ハードウェア抽象化記述SHIMと性能見積

2021/11/18

名古屋大学大学院 情報学研究科
組込みマルチコアコンソーシアム
枝廣　正人

アジェンダ

- SHIMとは
- SHIM・LLVM-IR・性能見積手法
- 最近の取組（回帰分析を用いたレイテンシ計測）
- まとめと今後の課題

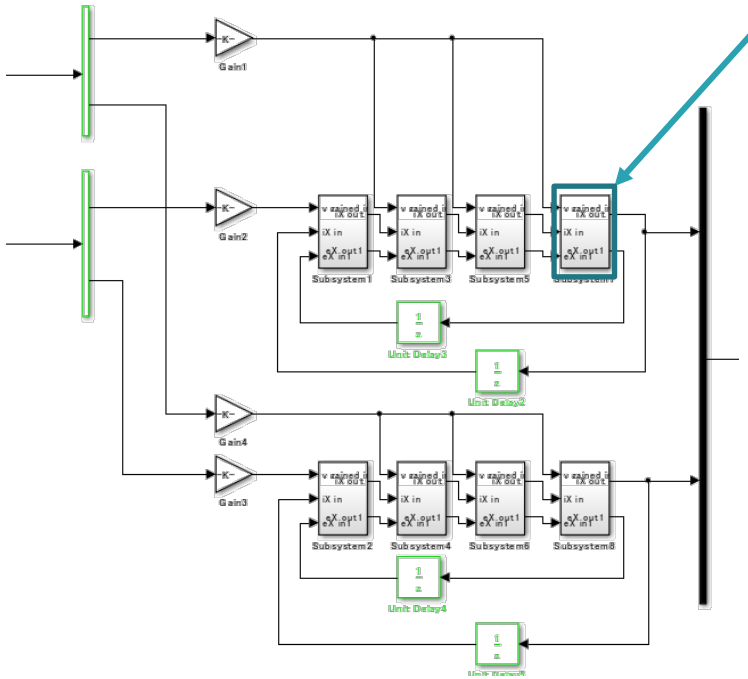
SHIM (Spacer)

- 〔隙間を埋めて位置を調整する〕くさび、詰め木、シム（アルクの辞書より）

SHIMが作られた背景

例えば枝廣研究室では：モデルベース並列化設計を研究

高い並列度を出すためにブロック粒度(処理時間)を考慮する必要がある



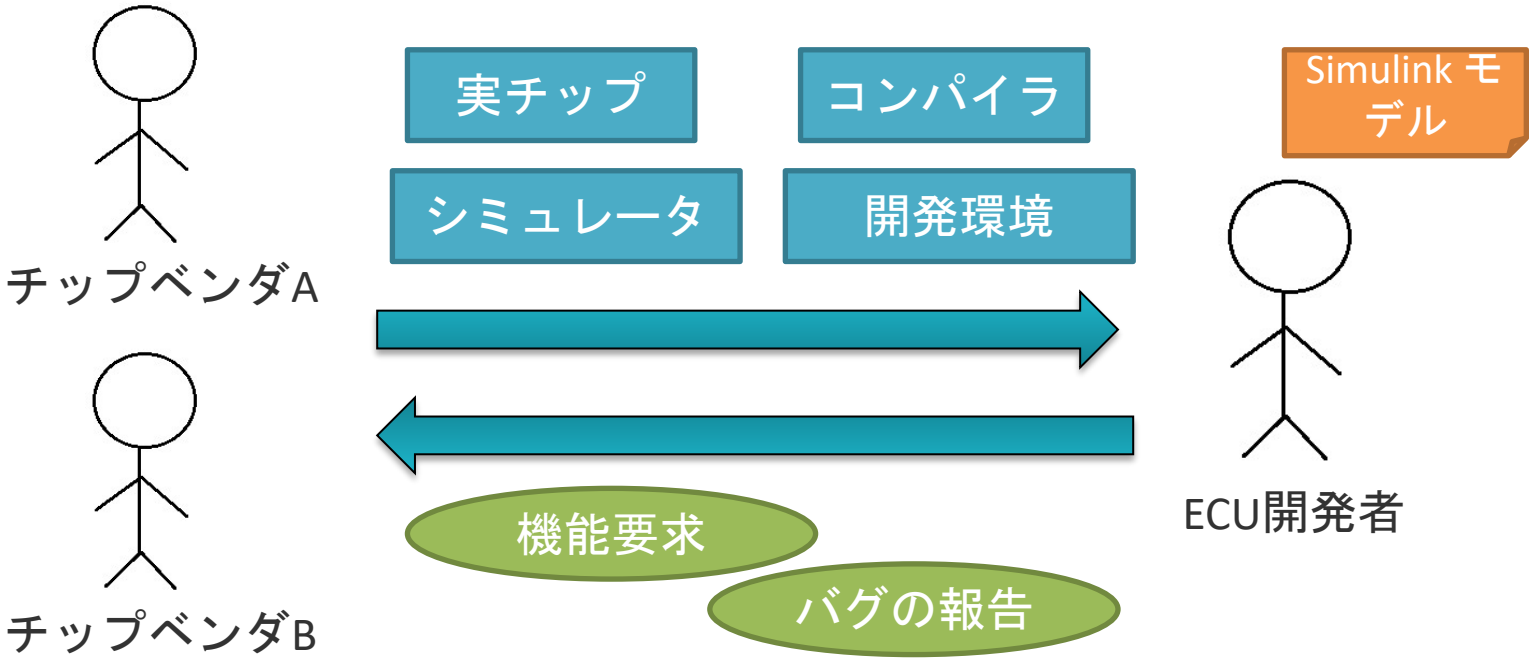
実機で計測すると高精度
- 実機がないと計測出来ない

保有していても
- それぞれの設定工数が大きい
- 使いこなすために知識と時間が必要

要望
- 実機がない状態で測定したい
- ボードの特徴を最大限に利用したい
- 様々なボードに対応してほしい

SHIMに期待されること

例えばアプリケーション向けECU開発(従来)

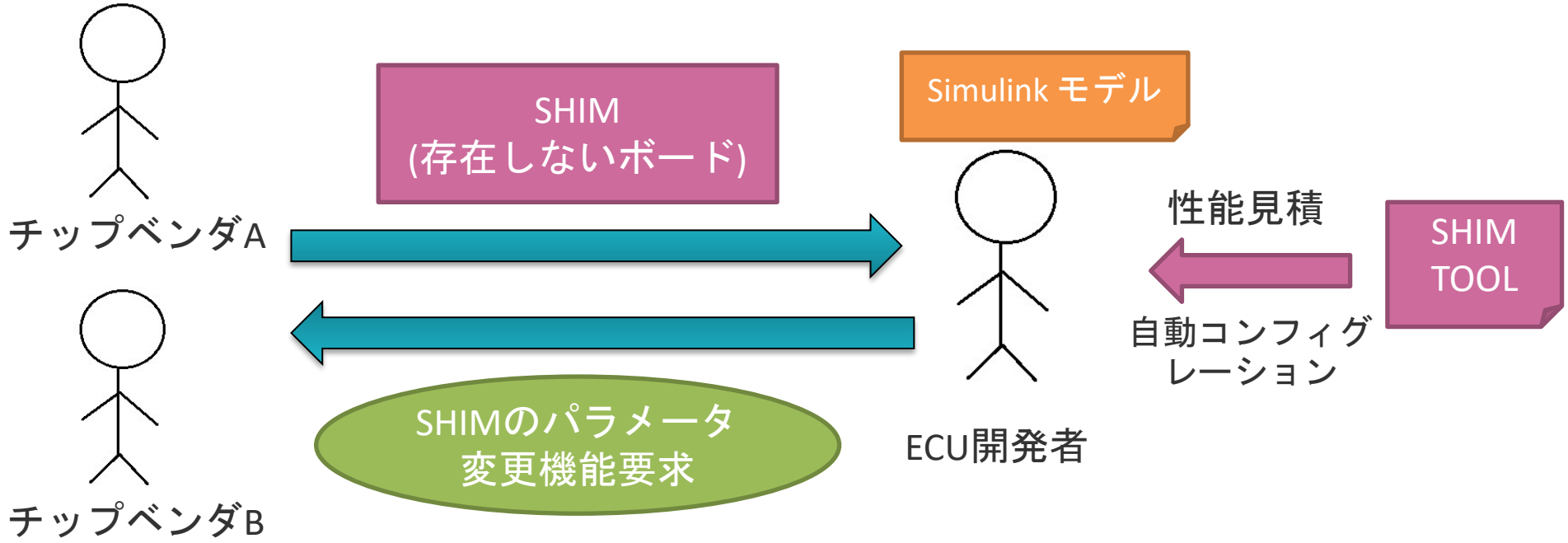


問題点：工数が多く、開発が非効率

複数のチップの検討は現実的ではない
シミュレータがないと検討すら出来ない

SHIMに期待されること

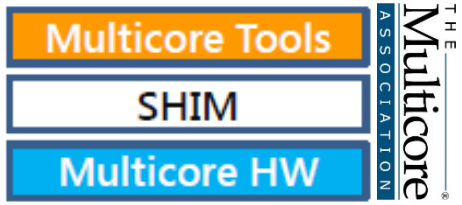
アプリケーション向けECU開発(SHIM)



上流工程(実現可能の検討)や複数チップ評価の工数を大幅に削減可能

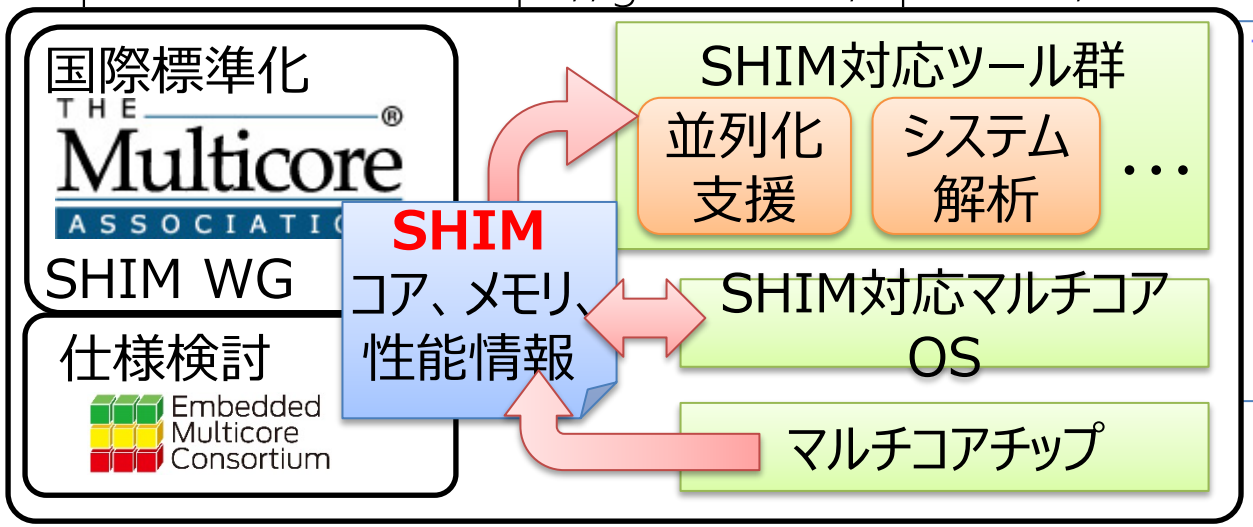
SHIMとは

Software-Hardware Interface for Multi-many-core



- 多様なマルチコアチップを抽象化したXML記述
  - コア種類・数、メモリ配置、アドレスマップ、通信、コア→メモリ性能情報等が、数百ページの説明書を読まずとも、機械的に読める
  - 性能情報の例：コアAからメモリ番地Xにアクセスしたときの(best, typ, worst)レイテンシ（右下図参照）
  - ツール群、OS等がSHIM対応することにより、多様なマルチコアチップを共通的に扱えるようにすることが目的

SHIM仕様書 http://www.multicore-association.org/workgroup/shim.php
Open SHIM Github https://github.com/openshim/shim




```
<MasterSlaveBinding slaveComponentRef="LRAM_B
  <Accessor masterComponentRef="CPU_BOCOP2">
    <PerformanceSet>
      <Performance>
        <accessTypeRef>Instruction_Fetch</acc
        <Pitch best="1.0" typical="1.0" worst
        <Latency best="1.0" typical="1.0" wor
      </Performance>
      <Performance>
        <accessTypeRef>Load_Algined_Byte</acc
        <Pitch best="1.0" typical="1.0" worst
        <Latency best="1.0" typical="1.0" wor
      </Performance>
```

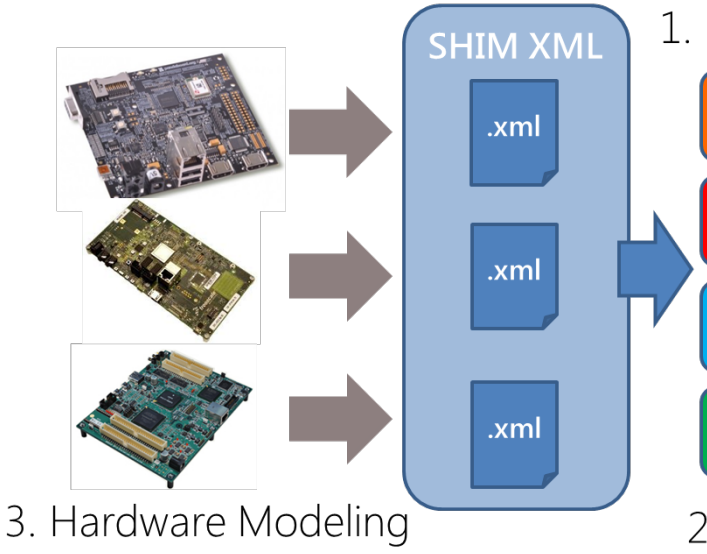
コア→メモリ性能情報
SHIM記述例

Embedded Multicore Consortium

SHIMとは

- ハードウェア機能の記述ではない
  - コア種類・数、メモリ配置、アドレスマップ、通信、性能情報などの記述である
- ハードウェアの完全な記述ではない
  - ソフトウェアから知りたいことのみに関する
- ツールではない
  - SHIMの記述内容を用い、各ベンダがツールを開発することを想定している

SHIMのユースケースとメリット

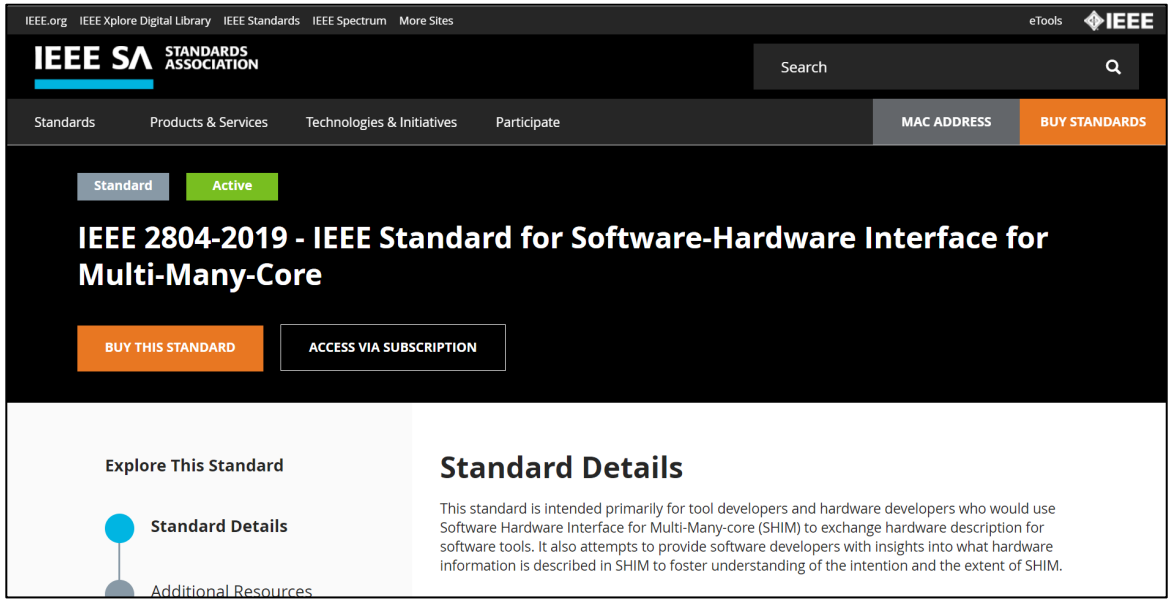


- システム性能見積
- ハードウェアモデリング/複数HWの対応
- ツールコンフィギュレーション

- マルチコアにおけるアプリケーション実行性能見積
- マルチコア選定時のアプリケーション実行性能比較
- 異なるマルチコアへのアプリケーション移植の際の性能見積
- 複数マルチコアをターゲットとしたソフトウェア部品開発
- 特定アプリケーション向けに特化したマルチコアを企画する際の性能評価
- マルチコア向け開発支援を行う各種ツールの開発コスト低減とSHIM対応ツールエコシステム

SHIM2.0がIEEE標準に！

- SHIM2.0では以下の課題について強化
  - ヘテロジニアス対応 / LLVM-IRでは表しきれない命令
    - ハードウェアが持つ画処理・知能処理関数アクセラレータ等
  - 電力見積
    - DVFS (Dynamic Voltage & Frequency Scaling)
  - 通信競合
    - 特にマルチコアでの見積に重要
  - アーキテクチャの表現強化
    - Out-of-Order, SIMDなど
  - キャッシュ/メモリアーキテクチャの表現強化
  - モジュール化による記述量削減
  - etc.



- IEEE標準として承認⇒IECとのDual Logoに向けて活動中

Embedded Multicore Consortium

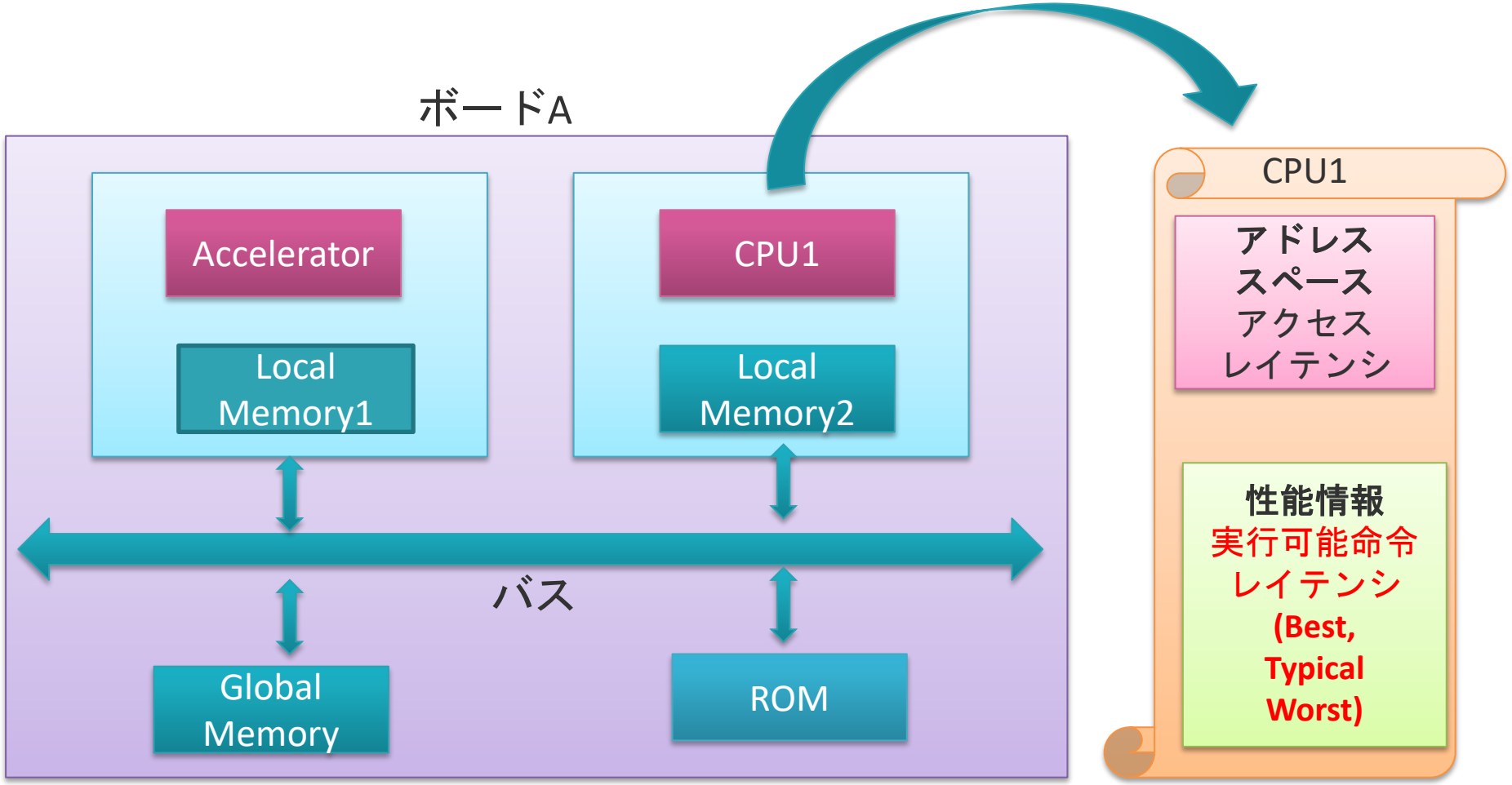
アジェンダ

- SHIMとは
- SHIM・LLVM-IR・性能見積手法
- 最近の取組（回帰分析を用いたレイテンシ計測）
- まとめと今後の課題

Embedded Multicore Consortium

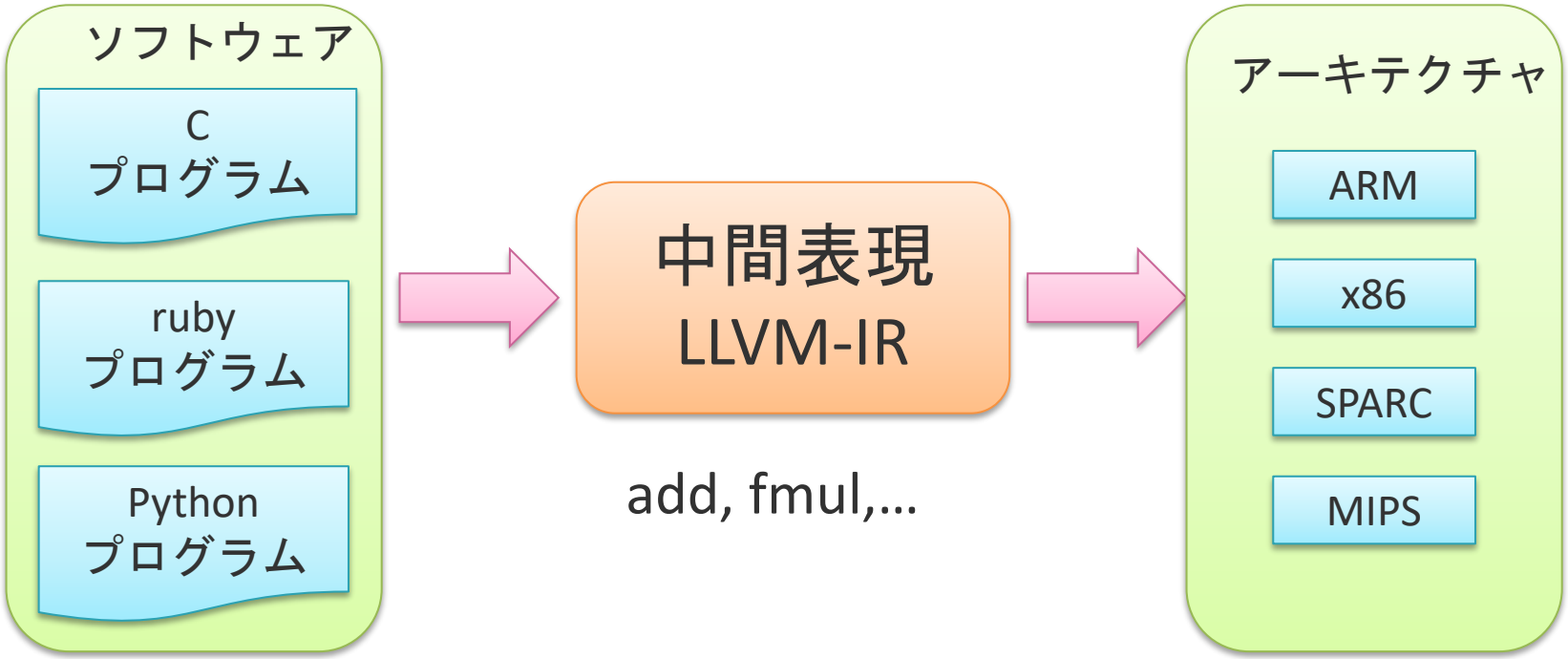
SHIM

- **SHIM記載情報**(XMLで記述)

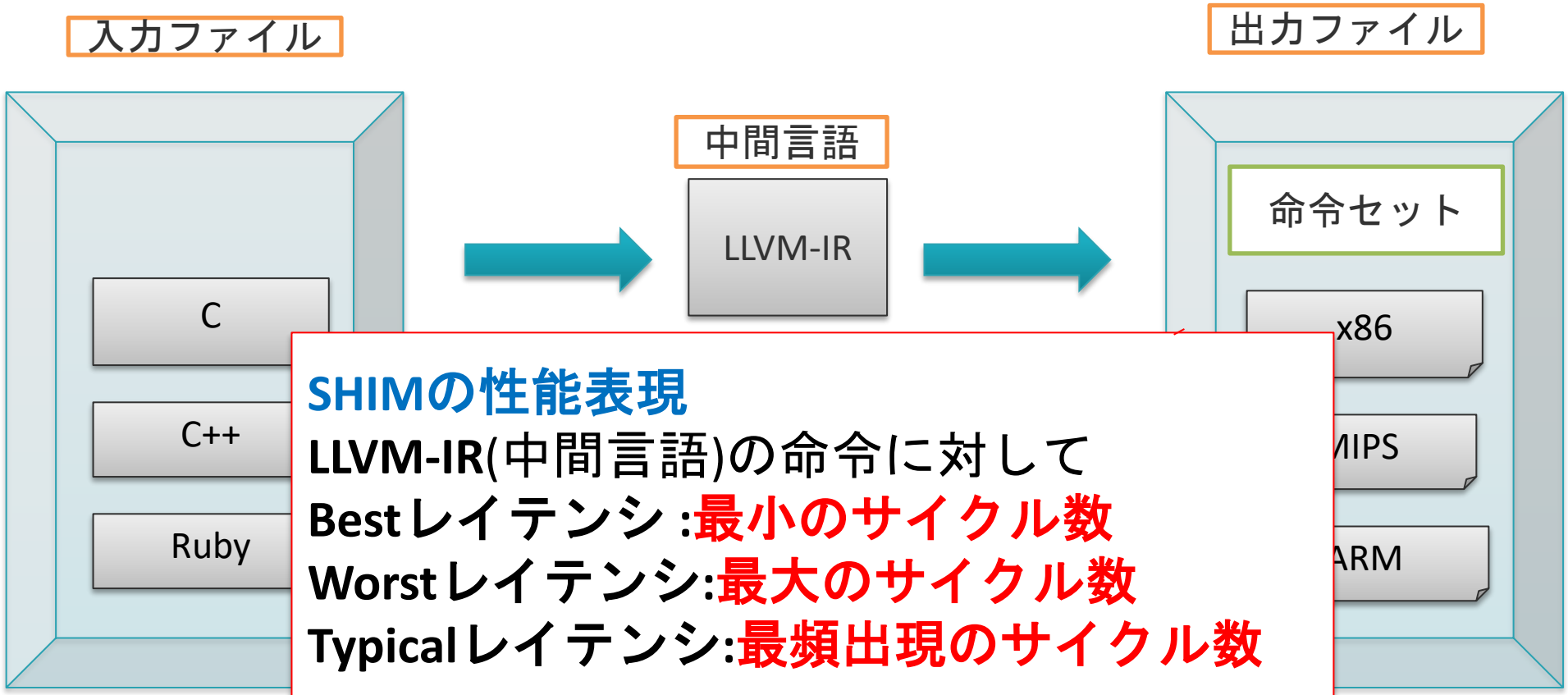


LLVMとLLVM-IR

- LLVMとは (The name "LLVM" itself is not an acronym. (from llvm.org))
  - オープンソースのコンパイラ基盤
- LLVM-IRとは (LLVM Intermediate Representation)
  - 言語やアーキテクチャから独立した中間表現
    プロセッサのアセンブラ命令に近い



LLVM-IRとSHIM



SHIMの性能表現

LLVM-IR(中間言語)の命令に対して

Bestレイテンシ :最小のサイクル数

Worstレイテンシ:最大のサイクル数

Typicalレイテンシ:最頻出現のサイクル数

メリット

- アーキテクチャに非依存
- 複数のプログラム言語に対応

性能見積手法

⇒基本的な考え方、課題は　SHIMとは独立によく知られている

- **静的手法**
  - LLVM-IR解析⇒性能見積
    - ループの実行回数やメモリアクセスが不明
    - 命令レイテンシ(Best, Typical, Worst)の選択が困難

- **動的手法**
  - LLVM-IR用シミュレータ ⇒ 性能見積
    - SHIMを用いたLLVM-IRシミュレータは存在しない

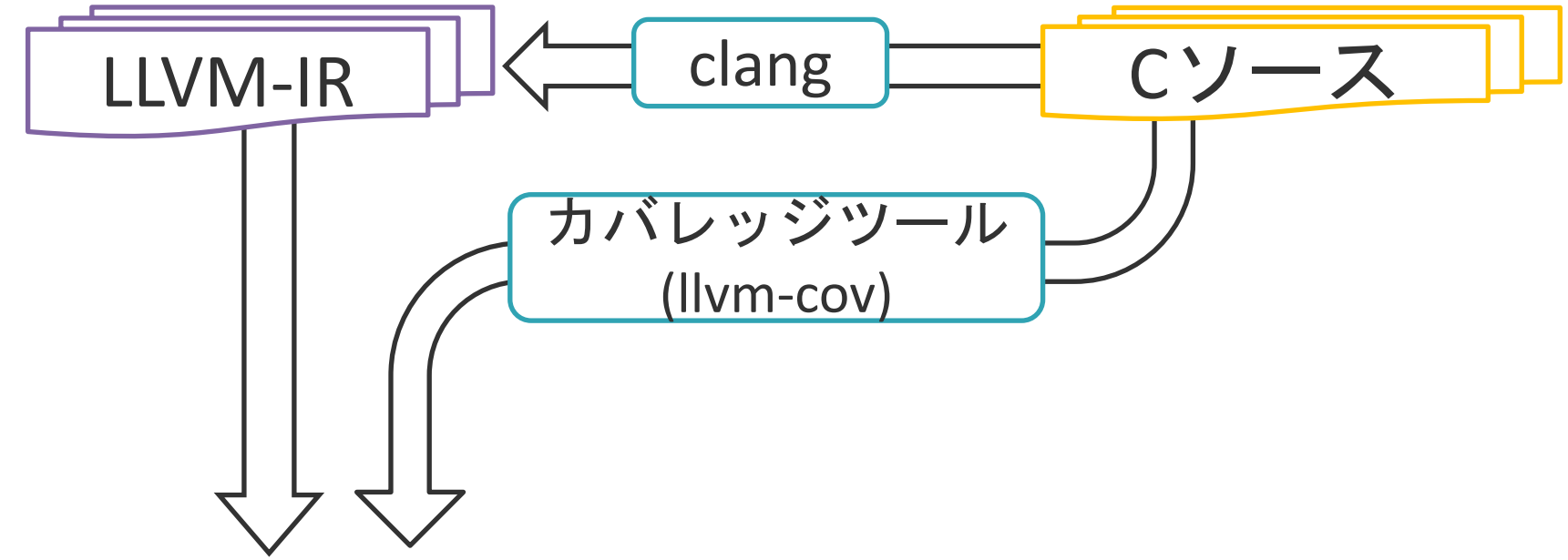
性能見積の課題（精度悪化要因）

- 古くから知られている課題
  - A) メモリシステム（例：キャッシュの影響）
  - B) ハード最適化（例：アウトオブオーダー実行）
  - C) コンパイラ最適化（例：複数プログラム文最適化）
  - D) 最適化ライブラリ（例：画像処理）
  - E) 動的要因（例：ループ回転数、分岐確率）
  - F) 周辺（例：割り込み）
- SHIM (LLVM) によって新たに出現した課題
  - G) 命令セットの違い
  - H) コンパイラの違い
  - I) LLVMにおける無限レジスタ数の影響
  - J) SHIMにおけるアーキテクチャ抽象化の影響

従来の取組

- 精度向上に向け、様々な取組を実施
  - 命令レイテンシの計測手法
  - 命令レイテンシとプロファイルを用いた見積手法
  - メモリアクセスの考慮
  - SHIMulator (SHIMによる動的性能見積)
  - SHIM2.0
- 本資料の付録、EMC WWWイベントページの2016年のマルチコアサミット資料をみてください。

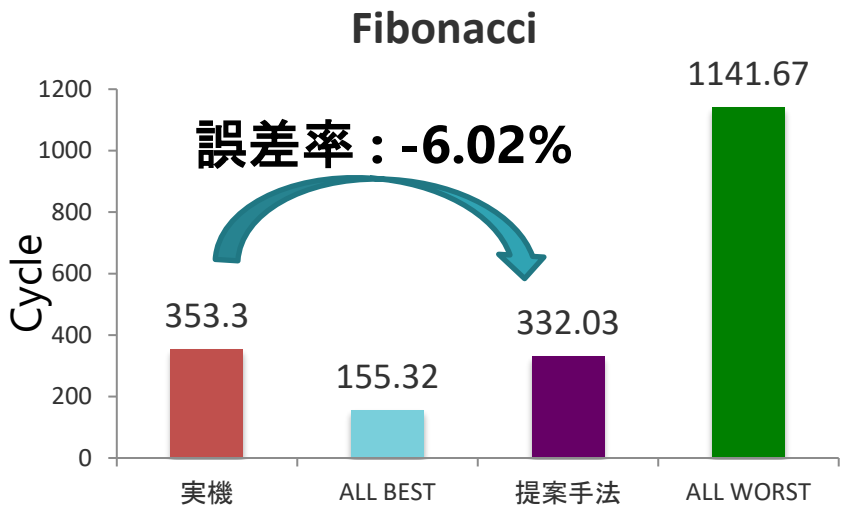
プロファイルを用いた見積手法

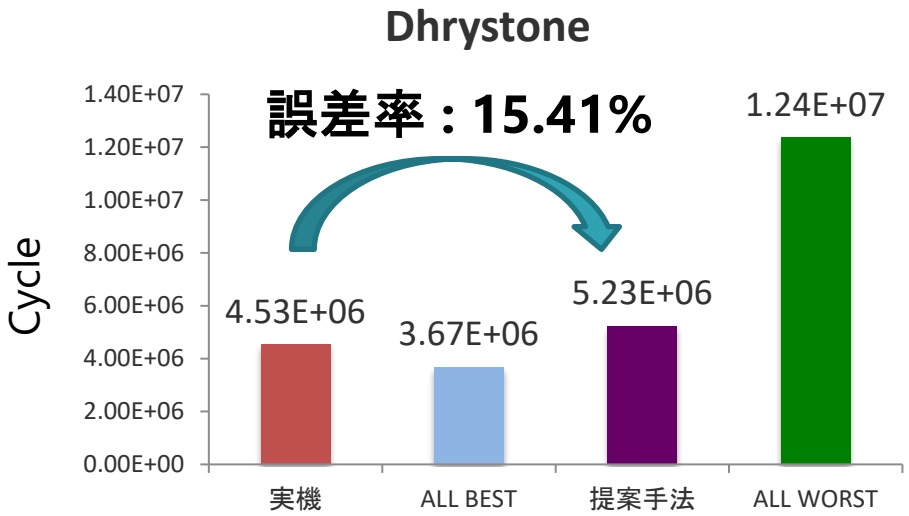


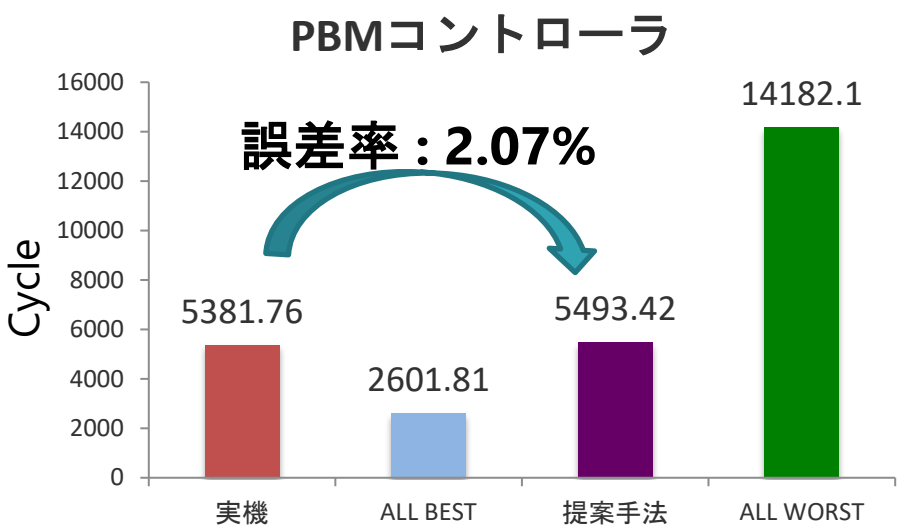
$$\sum IR_i\text{実行回数} \times IR_i\text{レイテンシ} = \text{実行サイクル}$$

SHIM

精度評価実験







提案手法は実機と比較して

平均絶対誤差 7.83%

上流工程としては**精度の高い見積**

誤差の要因

- 未計測の命令が存在(擬似命令)
- 標準ライブラリの考慮　など

アジェンダ

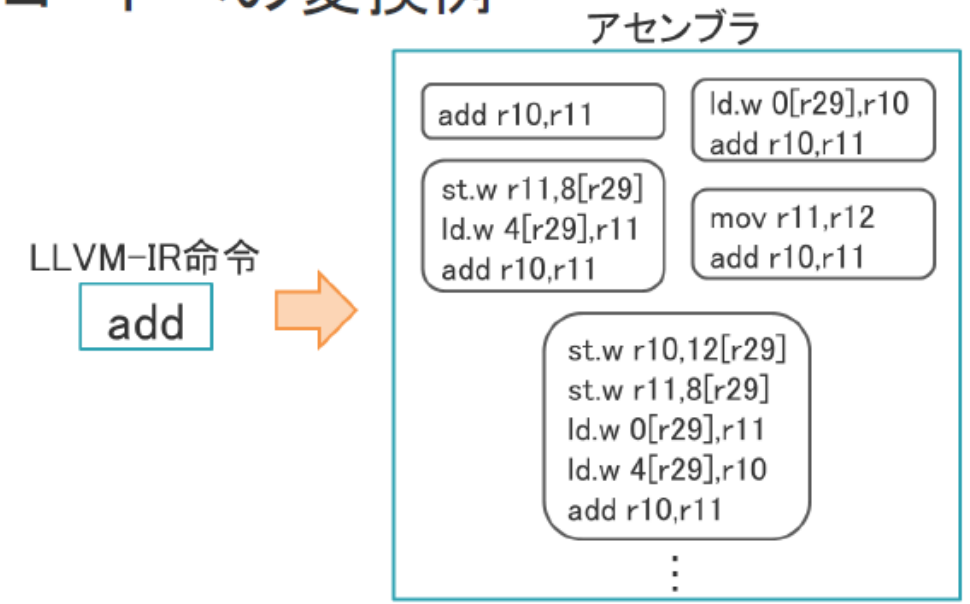
- SHIMとは
- SHIM・LLVM-IR・性能見積手法
- 最近の取組（回帰分析を用いたレイテンシ計測）
- まとめと今後の課題

さらなる課題

- 命令レイテンシをアセンブラプログラムによって求めている
- 一つのLLVM-IR命令に複数種類のターゲット命令列が対応
- SHIM2.0における拡張はシミュレーションによる精度向上には適するが、設計初期段階・上流工程での適用に課題

一つのLLVM-IR命令に複数種類のターゲット命令

- LLVM-IRでは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
  - 下図はRH850/E1M-S2(ルネサスエレクトロニクス社)のアセンブラコードへの変換例



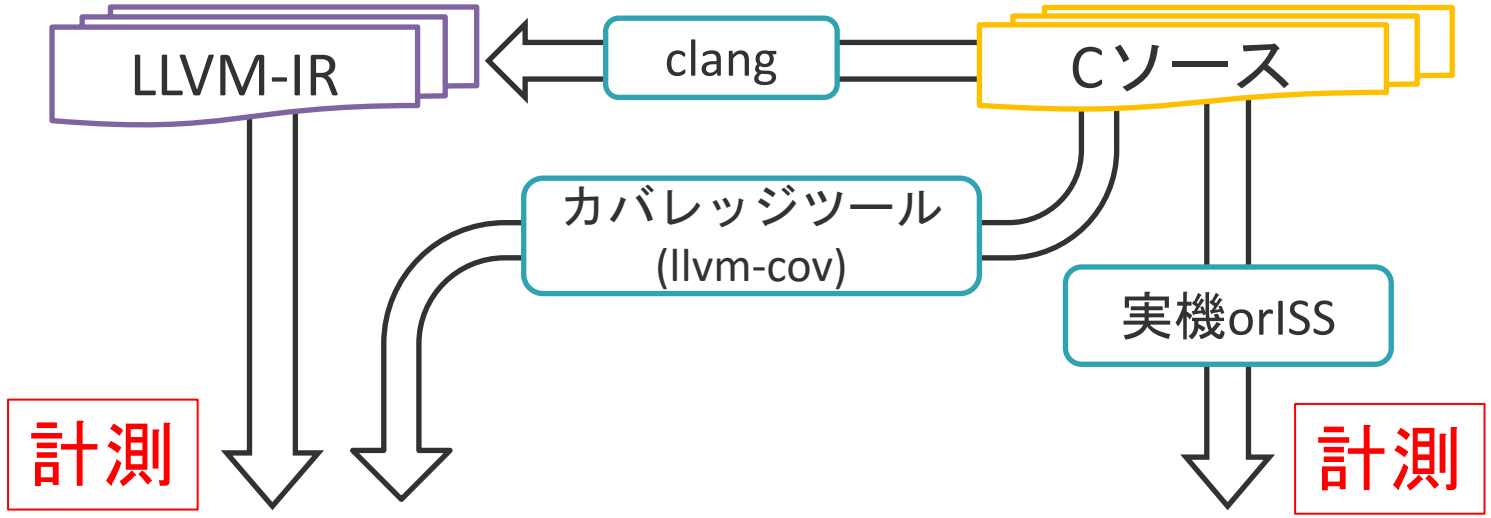
ターゲットコードごとの出現確率の考え方が難しい
種類が増えると計測に手間がかかる

SHIM1.0と2.0

- SHIM1.0
  - パイプラインやメモリアーキテクチャの詳細記述がなく、各LLVM-IR命令のレイテンシをBest-Typical-Worstで表現
  - 性能見積は命令数×レイテンシであり、計算が容易
  - Best, Typical, WorstのMixが課題
- SHIM2.0
  - パイプラインやメモリアーキテクチャの記述が可能であり、動的手法（パイプラインシミュレーション、メモリシミュレーション、バスシミュレーション等）を併用して見積もることができる
  - 例えば、レイテンシ表現はTypicalに正常時(hit)の遅延を記載し、例外時(miss)の性能は別シミュレーション結果を加算して見積もる方法が可能
  - 詳細設計時ならば可能であるが、設計初期段階・上流工程での適用が課題

提案する命令レイテンシ計測手法

- 最小二乗法を用いた命令レイテンシ計測



$$\sum IR_i\text{実行回数} \times IR_i\text{レイテンシ(変数)} = \text{実行サイクル}$$

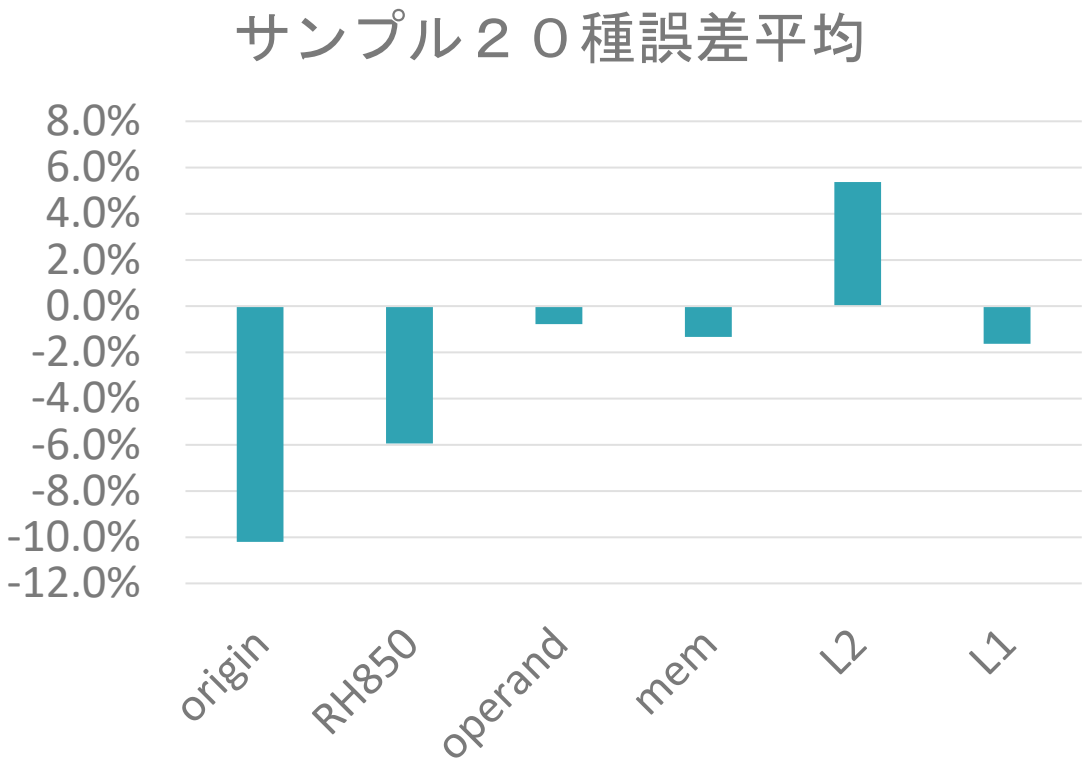
多数のCプログラムを用いてこの式を複数用意し、各LLVM-IR命令のレイテンシを変数とする誤差最小問題として解く

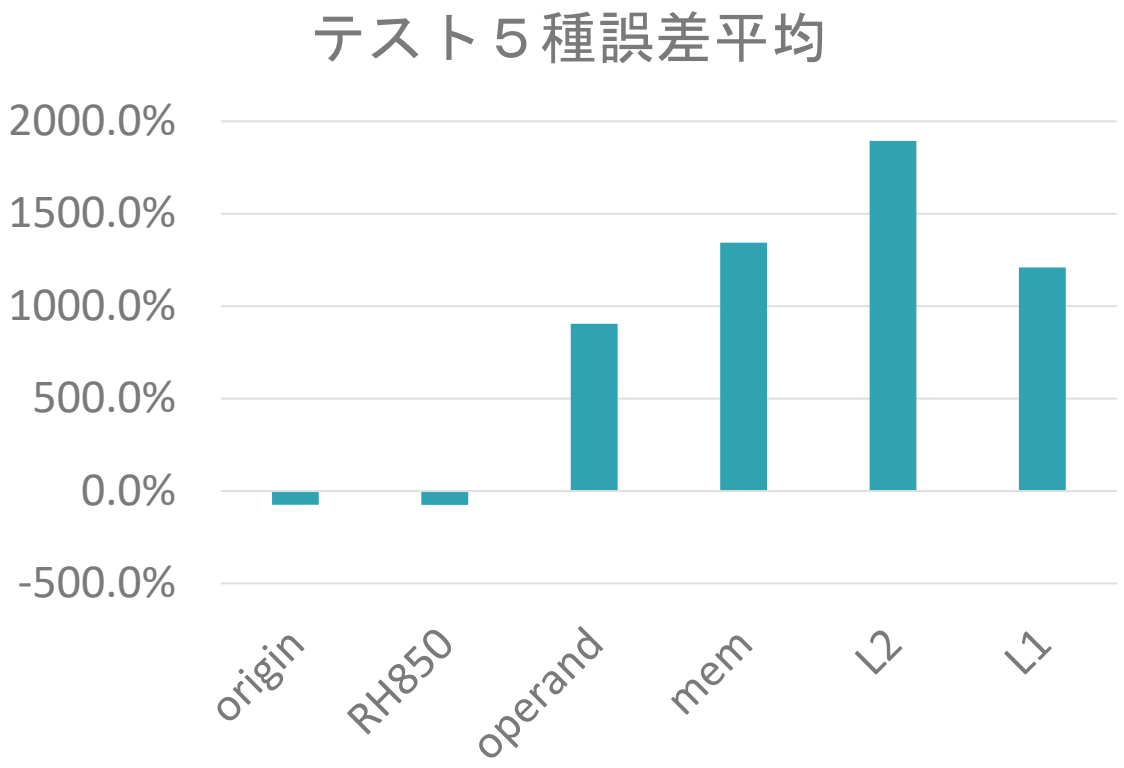
課題に対するアプローチ

- 命令レイテンシをアセンブラプログラムによって求めている
  - アセンブラプログラムは不要。C言語などを扱う
- 一つのLLVM-IR命令に複数種類のターゲット命令列が対応
  - 多数のプログラムから誤差最小２乗法により計算
- SHIM2.0における拡張はシミュレーションによる精度向上には適するが、設計初期段階・上流工程での適用に課題
  - 性能見積計算はSHIM1.0と同様

SHIM見積への適用

- 各分類でSHIM見積を行った
  - 見積対象はサンプルプログラム２０種＋テストプログラム５種



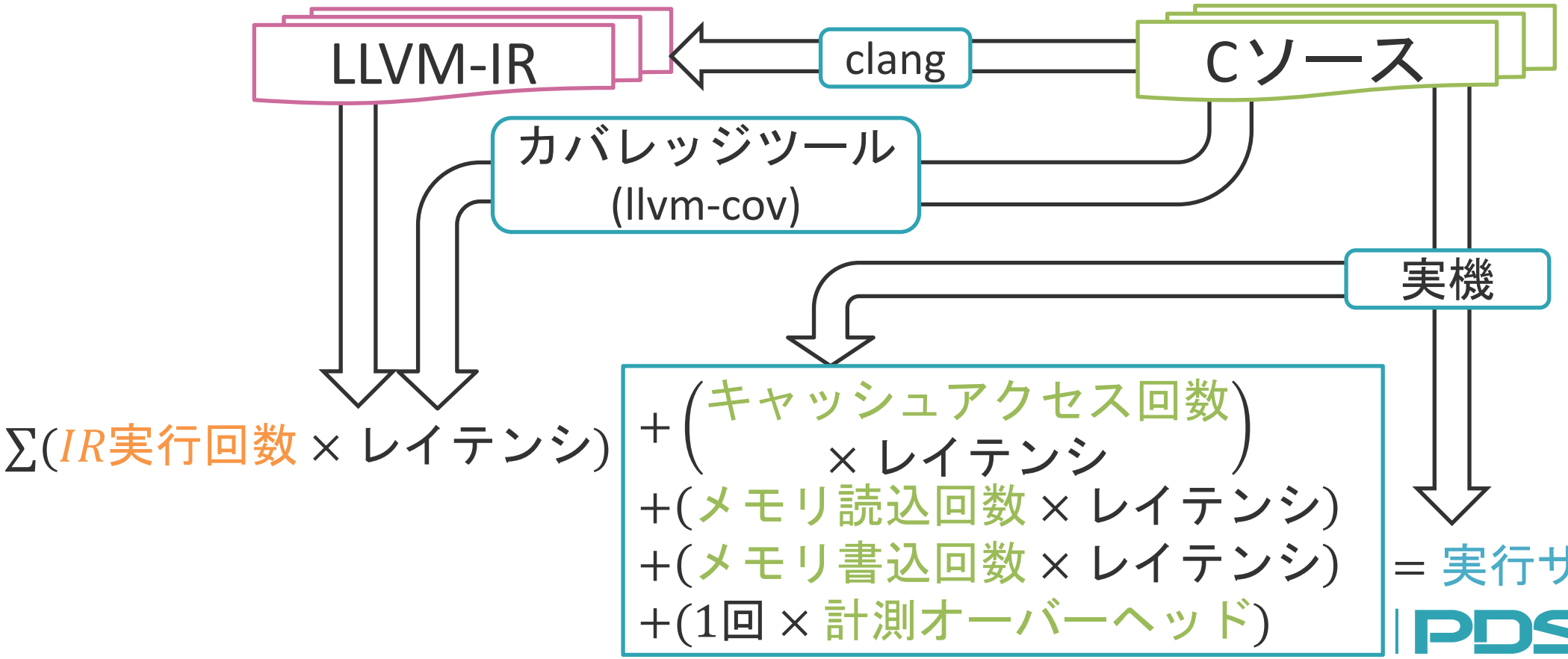


さらなる改善

段階	実施した改善
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
9	mul命令の分離
10	正規化
11	LLVM-IR命令すべてを分類

段階1・7 変数の追加

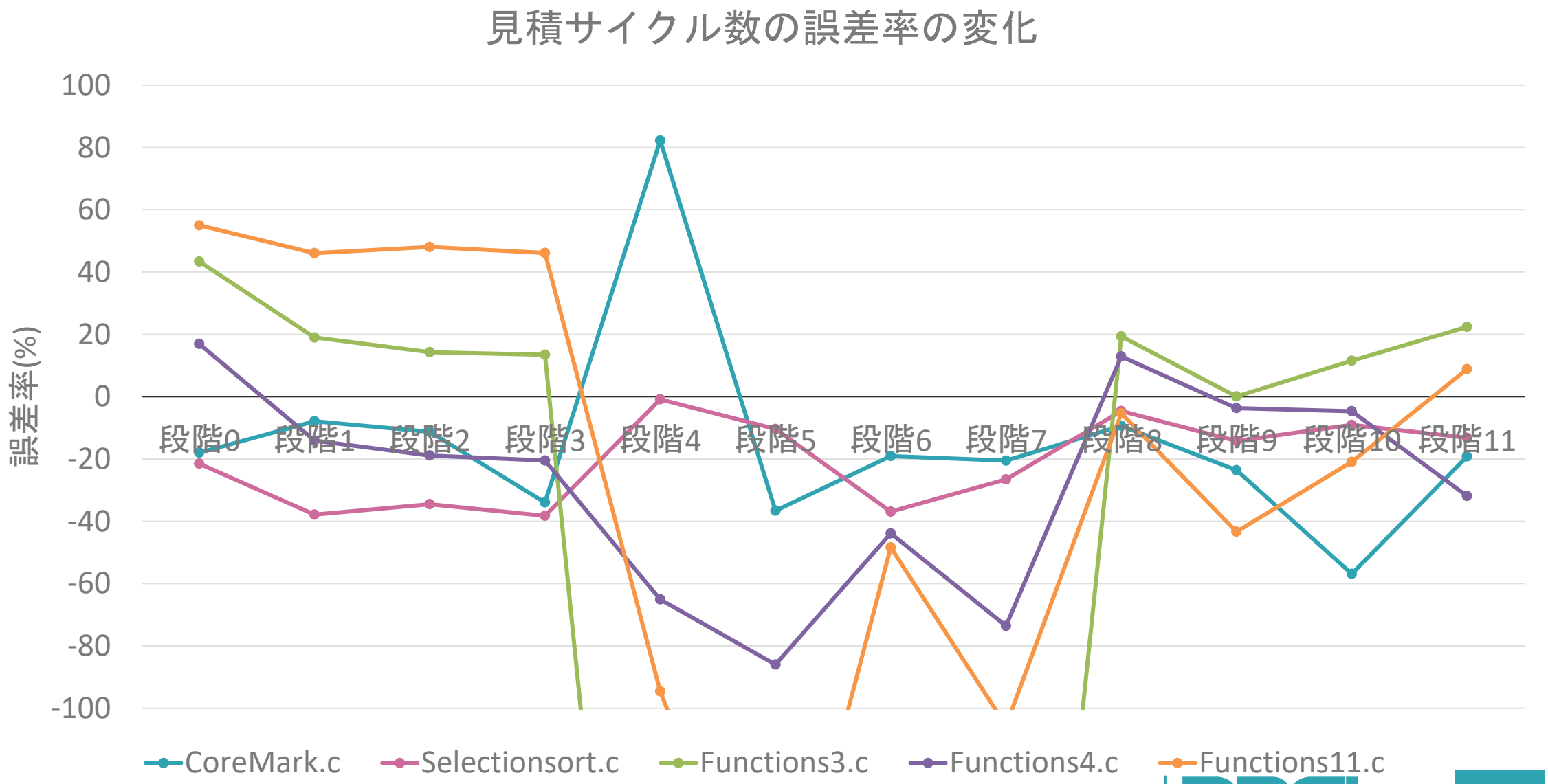
キャッシュ/メモリアクセスやオーバーヘッドを考慮して回帰式を改修
⇒性能見積時には、キャッシュミス率、レジスタスピル率などの
　パラメタを対象ソフトウェアごとに与えることを仮定



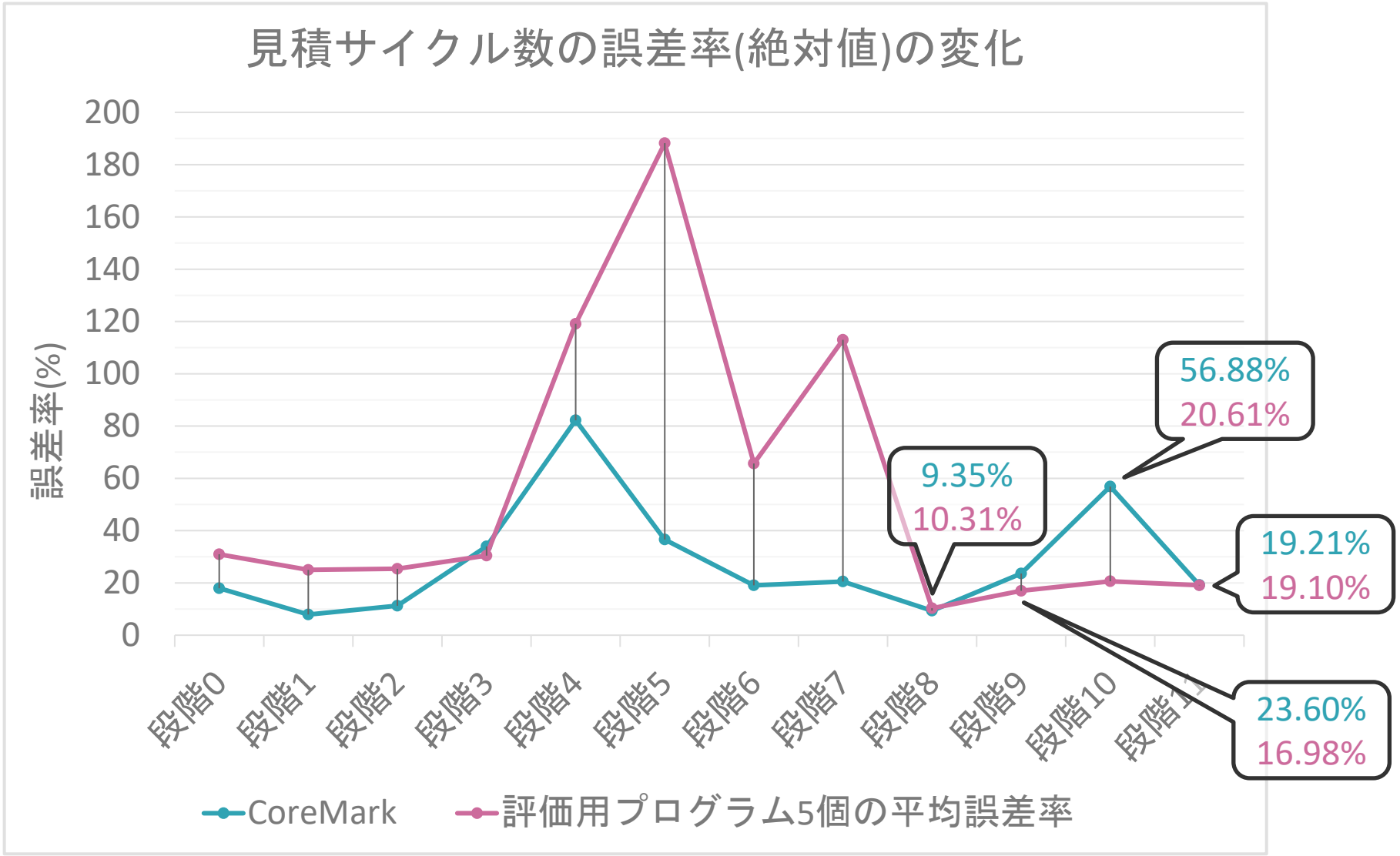
提案手法のアプローチ

- 性能計算は、SHIM1.0の３値(Best, Typical, Worst)のMixとせず、SHIM2.0と同様「正常時レイテンシ＋例外時レイテンシ＊例外確率」とする
- ターゲットソフトウェアのmiss rate等が外から与えられると仮定し、シミュレーションせずに見積もる
  - 例えばLoadのレイテンシであれば、
  - SHIM上のLoad Latency（SHIM標準内）の他にmiss penalty Latency（SHIM標準外）の存在を仮定し、
    Load命令数 * (hit rate * Load Latency + miss rate * miss penalty Latency) により見積もる

性能評価結果（１）


見積サイクル数の誤差率の変化
誤差率(%)
100
80
60
40
20
0
-20
-40
-60
-80
-100
段階0
段階1
段階2
段階3
段階4
段階5
段階6
段階7
段階8
段階9
段階10
段階11
CoreMark.c
Selectionsort.c
Functions3.c
Functions4.c
Functions11.c

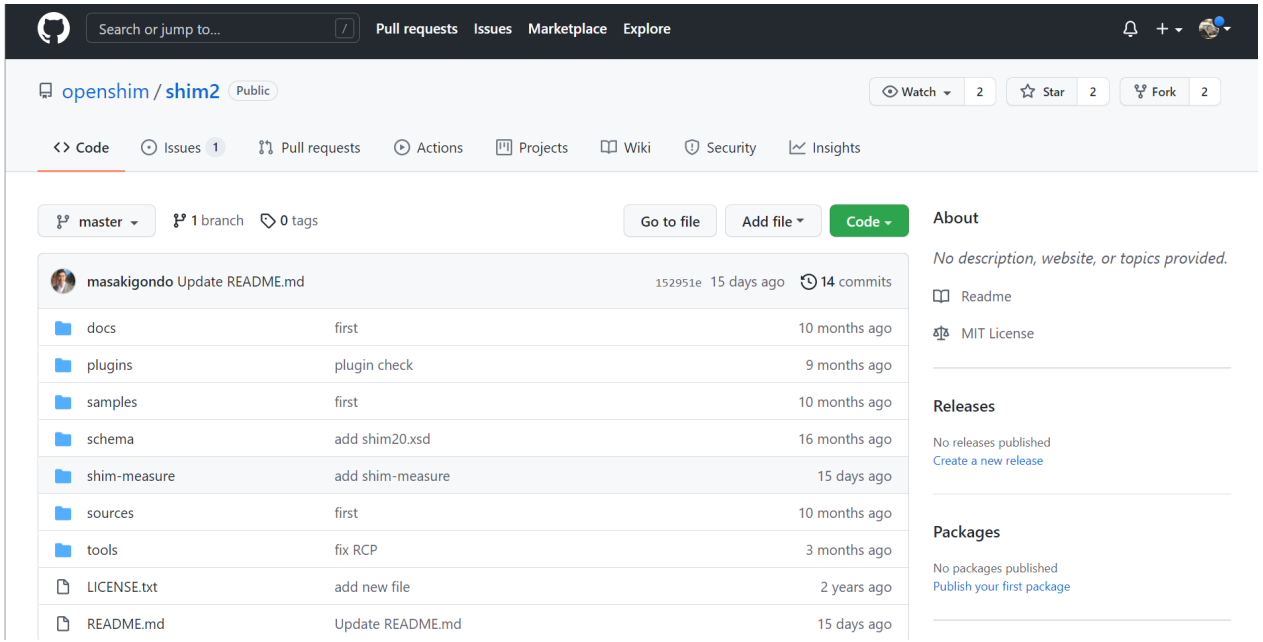
性能評価結果（２）


見積サイクル数の誤差率(絶対値)の変化
誤差率(%)
200
180
160
140
120
100
80
60
40
20
0
段階0
段階1
段階2
段階3
段階4
段階5
段階6
段階7
段階8
段階9
段階10
56.88%
20.61%
9.35%
10.31%
19.21%
19.10%
23.60%
16.98%
CoreMark
評価用プログラム5個の平均誤差率

計測プログラムのオープン化

- 遅延値見積の流れをオープンに（ターゲット：ラズパイ）
  - GitHubで”openshim”を検索
  - SHIM2のリポジトリで公開

- 見積の流れを動画化
  - https://youtu.be/rqpZ 3VtBH4



まとめと今後の課題

- SHIMの考え方、概要、性能見積の課題、取組について紹介
- 今後のSHIM、研究課題
  - SHIM3.0・・・基本ソフトウェアの通信レイテンシ
  - ライブラリ、システムコールのレイテンシ
  - SHIMの抽象度と精度のトレードオフ
    - 将来アーキテクチャでも容易に記述可能であることが重要
    - 高精度が必要ならば半導体ベンダ提供の環境を使うべき
    - 今回、キャッシュヒット率などを追加したが、どのパラメタがあれば、どの程度の精度になるのか、といった研究が必要

付録

方式と課題への対応

	A メモリ	B ハード	C コンパイラ	D Lib	E 動的要因	F 周辺	G 命令セット差	H コンパイラ差	I レジスタ数	J アーキ抽象化
ターゲット環境利用静的解析	要	要	要	要	要	要				
ターゲット環境利用動的解析	ー	ー	ー	ー	要	要				
SHIM利用静的解析	要	要	要	要	要	要	要	要	要	要
SHIM利用動的解析	要	要	要	要	要	要	要	要	要	要

要：要対応
ー：対応不要（ただし実行環境の持つ精度により対応が必要な場合あり）
斜線：考慮不要

方式と課題への対応（A～Fについては従来から多くの議論があるため本講演の対象外とする。SHIM環境はターゲット環境の±20%精度が目標であることに注意）

	A メモリ	B ハード	C コンパイラ	D Lib	E 動的要因	F 周辺	G 命令セット差	H コンパイラ差	I レジスタ数	J アーキ抽象化
ターゲット環境利用静的解析	要	要	要	要	要	要				
ターゲット環境利用動的解析	ー	ー	ー	ー	CS	要				
SHIM利用静的解析	S2/AL	要	要	S2	要	要	LP	LP	AL	S2/AL
SHIM利用動的解析	S2	S2	SH	S2	SH/CS	要	SH	SH	S2	S2

要：要対応
ー：対応不要（実行環境精度依存）
斜線：考慮不要

SHIM2.0では電力見積も検討中

LP: 命令レイテンシとプロファイルにて対応
AL: アクセスログ解析にて対応
SH: SHIMulatorで対応
S2: SHIM2.0で検討中
CS: 協調シミュレーションにて対応

変数削減のため命令を分類

arithmetic	arithmetic_d	div	div_d	float	float_d
add	add_d	sdiv	sdiv_d	fadd	fadd_d
sub	sub_d	srem	srem_d	fsub	fsub_d
mul	mul_d	urem	urem_d	fmul	fmul_d

fdiv	fdiv_d	load	load_d	store	callret	others
fdiv	fdiv	load	load_d	store	call	その他の命令
fcmp	fcmp				ret	

アジェンダ

- SHIMとは
- SHIM・LLVM-IR・性能見積手法
  （2016年のマルチコアサミット発表から）
  – 命令レイテンシの計測
  – 命令レイテンシとプロファイルを用いた見積
  – メモリアクセスの考慮
  – SHIMulator (SHIMによる動的性能見積)
  – SHIM2.0
- 最近の取組（回帰分析を用いたレイテンシ計測）
- まとめと今後の課題

命令レイテンシの計測

- Bestケース
  - 測定対象の命令機能のみが差分で出現するように調整
  - パイプラインの実行ステージのサイクル数に近い値

Base
C ：uiz=uix

```
st.w    r10, 28[sp]
st.w    r10, 32[sp]
ld.w    32[sp], r10
st.w    r10,36[sp]
```

Target
C ： uix=uiy+UiValue

```
st.w    r10, 28[sp]
st.w    r10, 32[sp]
ld.w    32[sp], r10
add     3 ,r10
st.w    r10,36[sp]
```

BaseとTargetをそれぞれ複数回測定し、差分を反復回数で割ることにより計測

命令レイテンシの計測

- Worstケース
  - 使用する変数をvolatile宣言
  - 前後命令から依存関係を切る
  - ローカルに配置されているスタックにアクセス

Base
C ：EMPTY();

該当部なし

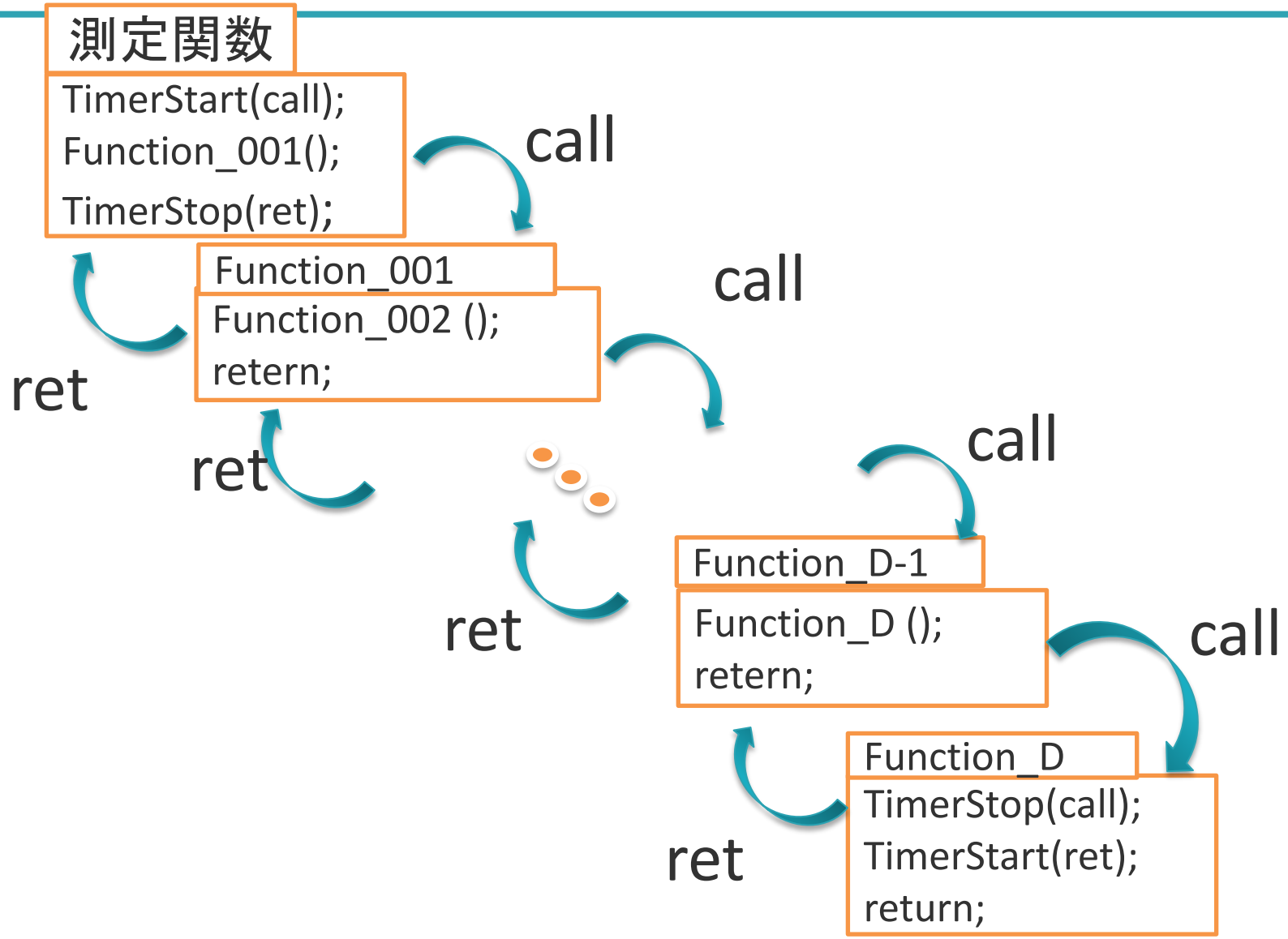
Target
C ： uiz=uiy+uix

```
st.w    r11, 28[sp]
st.w    r10, 32[sp]
ld.w    28[sp], r11
ld.w    32[sp], r10
add     r11 ,r10
st.w    r10,36[sp]
```

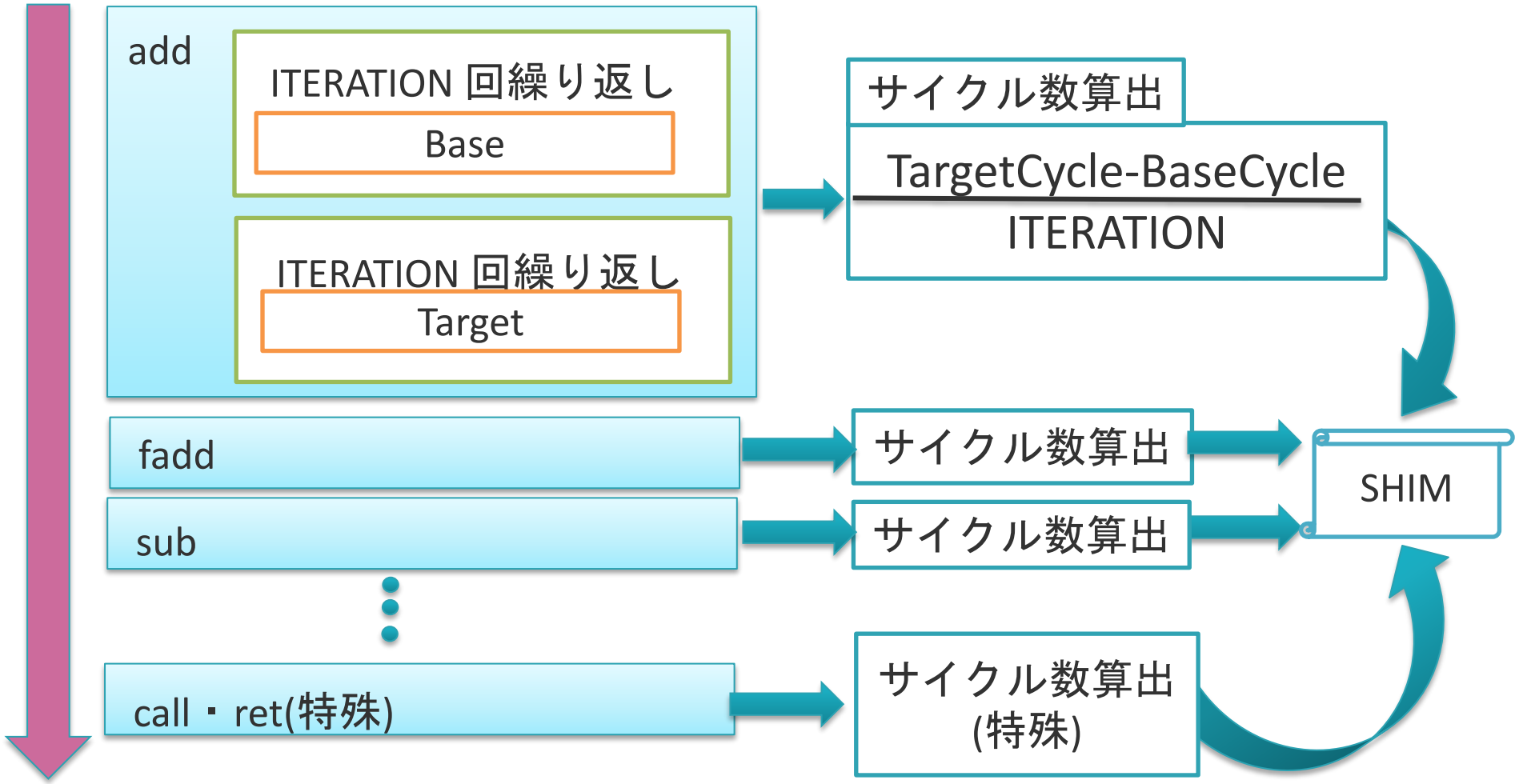
命令レイテンシの計測

- **メモリアクセス命令**
  - Best :ローカル領域にアクセス
  - Worst : グローバル領域にアクセス
- **型変換命令**
  - Best : メモリ格納の際に隠蔽される場合
  - Worst : ロードやストアでも隠蔽できない場合
- **Call・Ret命令**
  - 単一のレイテンシを使用(計測法は次スライド)

Call命令・Ret命令の計測

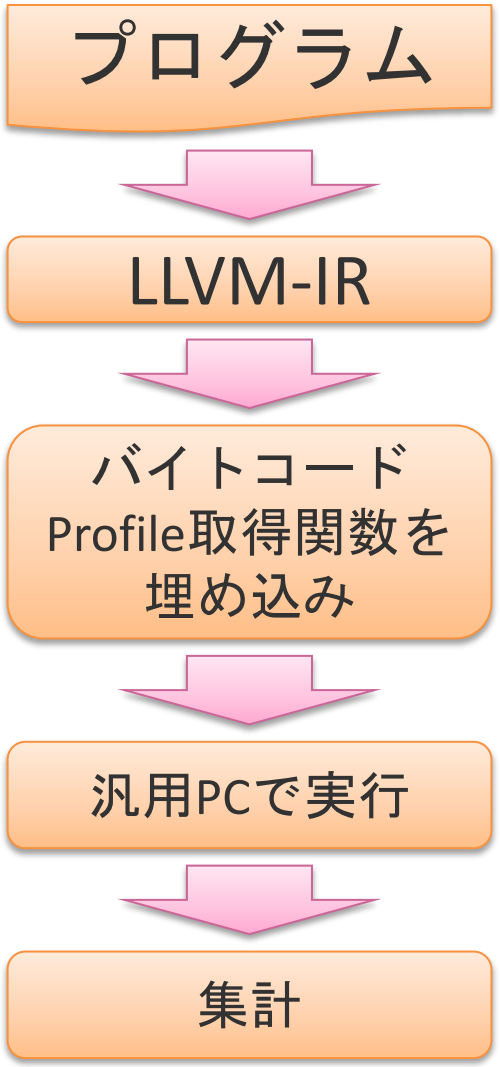


命令計測プログラムの全体像



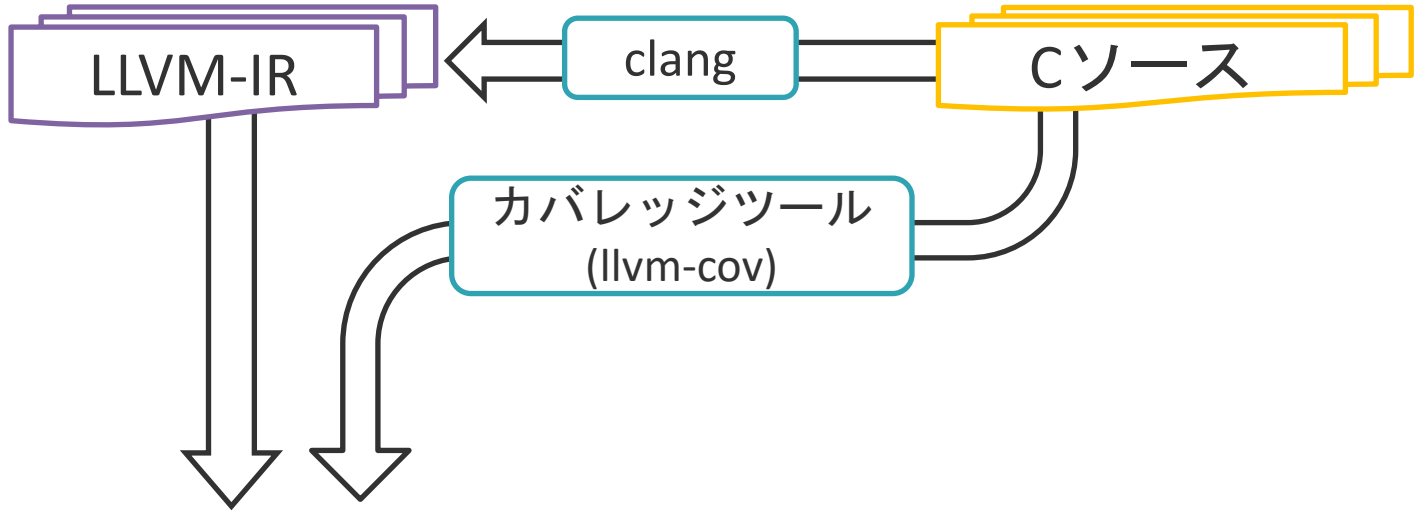
命令レイテンシとプロファイルを用いた見積

- ホストマシンで実行したアプリケーションのプロファイル情報から実行回数を取得し、静的手法で見積
  - 注：当初はllvm-profを用いていたが、現在のバージョンのllvmにはllvm-profは存在しないため、llvm-covで実現



[*]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. 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.

命令レイテンシとプロファイルを用いた見積



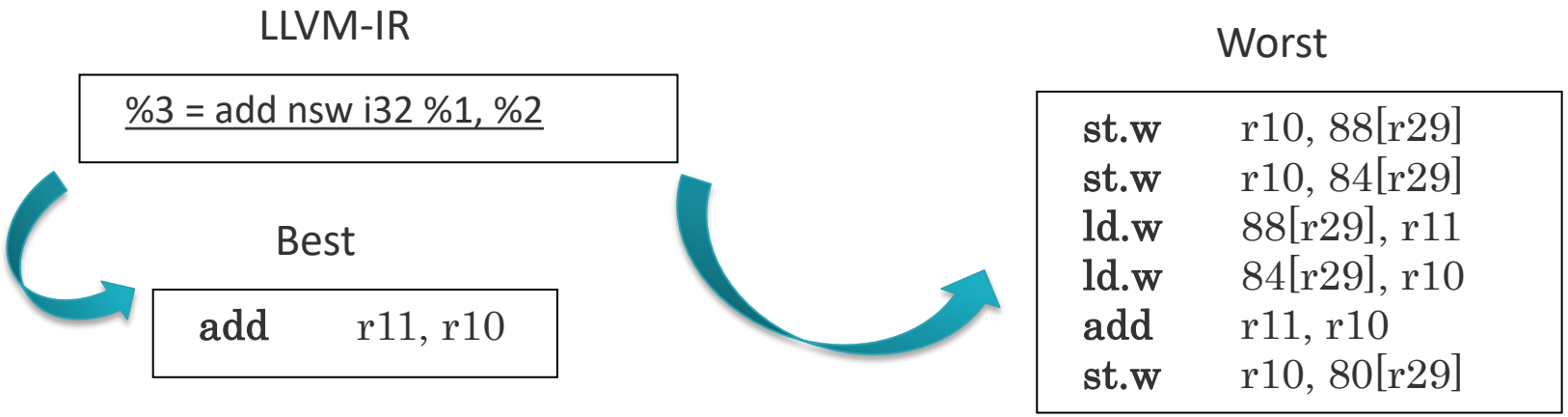
$\sum IR_i$実行回数 $\times IR_i$レイテンシ = 実行サイクル

SHIM

メモリアクセスを考慮したレイテンシ見積

- ローカルアクセス問題 (LLVMの問題)
  - グローバル変数に対してはload/store命令となるが、無限レジスタ数のため、実環境におけるスタック領域アクセス（以降ローカルアクセスとよぶ）はLLVM-IR上はメモリアクセスにならない

ローカルアクセスを考慮したレイテンシ見積



- LLVM-IRの重要な特徴と性能見積時の仮定
  - ローカルメモリ(スタック)領域へのアクセスは出現しない
  - アプリと実行環境によりローカルアクセス割合は異なる
    - アプリケーション依存＆ターゲット依存
  - この特徴を考慮することによって見積精度向上を測る

表　ローカルアクセスの割合

アプリケーションA	アプリケーションB
24.97[%]	18.00[%]

ローカルアクセスを考慮したレイテンシ見積

見積に最適なTypicalレイテンシを計算

⇒**Best**レイテンシと**Worst**レイテンシから
ローカルアクセス比率で決定

提案するTypicalレイテンシ

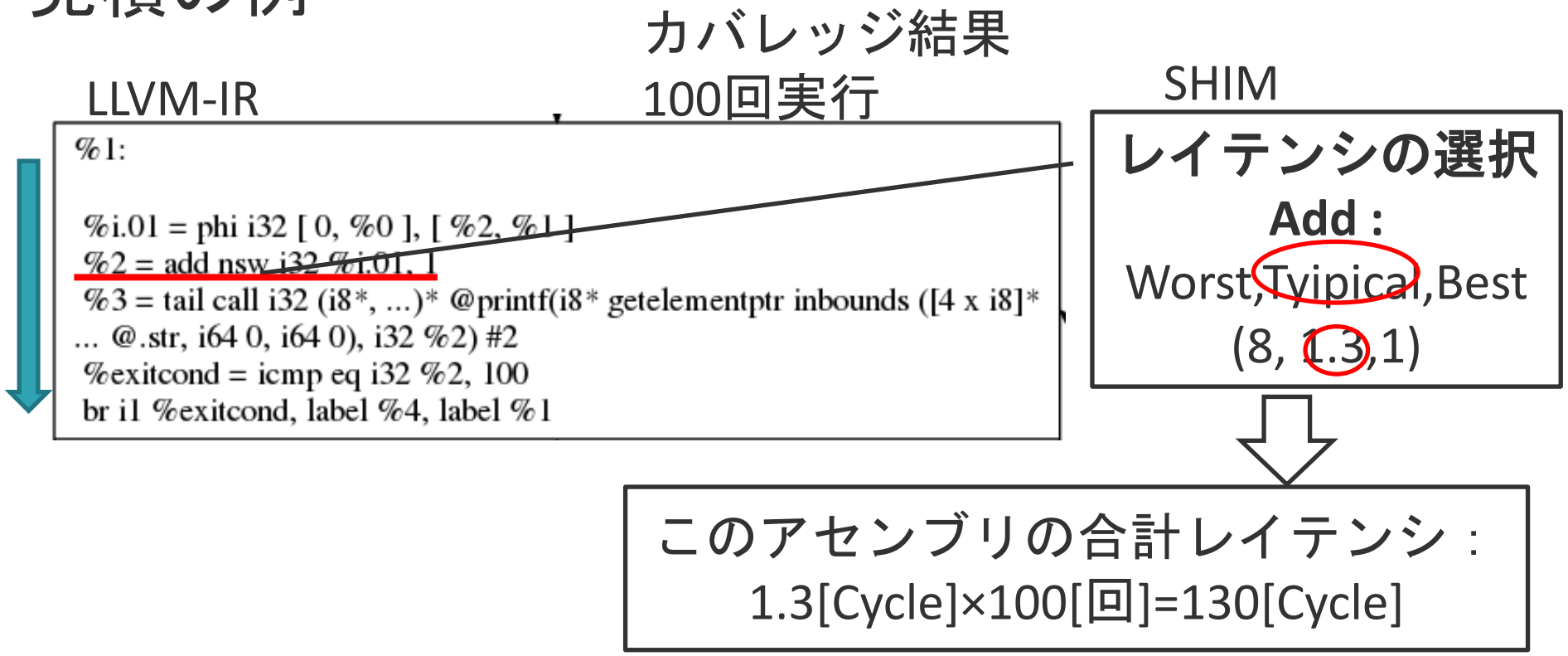
Typical = Best(1-Local)+Worst(Local)

$$\text{Local}[\%] = \frac{\text{ローカルメモリアクセス回数}}{\text{総命令数} - \text{グローバルメモリアクセス回数}}$$

LLVM-IRとアセンブリで1:1対応されている

見積の集計方法

見積の例



全てのLLVM-IRの命令に対して実行

⇒アプリケーション全体の総和が見積サイクル数

性能見積手法 ⇒基本的な考え方、課題はよく知られている

- **静的手法**
  - LLVM-IR解析⇒性能見積
    - ループの実行回数やメモリアクセスが不明
    - 命令レイテンシ(Best, Typical, Worst)の選択が困難

SHIMに限らない課題。
固定回転数の解析、
PC環境のプロファイラの利用など

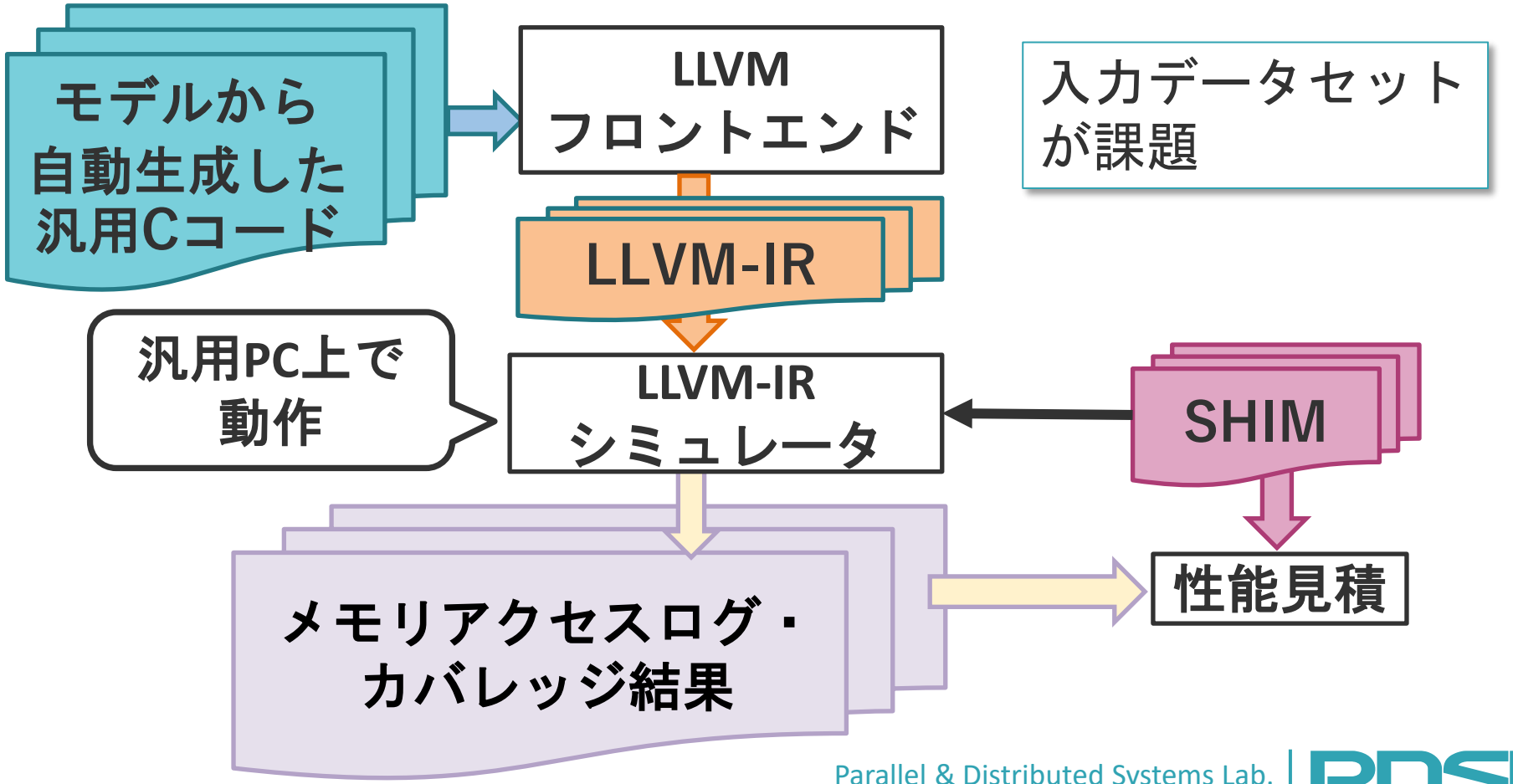
- **動的手法**
  - LLVM-IR用シミュレータ ⇒ 性能見積
    - SHIMを用いたLLVM-IRシミュレータは存在しない

SHIMを用いた静的性能見積の課題

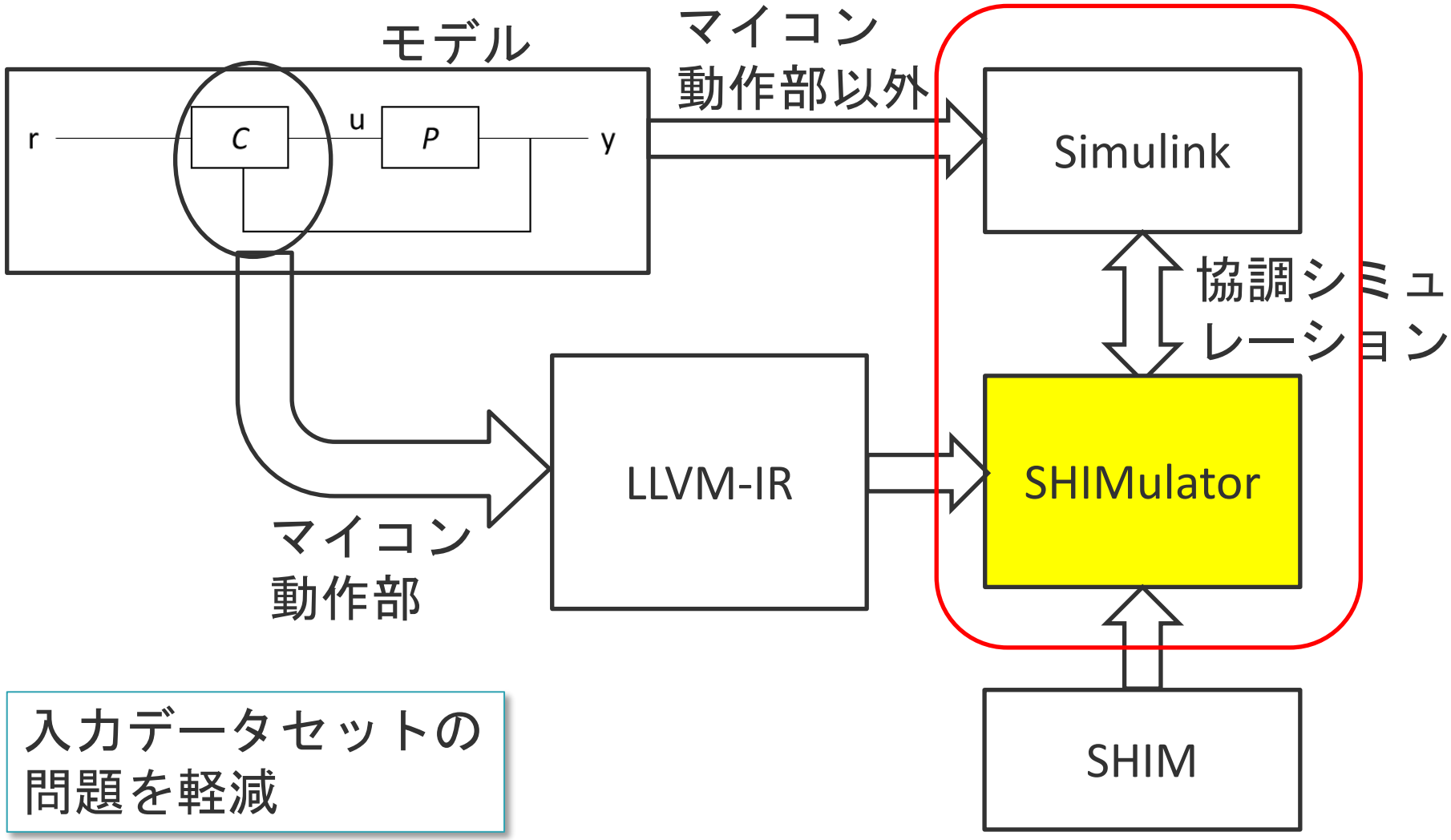
- LLVMではレジスタ数無限
  - レジスタからメモリにあふれる割合
    - アプリケーション依存、アーキテクチャ依存、コンパイラ依存
    - 今回、コンパイラが異なってもある程度合う可能性を示唆
- キャッシュミス率等も同様と考えられる
- 対策
  1. 現在のアプリ・アーキ・環境から将来の値を予測
  2. SHIMのアーキテクチャ情報、性能情報を使って動作するISSを利用

SHIMulator

- SHIMのアーキテクチャ情報、性能情報を使って動作するISS
- 動的見積、静的見積の双方に利用可能



SHIMulatorによるSPILS



入力データセットの
問題を軽減