

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.

1



Performance Modelling

Performance Measurements

Performance Analysis with VAMPIR

Trace view

VAMPIR: Interactive Session





- 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

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

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

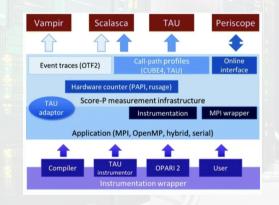
Performance Analysis with VAMPIR Visualization and Analysis of MPI Resources - VAMPIR



- 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

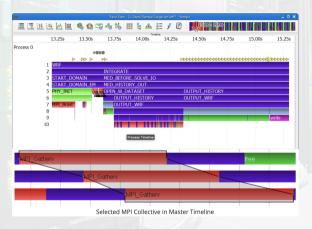
Trace view Master timeline and Functions summary.

84	4.70s 84	.75s 8	4.80s	84.85s	84.90s	84.95s	85.00s	85.05s	85.10s 85.15s	
Process 0	YSU		TIMUTU	S DRIVER	00 0			0 0	0000	
Process 0		S DRIVER								
Process 2	CUMULUS			MPI_W	lait		0 0			
Process 3		0				OMPL V				_
Process 4	YSU	_		JS DRIVER		1 1000	60			
Process 5	CUMULU	S DRIVER		MPI_Wa			6			_
Process 6	CUMULUS	DRIVER	00	MPI_W			0 0	0 00		
Process 7		0	0.0			OMPL_V	Vait			
Process 8	YSU		CUMULU	IS_DRIVER			••	0 0		
Process 9	CUMULUS	DRIVER		OMPI_Wa	ait 🕐		þ	0 0		
Process 10	CUMULUS	_DRIVER	0	MPI_W	/ait		þ 💽	စ် စ		
Process 11	CUMULUS_	DRIVER	0.0			OMPI_V	Vait	o (
Process 12	YSU	CUMULU	JS_DRIV	ER	0			0 00		
Process 13	CUMULUS_	DRIVER	MPI_W	alt MPI_Wa	ait 🕴			0 ¢0		
Process 14			• •	MPI_W	/ait		P 💽	<u>ao</u> ao		
Process 15	Process 15	9	0.0			o MPI_W	Vait 🔷 🔶	• • • •	0 000	
	100000									



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

Trace view Process 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.

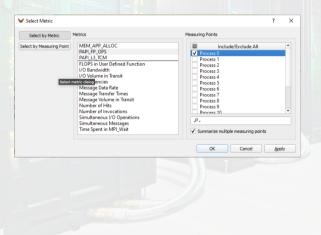
Trace view Counter Data Timeline





- 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.

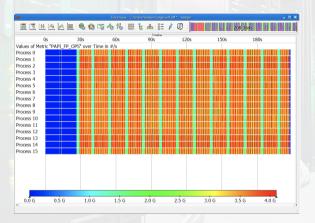
Trace view Counter Data Selection (Dialogue)





- 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

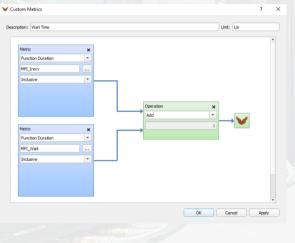




- Unlike "Counter Data Timeline", Performance Radar shows one counter for all processes at once, and provides a possibility to create custom metrics.
- The performance data overlay can also be used to identify functions with a certain amount of allocated memory

Performance

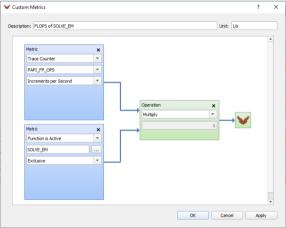
Customized Performance Metrics: Wait time





- 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)

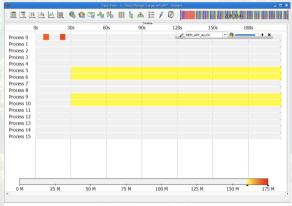




• Custom metrics are build from input metrics that are linked together using a set of available operations.

Performance

Performance Data Overlay: Memory Allocation

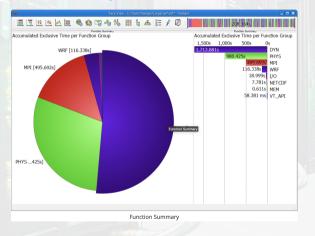


Functions with 160MB - 175MB allocated memory



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

Statistics Function Summary





- 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.

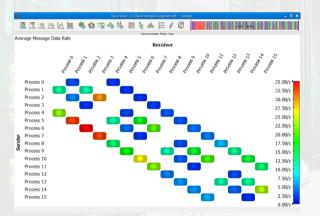
Statistics Process Summary

	cesses, Accumul	30s	50 for 60s	runcuon	90s		120s	150s	180s	
rocess 0	RADIARIVER	WSM3	SM MPI	ait CO	CI				Others	-
rocess 1	RADIARIVER	WSM3	MPI_Bcast	SOEM M	t CQ	CI				Others
rocess 2	RADIARIVER	WSM3	MPI_Bcast	OEM MF	It CQ	CI				Others
rocess 3	RADIARIVER	WSM3	MPI_Bcast	5M MPI_	Walt CC	CI				Others
rocess 4	RADIARIVER	WSM3	MPI_Bcast	5M MP	It CQ C	I				Others
rocess 5	RADIARIVER	WSM3	MPI_Bcast	SOEM	CQ (II				Others
rocess 6	RADIARIVER	WSM3	MPIast	SOEM 🛛 🕅	t CQ	CI				Others
rocess 7	RADIARIVER	WSM3	MPIașt							Others
rocess 8	RADIARIVER	WSM3	MPI_Bcast							Others
rocess 9	RADIARIVER	WSM3	MPI_Bcas	SOEM	CQ C	I				Others
rocess 10	RADIARIVER	WSM3	MPI_Bcas	SOEM	CQ	CI				Others
rocess 11	RADIARIVER	WSM3		SM MP						Others
rocess 12	RADIARIVER	WSM3		t SM M.						Others
rocess 13	RADIARIVER			st SOEM						Others
rocess 14	RADIARIVER			SOEM						Others
rocess 15	RADIARIVER	WSM3	MPI_Bcas	t SM MP	lt CC	2 CI				Others



- 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

EE ET EN EQ IA IEL 🔍 🏫 🖙 🐴 🐪	. ## % <u>& != /</u> (P)	0s 208 364t 208
	Call Tree	208.3645
Processes	Carina	
unctions	Min Accum Incl Time Max A	ccum Incl Time
SPEC_BDYUPDATE_PH	4.393 ms	0.239s
SPEC_BDY_SCALAR	1.023 ms	3.804 ms
SPEC_BDY_DRY	12.235 ms	0.154s
SMALL_STEP_PREP	3.951s	4.097s
SMALL_STEP_FINISH	1.989s	2.072s
SET_TILES2	3.373 ms	4.106 ms
WRF_MESSAGE	140.600 µs	282.700 µs
write	83.650 µs	209.900 μs
REGION_BOUNDS	1.572 ms	1.803 ms
NL_GET_TILE_SZ_Y	24.800 μs 24.950 μs	25.550 µs 28.450 µs
NL_GET_TILE_SZ_X	24.950 µs 25.850 µs	28.450 µs 26.800 µs
malloc	25.850 µs 7.400 µs	13.950 µs
	7.400 µs	12.300 hs
Calers Calees		
EXT_NCD_VentT_FIELD () EXT_NCD_VENT_FOR_WATT_BEGN (1) EXT_NCD_VENT_FOR_WATT_BEGN (1) EXT_NCD_VENT_FOR_WATT_BEGN (1) EXT_NCD_VENT_DOW_TL_VENT_(1) EXT_NCD_VENT_DOW_TL_VENT_(2) EXT_NCD_VENT_DOW_TL_VENT_(2) EXT_NCD_VENT_DOW_TL_VENT_(2) EXT_NCD_VENT_DOW_TL_VENT_(2) EXT_NCD_VENT_(1) DAVDEE_INT (1) DAVDEE_INT (1) DAVDEE_INT (1) DAVDEE_INT (1)		
rite		Previous Next



- 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



• 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]