Scaling High-Performance Python with Minimal Effort

Ehsan Totoni
Research Scientist, Intel Labs
STAC Summit NYC, Nov. 1st, 2017
Legal Disclaimer

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel, the Intel logo, Intel Xeon are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

2017 © Intel Corporation.

*Other names and brands may be claimed as the property of others
Motivation

Data analytics is the greatest value driver in technology

Financial services need insights from data
  • Exploit market data for financial modeling, etc.

High performance big data analytics is crucial
  • Democratize HPC for data scientists

http://www.businesscomputingworld.co.uk/
Productivity-Performance Gap

Scripting languages like Python are productive but slow and serial

Big data frameworks (Hadoop/Spark) are hard to use and slow

- High overhead runtime libraries
- Not based on parallel computing fundamentals

High performance requires low-level programming

- Not practical for interactive workflows of data scientists and their expertise
Motivation

Logistic Regression on Amazon AWS

NOT STAC BENCHMARK

Evolution Time (s)

Amazon AWS c4.8xlarge instances (vCPUs)

Spark

MPI/C++

4.28

2.15

1.11

127.22

64.1

53x

Totoni et al. “A Case Against Tiny Tasks in Iterative Analytics”, HotOS’17
Overview

High performance/scalability for analytics/ML/AI with little effort

- Minimal changes to scripting source code

Compiler optimization and parallelization

- Scripting program → efficient parallel binary

*High Performance Analytics Toolkit (HPAT)*

- Python (previously Julia)

https://github.com/IntelLabs/hpat

https://github.com/IntelLabs/HPAT.jl
def logistic_regression( iterations):
    f = h5py.File("lr.hdf5", "r")
    X = f['points'][:]
    Y = f['responses'][:]
    D = X.shape[1]
    w = np.ones(D) - 0.5
    for i in range( iterations):
        w -= np.dot(((1.0 / (1.0 + np.exp(-Y * np.dot(X, w)))) - 1.0) * Y), X)
return w
Data Parallelism Extraction

\[ D = A \times B + C \]

Recognize parallelism

\texttt{parfor } i=1:n \\
\texttt{t}[i]=A[i]*B[i]

\texttt{parfor } i=1:n \\
\texttt{D}[i]=t[i]+C[i]

Fuse loops

\texttt{parfor } i=1:n \\
\texttt{t}[i]=A[i]*B[i]+C[i]

---

\(^1\)Anderson et al. “Parallelizing Julia with a Non-invasive DSL”, \textit{ECOOP’17}
Spark Workflow

Rewrite by programmer

Python code → Spark API code → Spark Runtime → Cluster/cloud

HPAT Workflow

Compile by HPAT

Python code → Parallel binary (MPI) → Cluster/cloud

Driver

Executor 0 → Executor 1 → ... → Executor N-1

Rank 0 → Rank 1 → ... → Rank N-1
Performance Evaluation

NOT STAC BENCHMARKS

Amazon AWS
4 nodes c4.8xlarge (144 vCPUs)

20x-256x speedup of HPAT vs Spark

Cori at NERSC/LBL
64 nodes (2048 cores)

370x-2000x speedup of HPAT vs Spark

HPAT Julia used, Python will be similar

HPAT is within 2-4x MPI/C++

Totoni et al. “HPAT: High Performance Analytics with Scripting Ease-of-Use”, ICS’17
Pandas Example

```python
@hpat.jit(locals={'s_open': hpat.float64[:], ...})
def intraday_mean_revert():
    f = h5py.File("stock_data.hdf5", "r"); …
    for i in prange(nsym):
        symbol = sym_list[i]
        s_open = f[symbol+'/Open'][...]; …
        df = pd.DataFrame({'Open': s_open, ...})
        df['Stdev'] = df['Close'].rolling(window=90).std()
        df['Moving Average'] = df['Close'].rolling(window=20).mean()
        df['Criteria1'] = (df['Open'] - df['Low'].shift(1)) < -df['Stdev']
        df['Criteria2'] = df['Open'] > df['Moving Average']
        df['BUY'] = df['Criteria1'] & df['Criteria2']
        df['Pct Change'] = (df['Close'] - df['Open']) / df['Open']
        df['Rets'] = df['Pct Change'][df['BUY'] == True]
        n_days = len(df['Rets'])
        res = np.zeros(max_num_days)
        if n_days:
            res[-n_days:] = df['Rets'].fillna(0)
        all_res += res

```

100x speedup on 36 cores
HPAT Operations

Numpy:

- Element-wise operations: +, /, ==, exp, log, sqrt, ...
- Array creation: zeros, ones_like, random, normal, ...
- Others: sum, prod, dot, ...

Pandas:

- Column access, and operations: df.A, df['A'], df.A.std()
- Filter: df[df.A > .5]
- Rolling windows: df.A.rolling(window=5).mean()

Parallel loop:

```python
for i in prange(n):
    s += A[i]**2
```
Variable Type Limitation

Input code to HPAT should be statically compilable (type stable)

• Dynamic code example:

```python
if flag1:
    a = 2
else:
    a = np.ones(n)
if isinstance(a, np.ndarray):
    doWork(a)

if flag2:
    f = np.zeros
else:
    f = np.ones
b = f(m)
```

• Rare in analytics
Pandas Limitation

Data Frame column accesses should be static

- Dynamic code example:

```python
for i in range(5):
    A += df['c'+str(i)]
```

- Refactor to:

```python
A += df['c0']
A += df['c1']
A += df['c2']
A += df['c3']
A += df['c4']
```
Summary

Compiler approach superior to library approach for analytics

HPAT bridges productivity-performance gap

- Compiles Python programs to efficient parallel binaries
- Available on GitHub: https://github.com/IntelLabs/hpat
References


https://arxiv.org/abs/1611.04934


https://arxiv.org/abs/1704.02341