RISC-V の特徴と研究・開発への利用

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

塩谷 亮太（しおや りょうた）

- 所属：
  - ◇ 東京大学 情報理工学系研究科 創造情報学専攻 （准教授）
- 専門：コンピュータ・アーキテクチャや基盤ソフトウェア
  - ◇ 特に CPU のマイクロアーキテクチャ
  - ◇ 最近は言語処理系やセキュリティ関係の研究も多いです

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
- 状況：
  - ◇ 研究や教育の領域では，ほぼデファクトに
  - ◇ 企業でも開発や実チップの販売にまで進みつつある

RISC-V

- RISC: Reduced Instruction Set Computer
  - ◇ 単純化した命令セットをもつコンピュータ
- RISC-V:
  - ◇ 「リスク ファイブ」と発音

> RISC-V (pronounced "risk-five") is a new instruction-set architecture (ISA)

  - ◇ UC Berkeley における 5 つめの RISC 命令セット

> *The name RISC-V was chosen to represent the fifth major RISC ISA design from UC Berkeley (RISC-I [16], RISC-II [9], SOAR [23], and SPUR [12] were the first four). We also pun on the*

引用は「The RISC-V Instruction Set Manual Volume I: Unprivileged ISA」 より

RISC-V 命令セットの基本的な特徴

- 基本的な構造 = シンプルな RISC：
  - ◇ レジスタ・ベースのロード・ストア・アーキテクチャ
  - ◇ 32 ビット固定長の命令
  - ◇ 32 本の汎用レジスタを持ち 3 オペランドをとる

他の命令セットとは何が違うのか？

- RISC-V
  - ◇ レジスタ・ベースのロード・ストア・アーキテクチャ
  - ◇ 32 ビット固定長の命令
  - ◇ 32 本の汎用レジスタを持ち 3 オペランドをとる
- これだけであれば，他の RISC 命令セットにも大体当てはまる
  - ◇ MIPS, Alpha
  - ◇ SPARC, ARM, POWER, SH, Open RISC ...

Q. RISC-V の何が良いのか？

A. 使いやすい

RISC-V の「使いやすい」ところ

1. フリーである
   - ◇ ラインセンスを気にせずに自由に作れる
2. 命令セットとしての使いやすさ
   - ◇ 余計な機能がなく，整理された命令セットの構造
   - ◇ モジュラー構造による機能の付け外しの自由度
     - □ 最小構成による単純な実装から，独自拡張までを広くカバー
3. ソフトウェア・スタックの充実
   - ◇ コンパイラや OS，デバッガからテストスイートまで

本日の内容

1. RISC-V の特徴と使い安さ

2. 大学での事例：
    1. 教育での使用例
    2. 鬼斬：シミュレータ
    3. RSD：スーパスカラ・プロセッサ

3. 企業での事例：
    1. PEZY-SC シリーズ：HPC 向けプロセッサ
       （株式会社 PEZY Computing）
    2. HORNET：自動運転向け SIMT 型アクセラレータ
       （株式会社 ティアフォー）

RISC-V 命令セットの基本

- 基本的な構造：
  - ◇ 命令は 32 ビット固定長
  - ◇ レジスタ・ベースのロード・ストア・アーキテクチャ
  - ◇ 32/64/128 ビット・アーキテクチャが用意

- 雰囲気的には,
  Alpha や （遅延スロットをなくした） MIPS にかなり近い

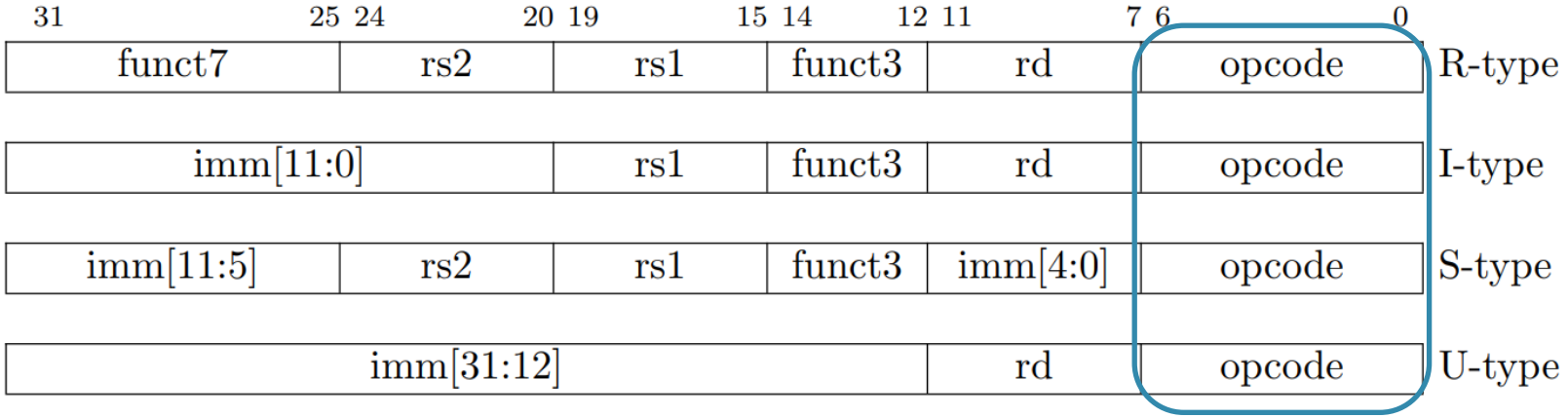
RISC-V の固有の特徴：
必須部分がコンパクトかつ，必要に応じて機能追加できる

- 整理された仕様：
  1. 必須となる整数演算部分は非常にコンパクト
  2. 用意された各拡張が独立に有効化/無効化できる

基本整数命令：
加減算，論理演算，ロード・ストア，即値設定，分岐とジャンプなど

- エンコーディング：R, I, S, U の4タイプがある
  - opcode によって，32 bit 中をどう区切って解釈するかが変わる
  - funct は追加の opcode (opcode が大分類，funct が小分類

- rs1, rs2, rd はオペランド
  - それぞれ 5bit: 2^5=32本のレジスタを指定可能
  - imm は即値



画像は下記より
The RISC-V Instruction Set Manual Volume I: User-Level ISA Document Version 2.2

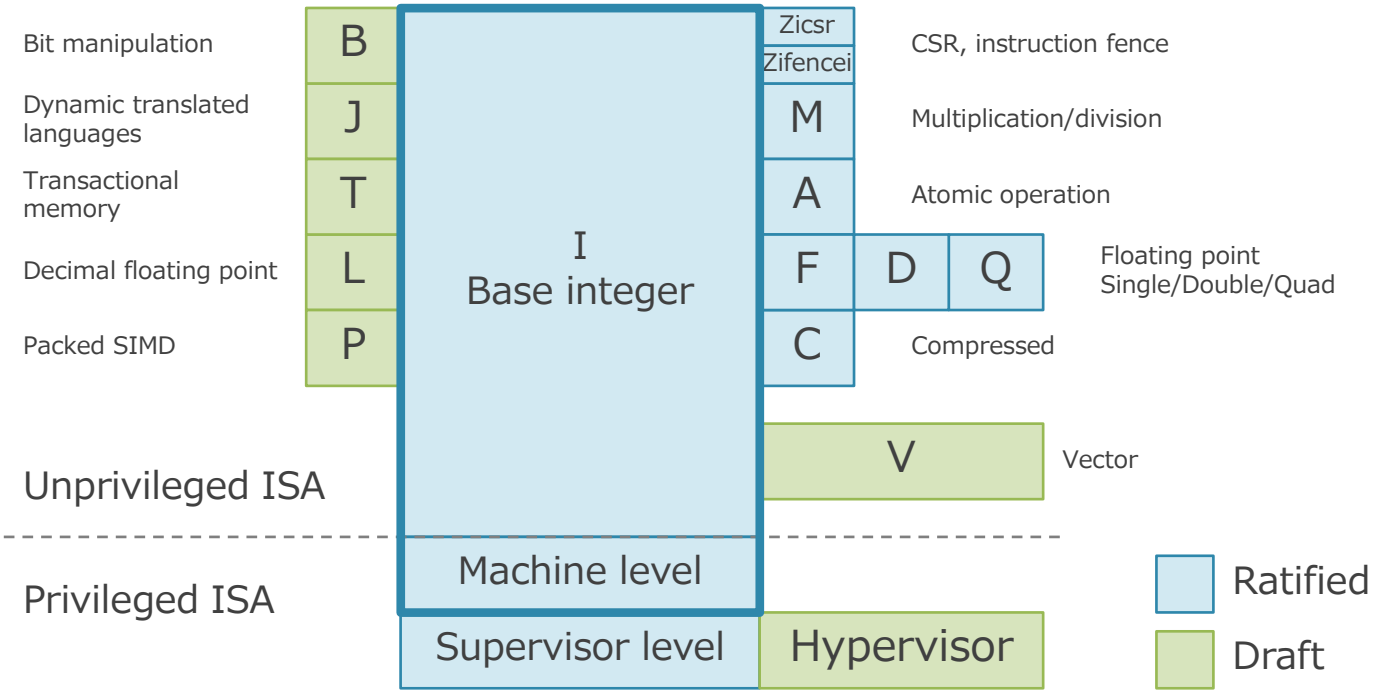
基本整数命令：
加減算，論理演算，ロード・ストア，即値設定，分岐とジャンプなど

RV32I Base Instruction Set

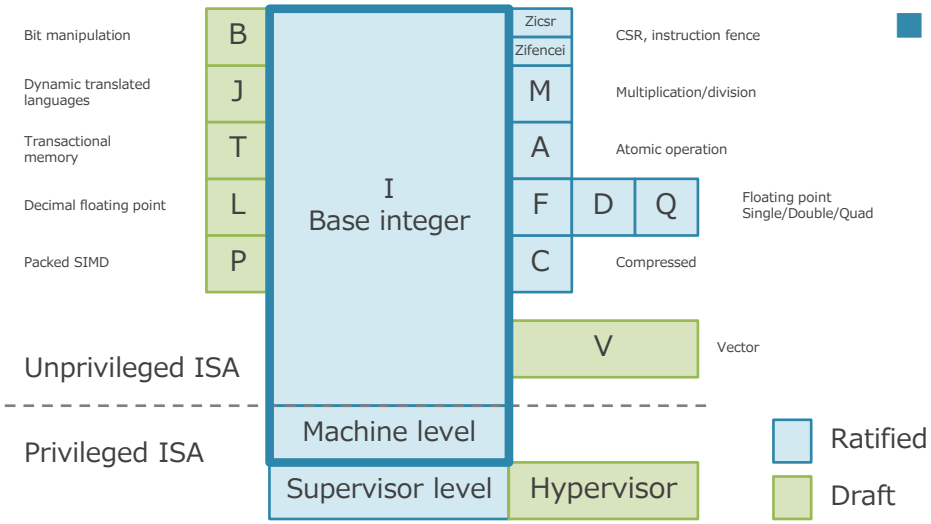
imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

RISC-V 拡張の概観

- 必須部分：下図の太枠部分
  - ◇ I（基本整数）+ Machine Level（割り込み関係の特権命令）
  - ◇ （この図に描ききれていない拡張もまだ結構あります



拡張部分



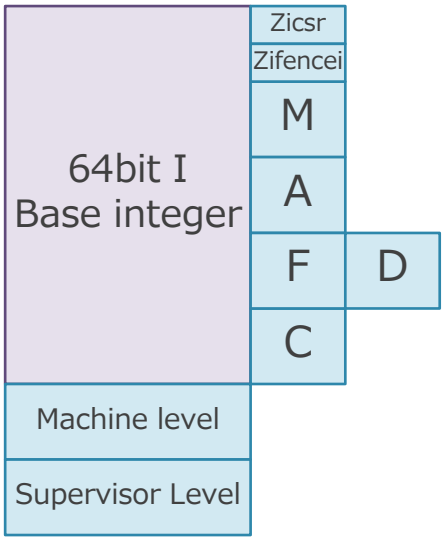
- 仕様が承認済み：
  - C：圧縮（ARM でいう thumb）
  - M：乗除算
  - A：アトミック（並列プログラム）
  - F, D, Q：浮動小数点
  - Zifencei：命令メモリ書き込みフェンス
  - Zicsr：制御レジスタ
- ドラフト段階：
  - L：十進浮動小数点
  - B：ビット操作
  - T：トランザクショナル・メモリ
  - J：動的コンパイル支援
  - P：パックド SIMD
  - V：ベクトル（SIMD じゃない）

RISC-V の拡張部分のいいところ

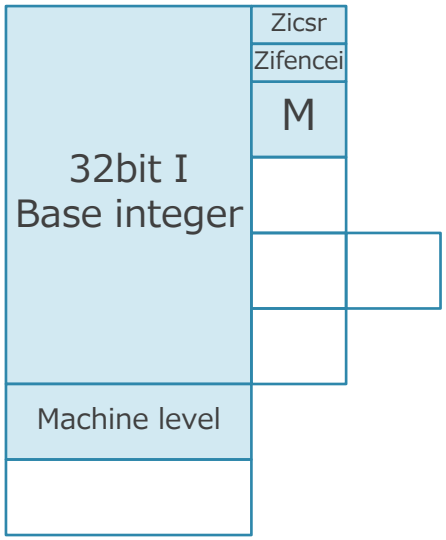
- 32bit/64bit や拡張部分の全体が最初から考慮されて作られている
  - ◇ 後付けの違法増築みたいなことになっていない
- 結果として
  - ◇ 命令コード内のオペランドのビット位置は全て共通
  - ◇ 各種の拡張も，最下位数ビットのみで容易に識別可能
- 拡張命令のモジュール性が高い
  - ◇ 現行の GCC などで簡単にオプション指定でコードを出せるので楽

プロセッサごとのサポート命令の例

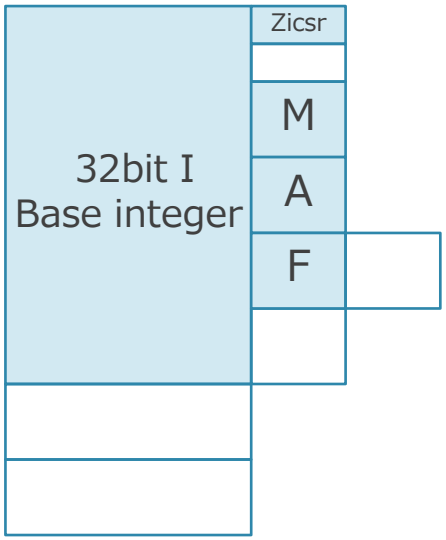
Rocket: RV64GC
(IMAD C Zifencei Zicsr)



RSD: RV32IM
(IM Zifencei Zicsr)

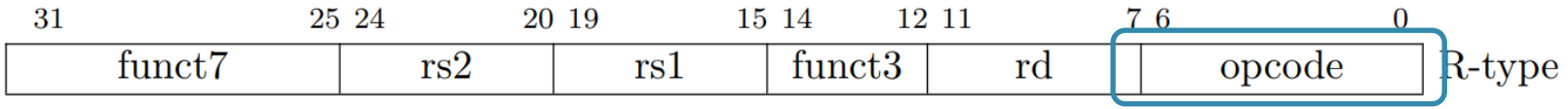


HORNET: RV32IMFA
(IMFA Zicsr)



- 拡張同士は基本的には自由に付け外し可能
  - Rocket と RSD はよくある組み合わせ
    - Linux 動作マシンと組み込み向け構成の典型
  - Hornet はかなり変則的
    - アクセラレータ目的の F（単精度 FP）と
      マルチスレッド対応のための A（アトミック） がある

ユーザーによる命令の拡張

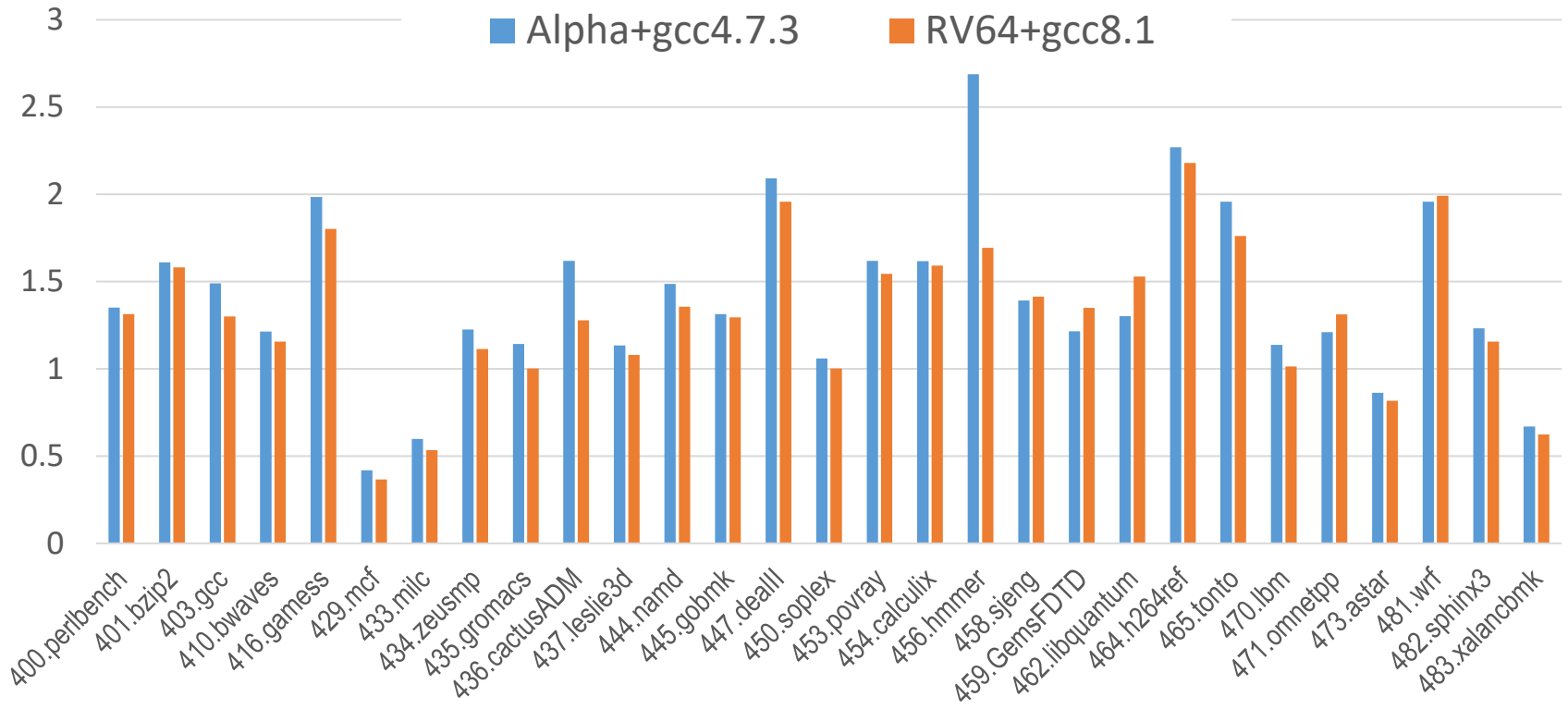


- opcode[6:0]: b0001011/b0101011 がユーザー用に確保されている
  - ◇ 安心して独自拡張が行える
  - ◇ 独自機能をいれつつも，既存のツールチェインの恩恵を受けれる
- 制御レジスタ（CSR）についても，ユーザー拡張用の領域が用意

RISC-V の性能面での特徴

- スーパスカラとしては Alpha とほぼほぼ同じ実装で作れる
  - ◇ ３オペランドの命令セットを普通に作れば良い
  - ◇ 典型的な RISC 命令セットの性能と特に変わらない（後述）
- 命令コードの容量効率は，C拡張（圧縮命令）採用時は x86 よりも良い
  - ◇ x86 は可変長なので密度が高いと思われがちだが，そうでもない
  - ◇ 多くの短いコードに今は使用されない命令が割り当てられている
    - □ 歴史的経緯による

Alpha と RISC-V の IPC の比較
6命令同時発行 OoO スーパスカラ, SPECCPU 2006 20G 命令実行時



- 全体の傾向はほぼ一致
  - hmmer のみ，RV64 の方が分岐が大きく増えて性能が落ちる
  - 一部 FP ベンチマークでは，Fused Multiply Add 命令の有無で差が付いている
  - （注意：高 IPC はこの比較では直接性能がよいことを意味しない

RISC-V の研究や開発での利点

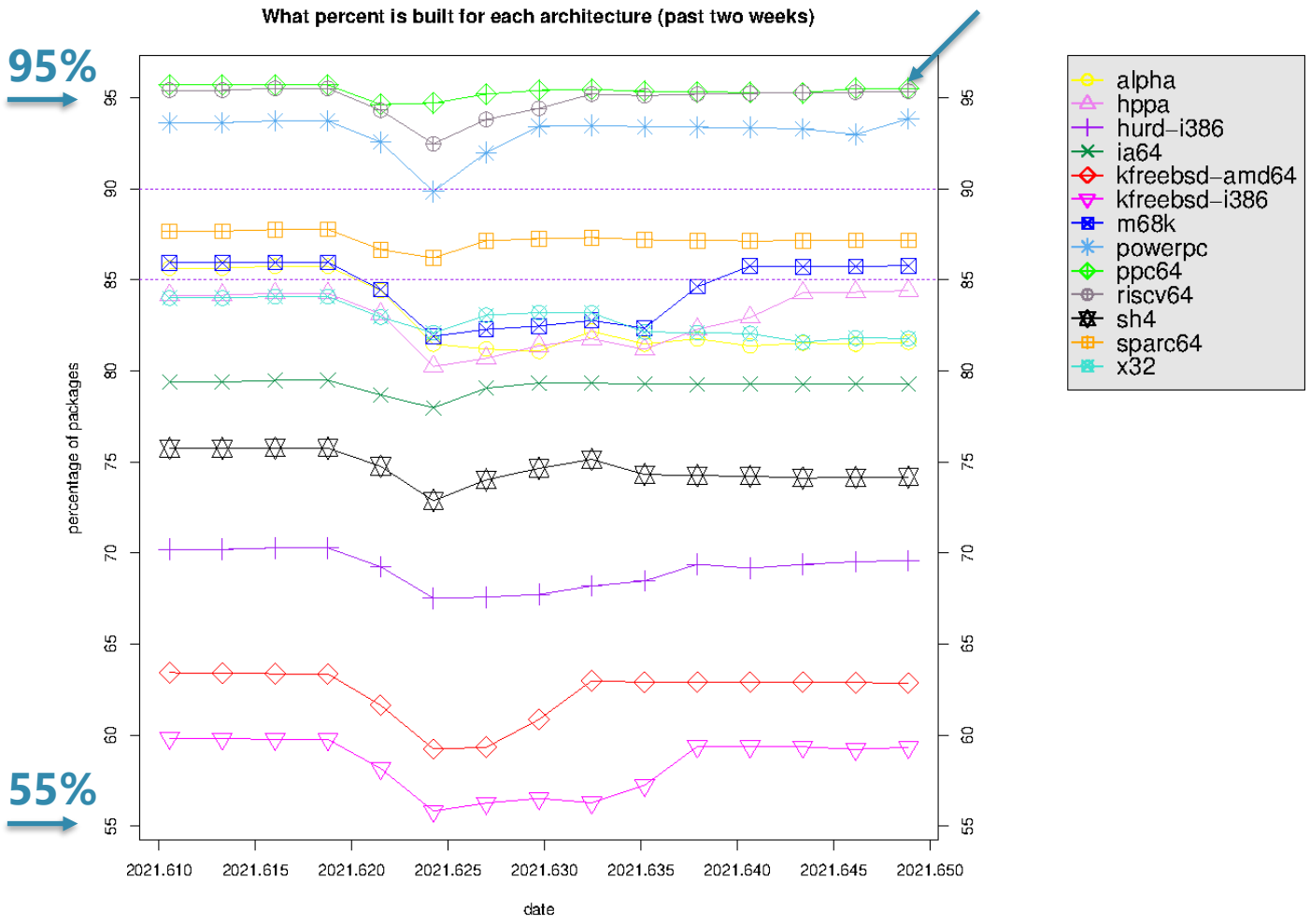
1. 必須部分が非常にコンパクトで，機能拡張も細かく追加可能
   - ◇ 最少の実装で，GCC などがそのまま使える
   - ◇ 目的に応じた取捨選択も自由度が高い

2. コンパイラやツール，OS が RISC-V を正式にサポートしている
   - ◇ GCC や LLVM，GDB，QEMU，Linux などの最新版が使える
   - ◇ RISC-V 公式のテストスイートも提供
     - □ riscv-tests: https://github.com/riscv/riscv-tests
     - □ riscv-arch-test: https://github.com/riscv/riscv-arch-test

Debian のビルド成功パッケージ率（2021/08/26 直近２週）

https://buildd.debian.org/stats/ より．だいたい 95% ぐらいはビルド済み

灰色が RISC-V



ソフトウェア環境に関する所感

- これまでにも，オープンな命令セットはあった
  - Open RISC など
- しかし，結局盛り上がらなかった
  - 周辺のツール・チェインの持続的かつ十分な提供がなかったため
  - 特定の Linux や GCC への移植にとどまり，使用も敷居が高い
- RISC-V は，この壁を越えた感がある
  - 主要なツールにおいて，本家のツリーでサポートされている
  - 企業が精力的・継続的にサポートに動いている

RISC-V の特徴と使い安さ

1. フリーである

2. 命令セットとしての使いやすさ
   - ◇ 整理された命令セットの構造
   - ◇ モジュラー構造による機能の付け外しの自由度

3. ソフトウェア・スタックの充実
   - ◇ コンパイラや OS，デバッガからテストスイートまで

大学での事例

1. 大学での事例：
    1. 教育での使用
    2. 鬼斬：シミュレータ
    3. RSD：スーパスカラ・プロセッサ
2. 企業での事例

教育面での取り組み

- 塩谷が担当する講義
  - 「先進計算機構成論」
    - https://github.com/shioyadan/advanced-computer-organization
  - CPU の仕組みを説明する際に，RISC-V を前提として説明
- 学生実験
  - CPU を作成する実験
  - 東大内では下記でそれぞれ実施
    1. 工学部 電子情報工学科（+電気電子工学科）
    2. 理学部 情報科学科

工学部 電子情報工学科

- 内容：
  - 単純な RISC-V のシングル・サイクル CPU を verilog で作成
  - HDL は全く知らない学生を想定し，10日間程度で実施
- 目標：
  - GCC でコンパイルした CoreMark の RISC-V バイナリの実行
- RISC-V 採用のメリット：
  - 基本整数命令のみのごく単純な実装で，C 言語をコンパイルしたバイナリが実行できる
  - MIPS では遅延スロットなどの再現のため，結構厳しい

理学部 情報科学科

- 内容：
  - ４人一組の班で，１学期かけて実施
  - CPU とコンパイラをセットで作成
  - 命令セットは自由だが，RISC-V と Power PC ベースが多い
- 目標：
  - 与えられた入力データに基づき，高速にレイトレーシングを行う
- 余興がある
  - RISC-V の MMU の実装と，その上で動く Rust で実装した オリジナル OS の発表など

教育面での今後

- これまで MIPS 等が担っていた位置を置き換えると思う
  - 基本命令セットの単純さと実装しやすさ
  - 周辺ソフトウェアの充実
  - 教科書の存在
    - パタヘネやヘネパタの RISC-V 版の登場
- 東大内の自分以外の講義や，他の大学でも取り扱うことが増えている

大学での事例

1. 大学での事例：
    1. 教育での使用
    2. **鬼斬：シミュレータ**
    3. RSD：スーパスカラ・プロセッサ
2. 企業での事例

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 も RISC-V をサポート

鬼斬への RISC-V サポート

- RV32G と RV64G 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
    - しかしツールチェインがもう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
- RISC-V の単純さとソフトウェア環境の恩恵を受けられる
  - 鬼斬の寿命が延びた

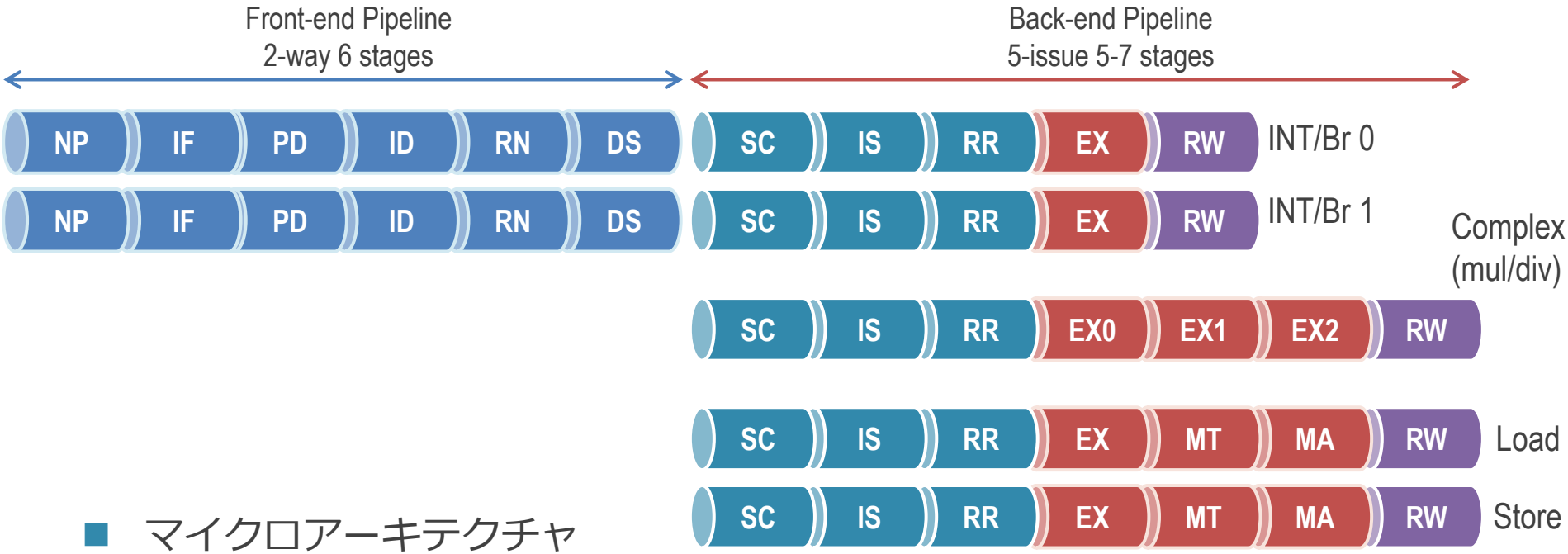
大学での事例

1. 大学での事例：
    1. 教育での使用
    2. 鬼斬：シミュレータ
    3. **RSD：スーパスカラ・プロセッサ**
2. 企業での事例

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
- 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

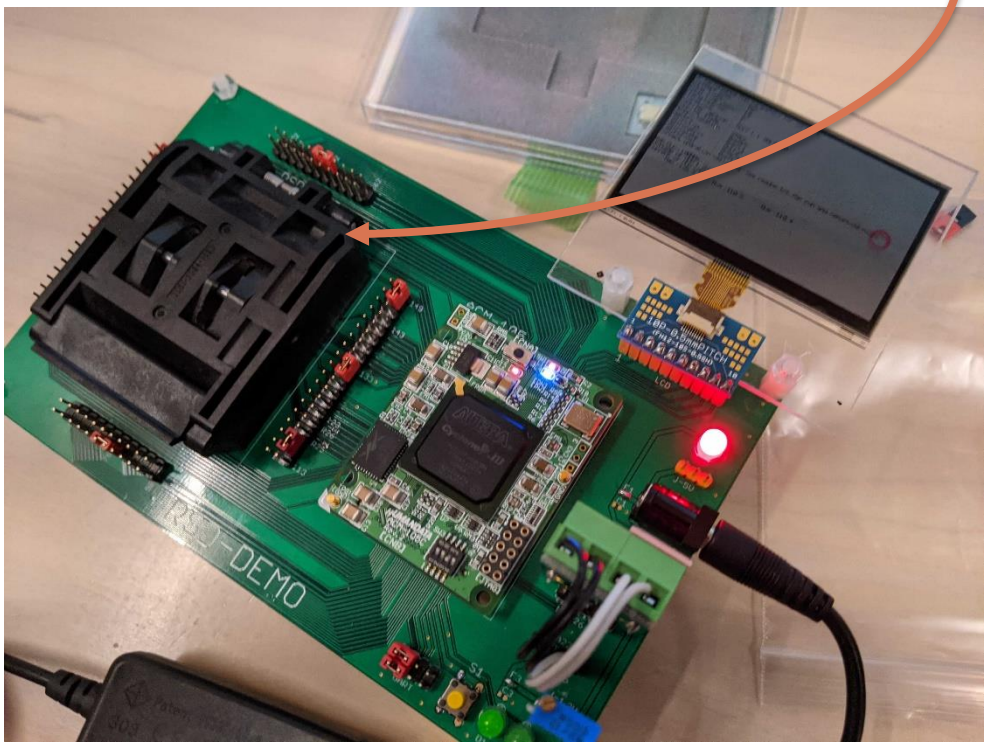


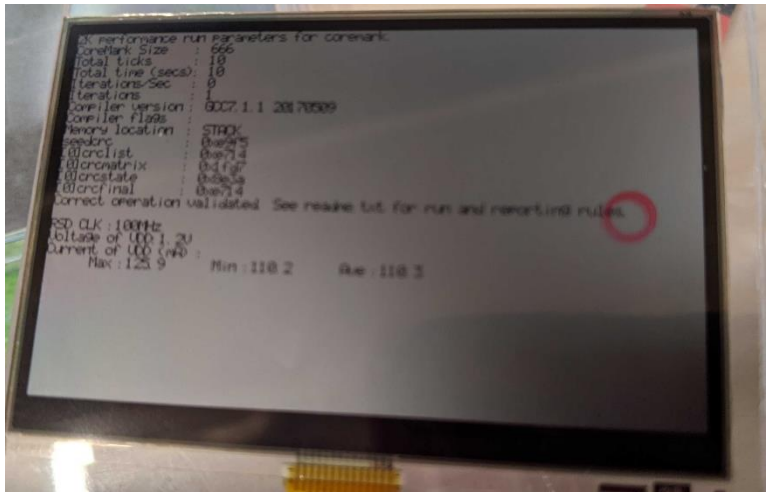
- マイクロアーキテクチャ
  （以下はデフォルト構成であり，柔軟に変更可能）
  - ◇ 最大 64 命令までスケジューリング可能
  - ◇ 5 命令同時発行可能

- 大きさや性能等
  - ◇ 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

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/130 mW 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
  - ◇ 他のプロジェクトで設計中の SoC のコントローラとして RSD が採用
- 遅延解析では 800MHz 超相当
  - ◇ クリティカル・パスは命令キャッシュの SRAM アクセスまわり
    - □ レイアウトが悪い
  - ◇ スーパスカラの命令スケジューリングや並列実行部分などは非常に高速

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

RSD での RISC-V 採用でよかったこと

- RISC-V がフリーであることや，その単純さ，ソフトウェア環境の恩恵を受けられた

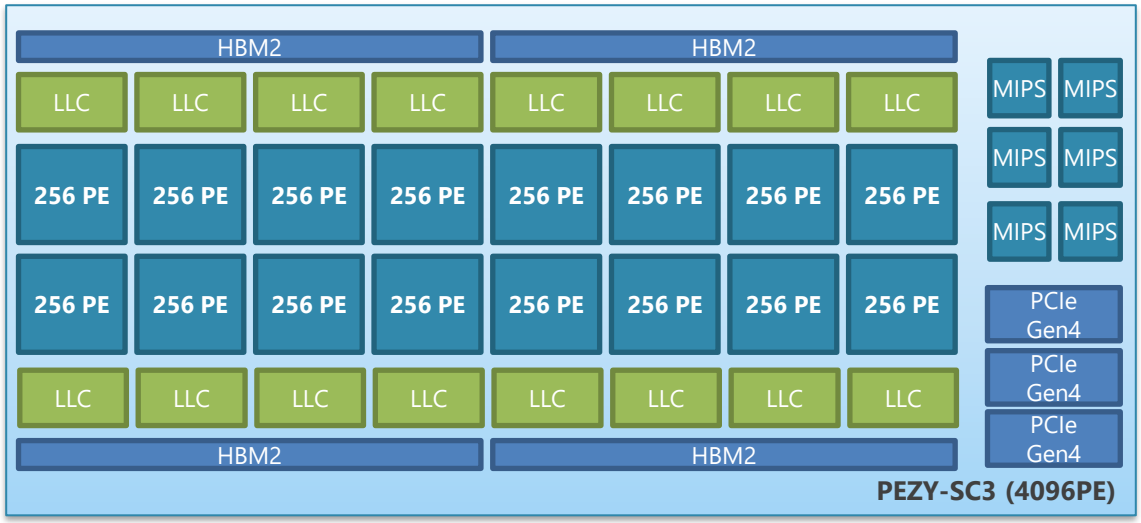
企業での使用事例

1. **PEZY-SC シリーズ：HPC 向けプロセッサ**
   （株式会社 PEZY Computing）

2. HORNET：自動運転向け SIMT 型アクセラレータ
   （株式会社 ティアフォー）

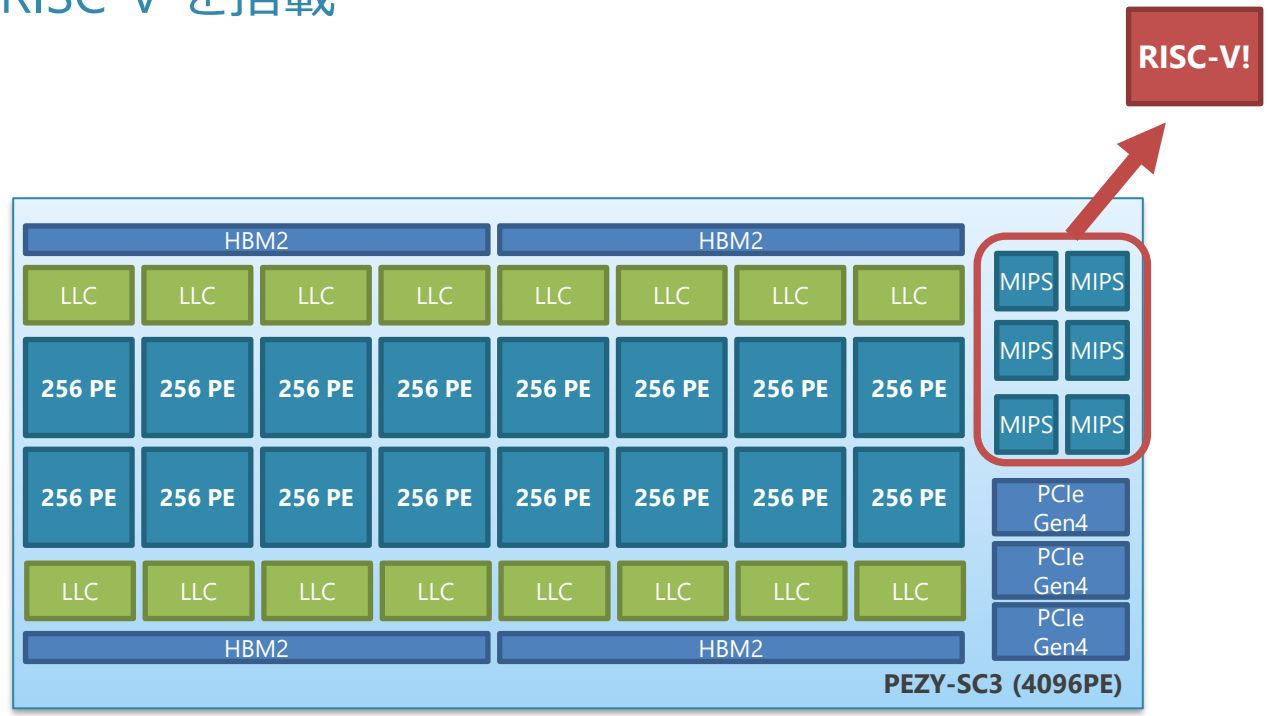
PEZY-SCx シリーズ

- HPC やゲノム解析などに適したプロセッサ
  - ◇ メニーコア・アーキテクチャ
    - □ 独自 ISA の Processing Element（PE）を持つ
  - ◇ OpenCL ライクな開発環境
- とても高い電力効率を持つ
  - ◇ PEZY-SC2 はスパコンの電力効率を競う GREEN 500 で世界 1 位に



PEZY-SCx シリーズ

- 最新の PEZY-SC3 まで複数チップをリリース
  - ◇ これまで制御用 CPUとして ARM や MIPS を搭載してきた
- 今後は RISC-V を搭載



PEZY-SCx シリーズに搭載する RISC-V

- 制御用の CPU として，デバッグやモニタに使用
  - Rocket Core ベースのコアを採用
- 設計中のチップ
  - 7nm プロセス / 1 GHz 動作 → 物理設計完了
  - 5nm プロセス / 1.5 GHz 動作 → 論理設計中

面積や性能など

- 面積が小さい：
  - 200um 角に収まる（7nm 時）
- 性能：MIPS 時の 60% 程度であるなら制御 CPU としては十分
  - MIPS（4-way OoO）：　　CoreMark/MHz = 3.06
  - Rocket（Scalar In-order）：　　CoreMark/MHz = 2.07

Rocket Core 採用の目的と利点

- 仮想メモリ方式として Sv39 と Sv48 に対応している：
  - 物理メモリのアドレス幅が広い
  - PEZY-SCx の内部バスのアドレス範囲が広く，32-bit CPUでは不足していた
- アドレス・マップ，バス幅，ID幅，割り込み本数などの変更が容易
- Linux のブートに対応している：
  - PCIe RootComplex の検証ができる
  - スタンドアロンなアプリの可能性もひらける

PEZY-SCx シリーズにおける RISC-V

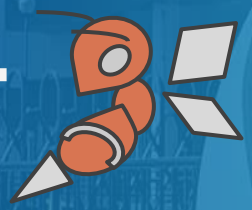
- PEZY Computing としての RISC-V の魅力
  - ◇ プロジェクトに合わせた選択肢がある
  - ◇ 商用，OSS，自社開発など

企業での使用事例

1. PEZY-SC シリーズ：HPC 向けプロセッサ
   （株式会社 PEZY Computing）

2. **HORNET：自動運転向け SIMT 型アクセラレータ**
   （株式会社 ティアフォー）

HORNET



RISC-V Based SIMT-Style Localization Accelerator for Autoware

Tier IV

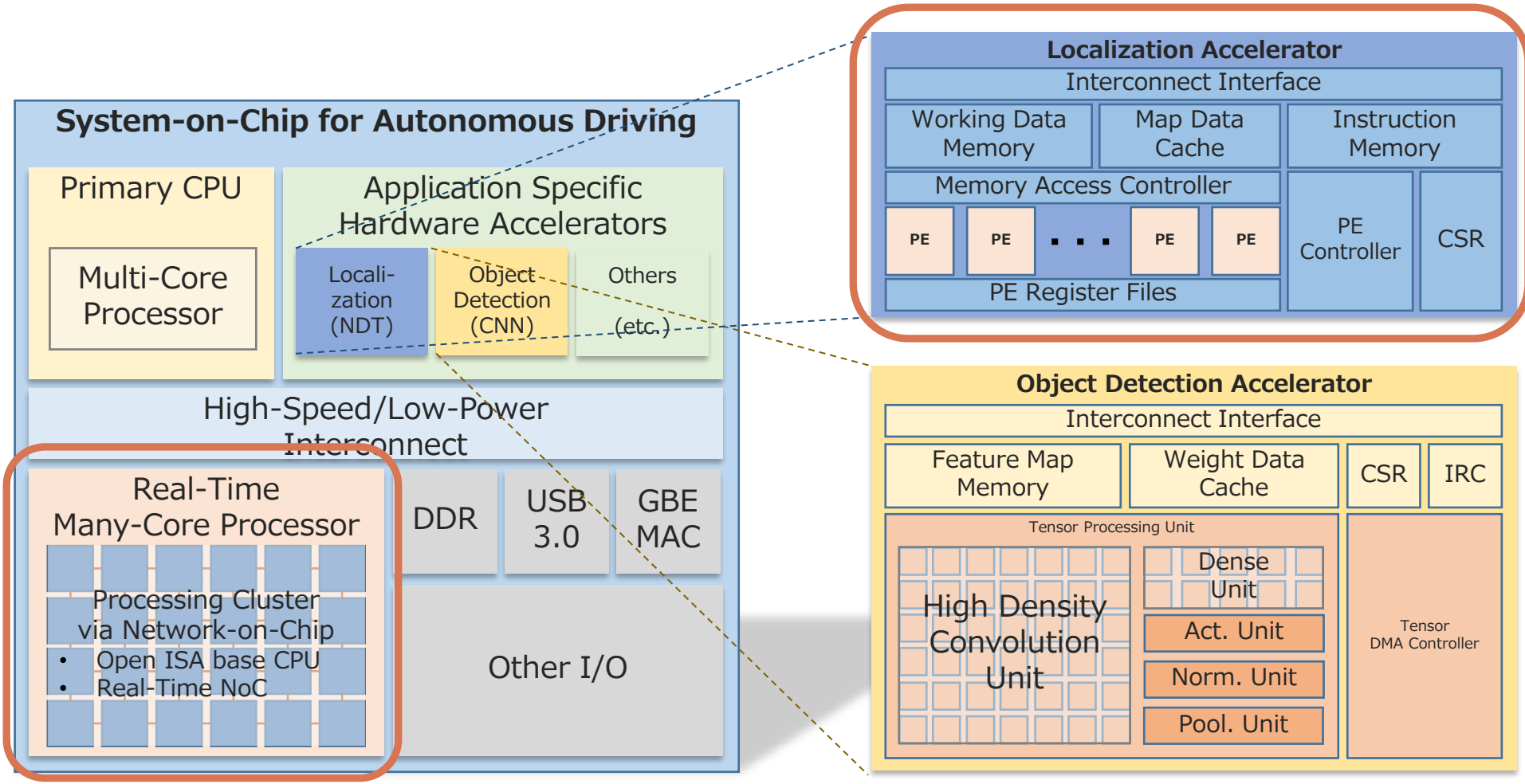
アクセラレータ HORNET

- 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
- アプローチ：アクセラレータで処理して消費電力を減らす
  - ◇ そのための SoC を研究・開発中

Autoware-Defined AI Accelerator Design Concept



- ◇ 太枠部分が RISC-V ベース
- ◇ 今回は右上の Localization Accelerator が主題

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

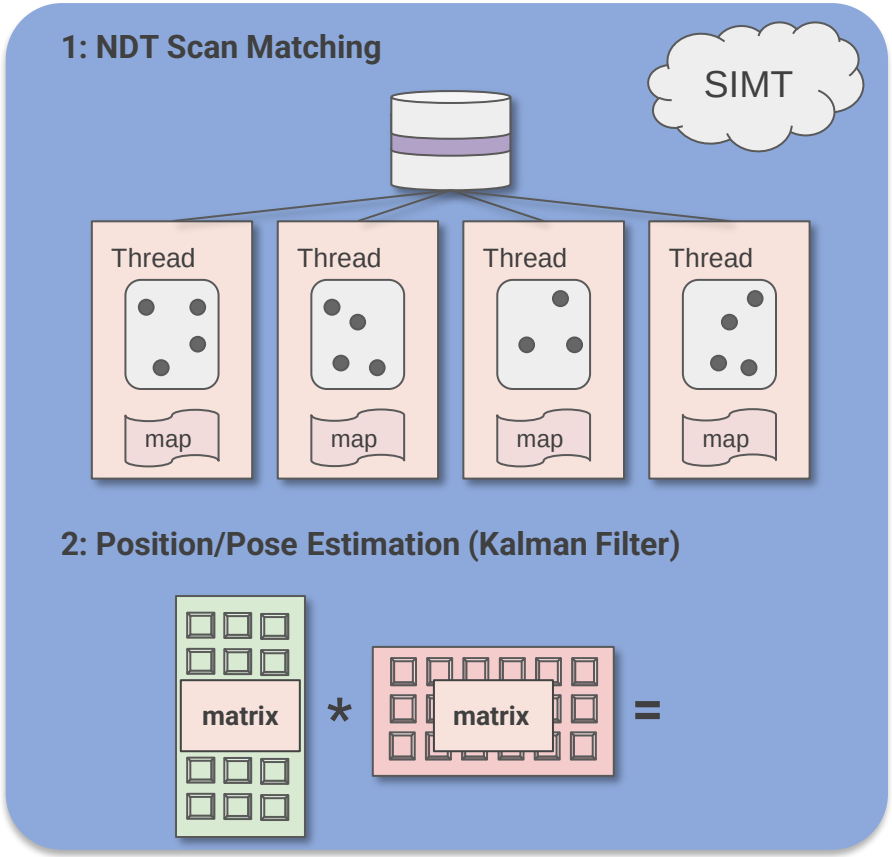
2. Position/Pose Estimation
   - ◇ Kalman Filter → そこそこ大きい行列演算

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/

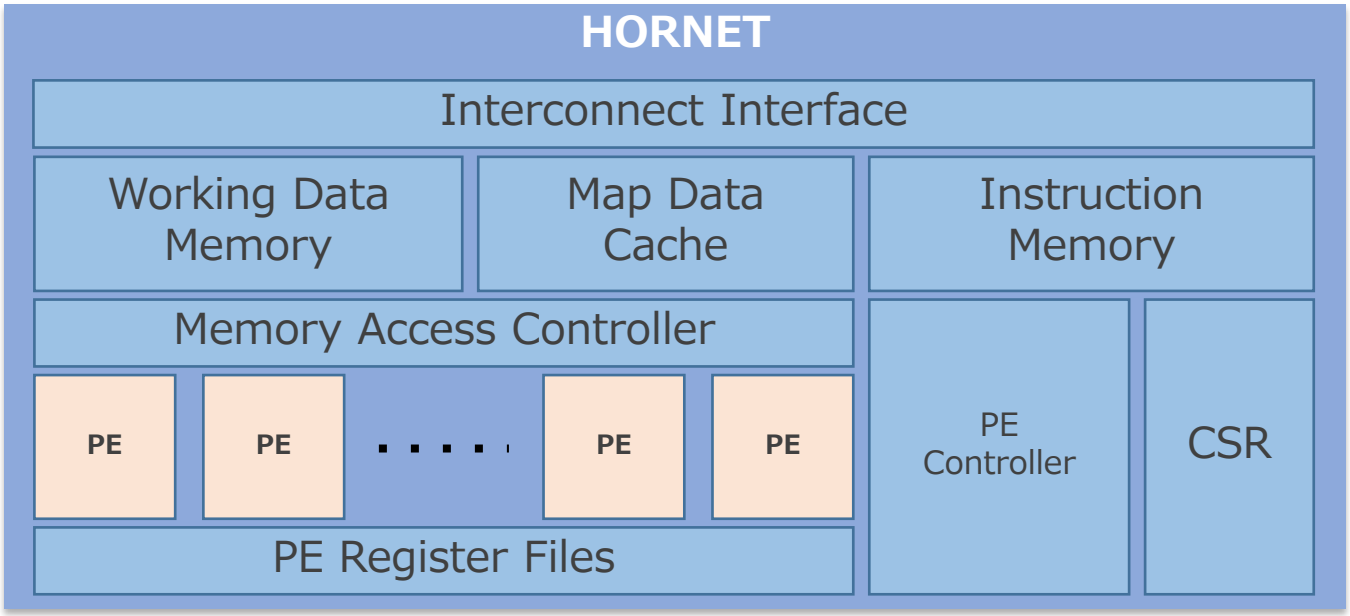
HORNET アーキテクチャ

- 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 RISC-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
- RISC-V がフリーであることや単純さ，ソフトウェア環境の恩恵を受けられる

まとめ

1. RISC-V の特徴と使い安さ
    1. フリーである
    2. 命令セットとしての使いやすさ
    3. ソフトウェア・スタックの充実
2. 大学での事例
    1. 講義や実験，シミュレータ鬼斬，プロセッサ RSD
3. 企業での事例：
    1. PEZY-SC シリーズ：HPC 向けプロセッサ
       （株式会社 PEZY Computing）
    2. HORNET：自動運転向け SIMT 型アクセラレータ
       （株式会社 ティアフォー）

まとめ

- RISC-V の，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

- 本発表に含まれる研究の一部は，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す．

- 本発表に含まれる研究の一部は，JSPS 科研費 20H04153 による助成のもと行われたものです．

- 資料提供に協力していただいた株式会社 PEZY Computing と株式会社ティアフォーのみなさまに感謝いたします．