Score-P
• Score-P is a joint instrumentation and measurement system for a number of PA tools.
  – Provide typical functionality for HPC performance tools
  – Support all fundamental concepts of partner’s tools

• Instrumentation (various methods)
• Flexible measurement without re-compilation:
  – Basic and advanced profile generation
  – Event trace recording
  – Online access to profiling data

• MPI/SHMEM, OpenMP/Pthreads, and hybrid parallelism (and serial)
• Enhanced functionality (OpenMP 3.0, CUDA, highly scalable I/O)
Score-P Architecture

Vampir  Scalasca  CUBE  TAU  TAUdb  Periscope

Event traces (OTF2)
Call-path profiles (CUBE4, TAU)

Hardware counter (PAPI, rusage)
Online interface

Score-P measurement infrastructure

Instrumentation wrapper

Process-level parallelism (MPI, SHMEM)
Thread-level parallelism (OpenMP, Pthreads)
Accelerator-based parallelism (CUDA)
Source code instrumentation
User instrumentation
CUBE
• Parallel program analysis report exploration tools
  – GUI for interactive analysis exploration
  – Algebra utilities for report processing
• Originally developed as part of Scalasca toolset
• Now available as a separate component
  – Can be installed independently of Score-P, e.g., on laptop or desktop
  – Latest release: CUBE 4.2.3 (June 2014)
Analysis presentation and exploration

- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call path (program location)
  - System location (process/thread)

- Three coupled tree browsers

- CUBE displays severities
  - As value: for precise comparison
  - As colour: for easy identification of hotspots
  - Inclusive value when closed & exclusive value when expanded
Analysis presentation

What kind of performance metric?
Where is it in the source code? In what context?
How is it distributed across the processes/threads?
Inclusive vs. Exclusive values

- **Inclusive**
  - Information of all sub-elements aggregated into single value

- **Exclusive**
  - Information cannot be subdivided further

```c
int foo()
{
    int a;
    a = 1 + 1;
    bar();
    a = a + 1;
    return a;
}
```
SCALASCA
Automatic trace analysis

• Idea
  – Automatic search for patterns of inefficient behavior
  – Classification of behavior & quantification of significance
  – Guaranteed to cover the entire event trace
  – Quicker than manual/visual trace analysis
  – Parallel replay analysis exploits available memory & processors to deliver scalability of the analysis
Scalasca workflow

Measurement library -> Optimized measurement configuration -> Summary report

Instr. target application -> Local event traces

Instrumenter compiler / linker

Source modules

HWC

Optimized measurement configuration

Local event traces

Wait-state report

Parallel wait-state search

Report manipulation

Scalasca trace analysis

Which problem?

Where in the program?

Which process?

Instrumented executable

Measurement library

Instr. target application

Instrumenter compiler / linker

Source modules

Measurement library

HWC

Instrumented executable

Optimized measurement configuration

Summary report

Local event traces

Wait-state report

Parallel wait-state search

Report manipulation

Source modules
Example: Late Sender

- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication
Example: Late Broadcast

- Waiting times if the destination processes of a collective 1-to-N operation enter the operation earlier than the source process (root)
- Applies to: MPI_Bcast, MPI_Scatter, MPI_Scatterv
Post-processed trace analysis report

Additional trace-based metrics in metric hierarchy
Vampir
Mission

- Visualization of dynamics of complex parallel processes
- Full details for arbitrary temporal and spatial levels
- Supplement to automatic analysis
- Client-server architecture

Typical questions that Vampir helps to answer:

- What happens in my application execution during a given time in a given process or thread?
- How do the communication patterns of my application execute on a real system?
- Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?
The Main Displays of Vampir

• **Timeline Charts:**
  - Master Timeline
  - Process Timeline
  - Counter Data Timeline
  - Performance Radar

• **Summary Charts:**
  - Function Summary
  - Message Summary
  - Process Summary
  - Communication Matrix View

Show application activities and communication along a time axis

Provide quantitative results for the currently selected time interval
Vampir: Visualization of the NPB-MZ-MPI / BT trace

- Master Timeline
- Navigation Toolbar
- Function Summary
- Function Legend
Vampir: Visualization of the NPB-MZ-MPI / BT trace

**Process Timeline**

Detailed information about different levels of function calls in a stacked bar chart for an individual process.
Vampir: Visualization of the NPB-MZ-MPI / BT trace

Counter Data Timeline

Detailed counter information over time for an individual process.
Performance Radar

Vampir: Visualization of the NPB-MZ-MPI / BT trace

Detailed counter information over time for a collection of processes.
Zoom in: Initialization Phase

Context View: Detailed information about function “initialize_”.
Vampir: Visualization of the NPB-MZ-MPI / BT trace

Process Summary

Function Summary: Overview of the accumulated information across all functions and for a collection of processes.

Process Summary: Overview of the accumulated information across all functions and for every process independently.