IBM Research - Tokyo

電子情報学特論：
ハードウェアの高速化を支える
システムソフトウェア

Hiroshi Inoue (井上 拓)
IBM Research – Tokyo

プロフィール

- Research Staff Member
  @ IBM Research – Tokyo (IBM 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，混相乱流のエネルギー伝達）

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

研究テーマ

- ハードウェアとソフトウェアのインターフェースのソフトウェア側

➔ハードウェアを意識して直接アクセスするソフトウェアの研究

- ユーザがハードウェアを意識しなくても良いようにする，縁の下の力持ち的存在
- 自分はハードを意識することで実行速度の向上を目指すのが好きですが，他の目的もいろいろあります（例えばセキュリティの向上）

Agenda

- **Introduction: A New Golden Age?**
- Overview of software stack for a deep learning accelerator
- Code generator for accelerator
- Simulator for hardware-software co-design

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システムソフトウェアが必要になる (hardware-software co-design)

ハードウェアの変化はソフトウェアの研究にとって論文ネタの宝庫（私見ですが）

- アクセラレータに限らず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

It's Golden Age for System Software too!

- Chris Lattner (main developer of LLVM, Swift, MLIR etc) The Golden Age of Compiler Design in an Era of HW/SW Co-design, ASPLOS 2021 keynote talk
  - DSAの活用のためには，コンパイラとプログラミング言語(e.g. domain-specific languages, DSL)が必要
  - コンピュータアーキテクチャの黄金時代は，システムソフトウェアにとっても新たな黄金時代！

高速化のためのハードウェアの例

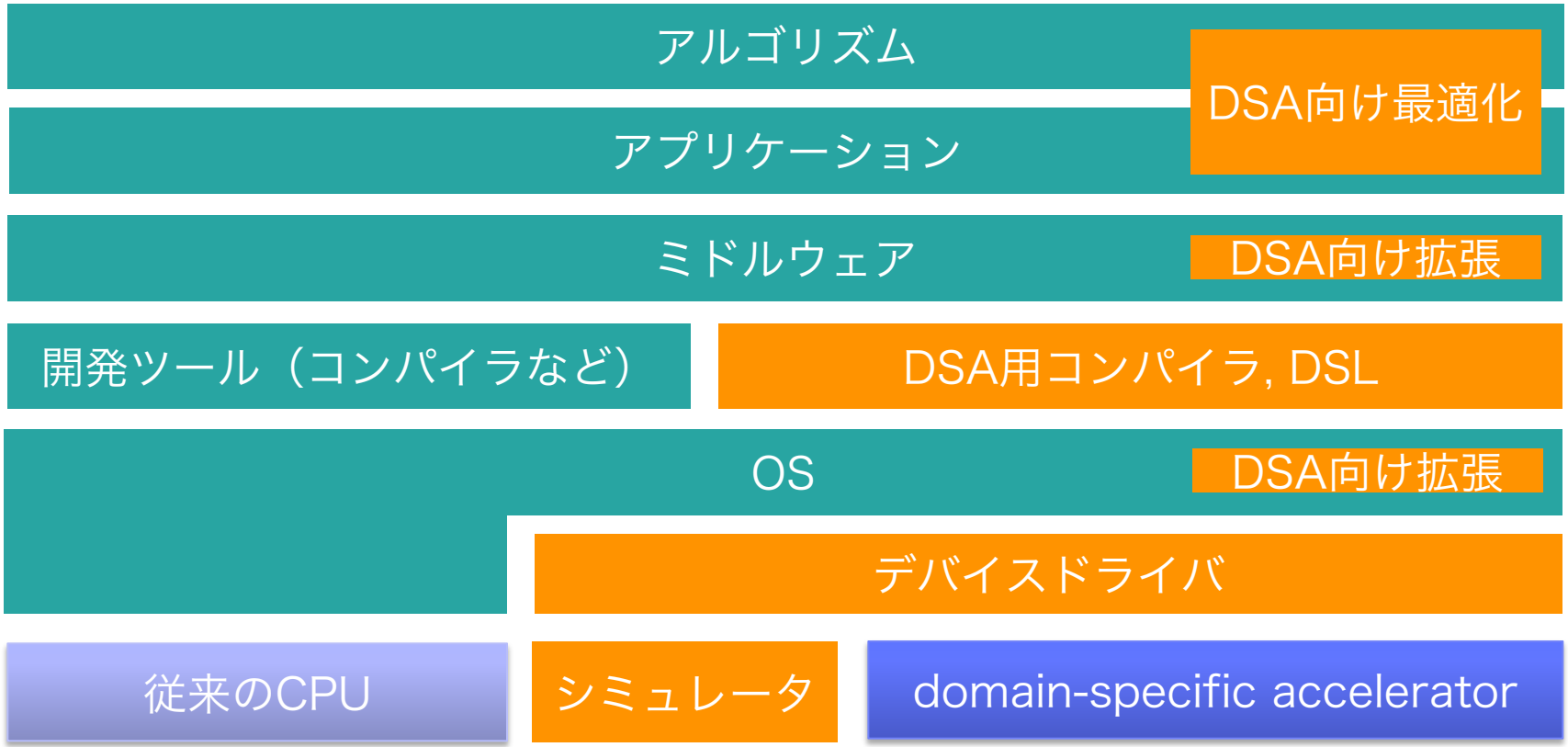
- 高い並列度の汎用プロセッサ
  - スレッド並列度: マルチコア, SMT (例えばHyperThreading)
  - データ並列度: SIMD命令 (例えばIntelのAVX, SSEなど), 行列計算命令 (Intel AMX)
- GPU
- FPGA
- ASIC (domain-specific acceleratorというとたいていここを指す)
  - Deep Learningアクセラレータ (GoogleのTPU, PFNのMN-Coreなど)
  - Bitcoin miningアクセラレータ・暗号アクセラレータ

- 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)
  - 量子 annealer (D-Waveなど)
- Spiking neural networkチップ, Analog neural networkチップ

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?
- Overview of software stack for a deep learning accelerator
- Code generator for accelerator
- Simulator for hardware-software co-design

Deep Learning アクセラレータ

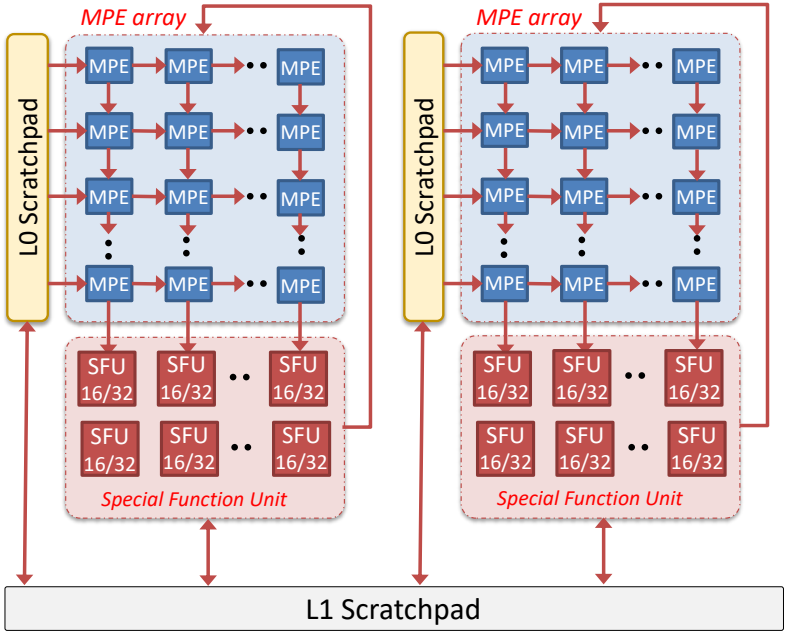
- ニューラルネットワークの処理は学習でも推論でも，多くの計算時間を畳み込み(convolution)や行列積計算に費やす

- 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，簡素な低精度(fp16, fp8)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

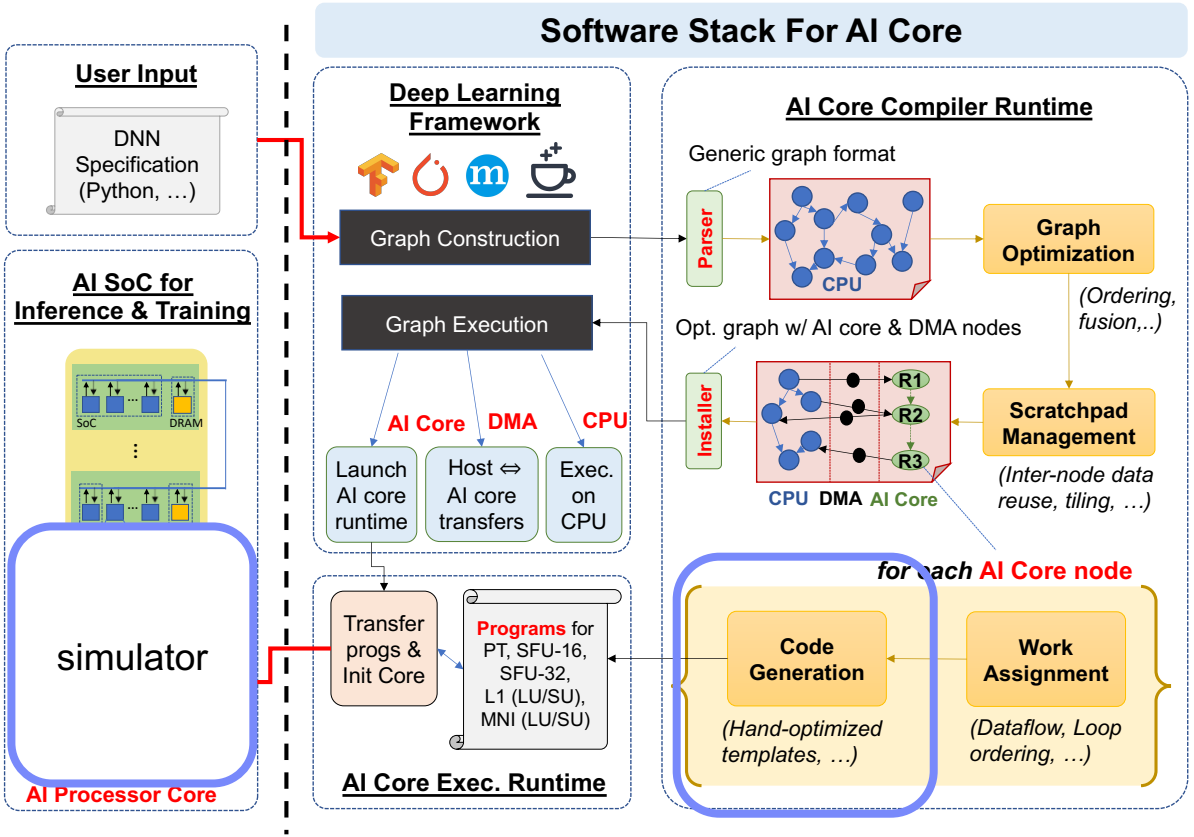


- ハードウェアの特徴
  - 畳み込みと行列積用のシストリックアレイとその他の処理用のvector演算器を持つ
  - データ移動をソフトウェアで行う階層的なスクラッチパッドメモリを使用
  - fp16 (DLFloat16), fp8, Int4, Int2などの低精度の演算をサポート
- ソフトウェアから見た特徴
  - 演算もデータ移動も全ユニットがプログラマブル．一つのコアあたり最大44個!!
  - 命令セットもユニットの種類ごとに違う

➔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: A. Agrawal et al. ISSCC2021, S. Venkataramani et al. ISCA 2021

DeepTools: RaPiDのためのソフトウェアスタック



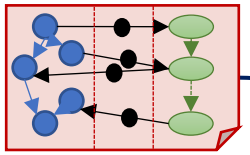
- **Compiler runtime:** TensorFlowやpytorchの計算グラフの各ノードをRaPiD用にコンパイルする
- **Exec. runtime:** コンパイルされたコードとデータをRaPiDに転送し実行する
- Ref: S. Venkataramani *et al.* IEEE Micro 2019, E . Ogawa et al., COOLChips 2019

Agenda

- Introduction: A New Golden Age?
- Overview of software stack for a deep learning accelerator
- Code generator for accelerator
- Simulator for hardware-software co-design

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

シストリックアレイ用
ベクタ演算ユニット用
データ転送ユニット用
コード

DeepTools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

パラメータ

テンプレート

コンパイラ

中間表現生成 (frontend)
パラメータとテンプレートから中間表現の形式に変換する

最適化器 (optimizer)
IR形式のプログラムを実行性能向上やコードサイズ削減のために変形する

コード生成 (backend)
IR形式のプログラムから，各ユニットの命令列を生成する

IR表現 (独自形式)

```
program l1lu:0:0 {
 sync recv l1su
 loop imm=64 {
  loop imm=224 {
   loop imm=224 {
    asm {
     テンプレートの命令列
    }
   }
  }
 }
 return
}
program pt:0:0:0 {...
```

各ユニット用実行コード

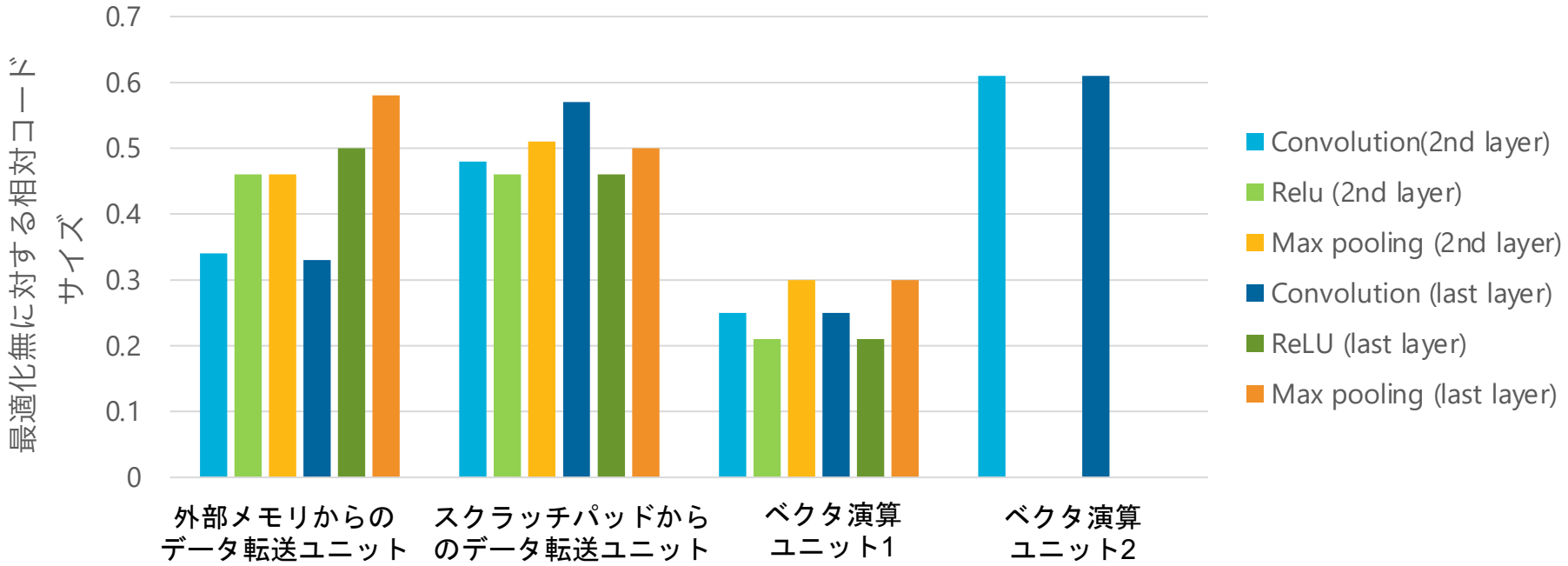
ベクタ演算ユニット用

データ転送ユニット用コード

最適化器の特徴

- 最適化の例 (IRでの最適化および命令列での最適化)
  - 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

(コンパイラについてのまとめとして) Challenges and lessons learned

- コード最適化
  - RaPiD特有の最適化だけでなく，一般的な最適化も自分たちで実装する必要があった
  - ➔既存のコンパイラフレームワーク (e.g. MLIR, LLVM) をうまく活用して，特有の部分だけにフォーカスできると良い
- 手書きテンプレート
  - DLフレームワークは，多種のオペレータを提供しており増え続けていて，それぞれへの対応が必要
  - ➔高級言語(e.g. DSL)の導入などで開発を容易にする必要がある
  - ➔（夢としては）既存の実装からの自動変換が望ましい

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?
- Overview of software stack for a deep learning accelerator
- Code generator for accelerator
- **Simulator for hardware-software co-design**

アクセラレータのためのシミュレータ

- 今回はハードウェアが完成する前にソフトウェアの開発を開始するために使用することを目的としたシミュレータ
  - ハードウェアの開発には時間がかかるため，ハードが完成してからソフトウェアの開発を行うのでは時間がかかりすぎる
  - ハードウェアと同時にソフトウェアの開発を行うことで，ハードウェアの設計へのフィードバックを返すことができる (hardware-software co-design)
- 別の目的のシミュレータも作成されることがある

シミュレータに求められる要件

- シミュレーション精度
  - プログラムの実行結果がハードウェアと一致していて欲しい
  - 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
- 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．
- デバッグ機能
  - プログラムのデバッグのために，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

DeepToolsのシミュレータの特徴

- シミュレーション速度とデバッグ機能にフォーカス
  - シミュレーション精度は妥協
    - 正確な実行時間は予測しない
    - 実際のデータ型によらずfp32(float)で計算
- 2つの実行モードをサポート
  - end-to-end mode (TensorFlow/pytorchから直接呼び出す)
    - ニューラルネットワーク全体のテストのために使用
    - TensorFlow/pytorch実行中に動的にコード生成を行い，メモリ内のデータを使用して実行する
  - standalone mode
    - 各レイヤーをテストするために使用
    - 事前に生成したプログラムとメモリイメージをファイルから読む

シミュレータの実装

- 各ユニット (ロード・ストアユニット，シストリックアレイ演算器，ベクタ演算器 etc)の処理は単純なインタプリタとして実装
  - JITコンパイルを行うことで，実行性能を大きく上がることが期待できるが命令セットが変更・拡張されるたびにインタプリタとJITでサポートしつづけるのは，開発コスト的に困難
- 各ユニットは互いにFIFO経由でデータや同期メッセージの通信を行う
  - 巨大なデータを各ユニット間で受け渡す必要があるために，全体のシミュレーション速度の向上のためにFIFOの性能が重要
  - FIFO実装は，ストリーミング処理に最適化された実装．*FastForward* (Giacomoni et al., PPoPP 2008)
  - 可能な限りデータのコピーを削減

シミュレータでのデータ型の扱い

- シミュレータの内部では，データ型によらずすべてのデータをfp32 (float)で保持し演算を行う．データ型の情報は別途保持している
  - 低精度演算をソフトウェアで実装するとコストが高いため，ホストCPUのfp32浮動小数点演算のハードウェアを使用する
  - デバイスメモリやスクラッチパッドに書き込む際に，データ型に合わせて精度を落とす
- デザインのトレードオフ
  - 精度と実行性能のトレードオフ
    - 低精度の演算を正確に再現するよりも実行性能を優先
  - 実行性能について，演算コストとメモリコストのトレードオフ
    - 内部で実際よりも高い精度でデータを保持していることで，メモリ消費やデータコピー，精度変換のコストが高くなる代わりに，ホストの演算命令がそのまま使えるので演算コストは低い

TensorFlow/pytorchとの接続

- シミュレータはライブラリとして実装されている
  - standaloneモード用のコマンドラインツールは，コマンドラインを受けとってライブラリをコールする小さなプログラム
  - end-to-endモード用に，TensorFlowやpytorchには，アクセラレータの呼び出しをするブリッジ機能を追加
    - このブリッジが，シミュレータの呼び出しを行う（ハードウェアで実行する場合はデバイスへのアクセスを行う）
    - ブリッジは仮想的なデバイスメモリを提供し，ホストメモリ⇔仮想デバイスメモリ間の転送を行う．シミュレータはこの仮想デバイスメモリにアクセス
    - ユーザプログラムからは，シミュレータとハードウェアは同等に見える

DLフレームワークと連携したデバッグ情報

- コンパイラがメモリ内でのデータ配置などのメタデータをTensorFlow/pytorchから取得し，シミュレータ用のデバッグ情報としてプログラムに含める
  - シミュレータはメモリアクセストレースとして，アドレス情報に加えて，テンソルのindexなどを表示することでデータの理解を助ける

```
SYNC send to lxlu:0:3, pc=28, id=(l3lu:3)1@28
SYNC recv from lxlu:0:3, pc=34, id=(lxlu:0:3)2@2
LX[04452(0x1164) KER(ds1):00000/00448] <- HBM[0595126(0x0914b6) KER(ds1):00000/00224[0/1,0/32,0/7,0/1]], pc=036, b=00, FP8_143 val=3
LX[04453(0x1165) KER(ds1):00001/00448] <- HBM[0595127(0x0914b7) KER(ds1):00001/00224[0/1,0/32,1/7,0/1]], pc=036, b=01, FP8_143 val=2.5
LX[04454(0x1166) KER(ds1):00002/00448] <- HBM[0595128(0x0914b8) KER(ds1):00002/00224[0/1,0/32,2/7,0/1]], pc=036, b=02, FP8_143 val=2.25
```

```
SYNC recv from lxlu:0:3, pc=34, id=(lxlu:0:3)2@2
LX[04452-04675 KER(ds1):00000-00223/00448] <- HBM[0595126-0595349 KER(ds1):00000-00223/00224 [:/1,:/32,:/7,:/1]], size=224
```

- ハードウェアが利用可能になったあとでも，プログラムのデバッグ用にはシミュレータは有用

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
  - S Venkataramani *et al,*. “RaPiD: AI Accelerator for Ultra-Low Precision Training and Inference”, ISCA 2021
  - ISSCC 2021, A. Agrawal et al, “A 7nm 4-Core AI Chip with 25.6 TFLOPS Hybrid FP8 Training, 102.4 TOPS INT4 Inference and Workload-Aware Throttling”
- 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

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