実行時コンパイラを用いたデータフレーム高速化技術

2023/06/22

石坂一久

概要

実行時コンパイラ技術を使ってPandasの高速化版を作りました

```
# import pandas as pd
import ducks.pandas as pd
```

※ Pandas: Pythonのデータフレームライブラリ

テーブルデータ

- 顧客データ，統計データ，センサーデータなどのテーブル形式のデータ
- 画像や音声などの生データではない

	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	1325317920	4.39	4.39	4.39	4.39	0.455581	2.000000	4.390000
1	1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
4857372	1617148560	58714.31	58714.31	58686.00	58686.00	1.384487	81259.372187	58692.753339
4857373	1617148620	58683.97	58693.43	58683.97	58685.81	7.294848	428158.146640	58693.226508
4857374	1617148680	58693.43	58723.84	58693.43	58723.84	1.705682	100117.070370	58696.198496
4857375	1617148740	58742.18	58770.38	58742.18	58760.59	0.720415	42332.958633	58761.866202
4857376	1617148800	58767.75	58778.18	58755.97	58778.18	2.712831	159417.751000	58764.349363

4857377 rows × 8 columns

bitcoinの1分ごとの価格情報

https://www.kaggle.com/datasets/mczielinski/bitcoin-historical-data

データフレーム（Pandas）

- テーブルデータ用のデータ構造と操作
- PythonのPandasが有名

ワイルドなデータの扱いに向く

- 欠損値を扱える
- 列の追加や削除が柔軟に行える

データサイエンティストに好まれる

- Pythonによる開発生産性
- 機械学習アルゴリズムとの連携

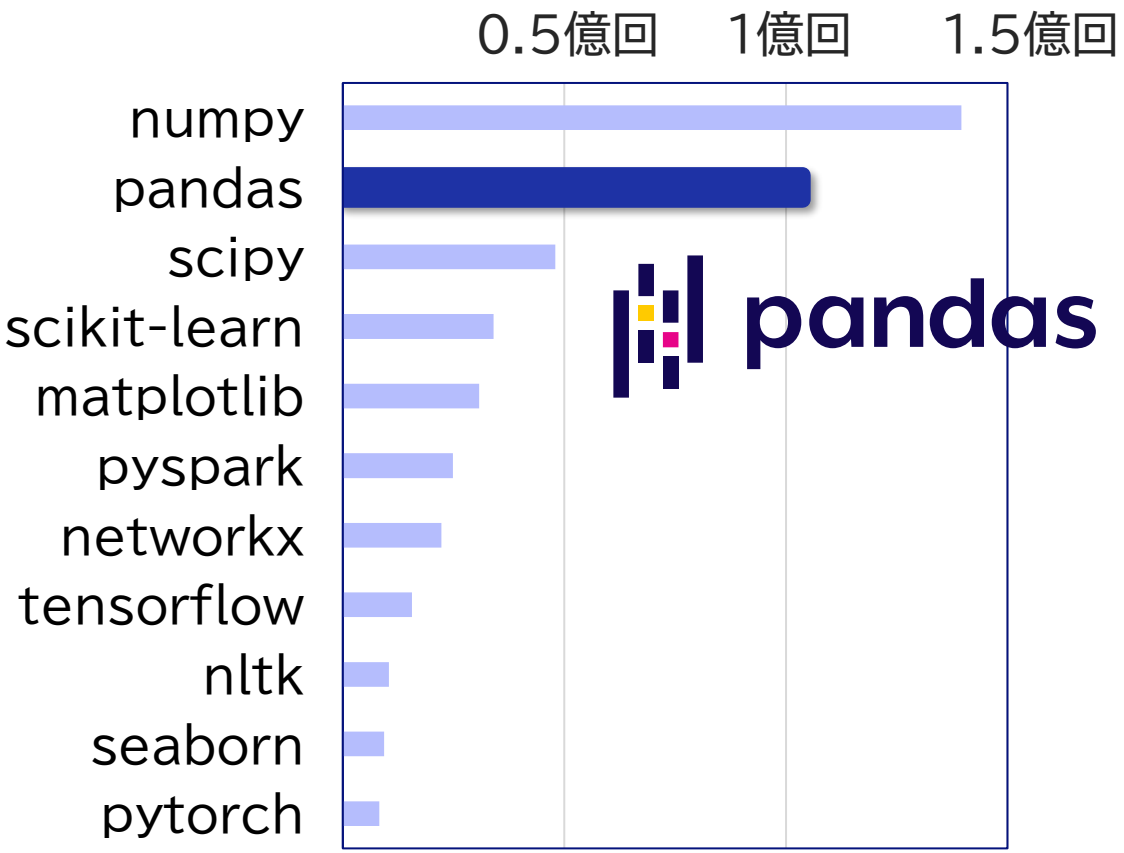
	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	1325317920	4.39	4.39	4.39	4.39	0.455581	2.000000	4.390000
1	1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
4857372	1617148560	58714.31	58714.31	58686.00	58686.00	1.384487	81259.372187	58692.753339
4857373	1617148620	58683.97	58693.43	58683.97	58685.81	7.294848	428158.146640	58693.226508
4857374	1617148680	58693.43	58723.84	58693.43	58723.84	1.705682	100117.070370	58696.198496
4857375	1617148740	58742.18	58770.38	58742.18	58760.59	0.720415	42332.958633	58761.866202
4857376	1617148800	58767.75	58778.18	58755.97	58778.18	2.712831	159417.751000	58764.349363

4857377 rows × 8 columns

欠損値（NaN）

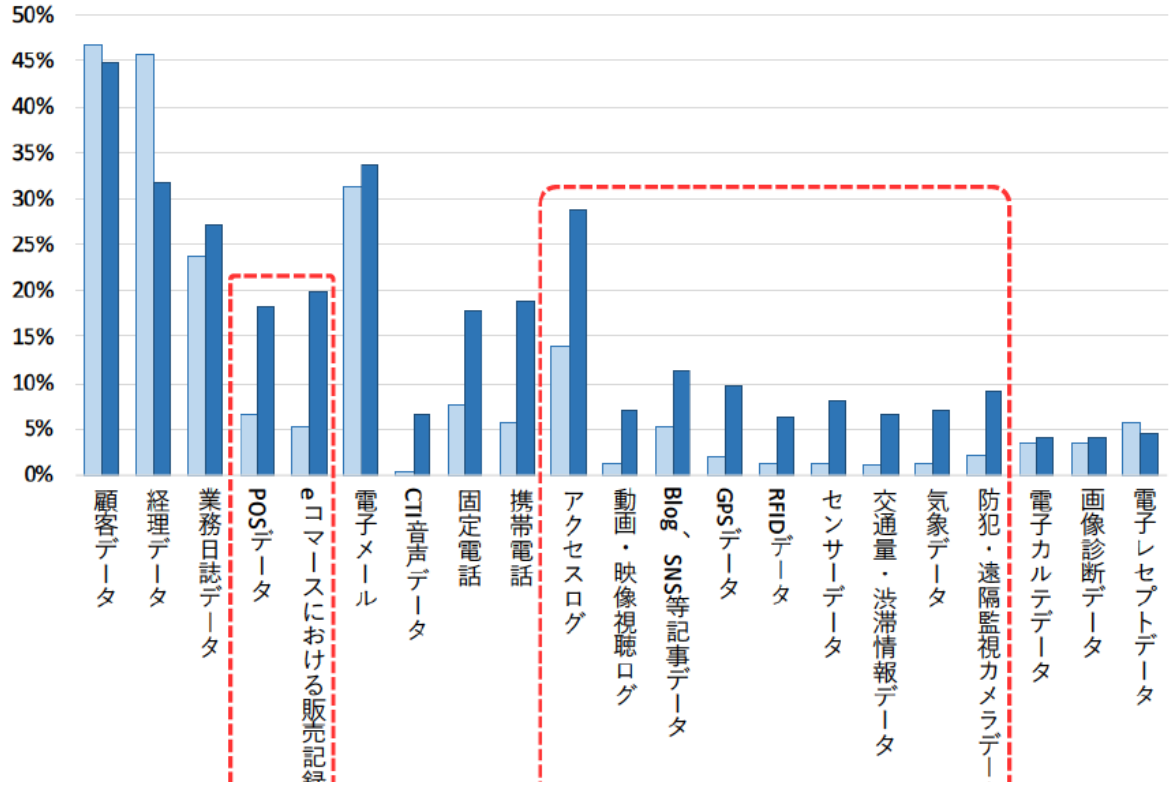
Pandas

月間一億回ダウンロードされるデータ分析の標準ツール



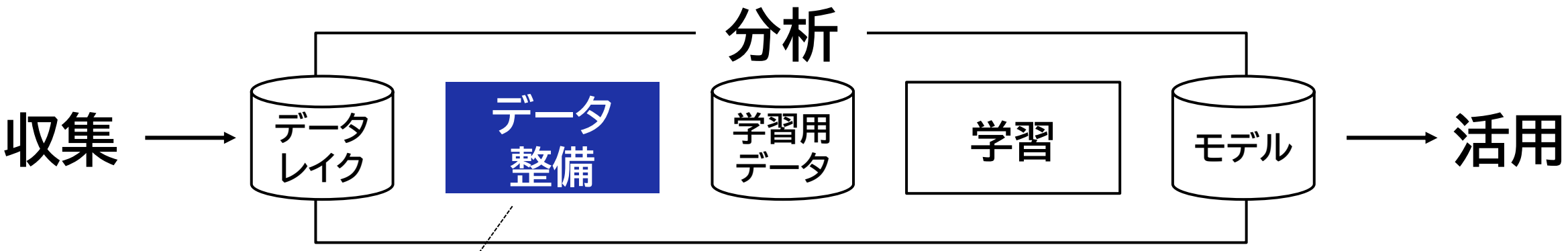
pypiの月間ダウンロード数
（データ分析関係のライブラリ）

分析に活用しているデータ



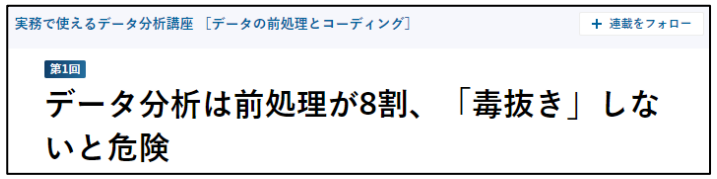
総務省「デジタルデータの経済的価値の計測と活用の現状に関する調査研究」(2020)

機械学習により重要度を増すデータフレーム



データ整備

- 探索的データ解析，学習用データの作成などの前処理
- 単純な整形だけでなく，複雑なアルゴリズムも登場
- データ規模の増大とともに分析のボトルネック
  - データ分析の時間の8割を占めるとも言われる

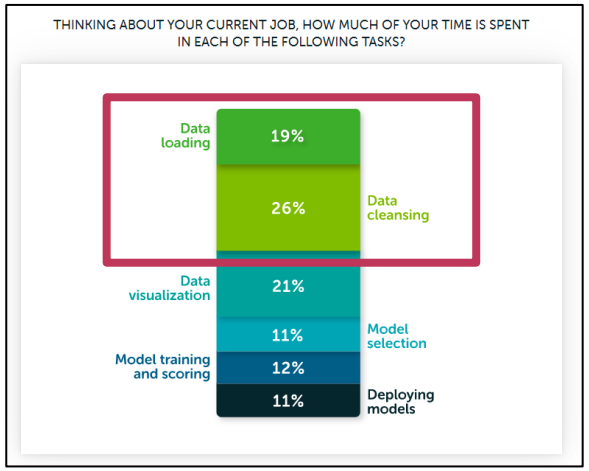

実務で使えるデータ分析講座［データの前処理とコーディング］ ＋ 連載をフォロー

第1回

データ分析は前処理が8割、「毒抜き」しないと危険

https://xtech.nikkei.com/atcl/learning/lecture/19/00110/00001/

データサイエンティストの時間の40%以上



Anaconda
The State of Data Science 2020

Pandasプログラムの例

Open列の一日ごとの平均を求める例

```
import pandas as pd

df = pd.read_csv("bitstampUSD_1-min_data_2012-01-01_to_2021-03-31.csv")

df["Day"] = df["Timestamp"] // (3600*24) * 3600*24

grouped = df[["Day", "Open"]].groupby("Day").agg("mean")
grouped.columns = ["Open_daily_mean"]

merged = df.merge(grouped, how="left", lef
```

csvファイル読み込み

ベクトルデータ演算→列の追加

グループ平均

列名変更

テーブル結合

入力データ(再掲)

	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	1325317920	4.39	4.39	4.39	4.39	0.455581	2.000000	4.390000
1	1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
4857372	1617148560	58714.31	58714.31	58686.00	58686.00	1.384487	81259.372187	58692.753339
4857373	1617148620	58683.97	58693.43	58683.97	58685.81	7.294848	428158.146640	58693.226508
4857374	1617148680	58693.43	58723.84	58693.43	58723.84	1.705682	100117.070370	58696.198496
4857375	1617148740	58742.18	58770.38	58742.18	58760.59	0.720415	42332.958633	58761.866202
4857376	1617148800	58767.75	58778.18	58755.97	58778.18	2.712831	159417.751000	58764.349363

4857377 rows × 8 columns

出力データ

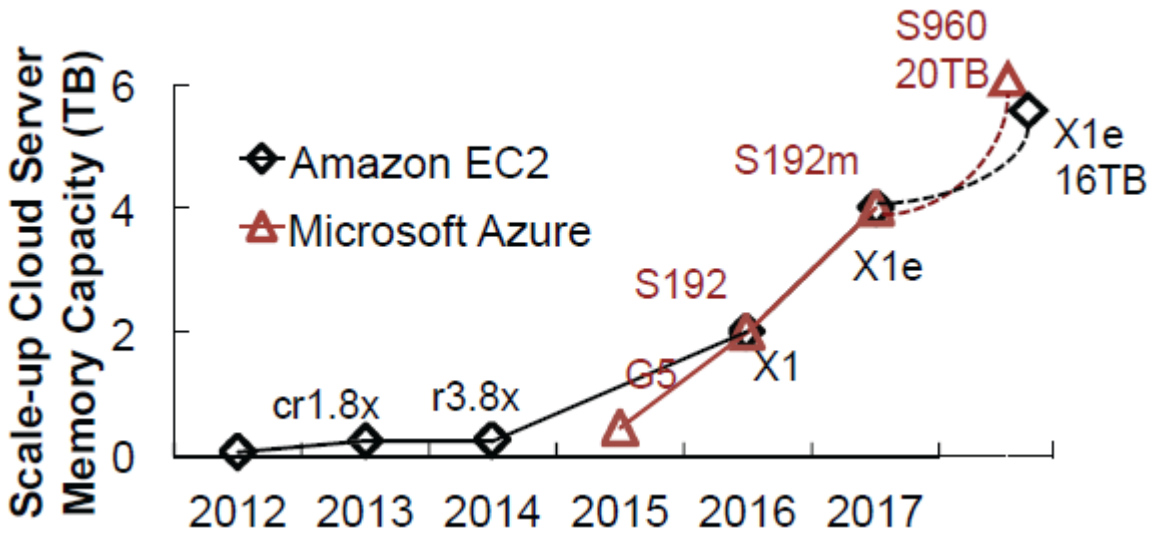
日毎の平均

	Timestamp	Day	Open	Open_daily_mean
0	1325317920	1325289600	4.39	4.465000
1	1325317980	1325289600	NaN	4.465000
2	1325318040	1325289600	NaN	4.465000
3	1325318100	1325289600	NaN	4.465000
4	1325318160	1325289600	NaN	4.465000
...	...	...	...	...
4857372	1617148560	1617062400	58714.31	58347.805624
4857373	1617148620	1617062400	58683.97	58347.805624
4857374	1617148680	1617062400	58693.43	58347.805624
4857375	1617148740	1617062400	58742.18	58347.805624
4857376	1617148800	1617148800	58767.75	58767.750000

Pandasの速度課題

扱うデータ量や処理の複雑化に伴い速度課題が顕在化

クラウドサーバーのメモリ容量



[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)

Pandasの速度課題の要因

- 多くの処理はシングルスレッド実行
- アクセラレータ非対応
- Eager実行モデル
- 「遅い」書き方ができてしまう

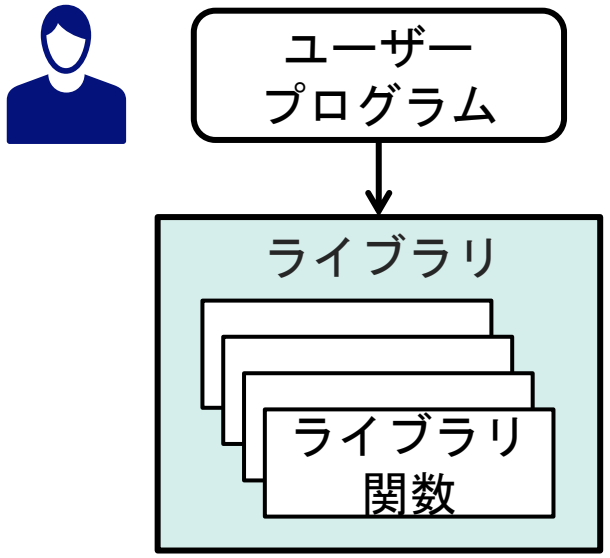
新しいデータフレームライブライも登場

polars　MODIN

dask　vaex.io

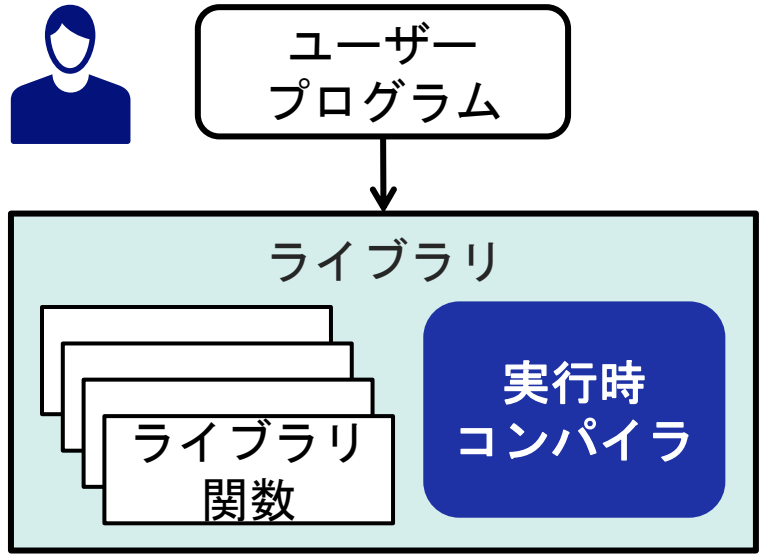
提案: 実行時コンパイラ技術を使ったデータフレーム高速化

従来型のライブラリ



ユーザーが呼び出した順に
ライブラリ関数が実行される

実行時コンパイラ入りライブラリ



ライブラリに埋め込まれた
実行時コンパイラが最適化してから実行

実行時コンパイラを活用しAPI互換での高速化**を目指す**

Pandasの速度課題へのアプローチ

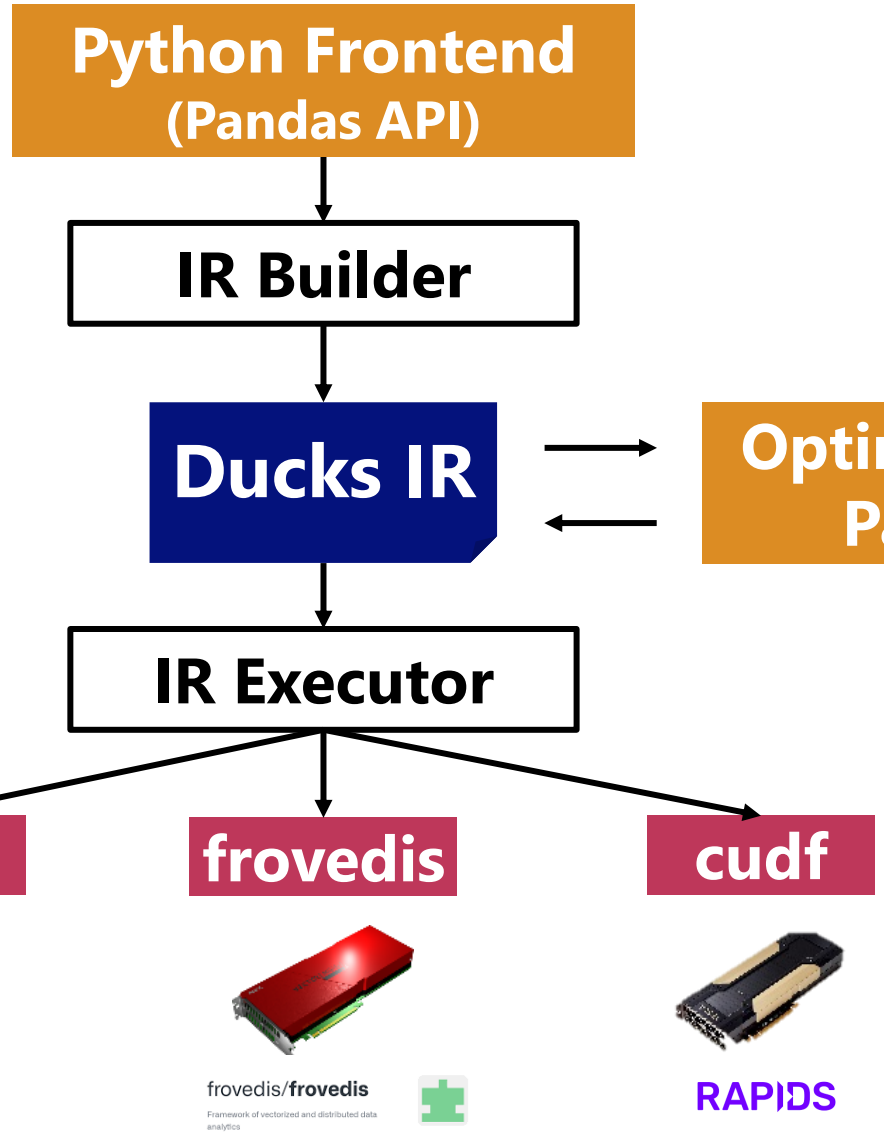
コンパイラを用いてAPIを中間言語に変換してから実行

Pandasの速度課題	アプローチ
◆ 多くの処理はシングルスレッド実行 ◆ アクセラレータ非対応	APIとその実装を分離し， 高速な実装を利用可能に
◆ Eager実行モデル ◆ 「遅い」書き方ができてしまう	遅延実行および データフレーム固有の最適化

本手法に基づく**データフレームライブラリDucks**を開発中

Ducks: Architecture

中間言語(IR)を中心とした
モジュール構造
- フロントエンド
- 最適化パス
- バックエンド

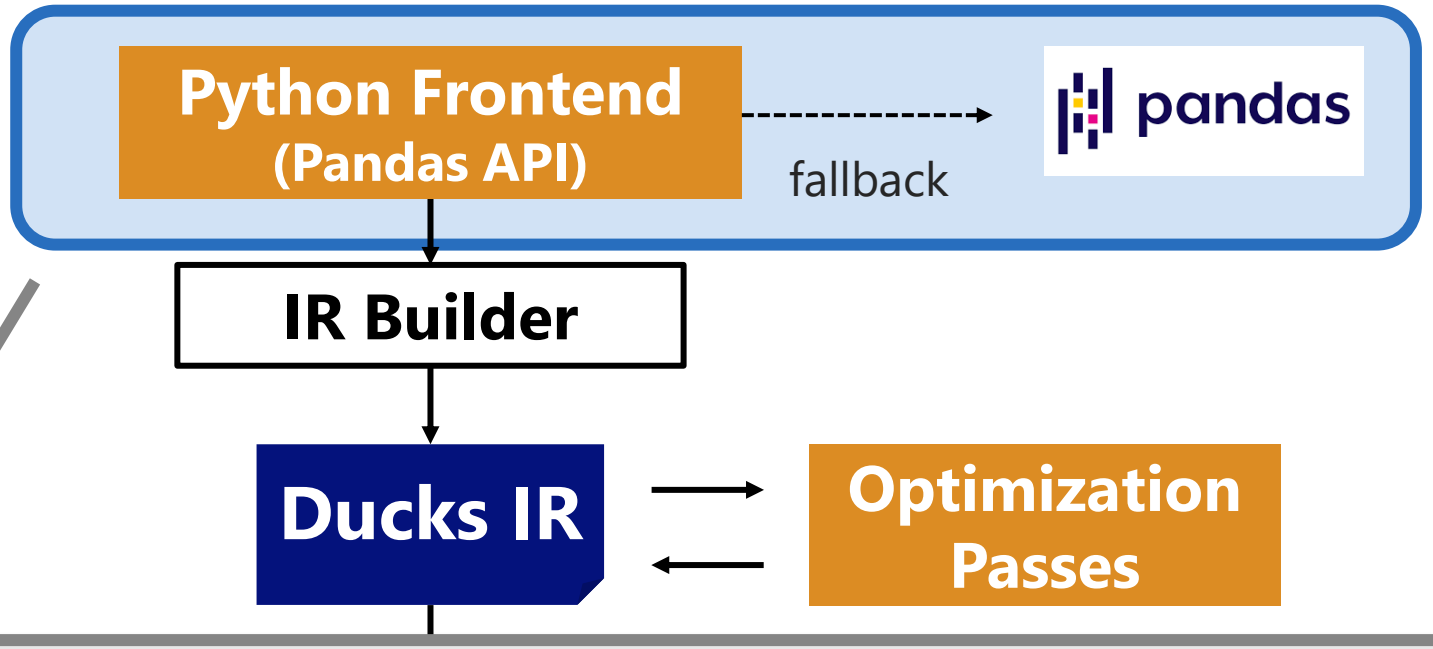


コンパイラフレームワーク
LLVM/MLIRを利用

MLIR

(1) フロントエンド

実行時にAPI呼び出し
毎に中間言語を生成
(Define-by-run)



中間言語の例

Pythonプログラム

```
import pandas as pd

df = pd.read_csv(“1.csv”)
g = groupby(“a”).sum()
```

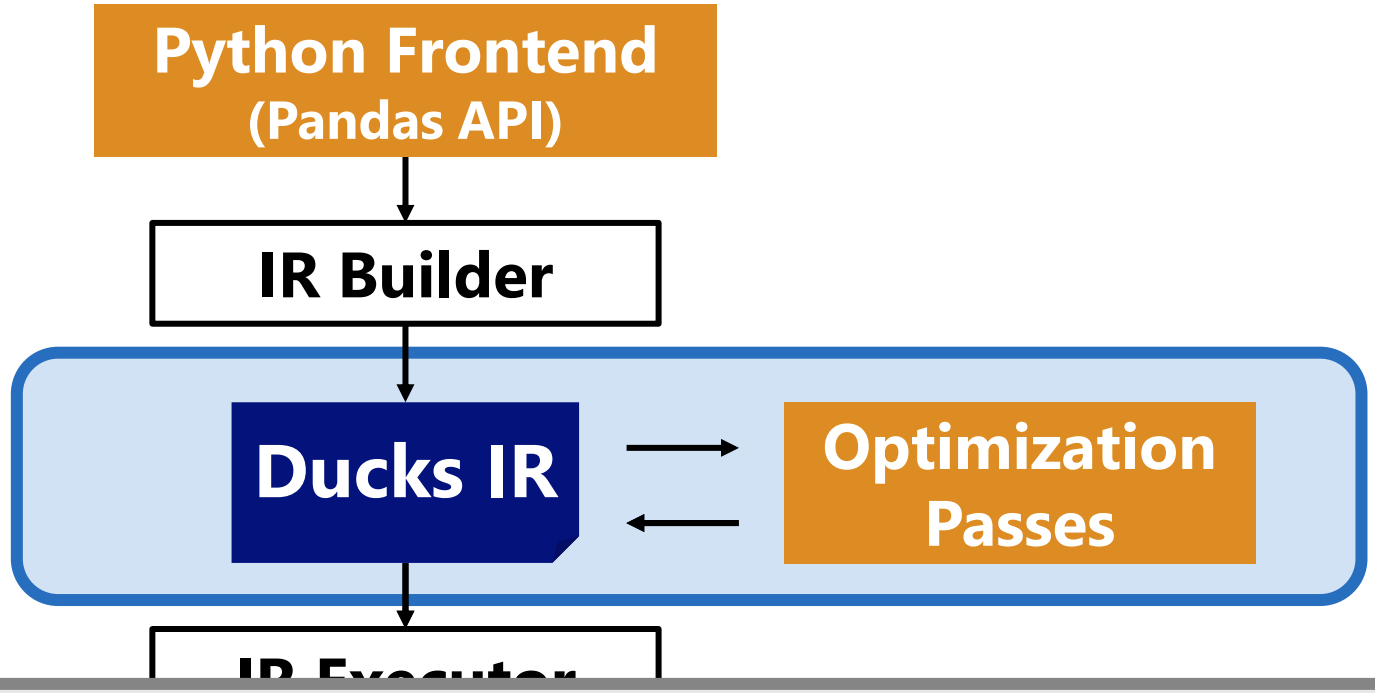
IR（中間言語）

```
%v0 = ducks.read_csv(“1.csv”)
%v1 = ducks.groupby_agg(
          %v0, “a”, “sum”)
```

(2) 最適化パス

最適化パスがIRをより良いIRに変換

色々な最適化アルゴリズムを実装することで性能を強化



最適化例

入力IR

```
%v1 = ducks.filter(%v0, %cond)
%v2 = ducks.project(%v1, “a”)
```

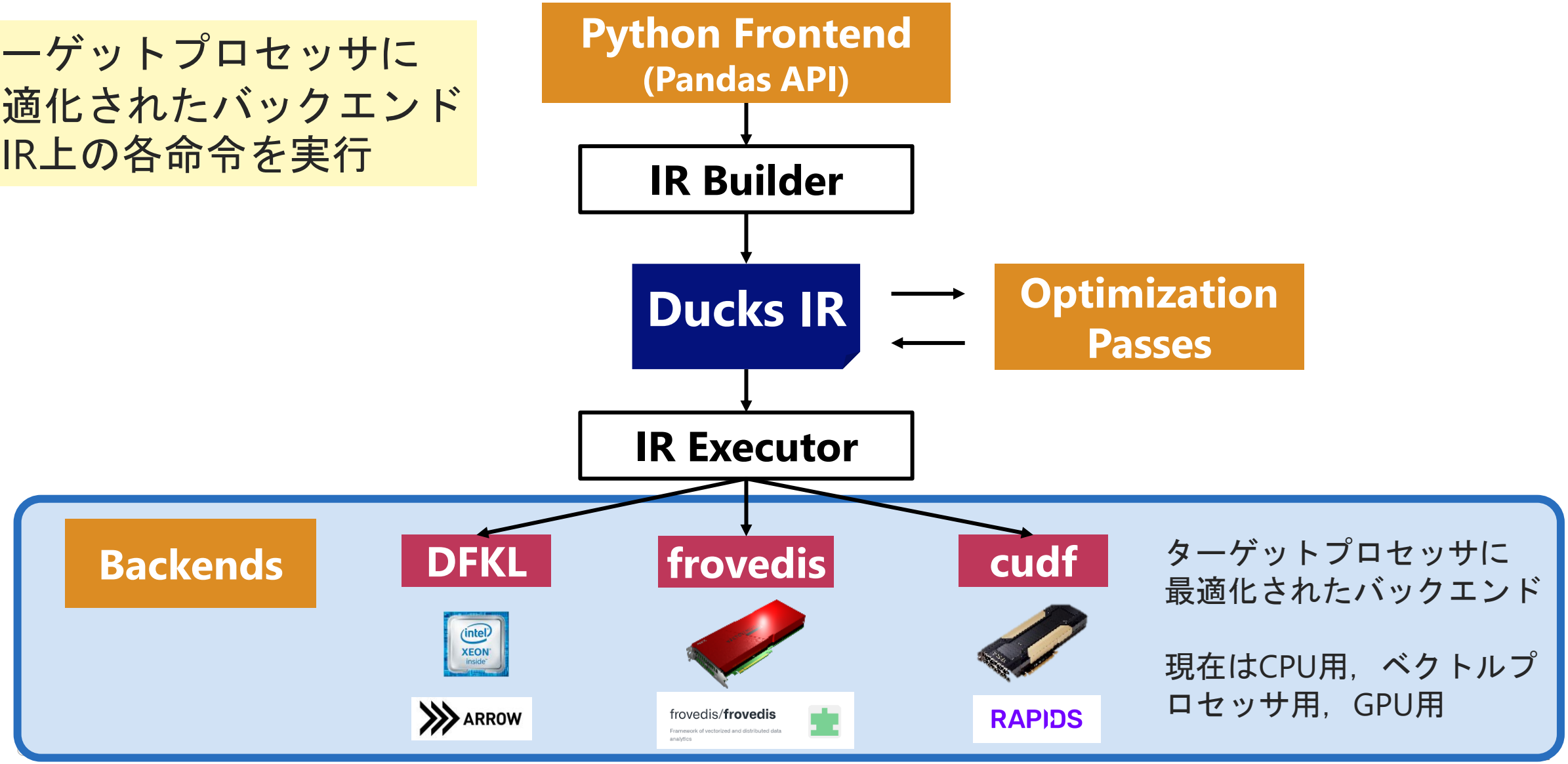
出力IR

```
%v1 = ducks.project(%v0, “a”)
%v2 = ducks.filter(%v1, %cond)
```

filter(行の抽出)とproject(列の抽出)の入れ替え

(3) バックエンド

ターゲットプロセッサに
最適化されたバックエンド
がIR上の各命令を実行



TPCベンチマークでの評価結果（CPU）

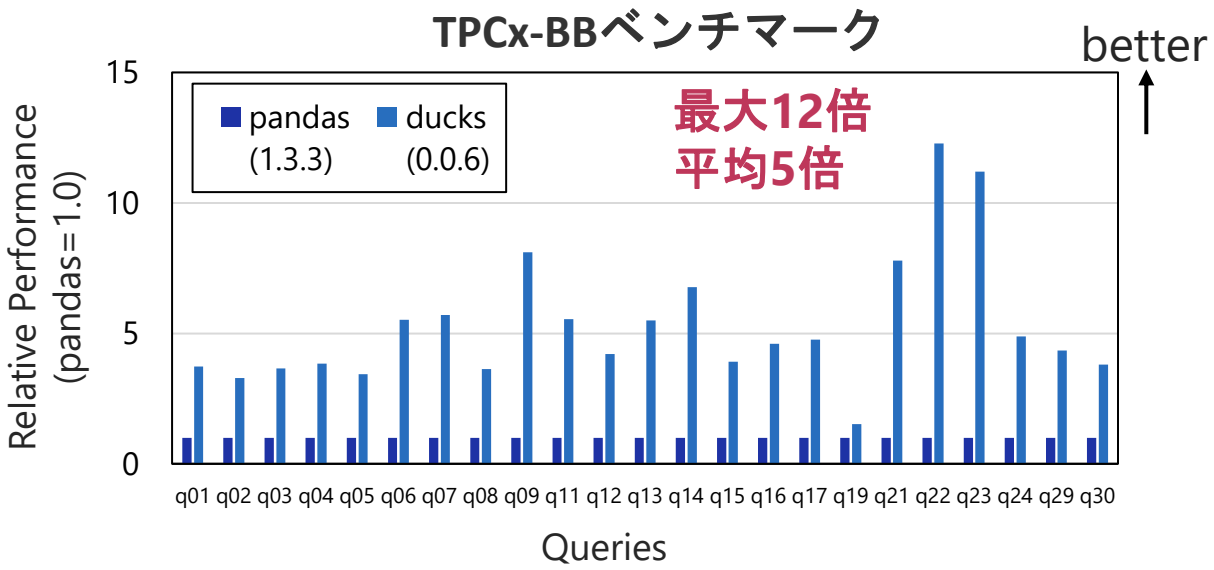
DFKLバックエンドを用いたCPU上での評価

1) import 文の変更のみで，

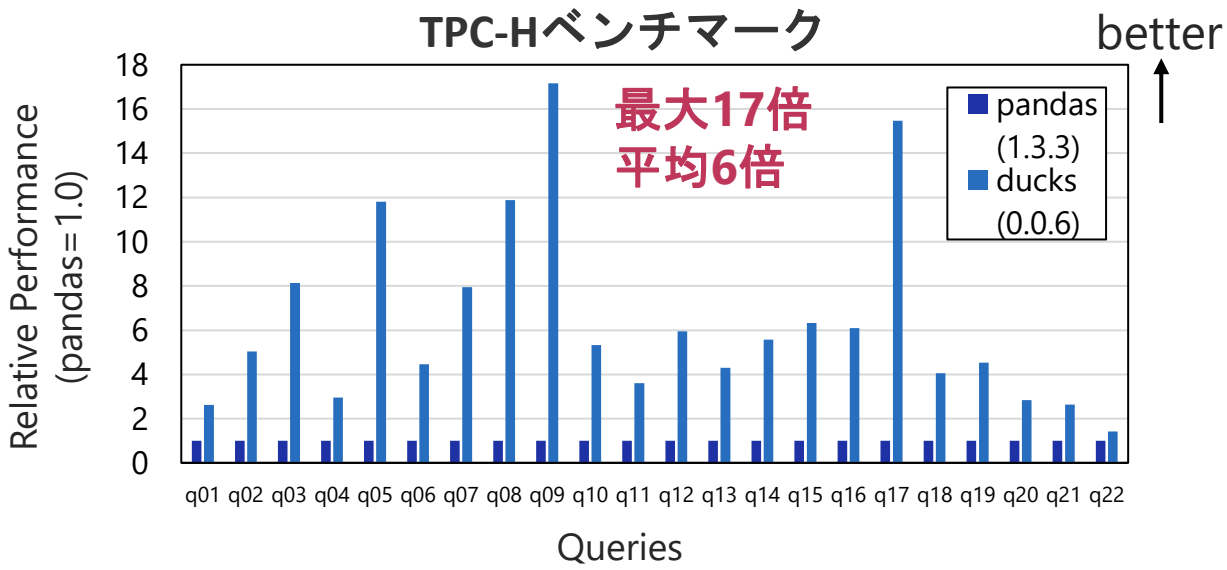
2) Pandas に対して最大17 倍，平均5.8 倍（45クエリ）

利用サーバー

Xeon Gold 6226
(12コア), 192GB



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ベンチマーク



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ベンチマーク

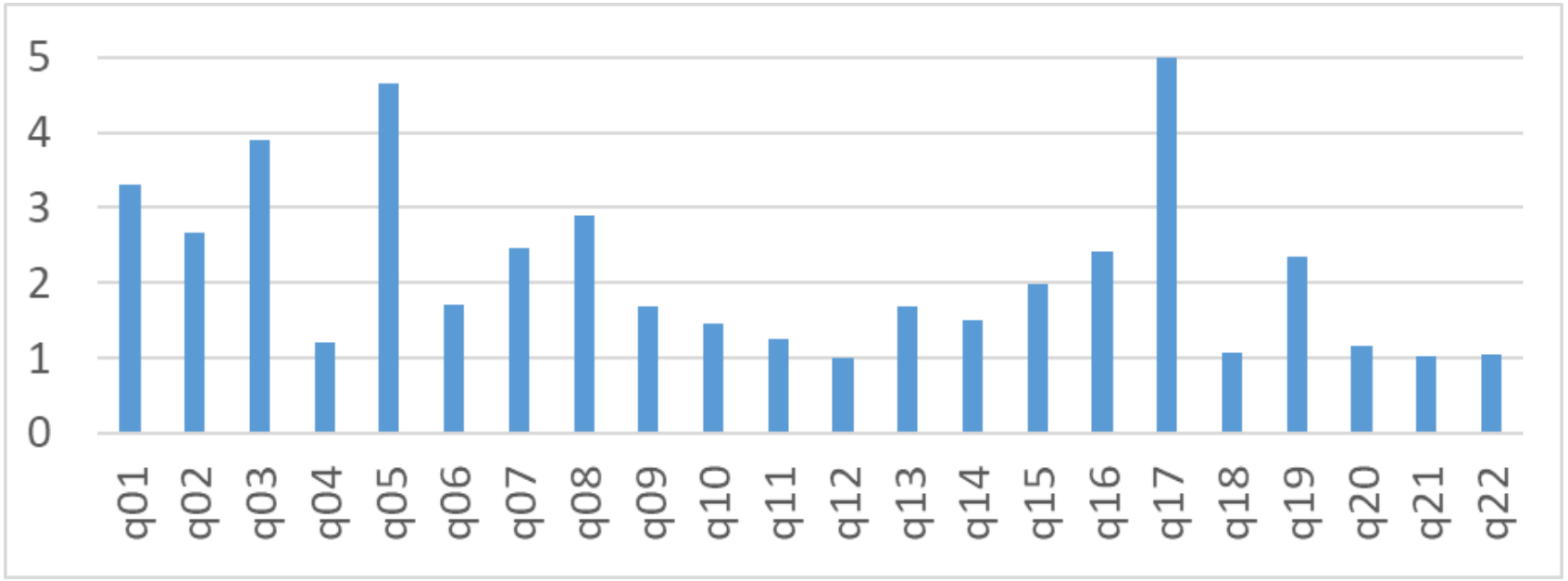
ドメイン特化自動最適化の例

Projection最適化

列の抽出(project)を可能な限り，前に出して演算量を削減する最適化

→ プログラマが遅い書き方をしても高速に実行

```
%v1 = ducks.filter(%v0, %cond)
%v2 = ducks.project(%v1, “a”)
```



Projection最適化による性能向上（TPC-Hベンチマーク）

最適化パスを充実させることで，プログラマの書き方によらない高速性を実現

まとめ

- 高速データフレームライブラリDucks
  - 実行時コンパイラ技術を活用して，PandasとAPI互換で高速化を実現

- ご興味がありましたら，ぜひお声がけください

NECのHPCプラットフォーム

ベクトルシステム/クラスタで、長い経験と豊富な実績を持ち、蓄積された技術とノウハウで、 HPCのトータルソリューションをご提供いたします

多様なアプリケーションに対応し、
最新のオープンアーキテクチャを採用した
HPCクラスタ

LXシリーズ

NECの独自技を採用し「超高性能」と
「使いやすさ」を両立する
ベクトルシステム

SX-Aurora TSUBASAシリーズ

NEC HPCクラスタソリューション LXシリーズ

必要とされる様々なコンポーネントを組み合わせて、お客様の利用環境や
アプリケーションに合わせた最適なシステム／ソリューションをご提供します

- ◆ 最新プロセッサ搭載
  - Intel Xeon Scalable Processor
- ◆ 省電力・省スペース設計
  - 高密度実装、高効率電源採用
- ◆ InfiniBand対応
  - 最新のInfiniBand NDRに対応
- ◆ GPUサポート
  - NVIDIA H100に対応
- ◆ 安心の全国オンサイト保守
  - 全国約400カ所の保守拠点による迅速な対応

2U 2Nサーバ

2U 4Nサーバ

2U 1Nサーバ

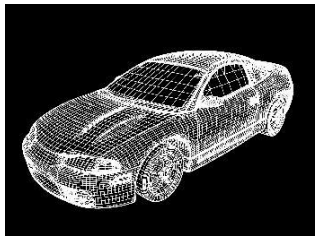
InfiniBandスイッチ

4U-8GPUサーバ

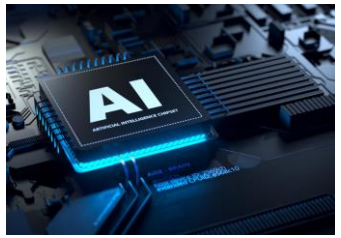
空力設計

ビッグデータ解析

素材開発



構造設計



AI

物流

NECのHPCプラットフォームに対する情報・お問合せ

◆最新事例、アプリ一覧、スペックなどは
製品WEBサイトをご覧ください。

https://jpn.nec.com/hpc/

◆お問い合わせ先

E-mail: info@hpc.jp.nec.com