HPS

https://hps.vi4io.org

Zoya Masih

Introduction to Performance Analysis

Objectives

In this lecture you will learn:

- What performance analysis is and why it matters
- How to approach performance analysis in practice
- Overview of common tools and techniques

Outline

Table of Contents

Recap

What we learned so far

- Performance engineering:
  - understand system behavior
  - improve performance
- Applications are limited by:
  - compute
  - memory
  - I/O
  - network
- Goal: identify bottlenecks and improve efficiency

The Missing Piece

- We know bottlenecks exist
- But we need to answer:
  - Where is time spent?
  - Which function is slow?
  - Are resources well utilized?
- **We need measurements from real executions**

Performance Engineering Cycle

- **Measure** → collect data (today's focus)
- **Analyze** → understand behavior
- **Optimize** → improve performance

Figure: Generated by AI

Performance Analysis Methods

Method	Idea	When to use
Monitoring	Observe system metrics over time	Quick system check
Sampling	Periodic snapshots of execution	Find hotspots
Profiling	Aggregated runtime statistics	Identify expensive functions
Tracing	Detailed timeline of events	Analyze parallel behavior

Key: Different methods provide different levels of detail and overhead.

Table of Contents

Monitoring

Idea

- Continuous observation of system behavior
- Tracks key metrics over time:
  - CPU, GPU, memory usage
  - I/O activity
  - network throughput

Characteristics

- low overhead
- system-wide view
- long time scale

Monitoring: Everyday Systems

We already use monitoring

- CPU and memory usage
- Disk and I/O activity
- GPU utilization (`nvidia-smi`)

Typical tools

- `top` , `htop`
- Task Manager (Windows)
- Activity Monitor (macOS)

Monitoring = observing system behavior over time

Monitoring in HPC Systems

Same idea at larger scale

Tools

- `vmstat` , `iostat`
- `sacct` , `sstat`
- dashboards (e.g. Grafana)

Example: Grafana

- visualizes metrics across nodes
- CPU, memory, I/O, network
- real-time + historical trends

Monitoring scales from a single machine to an entire cluster

Table of Contents

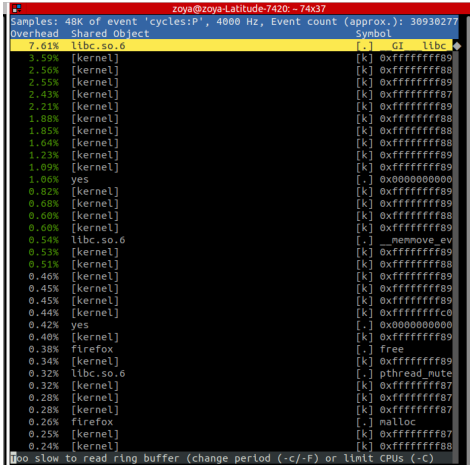
Sampling: How It Works

Example: Live output of `perf top`

- Row = function or code region
- % = frequency of samples

How it works

- CPU interrupted periodically
- Current execution point recorded
- Repeated many times



More samples ⇒ more time spent there

Sampling in Practice

Single machine

- `perf`
- `gprof`
- flame graphs

Use case

- identify hotspots
- understand CPU usage

HPC systems

- **HPCToolkit**
- **Intel VTune**
- **LIKWID**
- **perf**

Same principle → larger scale applications

Table of Contents

Profiling

Idea

- Collect aggregated runtime statistics
- Measure:
  - time per function
  - number of calls
  - memory usage

Characteristics

- more precise than sampling
- moderate overhead
- summary output (no timeline)

Profiling in Practice

Single machine

- `gprof`
- `perf stat`

HPC tools

- **Score-P (profiling mode)**
- **HPCToolkit**
- **TAU**

Output

- function-level statistics
- MPI time, I/O time
- call counts

Key idea

Summarized view of execution

Profiling: Strengths and Limitations

Advantages

- clear function-level breakdown
- easy to interpret
- low to moderate overhead

Best for

- identifying hotspots
- first optimization step

Limitations

- no temporal behavior
- cannot show:
  - communication delays
  - synchronization issues

Key idea

What is slow, not when or why

From Profiling to Tracing

Profiling

- aggregated view
- function-level statistics
- no timeline

Tracing

- event timeline
- execution order
- reveals waiting and communication

Key question:

Why is this function slow?

$\Rightarrow$ Profiling is not enough $\rightarrow$ need tracing

Table of Contents

Tracing

Idea

- Record detailed sequence of events over time
- Capture:
  - function calls
  - communication (MPI)
  - I/O operations

Characteristics

- very detailed
- full execution timeline
- reveals communication and synchronization

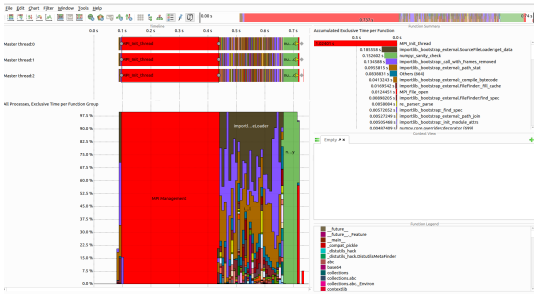
Tracing: Timeline View

What tracing provides

- event timeline of execution
- ordering of operations
- interaction between processes

Key difference

- Profiling → summary
- Tracing → timeline



When and how things happen

Tracing Workflow

Step 1: Data collection

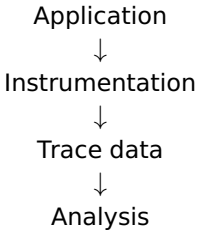
- Instrument the application
- Record events during execution

Step 2: Trace generation

- Events stored as a timeline
- Includes computation, communication, I/O

Step 3: Analysis

- Explore the timeline
- Identify patterns and bottlenecks



Tracing: Strengths and Limitations

Advantages

- full execution timeline
- detailed behavior
- essential for parallel programs

Best for

- debugging performance issues
- understanding communication

Limitations

- high overhead
- large trace files
- complex analysis

Key idea

Most detailed, but most expensive

Table of Contents

Why Tools Differ

- Performance tools are not interchangeable
- They differ in:
  - how data is collected
  - what is measured
  - how metrics are defined
- Tools are designed for different analysis goals

Why Profiling Results Differ

Even for the same application, different tools can report different results

Reasons

- Different instrumentation points (which functions are measured)
- Different grouping of operations
- Different levels of detail (coarse vs fine-grained)

Profiling results depend on how the tool measures the program

Example: Tool Design Differences

Aspect	**Darshan / Recorder**	**Score-P**
Instrumentation	Runtime-interception (LD_PRELOAD)	Compile-time instrumentation
Output	Binary logs	Profiles (Cube), Traces (OTF2)
Focus	I/O behavior	Full application (CPU, MPI, I/O)
Detail	Lightweight / medium detail	High-resolution traces
Analysis	Darshan-util, Recorder-Viz	Vampir, Cube, Scalasca

Different design choices $\Rightarrow$ different results

Why Metrics Are Not Comparable

Different tools can define metrics differently

Reasons

- Different sets of intercepted functions
- Different grouping (e.g., POSIX vs STDIO)

Impact

- I/O time depends on included functions
- Bandwidth depends on recorded data
- IOPS depends on definition

Same metric name $\neq$ same meaning

Example: I/O Time is Ambiguous

I/O time is not consistently defined

- depends on:
  - included functions (e.g., MPI_File_*)
  - metadata handling
  - overlapping operations

Example

- Score-P includes more MPI-IO calls
- Darshan may exclude metadata

Interpret metrics in the context of the tool

Table of Contents

Summary

Performance analysis methods

- Monitoring → system behavior over time
- Sampling → statistical view of execution
- Profiling → aggregated function-level statistics
- Tracing → detailed timeline of execution

Key insights

- Start simple → go deeper when needed
- Tools are not interchangeable
- Metrics depend on how they are measured

Performance analysis = measurement + interpretation