DASMatrix: A High-Performance Python Framework for Distributed Acoustic Sensing Data Analysis

Qianlong
Institute of Geophysics and Geomatics, China University of Geosciences

January 19, 2026

Abstract

Distributed Acoustic Sensing (DAS) technology is revolutionizing geophysics and engineering monitoring by transforming fiber optic cables into dense arrays of seismic sensors. However, the high spatial and temporal resolution of DAS generates massive datasets, often reaching terabytes per day, which poses significant challenges for data processing and analysis. Traditional tools often lack the scalability to handle such volumes efficiently or require proprietary software. We present **DASMatrix**, an open-source, high-performance Python framework designed specifically for DAS data. DASMatrix leverages *lazy loading* and *out-of-core* computing capabilities through Xarray and Dask to process datasets larger than available memory. It provides a fluent, chainable API for intuitive signal processing and supports high-performance visualization. We demonstrate the framework's architecture and performance through case studies, showing its effectiveness in handling large-scale DAS workflows.

1 Introduction

Distributed Acoustic Sensing (DAS) has emerged as a transformative sensing technology, enabling high-resolution strain rate measurements over long distances using standard fiber optic cables [5]. By interrogating the fiber with laser pulses and analyzing the backscattered Rayleigh light, DAS systems can act as thousands of synchronized sensors.

Despite its potential, the "Big Data" nature of DAS presents a major bottleneck [4]. A single DAS interrogator can sample thousands of channels at kilohertz frequencies, generating terabytes of data daily. This volume overwhelms traditional seismic processing workflows that rely on in-memory loading. Furthermore, existing solutions are often fragmented, relying on ad-hoc scripts or expensive proprietary software.

To address these challenges, we introduce **DASMatrix**, a comprehensive Python framework. Its key contributions are:

- **Scalability**: Built on the Xarray [2] and Dask [6] ecosystem to support out-of-core processing of massive datasets.
- **Usability**: A fluent `DASFrame` API that simplifies complex processing pipelines.
- **High Performance**: Optimized algorithms for filtering, spectral analysis, and beamforming.

2 System Architecture

The architecture of DASMatrix is designed with modularity and performance in mind. It consists of four main layers: Acquisition, Core API, Processing, and Visualization.

2.1 Data Acquisition Layer

The acquisition layer provides a unified interface for reading various DAS data formats. It supports standard formats like SEG-Y, MiniSEED, and proprietary raw binary formats (DAT, HDF5). The `DASReader` class abstracts the file I/O operations. Crucially, it supports *memory mapping*, allowing users to access metadata and signal slices without loading the entire file into RAM.

2.2 Core API: DASFrame

At the heart of the framework is the `DASFrame` class. It wraps an `xarray.DataArray` but adds DAS-specific semantics (distance, time, channel coordinates).

```
from DASMatrix import df
# Lazy load a large HDF5 file
frame = df.read("large_dataset.h5")
# Frame is lightweight; data is not
    loaded yet
print(frame)
```

Listing 1: Creating a DASFrame

2.3 Processing Engine

The processing engine leverages Dask to build a computation graph. Operations like filtering, detrending, and integration are added to the graph lazily. Execution is triggered only when `.collect()` or a visualization method is called. This strategy enables the processing of datasets that exceed the machine's physical memory.

3 Implementation Details

3.1 Lazy Evaluation & Chaining

DASMatrix implements a fluent interface where methods return a new `DASFrame` instance (or modify the internal graph). This allows for clean, readable code:

```
processed = (
    frame
    .detrend(axis="time")
    .bandpass(low=1.0, high=50.0)
    .normalize()
)
```

Listing 2: Chainable Processing Pipeline

Under the hood, `.bandpass` applies a Dask `map_overlap` operation to handle filtering across chunk boundaries seamlessly without edge artifacts.

3.2 High-Performance Kernels

For computationally intensive tasks, we utilize vectorized NumPy [1] operations and, where necessary, JIT compilation via Numba [3] to approach C-level speeds in pure Python.

4 Case Studies

4.1 Spectral Analysis Workflow

A common task in DAS is analyzing the frequency content of traffic or seismic signals. DASMatrix simplifies this to a few lines:

```
# Compute STFT spectrogram
frame.stft(nperseg=256).plot(cmap='
    inferno')
```

The framework automatically handles windowing, overlaps, and plotting axes.

4.2 F-K Filtering

To separate wavefields based on velocity, Frequency-Wavenumber (F-K) filtering is essential. DASMatrix provides an intuitive interface for this 2D transform operation.

```
# Remove low-velocity noise (v < 1000 m/s
    )
# dx specifies channel spacing in meters
filtered = frame.fk_filter(v_min=1000.0,
    dx=10.0)
```

Listing 3: F-K Velocity Filtering

The framework handles the forward F-K transform, mask generation based on the velocity thresholds, and the inverse transform transparently.

4.3 Complete Processing Pipeline

A typical real-world workflow involves chaining multiple operations. In this example, we process a raw DAS file to enhance traffic signals:

```
from DASMatrix import df

# 1. Lazy load raw data
frame = df.read("traffic_data.h5")

# 2. Build processing graph
processed = (
    frame
    .detrend()                  # Remove
        DC offset
    .bandpass(5.0, 100.0)       # Isolate
        traffic band
    .fk_filter(v_min=300)       # Remove
        air-waves
    .normalize()                # AGC
)

# 3. Compute and Visualize
# Only now is data loaded and processed
processed.plot_waterfall(
    title="Enhanced Traffic Flow",
    cmap="seismic"
)
```

Listing 4: End-to-End Processing Pipeline

This pipeline demonstrates the power of the fluent API, allowing researchers to express complex signal processing logic in a clear, declarative manner.

This pipeline demonstrates the power of the fluent API, allowing researchers to express complex signal processing logic in a clear, declarative manner.

5 Performance Evaluation

To validate the performance claims, we conducted a benchmark comparing DASMatrix's processing engine against a standard in-memory NumPy/SciPy implementation.

5.1 Benchmark Setup

The benchmark simulated a typical preprocessing workflow consisting of:

1. Linear Detrending (Time axis)
2. Absolute Value Calculation
3. Amplitude Scaling

The test dataset consisted of 50,000 time samples $\times$ 2,000 channels (approx. 400 MB float32 data). The test was performed on a standard workstation CPU.

5.2 Results

The results demonstrate a significant speedup using DASMatrix's graph-based execution engine compared to the eager execution of NumPy:

Table 1: Processing Time Comparison (lower is better)

Implementation	**Time (s)**	**Speedup**
Standard NumPy/SciPy	10.89	1.00x
DASMatrix (Warm Start)	**5.74**	**1.90x**

The **1.90x** speedup is attributed to operation fusion (reducing memory access overhead) and optimized Dask chunking. Furthermore, while the NumPy implementation requires loading the entire dataset into RAM, the DASMatrix implementation can scale to datasets terabytes in size without memory exhaustion, thanks to its out-of-core design.

6 Conclusion

DASMatrix fills a critical gap in the DAS research community by providing a free, open-source, and high-performance tool. By leveraging modern Python data science stack components like Xarray and Dask, it democratizes access to large-scale DAS data analysis, enabling researchers to focus on geophysics rather than data engineering. Future work will focus on integrating deep learning workflows directly into the pipeline.

References

[1] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[2] Stephan Hoyer and Joe Hamman. xarray: N-d labeled arrays and datasets in python. *Journal of Open Research Software*, 5(1), 2017.

[3] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler, 2015.

[4] Nathaniel J Lindsey, Gillan Martin, Douglas S Dreger, Barry Freifeld, Stephen Cole, Stephanie R James, Biondo L Biondi, and Jonathan B Ajo-Franklin. Fiber-optic network observations of earthquake wavefields. *Geophysical Research Letters*, 44(23):11–792, 2017.

[5] Tom Parker, Sergey Shatalin, and Mahmoud Farhadiroushan. Distributed acoustic sensing: a new tool for seismic applications. *First Break*, 32(12), 2014.

[6] Matthew Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*, volume 130, page 126. Citeseer, 2015.