DEIM Forum 2023 2a-4-5

実行時コンパイラを用いたデータフレーム処理の高速化

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

† NEC　〒 211-8666 神奈川県川崎市中原区下沼部 1753

あらまし　データ分析に広く用いられるデータフレームライブラリ Pandas と互換の API で高速化を実現するための実行時コンパイル技術を用いた高速化手法を提案する．提案手法では実行時にユーザープログラムを一度 HW 非依存のデータフレーム用の中間言語に変換し，その後ターゲット HW 用のバックエンドを用いて実行する．同手法を実装したデータフレームライブラリ Ducks はマルチコア CPU 用に並列化されたバックエンド，アクセラレータ用のバックエンドを持ち，ユーザープログラムの変更なく高性能な HW を活用可能である．TPC-H ベンチマークの 22 個のクエリでの性能評価では Ducks は，CPU 上での Pandas に対して，同一 CPU で平均 6.4 倍，アクセラレータ（ベクトルプロセッサ）で平均 35 倍の高速化を実現した．

キーワード　並列・分散処理，先進ハードウェア活用・GPU，問合せ処理

1　は じ め に

データの収集，管理，分析といったデータから知見を導きだすための技術の著しい発展により，IT 業界だけでなく様々な業界で，データを活用することによる業務効率の向上，生産プロセスの改善，顧客満足度の向上などに取り組んでいる．データ活用を進める組織は，データの収集・管理基盤を構築し，データサイエンティストの雇用や教育を進め，それぞれの組織や業界に適したデータ活用を模索している．このようなデータ活用は大企業で先行しているが，中小企業にも裾野を広げていくことが経済成長に重要であると言われている [1]．

図 1 は分析に活用されるデータの割合を示している [1]．様々なデータが分析に利用されているが，中でも顧客データ，経理データ，電子メール，アクセスログが多くの組織で利用されており，また POS データや e コマースにおける販売記録データ，センサーデータなどのマシンが生み出すデータの活用がここ 5 年で伸びている（図中赤枠）[1]．このような種類のデータの分析に活用されるのがデータフレームライブラリである．

データフレームライブラリは，表形式のデータを表すデータ構造とそれに対する各種の操作からなり，上記のようなデータの処理に適している．特に Python 用ライブラリの Pandas は，Python による高い開発効率に加えて，データのグループ化，結合，フィルタなどのデータ分析の頻出処理を直感的な API で提供しており，その高い使い勝手からデータサイエンティストに選ばれている．Stack Overflow[2] による人気調査では，データ分析用ライブラリである scikit-learn や TensorFlow, PyTorch などをダブルスコアで上回り第 3 位となっている [2][3]．また，同調査では初学者への人気が高いことも示されており，今後データ活用の裾野の広がりとともに益々利用が進むと考えられる．

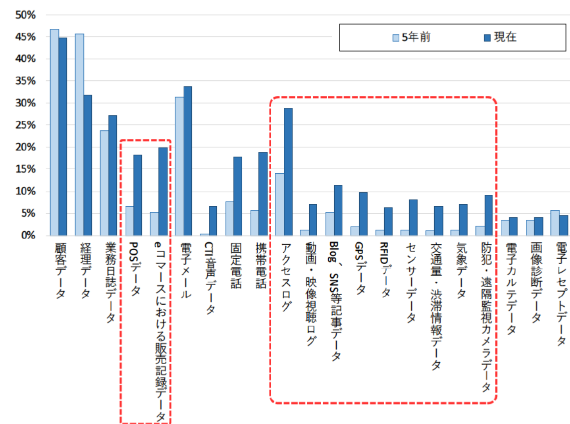

図 1　分析に活用しているデータ（総務省「デジタルデータの経済的価値の計測と活用の現状に関する調査研究」[1] を元に作成）

Pandas はシングルノードでのインメモリ処理を対象とし，並列処理や分散処理等の高性能計算の専門知識は不要であり，データサイエンスを学んだ各業界の専門家にも利用しやすい．一方で計算機の性能に目を向けると，メモリ容量に特化したハイエンドサーバーは数 TB のメインメモリを備え（図 2），コストパフォーマンスに優れたミッドレンジサーバーでも数百 GB のメモリを持つようになってきており，シングルノードでも大規模なデータを扱えるようになってきている．しかし，演算性能の向上はマルチコア化によって進んでおり，多くの処理がシングルスレッドで動作する Pandas は，データ量の増大や分析処理の複雑化に伴い処理速度の遅さが問題になってきている．高速なデータ分析を目的とした Dask [3] や VAEX [4] などのライブラリが開発されているが，Pandas とは API が異なり高速化を意識したプログラミングが必要で，Pandas に慣れ親しんだデータサイエンティストには敷居が高い．

1：製造業，情報通信業，サービス業等の全 2003 組織へのアンケート調査
2：プログラミング技術に関する Q&A サイト
3：過去一年間に良く利用もしくは今後利用したい技術のアンケート調査の Web 開発用以外のフレームワーク/ライブラリ部門．1 位は.NET，2 位は numpy

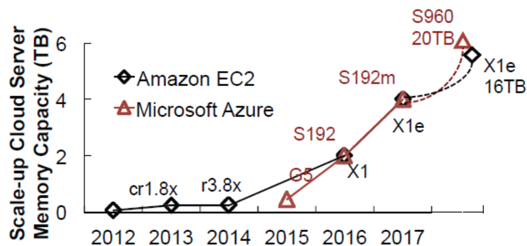

図 2　クラウドサーバーのメモリ容量 [5]

本稿では，実行時コンパイル技術を用いることで Pandas と互換の API を提供しながらデータフレーム処理を高速化する手法を，同手法を実装したライブラリ Ducks を用いて説明する．

2　データフレームライブラリ Ducks

Ducks の全体像を図 3 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モジュールが Ducks の構成要素を示す．白地のモジュールであるフロントエンド，最適化パス，バックエンドがデータフレーム固有のモジュールであり，灰色地は汎用性のあるモジュールである．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．

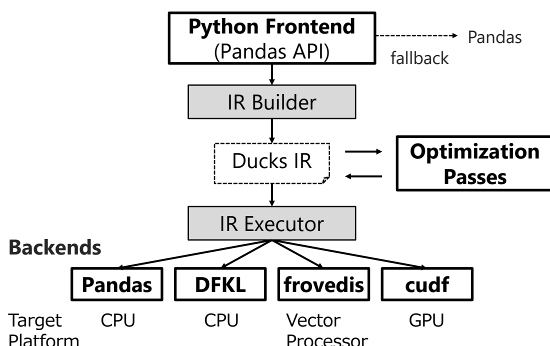

図 3　Ducks の全体像

2.1　Ducks IR

Ducks 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 ではなく，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，表を表す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．Ducks IR の例を図 4 に示す．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，列を取り出す projection という 4 つの Op が使われている．

2.2　フロントエンド

Define-by-run 方式によりユーザープログラムの実行中に IR を生成するのがフロントエンドである．Ducks は現在 Python 用のフロントエンドを備え，Pandas と互換の API を提供している．Ducks を用いたプログラム例を図 5 に示すが，Pandas との違いは 1，2 行目の import 文のみである [4]．Pandas はデータフレームに対する 200 以上の操作を提供しており，さらに各操作は引数によって動作を変えることができる（例えば read_csv の引数は 50 以上）．これら全てに対応する IR を定義することは非効率であるため Ducks のフロントエンドは IR 生成機能に加えて，フロントエンドで Pandas を呼び出す fallback 機能を備えている．

```
1 %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 4　Ducks IR で書かれたプログラム例（簡略化してある）

```
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 5　Ducks で記述したプログラム例

この二つの機能を図 6 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，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 [5]．read_csv メソッドは，kwargs 無しで呼ばれた場合は IR 生成を行い（5〜6 行目），ありの場合は Pandas の read_csv メソッドを呼び出す fallback を行う（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.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．

```
1 def read_csv(filename, **kwargs):
2   if kwargs: # fallback
3     return ducks.DataFrame.from_pandas(
4       pandas.read_csv(filename, **kwargs))
5   value = irbuilder.build_op(
6             read_csv_op, [filename])
7   return ducks.DataFrame(value)
```

図 6　フロントエンドによる IR 生成と fallback（簡略版）

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 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．

4：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．
5：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．

2.3 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 7 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，必要に応じて Pandas を呼び出す前に IR の実行を行う．

```
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 7 遅延実行の開始（簡略版）

2.4 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 を入出力する変換パスであり，複数の最適化パスを利用できる．Ducks IR はデータフレームに特化した IR であるため，最適化パスはデータフレーム特化の最適化を行うことができ，データベースの Query Optimizer のような機能を提供できる．

Pandas の速度課題の一つとして，同じ処理でもプログラムの書き方によっては著しく実行時間がかかると言う問題がある．Pandas のようなハイレベルな API を提供するライブラリを利用したプログラム開発では，高性能計算の専門家にとっても処理速度の観点で最適化な実装を行うためには，API に対する深い理解とプログラミングスキルが必要となる．そのような専門性を持たないデータサイエンティストが最適な実装を行うことは難しく，最適化による高速化が重要である．Ducks の最適化パスは IR をインターフェースとして他のモジュールと独立しているため拡張性が高く，最適化パスの拡充により高速化を実現できることは，コンパイラ技術を利用する利点の一つである．

2.5 データフレーム向け最適化の例（集約特徴量計算）

最適化パスの例として，特徴量エンジニアリング向けのライブラリ xfeat[6]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の結果を特徴量とするものである．図 8(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している（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いる）．この実装では，read_csv で読んだテーブルに対して，groupby_agg で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することで，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．

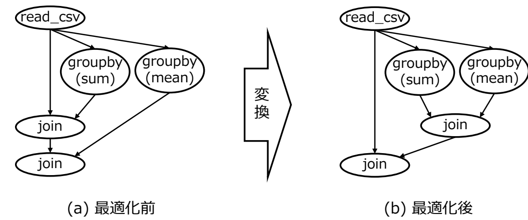

図 8 最適化パスによる変換例

一方，図 8(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おり，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．

2.6 バックエンドによる IR 実行

Ducks のバックエンドは IR が定義する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．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 に複数のバックエンドを登録し，実行時に選択することができる．これによりバックエンドを切替のみで，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可能で，アクセラレータの利用が容易になる．

2.7 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 [6] を用いている．MLIR は独自の IR を定義するためのインフラであり，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 TensorFlow（TF）向けに開発されているライブラリである．TFRT のコア部分は TF に依存せず，MLIR で記述された IR に対して，登録されたカーネルを用いて実行管理を行う機能であることに着目し，Ducks ではバックエンドの提供するカーネルを登録し IR の実行に利用している．

現在の Ducks は図 3 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を目的とした 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 は列指向データ向けのデータフォーマットを提

6 : https://github.com/pfnet-research/xfeat

供しているライブラリ Apache Arrow をベースとして，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，アクセラレータ用としては，ベクトルプロセッサ（SX-Aurora TSUBASA の Vector Engine [7]）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 [8] を，後者は GPU 向けのデータフレームライブラリである cudf[7]を利用している．なお DFKL と frovedis は筆者らが開発している．

3 評　　価

本節では，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．今回の評価では Ducks は Pandas と同じ CPU での実行に加え，アクセラレータとしてベクトルプロセッサ（SX-Aurora TSUBASA）を用いた場合を評価する．CPU は Intel Xeon Gold 6226（12 コア），ベクトルプロセッサは VE10BE，Pandas はバージョン 1.3.3 を用いた．また本評価では，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 ベンチマークを用いる（Scale Factor は 10）．TPC-H の SQL での参照実装を元に Pandas 版プログラムを開発し，その import 文を置き換えた．

評価結果を図 9 に 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．TPC-H の 22 クエリの平均で Ducks は同一 CPU 上で Pandas に対して 6.4 倍の高速化，ベクトルプロセッサを利用することで 35 倍の高速化を，プログラムとしては import 文の書き換えのみで達成できることが確認できた．

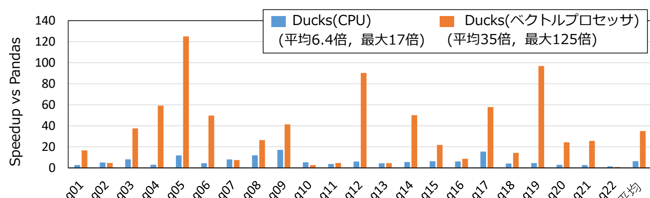

図 9　TPC-H（sf=10）を用いた Ducks と Pandas の性能比較

4 関 連 技 術

Ducks と同様に Pandas と互換の API で高速化を行うライブラリとして modin [9] がある．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．どちらも Python API を持つが，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．

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広がってきた．Deep Learning はその計算量の多さから GPU 等のアクセラレータの利用が進んでおり，複数アクセラレータへ対応などのため，TensorFlow の XLA [10]，pytorch の GLOW [11], Apache TVM [12] といった Deep Learning 用のコンパイラが開発されてきた．Ducks は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など互換性も主要なターゲットにしている．

5 お わ り に

本稿では，実行時コンパイラ技術によるデータフレーム処理の高速化手法を提案した．同手法を実装したデータフレームライブラリ Ducks は，Pandas と互換の API を提供しながら TPC-H の 22 クエリの平均で CPU 上で Pandas の 6.4 倍，ベクトルプロセッサを利用することで 35 倍の高速化を実現できること示した．

文　　献

[1] 情報通信総合研究所：デジタルデータの経済的価値の計測と活用の現状に関する調査研究の請負報告書，https://www.soumu.go.jp/johotsusintokei/linkdata/r02_05_houkoku.pdf (2020).
[2] Stack Overflow: Stack Overflow Annual Developer Survey 2022, https://survey.stackoverflow.co/2022 (2022).
[3] 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).
[4] Breddels, M. A. et al.: Vaex: Big Data exploration in the era of Gaia, *Astronomy & Astrophysics*, Vol. 618, p. A13 (2018).
[5] Ogleari, M. et al.: String figure: A scalable and elastic memory network architecture, *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, pp. 647–660 (2019).
[6] Lattner, C. et al.: MLIR: A compiler infrastructure for the end of Moore's law, *arXiv preprint arXiv:2002.11054* (2020).
[7] 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).
[8] 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).
[9] Petersohn, D. et al.: Towards scalable dataframe systems, *arXiv preprint arXiv:2001.00888* (2020).
[10] Leary and Wang, T.: XLA: Tensorflow, compiled, TensorFlow Dev Summit, 2017.
[11] Rotem, N. et al.: Glow: Graph lowering compiler techniques for neural networks, *arXiv preprint arXiv:1805.00907* (2018).
[12] Chen, T. 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).

7：https://github.com/rapidsai/cudf