Vol.2022-HPC-187 No.9
2022/12/1

Ducks: DataFrame with Compiler

石坂 一久[1]　大野 善之[1]　Sourav Saha[1]　大道 修[1]　小寺 雅司[1]　荒木 拓也[1]

概要：データフレームコンパイラを搭載し利便性と高速性の両方を実現するデータフレーム用 Python ライブラリ Ducks を紹介する．データフレームは，github で 35K star を誇る Pandas に代表されるように，データ分析や前処理に広く用いられているが，データ量の増大や分析の複雑化により高速化が求められている．Ducks は HPC で培われてきた実行時コンパイル技術を用いることで，ライブラリ呼び出しを直接実行するのではなく，データフーム用の中間言語（IR）を生成して遅延実行を行う方式を採用している．これにより API とその実行を分離し，Pandas 互換の API を提供しながら，IR 上でのドメイン特化の最適化，ターゲットプラットフォームに最適化されたバックエンドによる IR 実行により高速性も実現する．ベンチマーク集である TPCx-BB，TPC-H に含まれる 45 種類のデータ前処理・分析処理を用いた CPU 上での評価では，ライブラリを選択する import 文の変更のみで，Pandas に対して最大 17 倍，平均 5.8 倍の性能向上を得ることができ，本方式の有効性を確認した．

キーワード：実行時コンパイル，データフレーム，ドメイン特化最適化

1. はじめに

HPDA（High Performance Data Analytics）市場は，HPC 市場全体の 3 倍の成長率を示しており [1]，データ分析は HPC でも関心が高い領域であるが，その中でもデータフレームの重要性が高まっている．データフレームは，データ分析の中でも探索的データ解析と呼ばれるデータを理解するフェーズや，機械学習のための前処理で良く利用される．このようなフェーズは，“ワイルドな” データ分析とも呼ばれるように，フォーマットの定まってないデータや欠損値のあるデータを扱ったり，異なるドメインのデータも組み合わせながら，データに対する様々な操作を行い，データの特徴を抽出したり，より学習に適したデータを作り出すフェーズで，データサイエンティストが試行錯誤を繰り返しながら行う．DX が進む中このようなデータ分析はますます重要となっており，データサインティスの業務量の半分近くを占めているという報告もある [2]．

従来売上データの集計などの定型的なデータ分析には，リレーショナルデータベースや BI ツールなどが利用されてきが，このようなワイルドなデータ分析では，1970 年代に登場した SQL で処理を記述するデータベースシステムよりも，デバッグやモジュラリティなどに優れた開発効率の高いプログラミング言語が好まれ，Python で提供されているデータフレームが用いられている．データフレームは，データベースと同様に表形式のデータを表すデータ構造とそれを処理するための一連の操作からなる．スキーマを明確に定義する SQL とは違い，列の追加や削除が柔軟にできることや欠損値を柔軟に扱えることなど，このようなデータ分析に向いた特徴を持っている．また，多くの機械学習アルゴリズムは Python で開発されており，機械学習との連携もやりやすい．

データフレームとして標準的に利用されている Python ライブラリが Pandas である．Pandas は 2500 人以上のコントリビュータによって OSS として開発されており，2020 年時点で 3 億回以上のダウンロード数 [3] を記録するなど多くのデータ分析で利用されている．一方で計算機性能に目を向けると，メインメモリ量の増大や高速な SSD によりシングルノードで扱えるデータ量は格段に増大している．Pandas はシングルノードでのインメモリでの処理をターゲットとしているが，このような進歩により従来はストレージシステムを扱えるデータベースや DWH が必要とされてきたようなケースも，シングルノード上で手軽に行えるようになってきており，Pandas の利用シーンを増やしている．

このような背景の中でデータ分析の現場での普及が進む Pandas であるが，一方で多くの操作がシングルスレッドで動作するなど，処理速度の面では課題がある．例えば，数 GB 程度のデータであっても一回の操作に数秒～数分かかりインタラクティブ性を悪化させたり，より大きなデータ

[1] NEC デジタルテクノロジー開発研究所

や複雑な処理では数時間〜数日に及ぶこともある．Pandas の処理速度の課題としては実行モデルもある．データベースシステムでは，SQL で定義された複数の操作を Query Optimizer によって最適な操作や順序へ変換してから実行するこで処理時間の短縮をしている．一方でライブラリである Pandas では，ユーザーコードからのライブラリ呼び出しがその都度実行される eager 実行モデルであるため，このようなデータフレーム固有の最適化が行われない．また，GPU などのアクセラレータをサポートしていないことも処理速度の課題となっている．

Ducks はこのような Pandas の抱える速度面の課題を，その利便性を損なわずに解決することを目標とし，実行時コンパイル技術を導入したライブラリである．本稿では，まず 2 節でデータフレームと Pandas についての簡単なバックグラウンドを与え，3 節および 4 節で Ducks の設計と実装の特徴を説明する．その後，5 節では性能評価を行い Ducks の有効性を示し，6 節で関連技術を紹介する．

2. データフレーム

本節では Pandas を例にデータフレームについて説明する．図 1 に一日毎の売上数量（qty）と売上金額（amt）の合計を求めるプログラム例を示す．2 行目の `read_csv` は，Pandas が提供する API で，CSV ファイルを読み込むメソッドである．読み込まれた結果は，テーブルデータを表す `Pandas.DataFrame` クラスのオブジェクトとして返される．3 行目の `groupby` は，`DataFrame` クラスのメソッドでテーブルのグループ化を行う処理である（ここでは `date` 列をキーとしてグループ化を行う）．それに続く `sum` メソッドはグループ化されたテーブルに対して集約演算（ここでは和）を行う．4 行目では，その結果から `qty` と `amt` の二列を取り出し，`to_csv` メソッドでファイルに保存している．

```
1 import Pandas as pd
2 df = pd.read_csv("sales.csv")
3 daily = df.groupby("date").sum()
4 daily[["qty", "amt"]].to_csv("daily.csv")
```

図 1 データフレームプログラム（Pandas）の例

このように Pandas はトップレベルのメソッドや `DataFrame` クラスのメソッドとして，データの読み書きを含めた各種のデータフレーム操作を提供している．例に出てきたもの以外にも代表的な操作としては以下のようなものがある．

- テーブルの結合（`join/merge`）
- テーブルのソート（`sort_values`）
- テーブルの特定の行の取り出し（`filter`）
- 欠損値の除外（`dropna`）
- 各種の行列・ベクトル演算（可算・乗算など）

3. Ducks の設計

Ducks の全体像を図 2 に示す．図の中心にある「Ducks IR」は，Ducks が実行時にユーザープログラムから生成する中間言語（IR: Intermediate Representation）であり，その周囲に配されている箱が Ducks の構成要素を示す．白地の箱であるフロントエンド，最適化パス，バックエンドが Ducks 固有のモジュールであり，灰色地の箱は汎用性のあるモジュールである．このように Ducks では各モジュールが IR をインタフェースとして独立するように設計されている．本節では Ducks 固有モジュールを中心にその役割と狙いを説明する．

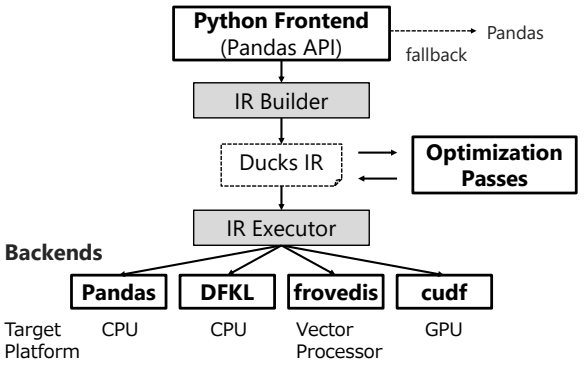

図 2 Ducks の全体像

3.1 Ducks IR

Ducks の中心となるのは IR である．本 IR は，従来のコンパイラで使われるようなプロセッサの命令に近い IR でなく，データフレームの操作に対応するドメイン特化型 IR である．Ducks IR では，整数型などの基本的なデータ型に加えて，Pandas の `DataFrame` クラスに相当するテーブル型を定義しており，それに対する各種操作を命令（Op: Operation）として定義している．図 1 に対応する処理を IR で記述した例を図 3 に示す．図中の`%`で始まる変数は，この例ではすべてテーブル型である．各行の括弧の前の “`Ducks.`” から始まる文字列がテーブル型を引数や返り値とする Op であり，この例では csv ファイルの読み書きを行う `read_csv`, `write_csv`，グループ化と集約を行う `groupby_agg`，表から列を取り出す `projection` という 4 つの Op が使われている．Op は Python のメソッドとほぼ一対一に対応しているが，`groupby_agg` は `groupby` メソッドと `sum` メソッドを合わせて一つの Op としているように，必ずしも一対一の必要はなくライブラリ API と IR は独立している．ライブラリ API を IR に変換するのはフロントエンドの役割である．

```
1 %df = Ducks.read_csv("sales.csv")
2 %daily = Ducks.groupby_agg(%df, "date", "sum")
3 %tmp = Ducks.project(%daily, ["qty", "amt"])
4 Ducks.write_csv("daily.csv", %tmp)
```

図 3　Ducks IR の例（見やすさのため簡略化してある）

3.2　フロントエンド

Ducks は現在 Python 用のフロントエンドを備えており，Pandas と互換の API を提供している．図 1 のプログラムの Ducks 版を図 4 に示すが，違いは 1，2 行目の `import` 文のみである[*1]．このような互換性を実現するため Ducks の Python フロントエンドは，Pandas と同じクラスやメソッドを定義している．Pandas はデータフレームに対する 200 以上の操作を提供しているが，さらに各操作はオプション引数によって動作を変えることができ（例えば `read_csv` は 50 以上の引数がある），それを含めると数えきれないほどのバリエーションがある．これら全てに対応する IR を定義することは大変であり，また非効率であるため Ducks のフロントエンドは IR を生成する機能に加えて，フロントエンドで Pandas を呼び出す fallback 機能を備えている．

```
1 # import Pandas as pd
2 import Ducks.Pandas as pd
3 df = pd.read_csv("sales.csv")
4 daily = df.groupby("date").sum()
5 daily[["qty", "amt"]].to_csv("daily.csv")
```

図 4　図 1 のデータフレームプログラムの Ducks 版

この二つの機能を図 5 に示す Ducks の `read_csv` メソッドを例に説明する．ここでは Ducks IR の `read_csv` Op が，Pandas の `read_csv` メソッドのキーワード引数 (図中 `kwargs`) に対応してない場合を例としている[*2]．そこで `read_csv` メソッドは，`kwargs` がない場合は IR 生成を行い（図 5 ～ 6 行目），ある場合は Pandas の `read_csv` メソッドを呼び出す fallback を行う（図 2 ～ 4 行目）．IR 生成の場合は，IR 生成部（IR Builder）に対して `read_csv` Op を生成するように指示する．IR Builder は Op を生成して記録すると伴に，その Op の出力結果へのハンドルを返し，Ducks の DataFrame はこのハンドルを保持するオブジェクトとして生成される．なお，`Ducks.DataFrame` クラスの各メソッドは，`read_csv` と同様に IR 生成および fallback 機能を持つメソッドとして定義されている．

一方で fallback の場合は，Pandas の `read_csv` メソッドを呼び出しその結果を `Ducks.DataFrame.from_Pandas` により Ducks の DataFrame に変換している．このように fallback 機能は，`Ducks.DataFrame` と `Pandas.DataFrame` の間の変換を自動で行うことで，Ducks IR が対応してない操作に対して透過的に Pandas を利用することを可能としている．これにより，ユーザーに対しては Pandas と同じ API を提供しているように見せることができる．

```
1 def read_csv(filename, **kwargs):
2   if kwargs:
3     return Ducks.DataFrame.from_Pandas(
4       Pandas.read_csv(filename, **kwargs))
5
6   value = irbuilder.build_op(
7             read_csv_op, [filename])
8   return Ducks.DataFrame(value)
```

図 5　フロントエンドによる IR 生成と fallback（簡略版）

3.2.1　遅延実行の開始

例に出てきた `to_csv` やデータフレームを文字列に変換する Python の特殊メソッド `__repr__` などのいくつかのメソッドは評価ポイントとして定義されており，これらのメソッドの中で IR Builder が記録した IR の実行が開始される．図 6 に示した `to_csv` の例では，まず `to_csv` メソッド本来の処理に相当する `write_csv` Op を生成し，その後 Ducks のコア API である `Ducks.core.evaluate` を呼び出すことによって，実行を開始している．なお，前述した fallback も評価ポイントの一つであり，Pandas を呼び出す前に IR の実行を行う．

実行の際には，IR Builder は Op 間の依存解析を行い，記録した Op の中から実行が必要な Op を抽出するとともに，最適化パスの自由度を上げるための変数の生死解析を行う．前述の例では各 Op の出力変数（`%df`, `%daily`, `%tmp`）が `to_csv` 以降では不要であると分かれば，それらが計算されなくなるような最適化も可能となる（例えば `groupby_agg` と `projection` の順序を入れ替える最適化）．Ducks は実行時コンパイラである特徴を生かし，フロントエンドで Python のインタプリタ情報を利用した独自の生死解析を行っている（詳細は本稿の範囲を超えるので省略する）．

```
1 class DataFrame:
2   def to_csv(self, filename):
3     value = irbuilder.build_op(
4       write_csv_op, [filename, self._value])
5     return Ducks.core.evaluate(value)
```

図 6　遅延実行の開始（簡略版）

3.3　最適化パス

IR の実行前には最適化パスが呼び出される．最適化パスは IR を入力として IR を出力する変換パスであるが，

*1　Python の import の hook 機能を利用して，自動で Pandas を Ducks に置き換えることも可能である．
*2　説明を簡潔にするためであり，実際には IR 化に対応している引数もある．

Ducks IR はデータフレームの操作に対応した Op からなる IR であるため，最適化パスはデータフレーム操作に特化した最適化を行うことができる．例えば，テーブルから条件に合う行だけを取り出すフィルタの後に特定の列を取り出す場合に対して，順序を逆にする最適化がある．データフレームで一般的な列指向のデータ構造（列をメモリ上で連続的に配置する構造）では，列を取り出す操作のコストは低く，列を先に取り出すことによって，コストのかかる行を取り出す操作の対象を削減できる．この様な最適化は大きな効果が期待できる一方で，例えばそれぞれの Op の C 言語実装からこのような最適化を行うためには，C 言語で記述されたデータ構造やループ構造の解析が必要であるため困難であり，ドメイン特化ならではの最適化である．

このような最適化は，ユーザープログラムのパターンに応じた複数の最適化アルゴリズムによって行われるが，ユーザープログラムのパターンはアプリケーションやデータによって変遷して行くため，最適化アルゴリズムもそれに応じて進化していく必要がある．Ducks ではこのようなドメイン特化最適化を，他のモジュールとは独立した IR 最適化として実現しており，最適化パスの追加によって性能を強化していくことができる．

3.4 バックエンドによる IR 実行

Ducks のバックエンドは IR のテーブル型に対する具体的なデータ構造と，各 Op を実行するカーネルを提供するライブラリとして定義される．Ducks の IR 実行部（IR Executor）は，IR 中の Op の依存関係に従って，バックエンドのカーネルを呼び出すことで IR の実行を行う．バックエンドは他のモジュールとは独立しているだけでなく，IR Executor は複数のバックエンドを登録し，実行時に選択する仕組みを持っている．これにより，バックエンドを切り替えることで，ユーザープログラムの変更なく複数のターゲットプラットフォームに対応することが可能で，アクセラレータを使った高速化を可能とする．

4. Ducks の実装

本節では Ducks の実装について述べる．Ducks IR の定義には LLVM プロジェクトで開発されている MLIR[4] を用いている．MLIR は独自の IR を定義するためのインフラストラクチャであり，IR 定義情報からコードを生成する機能を通して，IR のパース，Op の追加や削除などの IR の操作，基本的な最適化（Dead code elimination など）を提供している．最適化パスの開発には，LLVM の強力なエコシステムを利用することができる．

IR の実行には，Tensorflow runtime(TFRT) を用いている．TFRT は Deep Learning（DL）フレームワークの TensorFlow 向けに開発されているライブラリで，DL 用の IR を実行する機能を提供している．Ducks では TFRT のコア部分は DL に依存せず，MLIR で記述された IR に対して，中間結果のメモリ管理などの実行管理を行いながら，Op を登録されたカーネルを呼び出して実行する機能であることに着目し，Ducks のバックエンドの提供するカーネルを TFRT に登録し，IR の実行に利用している．

現在の Ducks は，図 2 に示すように，二つの CPU 用のバックエンドと，二つのアクセラレータ用のバックエンドを備えている．CPU 用のバックエンドとしては，主に動作テストや性能比較用に開発している IR 上の各 Op に対して Pandas を呼び出すバックエンドと，CPU 向けの高速化を目的とした DFKL（DataFrame Kernel Library）が提供するカーネルを利用するバックエンドがある．DFKL は筆者らが開発しているライブラリで，列指向データ向けのデータフォーマットを提供しているライブラリ Apache Arrow をベースとして，データフレーム向けのカーネルの追加や独自の並列化などを行ったものである．一方で，アクセラレータ用のバックエンドとしては，ベクトルプロセッサ（SX-Aurora TSUBASA の Vector Engine[5]）用バックエンドと GPU 用のバックエンドがある．前者はベクトルプロセッサ向けのデータ分析ライブラリ frovedis[6] のデータフレームを，後者は GPU 向けのデータフレームライブラリである cudf[*3]を利用している．

4.1 ドメイン特化最適化

ここではドメイン特化最適化の例として，特徴量エンジニアリング向けのライブラリ xfeat[*4]等で提供されている集約特徴量計算に対する最適化を取り上げる．集約特徴量とは，テーブルデータをキー列でグループ化し，ターゲット列に対して合計，平均，最大，最小など各種の集約演算を施した結果を特徴量とするものである．図 7(a) は，xfeat の集約特徴量の実装を元に Ducks で生成した IR を，Op をノードとした計算グラフとして表現したものである（ここでは簡単のため集約演算として合計と平均を行う場合の例を示しており，図は簡略化されている）．(a) では，`read_csv` で読み込んだテーブルに対して，`groupby_agg` を `sum` を引数として呼び出し合計を計算した結果を元のテーブルに結合（`join`）し，その後再び `groupby_agg` によって平均を計算し再度結合している．このような処理によって元のテーブルに集約特徴量を列として追加したテーブルを作成している．

一方図 7(b) は，最適化パスによって変換された計算グラフを示している．(a) では集約計算を行ったあとに元のテーブルへ結果を結合していたが，(b) ではまず集約演算同士の結果を結合したのちに元のテーブルへ統合している．グループ化の結果は元のテーブルよりも小さなテーブルになるため，集約計算後の結果同士を結合することで途中結

*3 https://github.com/rapidsai/cudf
*4 https://github.com/pfnet-research/xfeat

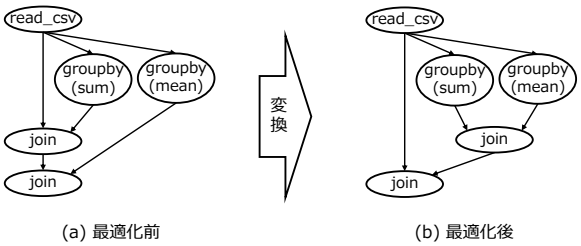

図 7　ドメイン特化最適化の例

果のテーブルを小さくすることができる．この最適化は，グループ化や結合といった演算の意味を理解した上で，グループ化結果を元のテーブルに連続して結合しているというパターンを検出し，順序の変更を行ってもプログラムの意味を壊さないことを判断して行っており，ドメイン特化な最適化パスでなければ実現は困難である．

5. 評価

本稿では，Ducks の評価として，Pandas で書かれたプログラムと，import 文を Ducks に変更したプログラムの性能を比較することで，同一 API での性能向上を評価する．本評価ではターゲットプラットフォームを合わせるため，Ducks では CPU 向けの DFKL バックエンドを用い，どちらも CPU 上で動かす．CPU には Intel Xeon Gold 6226（12 コア）を用いた．また Pandas はバージョン 1.3.3 を用いた．

本評価では，TPC（コンピュータシステムの性能評価の方法や基準などを策定する業界団体）が策定しているベンチマークの中から，現実世界の ETL（抽出，変換，ロード）および機械学習のワークフローを対象とした TPCx-BB ベンチマーク[*5]，BI やデータウェアハウス分野の検索や抽出などのワークロードを対象とした TPC-H ベンチマーク[*6]を用いる．これらのベンチマークは SQL での参照実装が公開されているが，今回はまずこれを元に Pandas 版プログラムを開発し，その import 文を置き換えた．

図 8 に TPCx-BB，図 9 に TPC-H での評価結果を Pandas の性能（実行時間の逆数）を 1 とした相対性能として示す．TPCx-BB には全部で 30 種類のクエリと呼ばれる処理パターンが含まれているが，ここでは Pandas が実行時間に支配的な 23 クエリの結果を示している．Ducks は 23 クエリにおいて最小でも 1.5 倍，最大では 12 倍，平均で 5.3 倍高速であった．一方，TPC-H は 22 クエリを含み，その全てが Pandas が支配的であり全クエリを評価した．ただし，TPC-H では入力ファイルの読み込み時間が支配的であり，これを含めるとクエリ毎の時間が見えなくなるため，ファイル読み込みは除いて計測した（ファイル読み込み単体の評価では Ducks は 12 倍高速であった）．その結果 22 クエリに対して，Ducks は最小で 1.4 倍，最大 1.7 倍，平均 6.4 倍高速であった．両ベンチマークでの評価から，多くのデータ分析や前処理で Ducks を用いることで，import 分以外の書き換えは不要で，数〜十数倍高速化できることが確認できた．

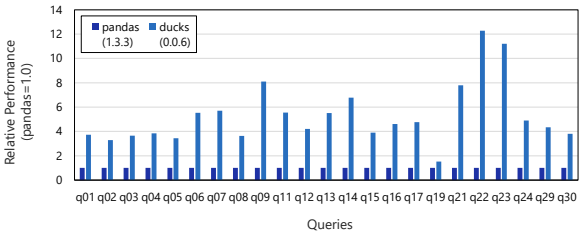

図 8　TPCx-BB での Ducks の性能評価（平均 5.3 倍）

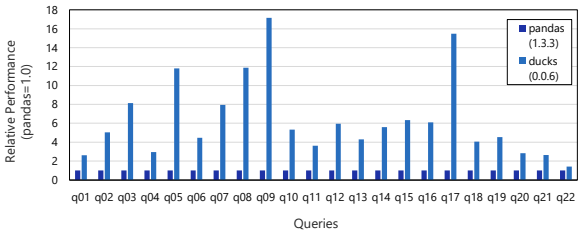

図 9　TPC-H での Ducks 性能評価（平均 6.4 倍）

6. 関連技術

Pandas の普及とともに高速化のニーズが高まり，いくつかの高速なデータフレームライブラリが開発されている．VAEX[7] や Dask[8] は，Pandas がシングルノードでのインメモリを対象としておりメモリサイズを超えるような大きなデータを扱えないことを主要な課題とし，これらは Ducks のように Pandas との互換性を重視するよりは，out-of-core に対応することや，マルチノードに対応することでより大きなデータセットを扱えるようにしている．

一方で Ducks と同様にシングルノードでの高速化を行うものとして modin[3] と polars[*7]がある．Rust で実装された polars は，Python API はあるものの Pandas 互換ではなく，SIMD 化やマルチコア化によった高速性を特徴としている．一方では，modin は Ducks と同様に import 文変更のみよる drop-in replace での高速化をターゲットとしているが，Ducks のような IR を中心としたモジュール構造や実行時コンパイルの仕組みは持っておらず，アクセラレータには対応していない．データフレーム をアクセラレータで高速化するものとしては，Ducks のバックエンドとしても利用しているベクトルプロセッサ向けの frovedis と GPU 向けの cudf がある．Ducks と異なりそれぞれターゲットとするアクセラレータ専用であるが，Python API を持ち，Pandas との間での変換 API も提供しており，Pandas の高速化に利用することができる．

[*5] https://www.tpc.org/tpcx-bb/
[*6] https://www.tpc.org/tpch/
[*7] https://github.com/pola-rs/polars

テーブルデータに対するグループ化，結合と行った要素処理の高速化は，データベースシステムを中心に古くから行われており，最近では Intel プロセッサの SIMD 命令を使うもの [9] や，ベクトルプロセッサを使うような研究 [10] も行われている．これらの研究は DFKL や frovedis バックエンドを Ducks に通して取り込まれことで，データフレームの高速化への寄与も期待できる．

最後に，実行時コンパイル技術の関連研究について述べる．Ducks のような実行時コンパイル技術は，Deep Learning 向けのフレームワークで利用が広がってきた．Deep Learning はその計算量の多さから GPU をはじめとするアクセラレータの利用が進んでおり，複数のアクセラレータへの対応や Deep Learning の各レイヤで行われる特徴的な演算向けのドメイン特化最適化を行うことを目的として，TensorFlow の XLA[11]，pytorch の GLOW[12]，Apache TVM[13] といった Deep Learning 用のコンパイラが開発されてきた．Ducks はこれらの進歩に触発され，データフレームにコンパイラ技術を導入することで，同様にアクセラレータやドメイン特化最適化による高速化を目的としているが，Pandas という標準的なライブラリの存在するデータフレームでは，fallback などの仕組みを導入することで既存ライブラリとの互換性を向上させることも主要なターゲットにしている．

7. おわりに

本稿では，データフレームに実行時コンパイラ技術を導入した Python ライブラリ Ducks の紹介を行った．Ducks はコンパイラ技術を利用することで，IR を中心としたモジュール化されたアーキテクチャを実現し，ドメイン特化最適化やアクセラレータによる高速化を，フロントエンド API とは独立して行うことができる．本特徴を利用することで，Ducks は Pandas から import 文の変更のみで利用可能な API を提供しながら高速化を行うことができ，同一 CPU 上で Pandas に対して TPCx-BB, TPC-H ベンチマークの 45 クエリでの評価で平均 5.8 倍の性能向上が得られることを示した．本稿では詳しく触れなかった，Ducks の最適化パスやアクセラレータを活用した高速化については後続の報告を待たれたい．

参考文献

[1] Conway, S., Sorensen, R., Joseph, E. C. and Monroe, K.: IDC FutureScape: Worldwide High-Performance Data Analysis 2017 Predictions, `https://hyperionresearch.com/wp-content/uploads/2018/01/HPDA_16-IDC-FutureScape-Worldwide-High-Performance-Data-Analysis-2017-Predictions.pdf`.

[2] : The State of Data Science 2020 Moving from hype toward maturity, `https://www.anaconda.com/state-of-data-science-2020`.

[3] Petersohn, D., Macke, S., Xin, D., Ma, W., Lee, D., Mo, X., Gonzalez, J. E., Hellerstein, J. M., Joseph, A. D. and Parameswaran, A.: Towards scalable dataframe systems, *arXiv preprint arXiv:2001.00888* (2020).

[4] Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N. and Zinenko, O.: MLIR: A compiler infrastructure for the end of Moore's law, *arXiv preprint arXiv:2002.11054* (2020).

[5] Yamada, Y. and Momose, S.: Vector engine processor of NEC's brand-new supercomputer SX-Aurora TSUBASA, *Proceedings of A Symposium on High Performance Chips, Hot Chips*, Vol. 30, pp. 19–21 (2018).

[6] Araki, T.: Accelerating machine learning on sparse datasets with a distributed memory vector architecture, *2017 16th International Symposium on Parallel and Distributed Computing (ISPDC)*, IEEE, pp. 112–121 (2017).

[7] Breddels, M. A. and Veljanoski, J.: Vaex: big data exploration in the era of gaia, *Astronomy & Astrophysics*, Vol. 618, p. A13 (2018).

[8] Rocklin, M.: Dask: Parallel computation with blocked algorithms and task scheduling, *Proceedings of the 14th python in science conference*, Vol. 130, SciPy Austin, TX, p. 136 (2015).

[9] Polychroniou, O. and Ross, K. A.: Towards practical vectorized analytical query engines, *Proceedings of the 15th International Workshop on Data Management on New Hardware*, pp. 1–7 (2019).

[10] Pietrzyk, J., Habich, D., Damme, P., Focht, E. and Lehner, W.: Evaluating the vector supercomputer sx-aurora TSUBASA as a co-processor for in-memory database systems, *Datenbank-Spektrum*, Vol. 19, No. 3, pp. 183–197 (2019).

[11] Leary and Wang, T.: XLA: Tensorflow, compiled, TensorFlow Dev Summit, 2017.

[12] Rotem, N., Fix, J., Abdulrasool, S., Catron, G., Deng, S., Dzhabarov, R., Gibson, N., Hegeman, J., Lele, M., Levenstein, R. et al.: Glow: Graph lowering compiler techniques for neural networks, *arXiv preprint arXiv:1805.00907* (2018).

[13] Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L. et al.: TVM: An automated End-to-End optimizing compiler for deep learning, *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594 (2018).