Enabling HPC software productivity with the TAU performance system

Jean-Baptiste BESNARD
<jbbesnard@paratools.fr>

PDC Summer School August 2023, KTH, Stockholm, Sweden.

Outline

Two main parts, theory and then practice.

- **9:00 - 10:00: Enabling HPC software productivity with the TAU performance system**
  - A discussion on the role of performance tools in HPC
  - General outline of monitoring techniques and principles
  - A practical tour of the TAU performance system (features / how to)
- **10:30 - 11:45: Hands-on Session**
  - Deploy TAU
  - Profile real code
  - Look at various instrumentation techniques

Enabling HPC software productivity with the TAU performance system

Introduction

On Performance in…

High
Performance
Computing

Introduction

What is performance in the end ?

- **#1** Solving challenging problems with computing resources:
  - Domain specific knowledge (physicists, numericians, …)
  - Produce invaluable simulation results by solving **large** problems (requiring distributed memory and scale)
- **#2** Taking advantage of the latest hardware:
  - Using it **productively** to fulfill **#1** in a tight loop
  - Consider new techniques and models thanks to computing power (3D, ab-initio, higher-resolutions, …)

Introduction

Mind the gap

- **HPC is subject to the knowledge gap**
  - It is a field full of experts in their various domains
  - It is crucial to acknowledge that cross-disciplinary collaboration is needed
- **A numerician is not (always) a computer scientist:**
  - Mastering simulation (or other domain specific applications) and low-level programming (runtimes, architectures) is increasingly difficult
  - HPC is suffering from lack of abstractions between the computational layers (compute and runtime)

Introduction

On the surprising nature of HPC

- **HPC is a field of constant challenges…**
  - Cutting edge hardware every 2 years
  - Evolving runtime abstractions
- **And surprising stability …**
  - Large (multimillions lines) applications to port
  - A lot of Fortran, C and C++
  - Mostly (up to recently) MPI and OpenMP
    - MPI (mostly) backward compatible for 25+ years !

BUT

Things may change …

LLM

Large Language Models

ML now drives High-End Computing

Both fields are now bound to converge

- **GPUs are the core vector for LLMs**
  - Multi-Billion dollar market (for sure larger than HPC).
- **AI/ML payloads come with different programming habits**
  - Multi-language (not only native ones) -> PyTorch (Python + CUDA)
  - Generally easy to deploy (pip, cargo, docker)
  - Cloud-aware
- **And HPC will transitively benefit from those**


Interesting times ahead…

Making a Productive HPC Application

Choose and dose your priorities

- **Fulfilling the initial goal of the program**
  - Generate valuable domain-specific results
  - Achieve the desired resolution/throughput
- **Efficient use of the machine**
  - Do not waste computing power (vectorized, GPUs, CPUs, …)
  - Run **efficiently** at scale (do you really need these cores ?)
- **Maintainable code**
  - Keep the code easy to write and to read
  - Document the code avoiding cryptic parts
  - Write tests (for both performance and functionalities)

Optimizing too much is a mistake

AKA Leave AVX512 to the compiler as ..

Runtimes and Hardware are less uniform

The challenge is now impacting applications (not only runtimes)

- **Up to recently parallelism was « simple » (MPI and OpenMP, namely MPI+X)**
  - Using MPI for distributed memory
  - OpenMP for shared-memory programming
- **GPUs added another layer (+X+Y)**
  - Need to coordinate differentiated local memory regions
  - Need to handle more complex program states (nested computation)
- **And the world is less uniform…**
  - Keep the code easy to write and to read
  - Cuda, HIP, OpenACC, OpenMP (+Targets)
  - With little abstractions: Kokkos (C++)

Consider Following Agile Methods

Prepare for change and cope with it

- **You will have a broader set of languages to choose..**
  - C, Fortran, C++
  - Python (+ GPU)
  - Rust, Julia
- **A runtime..**
  - For distributed memory (in HPC MPI is dominating), other such as Mercury are appearing
  - For shared-memory (OpenMP, Threads, task oriented languages (async))
  - For your GPU (OpenMP Targets, Intel, AMD, Nvidia, …)
- **It means ..**
  - Rewriting code is now a necessary process when retargeting
  - The value of the program is not only in the code (also in algorithms, file-formats, doc, toolchain)
  - We are entering a multi-language era for HPC
  - **You need to be ready to assess the performance and operation of your code in many configurations**

MPI Forum is working on an ABI

Facilitating (among others) the use of MPI with other languages

MPI Application Binary Interface Standardization

Jeff R. Hammond
NVIDIA Helsinki Oy
Helsinki, Finland
jeff_hammond@acm.org

Lisandro Dalcin
Extreme Computing Research Center
KAUST
Thuwal, Saudi Arabia
dalcinl@gmail.com

Erik Schnetter
Perimeter Institute for Theoretical Physics
Waterloo, Ontario, Canada
eschnetter@perimeterinstitute.ca

Marc Pérache
CEA DAM
Arpajon, France
marc.perache@cea.fr

Jean-Baptiste Besnard
ParaTools
Bruyères-le-Châtel, France
jbbesnard@paratools.fr

Jed Brown
University of Colorado Boulder
Boulder, Colorado, USA
jed@jedbrown.org

Gonzalo Brito Gadeschi
NVIDIA GmbH
Munich, Germany
gonzalob@nvidia.com

Joseph Schuchart
University of Tennessee, Knoxville
Knoxville, Tennessee, USA
schuchart@utk.edu

Simon Byrne
California Institute of Technology
Pasadena, California, USA
simonbyrne@caltech.edu

Hui Zhou
Argonne National Laboratory
Lemont, Illinois, USA
zhouh@anl.gov

To be presented in September at EuroMPI 23.

Not Optimizing is also a mistake

You need to have performance as a fitting function.

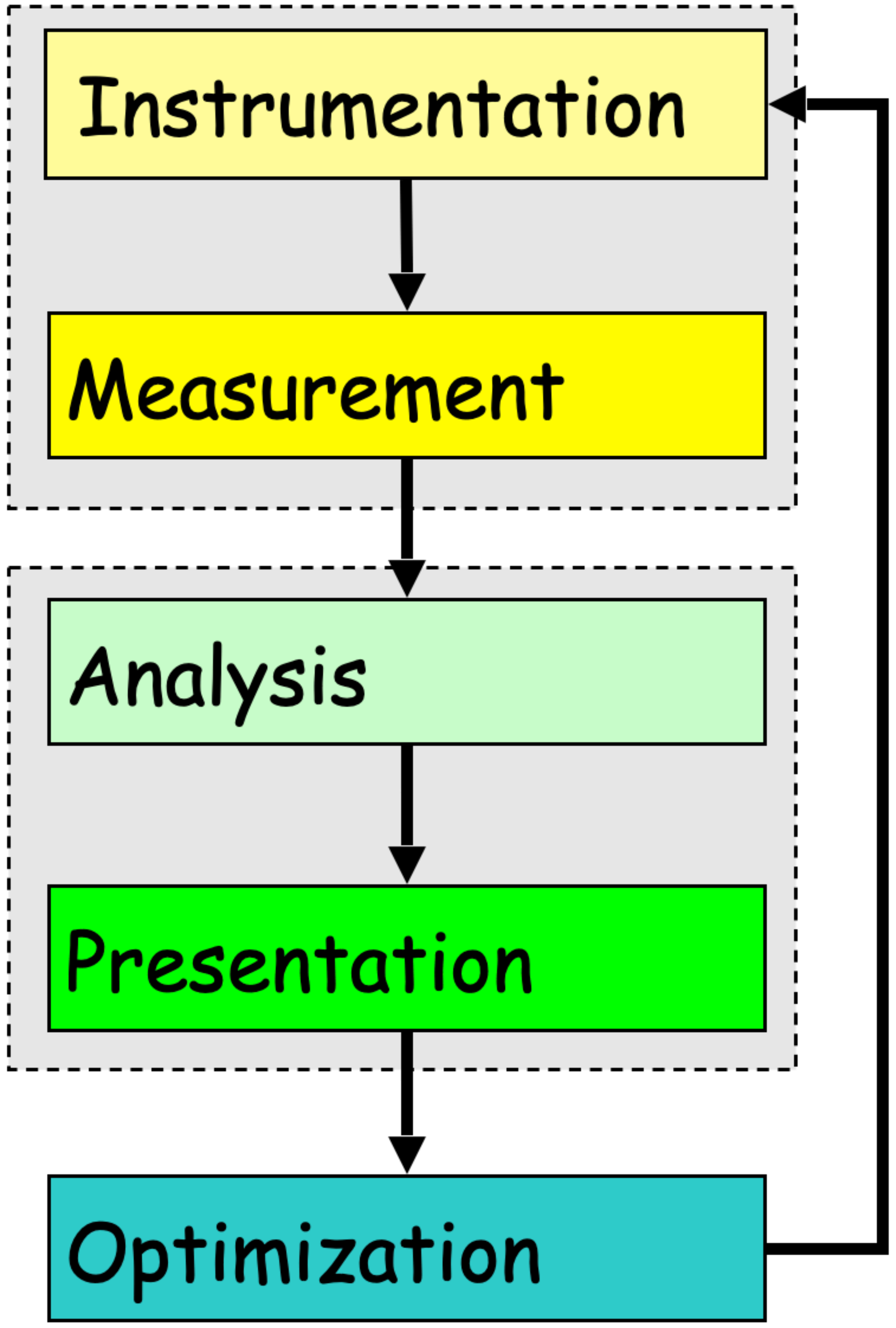
Performance Awareness

Make sure to track performance

- **Embed simple performance measurement metrics in your codes**
  - Simple counters testifying of the throughput
  - They should be domain specific and ideally language agnostic
- **Make your program modular**
  - Allows cross-language comparison
  - Forces separation of concerns
  - Enables plug & play replacement of some parts (for comparison)
- **Include performance measurement in your processes**
  - Automate in the CI (performance regression)
  - Include performance assessment in releases

Performance Optimization Cycle

- Expose factors
- Collect performance data
- Calculate metrics
- Analyze results
- Visualize results
- Identify problems
- Tune performance



TAU

Tuning and Analysis Utility

Performance Optimization Cycle

- **Tuning and Analysis Utilities (20+ year project)**
- **Comprehensive performance profiling and tracing**
  - Integrated, scalable, flexible, portable
  - Targets all parallel programming/execution paradigms
- **Integrated performance toolkit**
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- **Integrates with application frameworks**

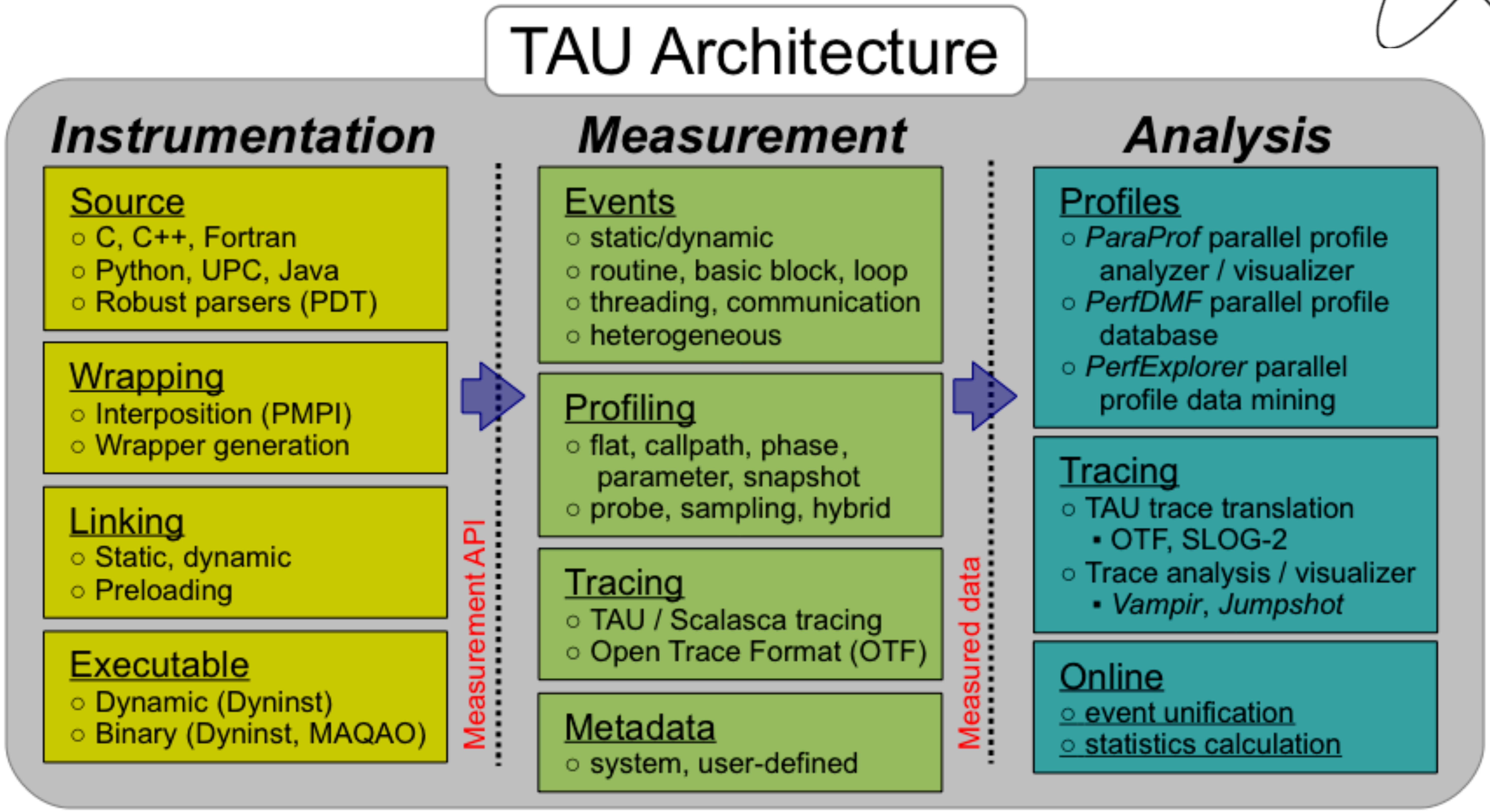
UNIVERSITY OF OREGON

ParaTools

Understanding Application Performance using TAU

- How much time is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*?
- How many instructions are executed in these code regions?
  Floating point, Level 1 and 2 *data cache misses*, hits, branches taken?
- What is the memory usage of the code? When and where is memory allocated/de-allocated? Are there any memory leaks?
- What are the I/O characteristics of the code? What is the peak read and write *bandwidth* of individual calls, total volume?
- What is the contribution of each *phase* of the program? What is the time wasted/spent waiting for collectives, and I/O operations in Initialization, Computation, I/O phases?
- How does the application *scale*? What is the efficiency, runtime breakdown of performance across different core counts?

TAU Architecture and Workflow



TAU Architecture and Workflow

Instrumentation: Add probes to perform measurements
- Source code instrumentation using pre-processors and compiler scripts
- Wrapping external libraries (I/O, MPI, Memory, CUDA, OpenCL, pthread)
- Rewriting the binary executable

Measurement: Profiling or tracing using various metrics
- Direct instrumentation (Interval events measure exclusive or inclusive duration)
- Indirect instrumentation (Sampling measures statement level contribution)
- Throttling and runtime control of low-level events that execute frequently
- Per-thread storage of performance data
- Interface with external packages (e.g. PAPI hw performance counter library)

Analysis: Visualization of profiles and traces
- 3D visualization of profile data in paraprof or perfexplorer tools
- Trace conversion & display in external visualizers (Vampir, Jumpshot, ParaVer)