IBM Research - Tokyo

電子情報学特論：
ハードウェアの高速化を支える
ソフトウェア技術

Hiroshi Inoue (井上 拓)
IBM Research – Tokyo

プロフィール

- Research Staff Member
  @ IBM Research – Tokyo (東京基礎研究所)
  - 2002年4月～現在

- 博士 @ 東大 情報理工 田浦研
  - 2013年4月～2016年3月 社会人博士課程修了

- 修士 @ 慶應 理工学研究科
  - 2002年3月 修了（流体力学）

研究テーマ

- ソフトウェアのパフォーマンス関連全般
  - 最適化コンパイラ (IBM Java, LLVM)
    - CGO 2011, OOPSLA 2012 etc
  - 並列処理アルゴリズム (特にSIMD命令活用)
    - PACT 2007, VLDB 2015 etc
  - メモリ管理 (malloc, GC)
    - PLDI 2009, ISMM 2012 etc
  - その他，システム性能評価，　OSなど
  - （趣味）deep learning アルゴリズム
    - AISTATS 2019 etc

論文詳細： http://ibm.co/inouehrs

Agenda

- **Introduction: A New Golden Age?**
- Topic 1: Software stack for a deep learning accelerator
- Topic 2: Algorithms for SIMD (vector) instructions

A New Golden Age for Computer Architecture

- John L. Hennessy and David A. Patterson
  2017年Turing賞受賞記念講演
  - 半導体技術の改善によるコンピュータシステムの性能改善は限界が近づいている
  - 2000年代に入り，ムーアの法則やデナード則が破れてきている
  - Domain-specific architecture (DSA)による高速化の時代が到来 ➔ コンピュータアーキテクチャの専門家が活躍できる黄金時代

🤔アーキテクチャの新たな黄金時代にソフトウェアの役割は？

Free lunch is Over

- Herb Sutter (Software architect @ Microsoft)
  The Free Lunch Is Over
  A Fundamental Turn Toward Concurrency in Software (2005 ← Intel Pentium D発売の年)
  - 半導体技術の向上でプロセッサのシングルスレッド実行性能が向上するので，ソフトウェアが高速化を考えなくても良い（free lunch = タダ飯）時代は終わった
  - 性能向上のためにはプログラムの並列化が必須に

➔時代は更に進み，DSAの時代には，単に並列化するだけでなく，DSAの新しい機能を活用するためのソフトウェア技術の開発が必要になっている (hardware-software co-design)

高速化のためのハードウェアの例

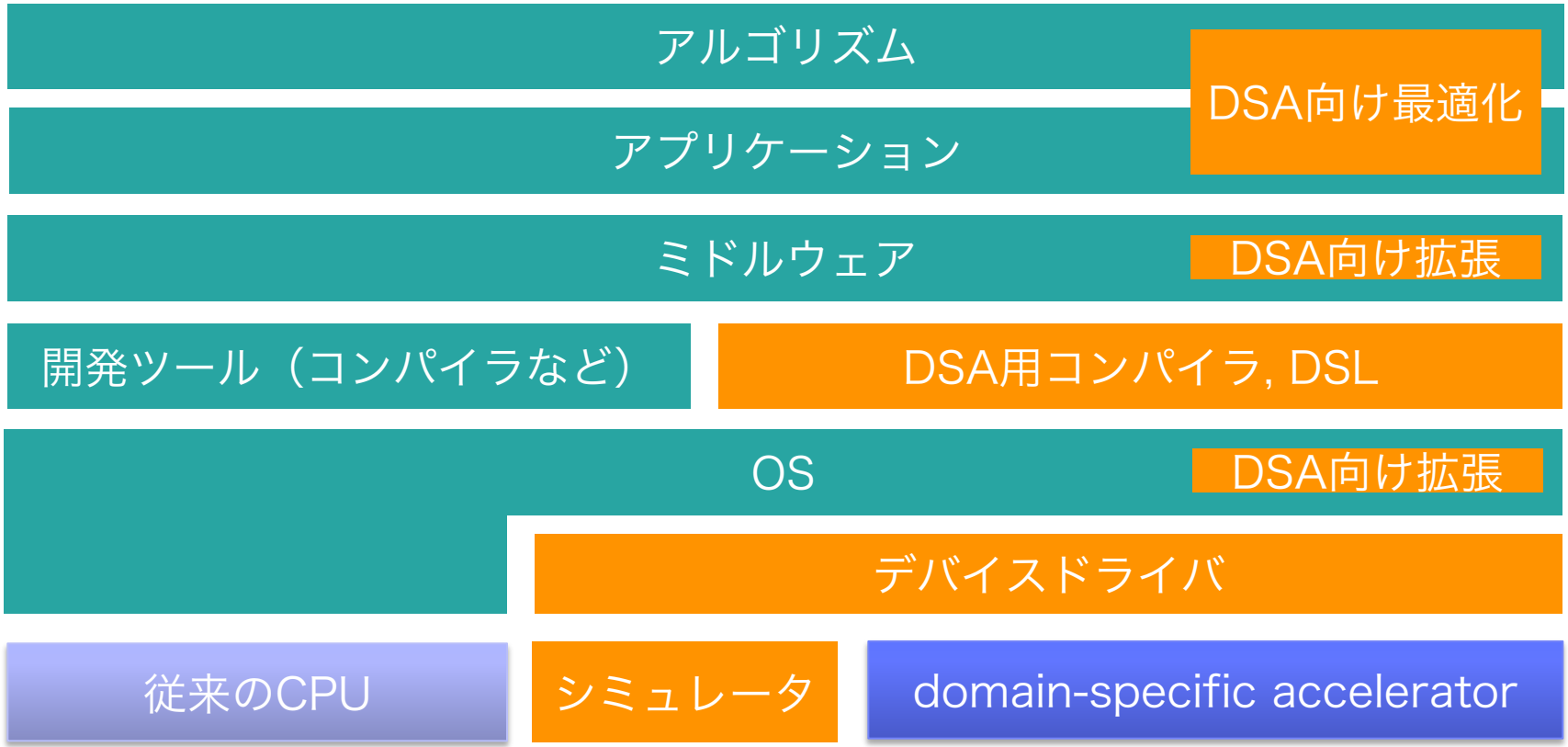
- 高い並列度の汎用プロセッサ
  - スレッド並列度: マルチコア，SMT
  - データ並列度: SIMD命令 (例えばIntelのAVX, SSEなど), 行列計算命令 (Intel AMX)
- GPU
- FPGA
- ASIC (domain-specific acceleratorというとたいていここを指す)
  - Deep Learningアクセラレータ (GoogleのTPU, PFNのMN-Coreなど)
  - Bitcoin miningアクセラレータ・暗号アクセラレータ

- 量子コンピュータ
  - ゲート方式量子コンピュータ (IBM Q, Googleなど)
  - quantum annealer (D-Waveなど)
- spiking neural network chip, analog neural network chip

アクセラレータの実装方法

- CPU内の命令として実装
  - 例：SIMD命令 (Intel AVXなど)，暗号処理命令（AES-NI）
- CPUとメインメモリをcoherentに共有するコプロセッサとして実装
  - 例：AMDのAPU, OpenCAPI接続のFPGA
- CPUと外部接続のインターフェースを経由して接続
  - 例：PCI-E接続のGPU，USB接続のEdge TPU

アクセラレータのためのソフトウェアスタックの例



今回は高速化のためのアクセラレータの話ですが

- プロセッサの進歩は高速化目的に限らない
  - データ保護のためのTrusted execution environment (例えばIntel SGX, ARM TrustZone)
- 高速化技術はプロセッサに限らない
  - 高速な不揮発メモリ（Intel 3D XPoint）
  - 広帯域メインメモリ（HBM）
  - 高速インターコネクトネットワーク（京・富岳のTofu）

Agenda

- Introduction: A New Golden Age?
- **Topic 1: Software stack for a deep learning accelerator**
- Topic 2: Algorithms for SIMD (vector) instructions

Deep Learning アクセラレータ

- ニューラルネットワークの処理は学習でも推論でも，多くの計算時間を畳み込み(convolution)や行列積計算に費やす

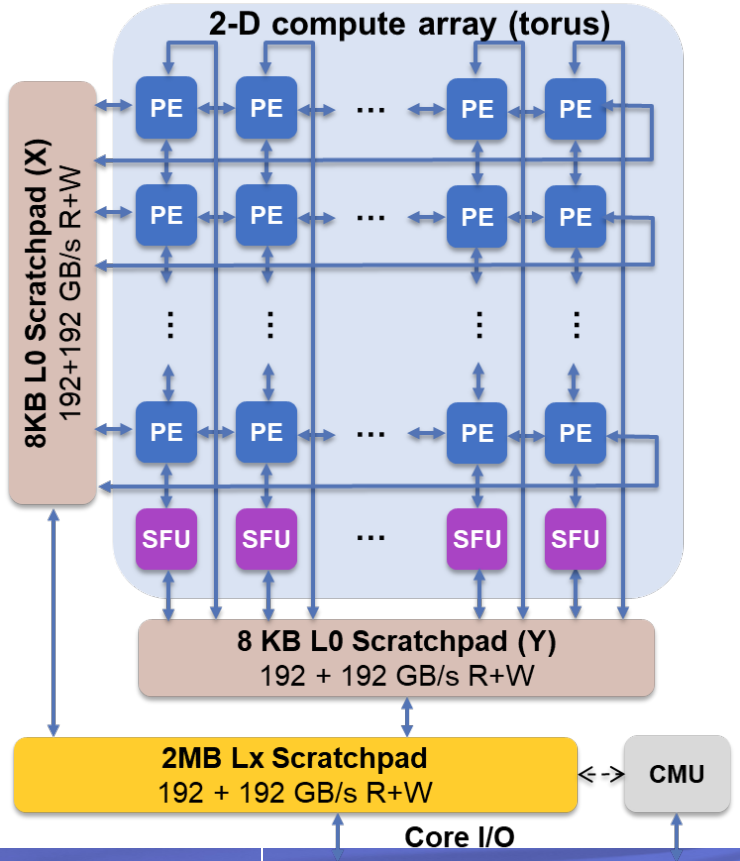
- 演算の特徴：
  - 演算の並列度が高い
  - 低精度の浮動小数点（場合によっては整数）演算でも，アルゴリズム次第で最終的な結果としては十分な精度が得られる

➔倍精度(fp64)や単精度(fp32)の浮動小数点でなく，簡素な半精度(fp16)の演算器を高密度で並べることで高速化が可能

➔様々なアクセラレータが提案されている（例 TPU, MN-Core）

IBM RaPiD AI Accelerator

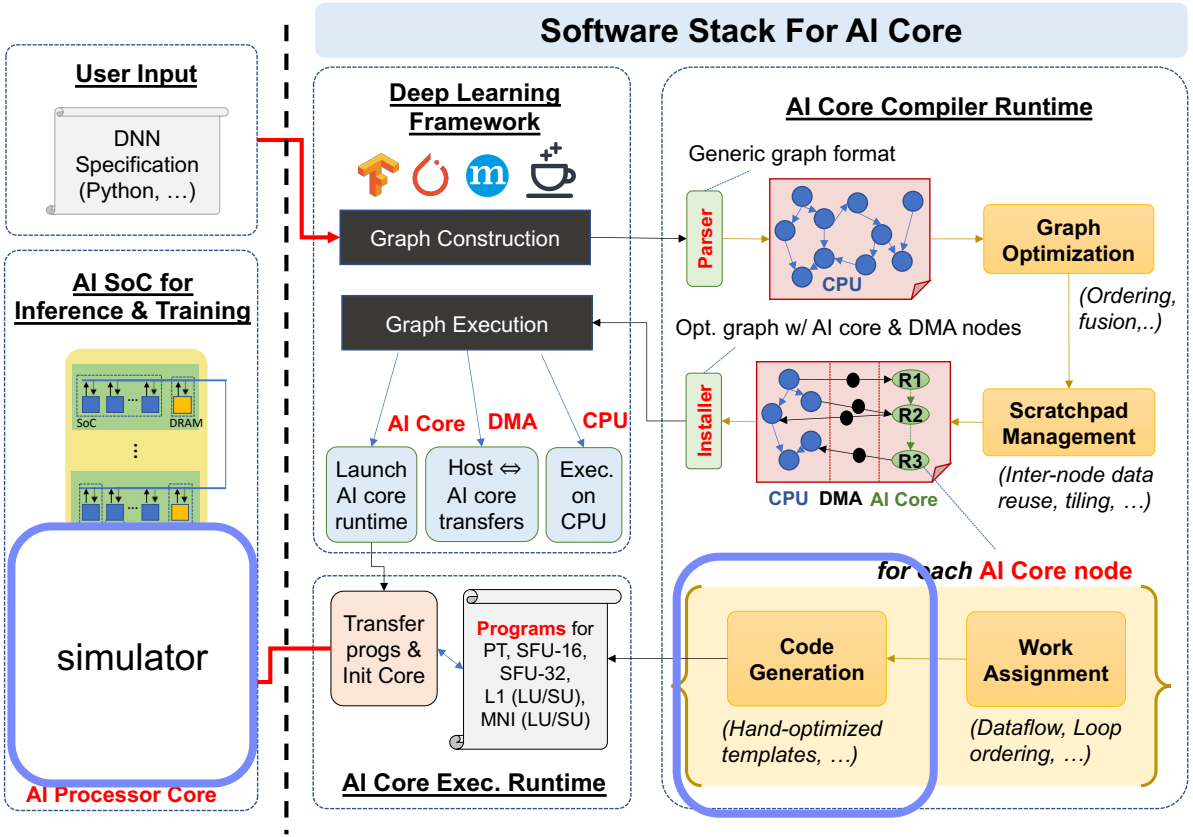
1st generation core design



- ハードウェアの特徴
  - Convolutionと行列積計算用のシストリックアレイ演算器とその他の処理用のvector演算器を持つ
  - データ移動をソフトウェアで行う階層的なスクラッチパッドメモリを使用
  - DLFloat16浮動小数点フォーマットを採用
- ソフトウェアから見た特徴
  - 演算もデータ移動も全ユニットがプログラマブル．一つのコアあたり44個(2nd gen)!!
  - 命令セットもユニットの種類ごとに違う

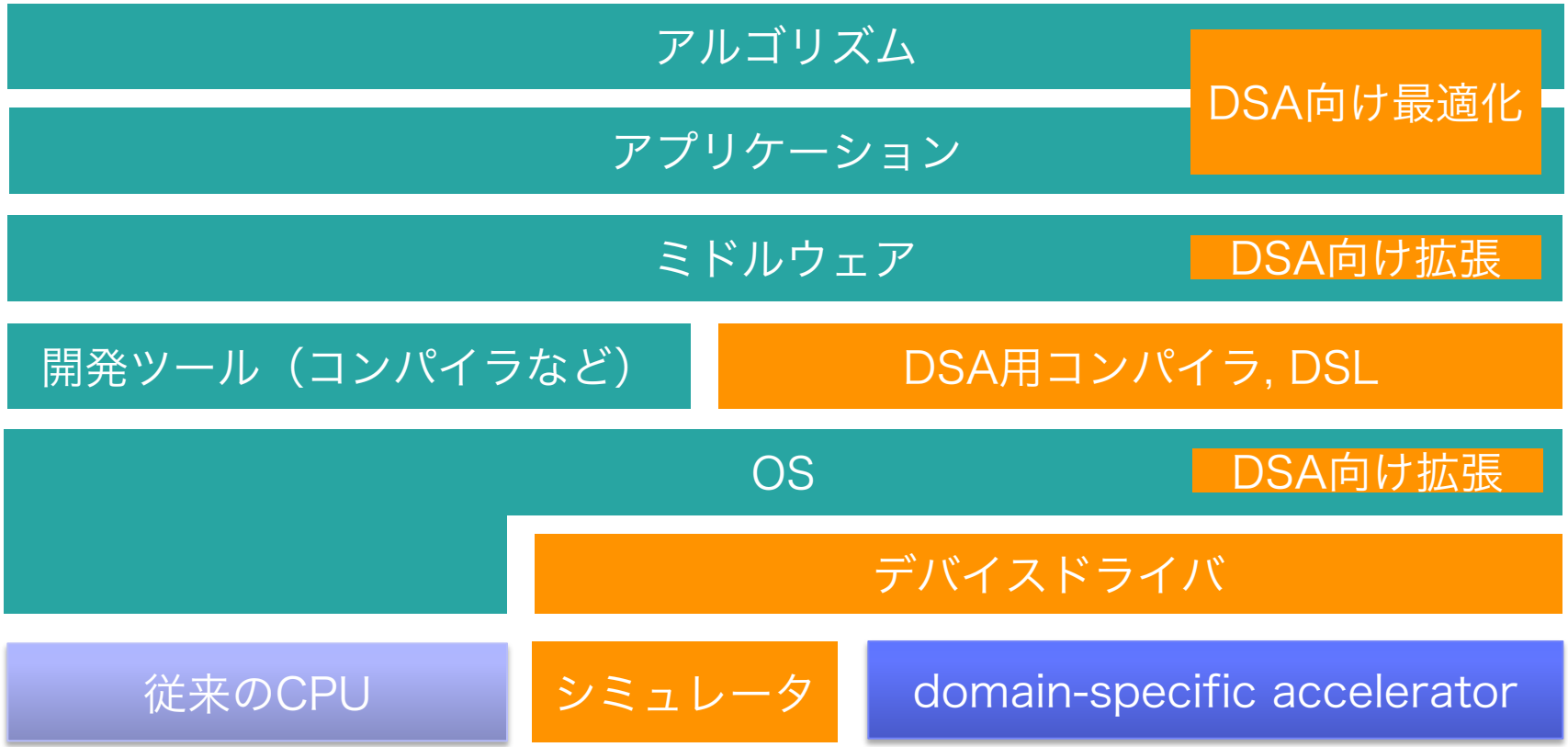
  ➔ハードの性能を引き出すには（というよりも動かすには），コンパイラなどを含めたソフトウェアスタックのサポートが必須
- Ref: B. Fleischer,*et al.*, VLSI 2018, J. Oh, *et al.*, VLSI 2020

DeepTools: RaPiDのためのソフトウェアスタック



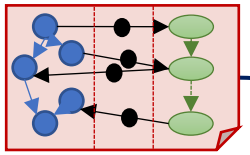
- TensorFlowの計算グラフの各ノードをRaPiD用にコンパイルする
- コンパイルされたコードとデータをRaPiDに転送し実行する
- Ref: S. Venkataramani *et al.* IEEE Micro 2019, E . Ogawa et al., COOLChips 2019

アクセラレータのためのソフトウェアスタックの例



コンパイラの役割

TensorFlow
計算グラフ



対象ノードの情報

・conv2d, kernel=3x3
・w=224, h=224, c=64, ..
・padding = same
・stride = {1,1,..}

スケジューラ

・処理の複数コアへの分割
・スクラッチパッドメモリの管理
・ループの順番，ブロッキング，
データ転送位置の最適化

パラメータ

・ループの順番
・ブロッキングのサイズ
・各スクラッチパッドの
メモリ割り当て
・...

コンパイラ

・スケジューラの出力するパラメータと，最適化された手書きの最内ループのコード（テンプレート）を元に実行コードを生成する

テンプレート

アセンブラで書かれた
最内ループでの演算内容

各ユニット用実行コード

ベクタ演算ユニット用

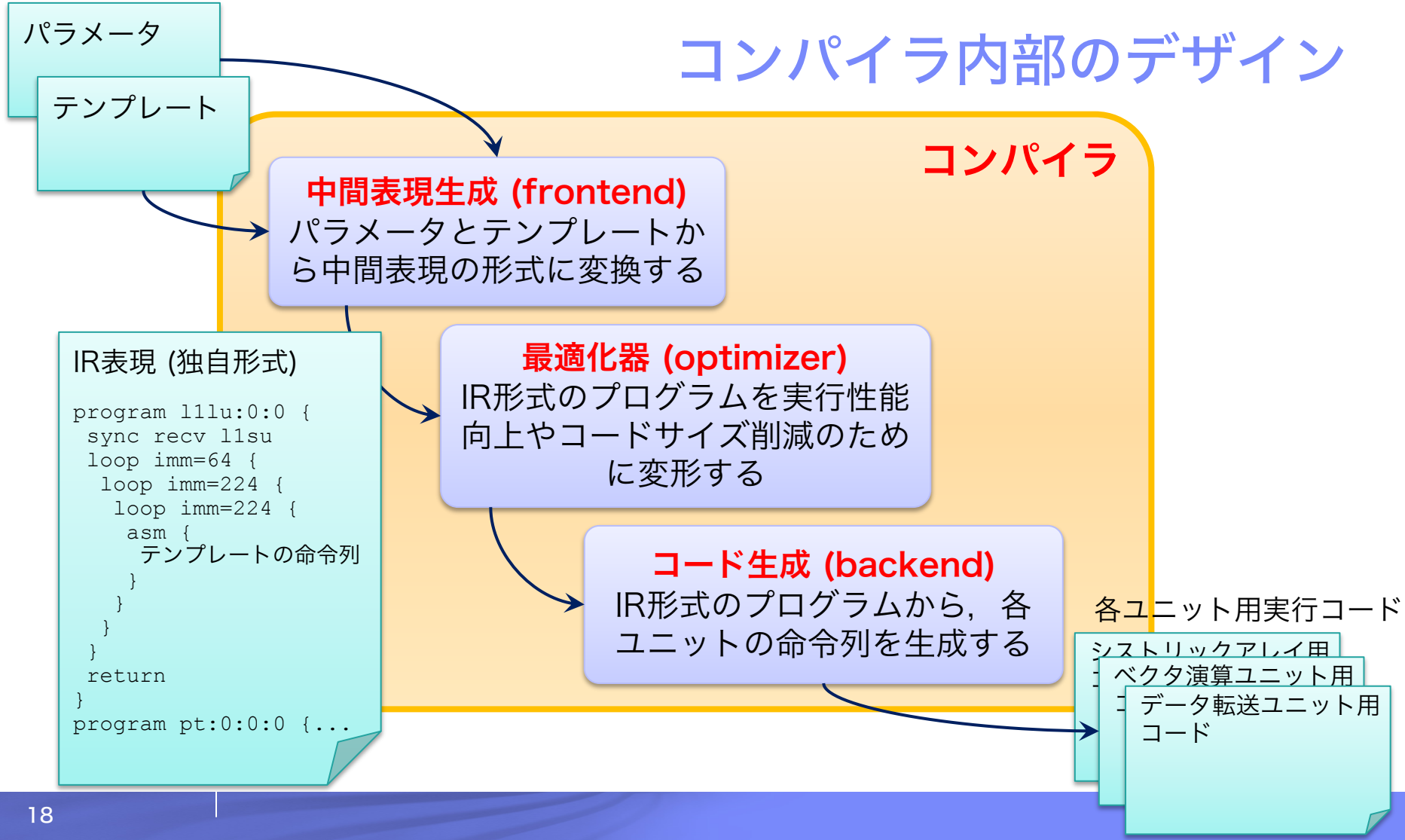
データ転送ユニット用
コード

DeepToolsでのコンパイラの特徴

- 最内ループで実行されるコードは最適化された手書きアセンブラのテンプレートを用いる
  →ハードの性能を限界まで引き出す

- 外側のループやデータ転送，ユニット間同期はコンパイラがパラメータに応じて自動的に生成する
  →パラメータの変化に柔軟に対応する
  →転送と演算の重ね合わせなどの複数ユニットにまたがる最適化が可能

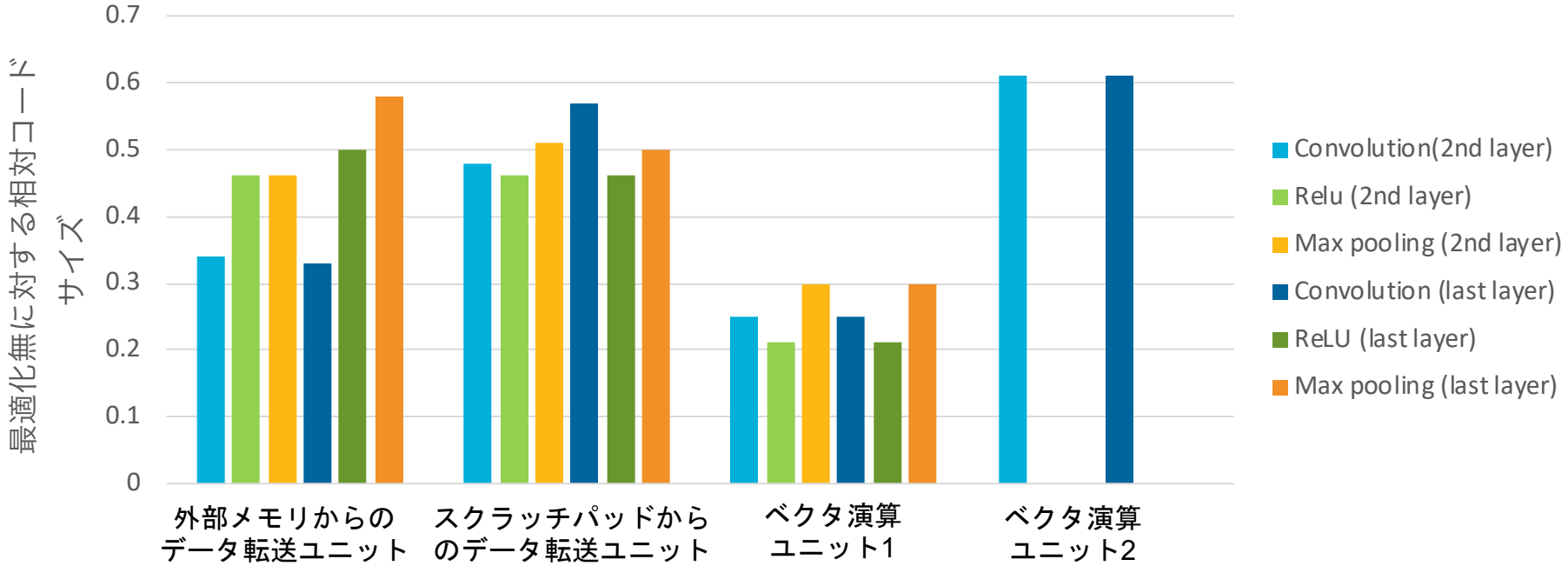
コンパイラ内部のデザイン



最適化器の特徴

- 最適化の例
  - ループの最適化
    - 複数段のループネストのマージ
    - 不要なループ（例えば実行回数１回のループ）の簡略化
  - 不要な命令の削除
    - dead code (結果がその後使われない命令)の削除
    - コンパイル時に分岐方向が確定できる条件分岐の削除
    - 演算のマージ (r1 += 1, r1 += 2 → r1 += 3)
- 普通のコンパイラの最適化と基本的に同じ
  - 命令バッファが小さいため速度向上よりもコードサイズ削減が重要（命令バッファに入らないと，そもそも実行できない）
  - 命令セットが限られているためできることが限られている

最適化によるコードサイズの削減の例



実験方法
コンパイラで最適化ありと最適化なしのコードを、vgg16の２層目と最終層のConvolution, Relu, Max poolingについてコンパイルし、ユニットごとにコードサイズを比較

- Ref: 石崎他，IPSJ プログラミング研究会 2020

シミュレータ

- ハードウェアが完成する前にソフトウェアの開発を行うために使用
- 必要とされる要件
  - シミュレーション速度
    - 現実的なサイズのニューラルネットワークが，正しくコンパイル・実行されるか検証するには速度は重要．遅いとデバッグやCIが辛い．
  - シミュレーション精度
    - ハードウェアでプログラムを実行した時の実行時間の精度の高い予測があるとアルゴリズムや最適化器の開発に有用
  - デバッグ機能
    - 結果が合わないような場合に，各ユニットでどのような演算，メモリアクセスがなされたかを見れるようにすることで，ソフトウェア開発の効率化を図る
- すべての要件を一度に満たすのは困難なため，複数のシミュレータが作られる場合もある

まとめ

- Domain-specific architectureを活用するためのソフトウェアの例として，RaPiD AI アクセラレータ向けのソフトウェアスタックの開発についての事例を紹介
  - 自分が主に関わっているコンパイラとシミュレータについて説明

- 多くの場合，新しいDSAには対応するソフトウェアの開発が必要
  - とは言え，共通部分はできるだけ再利用できるようにいろいろなソフトウェアフレームワークが開発されてきている（例えばMLIR, TVM, LLVMなど）
  - フレームワークをうまく使えれば新しいDSAに特有な部分にソフトウェアの開発資源を集中できる

参考文献

- ハードウェア
  - B. Fleischer *et al.*, “A Scalable Multi-TeraOPS Deep Learning Processor Core for AI Training and Inference” VLSI 2018
  - Jinwook Oh, *et al.*, “A 3.0 TFLOPS 0.62V Scalable Processor Core for High Compute Utilization AI Training and Inference” VLSI 2020
- ソフトウェアスタック
  - S. Venkataramani *et al.,* “DeepTools: Compiler and Execution Runtime Extensions for RaPiD AI Accelerator” IEEE Micro 2019
  - E. Ogawa *et al.,* “A Compiler for Deep Neural Network Accelerators to Generate Optimized Code for a Wide Range of Data Parameters from a Hand-crafted Computation Kernel” IEEE COOLChips 2019
  - 石崎他, “ソフトウェアによるメモリ階層管理を行うディープニューラルネットワークアクセラレータ用のコンパイラの概要”, IPSJ PRO 2020

閑話休題：社会人博士の勧め

- 博士に直接進学するのと比べたメリット
  - 入学前に業績になる論文があると余裕を持てる
    - 博士論文に含めることができるかは専攻などによるので要確認
    - 自分は博士論文を構成する3本の論文中の1本が入学前の論文
  - しばらく会社で働くと，大学（研究・授業・ゼミなど）が新鮮
  - 会社によっては補助（時間的・金銭的）があるかも
- 博士に直接進学するのと比べたデメリット
  - 仕事しつつなので在学中は時間的に辛い
    - 休職して博士取得に専念する人もいる
  - 時期的に数年は遅くなる
    - 修士での指導教員の先生が異動や退官をしてしまう可能性がある
    - 年齢的にライフイベントなどとぶつかる可能性がある

Agenda

- Introduction: A New Golden Age?
- Topic 1: Software stack for a deep learning accelerator
- Topic 2: Algorithms for SIMD (vector) instructions

Q: 新しいハードウェア向けの大規模なソフトウェアスタックの開発をする場合以外は，ハードウェアの変化はソフトウェアに関係しない？
（論文を書くようなネタにはならない？）

➔A: どんなソフトウェアにも影響する可能性がある

ハードウェアの変化はソフトウェアにとって論文ネタの宝庫（私見ですが）

- 新しいハードウェアが出てくると，今までの思い込みがひっくり返ることがよくある．それが論文のネタになる
- 自分の過去の論文の例：
  - シングルスレッドだと速いregion-baseのメモリ管理が，マルチスレッドだとかえって遅い（コア数の増加で，計算律速→メモリバンド幅律速に変化）PLDI 2009
  - コア数が多いマシンでかつCPU利用率が低い場合（要するに大半のサーバ）に置いては，SMT (HyperThreading) があるとかえってサーバのレスポンスタイムが悪化する． IISWC 2014
  - メモリアクセスが単純なマージソートはSIMD命令をうまく使うとクイックソートよりも速くなる． PACT 2007, VLDB 2015

Goal

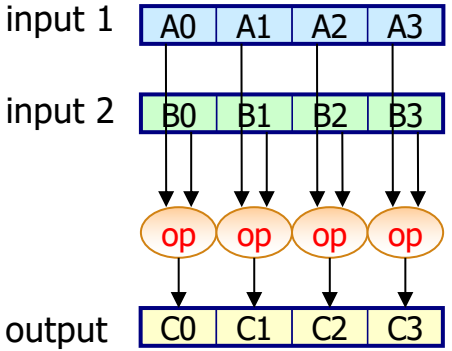
- Develop a fast sorting algorithm by exploiting the following features of today's processors
  - SIMD instructions (data parallelism)
  - Multiple cores (thread-level parallelism)

- We think about sorting of an integer array here. Refer to our VLDB 2015 paper for extending this algorithm for sorting larger records.

Benefit and limitation of SIMD instructions for sorting

- Benefits: SIMD selection instructions (e.g. select minimum instruction) can accelerate sorting by
  - parallelizing comparisons
  - avoiding unpredictable conditional branches
- Limitation SIMD load / store instructions are most effective when they access a properly-aligned contiguous memory block

4-wide SIMD instruction (e.g. 32-bit integers on 128-bit vector register)



Existing sorting algorithms suitable for SIMD instructions

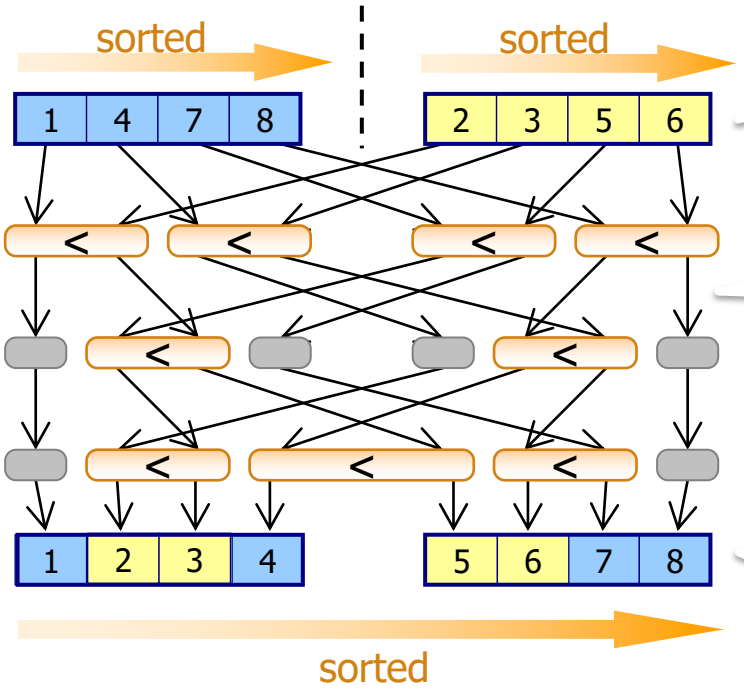
- Sorting networks, such as bitonic merge sort, odd-even merge sort [Batcher 1968], and their variants such as GPUTeraSort [Govindaraju 2005], are suitable for SIMD
  - They are easy to implement with SIMD without gather or scatter memory accesses since they compare data pairs in a sequential manner
  - However, they are slower than widely-used algorithms such as mergesort or quick sort for large datasets; their computational complexity is $O(N (\log (N))^2)$

Our approach: hybrid merge operation of odd-even merge and usual merge

- SIMD instructions are effective for sorting networks (e.g. odd-even merge and bitonic merge)
- However, their computational complexity are higher than usual (branch-based) merge operations

- Our solution
  - Integrate sorting network into the usual merge operation to take advantage of SIMD instructions while keeping the computational complexity of usual merge operation

Odd-even merge for values in two vector registers



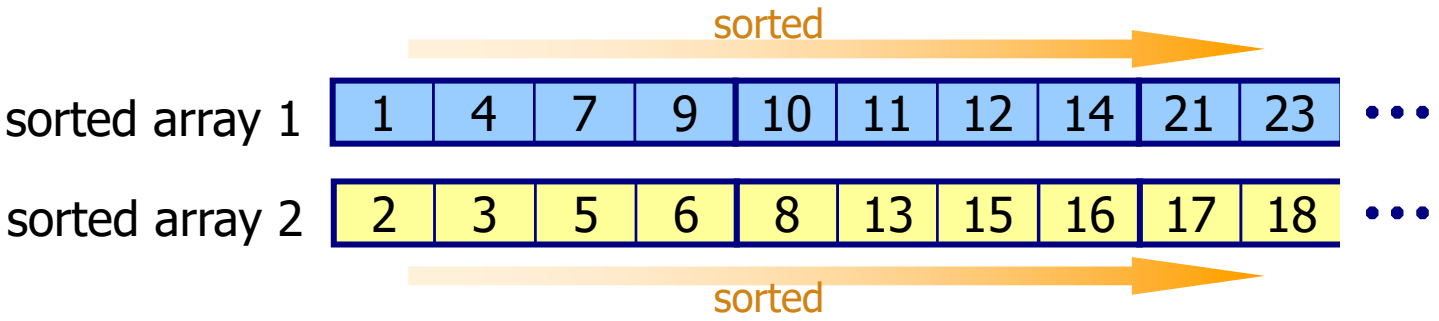
assuming four values in a vector
(e.g. 32-bit integer in 128-bit vector)

Input
two vector registers contain four presorted values in each

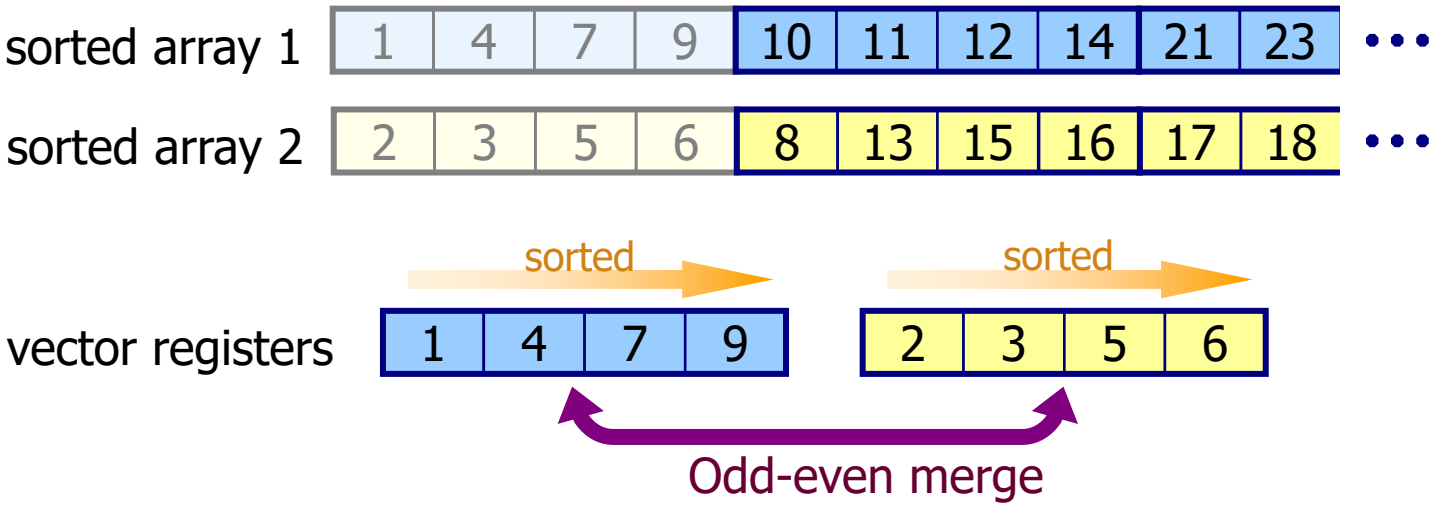
SIMD merging
one SIMD comparison and "shuffle" operations for each stage *without conditional branch*

Output
eight values in two vector registered are now sorted

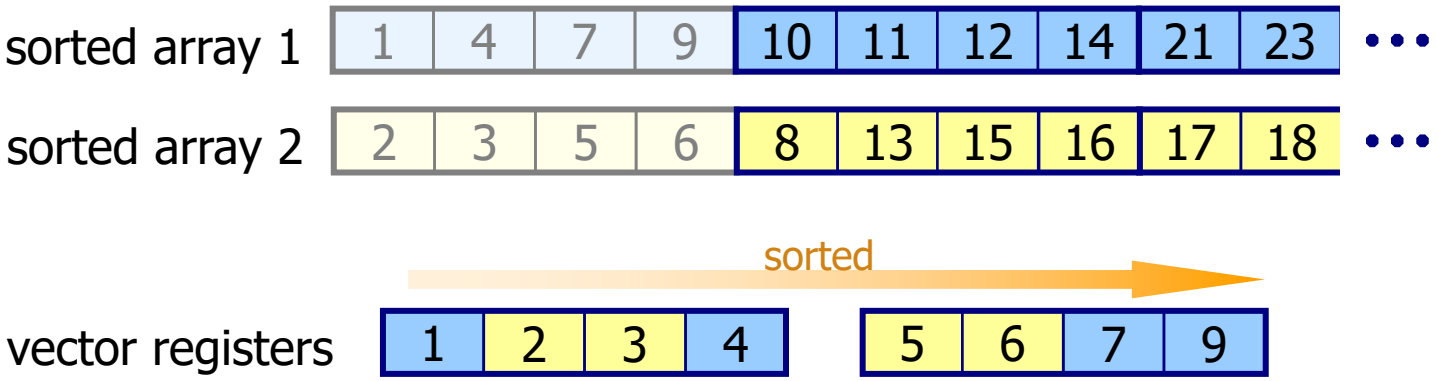
Our technique to integrate odd-even merge into usual merge



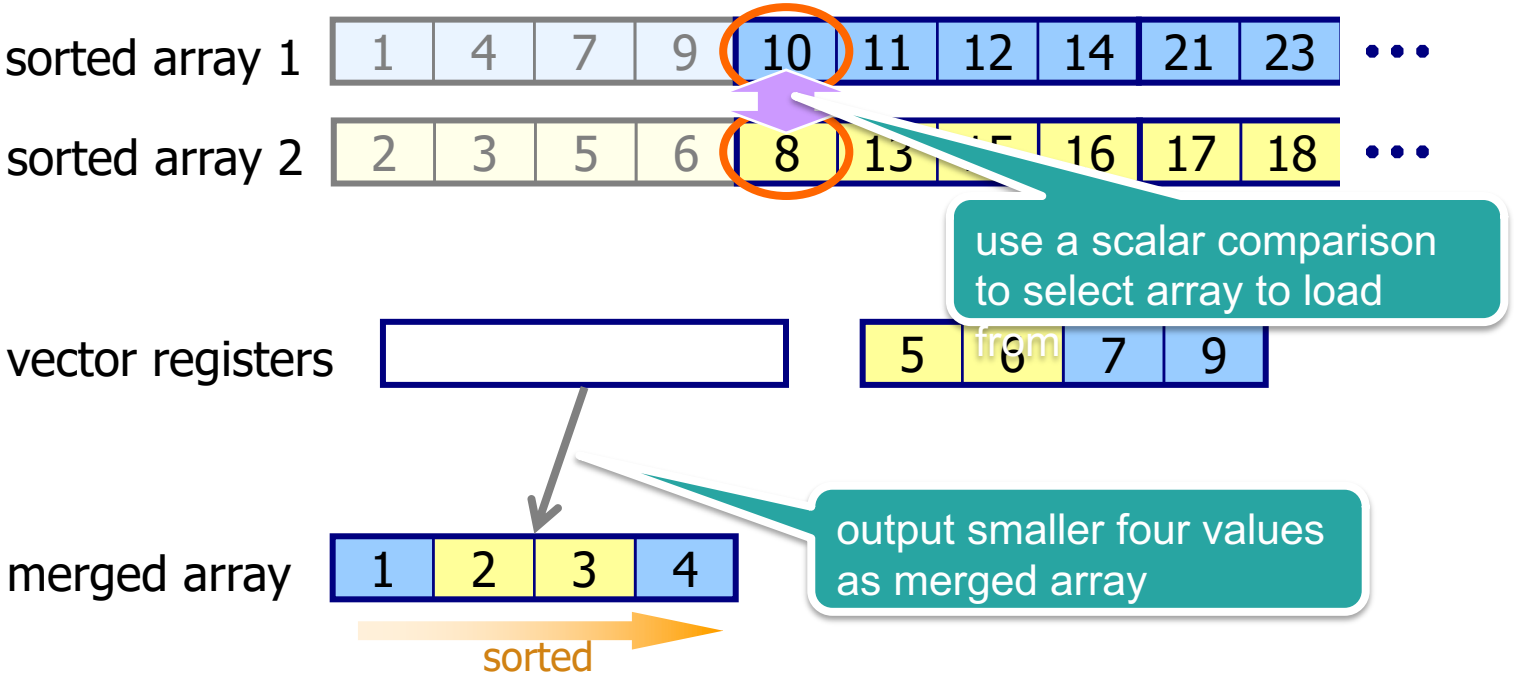
Our technique to integrate odd-even merge into usual merge



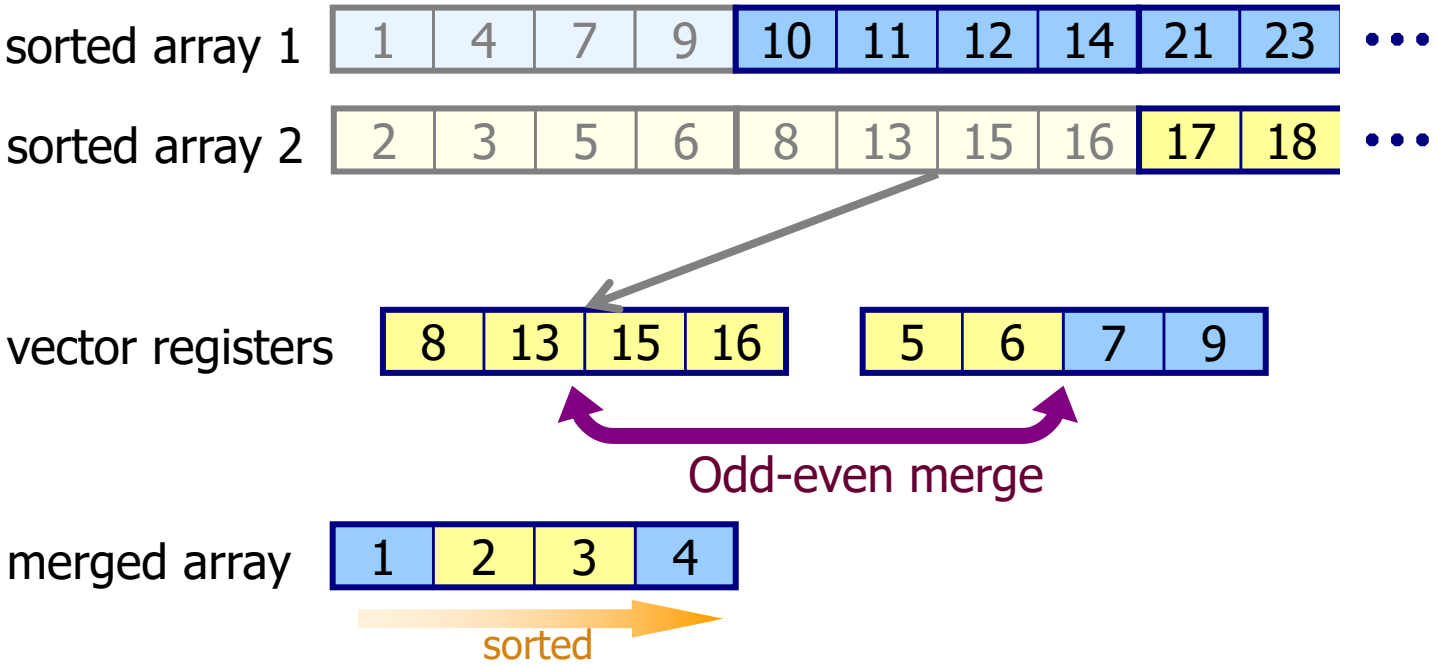
Our technique to integrate odd-even merge into usual merge



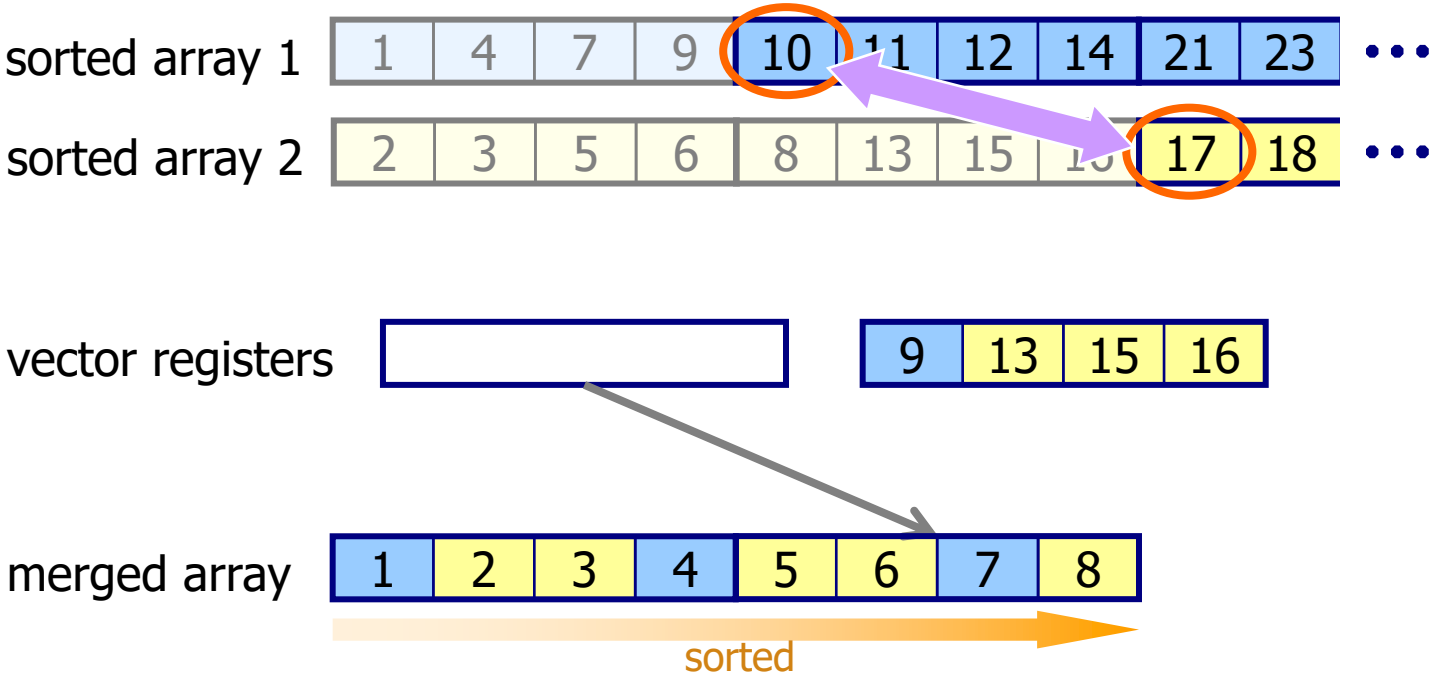
Our technique to integrate odd-even merge into usual merge



Our technique to integrate odd-even merge into usual merge



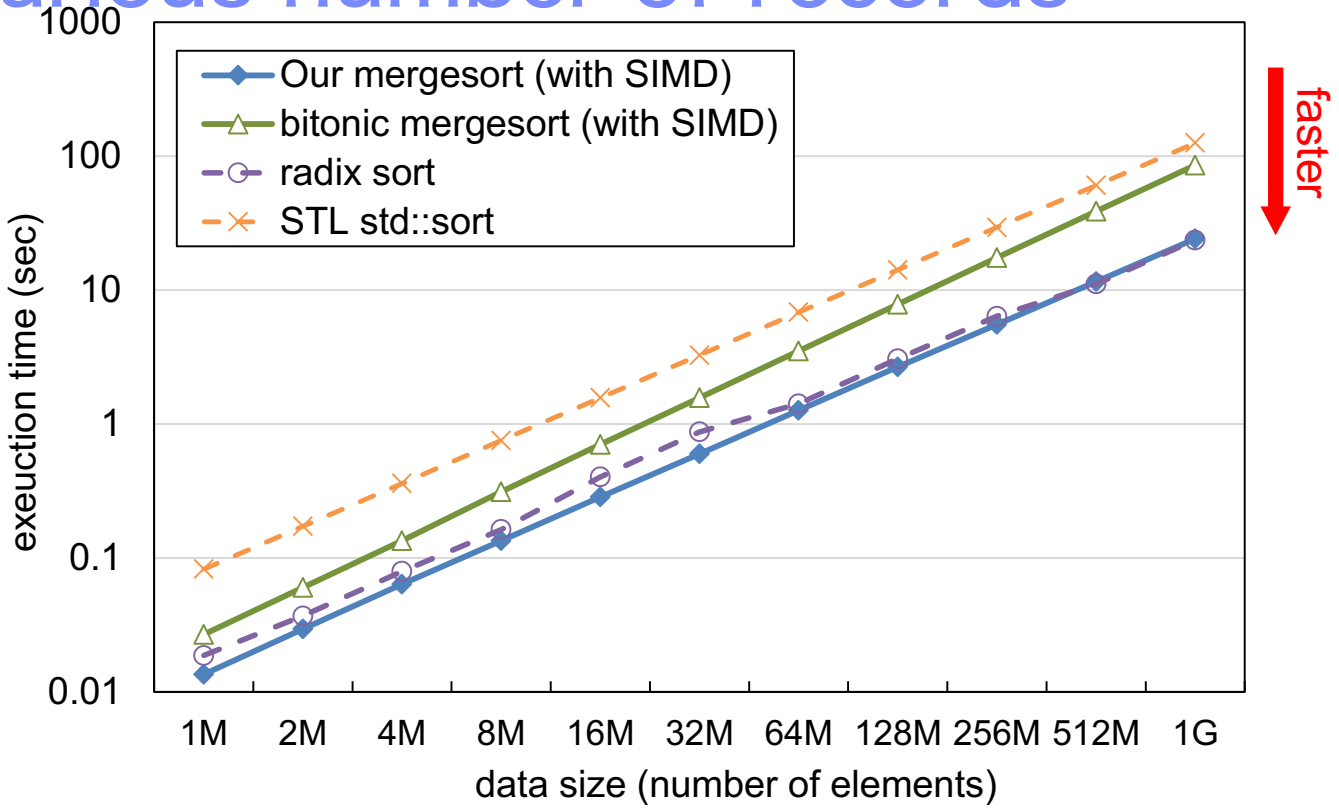
Our technique to integrate odd-even merge into usual merge



Comparing merge operations

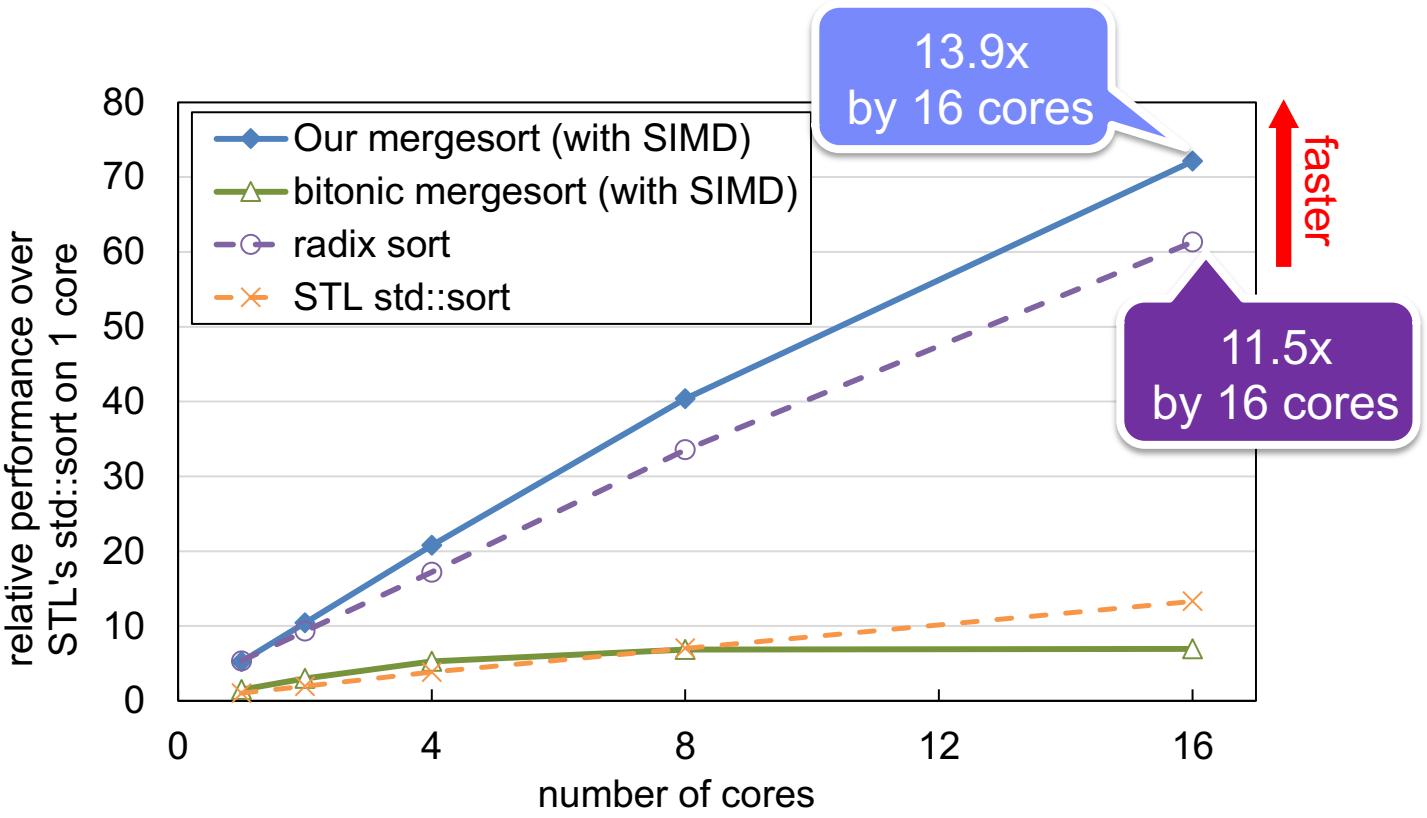
	our integrated merge operation	odd-even merge implemented with SIMD	usual (scalar) merge operation
Number of hard-to-predict conditional branches	1 for every output *vector*	0	1 for every output *element*
Computational complexity	$O(N)$	$O(N \log(N))$	$O(N)$

Single-thread performance with various number of records



for sorting random 32-bit integers for up to 1 billion records

Scalability with multiple cores



for sorting 1 billion random 32-bit integers

Related work: papers that implemented or enhanced my SIMD mergesort

- Chhugani et al. (VLDB 2008)
  - first implementation of the algorithm on Intel architecture (using bitonic merge as SIMD merge kernel)
  - showed the SIMD merge larger than vector register size increased the instruction-level parallelism
- Satish et al. (SIGMOD 2010)
  - first implementation on NVIDIA GPUs
- Kim et al. (VLDB 2009), Balkesen et al. (VLDB 2013)
  - adaptation to database management systems (sort-merge join)
- Kim et al. (SIGMOD 2012), Sundar et al. (ICS 2013)
  - adaptation for in-memory kernel of distributed sorting systems

まとめ

- 新しいハードウェアを効率的に使いこなすためのアルゴリズム開発の例として，SIMD命令向けのソートアルゴリズムの例を紹介

- 新しいハードウェアを活用する機会は，システムソフトウェア（例えばコンパイラやOSに限らず）どんなレイヤーのソフトウェアにもある

参考文献

- ソーティングのSIMD化
  - Hiroshi Inoue et al., "AA-Sort: A New Parallel Sorting Algorithm for Multi-Core SIMD Processors", PACT 2007
  - Hiroshi Inoue et al., "SIMD- and Cache-Friendly Algorithm for Sorting an Array of Structures", VLDB 2015
- JoinのSIMD化
  - Hiroshi Inoue et al., "Faster Set Intersection with SIMD instructions by Reducing Branch Mispredictions", VLDB 2015

全体まとめ

- コンピューティングシステムの性能向上の源泉は，半導体技術の向上から，ワークロードに特化したハードウェアの活用に移ってきている
- 新しいハードウェアを活用するためにはソフトウェアもまた変化が求められ，（いろいろ大変ながらも）ソフトウェア技術者の活躍の場も広がっている

最後に

- ハードウェアもソフトウェアもドメイン特化なものにするには，コンピューティングシステムの技術者が，その上のワークロードも（ある程度は）理解していることが大事
  - 間違った条件に特化したシステムを作ると悲劇 😱
  - その分野の専門家と協業するにしても，言葉が通じないと厳しい
- 大学の授業とか電子情報学輪講で聞いた，ちょっと違う分野の話が意外と後で役に立つ（かもしれない）ので，専門外の話と毛嫌いせずに良く聞いてみると良い