HPS

https://hps.vi4io.org

Zoya Masih

Trace Analysis in Practice

Outline

Table of Contents

Motivation

- In the previous sessions we introduced:
  - profiling as aggregated statistics
  - tracing as a detailed event timeline
- Score-P combines both ideas in one HPC toolchain
- Vampir is one of the main interfaces to explore Score-P traces

Objectives

Goal of this session

- Understand the workflow
- Run it on the cluster
- Interpret the output

Score-P and Vampir: Big Picture

- **Score-P** collects performance data from HPC applications
- **Vampir** is commonly used for trace analysis

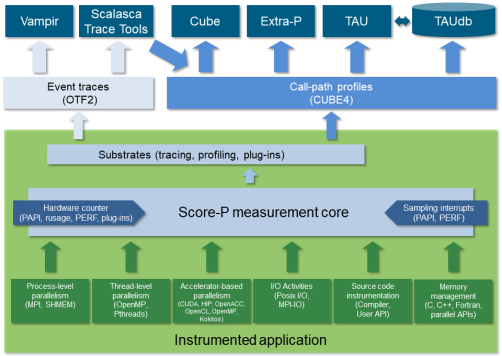


Figure: https://helmholtz.software

Table of Contents

Measurement Procedure

- Instrumentation inserts measurement probes into the code
- Probes are triggered during execution
- Execution records events and performance data
- Data includes time, visits, communication , hardware counters
- Output is stored as profile or trace for analysis

Output Files

- **scorep.cfg:** Listing of used environment variables
- **profile.cubex:** CUBE4 result file of the summary measurement
- **traces.otf2:** the anchor file for the OTF2 (Open Trace Format 2) trace, used to capture and storing performance data. the main entry point for accessing trace data.
- **traces.def:** contains global definitions for the trace data. Includes info about the structure and organization of the trace, like the definitions of events, locations, regions, etc.
- **traces/:** containing per-location trace data. It contains detailed trace info specific to each execution location (e.g., process or thread).

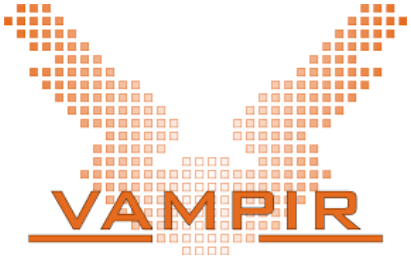
Analyzing Results without GUI Tools

- Score-P produces useful output even without GUI tools
- `scorep-score profile.cubex` provides:
  - time, visits, and region statistics
- Additional processing (advanced):
  - remapping can compute extra metrics and organize data hierarchically

```
Estimated aggregate size of event trace:                   334 bytes
Estimated requirements for largest trace buffer (max_buf): 334 bytes
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       4097kB
(hint: When tracing set SCOREP_TOTAL_MEMORY=4097kB to avoid intermedia
 or reduce requirements using USR regions filters.)

flt     type max_buf[B] visits time[s] time[%] time/visit[us]  region
         ALL        333      9    4.64   100.0      515655.50  ALL
         MPI        240      6    3.33    71.7      554434.59  MPI
     SCOREP         41      1    0.00     0.0          33.63  SCOREP
         USR         26      1    1.31    28.2     1308490.36  USR
         COM         26      1    0.01     0.1        5767.92  COM
```

Table of Contents



How to use Vampir

- Vampir on a Login Node
- Vampir Using VampirServer
- Vampir on Local Machine (limited)

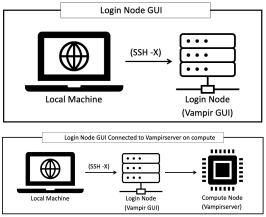
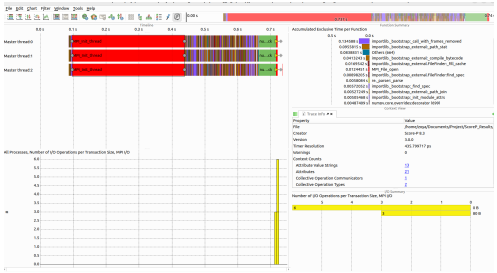


Figure: docs.olcf.ornl.gov

A simple MPI4PY code

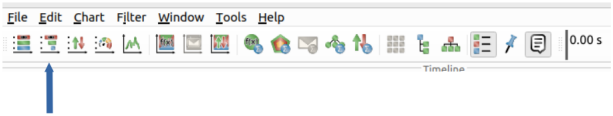
- `srun -n 3 python -m scorep --io=runtime:posix --mpp=mpi script.py`

```
1  from mpi4py import MPI
2  import numpy as np
3  amode =
       ↪ MPI.MODE_WRONLY|MPI.MODE_CREATE
4  comm = MPI.COMM_WORLD
5  fh = MPI.File.Open(comm,
       ↪ "./datafile.contig", amode)
6  buffer = np.empty(10, dtype=np.int)
7  buffer[:] = comm.Get_rank()
8  offset =
       ↪ comm.Get_rank()*buffer.nbytes
9  fh.Write_at_all(offset, buffer)
10 fh.Close()
```



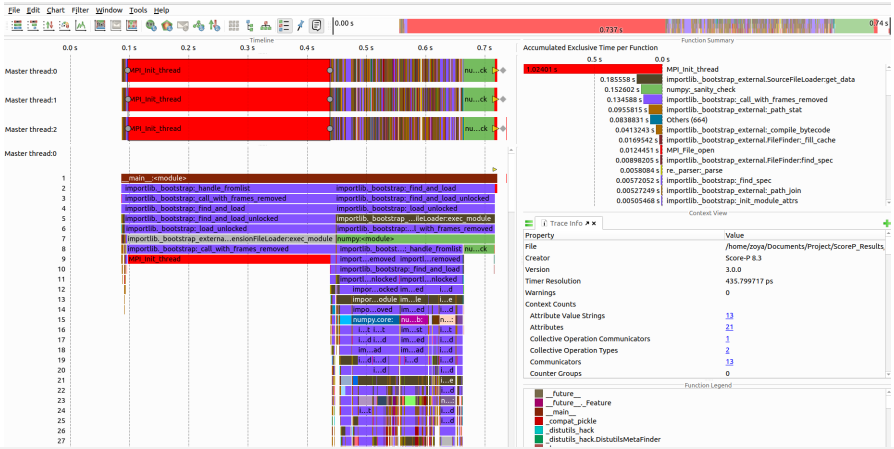
Process Timeline

- Information about different levels of function calls
- Levels show the execution flow and hierarchical relationships



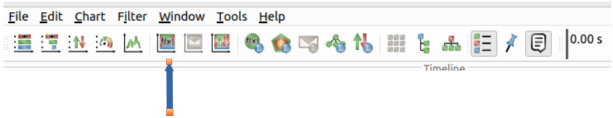
Process Timeline

- Shows when each rank or thread is active, waiting, or inside a specific function



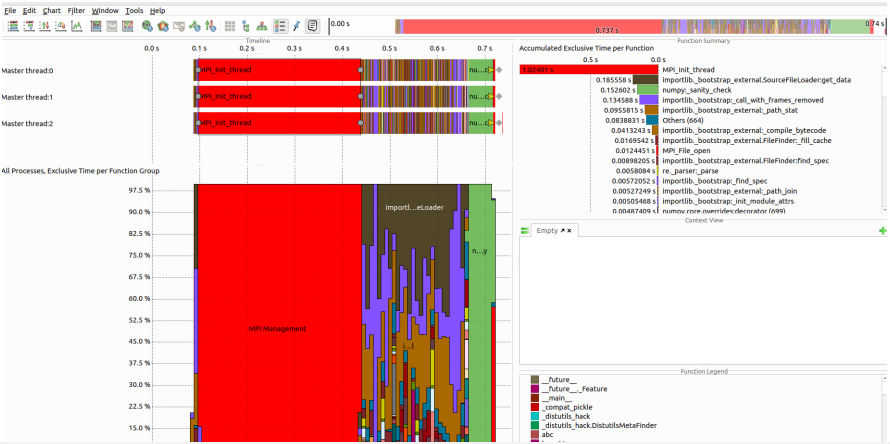
Summary Timeline

- Provides detailed information about functions, communications, and synchronization



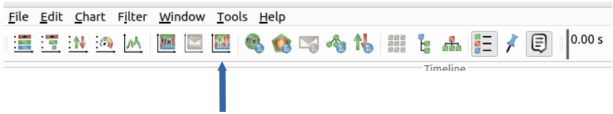
Summary Timeline

- Helps identify dominant phases and whether behavior is balanced across processes



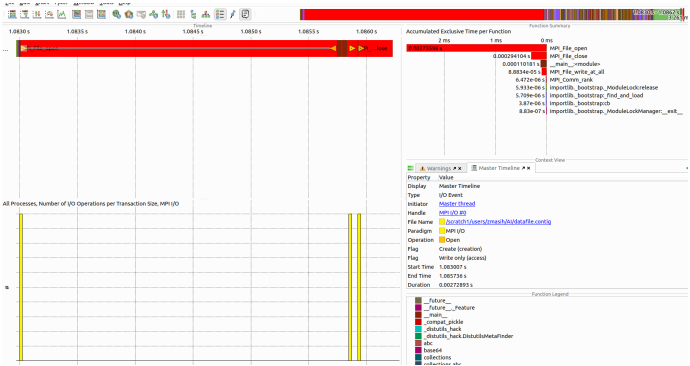
I/O summary

- Vampir highlights I/O operations if I/O performance data has been recorded.
  - `-io=runtime:posix`
- MPI I/O routines are automatically instrumented
  - if the scorep instrumenter detects the MPI programming paradigm



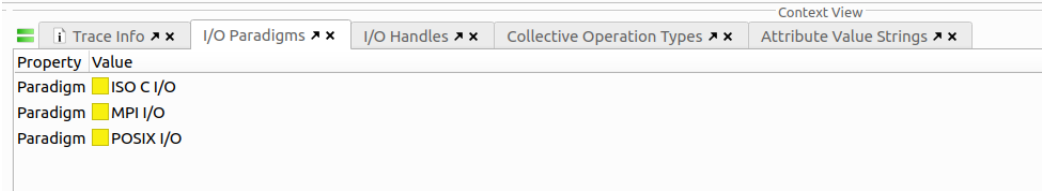
I/O summary

- Useful for checking whether I/O is fragmented into many small operations or concentrated in a few larger ones



I/O summary

- Confirms which I/O interfaces were recorded, such as POSIX or MPI I/O



Hands-on Plan

- **Exercise 1:** Build and trace a small C program
- **Exercise 2:** Analyze the output traces in Vampir
- **Exercise 3:** Compare the results with an optimized version
- **Exercise 4:** Build and trace a small Python program
- **Exercise 5:** Analyze the output traces in Vampir