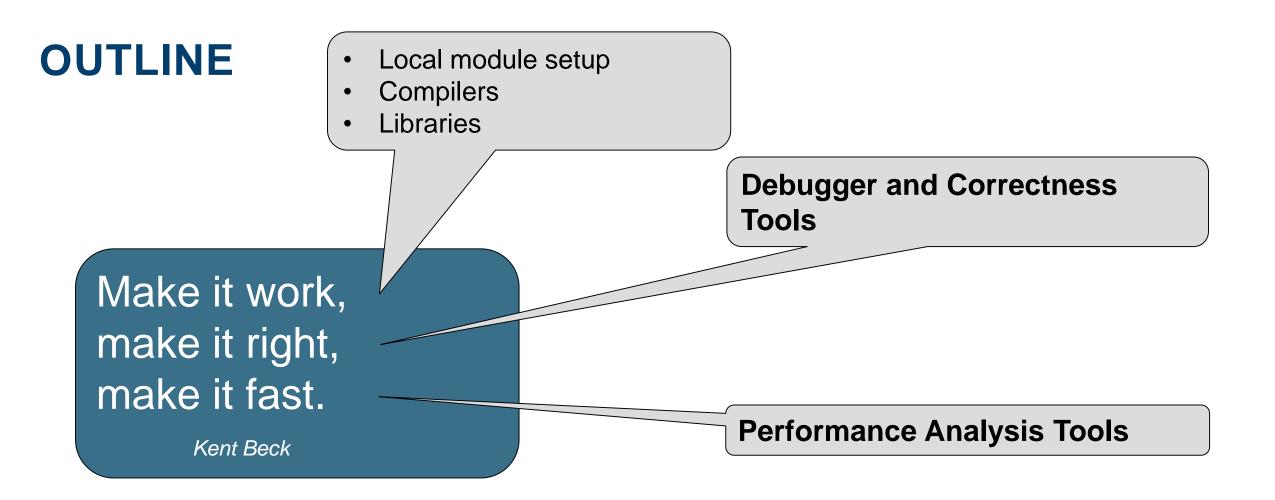


HPC SOFTWARE – DEBUGGER AND PERFORMANCE ANALYSIS TOOLS

MAY 23, 2024 I MICHAEL KNOBLOCH I M.KNOBLOCH@FZ-JUELICH.DE



Mitglied der Helmholtz-Gemeinschaft



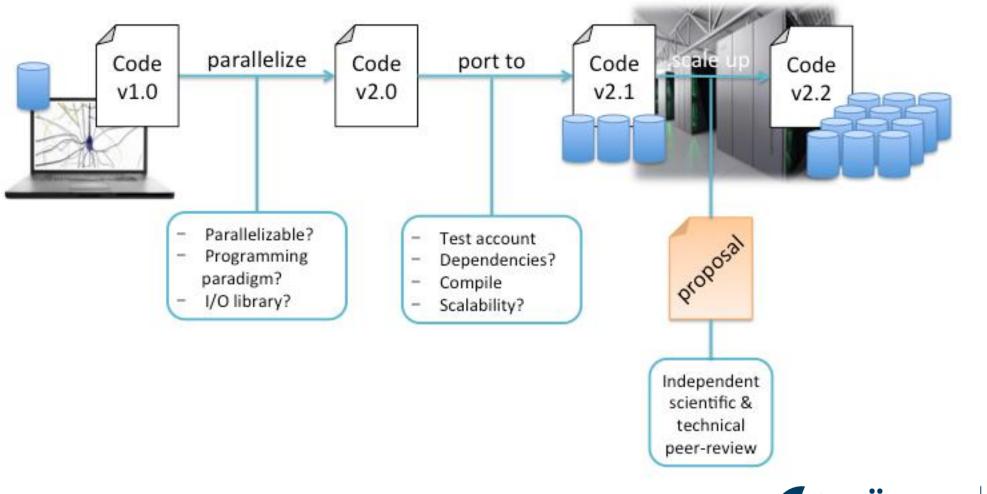


WHY SHOULD YOU CARE ABOUT TOOLS?





NEW APPLICATION?





WORKING WITH LEGACY CODES?





VETERAN HPC USER, BUT NEW TO JSC?



• Assess performance on a JSC machine



 Compare behavior on different machines



Investigate scaling behavior

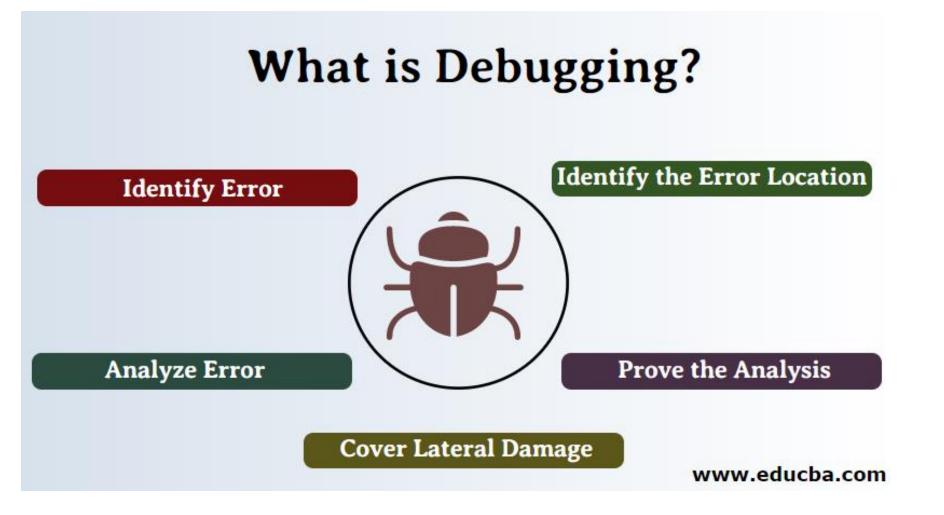


			100			12 Mar 10			
1 2 3			Name	Туре	Value	681 682	<pre>// Target nodes const char** c_target_oper_names, int ntargets,</pre>	œ	tensorflow::FunctionLibraryRunti
Thread State	TID		_ ●_	int	0x00000000000	693 694	<pre>TF_Buffer* run_metadata, TF_Status* status) { TF_Run_Setup(noutputs, c_outputs, status);</pre>	œ	tensorflow::DirectSession::GetOr
Breakpoint	1.1	tensorflow::SoftmaxXentWitl	nstar	int	0x0000006 (6)	605	<pre>std::yector<std::pairstensorflow::string, tensor="">> input_pairs(ni if (!TF_Run_Inputs(c_inputs, &input_pairs, status)) return;</std::pairstensorflow::string,></pre>	œ	std:: Function handler <tensorflo< td=""></tensorflo<>
Stopped	1.2	pthread_cond_wait	grap	. int	0x00000015 (21)	697	<pre>for (int i = 0; i < ninputs; ++i) {</pre>	0	stanander <tensorito< td=""></tensorito<>
Stopped	1.3	pthread cond wait	[Add			608 609	<pre>input_pairs[i].first = c_input_names[i]; }</pre>	œ	std::function <tensorflow::status (<="" td=""></tensorflow::status>
						610 611	<pre>std::vector<tensorflow::string> output_names(noutputs); for (int i = 0; i < noutputs; ++i) {</tensorflow::string></pre>	œ	tensorflow::\$unnamed_namespa
Stopped	1.4	pthread_cond_wait	-1			612	<pre>output_names[i] = c_output_names[i];</pre>	œ	tensorflow::NewLocalExecutor
						614 615	<pre>std::vector<tensorflow::string> target_oper_names(ntargets); for (int i = 0; i < ntargets; ++i) {</tensorflow::string></pre>	œ	tensorflow::DirectSession::GetOr
Select process or thread attributes to group by:						616	<pre>616 target_oper_names[i] = c_target_oper_names[i]; 617 }</pre>		tensorflow::DirectSession::Run
						618 619	TF_Run_Helper(s->session, nullptr, run_options, input_pairs, outp c_outputs, target_oper_names, run_metadata, status)	œ	TF_Run_Helper
						620 621	3	œ	TF_Run
						622 623	<pre>void TF_PRunSetup(TF_DeprecatedSession* s,</pre>	œ	tensorflow::TF_Run_wrapper_hel
				624 625	<pre>const char** c_input_names, int ninputs, // Output names</pre>	œ	tensorflow::TF Run wrapper		
				626 627	<pre>const char** c_output_names, int noutputs, // Target nodes</pre>	Ø	_run_fn		
				628 629	<pre>const char** c_target_oper_names, int ntargets, const char** handle, TF_Status* status) {</pre>	•	ext do call		
Action Points X Command Line X				630 631	<pre>status->status = Status::OK();</pre>				
						632	<pre>std::vector<tensorflow::string> input_names(ninputs);</tensorflow::string></pre>	C D	_do_call
ID❤	Туре	Stop Location		ı	Line	633 634	<pre>std::vector<tensorflow::string> output_names(noutputs); std::vector<tensorflow::string> target_oper_names(ntargets);</tensorflow::string></tensorflow::string></pre>	C7	_do_run

DEBUGGER & CORRECTNESS TOOLS



WHAT IS DEBUGGING?





Mitglied der Helmholtz-Gemeinschaft

DEBUGGING TOOLS (STATUS: MAY 2024)

- Debugger:
 - CUDA-GDB
 - TotalView
 - LinaroForge DDT
- Memory Analyzer:
 - Intel Inspector
 - Archer
- Correctness Checker:
 - MUST



I DON'T KNOW WHERE YOU ARE, I DON'T KNOW HOW YOU WORK, BUT I WILL FIND YOU AND I WILL FIX YOU



CUDA-GDB



- Part of the CUDA toolkit
- Extension to gdb
- CLI and GUI (Nsight)
- Simultaneously debug on the CPU and multiple GPUs
- Use conditional breakpoints or break automatically on every kernel launch
- Examine variables, read/write memory and registers
- Inspect GPU state when the application is suspended
- Identify memory access violations

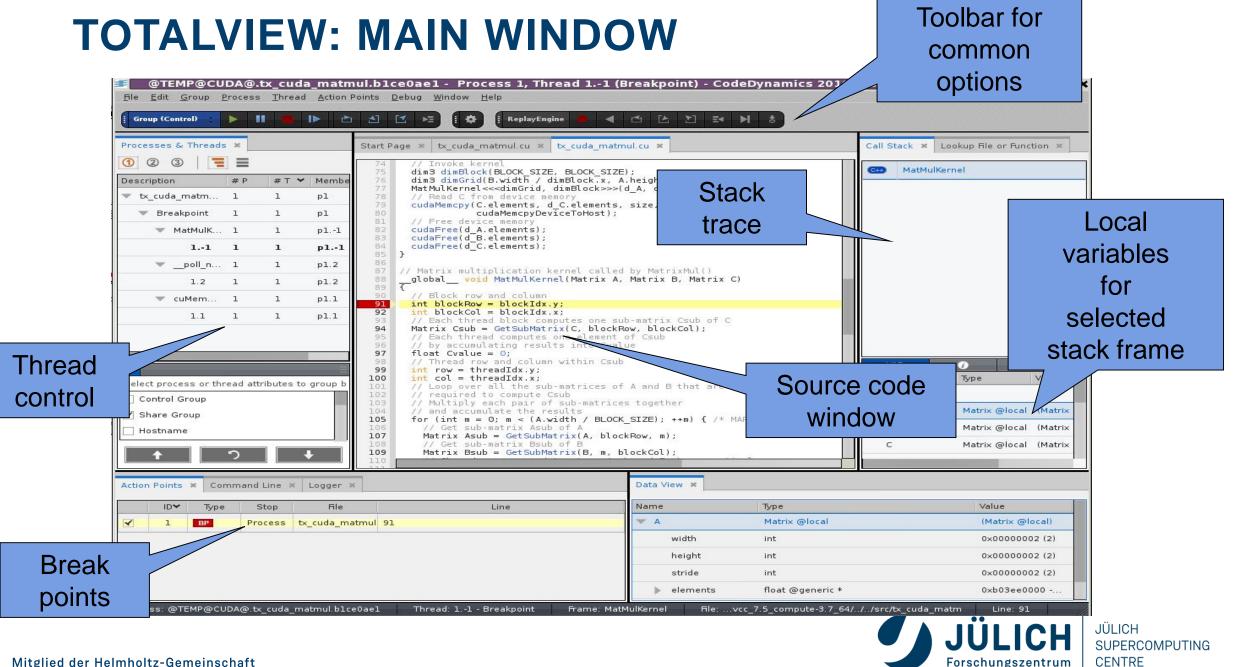
 vectorAdd {0} [device: gk110 (0)] (Breakpoint) CUDA Thread (0,0,0) Block (0,0,0) 	6				
,				* 🗆 🖻	<u>4</u>
			CUDA Information		~
CUDA Thread (1,0,0) Block (0,0,0)		Search o	JUDA Information	1	
🔻 🚳 All CUDA Threads	▼ 🎲 (0,0,0)	SM 11	256 threads of 256 are	run
🔻 🎡 Block (0,0,0) [sm: 11]		(0,0,0)	Warp 0 Lane 0 🖻 vectorAdd.cu:36 (0x9		
CUDA Thread (0,0,0) [warp: 0 lane: 0] (vectorAdd.cu:36)	S S S S S S S S S S S S S S S S S S S	(1,0,0)	Warp 0 Lane 1	🖻 vectorAdd.cu:36 (0x9a	653
vectorAdd.cu 🛛	- 8		🖩 Registers 😂		
32 VECTOTAUN CONST FLOAT TA, CONST FLOAT TD, FLOAT TC, I		Name	T(0,0,0)B(0,0,	,0) T(1,0,0)B(0,0,0)	
<pre>33 { 34 int i = blockDim.x * blockIdx.x + threadIdx.x;</pre>		lili R5	4	4	
35		1111 R6	3149824	3149824	
36 if (i < numElements) 37 {		1919 R7	4	4	
C[i] = A[i] + B[i];		lill R8	0	1	
39 }		1919 R9	0	1	
40 } 41		1919 R10	1060608	-271911904	
	•	1919 R11	0	2	
Console 🕱 🖉 Tasks 🗷 Problems 📀 Executables 🔋 Memory					-
ectorAdd [C/C++ Application] gdb traces					
x400300800"},{name="C",value="0x400301000"},{name="numElem				torAd\	
<pre>.cu",fullname="/home/eostroukhov/cuda-workspace/vectorAdd/ 70.340 (adb)</pre>	/src/vector	Add.cu",line	="36"}		
70,340 (gdb) 70,340 157^done,register-values=[{number="15",value="0x0"}	<u>}1</u>				



TOTALVIEW

- UNIX Symbolic Debugger for C/C++, Fortran, mixed Python/C++, PGI HPF, assembler programs
- JSC's "standard" debugger
- Advanced features
 - Multi-process and multi-threaded
 - Multi-dimensional array data visualization
 - Support for parallel debugging (MPI: automatic attach, message queues, OpenMP, Pthreads)
 - Scripting and batch debugging
 - Advanced memory debugging
 - Reverse debugging
 - CUDA and OpenACC support
 - Remote debugging
- NOTE: JSC license limited to 2048 processes (shared between all users)



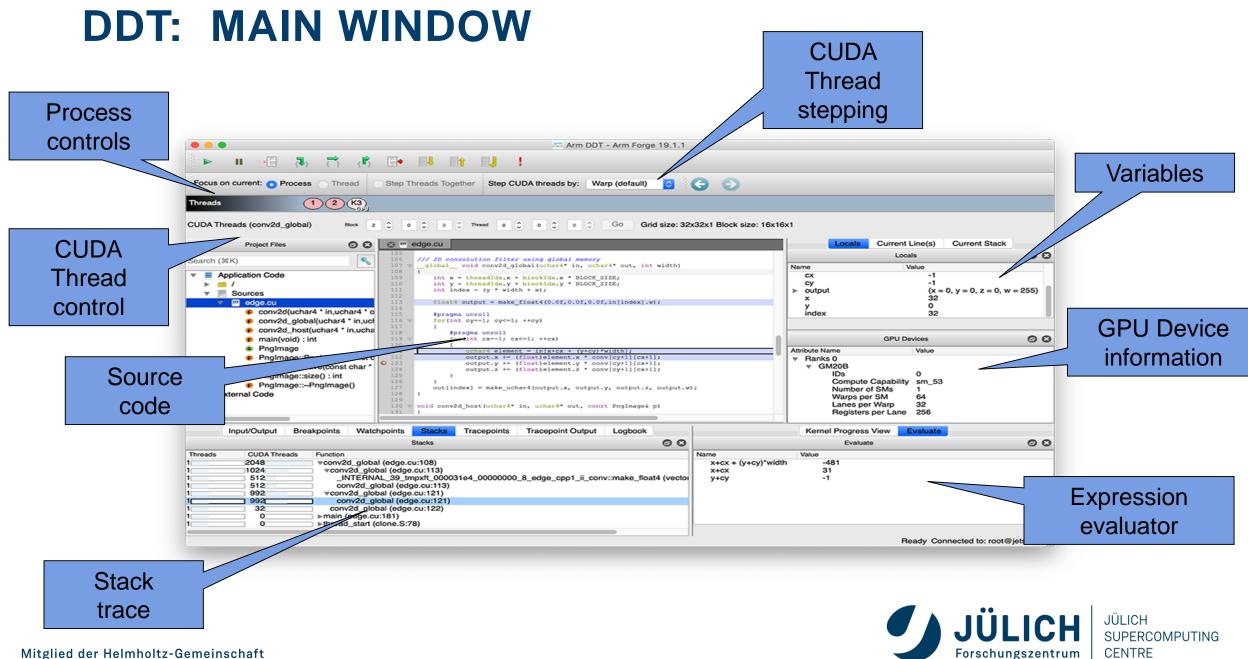


Forschungszentrum

LINARO FORGE - DDT

- UNIX Graphical Debugger for C/C++, Fortran, and Python programs
- Modern, easy-to-use debugger
- Advanced features
 - Multi-process and multi-threaded
 - Multi-dimesional array data visualization
 - Support for MPI parallel debugging (automatic attach, message queues)
 - Support for OpenMP (Version 2.x and later)
 - Support for CUDA and OpenACC
 - Job submission from within debugger
- https://linaroforge.com/linaroDdt
- NOTE: JSC license limited to 128 processes (shared between all users)

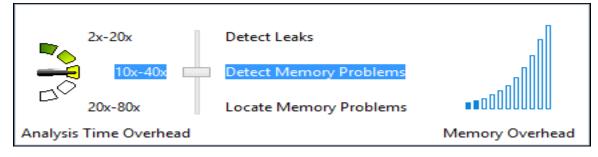




Mitglied der Helmholtz-Gemeinschaft

INTEL INSPECTOR

- Detects memory and threading errors
 - Memory leaks, corruption and illegal accesses
 - Data races and deadlocks
- Dynamic instrumentation requiring no recompilation
- Supports C/C++ and Fortran as well as third party libraries
- Multi-level analysis to adjust overhead and analysis capabilities
- API to limit analysis range to eliminate false positives and speed-up analysis





INTEL INSPECTOR: GUI

r000ti2 ×								*
🗾 Dete	ct Deadlocks and Data Races					INTEL	INSPECTOR	
a 🕒 Targe	t 📫 Analysis Type 🖪 Collection Log 📗	Summary					-// 8	
Problems					S.	Filters	Sort + - V	
1D	Type Sources	Modules	·	State		Severity	-	
EP1 0	Data race find_and_fix_threading_errors.cp		CONTRACTOR AND A DESCRIPTION OF A DESCRI			Error	3 item(s)	
IEP2 Q	Data race blocked_range.h; parallel_for.h; p Data race winvideo.h		threading_errors.exe threading_errors.exe			Туре		
uers •	Data race winnigeo.n	ring_ang_ru	unreading_endisiere	1 THEN		Data race	3 item(s)	
						Source blocked_range.h	1 item(s)	
						find_and_fix_threading_errors.		
						parallel_for.h	T item(s)	
						partitioner.h	1 item(s)	
						task.h task_scheduler_init.h	1 item(s) 1 item(s)	
						winvideo.h	1 item(s)	
						Hodule		
						find and fix threadinn errors.	exe Sitem(s) *	2
9.1	1 of 12 P Al	Code Locations: Data race	11.000000		S Time	eline	5	
Description	Source	Function	Module		Variable 🔿			
Write	find_and_fo_threadi primary.scene = &scene	ng_errors.cpp:105 render_one_pixel	find_and_fix_thread 1 and fix thread1		College	sad_video (2108)		
103	primary, scene = ascene,		and IIA onreadi	Au Caldie	Ackeptender_el			
105	col-trace(sprimary); //Threading		De De	tect N	Memory Proble	ems		Intel Inspector XE 2015
106	<pre>//2 ways to fix this threading et // 1) Make col a local var</pre>							
Write	find_and_fix_threads	ng_errors.cpp:105 render_one_pixel	🙃 🗹 🕀 Ta	rget 📝	Analysis Type	🍐 Collection Log 📗 🤇	Summary	D
103	primary.scene = sscene;	fins						9
104	col-trace(sprimary); //Threading	find	Troblems					ы
106	//2 ways to fix this threading er	TOT	ID	•	Туре		Sources	State
107	// 1) Make col a local var		. c	0	Mismatched al	location/deallocation	find and fix	_memory_errors Re New
1102	hronization allocation site task_scheduler_init. thread stack size 1= TBB USE		15 TI	8	Invalid memory	101	and the second se	memory errors 🔊 New
199910			and the second second	Ā		the second se	the second s	A CONTRACTOR OF
			● P3		Memory growt			find_and_fix_me P Not fixed
					Memory growt	h	[Unknown];	find_and_fix_me P Confirmed
			⊲ <u>1</u> 0			1 of 2 ▷ All C	ode Location	ns: Invalid memory access
			Descriptio	n Sou	rce	Function Mod	dule	Object Size Offset
			Write	find	and fix memory e	rror operator() find	and fix mer	The second s
			164			operatory find		d and fix memory errors.exe
			165		for lungions	d int i=0;i<=(mbox		d and fix memory errors.exe
			166			ox[i]=0; //Memory	the second se	d and fix memory errors.exe
			167		IOCAL RD	ox[1]-0; //Hemory		
					For line .	- heatingly and -	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	d_and_fix_memory_errors.exe
			168		for (inc A =	r.begin(); y != 1	cond con	_debug.dll!local_wait_for_a
ied der He	elmholtz-Gemeinschaft						FO	rschungszentrum GENTRE

Forschungszentrum | CENTRE



- Data race detector for large OpenMP programs
- Combination of static and dynamic techniques
 - Low runtime and memory overhead
 - Still high accuracy and precision
- Now part of LLVM
- Compile with -fsanitize=thread
- Can be used with GCC, but CLANG OpenMP runtime must be linked
- Creates output in text format



ARCHER EXAMPLE

RNING: ThreadSanitizer: data race (pid=2234)

Vrite of size 4 at 0x7fff81d209d0 by thread T1

- #0 .omp_outlined._debug_ /p/project/training2410/knobloch1/archer_test.c:11:10 (a.out+0xd1efb)
- #1 .omp_outlined. /p/project/training2410/knobloch1/archer_test.c:9:1 (a.out+0xd1f85)
- #2 __kmp_invoke_microtask <null> (libomp.so+0xb8782)
- #3 main /p/project/training2410/knobloch1/archer_test.c:9:1 (a.out+0xd1d55)

Previous read of size 4 at 0x7fff81d209d0 by main thread:

- #0 .omp_outlined._debug_ /p/project/training2410/knobloch1/archer_test.c:11:12 (a.out+0xd1ed6)
- #1 .omp_outlined. /p/project/training2410/knobloch1/archer_test.c:9:1 (a.out+0xd1f85)
- #2 __kmp_invoke_microtask <null> (libomp.so+0xb8782)
- #3 main /p/project/training2410/knobloch1/archer_test.c:9:1 (a.out+0xd1d55)

Location is stack of main thread.

Location is global '??' at 0x7fff81d03000 ([stack]+0x1d9d0)

Thread T1 (tid=2237, running) created by main thread at:

#0 pthread_create /dev/shm/swmanage/jusuf/Clang/16.0.6/GCCcore-12.3.0/llvm-project-16.0.6.src/compiler-rt/lib/tsan/rtl/tsan_interceptors_posix.cpp:1048:3 (a.out+0x2678b)
#1 __kmp_create_worker <null> (libomp.so+0x97676)

SUMMARY: ThreadSanitizer: data race /p/project/training2410/knobloch1/archer_test.c:11:10 in .omp_outlined._debug__

JÜLICH SUPERCOMPUTING CENTRE

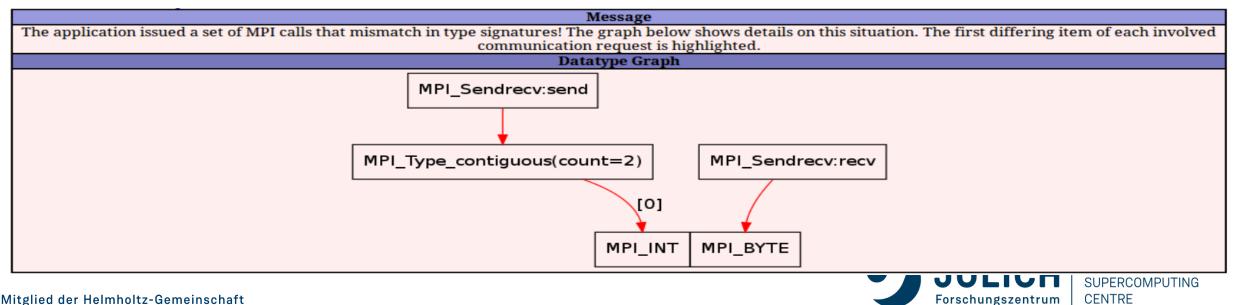
MUST

- Next generation MPI correctness and portability checker
- https://www.i12.rwth-aachen.de/go/id/nrbe
- MUST reports
 - Errors: violations of the MPI-standard
 - Warnings: unusual behavior or possible problems
 - Notes: harmless but remarkable behavior
 - Potential deadlock detection
- Usage
 - Compile with debug information (i.e. use the -g flag)
 - Run application under the control of mustrun (requires (at least) one additional MPI process)
 - E.g. on JUSUF: mustrun --must:mpiexec srun --must:np -n -n 4 ./app
 - Open output html report (might need to copy it to your local machine)

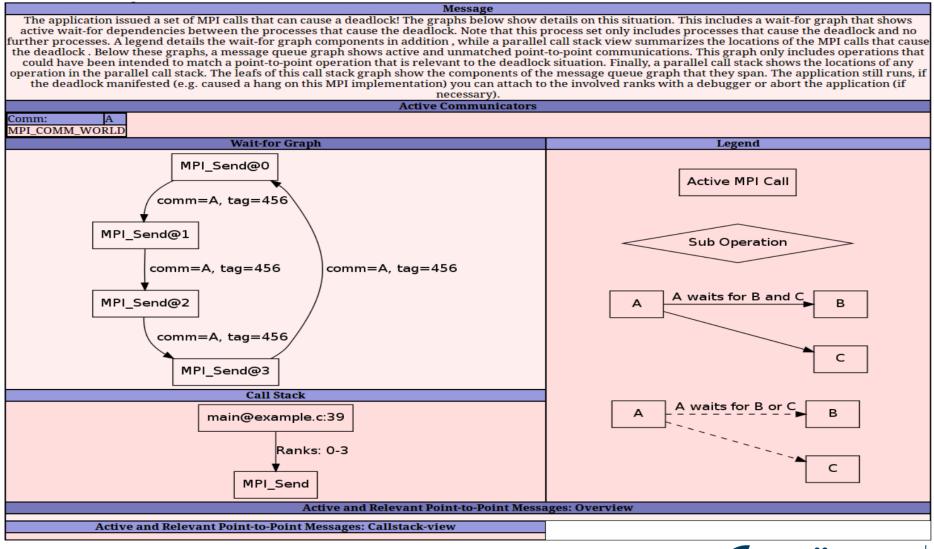


MUST DATATYPE MISMATCH

Rank	Туре	Message	From	References
0	Error	A send and a receive operation use datatypes that do not match! Mismatch occurs at (contiguous) [0](MPI_INT) in the send type and at (MPI_BYTE) in the receive type (consult the MUST manual for a detailed description of datatype positions). A graphical representation of this situation is available in a <u>detailed type mismatch view (MUST_Output-files/MUST_Typemismatch_0.html</u>). The send operation was started at reference 1, the receive operation was started at reference 2. (Information on communicator: MPI_COMM_WORLD) (Information on send of count 1 with type:Datatype created at reference 3 is for C, commited at reference 4, based on the following type(s): { MPI_INT}Typemap = {(MPI_INT, 0), (MPI_INT, 4)}) (Information on receive of count 8 with type:MPI_BYTE)	MPI_Sendrecv called from: #0 main@example.c:33	reference 1 rank 0: MPI_Sendrecv called from: #0 main@example.c:33 reference 2 rank 1: MPI_Sendrecv called from: #0 main@example.c:33 reference 3 rank 0: MPI_Type_contiguous called from: #0 main@example.c:29 reference 4 rank 0: MPI_Type_commit called from: #0 main@example.c:30



MUST DEADLOCK DETECTION





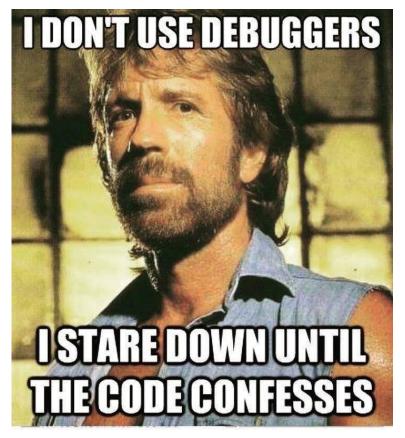
DEBUGGING RECOMMENDATIONS

- Always debug at the lowest possible scale!
- GPU Applications:
 - Single Node / Workstation: Use CUDA-GDB
 - Multi-Node / Supercomputer: Use TotalView/DDT
- MPI Applications:
 - Check with MUST at least once
 - Use TotalView/DDT at small scale (if error occurs there), else attach to as few processes as neccessary



DON'T BE THESE GUYS





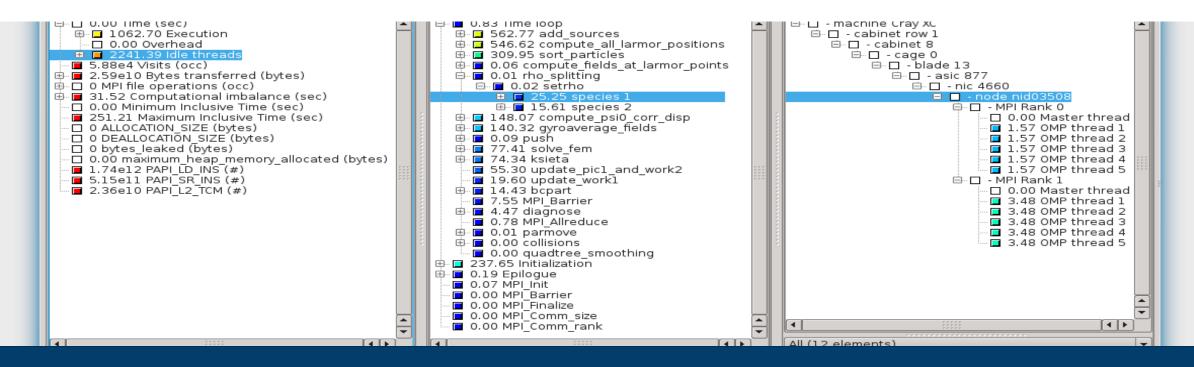


JÜLICH SUPERCOMPUTING CENTRE

REMINDER: DEBUGGING CAN BE FRUSTRATING







PERFORMANCE ANALYSIS TOOLS



TODAY: THE "FREE LUNCH" IS OVER

 10^{4}

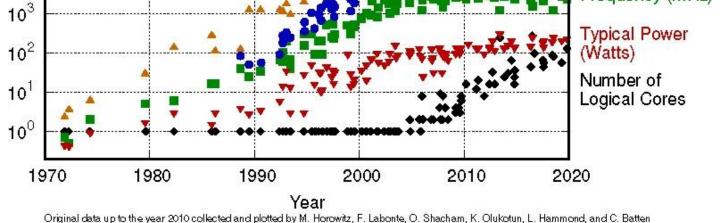
10⁶

10⁵

 10^{4}

New plot and data collected for 2010-2019 by K. Rupp

- Moore's law is still in charge, but
 - Clock rates no longer increase
 - Performance gains only through increased parallelism
- Optimization of applications more difficult
 - Increasing application complexity
 - Multi-physics
 - Multi-scale
 - Increasing machine complexity
 - Hierarchical networks / memory
 - Many-core CPUs and Accelerators
 - Modular Supercomputing Architecture
- Every doubling of scale reveals a new bottleneck!



48 Years of Microprocessor Trend Data



Transistors

(thousands)

Single-Thread

Performance

(SpecINT x 10³)

Frequency (MHz)

PERFORMANCE FACTORS

- "Sequential" (single core) factors
 - Computation
 - Thoose right algorithm, use optimizing compiler
 - Vectorization
 - Choose right algorithm, use optimizing compiler
 - Cache and memory
 - Choose the right data structures and data layout



PERFORMANCE FACTORS

- "Parallel" (multi core/node) factors
 - Partitioning / decomposition

Load balancing

- Communication (i.e., message passing)
- Multithreading
- Core binding / NUMA
- Synchronization / locking
- I/O
 - Often not given enough attention
 - Parallel I/O matters



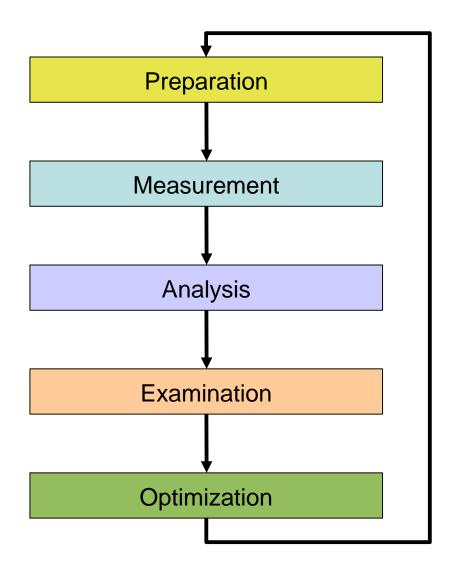
TUNING BASICS

- Carefully set various tuning parameters
 - The right (parallel) algorithms and libraries
 - Compiler flags and directives
 - Correct machine usage (mapping and bindings)

Get the most performance before tuning!

- Measurement is better than guessing
 - To determine performance bottlenecks
 - To compare alternatives
 - To validate tuning decisions and optimizations
 - After each step!

PERFORMANCE ENGINEERING WORKFLOW



- Prepare application (with symbols), insert extra code (probes/hooks)
- Collection of data relevant to execution performance analysis
- Calculation of metrics, identification of performance metrics
- Presentation of results in an intuitive/understandable form
- Modifications intended to eliminate/reduce performance problems



THE 80/20 RULE

- Programs typically spend 80% of their time in 20% of the code
 - Show what matters!
- Developers typically spend 20% of their effort to get 80% of the total speedup possible for the application

F Know when to stop!

Don't optimize what does not matter
 ^T Make the common case fast!



PERFORMANCE MEASUREMENT

Two dimensions

When performance measurement is triggered

- External trigger (asynchronous)
 - Sampling
 - Trigger: Timer interrupt OR Hardware counters overflow
- Internal trigger (synchronous)
 - Code instrumentation (automatic or manual)

How performance data is recorded

• Profile

• Summation of events over time

• Trace

• Sequence of events over time



MEASUREMENT METHODS: PROFILING

- Recording of aggregated information
 - Time
 - Counts
 - Calls
 - Hardware counters
- Across program and system entities
 - Functions, call sites, loops, basic blocks, ...
 - Processes, threads
- Statistical information
 - Min, max, mean and total number of values

Advantages

+ Works also for long-running programs

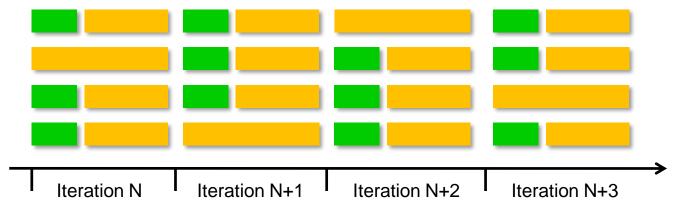
Disadvantages

 Variations over time get lost

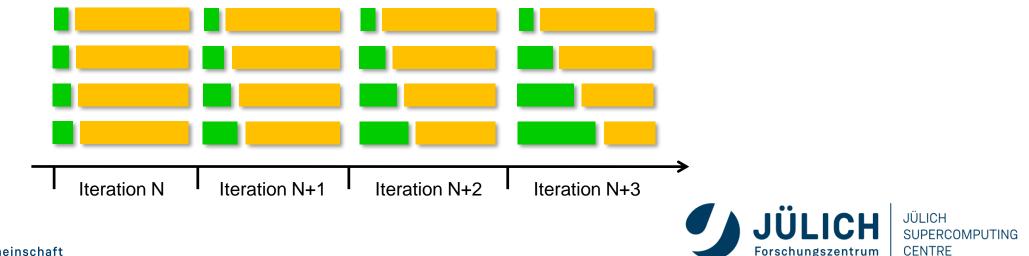


PROFILING: ISSUES RELATED TO "AVERAGING"

• Moving bottleneck across processors can "average out" imbalances



Imbalance changes over time ⇒ problem might not appear in short runs!



MEASUREMENT METHODS: TRACING

- Recording information about significant points (events) during execution of the program
 - Enter/leave a code region (function, loop, ...)
 - Send/receive a message ...
- Save information in event record
 - Timestamp, location ID, event type
 - plus event specific information
- Event trace := stream of event records sorted by time

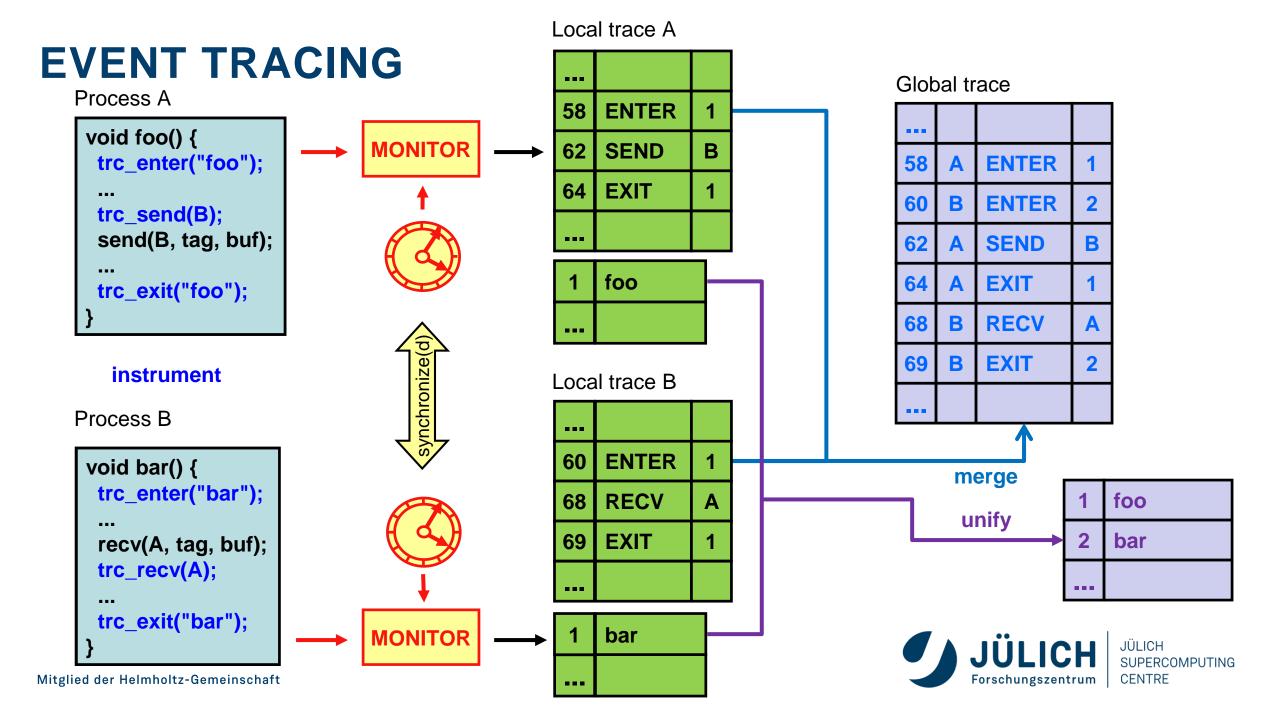
Advantages

- + Can be used to reconstruct the dynamic behavior
- + Profiles can be calculated out of trace data

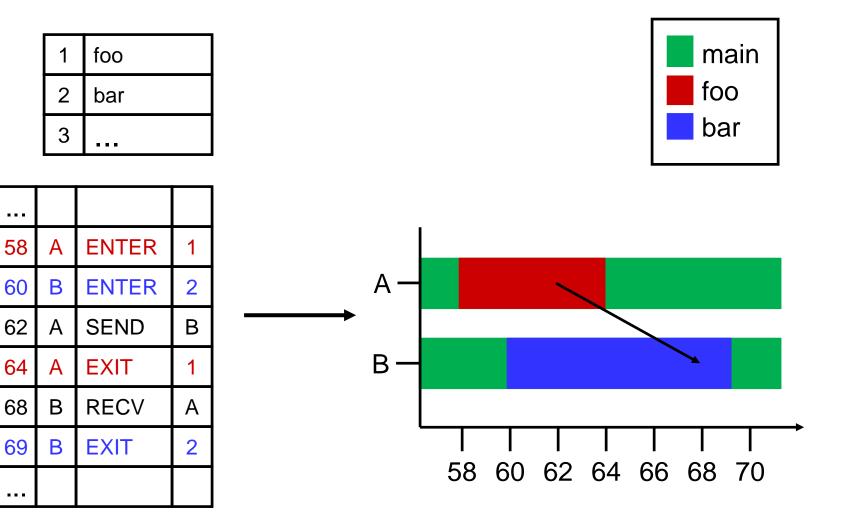
Disadvantages

- HUGE trace files
- Can only be used for short durations or small configurations
 - JÜLICH SUPERCOMPUTING CENTRE

⇒ Abstract execution model on level of defined events



EVENT TRACING: "TIMELINE" VISUALIZATION





CRITICAL ISSUES

- Accuracy
 - Intrusion overhead
 - Measurement takes time and thus lowers performance
 - Perturbation
 - Measurement alters program behaviour
 - E.g., memory access pattern
 - Accuracy of timers & counters
- Granularity
 - How many measurements?
 - How much information / processing during each measurement?
- Tradeoff: Accuracy vs. Expressiveness of data



REMARK: NO SINGLE SOLUTION IS SUFFICIENT!



A combination of different methods, tools and techniques is typically needed!

- Analysis
 - Statistics, visualization, automatic analysis, data mining, ...
- Measurement
 - Sampling / instrumentation, profiling / tracing, ...
- Instrumentation
 - Source code / binary, manual / automatic, ...

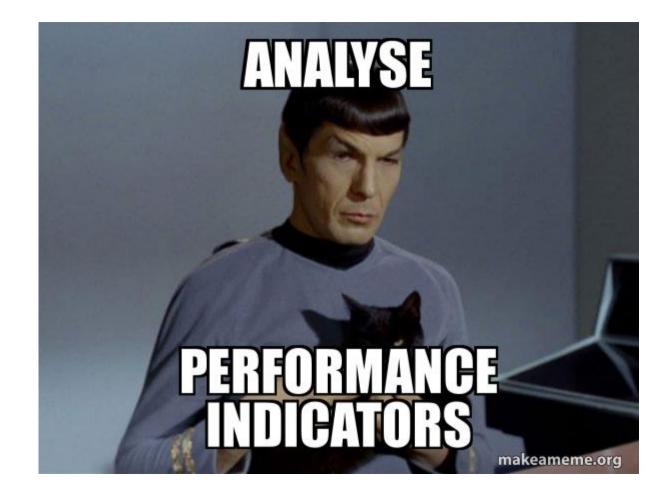


PERFORMANCE TOOLS (STATUS: MAY 2024)

- Score-P
- Scalasca
- Vampir[Server]
- Linaro Forge
 - Performance Reports
 - MAP
- Intel Tools
 - VTune Amplifier XE
 - Intel Advisor
- AMD uProf
- NVIDIA Tools
 - Nsight Systems
 - Nsight Compute
- Darshan

• ...

Mitglied der Helmholtz-Gemeinschaft







- Community-developed
 open-source
- Replaced tool-specific instrumentation and measurement components of partners
- <u>http://www.score-p.org</u>



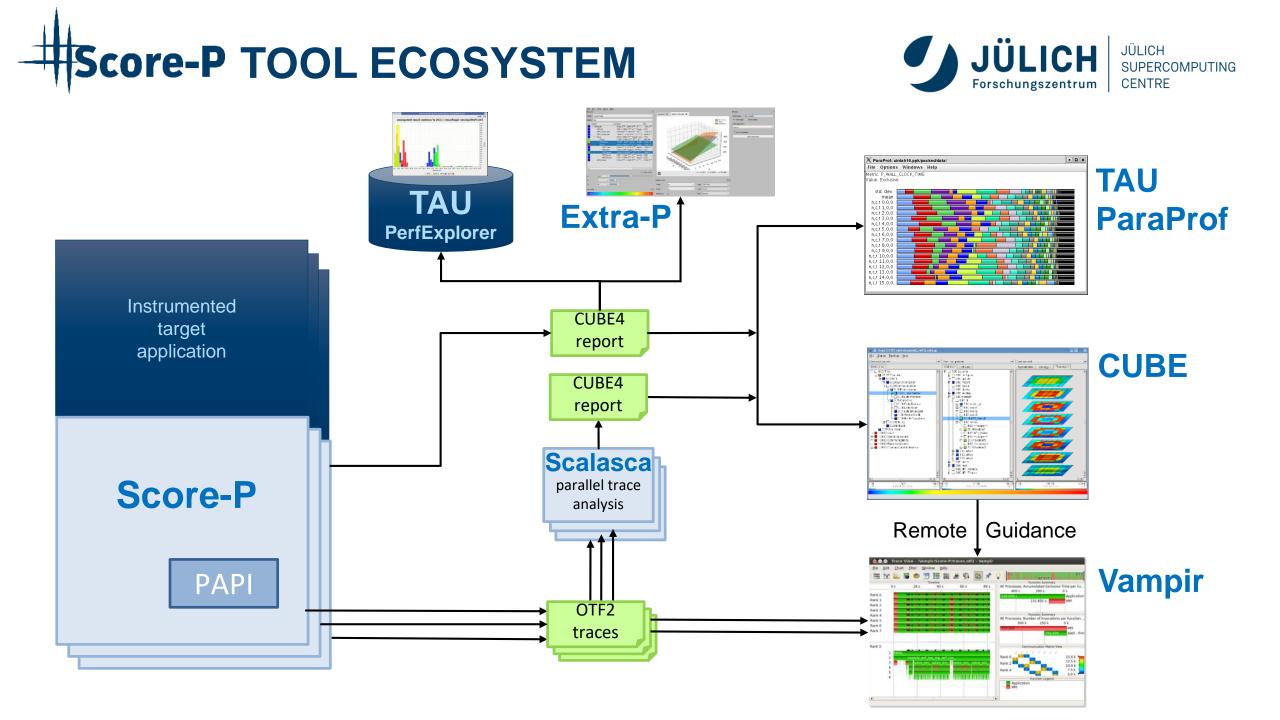


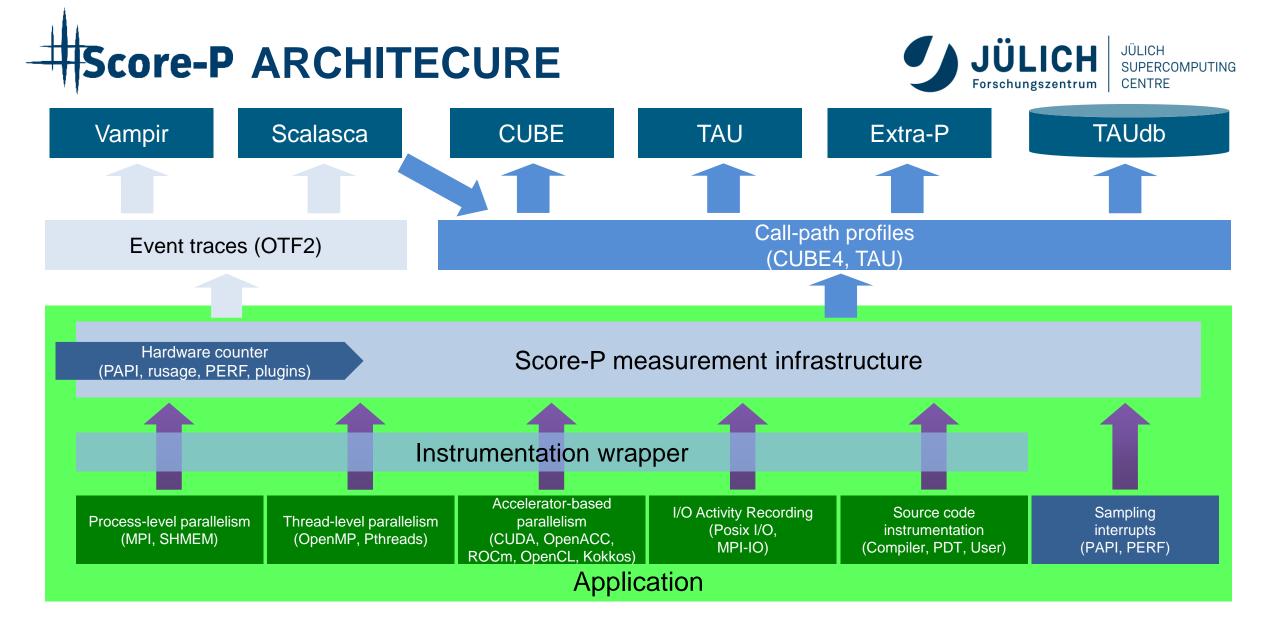




UNIVERSITY OF OREGON







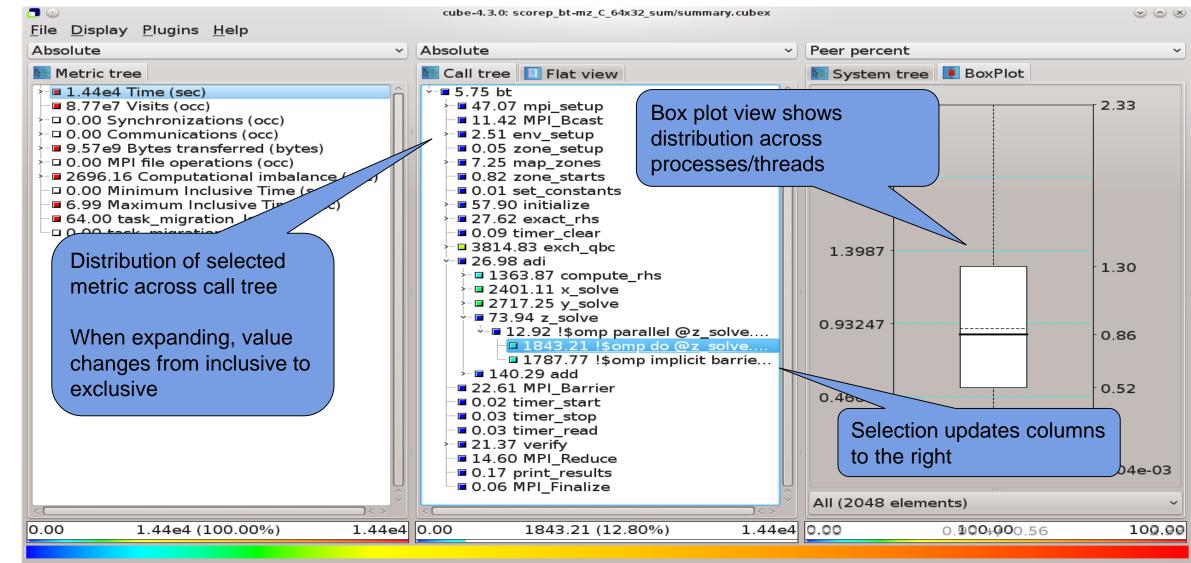
Score-P FUNCTIONALITY

- Provide typical functionality for HPC performance tools
- Instrumentation (various methods)
 - Multi-process paradigms (MPI, SHMEM)
 - Thread-parallel paradigms (OpenMP, POSIX threads)
 - Accelerator-based paradigms (OpenACC, CUDA, OpenCL. Kokkos)
 - In any combination!
- Flexible **measurement** without re-compilation:
 - Basic and advanced **profile** generation (⇒ CUBE4 format)
 - Event **trace** recording (⇔ OTF2 format)
- Highly scalable I/O functionality
- Support all fundamental concepts of partner's tools





CUBE EXAMPLE



Selected "!\$omp do @z_solve.prep.f.52"

Mitglied der Helmholtz-Gemeinschaft

Forschungszentrum CENTRE

SCORE-P: ADVANCED FEATURES

- Measurement can be extensively configured via
 environment variables
 ONE DOES NOT SIMPLY
- Allows for targeted measurements:
 - Selective recording
 - Phase profiling
 - Parameter-based profiling

- FIX PERFORMANCE ISSUES
- GPU support: CUDA, OpenACC, OpenCL, HIP, Kokkos, ...
- Please ask us or see the user manual for details

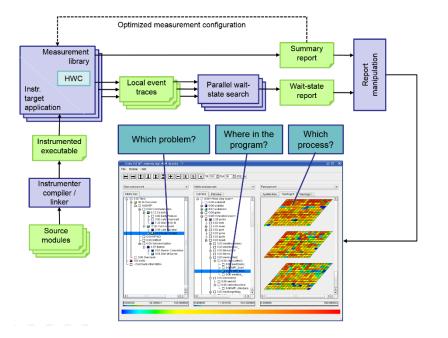


SCALASCA



http://www.scalasca.org/

- Scalable Analysis of Large Scale Applications
- Approach
 - Instrument C, C++, and Fortran parallel applications (with Score-P)
 - Option 1: scalable call-path profiling
 - Option 2: scalable event trace analysis
 - Collect event traces
 - Process trace in parallel
 - Wait-state analysis
 - Delay and root-cause analysis
 - Critical path analysis
 - Categorize and rank results

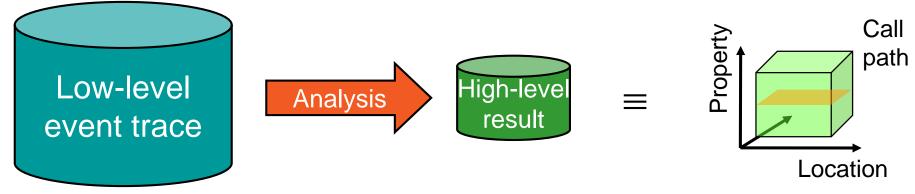




AUTOMATIC TRACE ANALYSIS

- Automatic search for patterns of inefficient behaviour
- Classification of behaviour & quantification of significance
- Identification of delays as root causes of inefficiencies



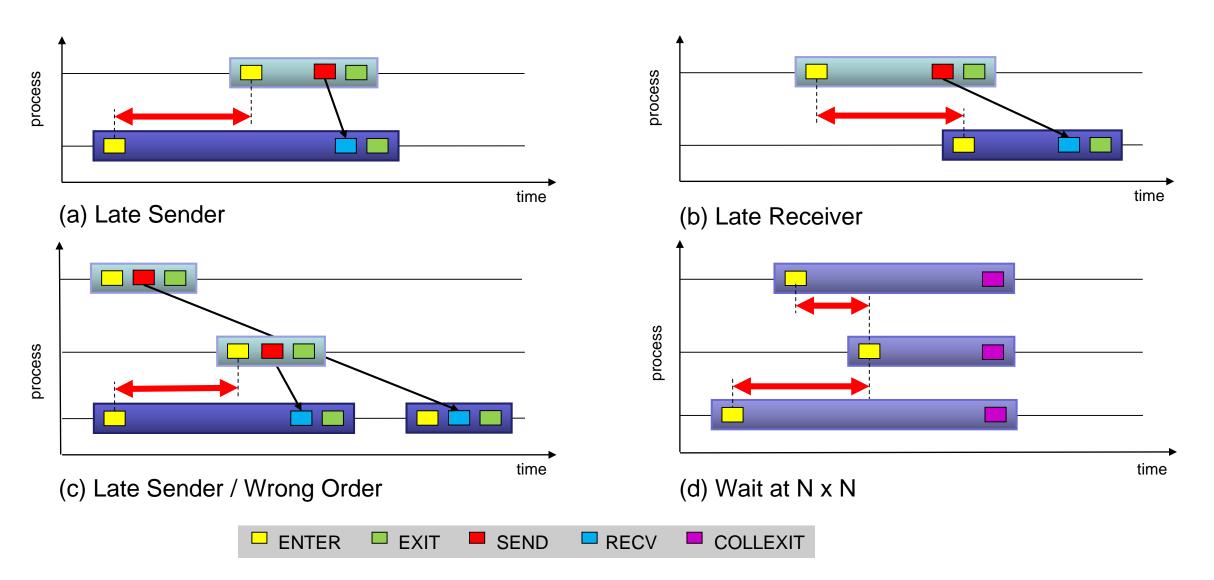


- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis exploits available memory & processors to deliver scalability



EXAMPLE MPI WAIT STATES





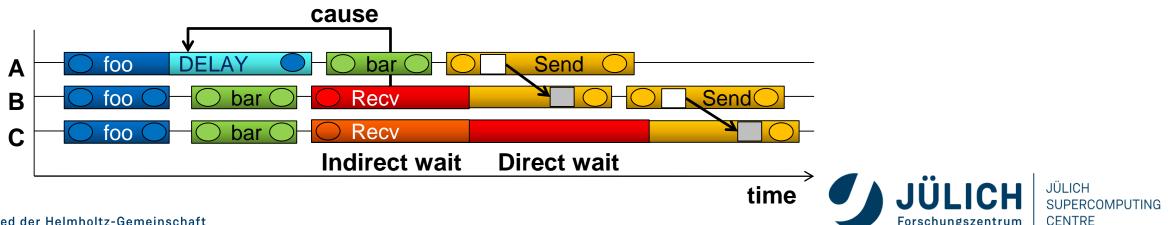
SCALASCA ROOT CAUSE ANALYSIS

Root-cause analysis

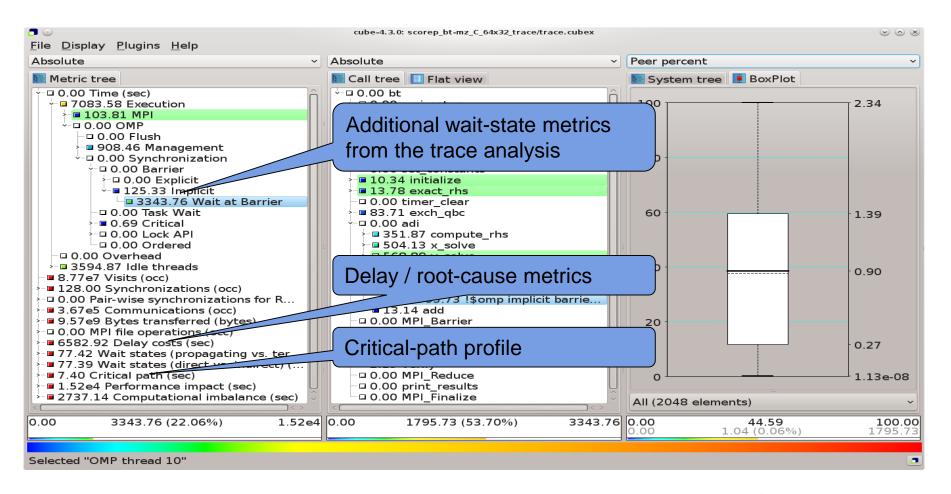
- Wait states typically caused by load or communication imbalances earlier in the program
- Waiting time can also propagate (e.g., indirect waiting time)
- Enhanced performance analysis to find the root cause of wait states

Approach

- Distinguish between direct and indirect waiting time
- Identify call path/process combinations delaying other processes and causing first order waiting time
- Identify original delay



SCALASCA TRACE ANALYSIS EXAMPLE





VAMPIR EVENT TRACE VISUALIZER

- Offline trace visualization for Score-Ps OTF2 trace files
- Visualization of MPI, OpenMP and application events:
 - All diagrams highly customizable (through context menus)
 - Large variety of displays for ANY part of the trace
- http://www.vampir.eu
- Advantage:
 - Detailed view of dynamic application behavior
- Disadvantage:
 - Completely manual analysis
 - Too many details can hide the relevant parts



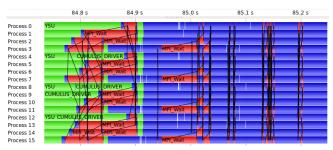


EVENT TRACE VISUALIZATION WITH VAMPIR

- Visualization of dynamic runtime behaviour at any level of detail along with statistics and performance metrics
- Alternative and supplement to automatic analysis
- 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?

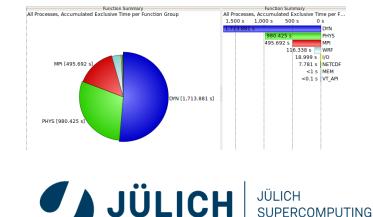
Timeline charts

 Application activities and communication along a time axis



Summary charts

 Quantitative results for the currently selected time interval



CENTRE

VAMPIR PERFORMANCE CHARTS

Timeline Charts



- Master Timeline
- Process Timeline
 - Summary Timeline
- Performance Radar
- Counter Data Timeline
 - I/O Timeline

all threads' activities

- single thread's activities
- all threads' function call statistics
- all threads' performance metrics
- single threads' performance metrics
- all threads' I/O activities

Summary Charts



Function Summary

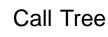


- Message Summary
- I/O Summary



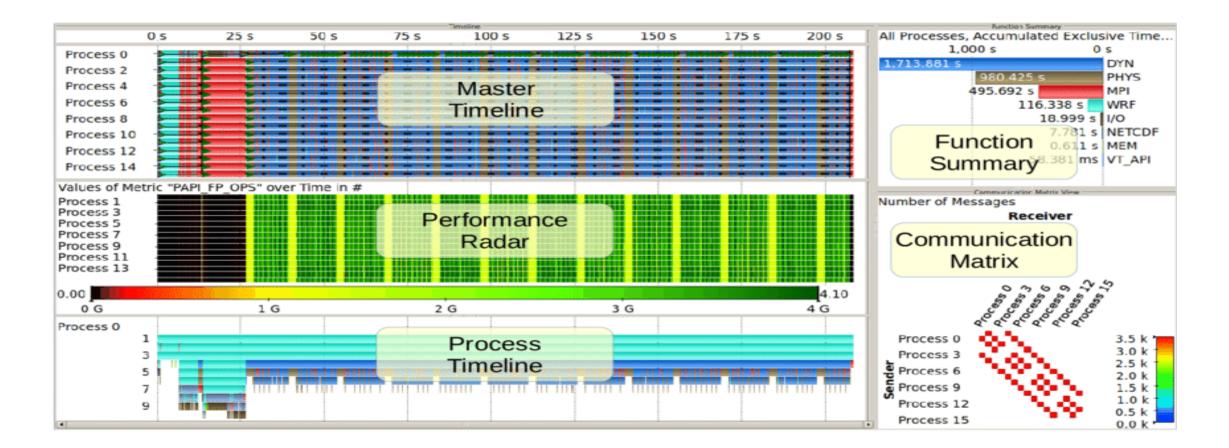
Process Summary







VAMPIR DISPLAYS





LINARO PERFORMANCE REPORTS



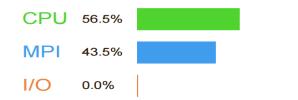
- Single page report provides quick overview of performance issues
- Works on unmodified, optimized executables
- Shows CPU, memory, network and I/O utilization
- Supports MPI, multi-threading and accelerators
- Saves data in HTML, CVS or text form
- <u>https://www.linaroforge.com/linaroPerformanceReports</u>
- Note: License limited to 128 processes (with unlimited number of threads)



EXAMPLE PERFORMANCE REPORTS

Summary: cp2k.popt is CPU-bound in this configuration

The total wallclock time was spent as follows:



Time spent running application code. High values are usually good. This is average; check the CPU performance section for optimization advice. Time spent in MPI calls. High values are usually bad. This is average; check the MPI breakdown for advice on reducing it. Time spent in filesystem I/O. High values are usually bad.

This is **negligible**; there's no need to investigate I/O performance.

This application run was CPU-bound. A breakdown of this time and advice for investigating further is in the CPU section below.

CPU

A breakdown of how the 56.5% total CPU time was spent:

Scalar numeric ops	27.7%	
Vector numeric ops	11.3%	1 - C
Memory accesses	60.9%	
Other	0.0	1

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

Little time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

I/O

A breakdown of how the 0.0% total I/O time was spent:

Time in reads	0.0%
Time in writes	0.0%

Estimated read rate 0 bytes/s

Estimated write rate 0 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

MPI

Of the 43.5% total time spent in MPI calls: Time in collective calls 0 20/

Time in conective cans	0.270	•
Time in point-to-point calls	91.8%	
Estimated collective rate	169 Mb/s	
Estimated point-to-point rate	50.6 Mb/s	

The point-to-point transfer rate is low. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait. Use an MPI profiler to identify the problematic calls and ranks.

Memory

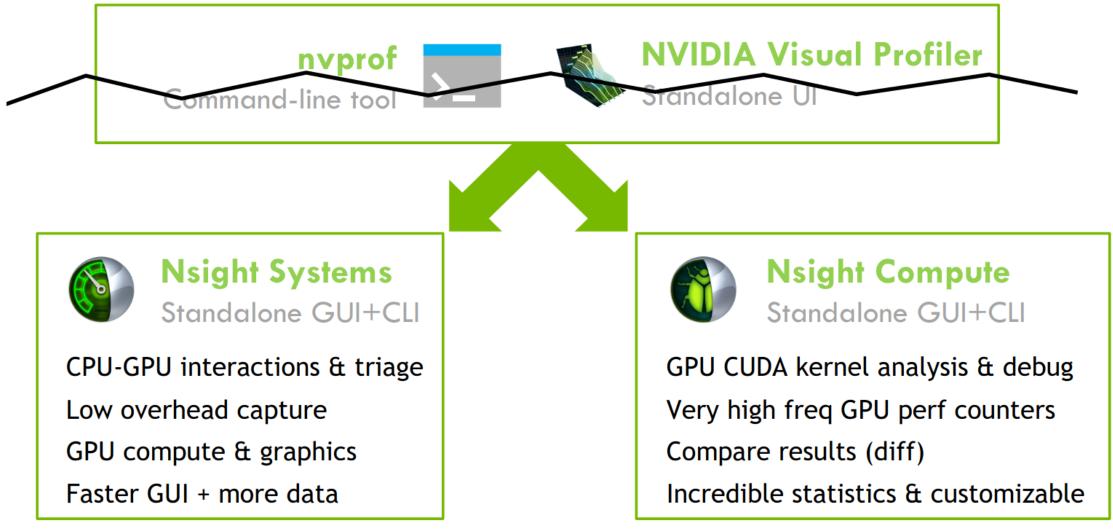
Per-process memory usage may also affect scaling:

Mean process memory usage 82.5 Mb Peak process memory usage 89.3 Mb Peak node memory usage 7.4%

The peak node memory usage is low. You may be able to reduce the total number of CPU hours used by running with fewer MPI processes and more data on each process.



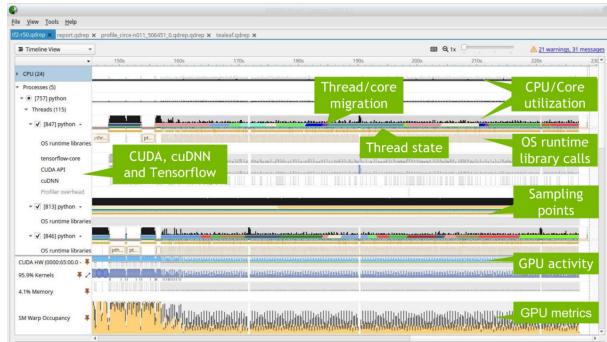
NVIDIA TOOLS -- LEGACY TRANSITION





NSIGHT SYSTEM

- System-wide application tuning
- Locate optimization opportunities
 - Visualize millions of events on a timeline
 - See gaps of unused CPU and GPU time
- Balance workloads across multiple CPUs and GPUs
 - CPU utilization and thread state
 - GPU streams, kernels, memory transfers, etc.
- Multi-platform support
 - Linux, Windows and Mac OS X (host-only)
- x86-64, Power9, ARM server, Tegra (Linux & QNX)





GPU METRIC SAMPLING

 16.66 16.76 16.8s 	16.9s 17s	17,18 17,29 17,39 17,48 17	56 17,66 17,76 17,88 17,95
CPU (256)			
GPU (A100 Graphics Device - 00	Distance of the second		
GPC Clock Prequency		Is my GPU full? Sufficient g	rids size & streams?
SY'S Clock Frequency	CONTRACT OF CALLER OF	Is my instruction rate low ((possibly IO bound)?
GR Active	CONTRACTOR DESCRIPTION	Is my instruction rate tow	(possibly to bound):
SM Instructions (Emonil Emonil Emonil Society	desides stated dates	Am I using tensor cores?	
		-	
SM Warp Occupancy	and a design of the	Can I see GPU Direct RDMA	/Storage or other transfers
> DRAM Bandwidth	Augustian Aug		
PCIe Bandwidth Control of the second secon		 System-wide GPU observat 	10N
GPU Metrics	· was hardward a farmer	(no app required, but sudo or regke	V)
GPC Clock Frequency			
SYS Clock Prequency GR Active	<u> </u>	 10kHz default can be incre 	ased depending on GPU
SM Adhe	1 10 0 00 00 00 00 00 00 00 00 00 00 00	Metrics:	
SM instructions without instructions into any instructions		Metrics:	
SM Wara Occupancy			
Malata Dialata	N. Contraction of the second s	SM utilizations	IO throughputs
DRAM Bandwidth Cla Bandwidth	4.	120000	.
CPU (A100 Graphics Device - 00	the Later A. Serre	 SMs active 	 PCIe
	- Leitensite influen hat	- Instructions	a NIV/Limb
GPU (A100 Graphics Device - 0) GPU Metrics	and hain deration	 Instructions 	 NVLink
GPU (A100 Grephics Device - OC		 TensorCores 	 DRAM
GPU Metrics CPU (A100 Graphics Device - 00	e bel, let d Writelitet de Jean		• DIAM
GPU Metrics	when the second or the second second	 Warp occupancy 	
GPU (A100 Graphics Device - 00 GPU Metrics	Land		
GPU (A100 Graphics Device - 00		(including unallocated	I SLOTS)
JDA HW (Unksown GPU)	fi		
a Streams) 🔹 > 100 100 100 100 100 100 100 100 100 1	10011	dipadid_proped_ident_innet) dipadid_proped_innet_innet) (00	in the second strate of the se
[29106] python •			
	daffernationer 💦 🚺 👬		sorFlow on 8xGA100 at 20kHz
[24832] pythin • 🕈 🗰			SOLITION OIL ONOATOU AL ZURITZ
JDA API		ususato (normalia:	

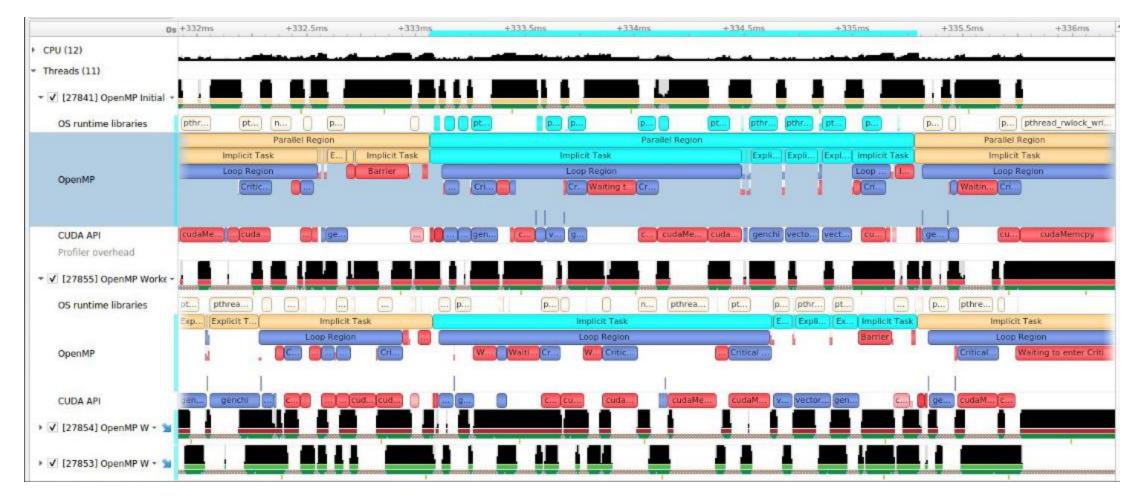


MULTI NODE SUPPORT – SHMEM, MPI, UCX, AND NCCL

OS runtime libraries	971 px; 0 msec						
SHMEM	971 px, ormset	shmem_float_p	parrier_all [18,019 μs]	shmem_finalize [1,279 s]	-		
Profiler overhead	971 px; 0 mset	Begins: 0,472616s Ends: 0,472619s (+2,470 µs)					
[1430917]	971 px; 0 msec	Thread: 1430905					
[174635] MPI Rank 0	•						
MPI	MPI_Jsend [7,319 µs]	MPI_Irecv [7,585 µs]		MPI_Waitall [11,493 µs]			
UCX	ucp_tag_send_nbx UCP tr	ucp_tag_recv_nbx UCP transf.		ucp_rkey	+5211s 521,6087ms1,62ms +521,64	ims +521,66ms +521,68r	Q 1x <u>23 mes</u> <u>23 mes</u> <u>23 mes</u> <u>521,7ms</u> +521,74ms
	ucp_tag_send_nbx Begins: 0,177984s	ucp tag send nbx UCP tra	ansfer processing [21,183 µs]		-32115 521,608/ms (102)15 -3321,0	115 +321,00115 +321,001	113 +321/105 +321/2013 +321/2015
Start & End	1019 px; 0 msec Ends: 0,177988s (+3,990 µs)	A (P transfer processing [14,243 μs]	clAllReduce [26,3		
Profiler overhead	1019 px; 0 msec CategoryId: 2	1	A		imemx_int_sum_r		nvshmem_free [91,979 µs]
threads hidden	+ 1019 px; 0 msec Category: UCP transfer submit						nvshmem_quiet []
					nctke L. cor rudaStreamSvr	ichronize cud	
	ompletion tracking of ne	l on-blocking UCP	l communicat	A A A A A A A A A A A A A A A A A A A	ncciKe) (cor) (cudaStreamSyr		
		l on-blocking UCP	Communicat	- 1 ¹ 1063 pxp ² msec			
		l on-blocking UCP	 CUDA HW (0000:06:00.0 [All Streams] 	1 ¹ 1063 proj msec	ncclKe) cor) (cudaStreamSyr		
		l on-blocking UCP	 CUDA HW (0000:06:00.0 [All Streams] 79.5% Default stream 3 	1063 px 1 msec	ncciKernel_AllReduce_RING_L		CuStreamSy) (barr) cuStrea)
		l on-blocking UCP	 CUDA HW (0000:06:00.0 [All Streams] 	1063 px p msec 1063 px p msec 1065 p	ncciKernel_AliReduce_RING_L ncclKernel_AliReduce_R Begins: 1,52161s Ends: 1,52165s (+41,504)	ING_LL_Sum_int32_t	CuStreamSy) barr) cuStrea) barrie) barrie)
		l on-blocking UCP	 CUDA HW (0000:06:00.0 [All Streams] 79.5% Default stream 1 7.335⁺ 19.8% Stream 22 	1063 px 1 msec	ncciKernel_AllReduce_RING_L ncciKernel_AllReduce_R Begins: 1,52161s	ING_LL_Sum_int32_t	CuStreamSy) [barr] cuStrea) [barrie] [barrie]
		on-blocking UCP	 CUDA HW (0000:06:00.0 [All Streams] 79.5% Default stream 3 	1063 px p msec 1063 px p msec 1065 p	ncciKernel_AllReduce_RING_L ncciKernel_AllReduce_R Begins: 1,52161s Ends: 1,52165s (+41,504) grid: <<<1, 1, 1>>> block: <<<64, 1, 1>>> Launch Type: Regular	ING_LL_Sum_int32_t	() cuStreamSy) barr) cuStrea barrie) barrie)
		on-blocking UCP	 CUDA HW (0000:06:00.0 [All Streams] 79.5% Default stream 77.335* 19.8% Stream 22 Events View * 	1063 px p msec 1063 px p msec 1065 p	ncciKernel_AliReduce_RING_L ncciKernel_AliReduce_R Begins: 1,521615 Ends: 1,521655 Ends: 1,521655 Ends: 1,521655 Use State Sta	ING_LL_Sum_int32_t us) 1.472 bytes r: 0 bytes	CuStreamSy) (barr) cuStrea) (barrie) (barrie)
		on-blocking UCP	 CUDA HW (0000:06:00.0 [All Streams] 79.5% Default stream 77.335* 19.8% Stream 22 Events View * # * Name 	1 1063 px 0 msec 1063 px 0 msec	ncciKernel_AllReduce_RING_L ncciKernel_AllReduce_R Begins: 1,52161s Ends: 1,52165s (+41,504) grid: <<<1, 1, 1>>> block: <<<64, 1, 1>>> Launch Type: Regular Static Shared Memory: 4' Dynamic Shared Memory Registers Per Thread: 96 Local Memory Per Thread: 96	ING_LL_Sum_int32_t µs) 1.472 bytes r: 0 bytes d: 0 bytes	CuStreamSy) barr cuStrea barrie barrie Name • Description:
		on-blocking UCP	 CUDA HW (0000:06:00.0 [All Streams] 79.5% Default stream 3 7.335* 19.8% Stream 22 Events View * # * Name 730 * nvshn 	1 1063 profimsec 1063 profimsec 1065 profims	ncciKernel AllReduce RING L ncciKernel AllReduce RING L Begins: 1,521615 Ends: 1,521655 (+41,504) grid: <<<1, 1, 1>>> block: <<<64, 1, 1>>> block: <<64, 1>, 1>>>	ING_LL_Sum_int32_t us) 1.472 bytes r. 0 bytes d: 0 bytes d: 0 bytes d: 0 bytes d: 9 bytes d: 9 bytes 19621 19621 19621	CuStreamSy barr barrie barrie.
		on-blocking UCP	CUDA HW (0000:06:00.0 [All Streams] 79.5% Default stream 1 7.335* 19.8% Stream 22 Events View # Name 730 Name 730 Name 731 rushm	1 1063 px 0 msec 1063 px 0 msec	IncciKernel AllReduce_RING_L AllReduce_RING_L Begins: 1,52161s Ends: 1,52165s (+41,504) grid: <<<1, 1, 1>>> block: <<<64, 1, 1>>> block: <	LS ING_LL_Sum_int32_t us) 1.472 bytes t: 0 bytes t: 0 bytes t: 0 bytes t: 98.304 bytes t: 98.304 bytes t: 48 19621	CuStreamSy barr barrie barrie.
6 threads hidden – CC		on-blocking UCP	CUDA HW (0000:06:00.0 [All Streams] 79.5% Default stream 7 7.335* 19.8% Stream 22 Events View # Name 730 Name 731 Roshn 1 731 Constant 732 Constant	1063 profimsec 1063 profimsec 1064 profimsec 1065 profimsec	ncciKernel AllReduce RING L ncciKernel AllReduce RING L Begins: 1,521615 Ends: 1,521655 (+41,504) grid: <<<1, 1, 1>>> block: <<<64, 1, 1>>> block: <<64, 1>, 1>>>	LS ING_LL_Sum_int32_t us) 1.472 bytes t: 0 bytes t: 0 bytes t: 0 bytes t: 98.304 bytes t: 98.304 bytes t: 48 19621	CuStreamSy,) barr, barrie barri



OPENMP



OMPT-capable OpenMP runtime required



EXPERT SYSTEM

le <u>View T</u> ools <u>H</u> elp									CUDA Synch	ronous Memcpy	E.
rofile_circe-n011_506451_0.qdrep.q	qdrep ×								CUDA Synch	ronous Memset	
■ Timeline View ·								📼 Q 1x	CUDA Synch	ronization APIs	
55 •		+676,05ms	+676,1ms	+676,150	ns +67	76,2ms	+676,25ms	+676,3ms	CUDA GPU S	tarvation	
CUDA API 🐺	cudaStreamS.			el] c	udaMemc	cudaMemcpyAsyn	cuda)	0	CUDA GPU L	ow Utilization	
OS runtime libraries						pthread_cond_wait			VULKAN GPI	U Starvation	
10 threads hidden - +	- 1179 px; 0 m	isec.							VUI KAN GPI	U Low Utilization	
▼ CUDA HW (0000:34:00.0 - Te	1179 pc m	iset	1 1						- VOLINIT GI	o con othization	
🕨 [All Streams] 🛛 🖡 🕽	ncciAll	isec	1 1		j.						
60.8% Default stream 1	1179 pic m	isec	1 1	1	1 I						
	neciAll 0 m		100								
10.791 39.2% Stream 29	C III										
10.791 39.2% Stream 29	4	70-									Þ
10.791 39.2% Stream 29 Expert System View *	CALL MAN POINT OF THE										Þ
	CALL MAN POINT OF THE										🖗 Settings
Expert System View 💌	4		Start	Src Kind	Dst Kind	Bytes	PID	Device ID	Context ID	Stream ID	
Expert System View 🔹 CUDA Async Memcpy with Pageabl	• ble Memory •		Start 6,38792s	Src Kind Device		Bytes 8 B		Device ID 0	Context ID	Stream ID 7	Setting API Name
Expert System View 👻 CUDA Async Memcpy with Pageabl The following APIs use PAGEABLE r which causes asynchronous CUDA	• ble Memory • memory A memcpy	Duration *	C 675-C		Pageable		75475		Context ID 1		Setting API Name cudaMemcp
Expert System View * CUDA Async Memcpy with Pageabl The following APIs use PAGEABLE r which causes asynchronous CUDA operations to block and be execute	• ble Memory • memory A memcpy ted	Duration * 2,048 µs	6,38792s	Device	Pageable Pageable	8 B	75475 75475	0	Context ID 1 1		Setting API Name cudaMemcp cudaMemcp
Expert System View * CUDA Async Memcpy with Pageabl The following APIs use PAGEABLE r which causes asynchronous CUDA operations to block and be execute synchronously. This leads to low G	• ble Memory • memory A memcpy ted	Duration * 2,048 µs 2,048 µs	6,38792s 6,8334s	Device Device	Pageable Pageable Pageable	8 B 4 B	75475 75475 75475	0	1	7	Setting API Name cudaMemcpy cudaMemcpy cudaMemcpy
Expert System View * CUDA Async Memcpy with Pageabl The following APIs use PAGEABLE r which causes asynchronous CUDA operations to block and be execute synchronously. This leads to low G utilization.	A memory A A memory A SPU	Duration * 2,048 µs 2,048 µs 2,016 µs	6,38792s 6,8334s 2,5394s	Device Device Device	Pageable Pageable Pageable Pageable	8 B 4 B 4 B	75475 75475 75475 75475 75475	0 0	1	7 7 7	Setting: API Name cudaMemcp cudaMemcp cudaMemcp cudaMemcp
Expert System View * CUDA Async Memcpy with Pageabl The following APIs use PAGEABLE r which causes asynchronous CUDA operations to block and be execute synchronously. This leads to low G utilization. Suggestion: If applicable, use PINM	A memory A A memory A SPU	Duration ▲ 2,048 µs 2,048 µs 2,016 µs 2,016 µs	6,38792s 6,8334s 2,5394s 3,90617s	Device Device Device Device Device	Pageable Pageable Pageable Pageable Pageable	8 B 4 B 4 B 48 B	75475 75475 75475 75475 75475 75475	0 0 0	1 1 1 1 1	7 7 7 7 7 7	Setting: API Name cudaMemcp cudaMemcp cudaMemcp cudaMemcp
Expert System View * CUDA Async Memcpy with Pageabl The following APIs use PAGEABLE in which causes asynchronous CUDA operations to block and be execute synchronously. This leads to low Gi utilization. Suggestion: If applicable, use PINM memory instead.	A memory A A memory A SPU	Duration 2,048 µs 2,048 µs 2,016 µs 2,016 µs 2,016 µs	6,38792s 6,8334s 2,5394s 3,90617s 4,25257s	Device Device Device Device Device	Pageable Pageable Pageable Pageable Pageable Pageable	8 B 4 B 4 B 48 B 48 B 48 B	75475 75475 75475 75475 75475 75475 75475	0 0 0 0	1 1 1 1 1	7 7 7 7 7 7	Setting: API Name cudaMemcpy cudaMemcpy cudaMemcpy cudaMemcpy cudaMemcpy cudaMemcpy
	Memory memory A memcpy ted SPU NED	Duration * 2,048 µs 2,048 µs 2,016 µs 2,016 µs 2,016 µs 2,016 µs	6,38792s 6,8334s 2,5394s 3,90617s 4,25257s 5,67617s	Device Device Device Device Device Device	Pageable Pageable Pageable Pageable Pageable Pageable Pageable	8 B 4 B 4 B 48 B 48 B 48 B 48 B	75475 75475 75475 75475 75475 75475 75475 75475	0 0 0 0 0	1 1 1 1 1	7 7 7 7 7 7 7 7	Settings API Name cudaMemcp cudaMemcp cudaMemcp cudaMemcp cudaMemcp cudaMemcp



NSIGHT COMPUTE

- Interactive CUDA kernel profiler
- Targeted metric sections for various performance aspects
- Customizable data collection and presentation (tables, charts, ...)
- GUI and CLI
- Python-based API for guided analysis and post-processing
- Support for remote profiling across machines and platforms

Page: Summary	- Launch:	0 - 43843 - device_tea_leaf	_ppcg_sot 👻	💎 👻 Add Baseline	e • Apply <u>R</u> ules	🗟 Occupancy Calcul	ator					Copy as In	hage
	Launch			Time	Cycles Reg	GPU		SM Freq	uency	CC Process			
Current	43843 - devi	ce_tea_leaf_ppcg_solve_init	(126, 1001, 1)x(32, 4, 1) 217.63 use	cond 297,114 40	0 - NVIDIA GeForce	RTX 2080 Ti	1.36 cycle	e/nseco	ond 7.5 [15958]tea_	eaf		
ID 🔺 Time	API Call ID	Function Name	Demangled N	Process	Device Name	Grid Size	Block Size			Cycles [cycle] Duratio	n [msecond] Compute	Throughput [%]	N
0 2021-Dec	43843	device_tea_leaf_ppcg_	device_te_	[15958] tea_leaf	NVIDIA GeForce.	126, 1001,		32,	4, 1	297,114	0.22	77.8	9
1 2021-Dec-1	43857	device_tea_leaf_ppcg_sol	device_tea_l	[15958] tea_leaf	NVIDIA GeForce RT			32,	4, 1	1,264,921	0.94	54.7	8
2 2021-Dec-1	43860	device_tea_leaf_ppcg_sol	device_tea_l	(15958) tea_leaf	NVIDIA GeForce RT			32,	4, 1	1,462,446	1.07	86.2	
3 2021-Dec-1_	43863	device_tea_leaf_ppcg_sol_	device_tea_I	[15958] tea_leaf	NVIDIA GeForce RT	126,1001,		32.	4, 1	1,443,836	1.06	23.8	
													_
Page: Details	✓ Launch:	0 - 43843 - device_tea_leaf	ppcg_sol +	√ → Add Baseline	✓ Apply <u>R</u> ules	🗟 Occupancy Calcula	ator					Copy as Im	age
	Launch			Time	Cycles Regs	GPU		SM Freq	uency	CC Process			
Current	43843 - devi	ce tea leaf ppcg solve init	(126, 1001, 1)x(32.4.1) 217.63 use			RTX 2080 Ti				af		á.
GPU Speed Of L	ight Through	put										- 5	5
Compute (SM) Throu	ighput [%]				77.89	Juration [usecond]						217.6	53
Memory Throughput	[%]				45.03	lapsed Cycles [cycle]						297,11	4
L1/TEX Cache Throu	ghput [%]				68.22 5	M Active Cycles [cycle	e]					293,385.1	19
L2 Cache Throughpu	ıt [%]				2.30	SM Frequency [cycle/n	second]					1.3	36
DRAM Throughput [%	() ()				0.12 [ORAM Frequency (cycle	e/nsecond]					6.7	9
A High Comp	ute Throughp	t Compute is more heav whether any computat					to see what t	he compute	e pipeli	nes are spending their t	ime doing. Also, consider		
					Initian in 20.4 The lines	nel achieved 0% of this	e device e fo?	2 nook norf	orman	re and 19% of its fn64 r	ask nerformance if Com		
🔥 FP64/32 Ut	ilization 🚻	ratio of peak float (fp32) to obligat Analysis determines ilysis.				on floating point oper					ing Guide for mode deta	ils on roofline	
▲ FP64/32 Ut	ilization wa	oload Analysis determines Ilysis.											D
	ilization and and and and and Analysis	oload Analysis determines Ilysis.			ler using 32-bit precisi								



PROFILER REPORT

Selected result

1

Metric values

1

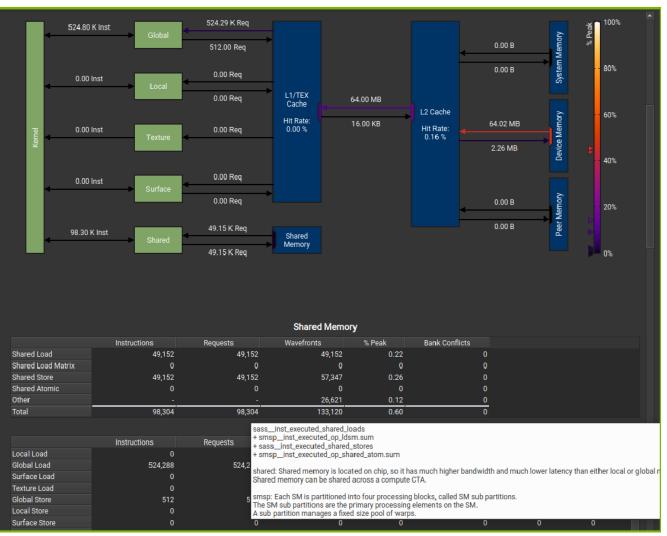
e: Details	Launch: 0 - 43843 - device_tea_leaf_ppcg.	_sol 👻 🍸 🔹 Add Baseline 👻 Ap	pply <u>R</u> ules 🔲 Occupancy Calculator		Copy as Ima
	Launch	Time Cy	ycles Regs GPU	SM Frequency CC Pre	ocess $\oplus \Theta$ 0
Current	43843 - device_tea_leaf_ppcg_solve_init (126, 1	1001, 1)x(32, 4, 1) 217.63 usecond 29	97,114 40 0 - NVIDIA GeForce RTX 2080) Ti 1.36 cycle/nsecond 7.5 [15	5958] tea_leaf
GPU Speed Of Li	ight Throughput				
ompute (SM) Throug	ghput (%)		77.89 Duration [usecond]		217.63
mory Throughput [45.03 Elapsed Cycles [cycle]		297,114
/TEX Cache Throug			68.22 SM Active Cycles [cycle]		293,385.19
Cache Throughput			2.30 SM Frequency [cycle/nsecond]		1.36
RAM Throughput [%]	4		0.12 DRAM Frequency [cycle/nsecon		6.79
High Compu			e Workload Analysis report section to see wh	at the compute pipelines are spend	ding their time doing. Also, consider
Å High Compu Å FP64/32 Util	The ratio of peak float (fp32) to doub	redundant and could be reduced or mov le (fp64) performance on this device is :		fp32 peak performance and 19% o	of its fp64 peak performance. If <u>Compute</u>
▲ FP64/32 Util	The ratio of peak float (fp32) to doub lization Workload Analysis analysis.	redundant and could be reduced or mov le (fp64) performance on this device is :	ved to look-up tables. 32:1. The kernel achieved 0% of this device's	fp32 peak performance and 19% o	of its fp64 peak performance. If <u>Compute</u>
A FP64/32 Util	The ratio of peak float (fp32) to doub lization Workload Analysis analysis.	redundant and could be reduced or mov le (fp64) performance on this device is :	ved to look-up tables. 32:1. The kernel achieved 0% of this device's	fp32 peak performance and 19% o	of it <mark>s fp64 peak performance. If <u>Compute</u> Kerrel Profiling Guide for mode details on roofline</mark>
	Iteration Whether any computation is in Iization The ratio of peak float (fp32) to doub Workload Analysis determines that the analysis. oad Analysis Iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii	redundant and could be reduced or mov le (fp64) performance on this device is :	ved to look-up tables. 32:1. The kernel achieved 0% of this device's 132-bit precision floating point operations to	fp32 peak performance and 19% o	of it <mark>s</mark> fp64 peak performance. If <u>Compute</u> Kerr <mark>el Profiling Guide</mark> for mode details on roofline

Expandable Sections Expert Analysis (Rules)



DATA TRANSFER ANALYSIS

- Detailed memory workload analysis chart and tables
- Shows transferred data or throughputs
- Tooltips provide metric names, calculation formulas and detailed background info





BASELINE COMPARISON

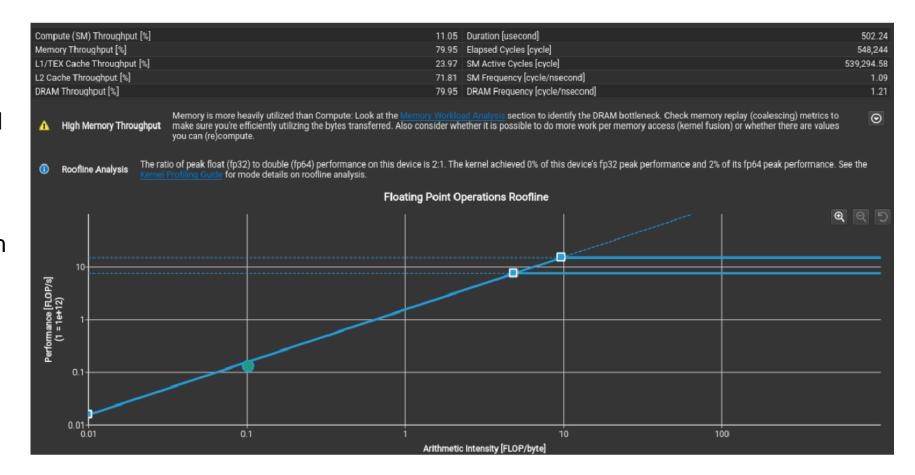
- Comparison of results directly within the tool with "Baselines"
- Supported across kernels, reports, and GPU architectures

Current	655 - reduceFinal (1,	1, 1)x(512, 1, 1)	5.98 usecond	4,807 16	5 NVIDIA GeF	orce RTX 2080 Ti	801.61 cycle/useco	nd 7.5 [496	9] simpleCuda	Graphs		
Baseline	654 - reduce (512, 1,	1)x (512, 1, 1)	234.72 usecond	318,783 16	5 NVIDIA GeF	orce RTX 2080 Ti	1.35 cycle/nsecond	7.5 [496	9] simpleCuda	Graphs		
 GPU Speed Of Lig 	ght Throughput									GPU 1	Throughput Cha	irt 🔻 🗘
Compute (SM) Thro					0.33 (-99.58%	Duration [useco	nd]					5.98 (-97.45%)
Memory Throughput					0.80 (-98.25%							4,807 (-98.49%)
1/TEX Cache Throu					11.66 (+12.76%)	SM Active Cycle	es [cycle]					46.09 (-99.98%)
2 Cache Throughpu					0.80 (-94.49%)							801.61 (-40.80%)
DRAM Throughput [%]				0.48 (-98.94%	DRAM Frequence	cy [cycle/nsecond]					3.98 (-41.14%)
🛕 Small Grid	This kernel grid is too sm	all to fill the avai	ilable resources or	n this device, re	esulting in only 0.	0 full waves acros	s all SMs. Look at <u>La</u>		or more deta	ils.		
Roofline Ana	lysis The ratio of peak t	loat (fp32) to do	ouble (fp64) perfor	mance on this	device is 32:1. T	he kernel achieved	10% of this device's f	o32 peak perfo	ormance and c	lose to 0% of	its fp64 peak p	erformance.
					GPU T	hroughput						
Compute (SM) [%]												
Compute (Sivi) [8]										<u> </u> ,		
Memory [%]						■						
Ċ	D.O 10.0	20	1.0 3	30.0	40.0	50.0	60.0	70.0)	80.0	90.0	100.0
					S	peed Of Light (SOI	_) [%]					
Compute Worklo	ad Analysis											Ω
xecuted Ipc Elapse	d [inst/cycle]				0.00 (-99.16%)	SM Busy [%]						34.72 (-59.93%)
executed Ipc Active					0.26 (-20.53%	Issue Slots Bus	y [%]					7.11 (-14.71%)
ssued Ipc Active [in	st/cycle]				0.28 (-14.71%)							
Balanced 1	No pipeline is over-utilized	l.										
 Memory Workloa 	id Analysis										All	÷ Q
Memory Throughput	t [Gbyte/second]				1.84 (-99.38%							0.80 (-94.49%)
1/TEX Hit Rate [%]					0 (-100.00%							0.57 (-98.75%)
2 Hit Rate [%]				71.2	25 (+43,797.06%)	Mem Pipes Bus	y [%]					0.11 (-99.15%)
					Mem	ory Chart						
			16	.00 Req							1000	
	17.00 Inst										🚤 🗖 100%	
	17.00 Inst (-100.00%)		(-1	00.00%)							ea	
_		Glot	bal	,	•				0.00 B (+0.00%)	nory	% Leak	
		Glob	bal (-1	00.00%) .00 Req 99.80%)	*			-	0.00 B (+0.00%)	Memory	% Pea	
_	(-100.00%)	Glot	(-1)	.00 Req 99.80%) .00 Req	*			-	(+0.00%)	L Lettern Memory	69 % % %	
		_	(-1) bal 1. (-5 0. (+	.00 Req 99.80%)	*			-		System Memory		
	(-100.00%)	Glot	(-1) bal 1. (-5 0. (+ cal	.00 Req 99.80%) .00 Req	► L1/TEX Cache	4.00 K (-99.99		-	(+0.00%) 0.00 B	System Memory		



ROOFLINE ANALYSIS

- Determine whether the application is memory bound or compute bound
- Guided analysis points to detailed analysis of the most severe problem



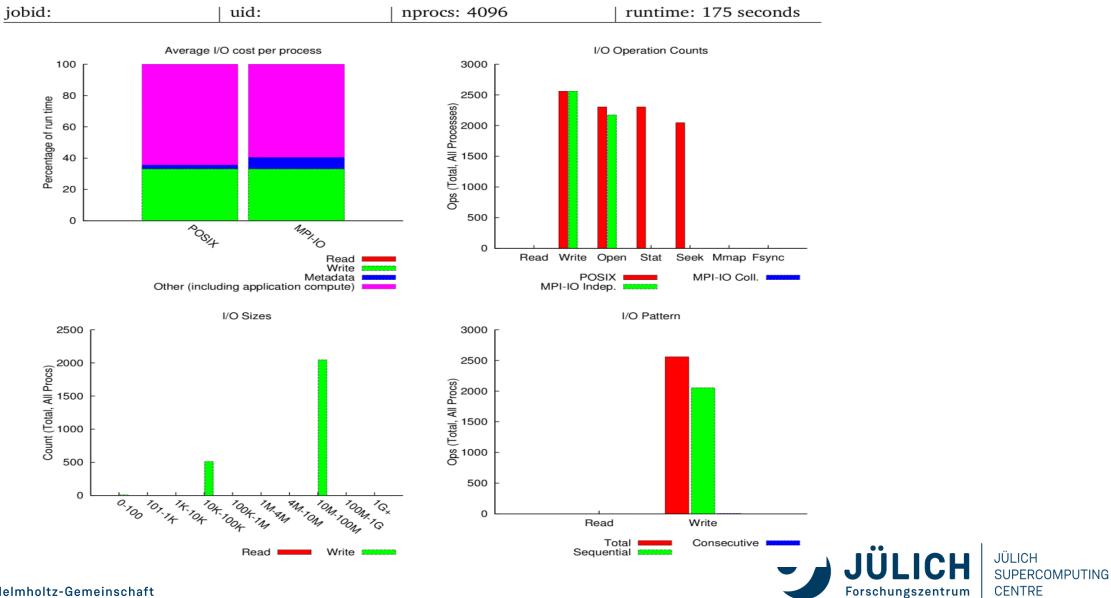


DARSHAN

- I/O characterization tool logging parallel application file access
- Summary report provides quick overview of performance issues
- Works on unmodified, optimized executables
- Shows counts of file access operations, times for key operations, histograms of accesses, etc.
- Supports POSIX, MPI-IO, HDF5, PnetCDF, ...
- Binary log file written at exit post-processed into PDF report
- http://www.mcs.anl.gov/research/projects/darshan/
- Open Source: installed on many HPC systems



EXAMPLE DARSHAN REPORT EXTRACT



Mitglied der Helmholtz-Gemeinschaft

PERFORMANCE ANALYSIS RECOMMENDATIONS

- Measure and analyze at the desired scale (once you have a reasonable measurement setup)
- Get performance overview with Performance Reports
 - CPU Issues:
 - Use Vtune (on Intel nodes) or uProf (on AMD nodes)
 - Use perf / LIKWID / PAPI
 - MPI Issues: Use Scalasca/Vampir
 - GPU Issues: Use NVIDIA tools
 - I/O Issues: Use DARSHAN
- OR: Do it all with Score-P/Scalasca/Vampir



NEED HELP?

- Talk to the experts
 - Use local 1st-level support, e.g. SC support, SimLab
 - Use mailing lists
 - JSC/NVIDIA Application Lab
 - ATML Parallel Performance
 - ATML Application Optimization and User Service Tools

© Successful performance engineering often is a collaborative effort





QUESTIONS



Mitglied der Helmholtz-Gemeinschaft