Ensemble of Forecasters on HPC: A Project Report

Tom WALTER, Parsa VARES, Luc CARDOSO

25/11/2024

1 Introduction

This report provides a comprehensive overview of the implementation of an ensemble of forecasters using JAX and HPC (High-Performance Computing) resources. The project involved adapting an initial sequential implementation to work efficiently in an HPC environment by leveraging parallel processing using MPI (`mpi4py`). The purpose of this project was to increase computational efficiency and scale the number of forecasters to provide a robust ensemble prediction.

2 Objective

The primary objective was to adapt the original single-core ensemble forecasting code to run on multiple cores using the HPC resources of the university (Aion). The goal was to:

- Scale the number of forecasters and parallelize their computation.
- Use HPC for distributed execution, ensuring efficient use of resources.
- Compute meaningful statistics from the ensemble forecasts.

3 Modifications to the Original Code

To adapt the original code, several modifications were made to support the parallel execution on HPC.

3.1 Original Code Overview

The original code implemented a simple forecaster using JAX and trained a small neural network to make predictions. It had the following major components:

- **Forecaster Functions**: `forecast_1step()` and `forecast()` for making predictions.
- **Training Loop**: Gradient descent training loop implemented in `training_loop()`.
- **Ensemble Creation**: An ensemble of forecasters with different initial conditions.

3.2 Key Changes Implemented

1. **Parallelization Using MPI**:

   The modified code used `mpi4py` to parallelize the ensemble creation. Each MPI process handled a subset of the forecasters, as shown below:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()  # Rank of the current
    process
size = comm.Get_size()  # Total number of
    processes

# Distribute the work among processes
for i in range(rank, num_forecaster, size):
    key = jax.random.PRNGKey(i)
    W_noise = jax.random.normal(key, W.shape) *
        noise_std
    b_noise = jax.random.normal(key, b.shape) *
        noise_std

    W_init = W + W_noise
    b_init = b + b_noise

    # Train the neural network
    W_trained, b_trained = training_loop(grad,
        num_epochs, W_init, b_init, X, y)

    # Make predictions
    y_predicted = forecast(horizon, X, W_trained,
        b_trained)

    # Append the prediction to the local list
    aggregated_forecasting_local.append(
        y_predicted)
```

2. **Ensemble Scaling**:

The number of forecasters (`num_forecaster`) was increased to 600, distributed across multiple MPI processes. Each process worked on a portion of the forecasters, ensuring that computations were spread evenly.

3. **Statistical Analysis**:

   To better understand the ensemble's performance, the following statistical measures were calculated: mean forecast, median forecast, standard deviation, and the 5th and 95th percentiles.

```
if rank == 0:
    # Flatten the list of forecasts
    aggregated_forecasting_flat = [forecast for
       sublist in aggregated_forecasting for
       forecast in sublist]
    aggregated_forecasting_array = jnp.array(
       aggregated_forecasting_flat)

    # Compute statistics
    mean_forecast = jnp.mean(
       aggregated_forecasting_array, axis=0)
    median_forecast = jnp.median(
       aggregated_forecasting_array, axis=0)
    std_forecast = jnp.std(
       aggregated_forecasting_array, axis=0)
    percentile_5 = jnp.percentile(
       aggregated_forecasting_array, 5, axis=0)
    percentile_95 = jnp.percentile(
       aggregated_forecasting_array, 95, axis=0)
```

4 HPC Implementation

4.1 Environment Setup

To execute the code on the HPC (Aion), the following steps were taken:

1. An interactive session was started using `salloc`:

   ```
   salloc -p interactive --qos=debug --time=2:00:00 -N 1 -n 6
   ```

2. Necessary modules were loaded:

   ```
   module load lang/Python/3.8.6-GCCcore-10.2.0
   module load mpi/OpenMPI/4.0.5-GCC-10.2.0
   ```

3. A virtual environment was set up and required Python packages were installed:

```
python -m venv myenv
source myenv/bin/activate
pip install jax jaxlib mpi4py
```

4.2 Execution of the Script

The script was executed using the following command, distributing the work across 6 MPI processes:

```
mpiexec -n 6 python ensemble_forecaster.py
```

5 Results and Analysis

The statistical analysis of the ensemble forecasts produced the following results (sample):

- **Mean Forecast**:

```
[[0.10913111, 0.6908691], [0.2848961, 0.9719817], ...]
```

- **Median Forecast**:

```
[[0.10908347, 0.6909168], [0.28448576, 0.9719324], ...]
```

- **Standard Deviation**:

```
[[0.00179059, 0.00179059], [0.01087735, 0.0114771], ...]
```

- **5th and 95th Percentiles**:

```
5th: [[0.10639098, 0.6878006], ...]
95th: [[0.11219949, 0.69360924], ...]
```

These results indicate that the ensemble provided stable predictions, with low variance among forecasters, as indicated by the small standard deviation values.

6 Conclusion

In this project, we successfully adapted an ensemble of forecasters to run on HPC resources, leveraging parallelism to efficiently manage a large number of forecasters. By utilizing MPI, we distributed the workload across multiple processes, thereby achieving significant scalability. The statistical analysis provided insights into the model's predictions, demonstrating the effectiveness of the ensemble approach.

7 Code Repository

The complete project code is available on GitHub: `https://github.com/parsavares/ML-ALGO-HPC-Projects`.