RISC-V 命令セットの特性と研究への利用

東京大学大学院 情報理工学系研究科 創造情報学専攻
塩谷 亮太
shioya@ci.i.u-tokyo.ac.jp

はじめに

- RISC-V 命令セット
  - ◇ オープンな命令セット・アーキテクチャ
  - ◇ 研究や教育の領域ではほとんどデファクトとなるぐらい普及
  - ◇ 企業でも開発や実チップの販売にまで進みつつある
  - ◇ 代表的なコンパイラや OS でも RISC-V を正式にサポート

RISC-V 命令セットの基本的な特徴

- 基本的な構造が単純：シンプルな RISC
  - 32 ビット固定長の命令（16 ビット長の拡張もあり）
  - レジスタ・ベースのロード・ストア・アーキテクチャ
  - 32 本の汎用レジスタを持ち，基本的には 3 オペランドをとる

RISC-V の固有の特徴：
必須部分がコンパクトかつ，必要に応じて機能追加できる

- 整理された仕様：
  - ◇ 必須となる整数演算部分は非常にコンパクト
  - ◇ 用意された各拡張が独立に有効化/無効化できる
    - □ 乗除算（M 拡張）
    - □ 浮動小数点演算（F 拡張と D 拡張）
    - □ 16 ビット長命令（C 拡張）
    - □ SIMD やベクトル，アトミックなどその他多数

RISC-V の研究や開発での利点

1. 代表的なコンパイラや OS が RISC-V を正式にサポートしている
   - ◇ GCC や LLVM，Linux などの最新版が使える
2. 命令セットの基本的な構造が単純
   - ◇ 実装が容易
3. 必須部分は非常にコンパクトで，機能拡張も細かく追加可能
   - ◇ 最少の実装で，GCC などがそのまま使える

- ■ **最少の努力で，最新の環境が使える**

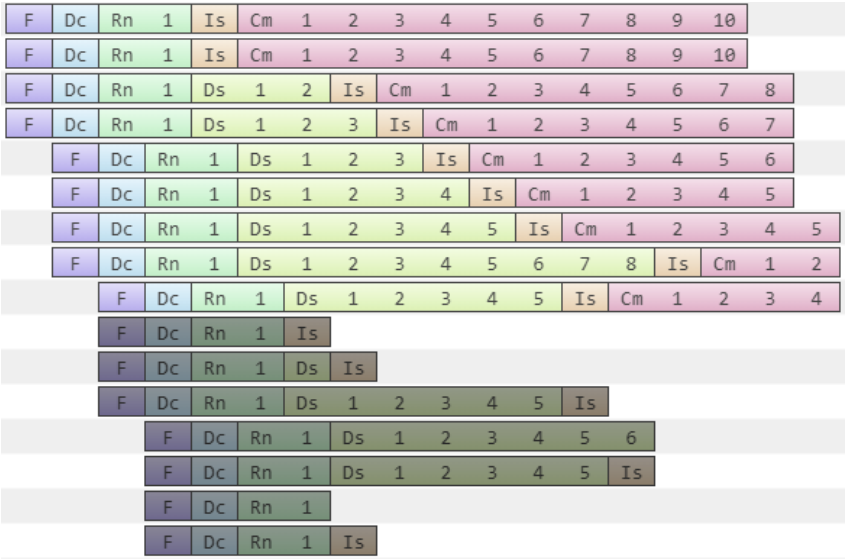
利点を生かした事例の紹介

1. **鬼斬**：サイクル・アキュレート・シミュレータ
2. **RSD**：スーパスカラ・プロセッサ
3. **HORNET**：SIMT 型アクセラレータ

シミュレータ鬼斬

- サイクル・アキュレートな CPU のシミュレータ
  - プログラマからは直接見えないマイクロアーキテクチャ部分まで再現
    - キャッシュ，分岐予測，out-of-order 実行の挙動など
  - 実行時間まで再現することができる
  - https://github.com/onikiri/onikiri2
- 広く使われているものでは，gem5 シミュレータに近い
  - gem5 よりも，より高度なシミュレーションが可能

gem5 と比べて複雑なパイプラインを再現可能



- gem5 による実行の様子.
  パイプラインはかなり
  単純化/理想化されている

- 鬼斬 による実行の様子.
  投機発行やリプレイのよう
  な動作を詳細に再現可能

鬼斬への RISC-V サポート

- RV32GA と RV64GA をサポート（主にユーザーモードのみ）
  - Linux 用 ELF バイナリをそのまま実行可能
- 鬼斬自体は 2006 年ごろ?から開発
  - もともとは Alpha と Power PC をサポート
- 命令セットへの課題：
  - Alpha：
    - 命令セットはクリーン
    - しかし最新のツールセットが使えなくなりつつある
  - Power PC：
    - Alpha が死んだ後のために実装
    - とても複雑．多数のマイクロ命令への分解が必要

鬼斬への RISC-V サポート

- RISC-V の登場は本当にありがたかった
  - ◇ Alpha は死につつあるし，PPC は複雑
  - ◇ ARM64 の実装に取りかかりつつあったが，仕様が大きくつらい
- RISC-V サポートは比較的容易
  - ◇ フレームワークや抽象化レイヤへの機能追加は不要だった
  - ◇ マイクロ命令への分解は原則必要無かった
- 最少の努力で，最新の環境が使える
  - ◇ 鬼斬の寿命が延びた

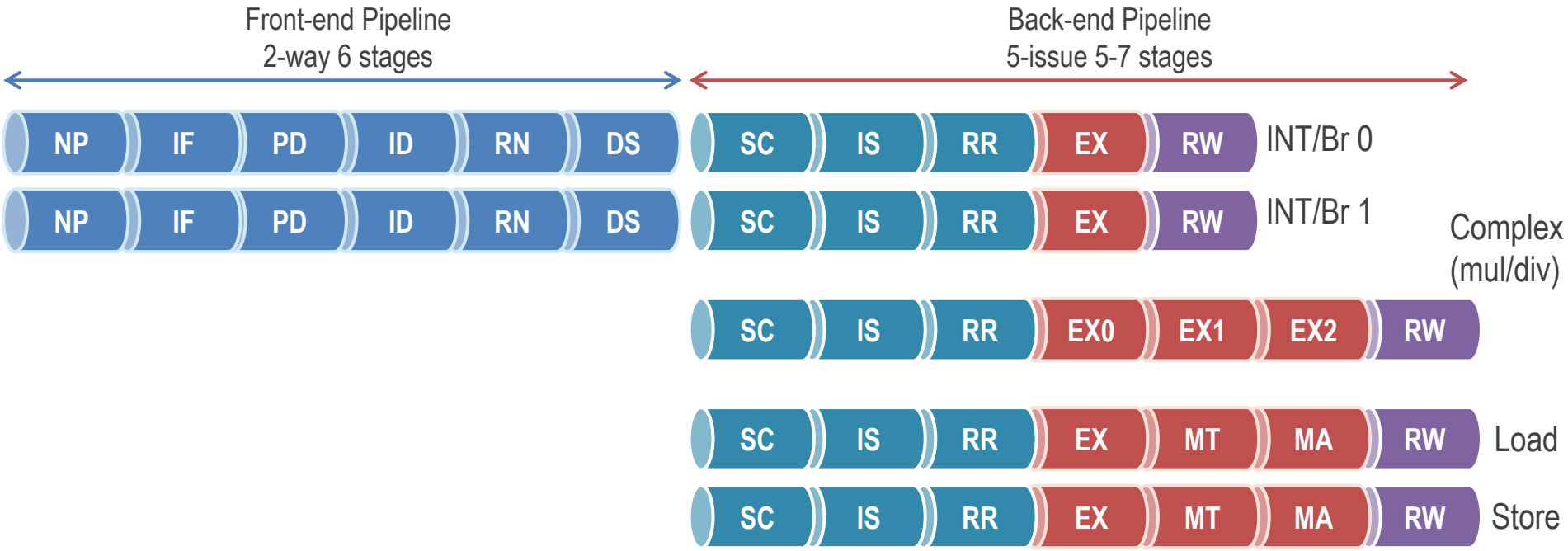
もくじ

1. シミュレータ鬼斬
2. **プロセッサ RSD**
3. アクセラレータ HORNET

プロセッサ RSD

- コンパクトかつ高速な RISC-V OoO スーパスカラ・プロセッサ
  - RISC-V RV32IM をサポート
  - SystemVerilog で 1 万行に達しないぐらいの規模
  - FPGA 上で BOOMv2 の 1/3，MicroBlaze の 5倍ぐらいの資源量
  - Dhrystone 実行時のクロックあたり性能で，BOOMv2 の 2.5 倍ぐらい
- 論文とリポジトリ
  - Susumu Mashimo et al., : An Open Source FPGA-Optimized Out-of-Order RISC-V Soft Processor, *IEEE Intl. Conf. on Field-Programmable Technology (FPT)*, 2019
  - https://github.com/rsd-devel/rsd

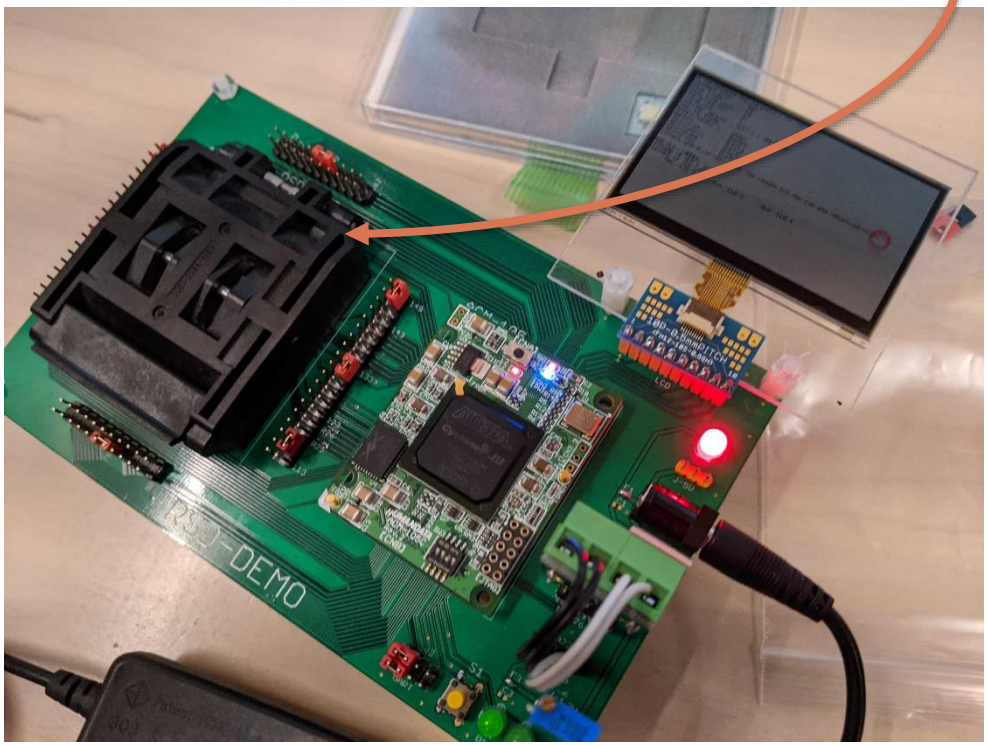
パイプライン構成

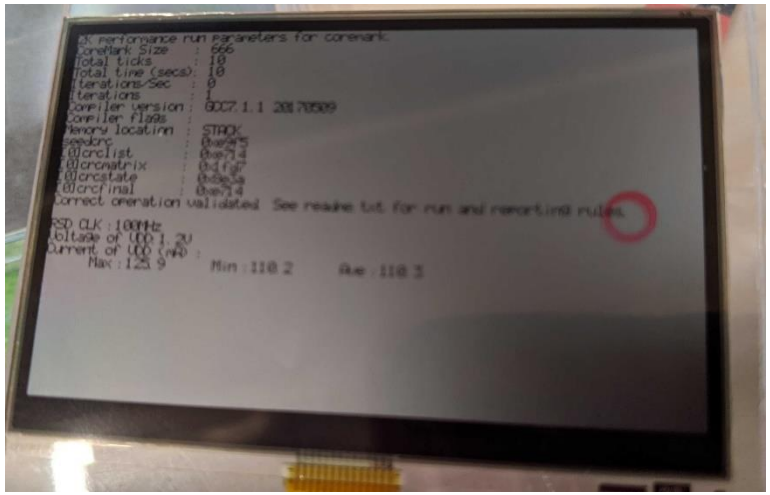


- マイクロアーキテクチャ（以下はデフォルト構成であり，柔軟に変更可能）
  - ◇ 最大 64 命令までスケジューリング可能
  - ◇ 5 命令同時発行可能

65nm プロセスでの ASIC 実装

- ルネサス 65nm で実装（2020）
  - ◇ ASIC 設計が初めてなため，動作させることを目的に単純化＆低速化
  - ◇ 100 MHz/130mw で無事動作（左が RSD，右は IO 用 FPGA）





- 協力：VDEC，株式会社ロジックリサーチ，天野先生，奥原さん

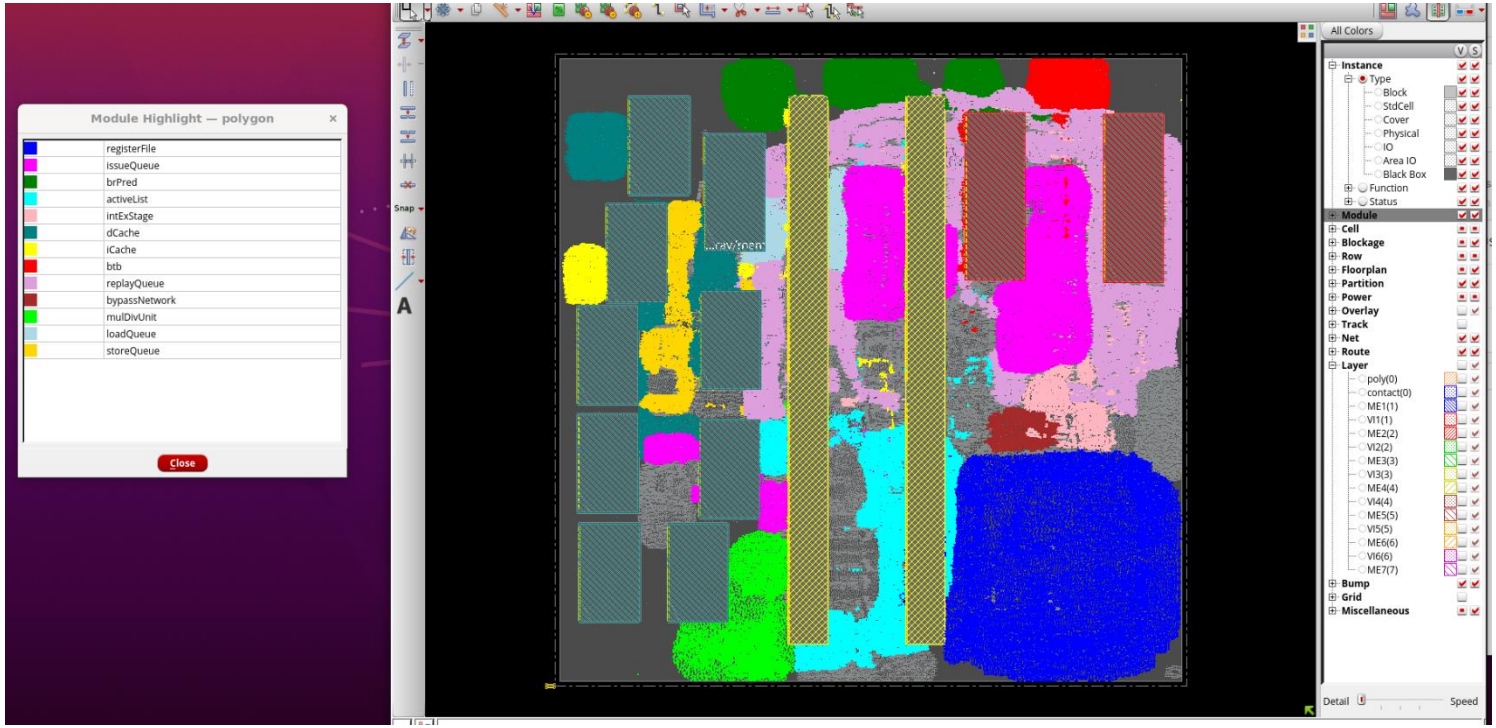
28nm プロセスでの ASIC 実装

- 28nm プロセスで実装中
  - NEDO プロジェクト
    「不揮発省電力FPGAコアを用いた低遅延AI処理コンピューティング技術の研究開発」（日本電気株式会社）の一貫
  - 設計中の SoC のコントローラとして RSD が採用
    - RSD そのものは通常のプロセスで実装

仮合成の結果
（4KB I/D cache + 5way OoO スーパスカラ構成）

- 論理合成時の遅延解析では 800MHz 超相当
  - ◇ クリティカル・パスは I-Cache の SRAM アクセスまわり
  - ◇ スーパスカラのパイプラインだけ見れば非常に高速

- レイアウト実験後の遅延解析では 500MHz 程度 / 0.58mm x 0.58 mm 角
  - ◇ SRAM の形が悪いため配線が伸びまくっている･･･



RSD での RISC-V 採用

- 経緯：
  - 元々は ARM のサブセットとして開発
  - 途中で RISC-V に移行
- RISC-V 採用の利点
  1. 公開ができる
  2. 大幅な簡単化

利点１　公開ができる

- ARM 互換時は，オープンソースとしての公開はできなかった
  - 当時の塩谷は問題ないと思い込んで ARM を選択してしまった
    - QEMU 等のエミュレータのソースが公開されていたため
  - ハードの設計は勝手に公開できない
    - 特許等で訴えられる
  - 内々で使うものになっていた
- RISC-V 採用により，大手を振って公開可能に

利点２　大幅な簡単化

- ARM の命令
  - ◇ 様々な機能が直交して使用できる
- その結果,
  - ◇ ロード命令であり,
  - ◇ アドレス計算の結果にさらに加算して並列にレジスタに書き戻し,
  - ◇ さらに条件付き実行であり,
  - ◇ 間接分岐命令（PC を書き換える）でもある
    ･･･のような命令が普通に現れる

利点２　大幅な簡単化

- スーパスカラのパイプラインではまともに処理できない
  - ◇ マイクロ命令への分解ではサポートしきれない
  - ◇ アセンブリコードの時点で複数の単純な ARM 命令に分解して無理矢理対応
- RISC-V 化により
  - ◇ マイクロ命令への分解自体が不要に
  - ◇ 命令デコーダが実装 4000 行から 1300 行に縮小
  - ◇ コンパイラ出力を無加工で実行可能に
- 最少の努力で，最新の環境が使える

もくじ

1. シミュレータ鬼斬
2. プロセッサ RSD
3. **アクセラレータ HORNET**

アクセラレータ HORNET

- 自動運転向けのアクセラレータとして開発
  - ◇ 株式会社アクセル，株式会社ティアフォーにより開発中
  - ◇ 塩谷はマイクロアーキテクチャに関与
- 現在の状況
  - ◇ 現状はFPGAで検証中
  - ◇ 2022年にテストチップの予定

背景：自動運転アルゴリズムと消費電力

- アルゴリズムの発展と，消費電力の肥大化
  - ◇ 現状でも高性能な CPU や GPU を使って処理する必要がある
  - ◇ 今後もさらに処理量が肥大化し続ける見込み
- 実際に売る車に搭載しようと思うと，相当に消費電力を削減していく必要がある

自動運転の処理

1. Sensing
   - ◇ センサー出力の前処理（フォーマット変換やノイズ除去など）

2. Perception
   - ◇ 現在の障害物の検知と将来の予測

3. **Localization ← 今回着目する部分**
   - ◇ LiDAR （レーザー照射による距離測定器）出力の点群と地図データのマッチングによる自己位置の計算

4. Planning
   - ◇ 上記を元に自分の動きをきめる

Localization の処理

1. NDT (Normal Distribution Transform) Scan Matching
   1. LiDAR から得られた点群から地図データ内の最近傍点を求める → ツリーのトラバース
   2. 評価値の計算と更新 → ある程度小さな行列演算

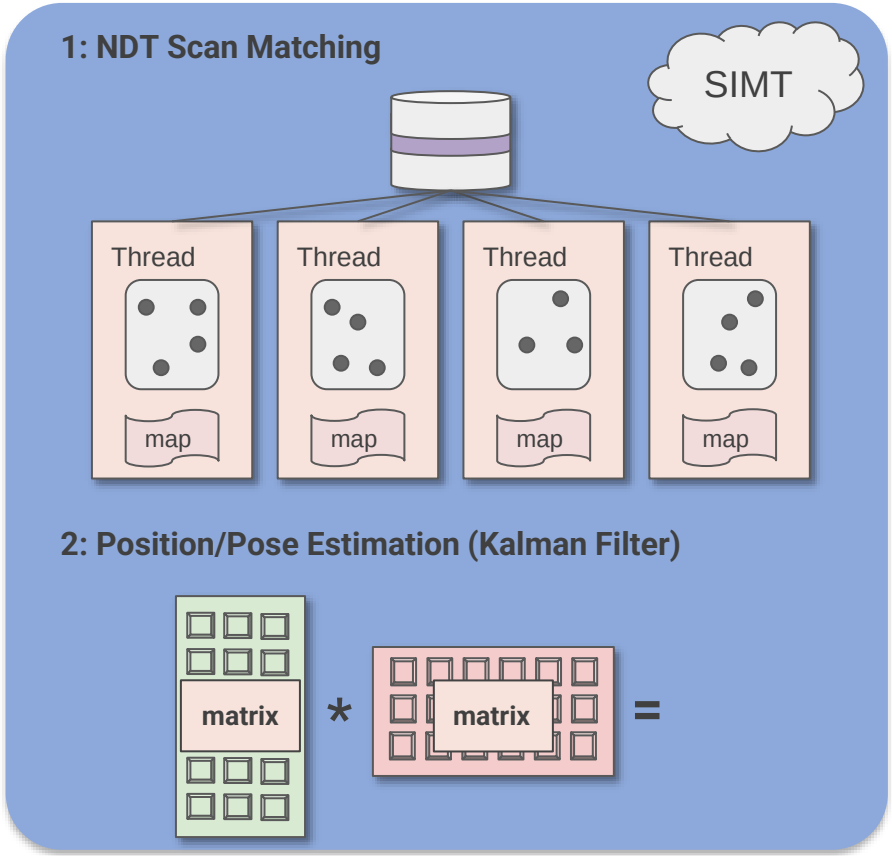
2. Position/Pose Estimation
   - ◇ Kalman Filter → そこそこ大きい行列演算

LiDAR の高精度化による出力点数の増加により処理量が肥大化
→ 自動運転の処理全体に占める割合がかなり大きい

https://velodynelidar.com/products/

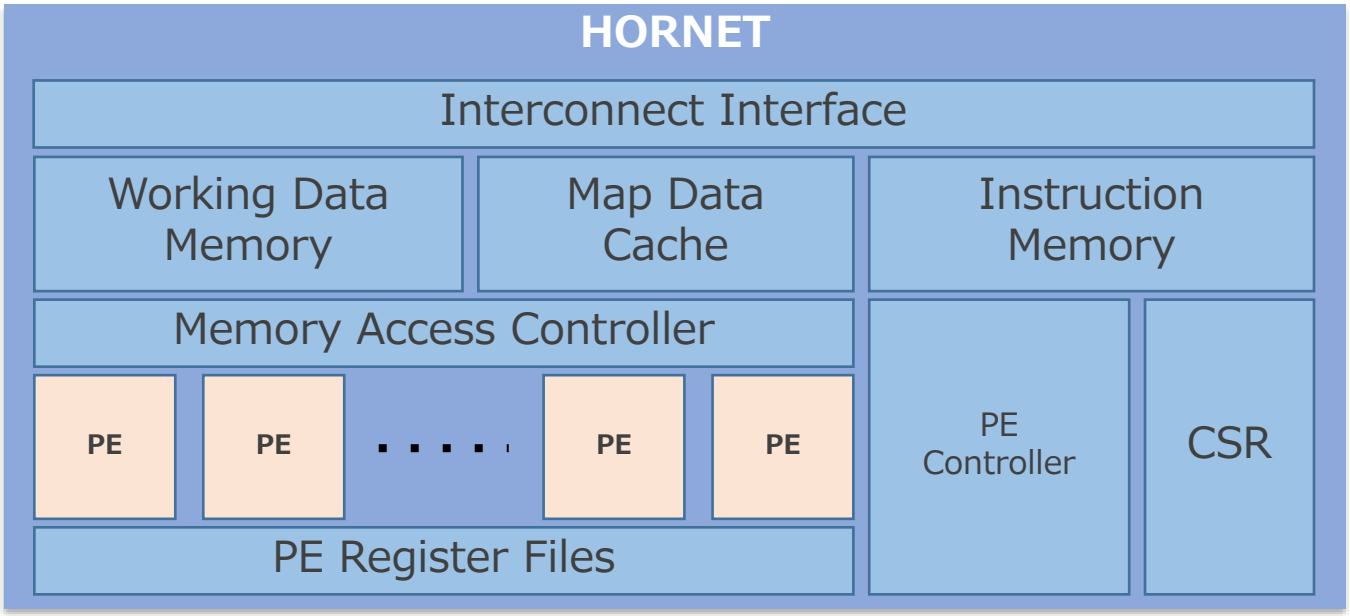
HORNET アーキテクチャ

- SIMT (Single Instruction Multiple Thread) ベースのアーキテクチャを採用
  - ◇ 単一命令で複数スレッドを駆動（複数データを処理）
  - ◇ 実行パスが分岐した場合は時分割でそれぞれ処理
- 狙い：
  - ◇ トラバースをマルチスレッドで効率良く処理
  - ◇ 行列は SIMD として効率良く処理
  - ◇ 専用回路ではなくプロセッサとすることで汎用性を持たせる



HORNET アーキテクチャ

- 比較的大きな読み出し専用キャッシュを搭載
  - ◇ 地図データなどの格納
- 汎用的な処理ができるため，その他の細かい前処理等も一括で処理可能
  - ◇ データ転送のオーバーヘッドを削減



アーキテクチャの特色

- ターゲットとする問題は，ある程度 GPU でも処理できるが･･･
  - GPU も SIMT アーキテクチャ
  - 既存の GPU では無駄が多い & 小回りが効かない
  - 分岐に弱い
- HORNET の特色：
  - ターゲットを絞って，シンプルなパイプライン構造で効率的に実現
    - GPU と比べて余計な機能を持たず，
      複雑なレイテンシ隠蔽機構などももたない
  - 分岐に耐性を持たせるために幅を狭く & cond. move 等の拡張

プログラミングモデルとツールチェイン

- 命令セットとして RISC-V を採用
  - SIMT 向けに拡張
- SIMT 用の API 拡張をした C 言語でプログラミング
  - Open CL の様なスキームでプログラミングが可能
- 既存の RSIC-V LLVM バックエンドを拡張して実装
  - コンディショナルムーブの追加など
  - 同期関係は atomic 拡張のものをそのまま使用

プログラミングモデルとツールチェイン

- LLVM バックエンドへの RISC-V からの変更量はかなり少ない
  - SIMT は通常の命令セットとはかなり実行モデルが異なる･･･ ような印象があった（塩谷個人の印象です）
  - かなり異なるコード生成器が必要だと考えていた
  - 実際に必要な変更はロード・ストアのアドレッシング関係ぐらい
    - 各レーンに暗黙に異なるオフセットが足し込まれるなど
- 最少の努力で，最新の環境が使える

まとめ

- RISC-V にかかわる事例の紹介
  - ◇ 鬼斬，RSD，HORNET
- RISC-V の研究上での利点：
  - ◇ オープンである
  - ◇ 最少の努力で，最新の環境が使える
- 個人的によかったこと
  - ◇ プロセッサを自分で作ってもよい空気にしてくれた
  - ◇ 大企業のみが作るものから，みんなが作るものに

謝辞

- 本研究は一部，東京大学ＶＤＥＣ活動を通して，シノプシス株式会社，メンター株式会社，日本ケイデンス株式会社の協力で行われたものである.

- この成果は一部，独立行政法人新エネルギー・産業技術総合開発機構（ＮＥＤＯ）の委託業務の結果得られたものである
（プロジェクト番号 JPNP16007）