Node-Level Performance Analysis: Using Vampir

Jack Ogaja
jack.ogaja@gwdg.de

7. April 2025

Learning Objectives

- To help develop ideas on how to use performance tools to explore the optimization space of widely used computational kernels in common computer architectures.

Node-Level Performance Analysis

- Performance Modelling
- Performance Measurements

Performance Analysis with VAMPIR

- Trace view

VAMPIR: Interactive Session

Node-Level Performance Analysis

Node-Level Performance Analysis

- Modelling: Derivation of a model based on the functionality and topology of interconnected elements of a computational unit of a specific architecture.
- Measurements: Collection of events data through program instrumentation and events sampling.
- Visualization: Usage of performance tools to visualize collected events' data and traces.

Node-Level Performance Analysis
Modelling

Performance models are important in application's performance engineering and analysis. Models are key for:

- Comparing application performance against the machine capabilities
- Evaluating the optimality of application
- Identify possible bottlenecks in application computational performance
- Identifying software and hardware limitations

Node-Level Performance Analysis

Measurements: Machine and Application Characterizations

1. Data Collection and Sampling
   - Automatic instrumentation - increases overhead, e.g. Compilers, Vampir, Score-P,
   - Manual instrumentation. e.g. Print-statements, Score-P
   - Binary instrumentation - requires re-addressing, replacements and patching of instructions and memory accesses, e.g. Gprof, Valgrind, GDB
   - Sampling - execution is itnerupted at regular intervals to sample addresses of executed instruction, e.g. LIKWID, Gprof
2. Data Processing
   - For simple applications with small amount of events, events can be counted and performance data can be processed and displayed in a graphical viewer in real-time.
3. Data transfer and storage
   - For complex applications, events data should be stored in disks. e.g. Vampir

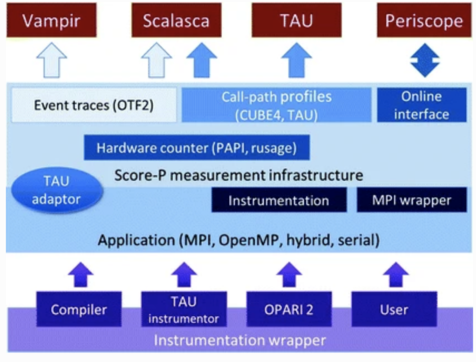
Performance Analysis with VAMPIR

Performance Analysis with VAMPIR

Visualization and **A**nalysis of **MPI** **R**esources - VAMPIR

1. Is a collection of tools for analysing performance of parrallel applications
   - instrumentation, measurement (e.g. Score-P) and visualization tools
2. Complex but powerful:
   - Performance analysis framework for parallel programs
   - Graphical representation of performance data -> enables detailed understanding of dynamic processes on massively parallel program.
   - in-depth event based analysis of parallel run-time behavior and inter-processor communications.
   - Helps identify performance bottleneck

VAMPIR Tool-suite Architecture



- Includes instrumentation, measurement (e.g. Score-P) and visualization tools, which give the user an insight into the dynamic run-time behaviour of their applications.
- Offers the capability of visualization of time ordered events, e.g. MPI, OpenMP, perfomance counters, events from manual instrumentation.

Key features

- Powerful zooming and scrolling in all displays
- Adaptive statistics for user selected time ranges
- Filtering of processes, functions, messages, collective operations
- Hierarchical grouping of threads, processes, and nodes
- Support of source code locations
- Integrated snapshot and printing for publishing
- Customizable displays
- Server:
  - For distributed performance data visualization
  - Highly scalable
  - Remote visualization of Performance data.

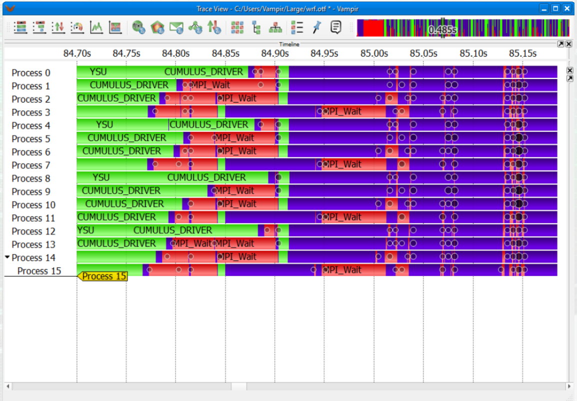
Performance Data

- Attach a working monitoring system to the program e.g. Score-P or VampirTrace(not developed anymore!)
- Score-P provides new OTF2 data format for trace data generation and CUBE4 for profiling data format.

  ```
  scorep mpicc app.c -o app
  ```

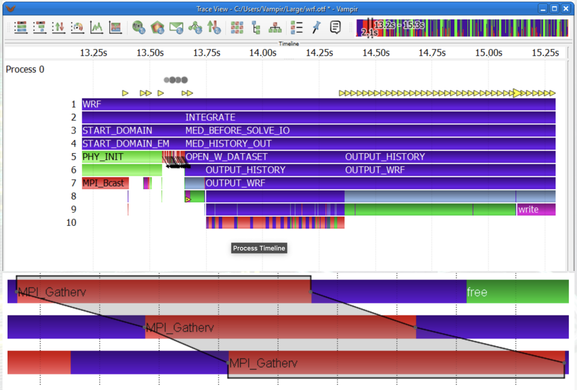
Trace view

Master timeline and Functions summary.



- Shows detailed information about functions, communication, and synchronization events
- `Process Timeline` shows different levels of function calls

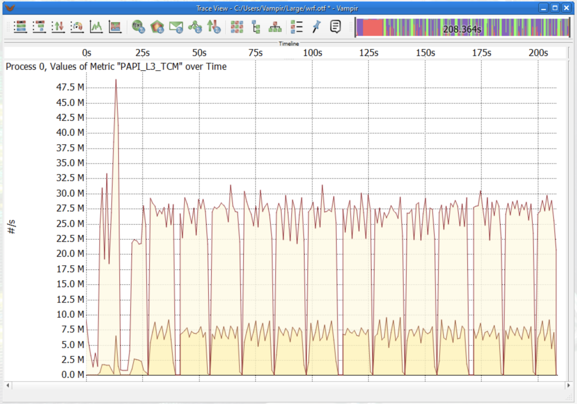
Trace view
Process Timeline



Selected MPI Collective in Master Timeline

- The chart's timeline is divided into levels, which represent the different call stack levels of function calls
- Messages exchanged between two different processes are depicted as black lines. In timeline charts, the progress in time is reproduced from left to right.

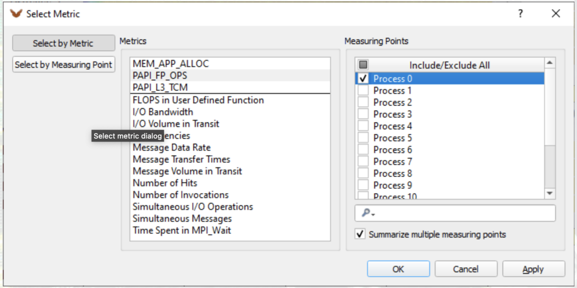
Trace view
Counter Data Timeline



- Counters are values collected over time to count certain events e.g. floating point operations (FLOPS) or cache misses (L3_TCM).
- Counters values can contain hardware performance counters, or a arbitrary sample values and statistical information like number of function calls or an iterative approximation of the final results.

Trace view

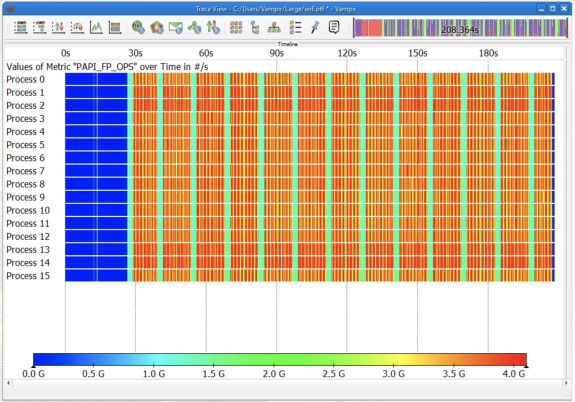
Counter Data Selection (Dialogue)



- The Counter chart is restricted to one counter at a time. It shows the selected counter for one measuring point (e.g., process)
- The actual measured data points can be displayed in the chart by enabling them via the context menu under Options....

Performance

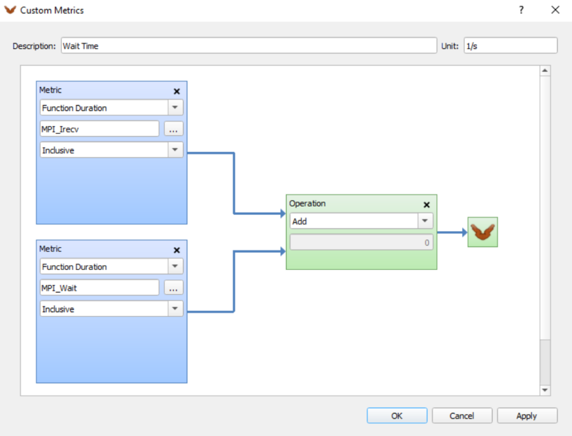
Performance Radar



- Unlike “Counter Data Timeline”, Performance Radar shows one counter for all processes at once, and provides a possibility to create custom metrics.
- The performance data overlay can also be used to identify functions with a certain amount of allocated memory

Performance

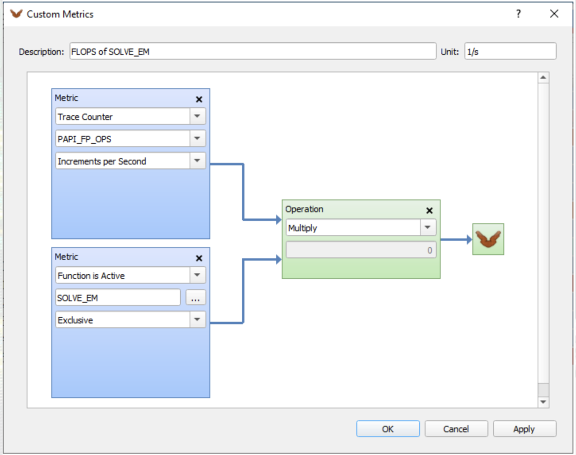
Customized Performance Metrics: Wait time



- The Custom Metrics Editor allows to derive own metrics based on existing counters and functions. This is particularly useful as the performance data overlay of the Master Timeline, is capable of displaying the own metrics.
- Custom metrics can be exported and imported in order to use them in multiple trace files.

Performance

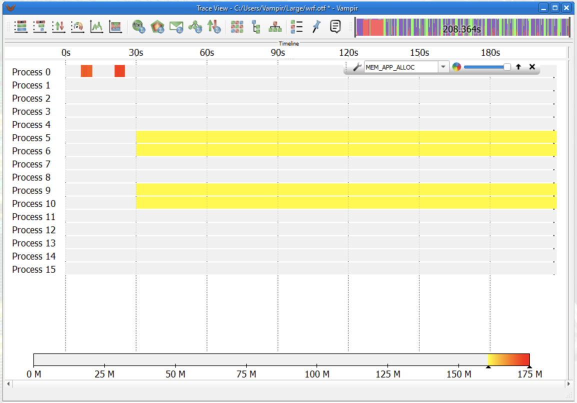
Customized Performance Metrics: FLOPS (per function)



- Custom metrics are build from input metrics that are linked together using a set of available operations.

Performance

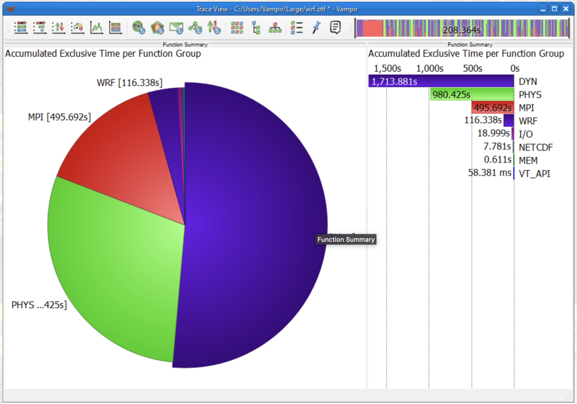
Performance Data Overlay: Memory Allocation



Functions with 160MB - 175MB allocated memory

- The performance data overlay can also be used to identify functions with a certain amount of allocated memory.

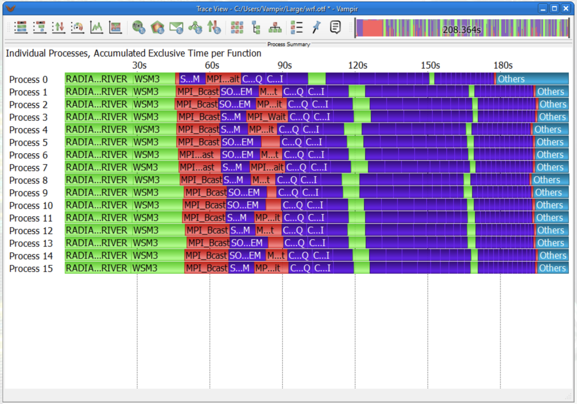
Statistics
Function Summary



Function Summary

- The Function Summary can be shown as Histogram (a bar chart, like in timeline charts) or as Pie Chart.
- *Inclusive* means the amount of time spent in a function and all of its subroutines. *Exclusive* means the amount of time spent in just this function.

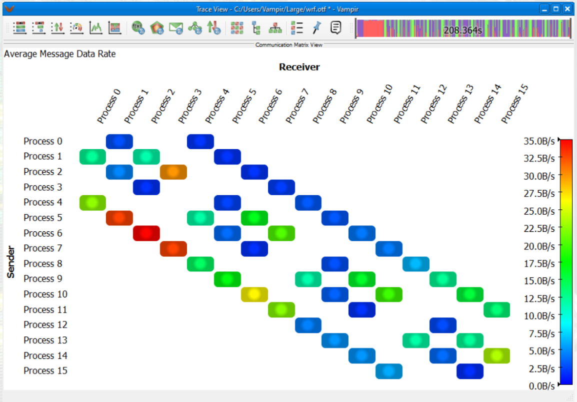
Statistics
Process Summary



Process Summary

- Shows the information for every process independently
- Is useful for analyzing the balance between processes to reveal bottlenecks.

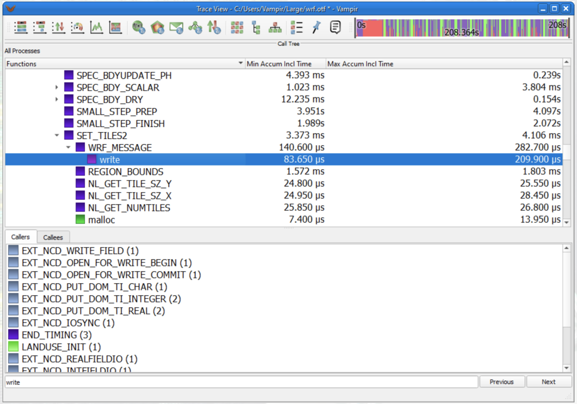
Communication
Communication Matrix



- One way of analyzing communication imbalances by showing information about messages sent between processes.
- **Caution:** A high duration is not automatically caused by a slow communication path between two processes.

Call Tree

Call Tree



- This illustrates the invocation hierarchy of all monitored functions in a tree representation.
- It reveals information about the number of invocations of a given function, the time spent in the different calls and the caller-callee relationship.

VAMPIR: Interactive Session

VAMPIR: Interactive Session

- Time Line Charts,
- Group Processes, Process Timeline,
- Communication Events,
- Performance Counter Data Overlays e.g. High and Low FLOP rates
- Statistical Charts
- Communication Matrix View
- Call Tree

Simple Access to Vampir - client only

GWDG
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

- login to SCC:
  `ssh -X user@login-mdc.hpc.gwdg.de`
- Set the environment:
  `module load vampir`
  `module load scorep`
  OR `module load vampirtrace`
- Start Vampir Client:
  `vampir [trace file]`