Vol.2014-SLDM-165 No.26
Vol.2014-EMB-32 No.26
2014/3/16

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

西村 裕[1,a)]　中村 陸[1]　荒川 文男[1]　枝廣 正人[1]

概要：多種多様なメニーコアアーキテクチャが提案されてきており，今後ますます増える傾向にある．このような状況において，ソフトウェア開発ツール等がアーキテクチャ毎に個別に対応することは非効率である．これをなくすため我々は，マルチ・メニーコア標準プラットフォームの開発を進め，ソフトウェア視点での性能情報を含むハードウェア記述の標準化を提案している．本研究では，組込みメニーコアプロセッサの一つを用い，標準記述による性能見積りと実際の処理時間の差異を評価し，精度を高めるための手法及び課題について述べる．

1. はじめに

近年，消費電力や，発生する熱などの問題により従来のようにシングルプロセッサの性能向上によって様々な性能要求を満たすことは困難となった．しかも，組込み分野では，消費電力や温度条件などの制約が特に厳しいため，性能向上は難しい．

一方で，自動車制御分野などでは，より高度な処理を行うために性能要求が増大している．そして性能向上の有力な手法として，マルチコア及びメニーコアによる並列処理が期待されている．また異なる特徴を持つマイクロプロセッサを組み合わせたヘテロジニアスマルチプロセッサを効率よく用いることでの性能向上手法も期待されている．

しかし，メニーコアシステムはコア数やコア種といったツールに見える仕様の違いによって性能のバリエーションを与えるため，メニーコア上で動作する各種 OS やツールは個別に多くの実装を行わなければならないのが現状である．そのため，開発コストが高くなり，開発環境が充実せず，機器メーカーが十分な開発プラットフォームを使用できないという課題がある．

この課題を解決するため，ルネサス エレクトロニクス株式会社，イーソル株式会社，株式会社トプスシステムズ及び名古屋大学は，NEDO 戦略的省エネルギー技術革新プログラムの一つとして「多様なマルチ・メニーコアの高度な活用を可能にする標準プラットフォーム開発とエコシステム構築による省エネルギー技術の実用化」プロジェクトを推進し，標準プラットフォーム開発とエコシステム構築を進めている．その中で，名古屋大学はマルチ・メニーコアプラットフォーム標準化委員会を主宰し，ソフトウェア視点でのマルチ・メニーコア向けソフトウェア/ハードウェア標準インタフェースの開発を推進している．また，この標準インタフェースは SHIM（Software-Hardware-Interface for Multi-many-core）[1] と命名され，これを国際標準化するための活動として，MCA（Multicore Association）に SHIM ワーキンググループが設置された．この標準インタフェースは，性能予測，システム・コンフィグレーション，ハードウェアモデリングの他，多くのマルチ・メニーコア向け支援ツールの開発に活用できる．特に，多種多様なハードウェアに対するツールや OS の開発を容易にするものと期待されている．そして，性能予測への活用に向けて，SHIM の性能記述は誤差 ±20%以内を目標としているが，精度に関する検証はなされていなかった．

本研究の目的は，SHIM を用いることで，従来性能評価に必要であったターゲットハードウェア固有のコンパイラやシミュレータなどのツール類の開発前に，誤差 ±20%以内でシステムの性能評価を可能にすることである．さらに，チップ作成前の仕様策定段階での性能評価も可能にし，多種多様な候補から最適な仕様を選択できるようにする．またコア数やコア種を変えた構成の見積りも柔軟に行うことが可能となる．

本論文では，SHIM と共に高級言語と実機アセンブラの中間表現である LLVM IR（Low Level Virtual Machine Intermediate Representation）[2] を用いることで，ターゲットハードウェアやツール類開発前のシステム性能見積りが可能であることを示した．またシングルコアで動作す

[1] 名古屋大学
Furo-cho, Chikusa-ku, Nagoya City, Aichi 464-8601, Japan
[a)] nishimura@ertl.jp

るプログラムの並列化を行い，ホモジニアス型マルチコア上での性能の見積りを行った．この際，SHIM を用い負荷バランスを計算することで，コア間の負荷バランスに偏りがあることを検出し，より消費エネルギーの少ない，片方のコアの周波数を抑えた周波数ヘテロジニアス型のマルチコアにおいても，性能がほぼ劣化しないことを示した．これにより，コア数やコア種などの構成の違いを柔軟に変更可能であることを示した．そして LLVM IR と実機アセンブラの違いを考慮することによって，見積り誤差 ±20%以内を達成した．

本論文の構成は以下のとおりである．まず第 2 章では関連研究を紹介する．次に第 3 章ではソフトウェア向けハードウェア記述である SHIM についての説明する．次に第 4 章では SHIM の性能記述に用いられる LLVM について説明する．第 5 章では SHIM と LLVM を用いた性能見積り手法について述べる．最後に第 6 章にて，本論文のまとめと今後の課題について述べる．

2. 関連研究

ハードウェア記述としては IEEE 規格の IP-XACT[3] がある．IP-XACT は，特定の言語やベンダに依存しない IP 記述用の XML スキーマで，IP ブロックの機能と動作 (インタフェースの動作) に関する情報を記述することが可能である．IP-XACT に記述される情報にはメモリマップ，レジスタ，バス・インタフェース，ポート，ジェネレータ，コンポーネントのデザインビューなどがある．しかし，IP-XACT はハードウェア視点で IP を記述し，記述された IP を組み合わせて接続してチップを作り，それを検証するための仕様であるため，性能情報を含まない．

また，LLVM IR を用いた見積りの先行研究としては Abhijit Ray らによるものがある [4]．Abhijit Ray らは LLVM IR と実機アセンブラを比較し，命令のサイクル数を決定した上で，プロセッサの性能見積りを行っている．しかし見積りには実機と実機アセンブラが必要である．

3. SHIM

SHIM は，ソフトウェア視点でマルチ・メニーコアチップの各種仕様をツールがパラメータをとして統一的に取り扱うためのハードウェア・ソフトウェア・インタフェースである．

SHIM の主要メンバを表 **1** に示す．ComponentSet には CPU や DMA といったマスター型構成要素，メモリなどのスレーブ型構成要素の情報が記述される．マスター型構成要素の情報には LLVM IR での性能情報記述が含まれる．AddressSpaceSet は複数の AddressSpace 情報を持つ．また MasterSlaveBinding 情報や性能情報として latency と pitch の情報などを持つ．CommunicationSet は通信に関する情報が記述される．記述される情報は，FIFO や共有

表 **1** SHIM XML の構造
Table 1 Structure of SHIM

主要メンバ	意味
ComponentSet	ハードウェア構成要素の情報
AddressSpaceSet	アドレス空間情報
CommunicationSet	コミュニケーションに関する情報

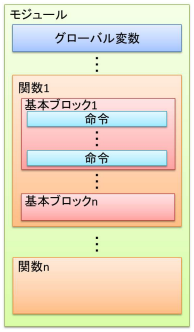

図 **1** LLVM-IR の構成
Fig. 1 Structure of LLVM-IR

レジスタを用いた通信，割込み，イベントなどがある．またそれぞれについてレイテンシ情報（best 値，typical 値及び worst 値）を持つ．

IP-XACT との違いは，IP-XACT は物理的な接続を情報として持つが，SHIM はソフトウェア視点で必要な情報の記述に特化するというポリシーから，物理的な接続に関する情報は持たない点と，IP-XACT は性能情報を持たないが，SHIM はメモリへのアクセスレイテンシやプロセッサの性能情報を持つ点である．

4. LLVM

LLVM はコンパイラ・インフラストラクチャ・フレームワークである．LLVM は言語やターゲットとなるハードウェアに依存しない中間表現である LLVM IR を規定する．LLVM IR の命令の多くは 3 番地形式に似ており，変数はレジスタに保存される．LLVM IR はレジスタ数に上限のないレジスタマシンをターゲットとしており，変数はプログラム上の一箇所でのみ代入される静的単一代入形式で表現される．

LLVM IR の構成を図 **1** に示す．LLVM IR は翻訳単位となるモジュールからなる．各モジュールは，関数，グローバル変数，シンボルテーブルエントリで構成される．関数は，基本ブロックで構成され，基本ブロックは複数の命令で構成される．各基本ブロックには一意のラベルが付与される．

LLVM の最適化支援機能の中に，汎用 PC 上でプログラムの実行プロファイルを取得する機能がある．実行プロファイルは各関数の呼び出された回数及び，各基本ブロックの実行回数を含む．

Vol.2014-SLDM-165 No.26
Vol.2014-EMB-32 No.26
2014/3/16

5. SHIM を用いた性能見積り

5.1 実行サイクル数の見積り手法

命令 i のサイクル数を C_i，実行回数を I_i とすると，実行サイクル数 T は式 1 で表すことができる．

$$T = \sum_{i=0}^{n} (C_i \times I_i) \quad (1)$$

ただしメモリアクセス命令のサイクル数についてはメモリアクセスのレイテンシを用いる．

5.1.1 命令の実行回数の取得

命令の実行回数は LLVM のプロファイル機能を用いて取得する．ある関数内の命令 i の実行回数 I_i は，LLVM の基本ブロック j の実行回数を B_j，基本ブロック j における命令 i の実行回数を N_{ij} とすると，式 2 で表すことができる．今回，それぞれの命令について関数毎にカウントする．

$$I_i = \sum_{j=0}^{n} (B_j \times N_{ij}) \quad (2)$$

5.2 命令サイクル数の取得

LLVM IR の各命令のサイクル数は本来は全て SHIM のマスター型の構成要素内に記述されている性能情報を使用する．ただし，メモリアクセス命令については，対応するアドレススペースのアクセスレイテンシを SHIM から取得する．しかし，本研究は SHIM XML を定義するプロジェクトの中で，その性能記述能力を検証するために，研究対象チップの SHIM XML 記述が不完全な状況で進めたため，SHIM XML の性能情報の記載が不完全な場合にも対応する必要があった．こうした場合の LLVM IR のサイクル数設定方法を述べる．

(1) 加算や減算などの基本的な命令についてはハードウェア仕様書より取得する．
(2) 複合的な命令については，チップでの動作を想定した上で複数命令のサイクル数の加算などにより決定する．

2 の複合的な命令の例としては要素ポインタ取得命令（getelementptr）がある．例えば 1 次元の int 型配列の n 番目の要素へのポインタを取得する場合は次のような計算を行うことが一般的である．

(1) 配列の先頭アドレスを取得する．（mov 命令など）
(2) n 番目までのアドレスのオフセットを計算する．（shl 命令など）
(3) 1 と 2 を足し合わせる．（add 命令）

したがって 1 次元の配列の要素ポインタ取得命令のサイクル数はこれらの命令の合計サイクル数で表すことが可能である．

5.3 シングルコアにおける性能見積り

シングルコア環境において 5.1 節で述べた見積り手法を用い JPEG アプリケーションの性能見積りを行う．

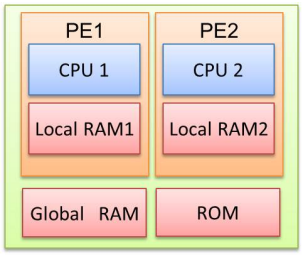

図 2 ハードウェア構成
Fig. 2 Hardware configuration

表 2 メモリアクセスレイテンシ
Table 2 Memory access latency

名前	CPU1 read, write	CPU2 read, write
Local RAM1	2,2	-,-
Local RAM2	-,-	2,2
Global RAM	4,4	4,4
Global ROM	4,-	4,-

5.3.1 実験環境

本実験環境は下記のとおりである．

- アプリケーション
  - JPEG Decoder[5]
    - DecodeHuffMCU：ハフマン復号
    - iZigzagMatrix：逆ジグザグスキャン
    - IQuantize：逆量子化
    - ChenIDct：逆 DCT 変換
    - PostshiftIDctMatrix：要素に 128 を加える
    - BoundIDctMatrix：要素の上限を確認
- 対象とするチップ
  - v850 アーキテクチャ
  - ハードウェア構成（図 **2**）
    - PE1 のみ使用する
  - メモリアクセスレイテンシ（表 **2**）
- 比較
  - v850 シミュレータ Cforest_mp
- 使用するコンパイラ
  - Clang （見積り用）
  - v850-renesas-elf-gcc （比較用）
  - 最適化オプション：-O0

5.3.2 LLVM IR の命令サイクル数

今回設定した LLVM IR のサイクル数を表 **3** に示す．なお load 命令及び store 命令は表 2 のサイクル数を使用した．

5.3.3 見積り結果

シングルコアにおける見積り結果を図 **3** に示す．DecodeHuffMCU 関数において見積り値とシミュレーション結果との誤差が-19.6%，ChenIDct 関数においては 169%発生していることが分かる．

表 3　LLVM IR のサイクル数
Table 3　Cycles of LLVM-IR

LLVM IR	cycle	意味
br	2	分岐命令分岐（確率を 1:1 と仮定）
add	1	加算命令
sub	1	減算命令
sdiv	36	除算命令
mul	1	乗算命令
and	1	論理積命令
or	1	論理和命令
ashr	1	算術右シフト命令
lshr	1	論理右シフト命令
shl	1	左シフト
alloca	0	スタックメモリ割当命令
getelementptr	3	アドレス計算（1 次元配列と仮定）
sext	1	整数型符号拡張命令
zext	1	整数型ゼロ拡張命令
icmp	1	比較命令
phi	0	phi 命令
call	4	関数呼び出し命令
ret	4	復帰命令

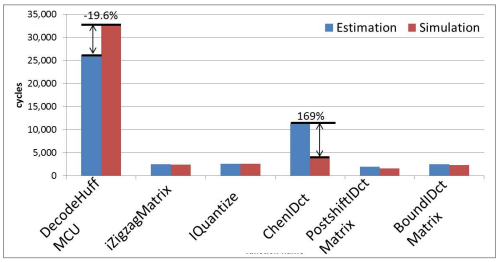

図 3　シングルコアでの見積りとシミュレーション結果の比較
Fig. 3　Comparison of the estimation and the simulation on the single-core

5.3.4　考察

DecodeHuffMCU 関数及び ChenIDct 関数における誤差の要因は次のことが考えられる．

(1) 除算命令
(2) register 修飾子
(3) 関数呼び出しのオーバーヘッド

(1)，(2) に関しては，中間表現からターゲットのマシンアセンブラに変換する際の，バックエンドコンパイラによる最適化が原因であると考えられる．中間表現において除算命令が使用されている場合でも，バックエンドコンパイラにより実機ではシフト命令で除算を実装したほうが高速であるとされた場合，除算命令はシフト命令を使用した複数命令に置き換えられて実行される．これは定数による除算（特に 2 冪乗で割る場合）において発生することがある．今回 ChenIDct 関数では 16 で割る処理が記述されており，この処理がシフト命令に変換されたために大きな誤差が発生した．register 修飾子は実機アセンブラに変換する際に出来るだけ指定の変数をレジスタに保持し実行するという意味である．しかし，中間表現レベルではレジスタ修飾子については考慮されていない．そのため中間表現

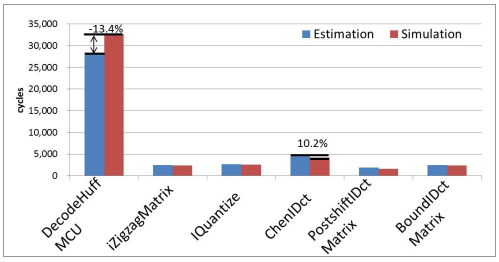

図 4　シングルコアでの見積りとシミュレーション結果の比較 (改善)
Fig. 4　Comparison of the estimation and the simulation on the single-core (Improvement)

での load/store 命令が多くなり，実機よりも見積り値が大きくなったと考えられる．(3) の関数呼び出しのオーバーヘッドに関しては，関数呼び出し時の動作が実機と LLVM IR とで異なるためと考えられる．LLVM IR では関数呼び出しは call 命令で実装され，呼び出し元への復帰は ret 命令で行われる．しかし，これに加えて実機での動作時には関数内で使用するレジスタの退避回復動作などが必要となる．DecodeHuffMCU 関数は内部で複数の関数呼び出しを行っているため，レジスタ退避回復動作のオーバーヘッドを加える必要がある．

以上より，(1) については 2 の冪乗で割る際の除算命令のコストをシフト命令を使用した複数命令のコストとして計算し，(2) については register 修飾子のある変数はすべてレジスタにあると仮定し，load/store のコストをメモリからレジスタに変更する．また (3) については関数呼び出しにおけるレジスタ退避回復動作を加えることで誤差が少なくなると考えられる．

以上を考慮した場合の見積り結果を表 **4** に示す．DecodeHuffMCU 関数において見積り値とシミュレーション結果との誤差は-20%から-13%に，ChenIDct 関数においては 169%の誤差を 10%まで削減することができた．

5.4　ホモジニアス型マルチコアにおける性能見積り

ホモジニアス型マルチコア環境において JPEG アプリケーションの性能見積りを行う．

5.4.1　実験環境

本実験環境は 5.3.1 小節における環境において PE1 に加えて PE2 を使用可能とした環境である．

5.4.2　パイプライン並列化

5.3.1 小節に示した 6 つの関数について図 **5** のように二段のパイプライン並列化を行う．分割の基準は処理の特性によるもので，前半部分は逐次処理に向いており，後半部分は SIMD（Single Instruction/Multiple Data）処理に向いているという特徴を持つ．今回の入力画像はサンプリングファクタが 4:1:1 であるため 6 回くり返し実行する．この時コア 1 からコア 2 へのデータ転送が必要となる．今回は共有メモリを用いたコア間通信によってデータ転送する

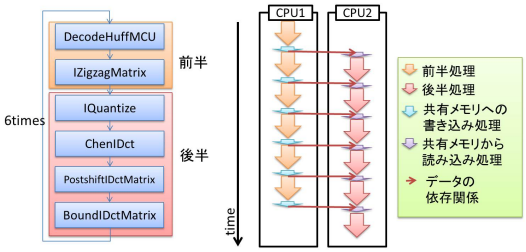

図 5　パイプライン並列化
Fig. 5　Pipeline parallelization

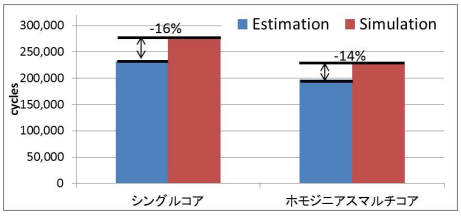

図 6　ホモジニアス型マルチコアでの見積り
Fig. 6　Performance estimates on the homogeneous multi-core

ことを想定する．この際，コア 1 は共有メモリにデータを書き込み，コア 2 は共有メモリからデータを読み込む処理が必要となる．したがって前半処理時間を P_f，後半処理時間を P_l，共有メモリへの書込処理を M_w，読込処理を M_r とすると処理時間 T_{pipe} は式 3 で表すことができる．

$$T_{pipe} = \max\{(P_f + M_w), (P_l + M_r)\} \times 5 + (P_f + M_w) + (P_l + M_r) \quad (3)$$

5.4.3　見積り結果

ホモジニアスマルチコアにおける見積り結果を図 **6** に示す．シングルコアで 6 回実行した時は，見積りとシミュレーションの誤差は-16%，パイプライン化し 2 コアで実行した際の誤差は-14%である．またシングルコアからマルチコア化した際の性能向上率は，シミュレーションでは 20%，見積りでは 17%となった．

5.4.4　考察

パイプライン並列化を行うことで，性能は 17%程度向上することが見積りによって分かる．理想的な分割 (処理量 1:1) で，通信のオーバーヘッドが無いと仮定すると，式 3 より $((2 \times 6)/(1 \times 5 + 1 + 1)) - 1 = (12/7) - 1 \fallingdotseq 71\%$ の性能向上が可能である．また，より大きな処理の中で並列化されると，パイプライン並列化では並列動作しない前処理後処理を前後の他の処理と同時に処理することが可能であり，性能向上率は 2 倍に近づくと考えられる．しかし，今回の場合性能向上は 17%と低い．ここで，シングルコアにおける各関数の SHIM を使用した見積り結果を基にコア 1 とコア 2 の負荷バランスを計算すると，図 **7** のようになる．コア 1 とコア 2 の負荷バランスはおおよそ 5:2 となっていることが分かる．したがって，コア 2 の性能をコア 1 より低くしても性能がそれほど変わらないと考えられる．

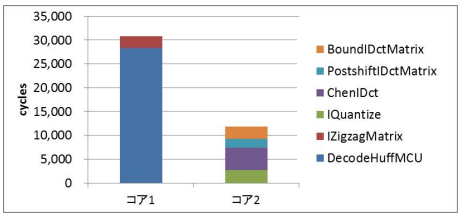

図 7　ホモジニアスマルチコアにおける負荷バランス
Fig. 7　Load balance of the homogeneous multi-core

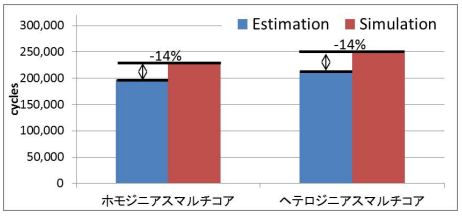

図 8　ヘテロジニアス型マルチコアでの見積り
Fig. 8　Performance estimates on the heterogeneous multi-core

また，負荷バランスが 5:2 の状態において，通信のオーバーヘッドが無いと仮定すると，性能向上率は式 3 より $((7 \times 6)/(5 + 5 \times 5 + 2)) - 1 \fallingdotseq 31\%$ となる．したがって，通信オーバーヘッドを隠ぺいすることによっても性能の向上が期待できる．通信オーバーヘッド削減方式としては，共有メモリへの書き込みや読み出しを DMAC（Direct Memory Access Controler）などにオフロードして，CPU コアの処理と並列に行う方式が考えられる．

また，今回のパイプライン化では，コア 1 の処理がデータをメモリに書き込み，その後コア 2 の処理がメモリからデータを読み込む．またデータをローカルメモリにコピーするので，計算処理中の共有メモリアクセスは少ない．さらに，計算処理に比べデータ転送の時間が非常に短いため，共有メモリへのアクセス衝突が起こりにくい．そのため，マルチコアに起因する精度の劣化が少ないと考えられる．

5.5　ヘテロジニアス型マルチコアにおける性能見積り

5.4 節にてコア間の負荷バランスが 5:2 と不均衡であることが示せたため，コア 2 の周波数をコア 1 の周波数の半分にした周波数ヘテロジニアス型マルチコア環境において JPEG アプリケーションの性能見積りを行う．

5.5.1　実験環境

本実験環境は図 2 において周波数比を CPU1 : CPU2 = 1 : 1/2 とした環境である．

5.5.2　見積り結果

周波数ヘテロジニアスマルチコアにおける見積り結果を図 **8** に示す．ホモジニアス型のマルチコアと比較して見積りではホモジニアス型の方が 6%程度性能が良い結果となった．またシミュレーションでも同様に 7%程度ホモジニアスの方が性能が良い結果となった．

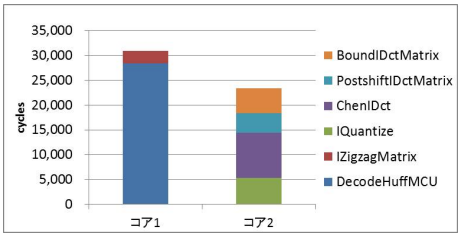

図 9　ヘテロジニアスマルチコアにおける負荷バランス
Fig. 9　Load balance of the heterogeneous multi-core

5.5.3　考察

ホモジニアス型のマルチコアと比較して，片方の CPU の周波数を半分にしているにもかかわらず，性能の劣化は 6%程度であるということが見積りより分かる．SHIM から求めた負荷バランスは図 **9** となる．このため，ホモジニアス型からヘテロジニアス型にした際に最も差が出るのは，式 3 における第 3 項と分かる．これによる性能劣化 P_d は式 4 によって表すことができる．

$$P_d = \frac{(P_l + M_r)_{Hetero} - (P_l + M_r)_{Homo}}{(T_{pipe})_{Homo}} \qquad (4)$$

この時，

$$\frac{23979 - 12181}{197975} = 0.059 \qquad (5)$$

となるため，性能劣化の原因は式 3 における第 3 項の実行時間の増加であることが分かる．したがって，より大きな処理の中で並列化を行った場合，前後の処理時間は無視できるため，ヘテロジニアス化による性能劣化は生じない．よってオーバーヘッドなく低電力化ができることが示された．また，周波数を半分にしたコア 2 の処理は SIMD 演算に適した処理を集めており，SIMD 処理に適したコアを採用してアーキテクチャレベルでのヘテロジニアス化を図ることで，更に低周波数のプロセッサで実行でき，更なる低電力化が期待できる．

6.　おわりに

6.1　まとめ

本論文では，SHIM 準拠のソフトウェア向けハードウェア性能記述と LLVM IR を用いることで性能見積りを行った．シングルコア，ホモジニアス型マルチコア及びヘテロジニアス型マルチコアにおける性能見積りを評価した．その上で，SHIM を用いることで，コア数やコア種を変えての見積りが容易であることを示し，また SHIM から導き出したコア毎の負荷バランスから，より省エネルギーな構成でも性能はそれほど劣化しないということを示した．これにより，SHIM を使用することで様々なハードウェア構成の性能を柔軟に見積もれることを示した．また LLVM IR と実機アセンブラの違いを考慮することで見積り精度を向上させることが可能であることを示した．今回，本手法を適用することによって，JPEG アプリケーションに含まれる関数に対して，最大 169%あった誤差を 10%まで削減することができた．また，SHIM の目標である誤差 ±20%を達成することができた．

SHIM を用いることで，ハードウェアやシミュレータがない場合でもおおまかな性能見積りが可能である．そのため，シミュレータで対応できないような大規模アプリケーションの性能見積り及び並列化支援や，多種多様なメニーコアアーキテクチャの中から最適な構成の選択が容易になると考えらる．また，逆に特定のソフトウェアが最適に動作するよう SHIM のパラメータを最適化して，それに基づくハードウェアを開発するといったことにも使用可能であると考えられる．

6.2　今後の課題

本研究では，命令の出現頻度をベースとした性能評価を行ったが，実際は分岐によるハザードやキャッシュなど，命令の出現頻度だけでは推定できない影響を考慮する必要がある．SHIM では様々な要因による性能のばらつきを woest-typical-best の 3 種類の値で表す．またメモリアクセスでは連続してアクセスした場合を考慮するため latency と pitch の 2 種類の値を用意している．これらの値を使う際にどのように用いるか，またマルチ・メニーコアの性能のばらつきを適切に表現できるかを調査する必要がある．

また，本研究では，SHIM による性能評価と実機シミュレータによる性能評価において，違いが生じた場合の原因分析を再優先したため，コードを目で追えるようにコンパイルオプションを最適化を行わないレベルに設定した．そのため，比較的容易に見積りを行うことができたが，最適化を行うとレジスタ数に上限のない LLVM では load/store が少なくなると考えられるが，実機のレジスタは有限個であるため，load/store の命令数などに大きな差が生じると考えられる．したがって，LLVM IR と実機アセンブラの違いを更に考慮する必要があると考えられる．

また，SIMD 処理に適したコアを採用するといった，アーキテクチャレベルでのヘテロジニアス化についても性能評価を行う必要があると考える．

参考文献

[1] SOFTWARE-HARDWARE INTERFACE FOR MULTI-MANY-CORE (SHIM), `http://www.multicore-association.org/workgroup/shim.php`.
[2] The LLVM Compiler Infrastructure, `http://llvm.org/`.
[3] IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tools Flows, IEEE Std 1685-2009.
[4] Abhijit Ray, Thambipillai Srikanthan and Wu Jigang, Rapid Techniques for Performance Estimation of Processors, Research and Practice in Information Technology, Vol. 42, No. 2, 2010.
[5] CHStone, `http://www.ertl.jp/chstone/`.