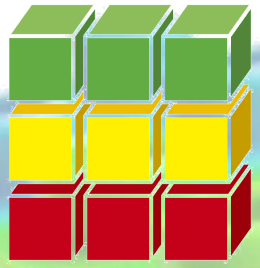
ET West併催セミナー
組込みマルチコアサミット2017West
2017年7月13日(木) 10:00-12:00
タワーC 8F カンファレンスルーム C03

- 主催　組込みマルチコアコンソーシアム
- アンケートへの記入をお願いします
- 一部の資料は、組込みマルチコアコンソーシアムWWWページのeventページにアップロードされています
- 講演
  - 10:00- 組込みマルチコアコンソーシアムについて
  - 10:10- SHIM: ソフトのための国際標準ハードウェアモデル記述
  - 10:45- MBP: モデルベース並列化ツールの実用化
  - 11:20- MCoT マルチコア・オブ・シングス？ マルチコア適用ガイド策定に向けて



Embedded Multicore Consortium

www.embeddedmulticore.org

組込みマルチコアコンソーシアム

ハードベンダ/ソフトベンダ/メーカを繋ぎマルチコア活用を支援

2017-7

名古屋大学　枝廣　正人

イーソル(株)　権藤　正樹

ガイオテクノロジー(株)　岩井 陽二

Embedded Multicore Consortium

Agenda

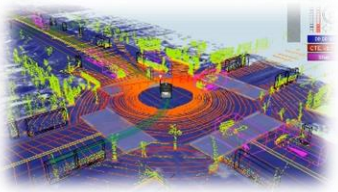
- インテリジェント化する組込みシステム
- 海外の組込みマルチコア活動
- 組込みマルチコアの課題
- 組込みマルチコアコンソーシアム
- コンソーシアム活動予定
- ロードマップ
- 入会メリット
- メンバーシップ

インテリジェント組込みシステム
自動運転システム

- 自動運転技術はロボット技術とも多くの共通要素を持つ、今後有望視されるインテリジェント組込みシステムの一つ
- 自動運転に至るまでの多くの技術が段階的に実用化されつつある
- 自動運転における認知、判断では非常に大きなコンピューティングパワーが必要
- 操舵においても、例えば欧州OEMでは3コアは量産済み、次は6コアと述べている

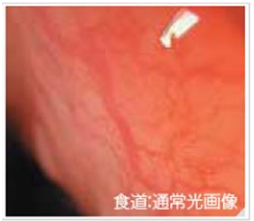
これらのセンサー情報は「センサーフュージョンによって統合しており、車両周囲の状況を正確に把握できるようにした」(同社)とする。センサーフュージョンには、マルチコアのプロセッサーを使用したという。

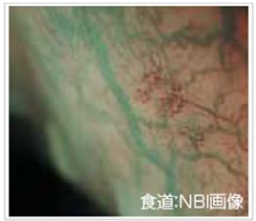
※日経エレ2014年10月13日号



インテリジェント組込みシステム
医療機器におけるリアルタイム画像処理

- 多くの先端診断支援機能が高度かつリアルタイムな画像処理技術によって実現されている
- アルゴリズムの多様化、複雑化、機器の間での部分的再利用などの開発効率向上などの課題がある
- 現在ハードウェアで実現しているものをソフトウェアで実現することで解決する課題は少なくない
- それには強力で信頼性の高いマルチコアが必要





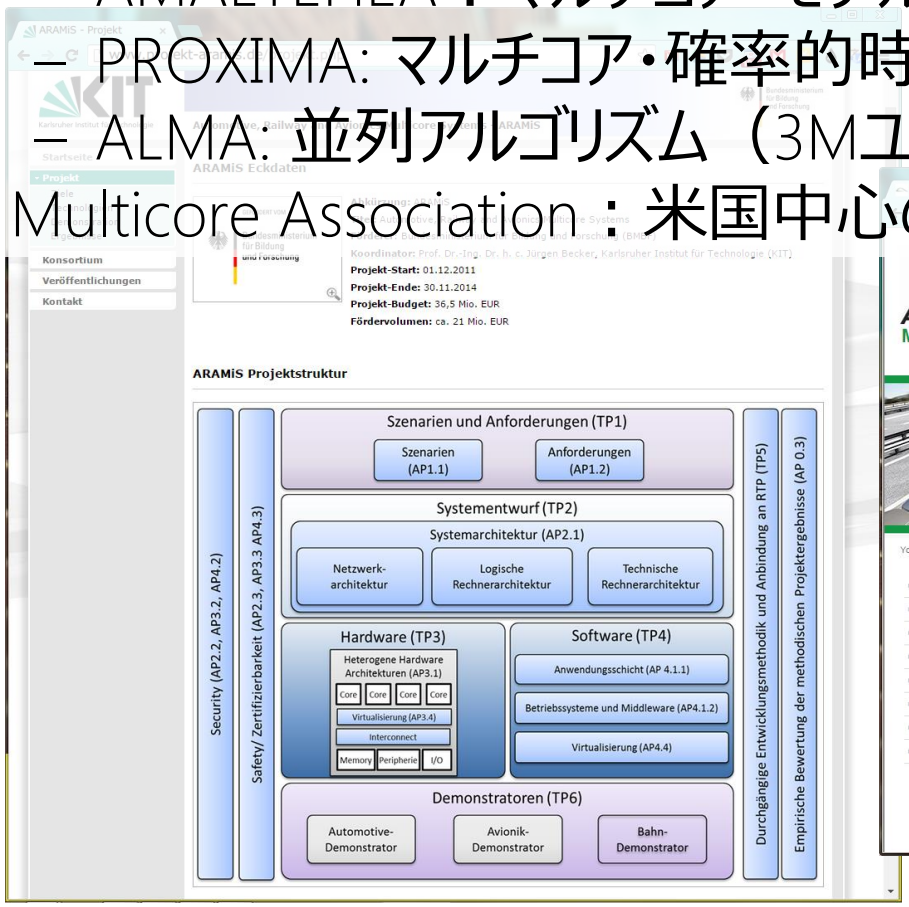
マルチコアの必要性

- 車載システム、医療機器、ロボットとあらゆる組込み機器が急速にインテリジェント化
- インテリジェンスの進化にはソフトウェアが鍵
- インテリジェントなソフトウェアは強力なコンピューティングパワーが必要
- 高い安全性が求められる組込みシステムではクラウドは「利用する」ものではあっても「依存する」ことはできない
- 組込みシステムにハイパフォーマンスなコンピューティングプラットフォームが求められている

ソフトウェアが迅速に開発でき高効率なマルチ・メニーコアが不可欠

海外での組込みマルチコア活動

- EUのマルチコア向けプロジェクト（自動車/航空/鉄道）
  - ARAMIS：安全系マルチコアプラットフォーム（36Mユーロ）
  - AMALTEHEA：マルチコア・モデルベース開発環境（8Mユーロ）
  - PROXIMA: マルチコア・確率的時間解析（7Mユーロ）
  - ALMA: 並列アルゴリズム（3Mユーロ→スピンアウト）など
- Multicore Association：米国中心のマルチコア技術標準化団体



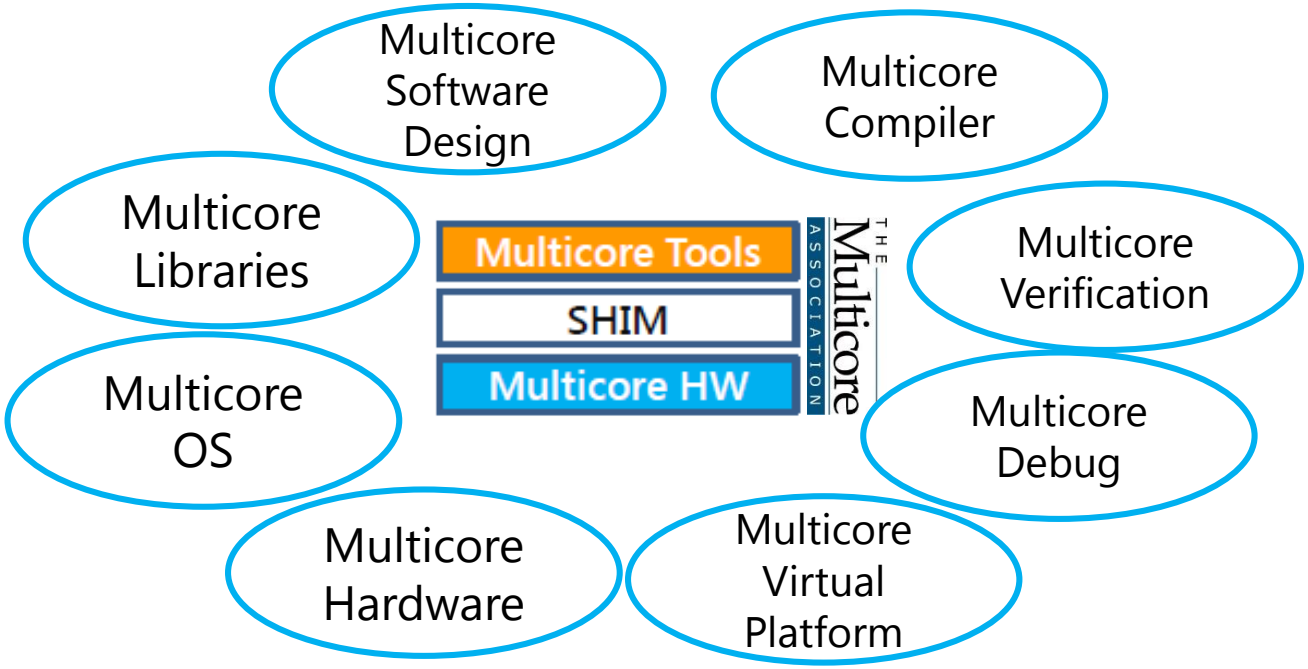
組込みマルチコアの課題

- マルチコアプロセッサはアーキテクチャの自由度が高く、各種ツールやプラットフォーム支援が重要
- 様々な並列化手法、ライブラリ、ツールを組合せるには様々な知見が必要
- システムベンダから半導体ベンダまで、すべての関連技術の協働が必要
- 関連業界で協力・連携し、(1) 活用支援、(2) ビジネス推進、(3)市場の活性化貢献を実現することが必要

様々なベンダや大学が集まり連携するための場が求められている

組込みマルチコアコンソーシアム

www.embeddedmulticore.org



- システム、ソフトウェア、ツール、半導体の各レイヤが協力・連携し、前述の課題を解決するエコシステムを構築するための産学合同の場
- 組込みマルチコアに関する技術開発加速と利用促進
- 開発フローの確立とベンダ間ツール協調を支援
- Multicore Association (MCA)とのアライアンス

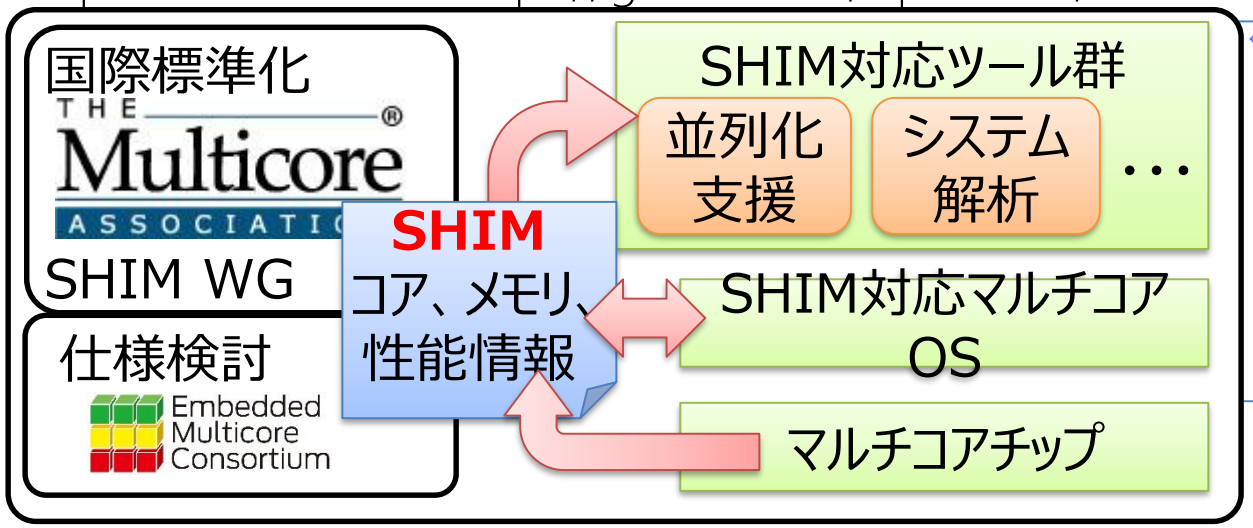
SHIMとは

Software-Hardware Interface for Multi-many-core

Multicore Tools
SHIM
Multicore HW

- 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

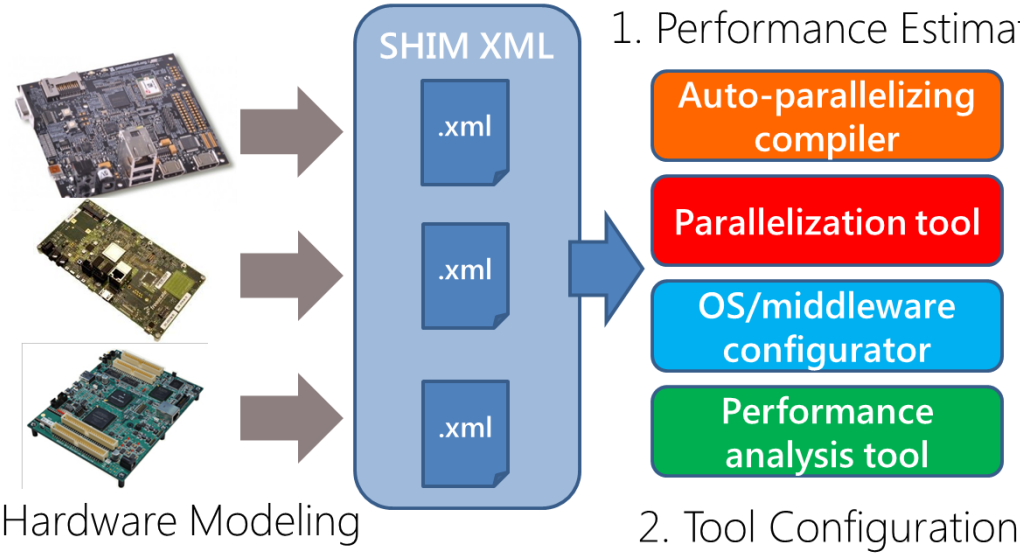
SHIM仕様書 http://www.multicore-association.org/workgroup/shim.php
Open SHIM Github https://github.com/openshim/shim




```
<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

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

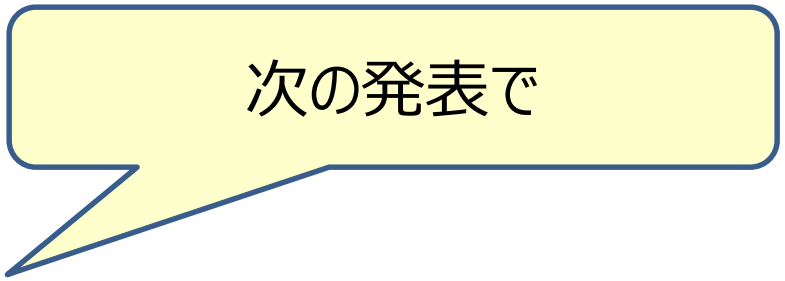
- マルチコアにおけるアプリケーション実行性能見積
- マルチコア選定時のアプリケーション実行性能比較
- 異なるマルチコアへのアプリケーション移植の際の性能見積
- 複数マルチコアをターゲットとしたソフトウェア部品開発
- 特定アプリケーション向けに特化したマルチコアを企画する際の性能評価
- マルチコア向け開発支援を行う各種ツールの開発コスト低減とSHIM対応ツールエコシステム

コンソーシアム活動

- マルチコア向け開発支援ツールのためのハードウェア抽象化記述SHIM標準化と導入支援（SHIM委員会）
  - SHIM (Software-Hardware Interface for Multi-Many-Core)
  - SHIM WG, Multicore Association (Chair: M. Gondo (eSOL))
  - NEDO省エネPJから仕様提案、2015年2月公開
- リファレンスとしてSHIMを利用したマルチコア向け設計支援ツール群を開発
  - MCAとしても公開するSHIM Editorと性能計測ツールに加え、設計支援ツール群を会員向けに無償公開予定。所定の期間経過後に一般にも公開する可能性有
  - モデルベース並列化委員会
- 様々な並列化手法の知見共有とガイドラインの検討
  - マルチコア適用委員会
- セミナー開催、技術情報提供

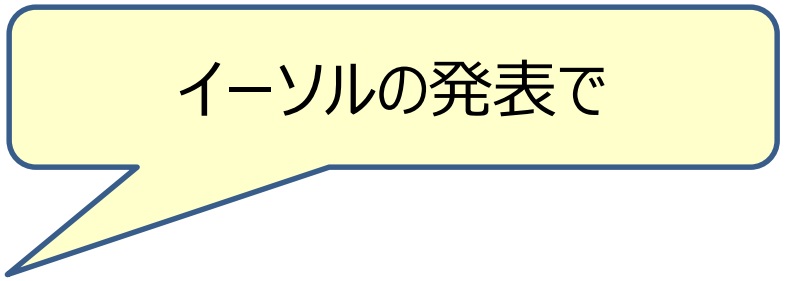
SHIM委員会

- WG構成
  - 委員長：権藤（イーソル）
- 活動計画
  - 定例委員会を開催
- 期間：2015/3～2018/3（原則として継続）
- 対象：SHIM仕様及びその適用
- 活動内容
  - SHIMの適用支援：SHIMの利用を検討しているメンバへの適用支援
  - SHIM v2.0に向けた検討



モデルベース並列化委員会 (MBP)

- WG構成
  - 委員長：枝廣（名大）
- 活動計画
  - 定例委員会を開催
- 期間：2015/3～2018/3（原則として継続）
- 対象：Simulinkモデルベースからマルチコア向けの設計方法論
- 活動内容
  - Simulinkモデルから抽出したグラフ構造（現状XML）の仕様化（標準化）
  - 上記をもとにした設計フローの構築



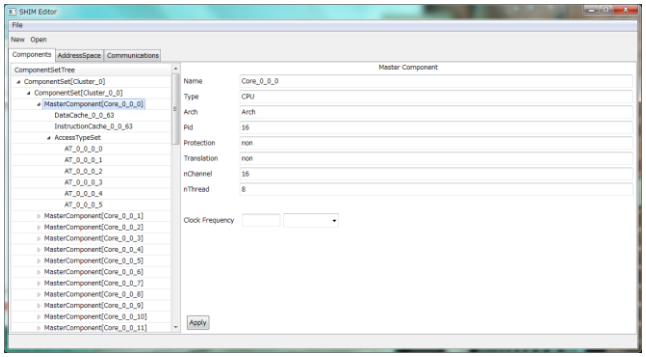
マルチコア適用委員会 (WG3)

- WG構成
  - 委員長：岩井（ガイオテクノロジー）
- 活動計画
  - 定例委員会を開催
- 期間：2017/1～2018/3（原則として継続）
- 対象：マルチコアを積極的に活用する方法やマテリアル
- 活動内容
  - 詳細ガイドの作成・配布
  - 手順書やテンプレートの作成・配布
  - ツール要件（集）の検討および情報配信
  - 実証 PJ やツール試作および提供

ガイオテクノロジーの発表で

会員向け公開成果 (1)

- SHIM利用サンプルプログラム類
  - アクセス関数、SHIM利用性能見積サンプルプログラム
  - SHIM Editor, 性能計測ツール（評価環境での性能自動計測、SHIMへの自動入力）



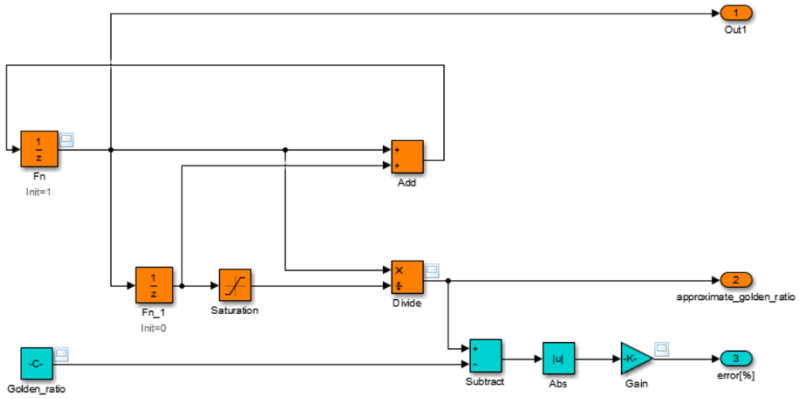
SHIM Editor

```
<Performance>
  <accessTypeRef>Load_Algined_Halfword</accessTypeRef>
  <Pitch worst="10.0" typical="10.0" best="10.0" />
  <Latency worst="10.0" typical="10.0" best="10.0" />
</Performance>
<Performance>
  <accessTypeRef>Load_Algined_Halfword</accessTypeRef>
  <Pitch worst="31.2" typical="12.96" best="12.0"/>
  <Latency worst="4.0" typical="3.05" best="3.0"/>
</Performance>
<Performance>
  <accessTypeRef>Load_Algined_Word</accessTypeRef>
  <Pitch worst="30.2" typical="12.91" best="12.0"/>
  <Latency worst="11.0" typical="2.45" best="2.0"/>
</Performance>
```

性能計測ツール

会員向け公開成果 (2)

- モデルベース並列化プログラム類
  - 簡単なモデルで動作する評価版バイナリ
  - 簡単なサンプルモデルと結果（一般公開）
- MCA公開文書参考和訳
  - SHIM仕様
  - MPP (Multicore Programming Practice)（一般公開）



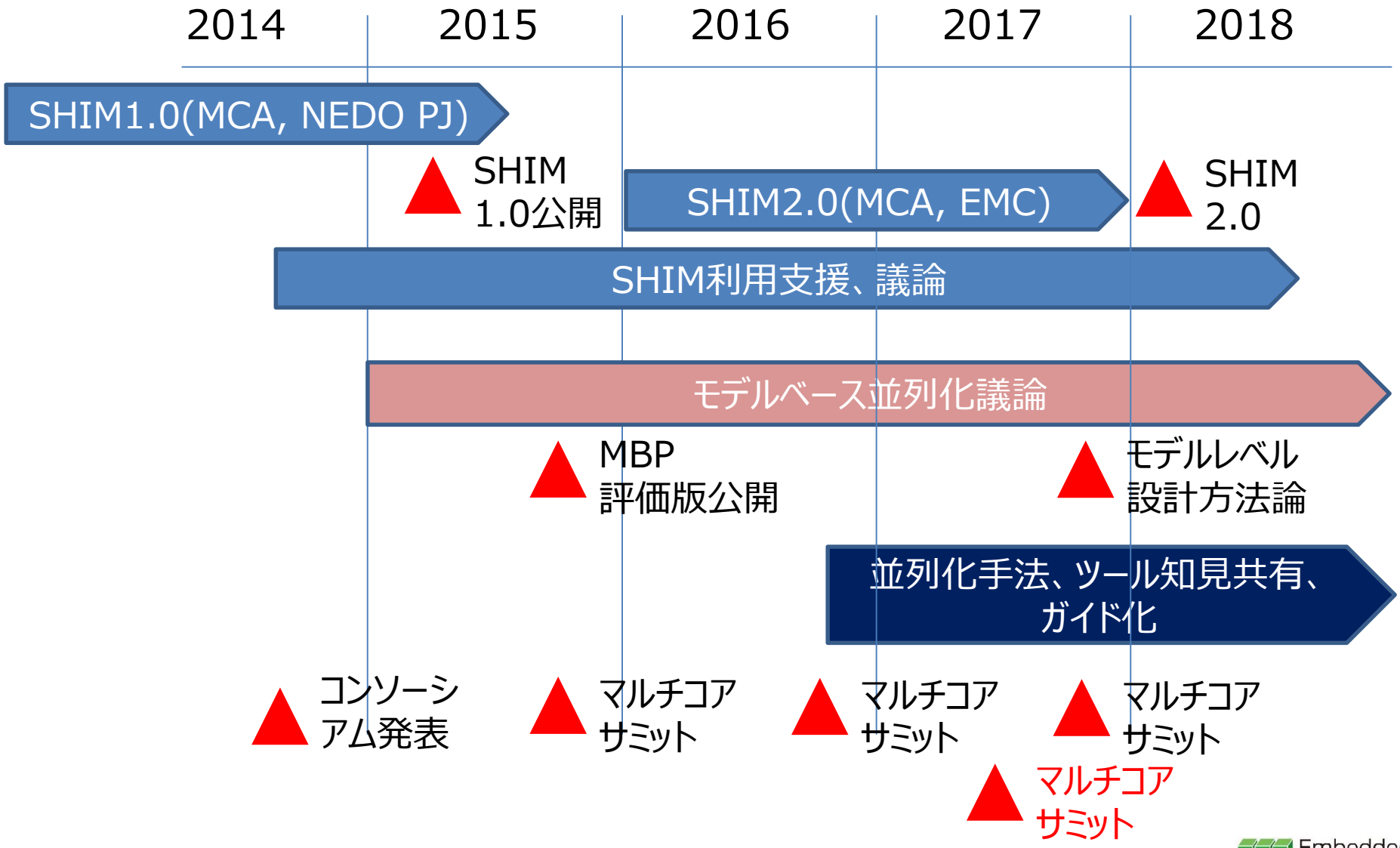
並列化モデル

データ読み書き間の依存性は計算の部分的な順序を決定する。順序を制限するデータ依存には3つのタイプがあり、真のデータ依存、逆依存、出力依存がある。(図8)

真のデータ依存は、あるデータ値への書き込みが終わるまでは読み込みができないような操作間の順序を示す。これはアルゴリズム内の基本的な依存であるが、このデータ依存性の影響を最小化するようアルゴリズムを改良することもできる場合もある。

MPP

ロードマップ


2014
2015
2016
2017
2018
SHIM1.0(MCA, NEDO PJ)
SHIM
1.0公開
SHIM2.0(MCA, EMC)
SHIM
2.0
SHIM利用支援、議論
モデルベース並列化議論
MBP
評価版公開
モデルレベル
設計方法論
並列化手法、ツール知見共有、
ガイド化
コンソーシ
アム発表
マルチコア
サミット
マルチコア
サミット
マルチコア
サミット
マルチコア
サミット

入会のメリット

- 多様なメニーコアをうまく使いこなすためのリファレンスプラットフォームの構築または情報収集
  - マルチコア製品情報、ソフトウェア、ツール連携情報
  - SHIM, MBP等の利用技術情報、事例、ノウハウ
  - リファレンスツール等の利用支援
  - MCA(Multicore Association)関連情報
- メニーコア開発で必要なツール類に関する要求出し、構築または情報収集
  - メーカとベンダが枠を超えて意見交換
- 標準化活動に対するインプット
  - 日本市場としてMCAなど標準化団体にインプット
- EMC成果物の先行リリース

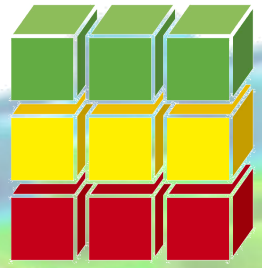
入会の意義とSHIMのメリット

立場	コンソーシアム活動の意義
組込み機器メーカ	マルチ・メニーコア技術は性能向上には必須技術。自社設計フロー構築のベースとなり、かつ自社の要望を入れられる。非競争領域と考える場合には、同業他社と協力し、ハード・ソフト・ツールベンダ全体に働きかけることができる
各種ベンダ	自社製品に適合したプラットフォーム、設計方法論、ガイドライン策定への参画、標準やリファレンスツールに自社の要望を入れられる

立場	SHIM活用メリット
組込み機器メーカ	SHIM対応ツールにより、ハードウェアプラットフォームの切り替えなどにより追従しやすく、またより充実したツールチェーンの選択種が得られる
ソフトウェア/ツールベンダ	SHIMを利用することにより、ソフトウェアの共通化と依存データの外部化による様々なマルチ・メニーコアアーキテクチャへの対応コスト削減と市場化期間の短縮が図れる
ハードウェア/半導体ベンダ	SHIMで自社ハードウェア/マルチ・メニーコアチップのアーキテクチャ情報をツールベンダ、機器メーカに提供することで、ベンダサポートコスト低減とSHIMに対応した多様なツールのエコシステムを活用することができる

メンバーシップ

- 会員（2017年7月現在16団体）
  - デンソー、dSPACE、ルネサス エレクトロニクス、オリンパス、eSOL、ガイオテクノロジー、CATS、萩原電気、アバールデータ、大阪大学、名古屋大学、立命館大学、早稲田大学アドバンスドマルチコアプロセッサ研究所、他
  - 相互協力：JASA、MCA(Multicore Association)
- メンバーシップ構成
  - 正会員（入会金なし、年会費20万）準会員、特別会員
  - 詳細は http://www.embeddedmulticore.org/
- （参考）SHIM WG Primary Contributing Members
  - Cavium Networks, CriticalBlue, eSOL, Freescale, Nagoya University, PolyCore Software, Renesas, Texas Instruments, TOPS Systems, Vector Fabrics, and Wind River.



Embedded
Multicore
Consortium

www.embeddedmulticore.org

Embedded
Multicore
Consortium

www.embeddedmulticore.org

2017/7/13
SHIM: ソフトのための
国際標準ハードウェアモデル記述

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

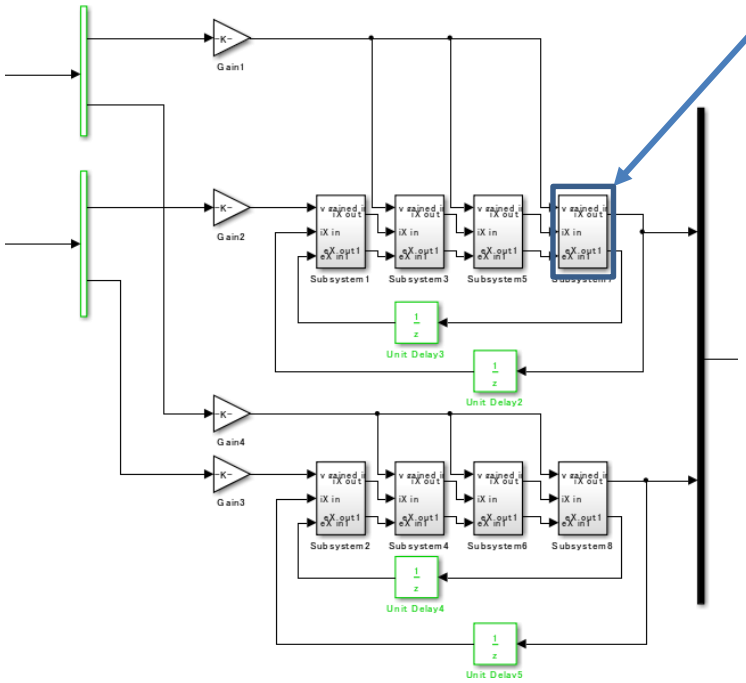
- SHIMとは
- 準備：SHIM・LLVM-IR・性能見積手法
- SHIMによる静的性能見積
  - 命令レイテンシの計測
  - 命令レイテンシとプロファイルを用いた見積
  - メモリアクセスの考慮
- まとめと今後の課題
  - SHIMulator (SHIMによる動的性能見積)
  - SHIM2.0

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<u>ブロック粒度(処理時間)</u>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

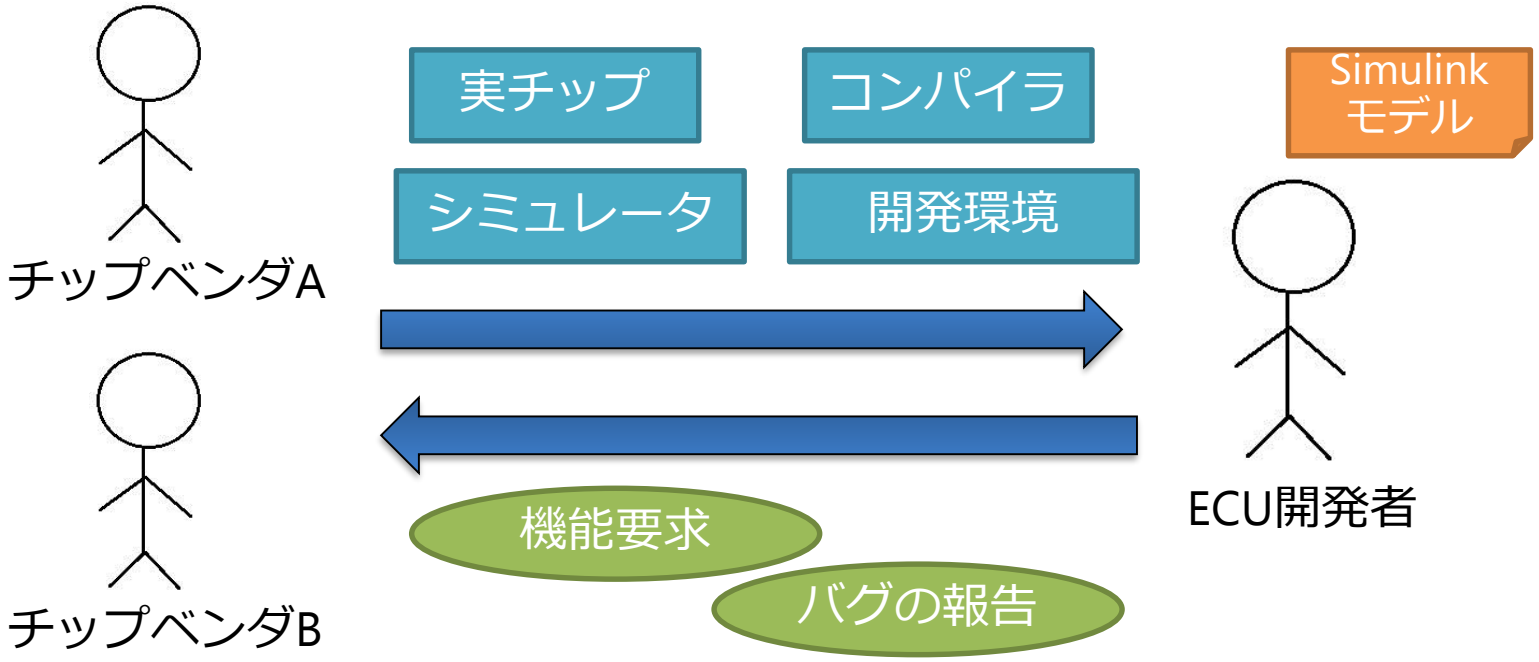
保有していても
- それぞれの設定工数が大きい
- 使いこなすために知識と時間が必要

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

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

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)

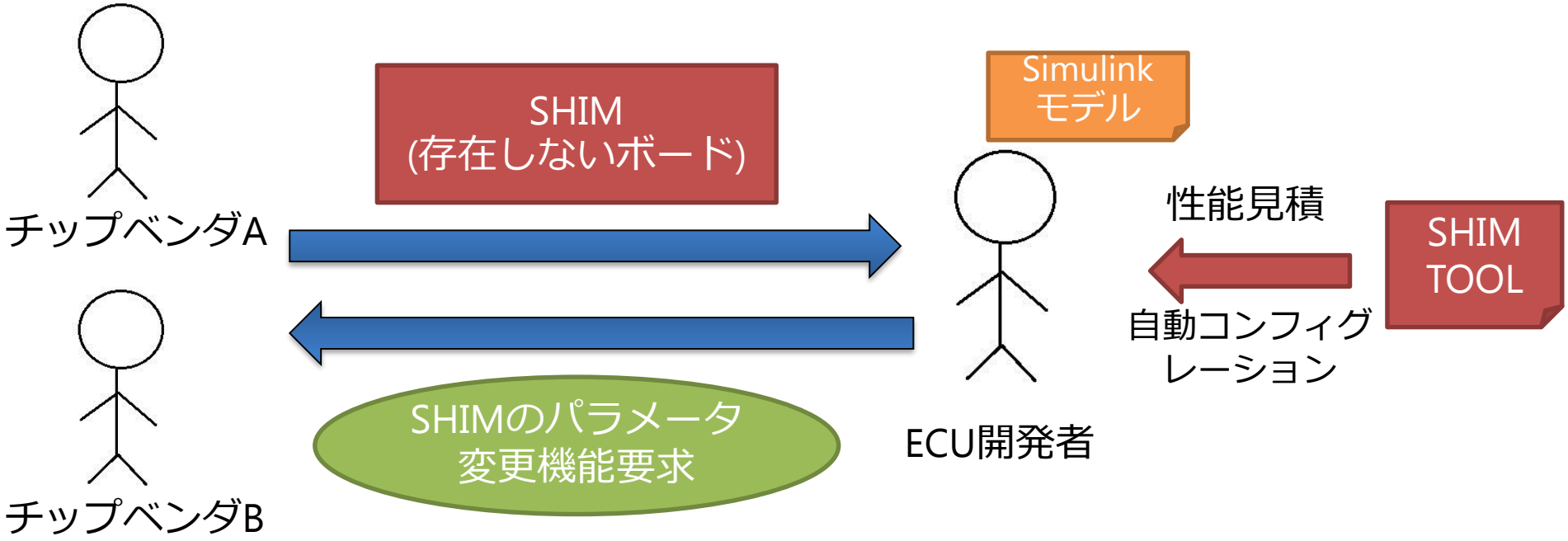


問題点：**工数が多く**、開発が**非効率**

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

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.
PDSL

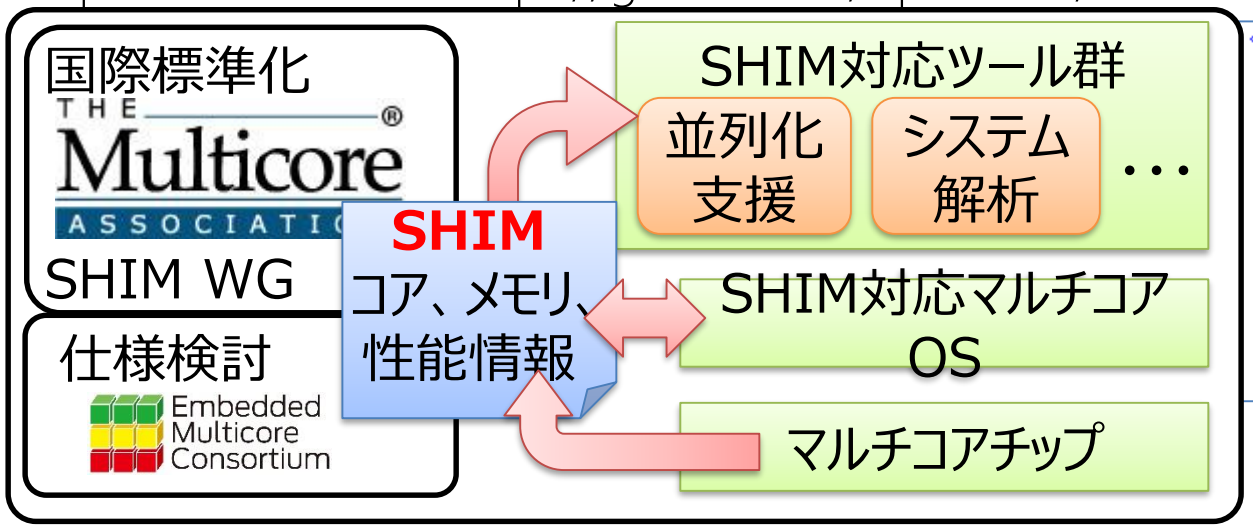
SHIMとは

Software-Hardware Interface for Multi-many-core

Multicore Tools
SHIM
Multicore HW

- 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



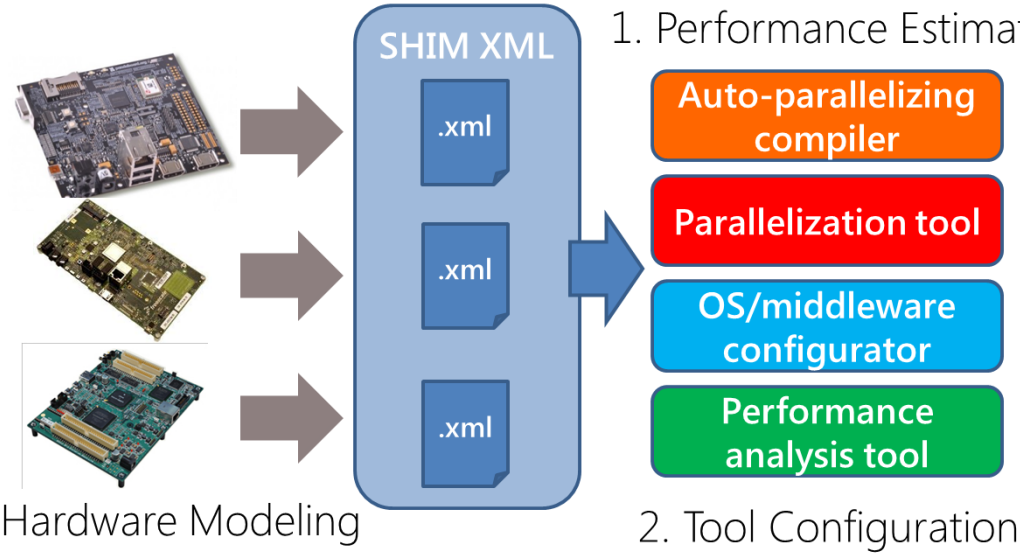

```
<MasterSlaveBinding slaveComponentRef="LRAM_B
  <Accessor masterComponentRef="CPU_BOCOP2">
    <PerformanceSet>
      <Performance>
        <accessTypeRef>Instruction_Fetch</acc
        <Pitch best="1.0" typical="1.0" worst
        <Latency best="1.0" typical="1.0" wor
      </Performance>
     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

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

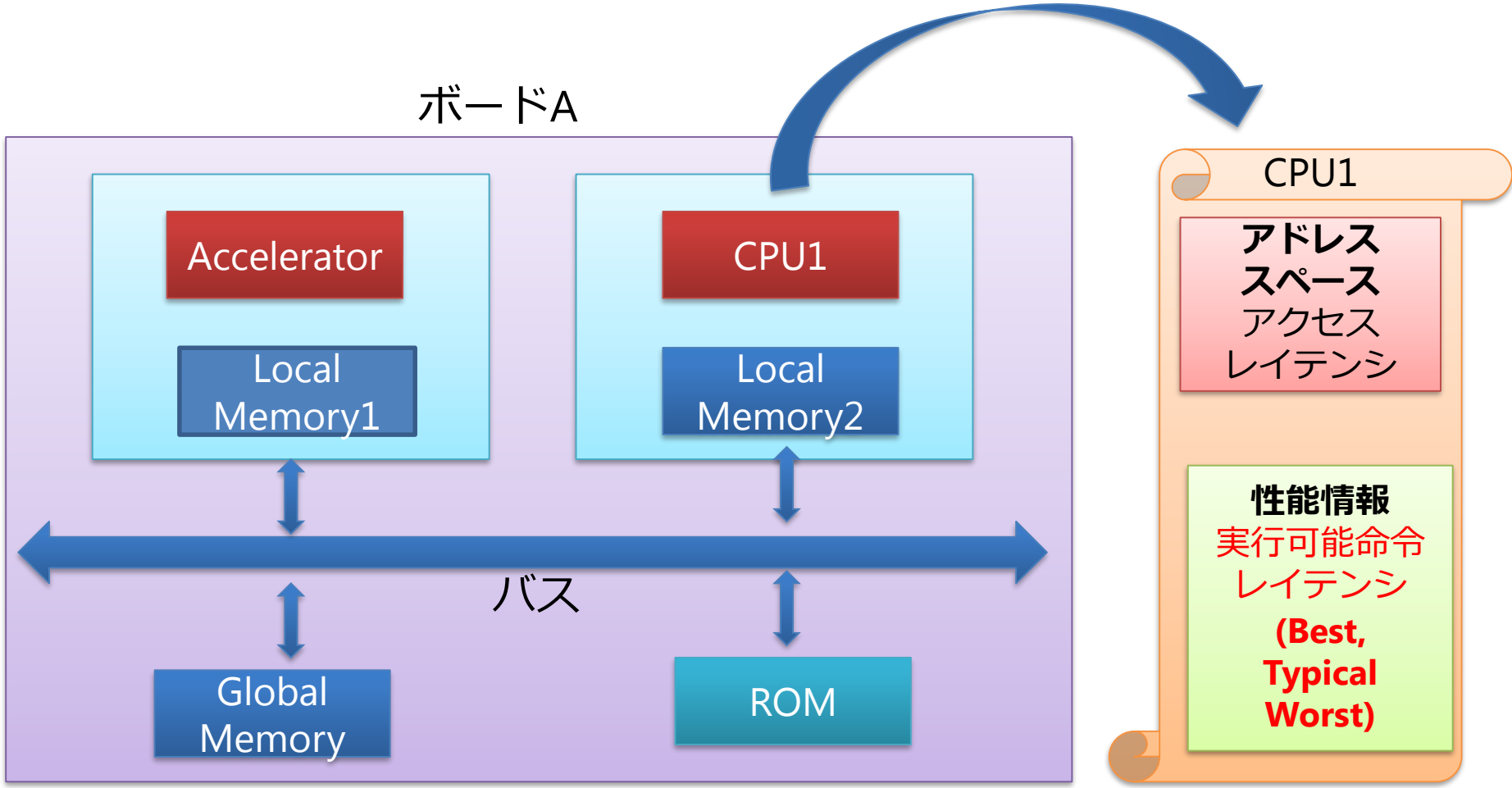
- マルチコアにおけるアプリケーション実行性能見積
- マルチコア選定時のアプリケーション実行性能比較
- 異なるマルチコアへのアプリケーション移植の際の性能見積
- 複数マルチコアをターゲットとしたソフトウェア部品開発
- 特定アプリケーション向けに特化したマルチコアを企画する際の性能評価
- マルチコア向け開発支援を行う各種ツールの開発コスト低減とSHIM対応ツールエコシステム

アジェンダ

- SHIMとは
- 準備：SHIM・LLVM-IR・性能見積手法
- SHIMによる静的性能見積
  - 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
- まとめと今後の課題
  - SHIMulator (SHIMによる動的性能見積)
  - SHIM2.0

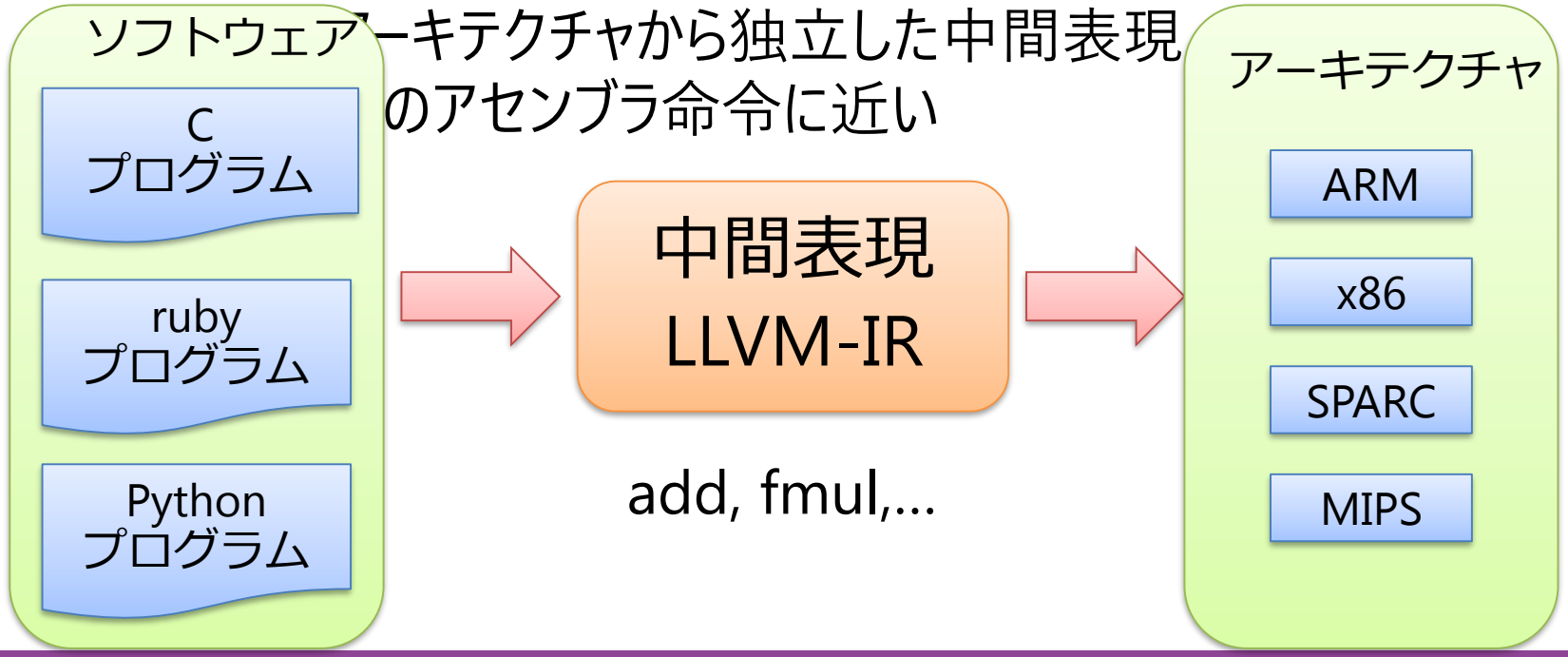
SHIM

- **SHIM記載情報**(XMLで記述)



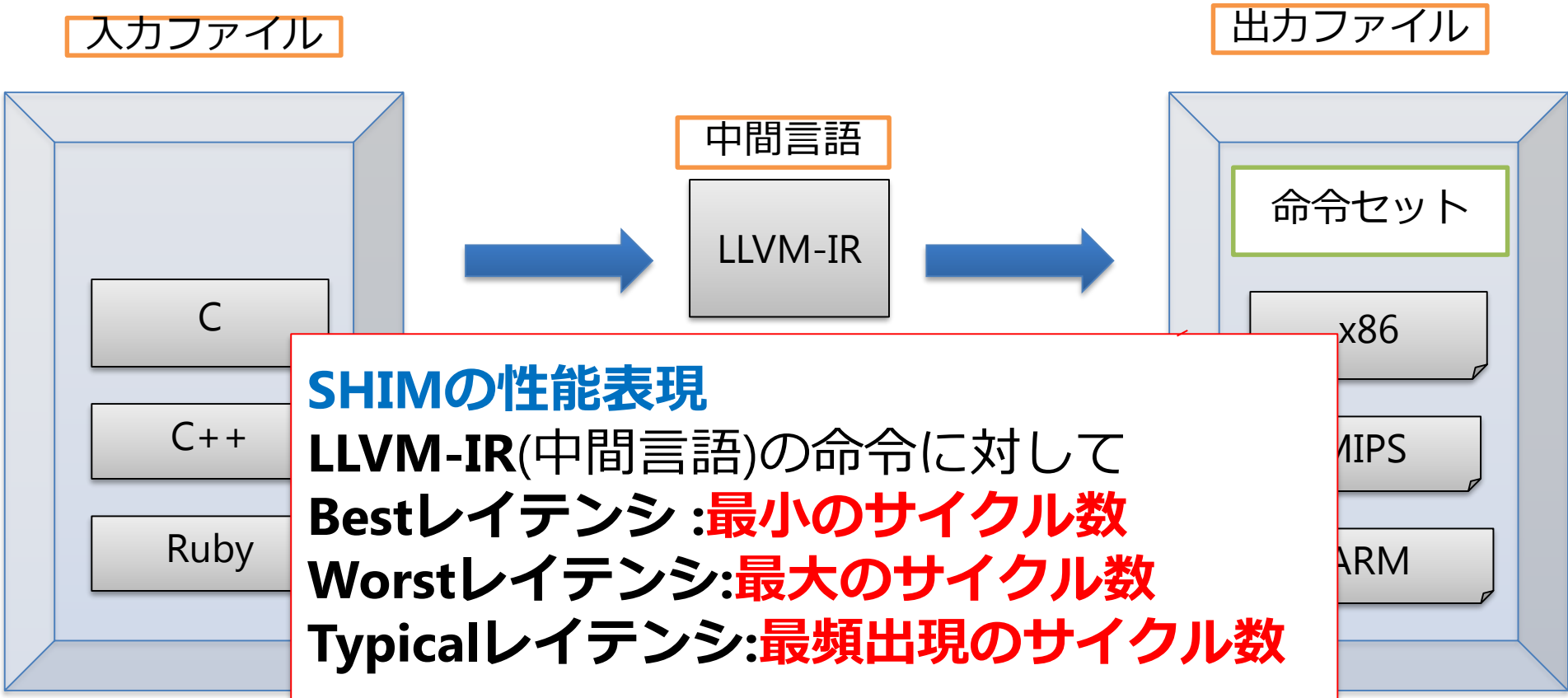
LLVMとLLVM-IR

- 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)



34

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

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	–	–	–	–	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
－：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

アジェンダ

- SHIMとは
- 準備：SHIM・LLVM-IR・性能見積手法
- SHIMによる静的性能見積
  - 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
- まとめと今後の課題
  - SHIMulator (SHIMによる動的性能見積)
  - SHIM2.0

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

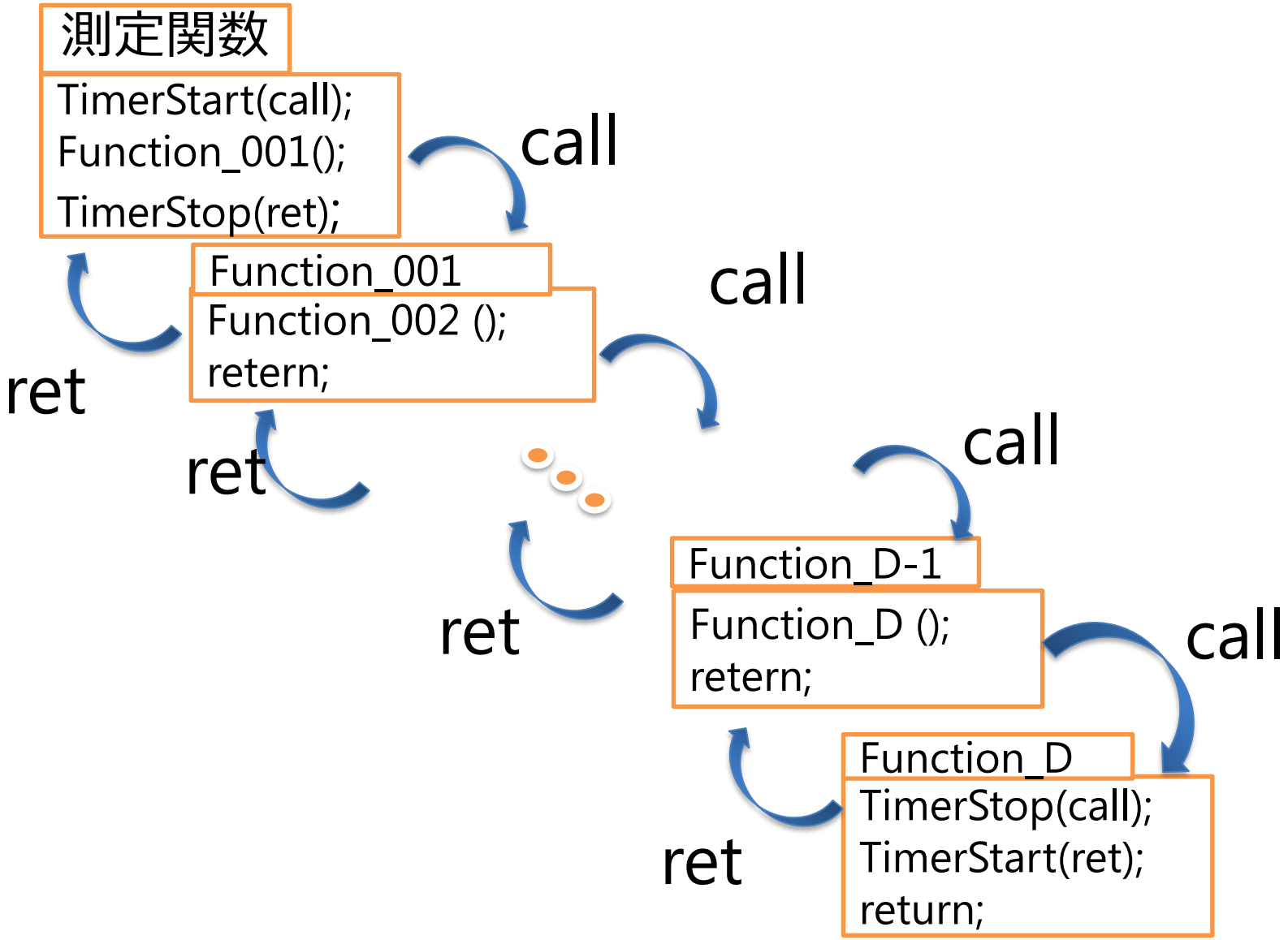
Target
C ： uiz=uiy+uix

```
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]
```

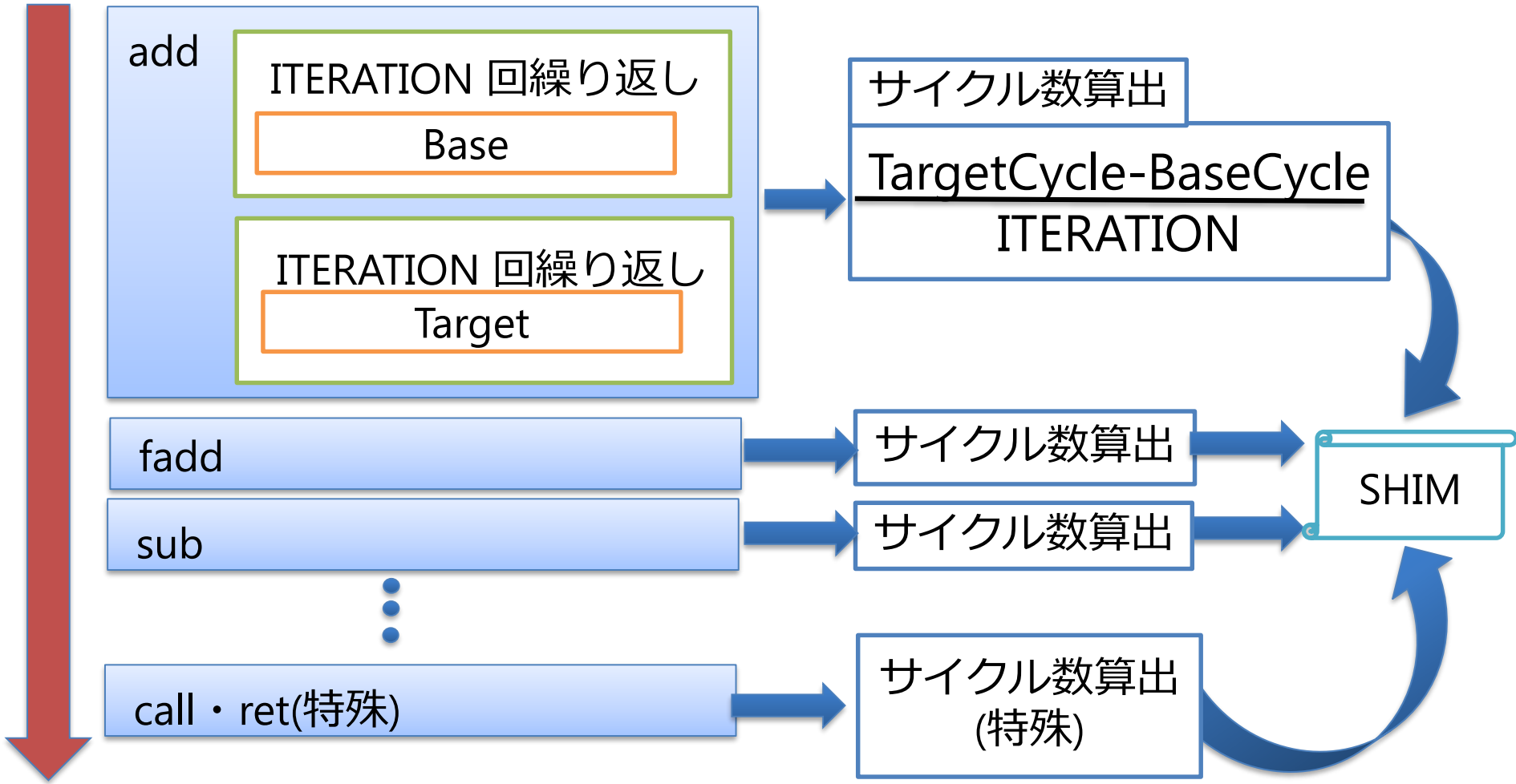
命令レイテンシの計測

- **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

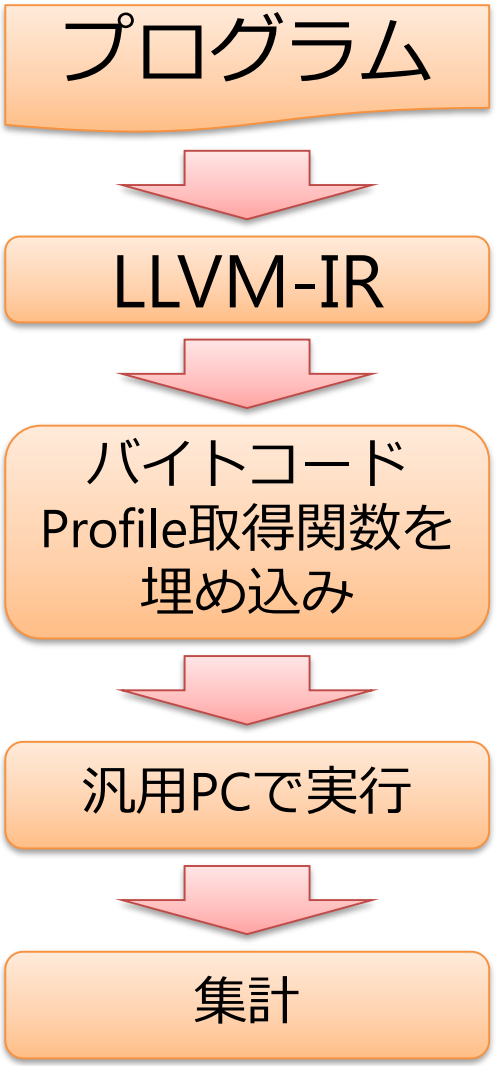


命令計測プログラムの全体像



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llvm-prof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：現在のバージョンのllvmにはllvm-profは存在しない。外部プロファイラで実現する



[*]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.

命令の出現回数の取得方法

Profile例

```
;;; %main called 1 times.
;;;
define void @main() {
  ;;; Basic block executed 1 times.
  store i32 0, i32* @main_result, align 4
  %1 = call i32 @jpeg2bmp_main()
  ret void
}
```

main関数のProfile結果

LLVM-IR	回数
store	1
call	1
ret	1

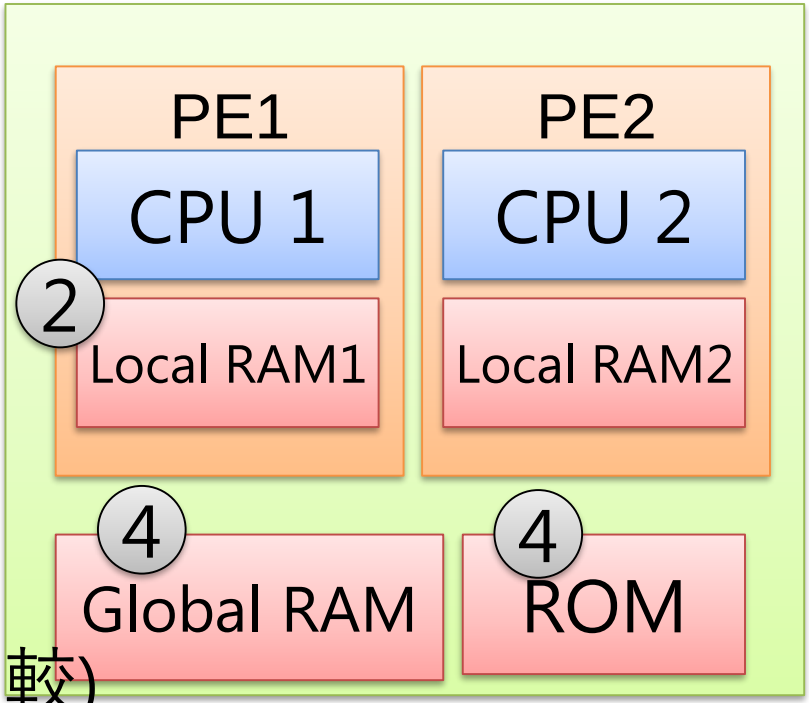
メモリアクセス情報

変数名	アクセス回数
main_result	1

46

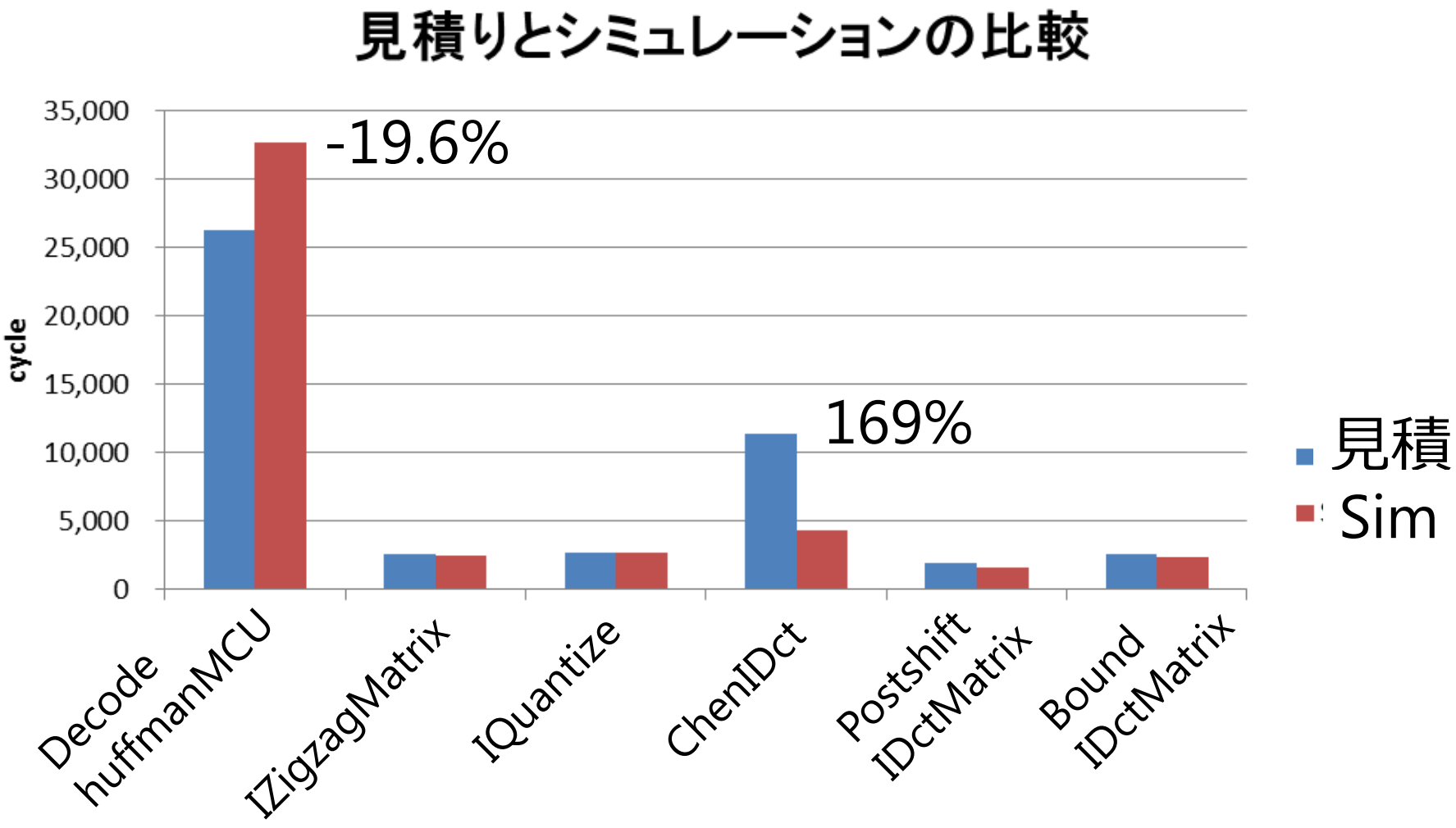
実験環境

- アプリケーション
  - JPEG decoder
- 対象とするチップ
  - v850アーキテクチャ デュアルコア
- コンパイラ
  - Clang　(見積)
  - v850-renesas-elf-gcc (比較)
  - 最適化オプション: -O0
- シミュレーター
  - cforest_mp（比較）



はCPU1のアクセスレイテンシ

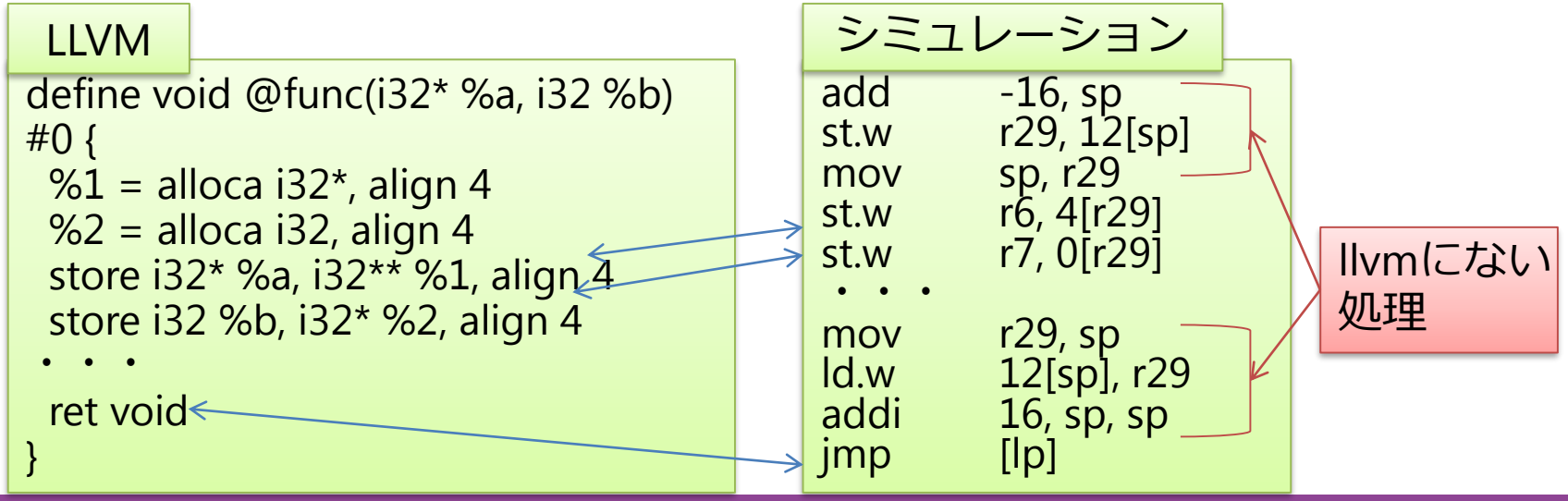
シングルコアでの見積



48

シングルコアでの見積（誤差の要因）

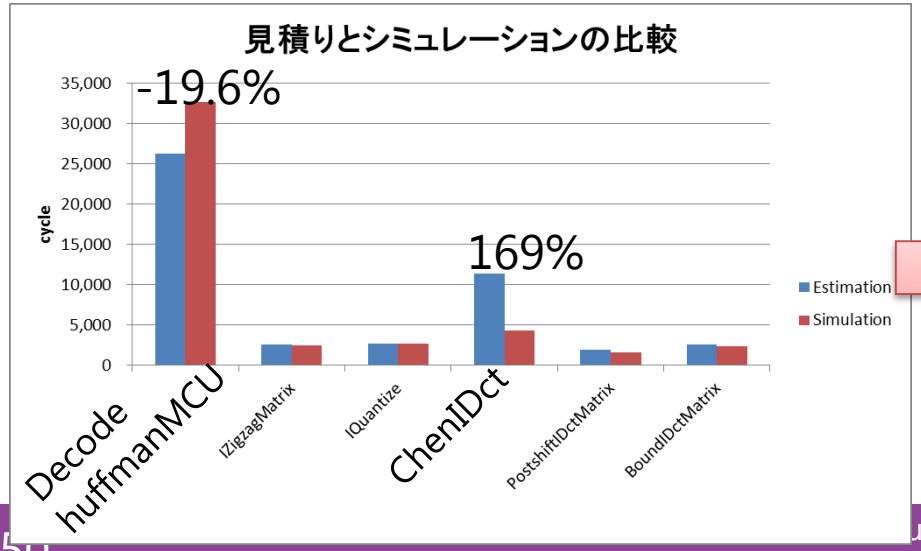
- 除算命令
  - シフト命令を含む複数命令に置き換えられる場合がある→定数，特に２の冪乗の場合
- register修飾子
  - LLVMでは無視されている
- 関数呼び出し･復帰
  - レジスタの退避回復動作がLLVMにはない

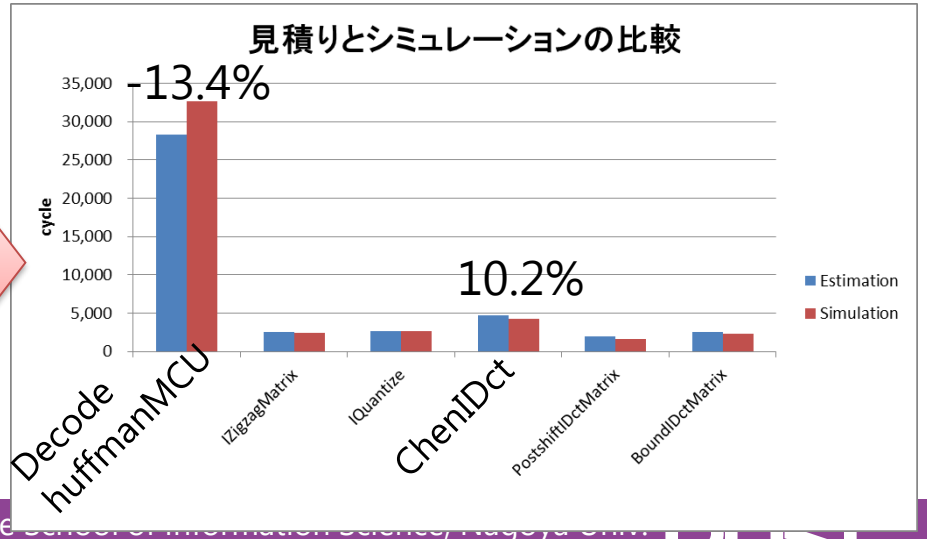


49

シングルコアでの見積（誤差対策）

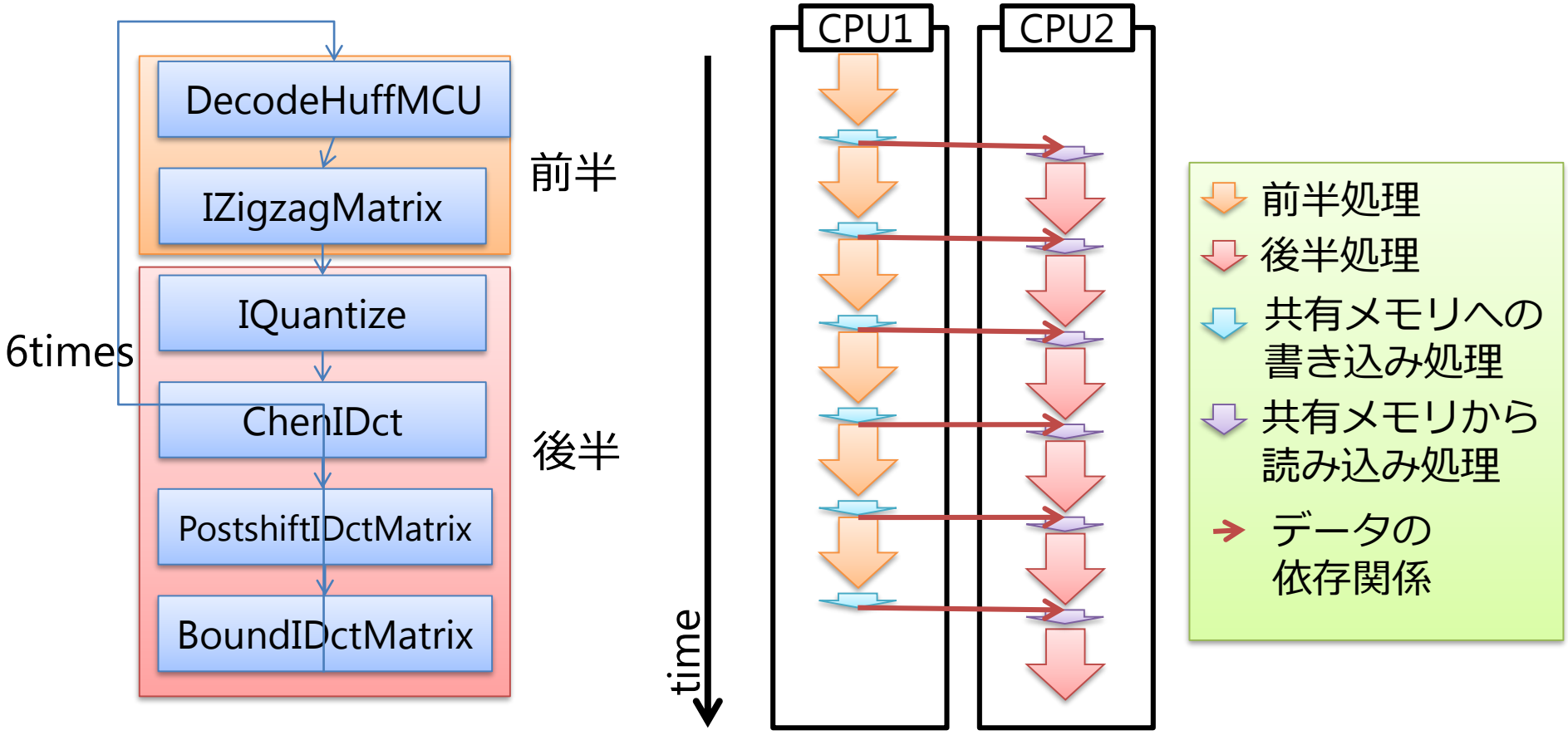
- 除算
  - 2の冪乗で割る場合のコストをシフトを含む複数命令のコストに変更
- register修飾子
  - register修飾子のある変数はレジスタにあると仮定
  - load/storeのコストをレジスタへのアクセスに変更
- 関数呼び出し・復帰
  - レジスタ退避回復動作のコストを加算





JPEGプログラムのパイプライン並列化

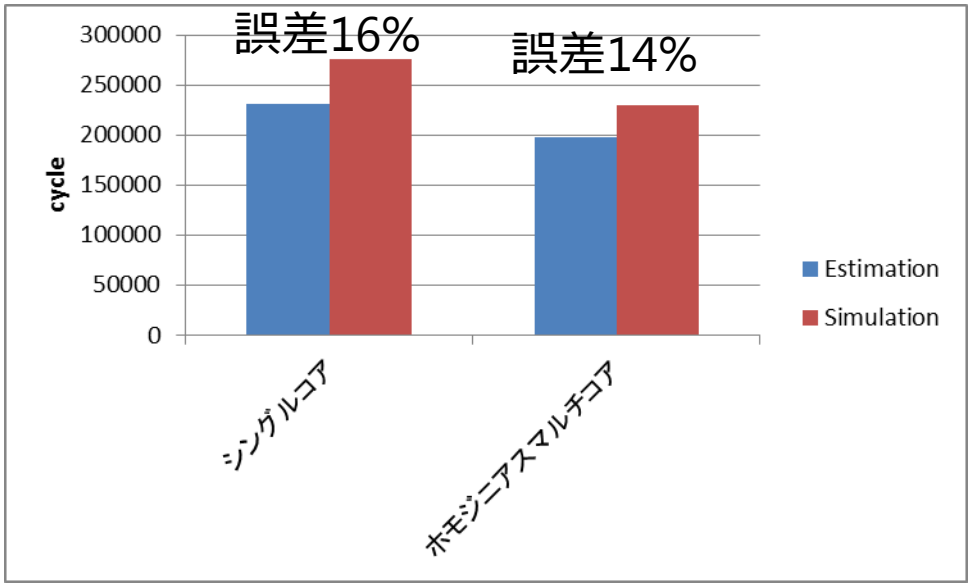
- 処理をパイプライン並列化する
  - コア間のデータ通信は共有メモリを使用する



51

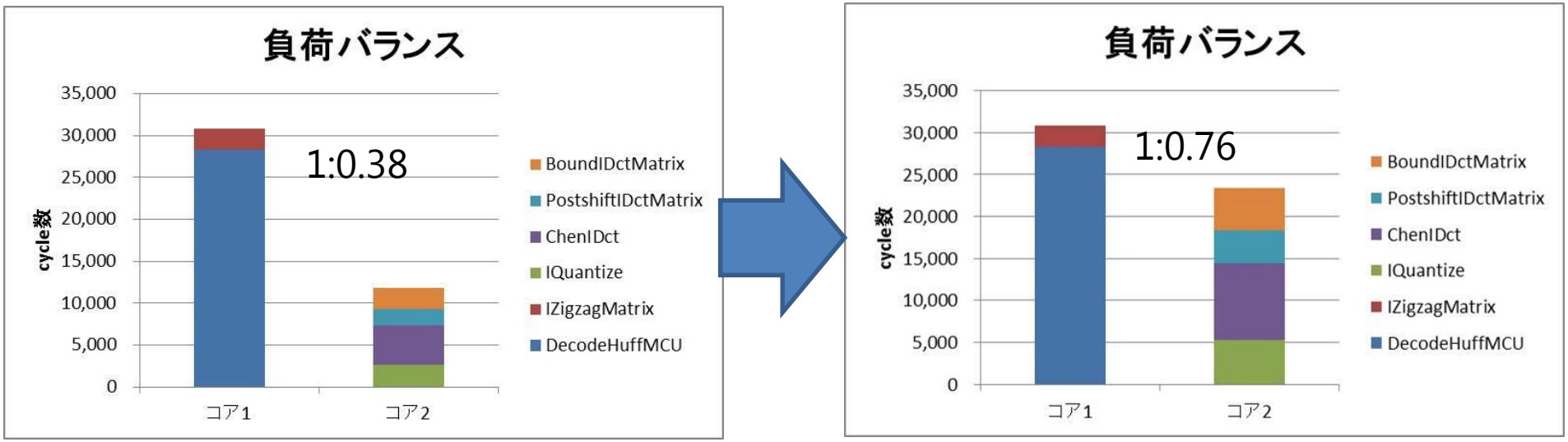
ホモジニアスマルチコアでの見積

- コア間のデータ通信オーバーヘッドを考慮
  - 実行時間（２段にパイプライン並列化して６回実行）
    - MAX(前半処理+共有メモリ書込, 後半処理+共有メモリ読込）×5
      +（前半処理+共有メモリ書込)+(後半処理+共有メモリ読込)
- 誤差は20%以内
- 性能向上は20%程度
  - 理想（上記式の場合）
    - 負荷バランス１：１であれば、通信オーバーヘッド無しのとき(12/(5*1+2))-1=71%向上



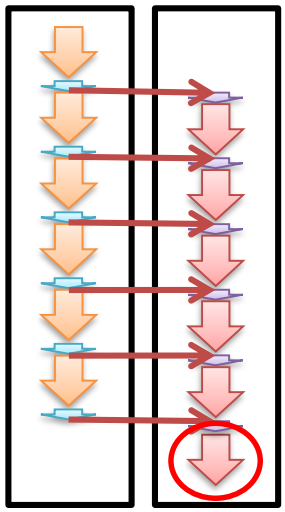
電力効率の改善

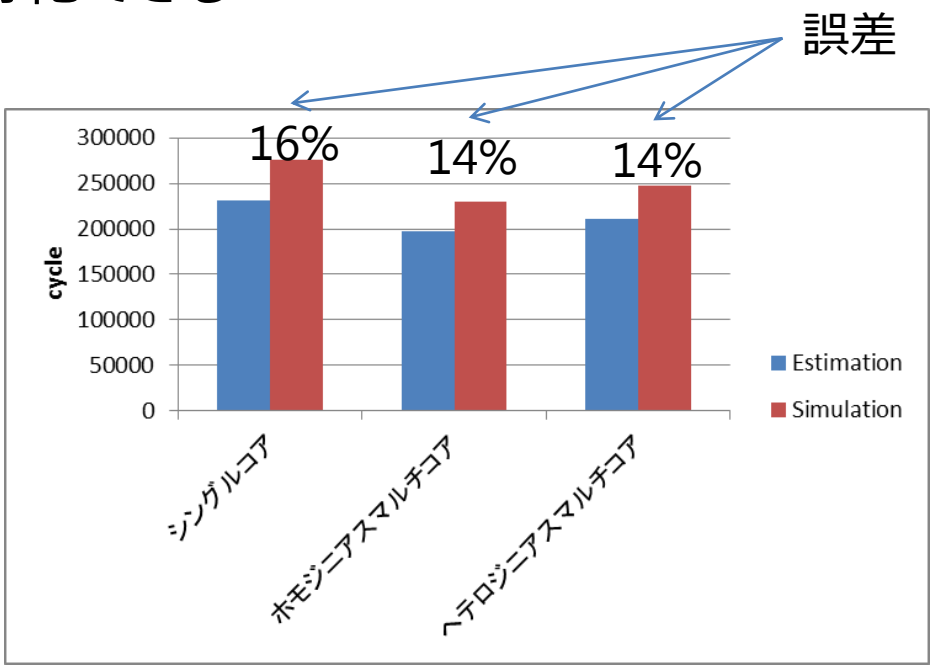
- コア間の負荷バランスが不均衡
  - 不均衡であることを活用する
    - コア2の周波数を低くしても性能劣化は起こりにくい
- コア2の周波数を半分にする
  - 電力効率の改善



ヘテロジニアスマルチコアでの見積

- 周波数ヘテロジニアスマルチコア
  - CPU1：CPU2= 1 ： 1 / 2
- ホモジニアス型からの性能劣化は無い
  - ○部分による劣化が約 6 %
    - より大きな単位では並列化できる
  - 電力効率がいい





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	－	－	－	－	要	要				
SHIM利用静的解析	要	要	要	要	要	要	LP	LP	要	要
SHIM利用動的解析	要	要	要	要	要	要	要	要	要	要

要：要対応
－：対応不要（実行環境精度依存）
斜線：考慮不要

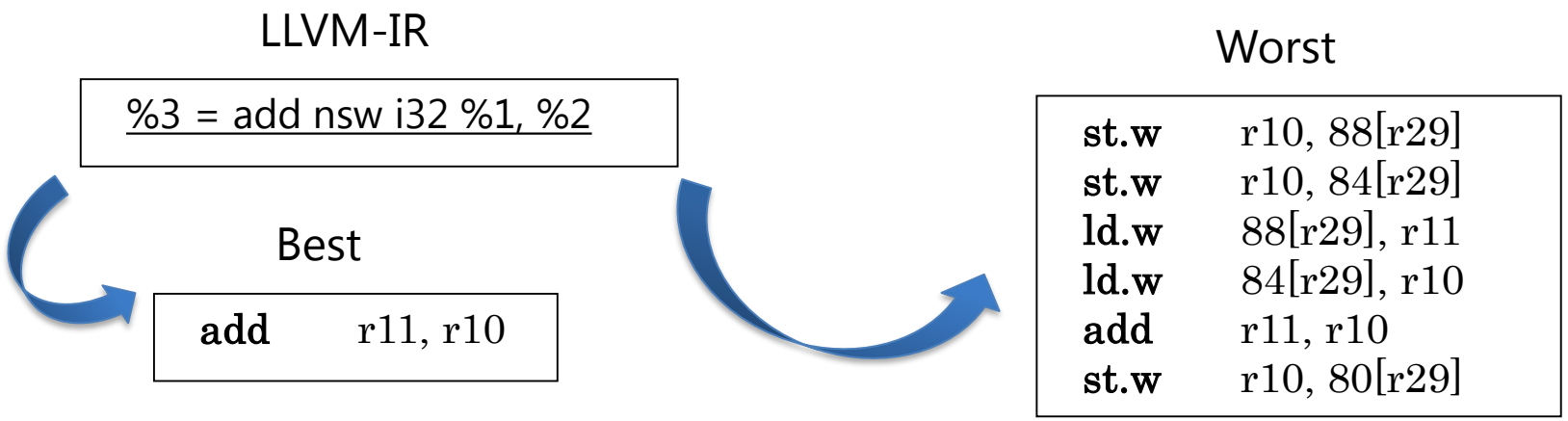
LP: 命令レイテンシとプロファイルにて対応

メモリアクセスを考慮したレイテンシ見積

- JPEGの場合、メモリアクセスパタンなど、様々なパラメタが計算可能。実験ではこれを適用
- これにより、SHIMの可能性の証明にはなったが、汎用的な見積方法とは言えない
- 「除算→シフト」などは既存技術で解決可能。SHIM固有問題に「ローカルアクセス問題」がある

- 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)

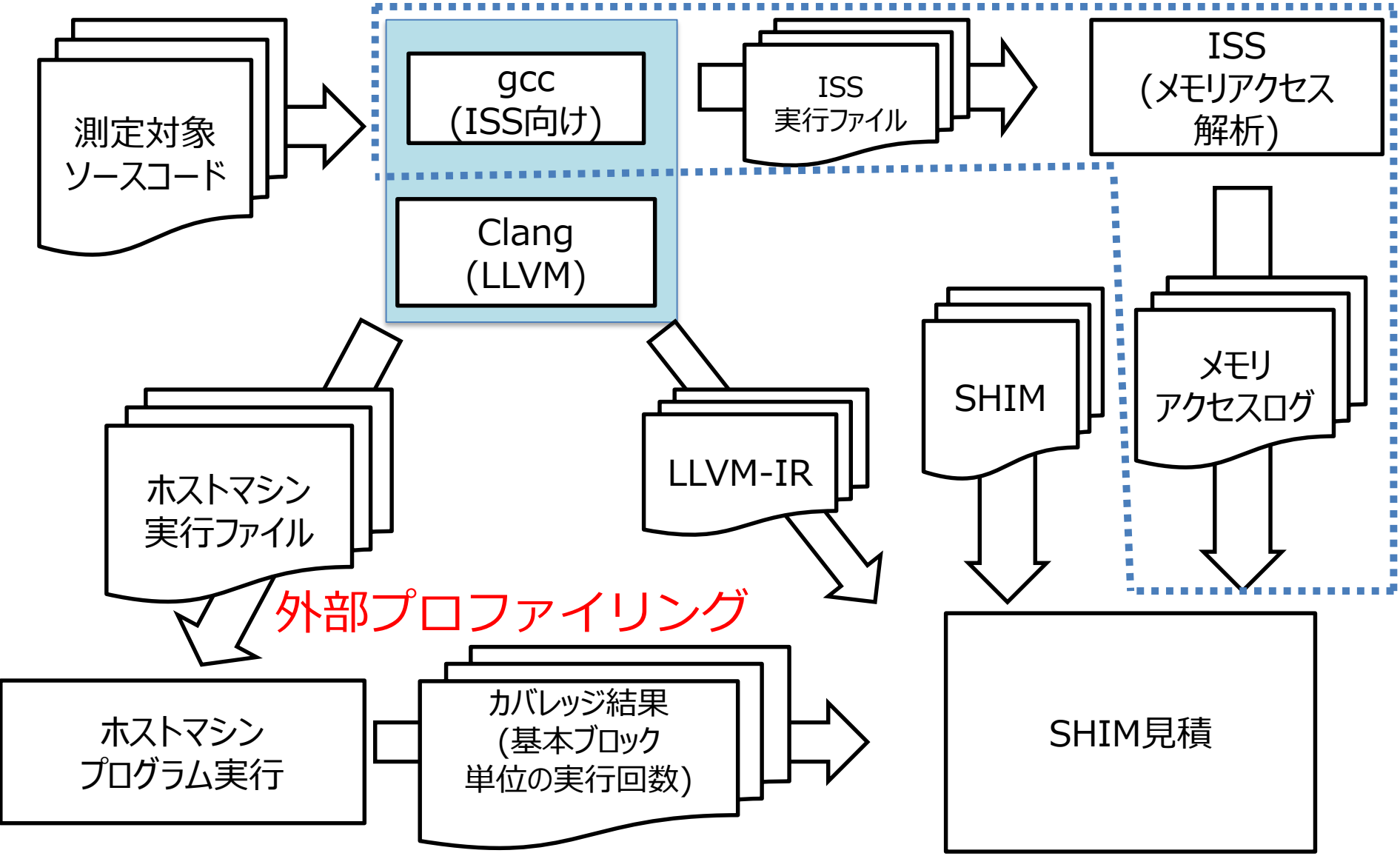
$$\text{Local}[\%] = \frac{\text{ローカルメモリアクセス回数}}{\text{総命令数} - \text{グローバルメモリアクセス回数}}$$

LLVM-IRとアセンブリで1:1対応されている

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

見積フロー

ISS（命令セットシミュレータ）を
用いたローカルアクセス回数計測



見積の集計方法

見積の例

カバレッジ結果
100回実行

LLVM-IR

```
%1:

%i.01 = phi i32 [ 0, %0 ], [ %2, %1 ]
%2 = add nsw i32 %i.01, 1
%3 = tail call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]*
... @.str, i64 0, i64 0), i32 %2) #2
%exitcond = icmp eq i32 %2, 100
br i1 %exitcond, label %4, label %1
```

SHIM

レイテンシの選択

Add :

Worst,Typical,Best

(8, 1.3,1)

このアセンブリの合計レイテンシ：
1.3[Cycle]×100[回]=130[Cycle]

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

精度評価実験

- 実機とSHIMを用いた見積を比較
  - **実機環境**
    - ボード:RH850メニーコアチップ
    - シングルコア: **RH850(32MHz)**
  - **評価対象アプリケーション**

アプリケーション名	概要
Fibonacciモデル	Fibonacci数列を計算するモデル
PBMコントローラモデル	永久磁石モータコントローラモデル
Dhrystoneベンチマーク	整数プログラムベンチマーク

精度評価実験

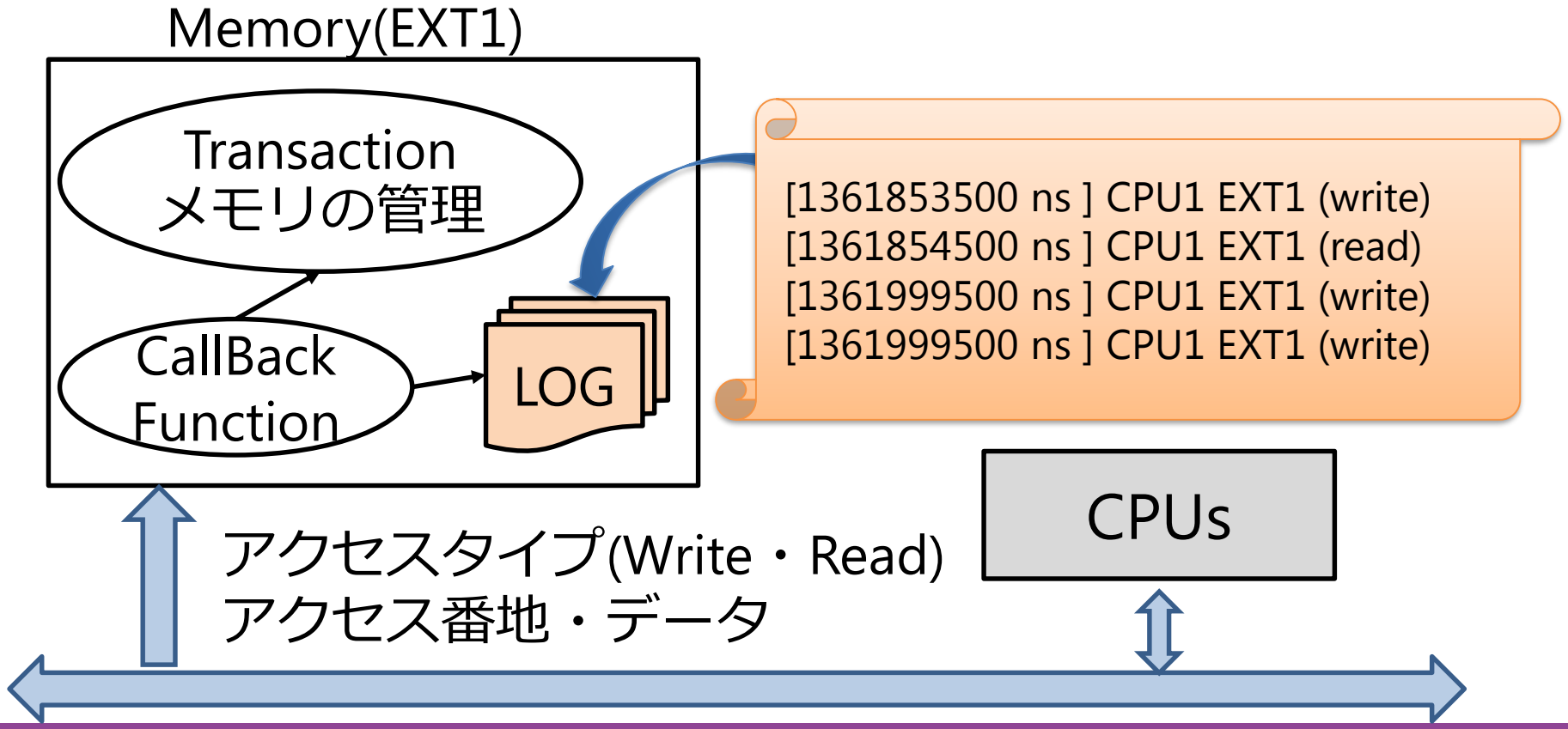
- **メモリアクセス解析(ISS)**
  - 本来はSHIMを用いたLLVM-IR ISSを使うべきだが、存在しないため、Imperas社のOVP Sim (V850)+メモリモデル(SystemC記述)を利用（→今後の課題）
  - リンカスクリプトで測定対象を専用メモリに割当
    - 実機に近いメモリの配置
  - メモリアクセス履歴のみ取得

- **見積項目**
  - ALL BEST：全てBESTレイテンシを選択
  - ALL WORST：全てWORSTレイテンシを選択
  - 提案手法：メモリアクセス割合考慮レイテンシを選択

ローカルメモリアクセス解析

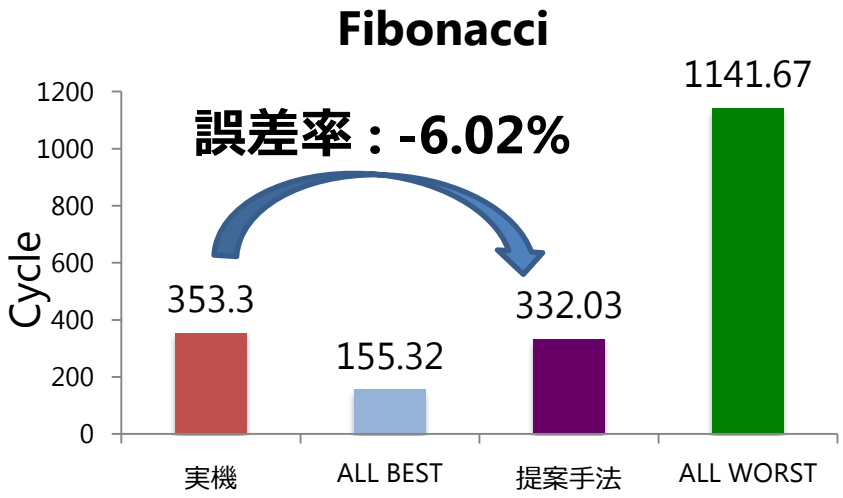
OVPSim：高速な命令セットシミュレータ

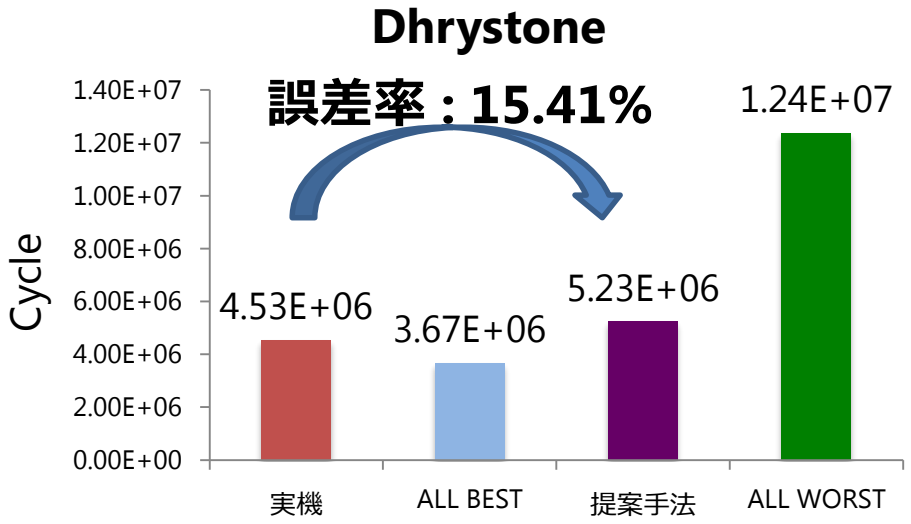
- システム(OS)も動作可能。命令数精度
- SystemCで記述した専用メモリを用いて**アクセスログ**を取得

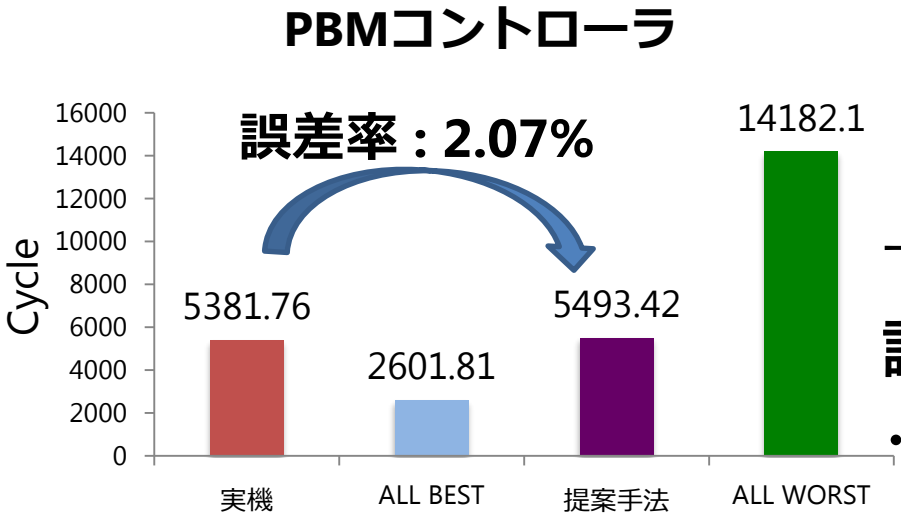


Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

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

メモリ解析結果

表 メモリ解析結果

プログラム	ローカルアクセス割合[%]
Fibonacci	24.96[%]
PBMコントローラ	24.97[%]
Dhrystone	18.00[%]

- Dhrystoneがローカルアクセスの割合が少ない
  - ALL BESTに近い実行時間であることが分かり、提案手法は実機に近い結果が得られた
  - ローカルメモリ解析結果の活用は精度向上に効果あり
- 全体で**20%前後の割合**でローカルメモリにアクセス
  - メモリシミュレーションなしでも20%前後の比で選択にすると適する
    ⇒Typicalレイテンシとして活用可能

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	要	要				
SHIM利用静的解析	要	要	要	要	要	要	LP	LP	AL	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（実行環境精度依存）
斜線：考慮不要

LP: 命令レイテンシとプロファイルにて対応
AL: ターゲットに近いISS/実機を利用したアクセスログ解析にて対応

アジェンダ

- SHIMとは
- 準備：SHIM・LLVM-IR・性能見積手法
- SHIMによる静的性能見積
  - 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
- まとめと今後の課題
  - SHIMulator (SHIMによる動的性能見積)
  - SHIM2.0

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

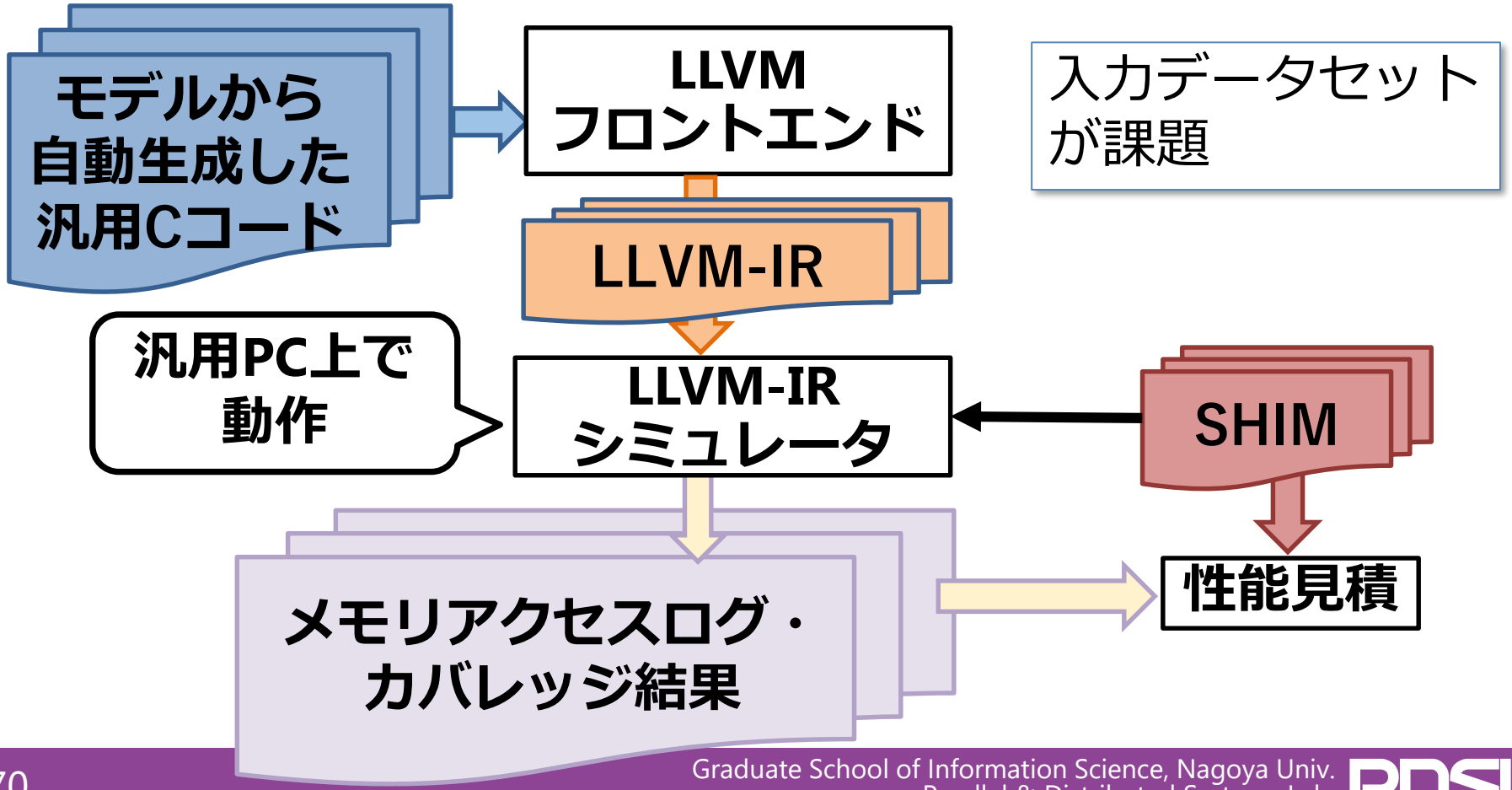
- **動的手法**
  - LLVM-IR用シミュレータ ⇒ 性能見積
    - SHIMを用いたLLVM-IRシミュレータは存在しない

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



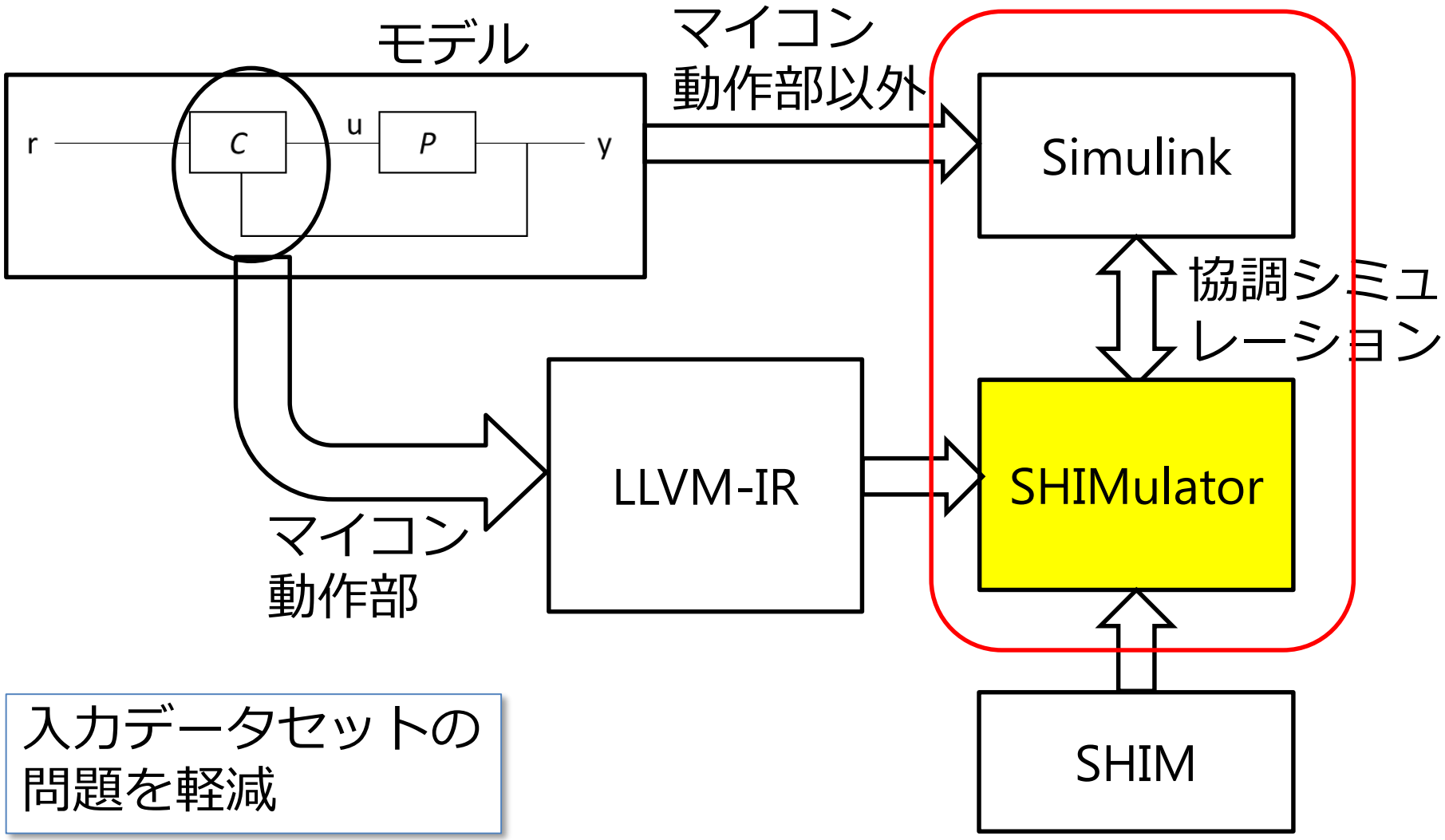
方式と課題への対応（A～Fについては従来から多くの議論があるため本講演の対象外とする。SHIM環境はターゲット環境の±20%精度が目標であることに注意）

	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	－	－	－	－	要	要				
SHIM利用静的解析	(SH/AL)	要	要	要	要	要	LP	LP	(SH/AL)	(SH/AL)
SHIM利用動的解析	(SH)	(SH)	SH	要	要	要	SH	SH	(SH)	(SH)

要：要対応
－：対応不要（実行環境精度依存）
斜線：考慮不要

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。(カッコ)はSHIM1.0では限定的であることを表す

SHIMulatorによるSPILS



入力データセットの
問題を軽減

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	－	－	－	－	CS	要				
SHIM利用静的解析	(SH/AL)	要	要	要	要	要	LP	LP	(SH/AL)	(SH/AL)
SHIM利用動的解析	(SH)	(SH)	SH	要	SH/CS	要	SH	SH	(SH)	(SH)

要：要対応
－：対応不要（実行環境精度依存）
斜線：考慮不要

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。(カッコ)はSHIM1.0では限定的であることを表す
CS: 協調シミュレーションにて対応

SHIM2.0

- SHIM2.0では以下の課題について議論中
  - LLVM-IRでは表しきれない命令
    - ハードウェアが持つ画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
  - キャッシュの表現強化
  - モジュール化による記述量削減
  - Etc.

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	－	－	－	－	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
－：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

まとめ

- SHIMの考え方、概要について紹介
- SHIMを使った性能見積の課題、取り組みについて紹介
- SHIMの今後の発展について紹介
- SHIMの抽象度と精度のトレードオフ
  - 精度を落としているが故に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
  - 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