Performance analysis is an imperative part of performance tuning during the development of parallel programs. Parallel execution traces enable in-depth analysis of the program’s performance. Current trace analysis tools/workflows have some gaps:

- Most trace analysis tools support different formats and analyses
- GUI-based tools limit data exploration to their graphical views

We have developed Pipit, a Python-based tool, to fill in the gaps in trace-analysis:

- Supports traces in different file formats (OTF2, HPCToolkit, Projections, etc.)
- Provides a uniform data structure in the form of a pandas DataFrame
- Provides a programmatic API to analyze traces
- Provides interactive visual functions to display the traces

Performance Analysis Case Studies Using Pipit

1. Finding the Most Idle Processes
We analyze traces of a Loimos (a Charm++-based epidemiology simulator) execution on 64 processes. We want to find which processes are idling the most while the others are overloaded. Pipit’s "idle_time" function can help us with this task. Then we plot a timeline filtered to the most and least idling processes.

2. Analyzing Overall Performance
Here, we analyze a trace of a Tortuga execution on 64 cores. We use pipit to understand the machine utilization over time for the duration of the run. Pipit’s "time_profile" function provides an overview of the execution’s activity/utilization over time.

3. Pattern Detection
We can use pipit’s "detect_pattern" function to find recurring sets of events in the trace. Below, we analyze a Tortuga execution on 16 cores. The function uses matrix profile to detect patterns in the trace. We use this function to automatically identify loops in the program.

4. Analyzing Communication
Here, we analyze the 32-process executions of Laghos. We can use pipit’s "plot_comm_matrix" and "plot_comm_over_time" functions to examine the communication between ranks and over time respectively.

Background and Pipit’s Structure

Traces: Time series data representing all the events that occur during the program’s execution:
- When functions are entered and exited
- When messages are exchanged between processes
- Different performance metrics (such as hardware performance counters)

How does Pipit store trace data:
- A pandas DataFrame: two-dimensional labeled table-like data structure
- Every trace event is stored as a row in the DataFrame
- DataFrame is sorted by event timestamps

The Calling Context Tree:
- Represents caller-callee relationships between functions
- Stored as a graph in Pipit, and each event in the DataFrame corresponds to some node in the calling context tree

Performance of Pipit
All the experiments in this section were performed on a single node of an HPC cluster with a dual 64-core AMD EPYC 7763 processor.

Getting Started
Install pipit with pip

pip install pipit

Scan the QR code for pipit on GitHub