TOOLS FOR GPU COMPUTING

With focus on NVIDIA GPUs

03.12.2019 | MICHAEL KNOBLOCH

MOTIVATION

Make it work,
make it right,
make it fast.

Kent Beck

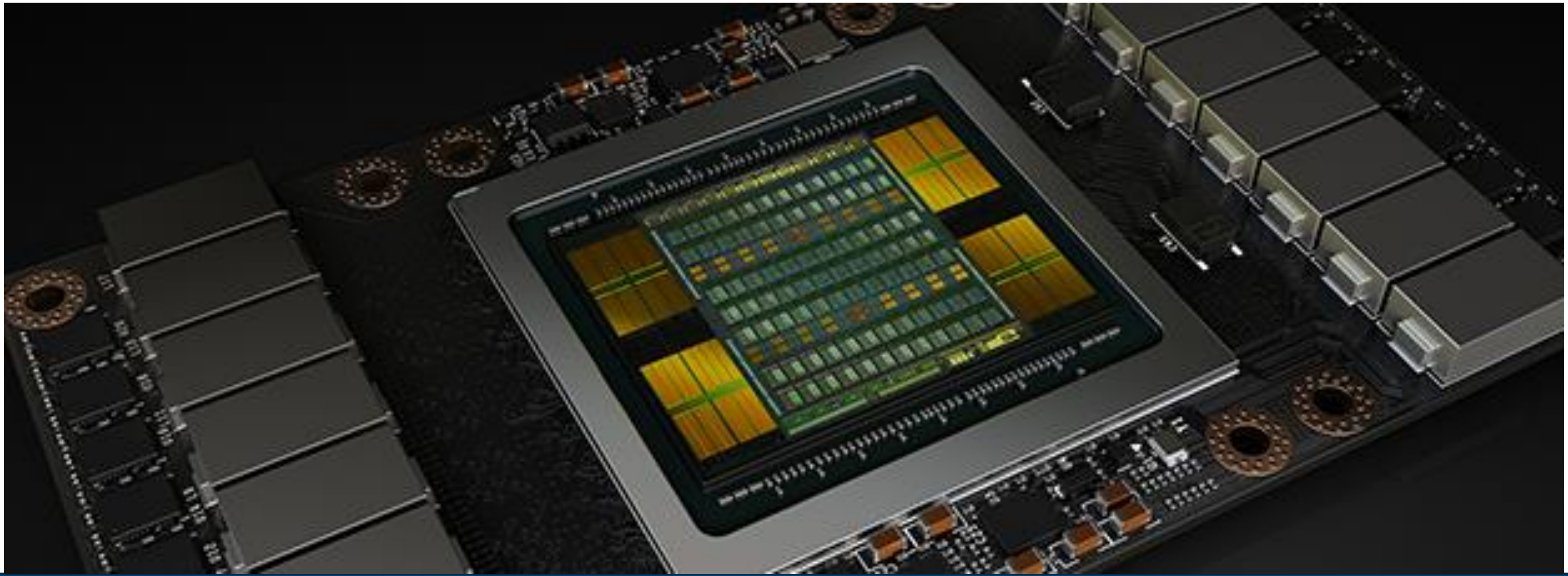
- IDEs
- Porting
- Libraries

Debugger:

- CUDA-MEMCHECK
- CUDA-GDB
- TotalView
- DDT

Performance Tools:

- NVIDIA Visual Profiler
- NVIDIA Nsight System
- NVIDIA Nsight Compute
- Score-P
- Vampir
- Performance Reports
- TAU
- HPCToolkit



GPU PROGRAMMING MODELS

CURRENT STATE OF THE MESS

TRADITIONAL HPC

- **Inter-node:**
  - MPI
- **Intra-node:**
  - OpenMP
  - Pthreads

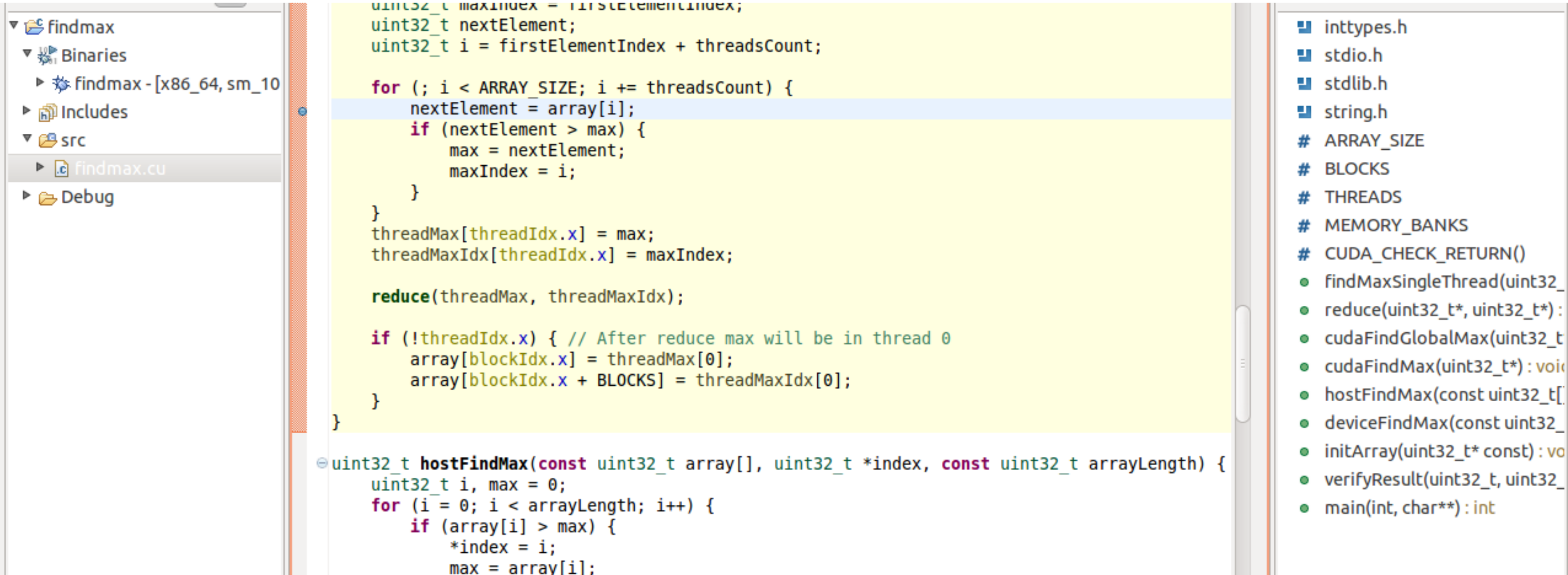
- C/C++ and Fortran

MODERN HPC

- **Inter-node:**
  - MPI
  - PGAS (SHMEM, GASPI, …)
- **Intra-node:**
  - OpenMP
  - Pthreads
  - Tasking, C++11 threads, TBB, …
- C/C++, Fortran and Python

GPU PROGRAMMING

- **Low-level:**
  - CUDA (NVIDIA), ROCm (AMD)
  - OpenCL
- **Pragma-based:**
  - OpenACC
  - OpenMP target
- On top: SYCL, oneAPI, HIP, KOKKOS, …



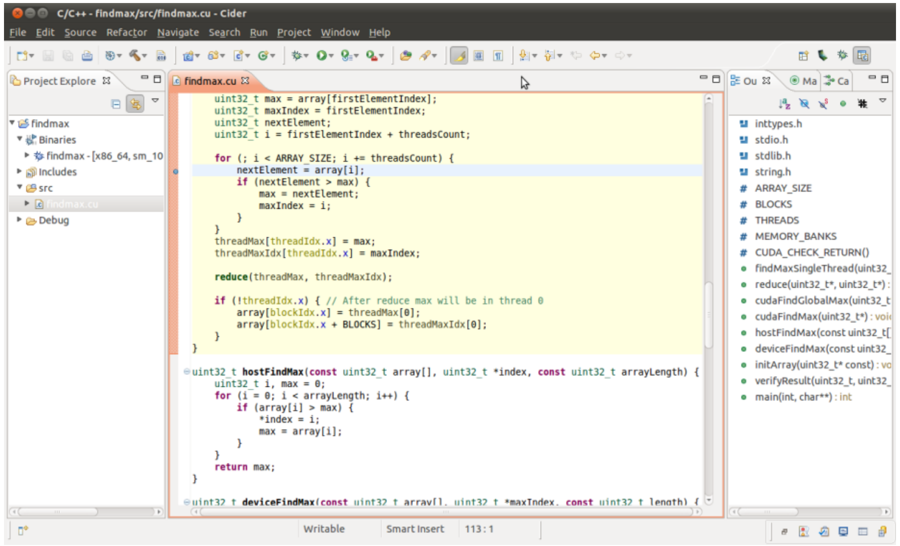
MAKE IT WORK

DEVELOPMENT OF GPU APPS

IDE

Integrated Development Environment

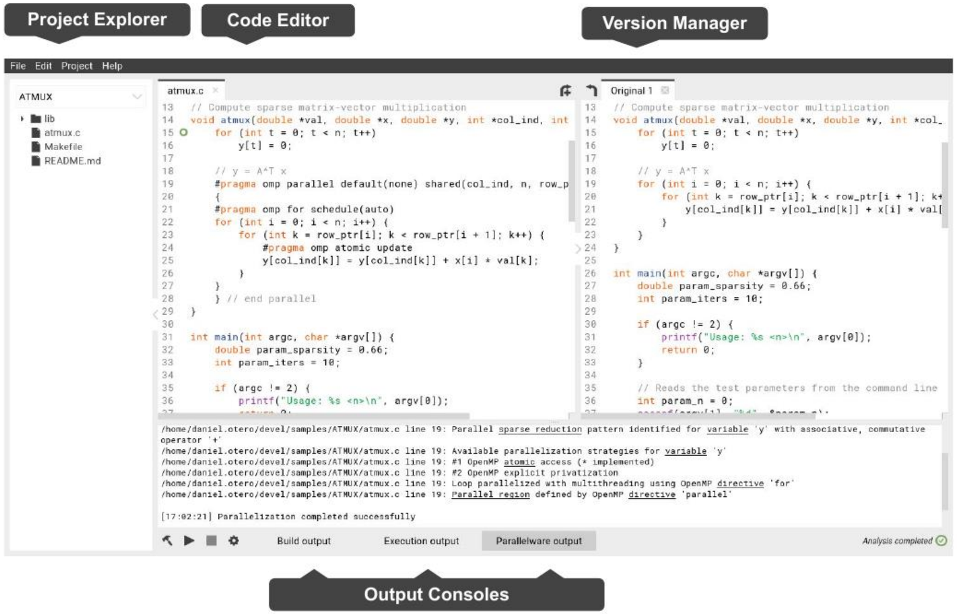
- Integrates Editor, Build system, Debugger, and Profiler
- NVIDIA Nsight (Linux: Eclipse, Windows: Visual Studio)
- Nsight Code Editor
  - CUDA aware code completion and inline help
  - CUDA code highlighting
  - CUDA aware refactoring

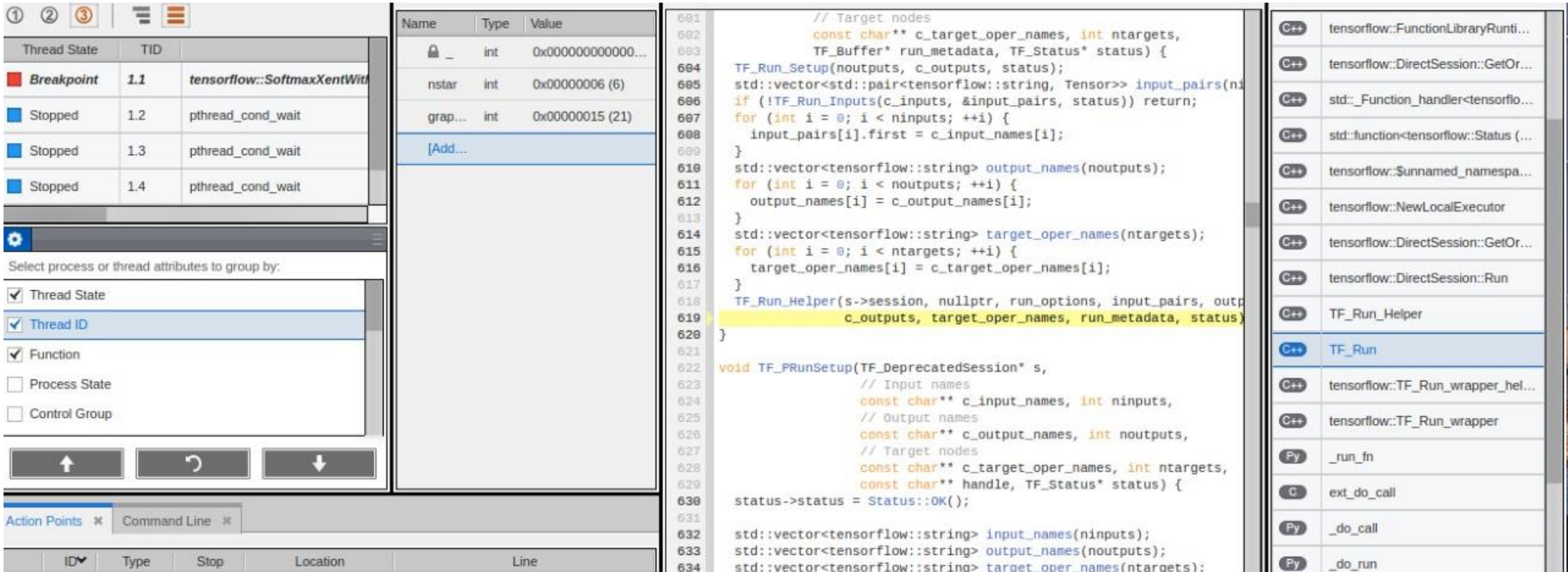


Mitglied der Helmholtz-Gemeinschaft

PORTING

- Several tools exist helping expose parallelism
- Example: Appentra Parallelware Trainer
  - Identifies parallelizable sections in sequential applications
  - Supports OpenMP and OpenACC
  - Supports versioning of changes
  - Start program directly from GUI





MAKE IT RIGHT

DEBUGGER AND MEMORY ANALYZER

DEBUGGER COMPATIBILITY MATRIX

Tool	CUDA	OpenACC	OMPD	OpenCL
CUDA-MEMCHECK	✓	*	✗	✗
CUDA-GDB	✓	*	✗	✗
TotalView	✓	✓	**	✗
DDT	✓	✓	✗	✗

* = Indirect support via CUDA (Nvidia only)

** = Prototype with non-public OMP(D) runtime

CUDA-MEMCHECK

- Valgrind for GPUs
- Monitors hundreds of thousands of threads running concurrently on each GPU
- Reports detailed information about global, local, and shared memory access errors (e.g. out-of-bounds, misaligned memory accesses)
- Reports runtime executions errors (e.g. stack overflows, illegal instructions)
- Reports detailed information about potential race conditions
- Displays stack back-traces on host and device for errors
- And much more

- Included in the CUDA Toolkit

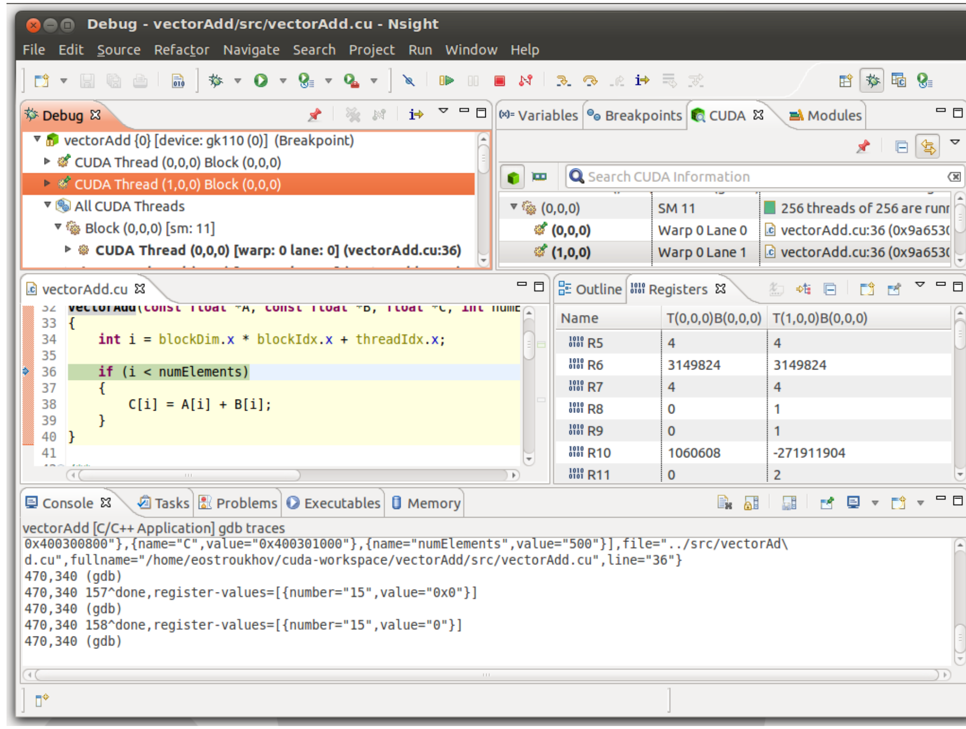
Applications Places System

File Edit View Terminal Help

```
linux64:~/demo2010$ ./ptrchecktest
unspecified launch failure : 79
linux64:~/demo2010$ cuda-memcheck ./ptrchecktest
========= CUDA-MEMCHECK
unspecified launch failure : 79
========= Invalid __global__ read of size 4
=========     at 0x00000158 in ptrchecktest.cu:27:kernel2
=========     by thread (0,0,0) in block (0,0)
=========     Address 0xfd00000001 is misaligned
=========
========= ERROR SUMMARY: 1 error
linux64:~/demo2010$ cuda-memcheck --continue ./ptrchecktest
========= CUDA-MEMCHECK
Checking...
Done
Checking...
Error: 3 (0)
Done
Checking...
Error: 1 (0)
Error: 3 (0)
Error: 5 (0)
Error: 7 (0)
Done
========= Invalid __global__ read of size 4
=========     at 0x00000158 in ptrchecktest.cu:27:kernel2
=========     by thread (0,0,0) in block (0,0)
=========     Address 0xfd00000001 is misaligned
=========
========= Invalid __global__ read of size 4
=========     at 0x00000198 in ptrchecktest.cu:18:kernel1
=========     by thread (3,0,0) in block (5,0)
=========     Address 0xfd00000028 is out of bounds
=========
========= Invalid __global__ write of size 8
=========     at 0x000001d0 in ptrchecktest.cu:38:kernel3
=========     by thread (1,0,0) in block (8,0)
=========     Address 0xfd00000204 is misaligned
=========
========= Invalid __global__ write of size 4
=========     at 0x000000f0 in ptrchecktest.cu:44:kernel4
=========     by thread (63,0,0) in block (22,0)
=========     Address 0x00000000 is out of bounds
=========
========= ERROR SUMMARY: 4 errors
```

CUDA-GDB

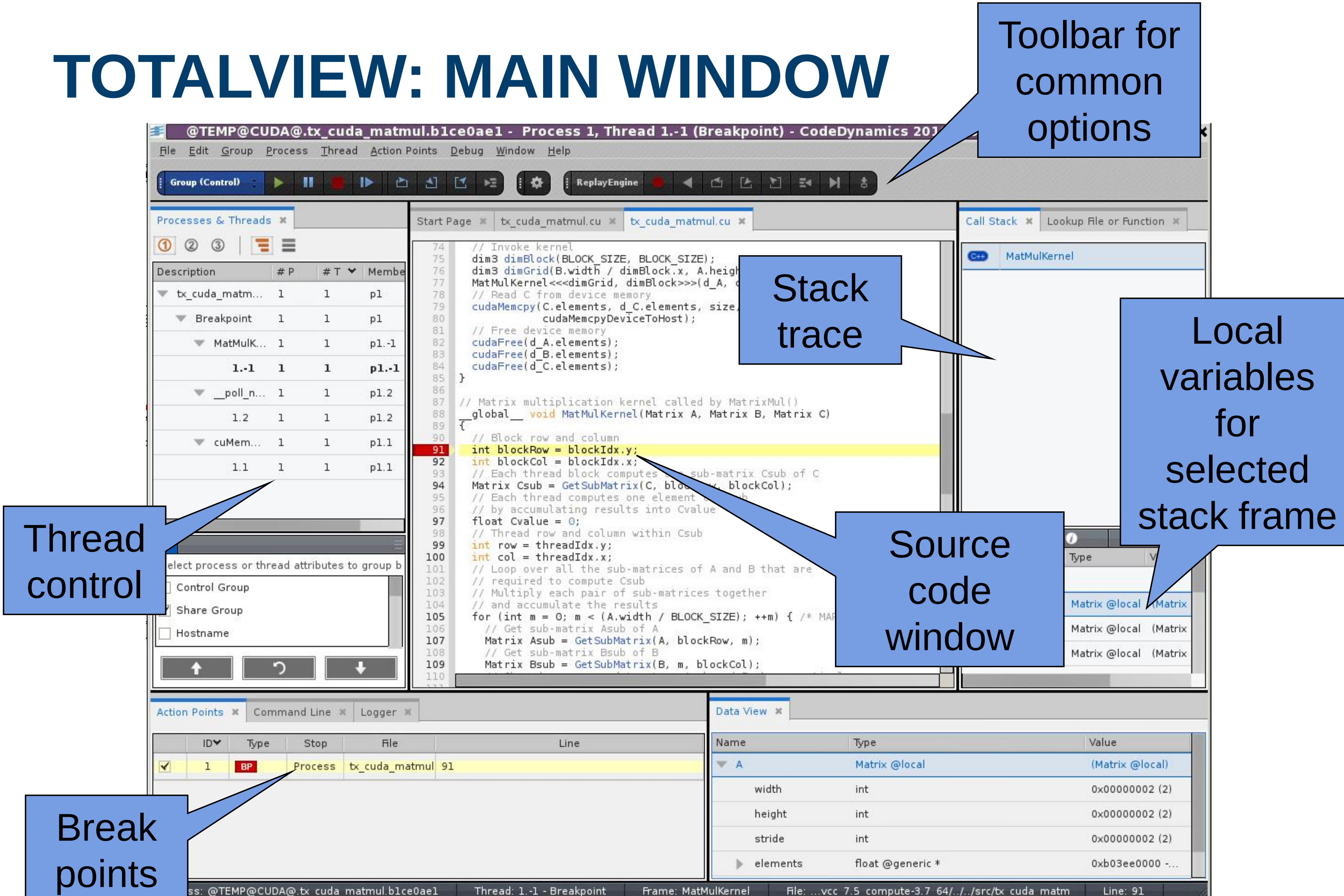
- Extension to gdb
- CLI and GUI (Nsight)
- Simultaneously debug on the CPU and multiple GPUs
- Use conditional breakpoints or break automatically on every kernel launch
- Can examine variables, read/write memory and registers and inspect the GPU state when the application is suspended
- Identify memory access violations
  - Run CUDA-MEMCHECK in integrated mode to detect precise exceptions.



TOTALVIEW

- UNIX Symbolic Debugger
  for C/C++, Fortran, Python, PGI HPF, assembler programs
- JSC's "standard" debugger
- Special, non-traditional 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
  - CUDA and OpenACC support
- http://www.roguewave.com

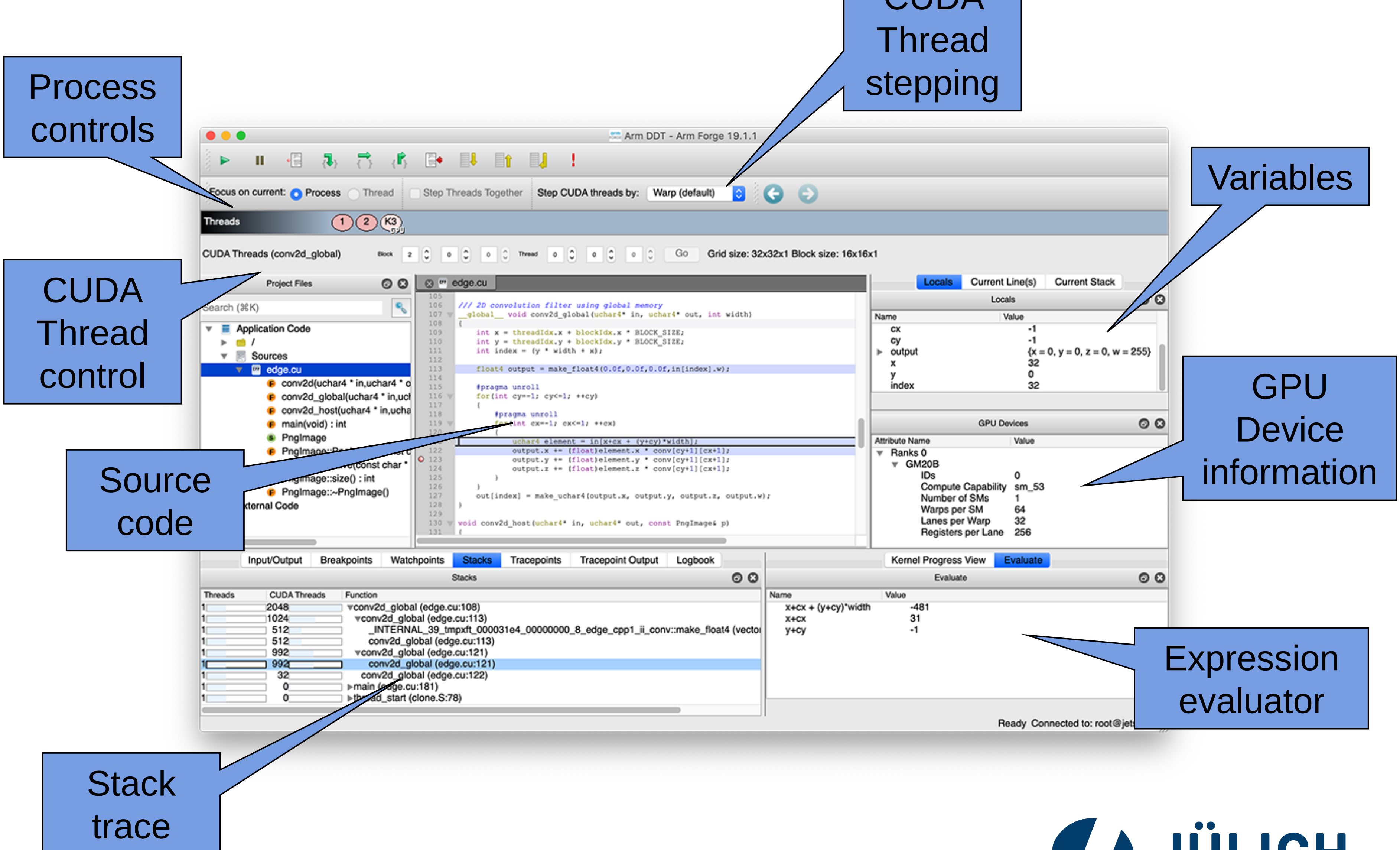
TOTALVIEW: MAIN WINDOW



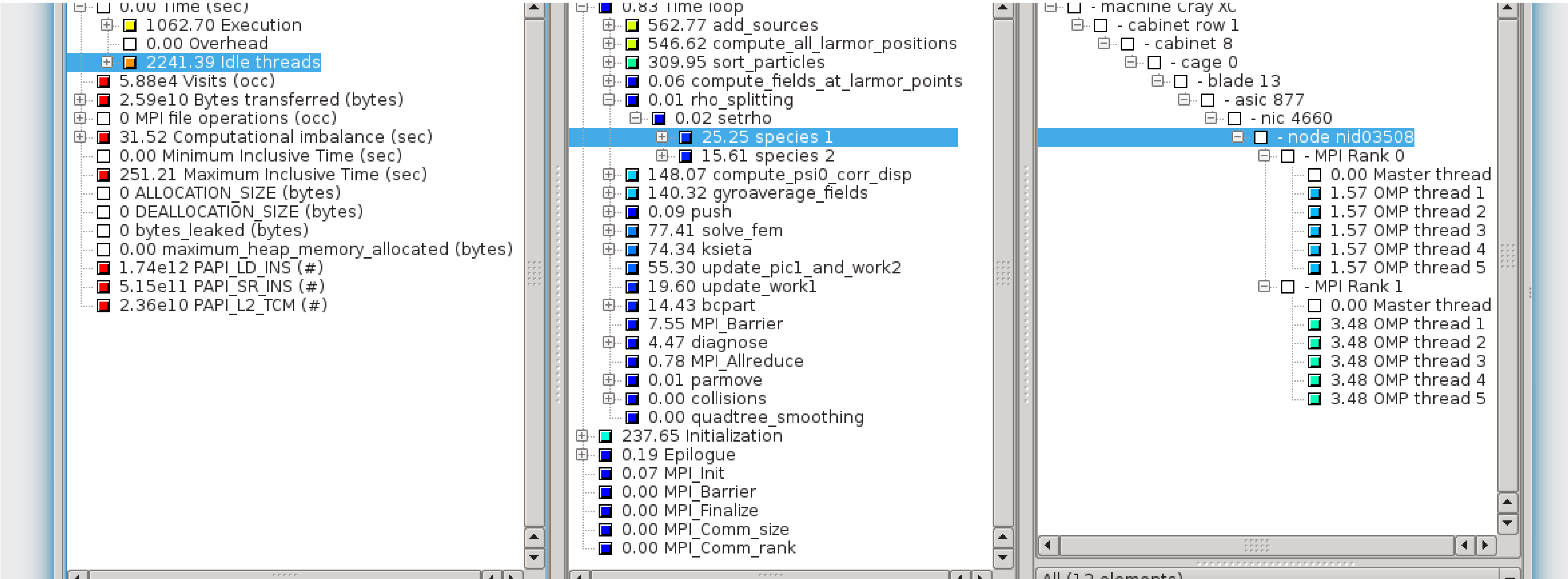
DDT arm FORGE

- UNIX Graphical Debugger for C/C++, Fortran, and Python programs
- Modern, easy-to-use debugger
- Special, non-traditional 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://developer.arm.com

DDT: MAIN WINDOW



JÜLICH Forschungszentrum



MAKE IT FAST

PERFORMANCE ANALYSIS TOOLS

PERF TOOL COMPATIBILITY MATRIX

Tool	CUDA	OpenACC	OMPT	OpenCL
Score-P	✓	✓	**	✓
NVIDIA Tools	✓	✓	✗	✗
Perf. Reports	✓	*	✗	✗
TAU	✓	✓	**	✓
HPCToolkit	✓	✗	**	✗
Extrae	✓	✗	✗	✓

* = Indirect support via CUDA (Nvidia only)

** = Prototype with non-public OMP(T) runtime

ARM PERFORMANCE REPORTS

- Single page report provides quick overview of performance issues
- Works on unmodified, optimized executables
- Shows CPU, GPU, memory, network and I/O utilization

- Supports MPI, multi-threading and accelerators
- Saves data in HTML, CVS or text form

- https://www.arm.com/products/development-tools/server-and-hpc/performance-reports

EXAMPLE PERFORMANCE REPORTS

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

The total wallclock time was spent as follows:

CPU	56.5%	Time spent running application code. High values are usually good. This is **average**; check the CPU performance section for optimization advice.
MPI	43.5%	Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it.
I/O	0.0%	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%
Memory accesses	60.9%
Other	0.0

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.

MPI

Of the 43.5% total time spent in MPI calls:

Time in collective calls	8.2%
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.

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!

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.

PERFORMANCE REPORTS ACCERLERATOR

Accelerators

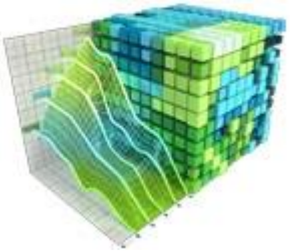
A breakdown of how accelerators were used:

GPU utilization	47.8%
Global memory accesses	1.6%
Mean GPU memory usage	0.8%
Peak GPU memory usage	0.8%

GPU utilization is low; identify CPU bottlenecks with a profiler and offload them to the accelerator.

The peak GPU memory usage is low. It may be more efficient to offload a larger portion of the dataset to each device.

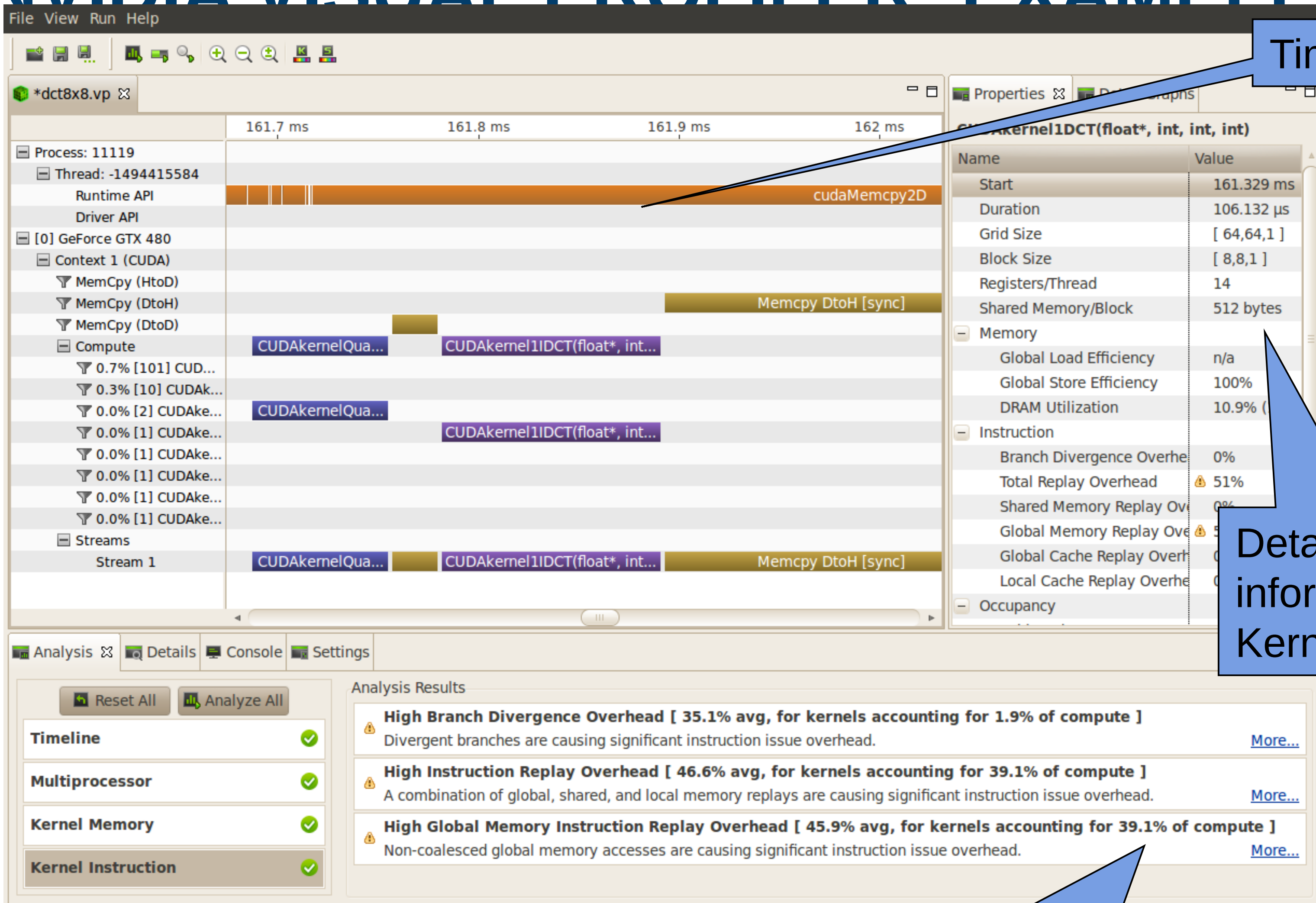
NVIDIA VISUAL PROFILER



- Part of the CUDA Toolkit
- Supports all CUDA enabled GPUs
- Supports CUDA and OpenACC on Windows, OS X and Linux

- Unified CPU and GPU Timeline
- CUDA API trace
  - Memory transfers, kernel launches, and other API functions
- Automated performance analysis
  - Identify performance bottlenecks and get optimization suggestions
- Guided Application Analysis
- Power, thermal, and clock profiling

NVIDIA VISUAL PROFILER: EXAMPLE



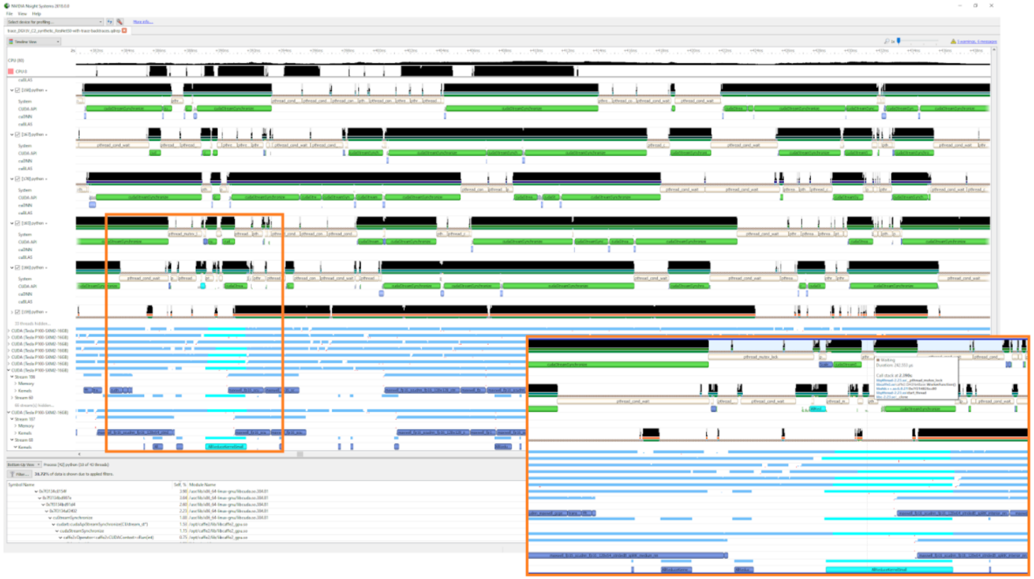
Timeline view

Detailed information on Kernel execution

Automatic analysis of performance bottlenecks

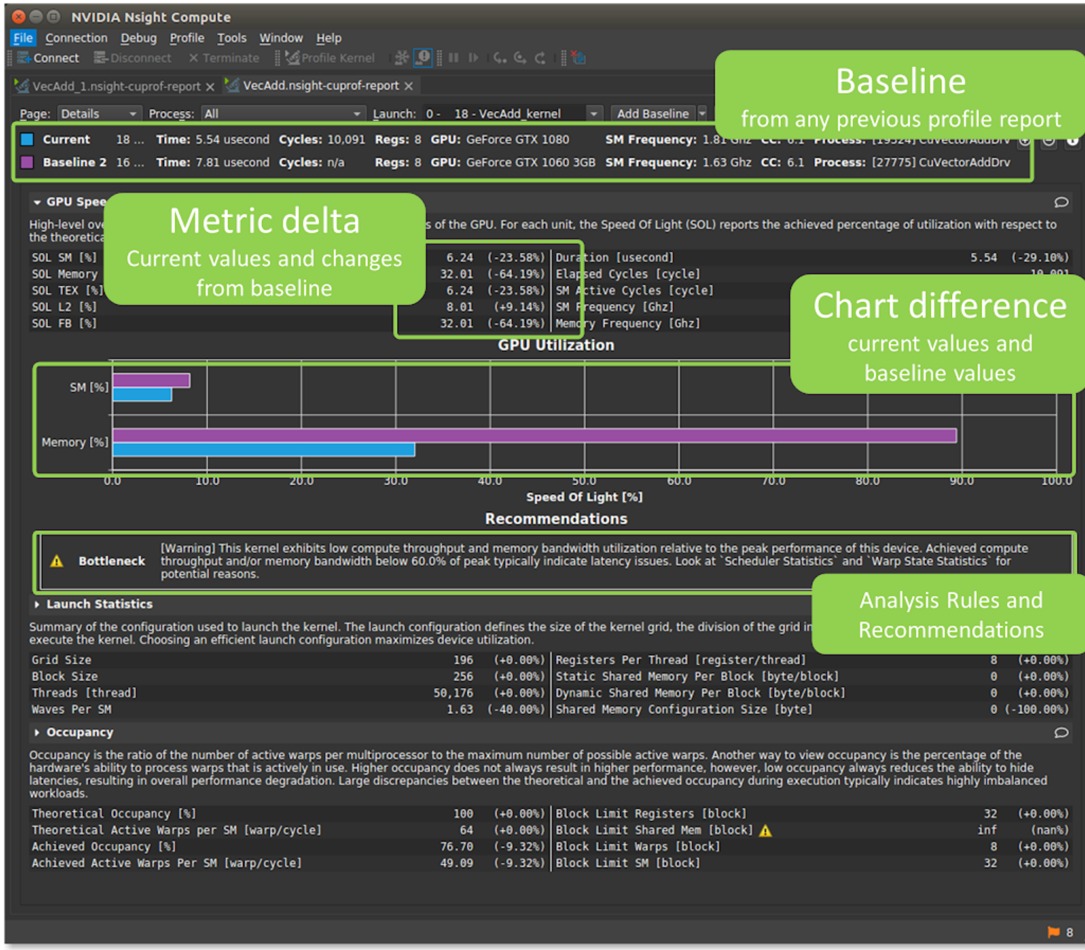
NVIDIA NSIGHT SYSTEMS

- System wide performance analysis tool
- High-level, low overhead
- Similar functionality as NVVP
    - No automated/guided analysis
    - Can launch Nsight Compute for in-depth kernel analysis
- CLI and GUI



NVIDIA NSIGHT COMPUTE

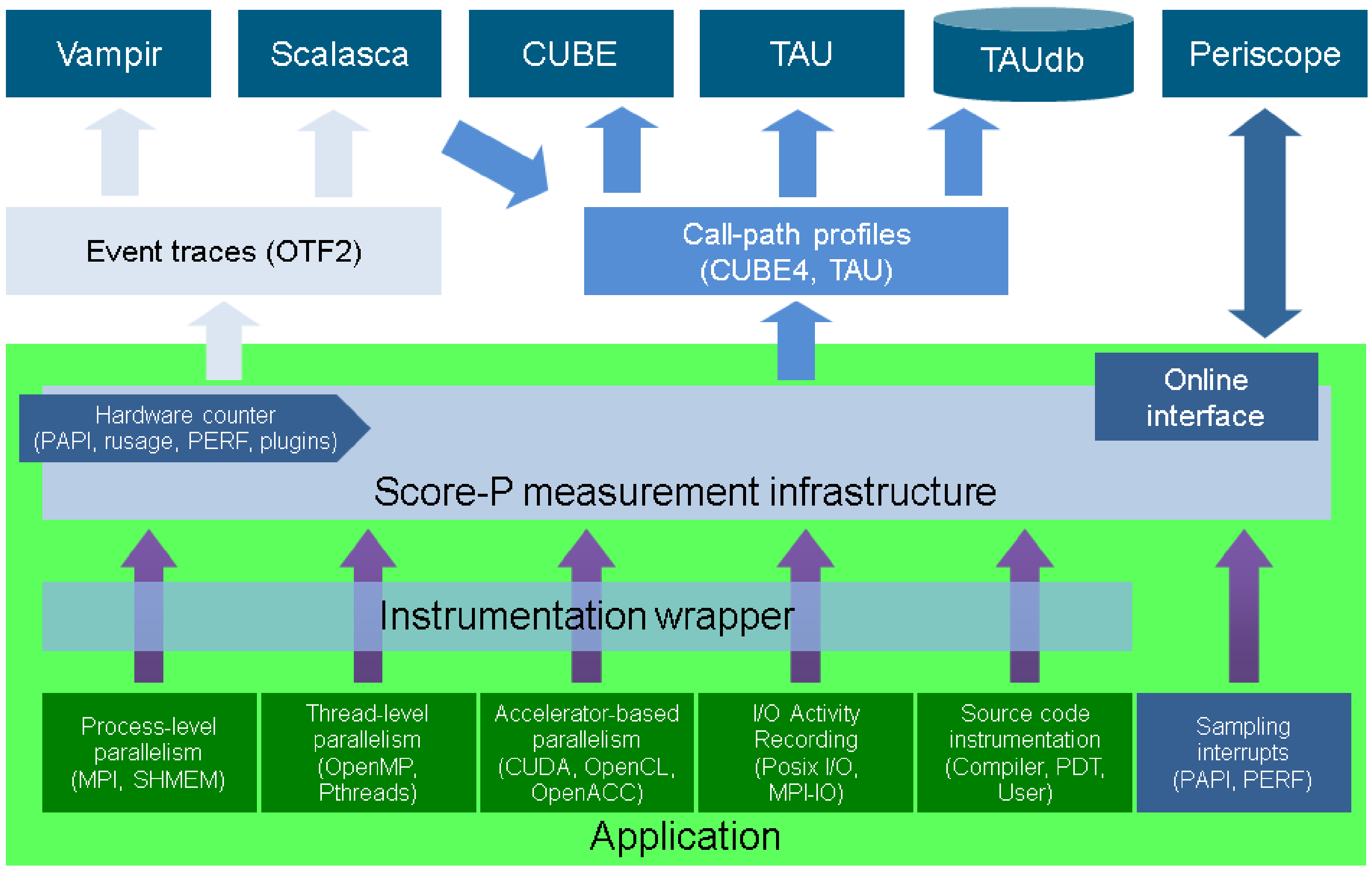
- Interactive kernel profiler
- Detailed performance metrics
- Guided analysis
- Baseline feature to compare versions
- Customizable and data-driven UI
- Supports analysis scripts for post-processing results
- CLI and GUI



SCORE-P

- Community instrumentation and measurement infrastructure
  - Developed by a consortium of performance tool groups
  - Next generation measurement system of
    - Scalasca 2.x
    - Vampir
    - TAU
    - Periscope
- Common data formats improve tool interoperability
- http://www.score-p.org

SCORE-P OVERVIEW

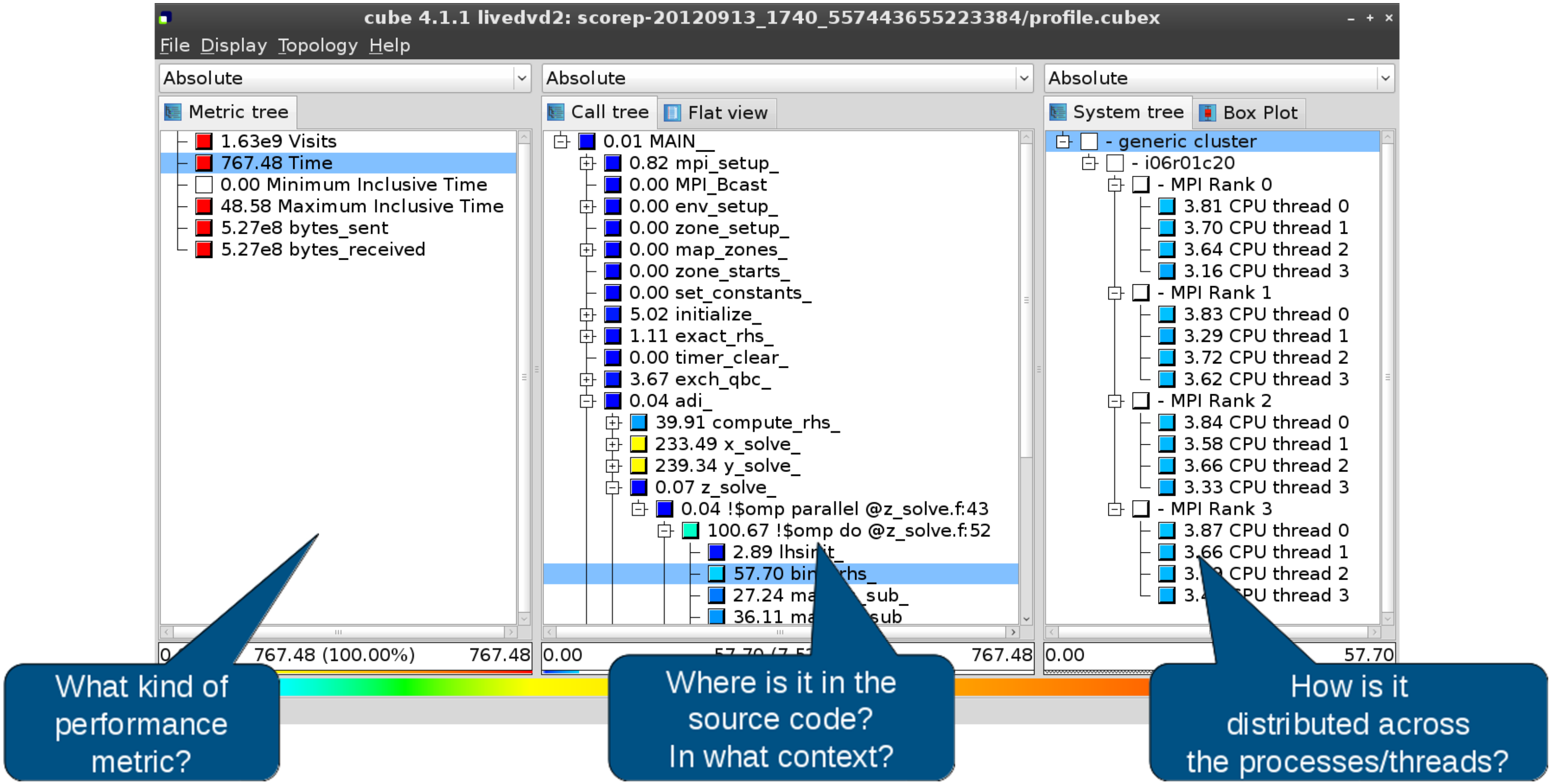


SCORE-P GPU MEASUREMENTS

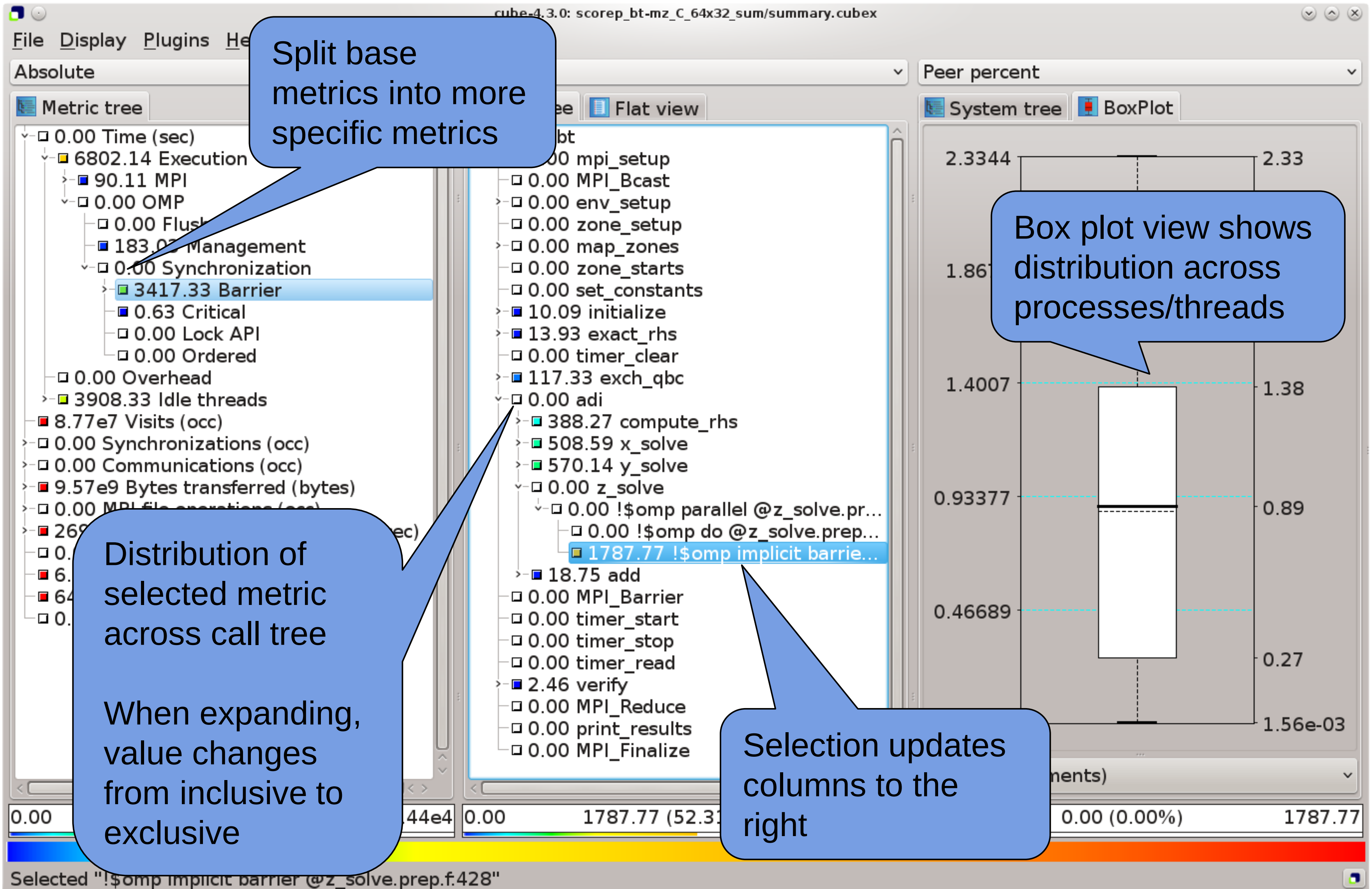
- OpenACC
  - Prefix compiler and linker command with scorep --openacc
  - export ACC_PROFLIB=$SCOREP_ROOT/lib/libscorep_adapter_openacc_event.so
  - export SCOREP_OPENACC_ENABLE=yes
  - yes refers to: regions, wait, enqueue
  - Full list of options in User Guide
- CUDA
  - Prefix compiler and linker command with scorep --cuda
  - export SCOREP_CUDA_ENABLE=yes
  - yes refers to: runtime, kernel, memcpy
  - Full list of options in User Guide

- OpenCL similar (use SCOREP_OPENCL_ENABLE=yes)

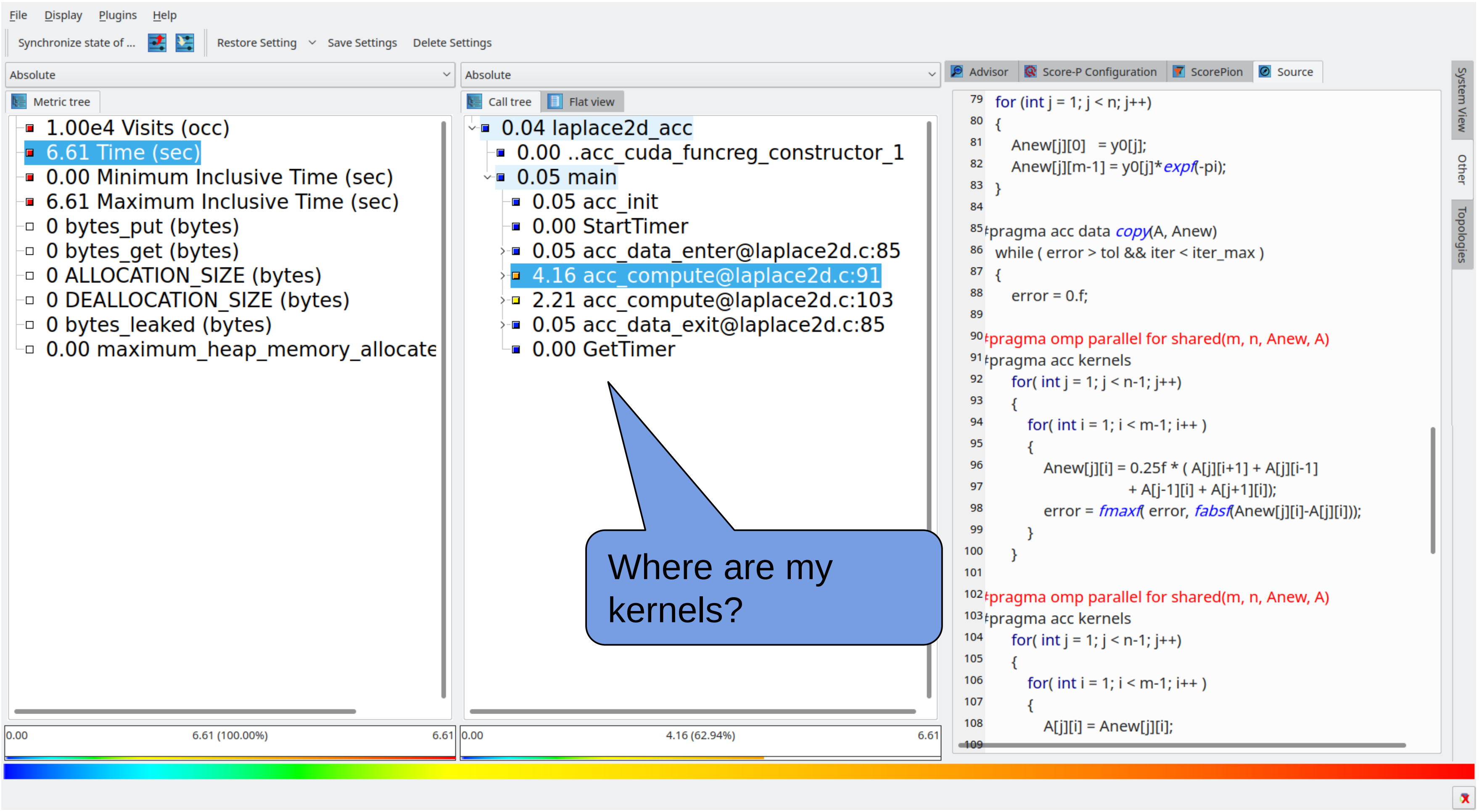
CUBE OVERVIEW



CUBE – OVERVIEW

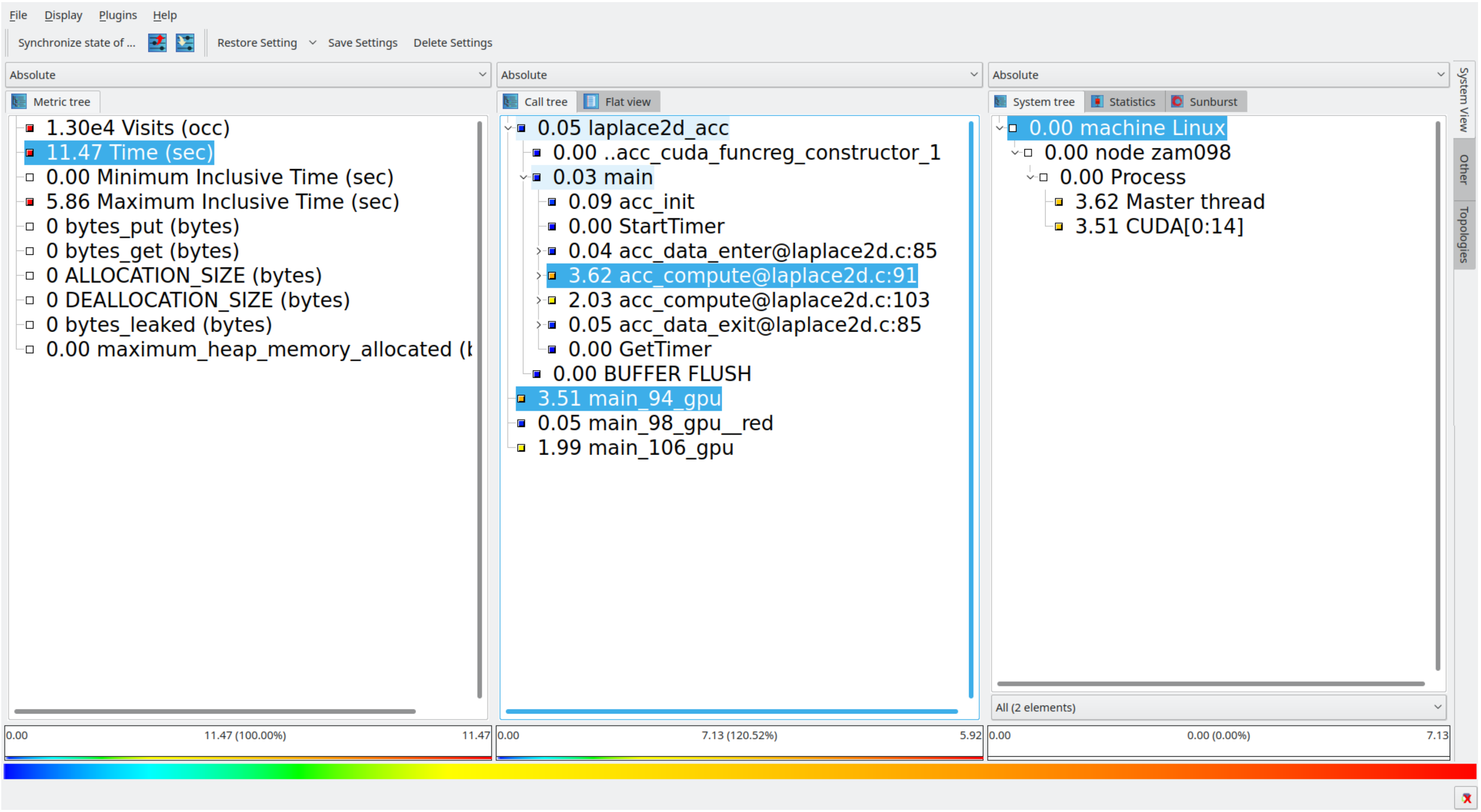


EXAMPLE: OPENACC



Pure OpenACC measurements give host-side events only

EXAMPLE: OPENACC + CUDA



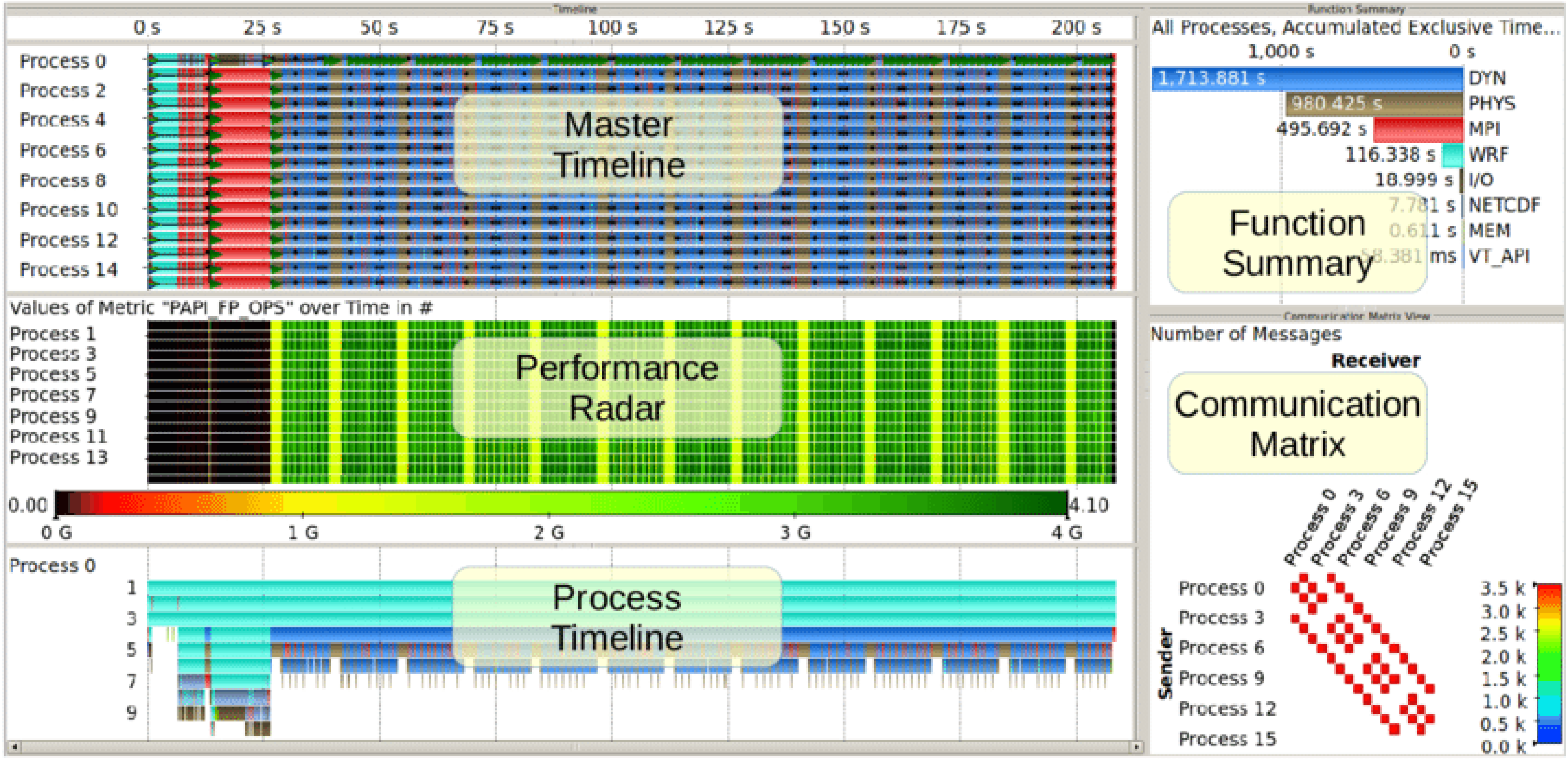
Enabling CUDA also shows kernels on the GPU

VAMPIR EVENT TRACE VISUALIZER

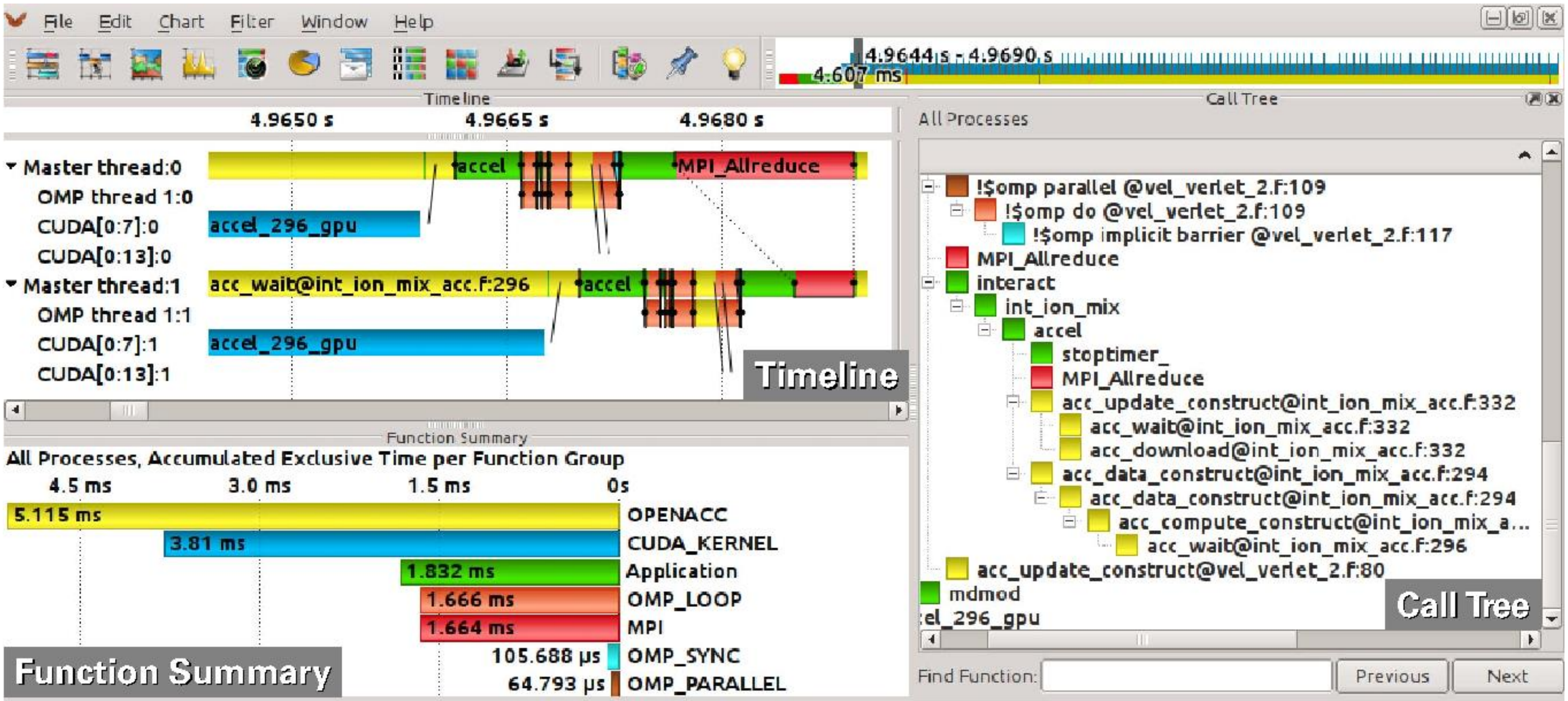
- Offline trace visualization for Score-P's OTF2 trace files
- Visualization of MPI, OpenMP, GPU and application events:
  - All diagrams highly customizable (through context menus)
  - Large variety of displays for ANY part of the trace
- http://www.vampir.eu

- Advantage:
  - Detailed view of dynamic application behavior
- Disadvantage:
  - Requires event traces (huge amount of data)
  - Completely manual analysis

VAMPIR DISPLAYS



VAMPIR COMPLEX APPLICATION



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, ...

WHAT NOW?

- The tools are there – what now?

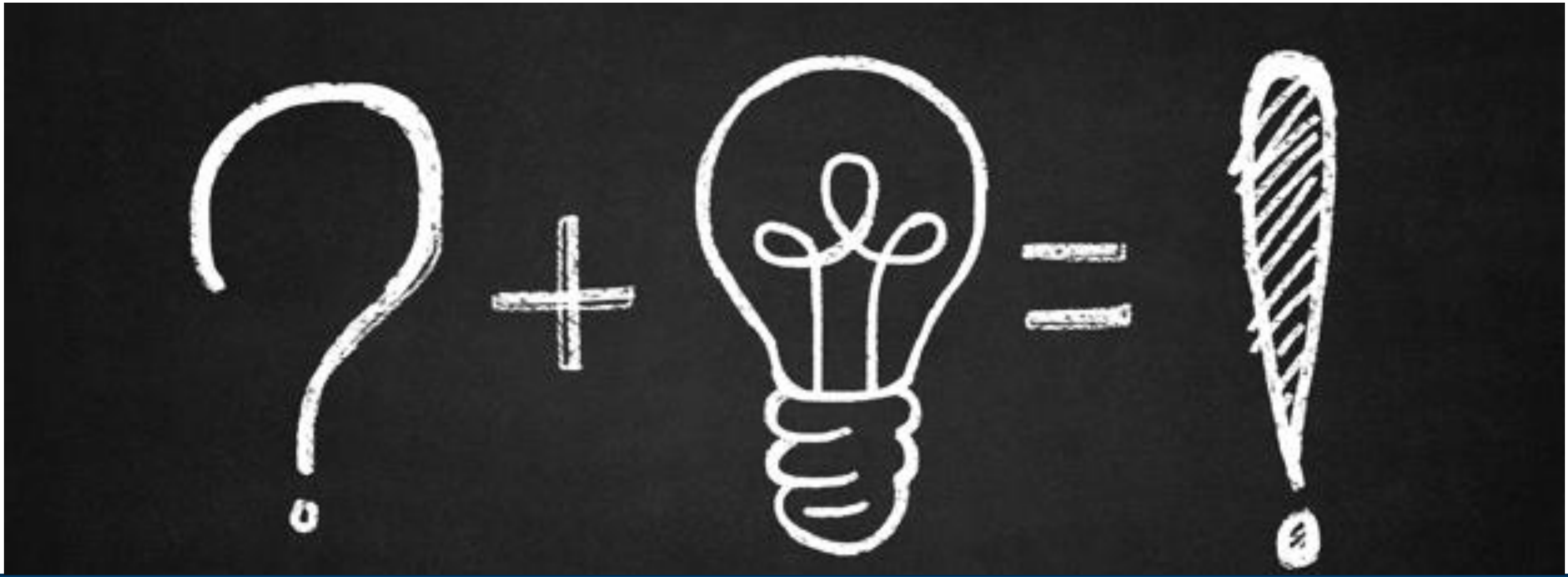
- Development phase:
  - Use NVDIA tools
  - Debug: CUDA-MEMCHECK/CUDA-GDB
  - Performance: Nsight Systems and Compute

- Scaling up:
  - Use 3rd-party tools
  - Debug: TotalView/DDT
  - Performance: Score-P, Vampir

NEED HELP?

- Talk to the experts
  - NVIDIA Application Lab
  - JSC team “Performance Analysis”
  - JSC team “Application Optimization”
  - Apply for a POP audit

☞ Successful performance engineering often is a collaborative effort



QUESTIONS