (ISS) が存在しない状況でも，実機/ISS の特徴を反映させた性能見積が可能でなければならないという要件が存在する．

そこで，本論文ではハードウェア抽象化記述 SHIM[2] を利用したソフトウェア実行サイクル見積手法と，LLVM-IR の実行・解析を行うシミュレータである SHIMulator を利用した SHIMulator 動的性能見積フローを提案する．

評価では現存するプロセッサのサイクル精度の命令セットシミュレータを用いて実行サイクル数を計測し，性能見積結果と実行サイクル数の比較を行った．対象のプロセッサにはルネサス エレクトロニクス社の RH850/E1M-S2[1]

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

を用いた．評価実験では，サンプルプログラムとエンジン制御モデルの性能見積結果に対して精度評価を行い，誤差±20% 以内を達成した．

2. 準備

2.1 SHIM

SHIM[2] は Software-Hardware Interface for Multi-Many Core の略称であり，The Multicore Association[3] によって策定された，ハードウェアの特徴を XML 形式で記述する国際規格である．SHIM の特徴はハードウェアの詳細を厳密に記述するのではなく，ソフトウェア開発の視点から，ハードウェアの重要な情報のみを抽出していることである．SHIM の基本的な原則はアーキテクチャ設計レベルでソフトウェアに影響を与えるハードウェアの特徴を把握することである．つまり，SHIM で記述された特定のハードウェアに適したソフトウェアを設計すれば，その設計はシステム開発の後期段階でソフトウェアアーキテクチャの変更を必要としないはずである．

2.2 SHIM 性能見積の概要

SHIM 性能見積とは，ターゲットハードウェアから作成した SHIM(ターゲット SHIM) を用いて，ターゲットハードウェア上でソフトウェアを実行した際の性能を予測することである．また，本研究における SHIM 性能見積とは，対象のハードウェア上でソフトウェアを動作させた際の実行サイクル数を予測することを指す．SHIM にはターゲットハードウェアの性能情報を表す値として Performance オブジェクトが存在する．Performance はそれが関連付けられたオブジェクトごとに表す性能が異なり，SHIM1.0 では Performance は 3 つのオブジェクトに関連付けられている．以下に関連付けられたオブジェクトとその Performance が表す性能情報を示す．

- CommonInstructionSet, Instruction
  対象のプロセッサが命令を実行した際の実行サイクル数を表す．それぞれの命令はターゲットの命令セットの命令ではなく LLVM-命令セット IR の命令として記述されており，実行サイクル数の値はターゲットハードウェアで LLVM-IR 命令を実行した場合のサイクル数である．
- SubSpace, MasterSlaveBingings, Accessor, AccessType
  対象のプロセッサがメモリアクセスを行った際の実行サイクル数を表す．メモリアクセスは AccessType で区別され，AccessType はそのアクセスの識別 (読み/書き) や，アクセスするバイト数などの情報を持っている．
- CommunicationSet
  プロセッサ等の MasterComponent 間の通信にかかるプロセッサのサイクル数を表す．共有メモリや割込みのような通信の手段ごとに異なる性能情報を持っている．

CommonInstructionSet, Instruction に関連付けられた Performance がターゲットの命令セットの命令ではなく LLVM-IR 命令セットの命令で記述されている理由は，インターフェースとしての汎用性を持たせるためである．SHIM から命令の実行サイクル数を取得する場合，その命令の名前を用いて CommonInstructionSet，Instruction に関連付けられた Performance にアクセスする．仮にこの Performance がターゲットの命令セットの命令で記述されていたとすると，異なる命令セットを持つハードウェアの SHIM から命令の実行サイクル数を取得するために，ターゲットの命令セットを把握している必要がある．LLVM-IR の命令セットの命令で Performance を記述することで，命令セットが異なるターゲットの SHIM からも簡単に実行サイクル数を取得することが可能である．

SHIM の性能情報が LLVM-IR 命令で記述されているため，SHIM 性能見積では見積対象のソフトウェアは LLVM-IR で表現される．ソフトウェアを LLVM-IR で表現する最も簡単な方法は，C 言語などのプログラミング言語で記述されたソフトウェアを LLVM フロントエンドで LLVM-IR に変換することである．最も代表的な LLVM フロントエンドとして clang[4] がある．

SHIM 性能見積では LLVM-IR で表現されたソフトウェアを解析し，解析結果と SHIM の情報に基づいて実行サイクル数の見積を行う．解析の際に LLVM-IR を実際に実行する解析方法を動的解析，動的解析に基づく性能見積を動的性能見積と呼び，実行しない解析方法を静的解析，静的解析に基づく性能見積を静的性能見積と呼ぶ．

動的性能見積はメモリアクセスなど静的解析では取得できない情報を取得可能であるが，LLVM-IR を実行する環境が必要である．LLVM-IR の実行環境には，LLVM-IR をホストマシンのバイナリに変換して実行するものが存在するが，この実行環境で動的解析を行うには，LLVM-IR コードに解析用のコードを埋め込む必要があり，正確な解析が難しい．そこで，本研究では既存の LLVM-IR の実行環境には存在しない，ホストマシンのバイナリ依存せず LLVM-IR レベルでの実行を行うシミュレータである SHIMulator を開発した．この SHIMulator を用いて動的性能見積を行うフローを提案する．また，動的性能見積では見積対象のソフトウェアに対して適切なデータセットを用意する必要がある．そのため MATLAB/Simulink と SHIMulator 連携させる SHIMulator_PILS を提案する．

2.3 SHIM 性能見積の課題

SHIM 性能見積の最大の課題が SHIM の Performance 値の設定である．上記の通り，Performance は性能情報を表

Vol.2019-ARC-235 No.14
Vol.2019-SLDM-187 No.14
Vol.2019-EMB-50 No.14
2019/3/17

す値であり，各 Performance はそれぞれ Latency と Pitch という値を持っている．Latency は特定の命令の実行が完了するまでのプロセッサのサイクル数である．Pitch はより複雑な数値であり，連続して命令を実行した際の 1 つの命令当たりの実行サイクル数を表している．現代のハードウェアには多くの場合，次の操作を予測するメカニズムが搭載されている．例えばメモリアクセスの場合，ハードウェアはキャッシュとしてメモリの内容をより高速なメモリに読み込み，次のメモリアクセスを高速化するような動作を行う．このようにハードウェアにはソフトウェアの反復動作を効率化するためのメカニズムがあり，Pitch はこのような効率化が有効になっている場合の命令の実行サイクル数を表している．

また，Latency と Pitch は Best・Typical・Worst の値で構成されており，それぞれ最短・最頻・最悪の場合のサイクル数である．ここで問題となるのがこれらの値である．SHIM 作成者は Performance オブジェクトが表している処理がどのような場合に，最短，最頻，最悪の実行サイクルになるか理解している必要がある．そのため，SHIM 作成者はターゲットハードウェアの詳細なアーキテクチャを熟知していなければならず，ユーザーによる完全な SHIM の作成はかなり困難である．

そこで本研究では，最短の実行状況は最頻・最悪の実行状況よりも定義しやすいことに着目した．これはハードウェアは可能な限り多くの命令を最短の実行サイクルで実行するように設計されているという原則に基づいており，最短の実行状況は，最頻・最悪の実行状況と比較して前提となる条件が少ないためである．また，SHIM 性能見積手法で利用する Performance を基本的に Best 値のみに限定し，Typical や Worst の値を利用しないことで，Typical と Worst の値を計測する必要がなくなり，SHIM 作成を簡易化した．

2.4 既存研究

SHIM 性能見積の既存研究として 2014 年に西村ら [6] によるものと 2016 年に溝口裕哉 [7] によるものがある．これらの研究は SHIM 静的性能見積に関する研究であり，見積対象の LLVM-IR から実行ファイルを生成し，ホストマシン上で実行した結果をもとに LLVM-IR の解析と性能見積を行う．本論文で提案する SHIM 性能見積手法は，SHIM 動的性能見積手法であり位置づけが異なる．

3. SHIM 動的性能見積手法

本研究では見積対象のソフトウェアは C 言語で記述されており，基本的に main 関数の実行開始から終了までの区間を性能見積の対象としている．以下に示す見積手法は，命令 $Inst$ に対する見積サイクル数 $EstimationCycle(Inst)$ を算出する手法であり，ソフトウェア全体の見積サイクル数 $EstimationResult$ は式 (1) で示すように，実行したすべての命令に対して，$EstimationCycle(Inst)$ の総和を求めることで算出する．

$$EstimationResult = \sum_{Inst}^{All} EstimationCycle(Inst) \quad (1)$$

3.1 オペランド依存関係による見積

SHIM の Performance の Pitch と Latency の定義より，CommonInstructionSet, Instruction に関連付けられている Performance における Pitch は，ハザードによるストールが発生しない場合の命令の実行サイクル数であり，Latency はハザードによるストールが発生した場合の実行サイクル数であるとみなせる．そこで LLVM-IR 上の命令のオペランドのデータ依存関係に着目し，命令の実行サイクル数の見積に使用する Pitch と Latency の値を選択する．2 つの命令があり，1 つ目の命令で書き込みを行ったレジスタが，2 つ目の命令でオペランドとして利用されている場合に，2 つの命令間にオペランド依存関係があると定義する．図 1 に LLVM-IR コードの例を示す．この例では 1 行目の fmul 命令の結果を`%mul` という名前のレジスタに書き込んでいる．2 行目の fadd 命令ではレジスタ`%mul` をオペランドとして利用しており，1 行目と 2 行目の命令間にはオペランド依存関係が存在する．2 行目の命令の結果が書き込まれたレジスタ`%add` は，3 行目の命令でオペランドとして利用されておらず，2 行目と 3 行目の命令間にはオペランド依存関係は存在しない．

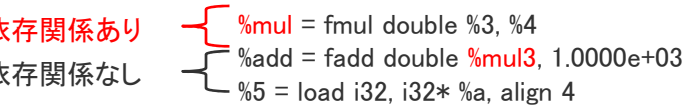


図 1 LLVM-IR 命令間のオペランド依存関係の有無

オペランド依存関係を利用した見積手法では，見積対象の命令とその直後の命令間にオペランド依存関係が存在するかどうかを調べ，オペランド依存関係が存在する場合には，見積対象の命令の見積サイクル数を CommonInstructionSet, Instruction に関連付けられている Performance の Latency の Best の値とする．オペランド依存関係が存在しない場合には，見積対象の命令の見積サイクル数を Pitch の Best の値とする．終端命令 (Terminator Instructions) と LLVM-IR 上でレジスタへの書き込みを行わない命令に関しては，オペランド依存関係を発生させないとし，常に Pitch の Best の値を見積サイクル数とする．

3.2 メモリ配置とアクセス性能による見積

メモリアクセスの実行サイクル数はアクセス先のメモリとアクセスするバイト数等で性能が変化する．そこでメモリアクセスを行う命令である LOAD 命令と STORE 命令の

Vol.2019-ARC-235 No.14
Vol.2019-SLDM-187 No.14
Vol.2019-EMB-50 No.14
2019/3/17

見積は他の命令と異なり，命令の実行サイクル数が記述されている CommonInstructionSet, Instruction に関連付けられた Performance ではなく，メモリアクセスの性能が記述されている SubSpace, MasterSlaveBingings, Accessor, AccessType に関連付けられた Performance の性能情報を利用する．

メモリ配置とアクセス性能を考慮した性能見積手法では，図 2 のように，LLVM-IR 上のメモリアクセス命令とそのオペランドの変数から section attribute とデータ型を読み取り，それに該当する SubSpace, MasterSlaveBingings, Accessor, AccessType に関連付けられた Performance の性能情報を取得する．さらに，オペランド依存関係を利用した性能見積手法と組み合わせ，取得した Performance の Latency と Pitch を選択する．

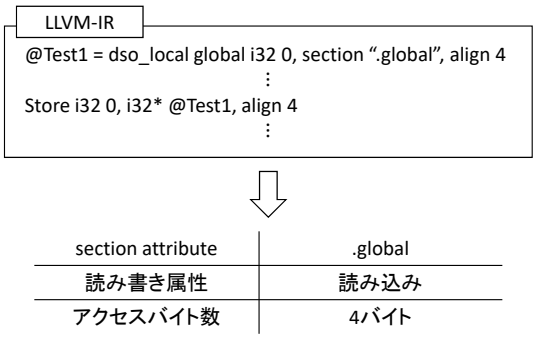


図 2 メモリ配置とアクセス性能を考慮した性能見積の LLVM-IR の解析

3.3 即値のレジスタ転送命令を考慮した見積

即値は機械語のバイナリコード中に直接記述される数値であり，実際の機械語の命令セットでは命令長やハードウェアの複雑さなどの要因から，即値オペランドを指定可能な命令は一部のみであることが多い．特に浮動小数点演算で即値オペランドが扱える命令はかなり少ない．LLVM-IR 命令セットは仮想的な命令セットであるため，前述の制約が存在せず，浮動小数点演算を含む多くの命令で即値オペランドを扱うことが可能である．そのため，LLVM-IR コード上である命令が即値オペランドを持っており，その LLVM-IR 命令に対応するアセンブリ命令が即値オペランドを扱えない場合，レジスタに即値を格納するための即値のレジスタ転送命令が暗黙的に生成される．例として図 3 に LLVM-IR 命令と，それに対応する RH850/E1M-S2 のアセンブリ命令を示す．浮動小数点型の加算を行う LLVM-IR 命令 fadd は即値オペランドを扱うことが可能だが，RH850/E1M-S2 の倍精度浮動小数点型の加算を行う addf.d 命令は即値オペランドを扱うことができない．そのため，アセンブリコード上では，即値をレジスタに格納するための mov 命令が生成される．

即値オペランドを持つ LLVM-IR 命令の実行サイクル

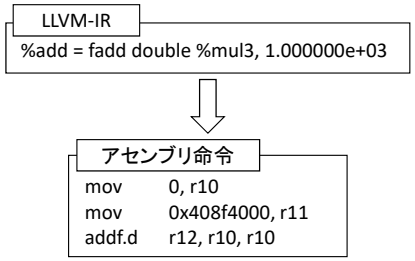


図 3 即値オペランドを扱うことが可能な LLVM-IR 命令とその命令に対応する RH850/E1M-S2 のアセンブリ命令

数の見積値に，即値のレジスタ転送命令の実行サイクル数を加算することで，即値のレジスタ転送命令を考慮した性能見積を行う．そのために SHIM に即値のレジスタ転送命令の実行サイクル数を記載する必要がある．しかし，LLVM-IR 命令セットには即値のレジスタ転送命令が存在しないため，他の命令のように CommonInstructionSet, Instruction に関連付けられた Performance が存在しない．新たに即値のレジスタ転送命令を定義し，Performance を記載しても良いが，即値のレジスタ転送命令はレジスタに転送するデータのビット幅によって実行サイクル数が異なる．そのため，アクセスタイプとしてビット幅を指定可能な SubSpace, MasterSlaveBingings, Accessor, AccessType に関連付けられた Performance を利用して，即値のレジスタ転送命令の実行サイクル数を記述する．新たに即値のレジスタ転送命令の AccessType オブジェクトを定義した．AccessType オブジェクトは Name，rwType，Access Size，Alignment Size，nBurst の属性を持っており，それぞれオブジェクト名，読み書き属性，アクセスのバイト数，アライメントのバイト数，バースト転送の長さのバイト数を表している．即値のレジスタ転送命令専用の AccessType オブジェクトとして，読み書き属性を持たない，Access Size が 4 バイトと 8 バイトの 2 つの AccessType オブジェクトを定義した．この AccessType オブジェクトに関連付けられた Performance を利用して即値のレジスタ転送命令の実行サイクル数を記述した．また，実行サイクル数はマニュアルと ISS 上での計測プログラムの実行により取得した．性能見積を行う際には見積対象の LLVM-IR 命令が即値オペランドを持っていた場合に，即値オペランドが整数型の場合は 4 バイト，浮動小数点型の場合は 8 バイトの性能値を，オペランド依存関係を利用した見積やメモリ配置を考慮した性能見積の見積サイクル数に加算する．即値のレジスタ転送命令を考慮する見積手法の対象となる命令は，RH850/E1M-S2 の命令セット上で即値オペランドが扱える命令を参考にし，除算・剰余算命令と浮動小数点演算，条件判定命令等を選択した．

Vol.2019-ARC-235 No.14
Vol.2019-SLDM-187 No.14
Vol.2019-EMB-50 No.14
2019/3/17

3.4 個別の命令に対する見積

- getelementptr 命令
  getelementptr 命令の実行サイクル数の見積では，ベースポインタが単純なポインタである場合には Best の実行サイクル数，ベースポインタが配列や構造体のポインタである場合には Typical の実行サイクル数で見積もるとした．
- call・ret 命令
  call・ret 命令は関数呼出しに関する命令であり，Best の実行サイクル数の計測が困難であったため，代わりに Typical の実行サイクル数を用いて見積を行うとした．
- 整数型の除算・剰余算
  除算や剰余算では，一般的なコンパイラでは除数が 2 のべき乗の即値である場合に，除算命令をシフト命令に変換する最適化が広く行われている．このことを考慮し，除数が 2 のべき乗の即値の場合には，除算や剰余算の実行サイクル数を，対応するシフト演算の性能値で見積るとした．

4. SHIMulator 動的性能見積フロー

図 4 に SHIMulator 動的性能見積フローを示す．このフローは，提案した見積手法を用いて SHIM 性能見積を実行するフローである．測定対象のソースコードを LLVM フロントエンドを用いて LLVM-IR コードへ変換する．LLVM フロントエンドには clang7.0.0 を使用している．生成された LLVM-IR と，あらかじめ作成しておいたターゲット SHIM を LLVM-IR シミュレータである SHIMulator に入力することで，SHIM 動的性能見積を実行する．SHIMulator は SHIM 性能見積の結果である測定対象のソースコードの見積サイクル数を出力する．

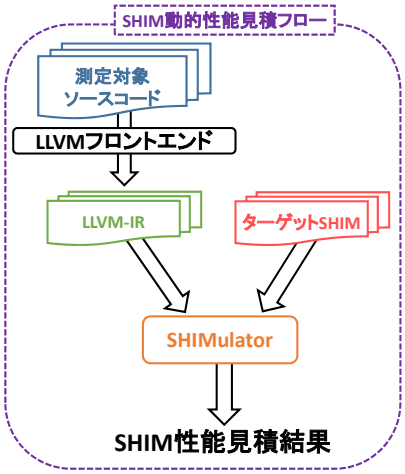


図 4 SHIMulator 動的性能見積フロー

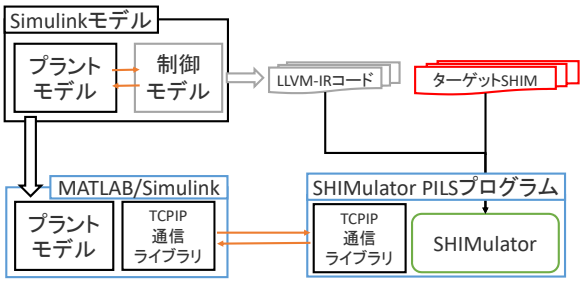


図 5 SHIMulator と MATLAB/Simulink の PILS の概略図

4.1 SHIMulator

SHIMulator は SHIMulator 動的性能見積フローにおいて，LLVM-IR コードの実行・解析と SHIM 性能見積を行うシミュレータである．C++で実装されており，LLVM7.0.0 のライブラリを使用している．LLVM-IR の構文解析には LLVM のライブラリ関数を利用している．仮想レジスタ，仮想メモリ，関数呼出し用のスタックを備えており，これらのリソースを操作することで LLVM-IR 命令を実行する．現在は一部の LLVM-IR 命令のみが実行可能である．各命令の動作は基本的に LLVM-IR のリファレンスマニュアル [5] に従うが，マニュアルによって指定されていない部分は独自仕様である．

SHIMulator の機能として，MATLAB/Simulink と連携したシミュレーションが可能である．これは MATLAB/Simulink のプロセッサインザループシミュレーション (PILS) 機能を利用している．PILS とは，実機や ISS とモデルを組み合わせたシミュレーションであり，制御アルゴリズムを実装した実機に対して，プラントモデルを用いて計算した値をセンサー等に入力し，出力値の検証等を行う手法である．

本研究では，SHIMulator と MATLAB/Simulink の連携による PILS を SHIMulator_PILS と呼ぶ．SHIMulator_PILS を行う際の概略図を図 5 に示す．SHIMulator_PILS では Simulink モデルから見積対象となる制御モデルを分離し，分離した制御モデルを MATLAB/Simulink の機能で C コードに変換し，LLVM フロントエンドを用いて LLVM-IR に変換する．シミュレーション実行時は SHIMulator_PILS プログラムが，SHIMulator の実行と MATLAB/Simulink との TCP/IP 通信を行う．通信を介して MATLAB/Simulink 上でプラントモデルのシミュレーションによって計算された数値を，シミュレーション中の制御モデルへの入力として受け取り，SHIMulator 上で制御モデルのシミュレーションを行い，出力値を MATLAB/Simulink に返す．SHIMulator_PILS プログラムは．SHIMulator 本体，TCP/IP の通信ライブラリ，SHIMulator の API を操作するラッパー関数で構成されている．

Vol.2019-ARC-235 No.14
Vol.2019-SLDM-187 No.14
Vol.2019-EMB-50 No.14
2019/3/17

5. 精度評価実験

5.1 実験環境

本実験のターゲットハードウェアは Renesas Electronics 社のマイコンである RH850/E1M-S2 である．このハードウェアには CPU として RH850G3MH コア，周辺制御用 (PCU) として RH850G3K が搭載されている．実験ではすべてのソフトウェアを RH850G3MH コアでのみ動作させ，E1M-S2 に搭載されている周辺機能は使用しなかった．RH850G3MH コアの特徴を以下に示す．

- 32 ビット命令セットアーキテクチャ
- 8K バイト命令キャッシュ
- 9 段パイプライン
- 動作周波数 320MHz
- 2 命令同時発行可能

実験では精度評価のリファレンスとして RH850/E1M-S2 の ISS を使用し，すべてのプログラムをユーザーモードで実行した．マニュアルと ISS を利用して RH850/E1M-S2 の SHIM を作成した．

表 1 サンプルプログラム

名称	概要
{Bubble/Merge/Quick}Sort.c	要素数100個のint型配列をLocal RAMに配置した {バブルソート/マージソート/クイックソート}
{Bubble/Merge/Quick}Sort_1000.c	要素数1000個のint型配列をLocal RAMに配置した {バブルソート/マージソート/クイックソート}
{Bubble/Merge/Quick}Sort_section.c	要素数1000個のint型配列をGlobal RAMに配置した {バブルソート/マージソート/クイックソート}
EuclideanAlgorithm.c	疑似乱数の組の最大公約数を ユークリッドの互除法で求める
GaussianElimination.c	10個連立一次方程式を解くガウスの消去法
NumberPlace.c	数独を解くプログラム

5.2 サンプルプログラムに対する精度評価実験

本実験では，複数の C プログラムを用いて SHIMulator 動的性能見積フローの見積精度評価を行った．使用したサンプルプログラムの概要を表 1 に示す．見積の対象は main 関数の開始から終了までとした．サンプルプログラムの見積サイクル数 $EstimationCycle$ を SHIMulator 動的性能見積フローにより求め，実行サイクル数 $ExecutionCycle$ を ISS での実行により求め, 見積誤差 $EstimationError$ を式 (2) により求めた．

$$EstimationError = \frac{(EstimationCycle - ExecutionCycle)}{ExecutionCycle} \quad (2)$$

求めた見積誤差を表 2 に示す．この表では見積精度は $EstimationCycle \leq ExecutionCycle$ の場合にマイナスで表記され，$EstimationCycle > ExecutionCycle$ の場合にプラスで表記される．見積誤差の絶対値は平均 11.29%，最大 16.74%，最小 0.27%となった．

表 2 サンプルプログラムの見積精度

プログラム	誤差 (%)
MergeSort.c	-15.68%
MergeSort_section.c	-14.22%
MergeSort_1000.c	-15.12%
QuickSort.c	-16.25%
QuickSort_section.c	-14.89%
QuickSort_1000.c	-16.74%
BubbleSort.c	-14.11%
BubbleSort_section.c	-8.68%
BubbleSort_1000.c	-14.64%
EuclideanAlgorithm.c	4.62%
GaussianElimination.c	0.27%
NumberPlace.c	0.29%
絶対値の平均値	11.29%

また，各サンプルプログラムの見積結果に対して誤差解析を行った．例として要素数 100 の int 型の配列のマージソートのプログラムである MergeSort.c の 1 命令当たりの誤差を表 3 に示す．各命令の誤差は式 (2) を各命令の見積結果に適用することで求めた．

表 3 から，多くの命令で見積サイクル数 < 実行サイクル数であることが分かる．提案手法では SHIM の Best の性能値を用いて見積サイクル数を算出しているため，見積サイクル数 < 実行サイクル数となる傾向があると考えられる．各命令の出現回数を比較すると，load 命令が全体の 35.38%を占めており，次いで br 命令が 18.06%，store 命令が 10.07%であることが分かった．これらの出現回数が多い命令の見積精度を向上させることで，見積精度を向上させることが可能だが，load 命令や store 命令の正確な性能見積にはターゲットハードウェアのメモリアーキテクチャに関する情報が必要であり，br 命令の正確な見積には分岐予測メカニズムの詳細が必要である．SHIM の性能値の計測方法を変更するか，新たな情報を追加しなければ，今後これらの命令の見積精度を向上させることは難しいと考えられる．一方で，fdiv 命令や fptosi 命令は見積精度が悪いが，プログラム全体に対する出現数が 1%以下であるために，全体の見積精度にそれほど影響していない，これらの命令は SHIM の計測の変更や個別の見積手法を導入することで見積精度を向上させる余地があると考えられる．

5.3 エンジン制御モデルに対する精度評価実験

MATLAB/Simulink のエンジン制御モデルに対し，SHIMulator_PILS による性能見積を行った．性能見積の対象であるエンジン制御モデルの情報を表 4 に示す．エンジン制御モデルの実行サイクル数については, ルネサス エレクトロニクス社の開発ツールである CS+による PILS シミュレーションを用いた，Embedded Target for RH850(ET)[8] の性能解析で取得した．この結果と SHIMulator_PILS の性能見積結果を比較して見積誤差を求めた．ET 性能解析

Vol.2019-ARC-235 No.14
Vol.2019-SLDM-187 No.14
Vol.2019-EMB-50 No.14
2019/3/17

表 3　MergeSort.c の 1 命令当たりの誤差

LLVM-IR 命令	add	alloca	br	fadd	fdiv	fmul	fptosi	getelementptr	icmp
命令出現回数	2512	1695	8698	200	100	400	200	3773	3627
各命令の見積誤差	88.63%	-100.00%	-54.07%	-24.14%	1500.00%	-50.00%	300.00%	-13.96%	-6.30%
LLVM-IR 命令	load	pseudocall	sdiv	sext	sitofp	srem	store	call+ret	全体
命令出現回数	17044	300	99	3772	200	100	4853	300	48172
各命令の見積誤差	-11.30%	-100.00%	14.29%	0.00%	-11.11%	-16.67%	-41.36%	-13.95%	-15.68%

と SHIMulator_PILS の見積結果を比較する際のコード生成フローを図 6 に示す．

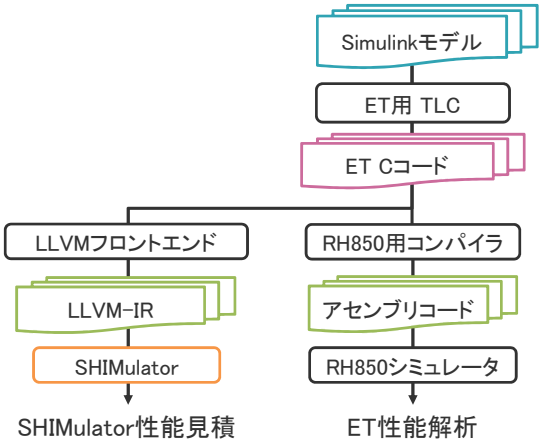


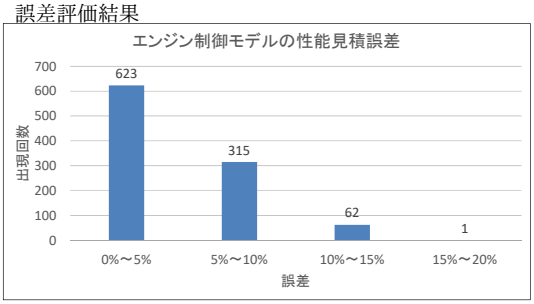
図 6　ET 性能解析と SHIMulator_PILS のコード生成フロー

表 4　エンジン制御モデルの特徴

モデル上のブロック数	733
LLVM-IR 命令数	2394
LLVM-IR 基本ブロック数	424

シミュレーションは合計 1001 回のイテレーションである．ET 性能解析と SHIMulator_PILS のシミュレーションで制御モデルの処理が同じであることを確かめるためにシミュレーション中の制御モデルへの入出力を全て記録し，一致することを確認した．表 5 に精度評価の結果を示す．この表は，各イテレーションにおいて ET 性能解析で取得した実行サイクル数と SHIMulator_PILS で取得した見積サイクル数から，式 (2) により求めた見積誤差の絶対値を集計し，ヒストグラム化したものである．結果としてすべてのイテレーションで誤差 ±20% 以内を達成した．また，最も誤差が大きかったイテレーションは，一番最初のイテレーションであり命令キャッシュミスの影響だと考えられる．提案手法では命令キャッシュの動作を考慮していないため，ミスによるペナルティで実行サイクル数が増加したことで，誤差が拡大したと考えられる．なお，この実験では，LLVM-IR とアセンブリコードを手動で対応付けることが困難であったため，詳細な誤差解析を行えなかった．

表 5　エンジン制御モデルの SHIMulator_PILS による性能見積の誤差評価結果



6. おわりに

本研究では，モデルベース並列化において，モデル設計段階でソフトウェアの性能を見積るための性能見積手法として，SHIM 動的性能見積手法を提案した．LLVM-IR 上のオペランドのデータ依存関係，LLVM-IR 上のメモリセクションの指定と SHIM のメモリアクセス性能を用いた見積手法，LLVM-IR 命令の即値オペランドから生成される即値のレジスタ転送命令を考慮した見積手法，個別の命令に対する特別な見積手法の 4 つの SHIM 性能見積手法を提案した．また，これらの性能見積を実行する LLVM-IR シミュレータである SHIMulator を開発し，SHIMulator 動的性能見積フローの提案を行った．

見積のターゲットハードウェアである RH850/E1M-S2 に対して，ハードウェアマニュアルとサイクル精度の ISS を用いて SHIM を作成した．SHIM 性能値の計測では，LLVM-IR から RH850/E1M-S2 のアセンブリ命令の対応関係を考慮し，インラインアセンブラ記述を用いた，性能値計測プログラムを作成した．

提案した SHIMulator 動的性能見積フローと作成した RH850/E1M-S2 の SHIM を用いて，サンプルプログラムに対する見積精度評価実験と，MATLAB/Simulink との連携を利用したエンジン制御モデルに対する精度評価実験を行い，精度目標であった誤差 ±20% を達成した．

精度目標であった誤差 ±20% 以内を達成したことにより，システム設計者が SHIM を用いて，モデル設計段階でモデルから生成した逐次コードの SHIM 性能見積を行うことで，実機や ISS が存在しない状況でもソフトウェアの実行サイクル数を簡単に予測できることが確認できた．

Vol.2019-ARC-235 No.14
Vol.2019-SLDM-187 No.14
Vol.2019-EMB-50 No.14
2019/3/17

7. 今後の課題

複数のハードウェアに対して性能見積手法を適用し，精度評価を行う必要がある．本研究では性能見積の対象のハードウェアが 1 種類のみであったため，異なるハードウェア間での精度評価を行っていない．また，提案した見積手法は見積対象のソフトウェアから生成された LLVM-IR と，ターゲットハードウェア上で実行されるコードが大きく異なる場合に，見積精度が大きく悪化することが予想される．特にオペランド依存関係を利用した見積手法は，LLVM-IR 上の命令の実行順に依存した見積手法であるため，アウトオブオーダー実行が行われるハードウェアに対して，見積精度が大きく悪化すると考えられる．このような特徴を持つハードウェアに対して見積精度評価を行い，見積手法の改善が必要であると考えられる．複数のハードウェアに対する性能見積の適用では，SHIM の作成も必要であり，SHIM 作成方法の改善も必要である．

また，SHIMualtor の拡張が必要である，今後は実行可能な命令の拡張，マルチコア対応，実行速度の向上．誤差解析手法の強化などの拡張が必要である．同時に，SHIMulator 動的性能見積フローについても，LLVM フロントエンドが特定のバージョンの clang に依存しているため，LLVM フロントエンドを開発し，生成される LLVM-IR の品質についても考慮する必要があると考えられる．

謝辞 ルネサス エレクトロニクス株式会社 共通技術開発第一統括部 技術ソリューション企画部 鈴木均氏に資料を提供していただくとともに有益なご助言をいただいた．ここに同氏に対して感謝の意を表する．

参考文献

[1] RH850/E1M-S2，2019，
https://www.renesas.com/jp/ja/products/microcontrollers-microprocessors/rh850/rh850e1x/rh850e1ms2.html
[2] SOFTWARE-HARDWARE INTERFACE FOR MULTI-MANY-CORE ($SHIM^{TM}$)，2019，
https://www.multicore-association.org/workgroup/shim.php
[3] The Multicore Association，2019，
https://www.multicore-association.org/
[4] Clang: a C language family frontend for LLVM，2019，
https://clang.llvm.org/
[5] LLVM Language Reference Manual，
https://releases.llvm.org/7.0.0/docs/LangRef.html
[6] 西村裕，中村隆，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．組込み技術とネットワークに関するワークショップ (ETNET2014), 2014.
[7] 溝口裕哉, ソフトウェア向けハードウェア性能記述を用いたプロセッサ性能見積手法に関する研究，名古屋大学大学院情報科学研究科，Master's thesis, 2016.
[8] RH850 マルチコア・モデルベース開発環境 [Embedded Target for RH850 Multicore]，2019，
http://www.renesas.com/mbd-rh850-multicore