TECHNISCHE UNIVERSITÄT DRESDEN
Center for Information Services and High Performance Computing (ZIH)

In-Depth Performance Analysis for OpenACC/CUDA/OpenCL Applications with Score-P and Vampir

Hands-on-Lab @ GTC2015

Guido Juckeland (guido.juckeland@tu-dresden.de)

ZIH
Center for Information Services & High Performance Computing

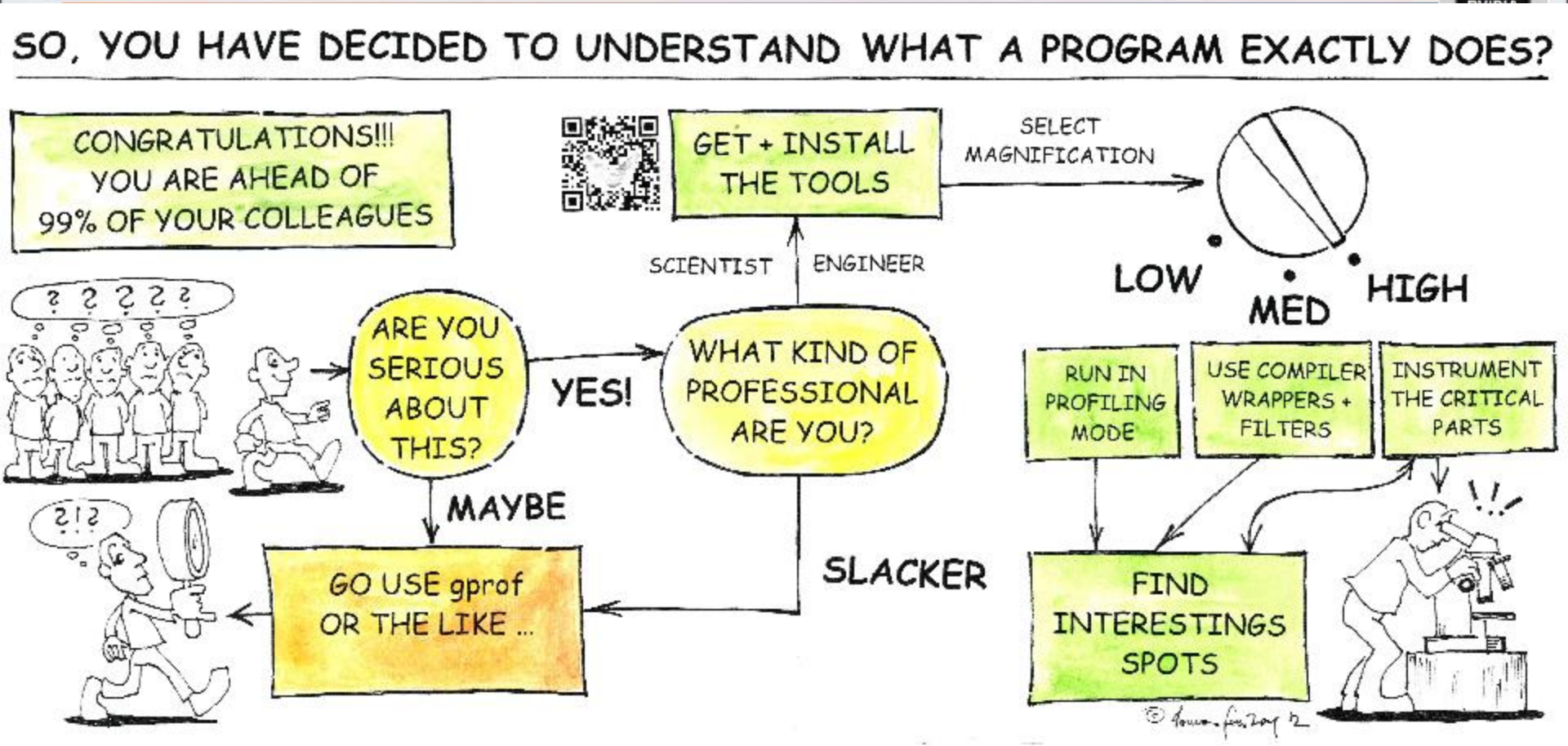
Agenda

- Motivation
- Performance Analysis 101
- Generating Traces with Score-P
- Visualizing Traces with Vampir
- Special Treat: OpenACC Tracing
- Looking a Little Deeper

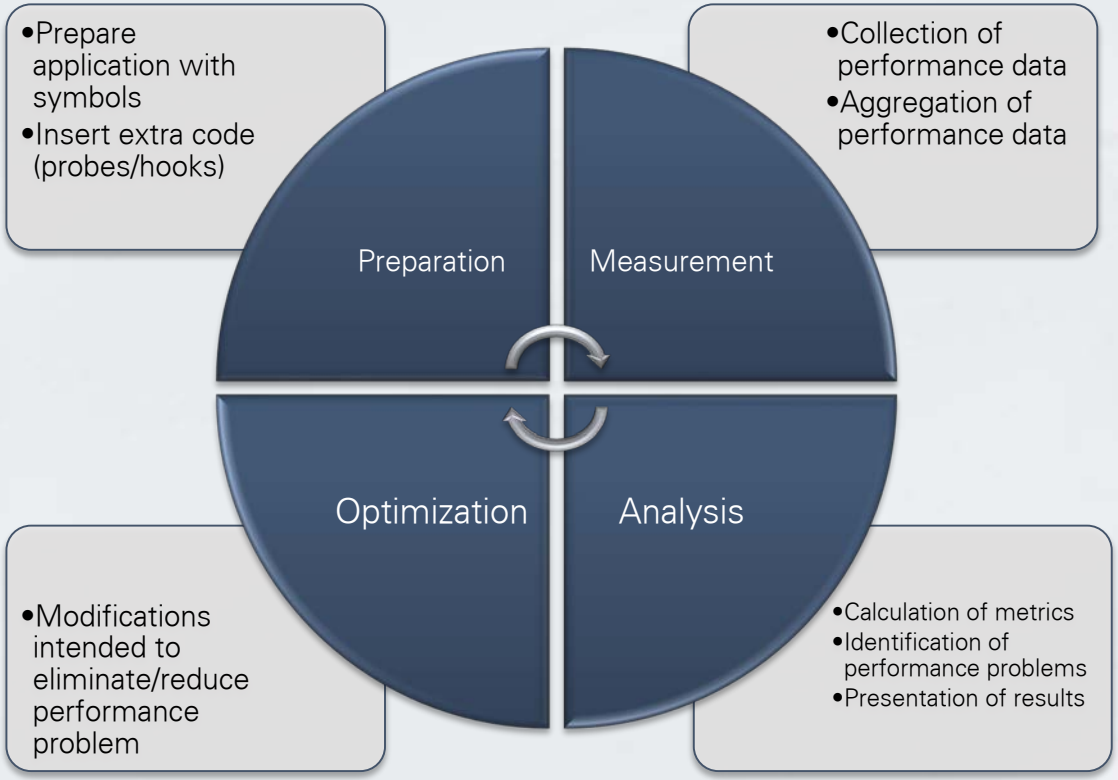
Motivation

Guido Juckeland (guido.juckeland@tu-dresden.de)

Why are you here?



Performance engineering workflow

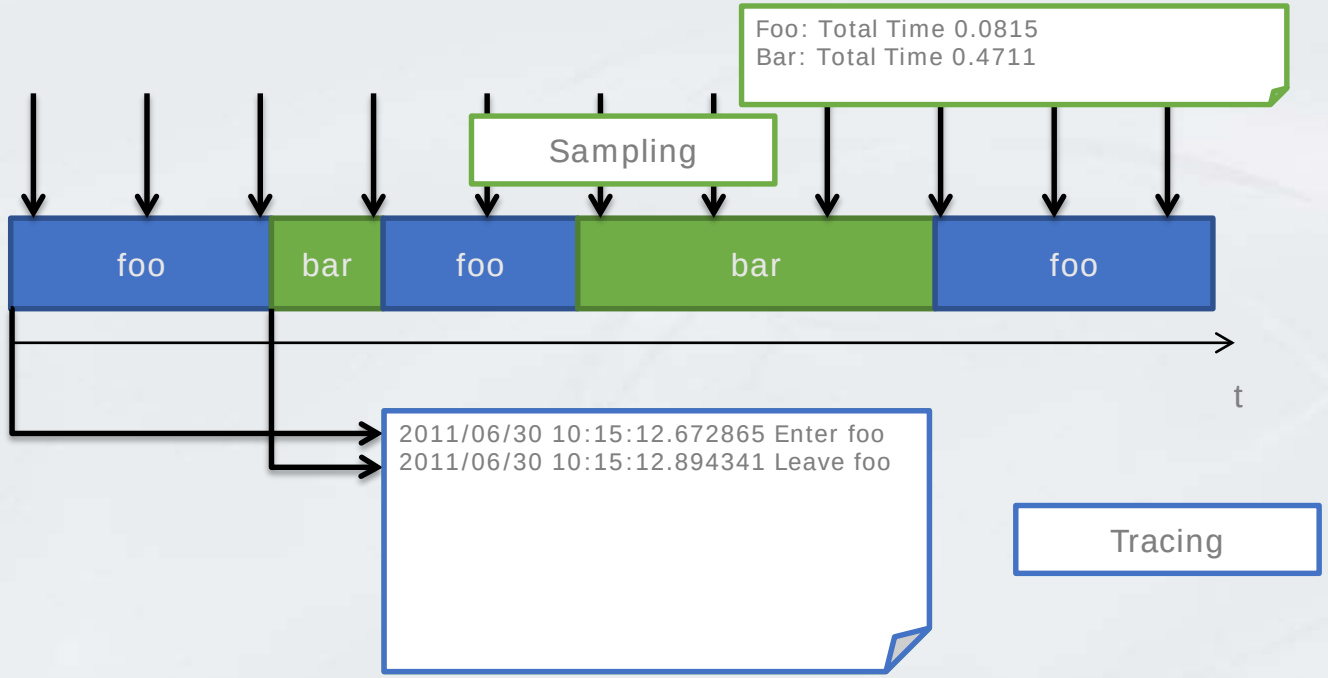


TECHNISCHE UNIVERSITÄT DRESDEN
Center for Information Services and High Performance Computing (ZIH)

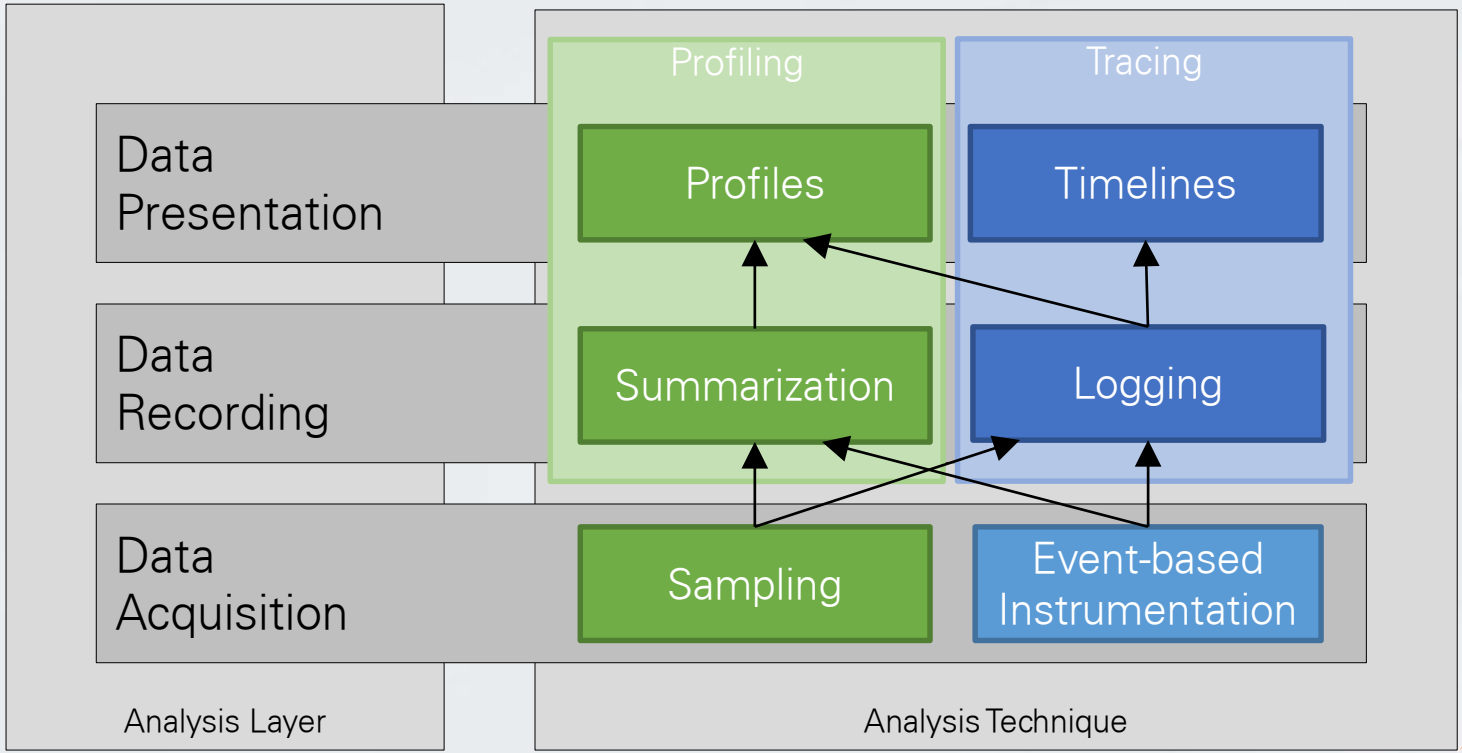
Performance Analysis 101

Guido Juckeland (guido.juckeland@tu-dresden.de)

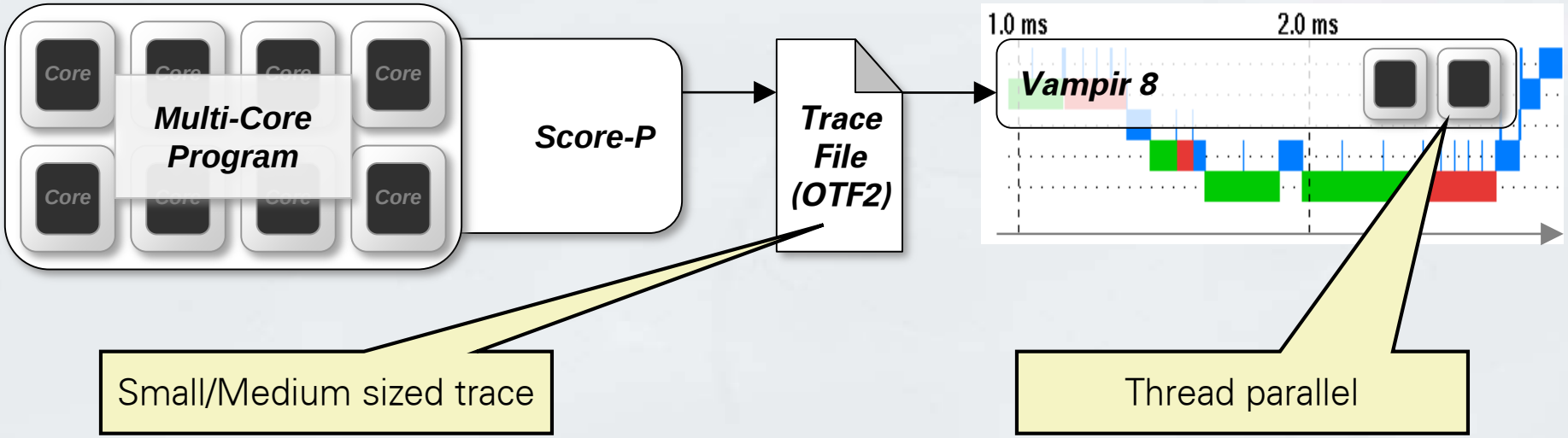
Sampling vs. Tracing



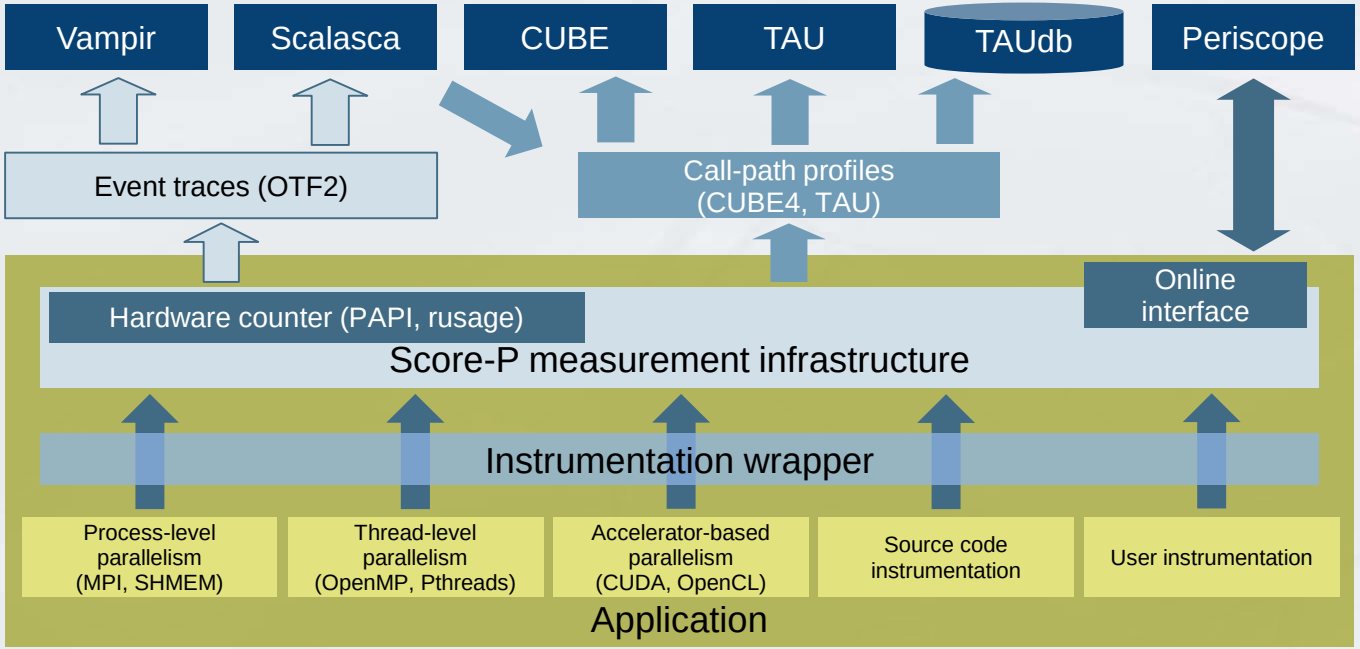
Terms Used and How They Connect



Score-P/Vampir Workflow for Small-Medium Sized Applications



Score-P Overview



Partners

- Forschungszentrum Jülich, Germany
- German Research School for Simulation Sciences, Aachen, Germany
- Gesellschaft für numerische Simulation mbH Braunschweig, Germany
- RWTH Aachen, Germany
- Technische Universität Dresden, Germany
- Technische Universität München, Germany
- University of Oregon, Eugene, USA

TECHNISCHE UNIVERSITÄT DRESDEN
Center for Information Services and High Performance Computing (ZIH)

Hands-on:
CUDA Tracing in Your Own AWS Instance

Guido Juckeland (guido.juckeland@tu-dresden.de)

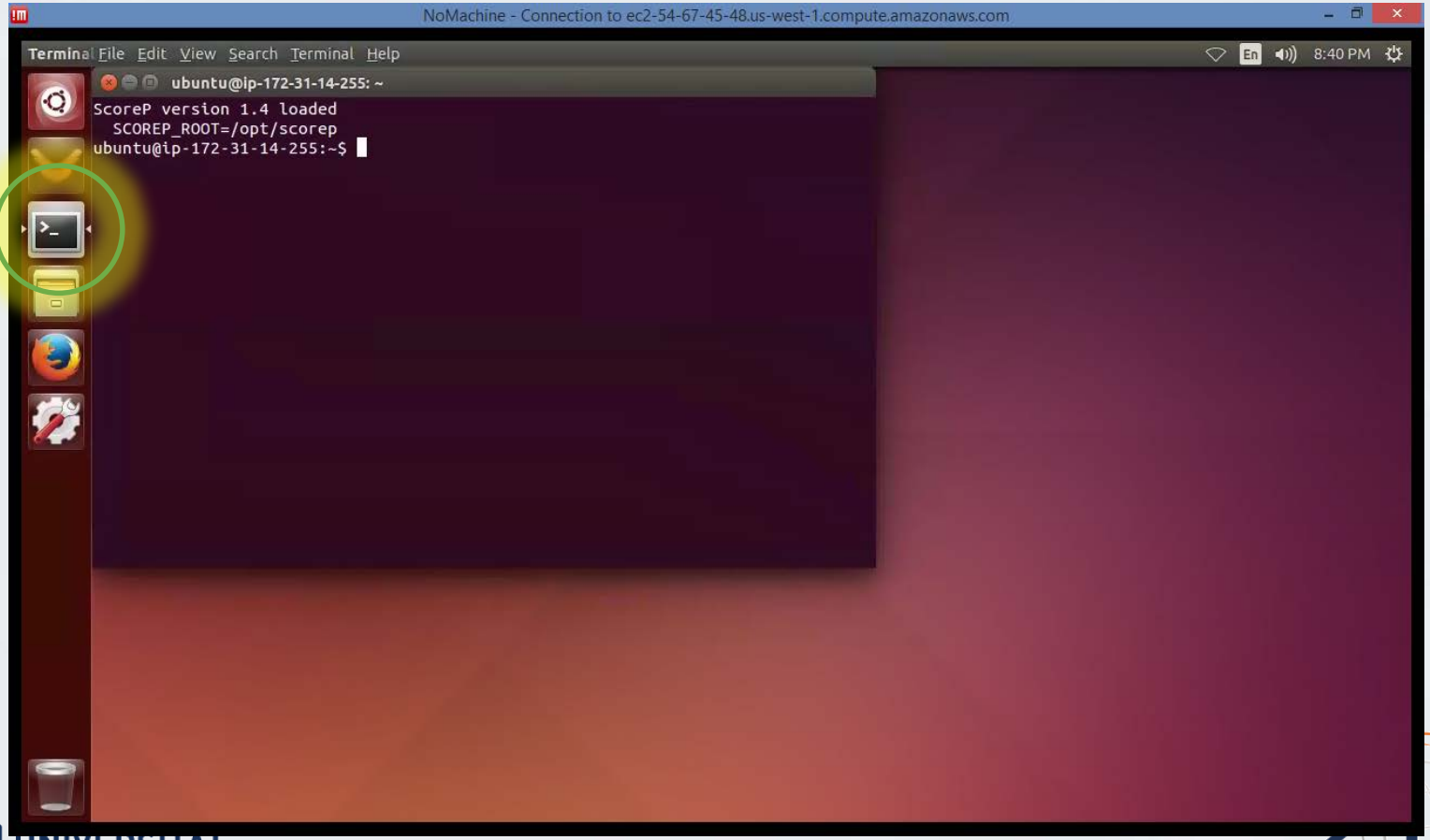
Connection Instructions

- Navigate to nvlabs.qwiklab.com
- Login or create a new account
- Select the "Instructor-Led Hands-on Labs" class
- Find the lab called "Analysis for OpenACC/CUDA/OpenCL Applications with Score-P and Vampir (S5721 - GTC 2015)" and click Start
- After a short wait, lab instance connection information will be shown
- Please ask Lab Assistants for help!

Performance Analysis Steps

1. Reference preparation for validation
2. **Program instrumentation**
3. **Event trace collection**
4. **Event trace examination & analysis**

Start a Terminal



Go to CUDA Example and Compile

- Go to CUDA Example

```
% cd codes/cuda
```

- Compile

```
% make
scorep --cuda /usr/local/anaconda/bin/mpicxx -Icommon/inc
-o simpleMPI_mpi.o -c simpleMPI.cpp
scorep --cuda "/usr/local/cuda-6.5"/bin/nvcc -ccbin g++ -Icommon/inc
-m64 -gencode arch=compute_30,code=sm_30 -gencode arch=compute_35,
code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode
arch=compute_50,code=sm_50 -gencode arch=compute_50,code=compute_50
-o simpleMPI.o -c simpleMPI.cu
scorep --cuda /usr/local/anaconda/bin/mpicxx -o simpleMPI
simpleMPI_mpi.o simpleMPI.o  -L"/usr/local/cuda-6.5"/lib64 -lcudart
```

Run Example

- Run

```
% mpiexec -np 4 ./simpleMPI

Running on 4 nodes
Average of square roots is: 0.667305
PASSED
```

- Find Tracefile appearing

```
% ls

Makefile                                simpleMPI      simpleMPI_mpi.o
NsightEclipse.xml                       simpleMPI.cpp  simpleMPI.o
readme.txt                              simpleMPI.cu
scorep-20150311_2045_907655747320       simpleMPI.h
```

Score-P performance monitor loaded on login

- Done via an environment module
- Also sets the following environment variables (it would be up to you)

```
% export SCOREP_ENABLE_TRACING=true
% export SCOREP_ENABLE_PROFILING=false
% export SCOREP_OPENCL_ENABLE=true
% export SCOREP_CUDA_ENABLE=driver,kernel,memcpy,flushatexit
% export SCOREP_OPENACC_ENABLE=true
```

What Happened Behind the Scenes? (2)

Makefile modified to instrument application

- Using **`scorep`** compiler wrapper
- Before:

```
NVCC := $(CUDA_PATH)/bin/nvcc -ccbin $(GCC)
MPICXX ?= $(shell which mpicxx 2>/dev/null)
```

- After:

```
NVCC := scorep --cuda $(CUDA_PATH)/bin/nvcc -ccbin $(GCC)
MPICXX ?= scorep --cuda $(shell which mpicxx 2>/dev/null)
```

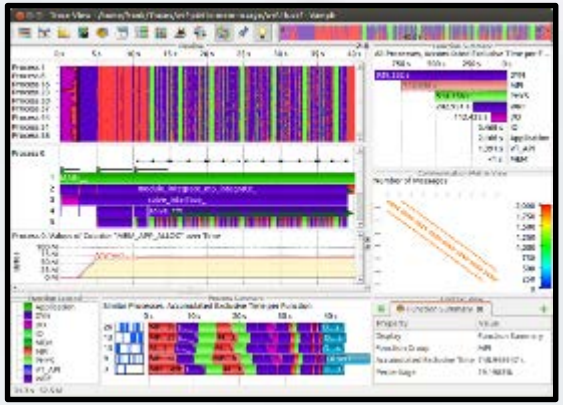
TECHNISCHE UNIVERSITÄT DRESDEN
Center for Information Services and High Performance Computing (ZIH)

Trace Visualization with Vampir

Guido Juckeland (guido.juckeland@tu-dresden.de)

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?

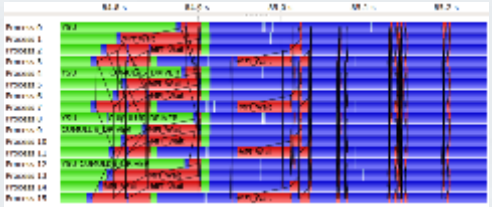


Event Trace Visualization with Vampir

- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior graphically at any level of detail
- Provide statistics and performance metrics

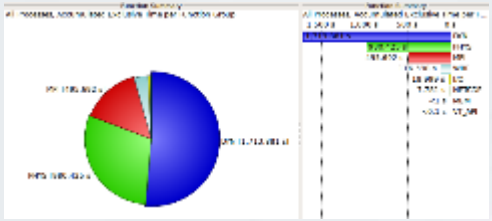
Timeline charts

– Show application activities and communication along a time axis



Summary charts

– Provide quantitative results for the currently selected time interval



The main displays of Vampir

Timeline Charts:

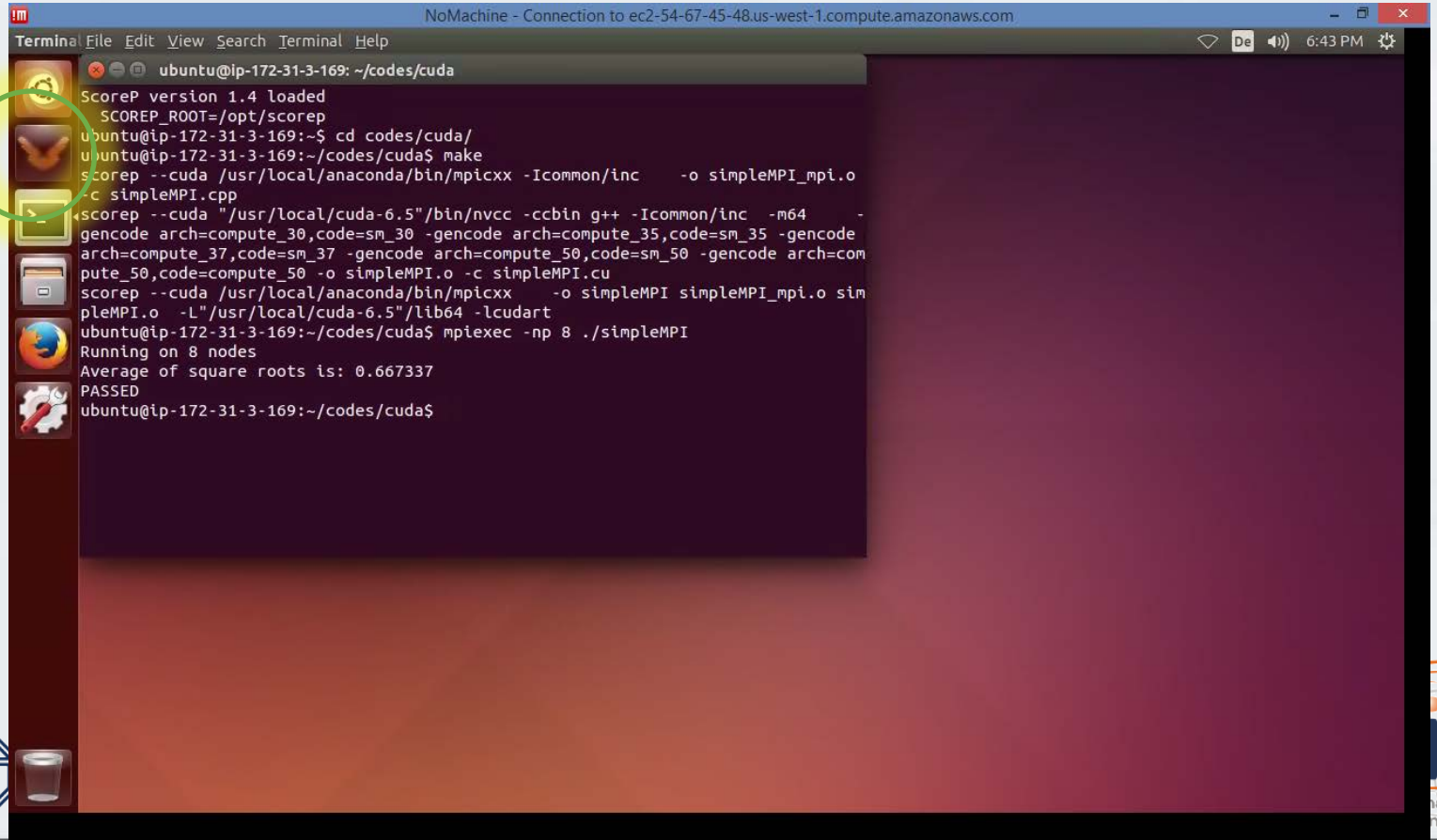
-  Master Timeline
-  Process Timeline
-  Counter Data Timeline
-  Performance Radar

Summary Charts:

-  Function Summary
-  Message Summary
-  Process Summary
-  Communication Matrix View

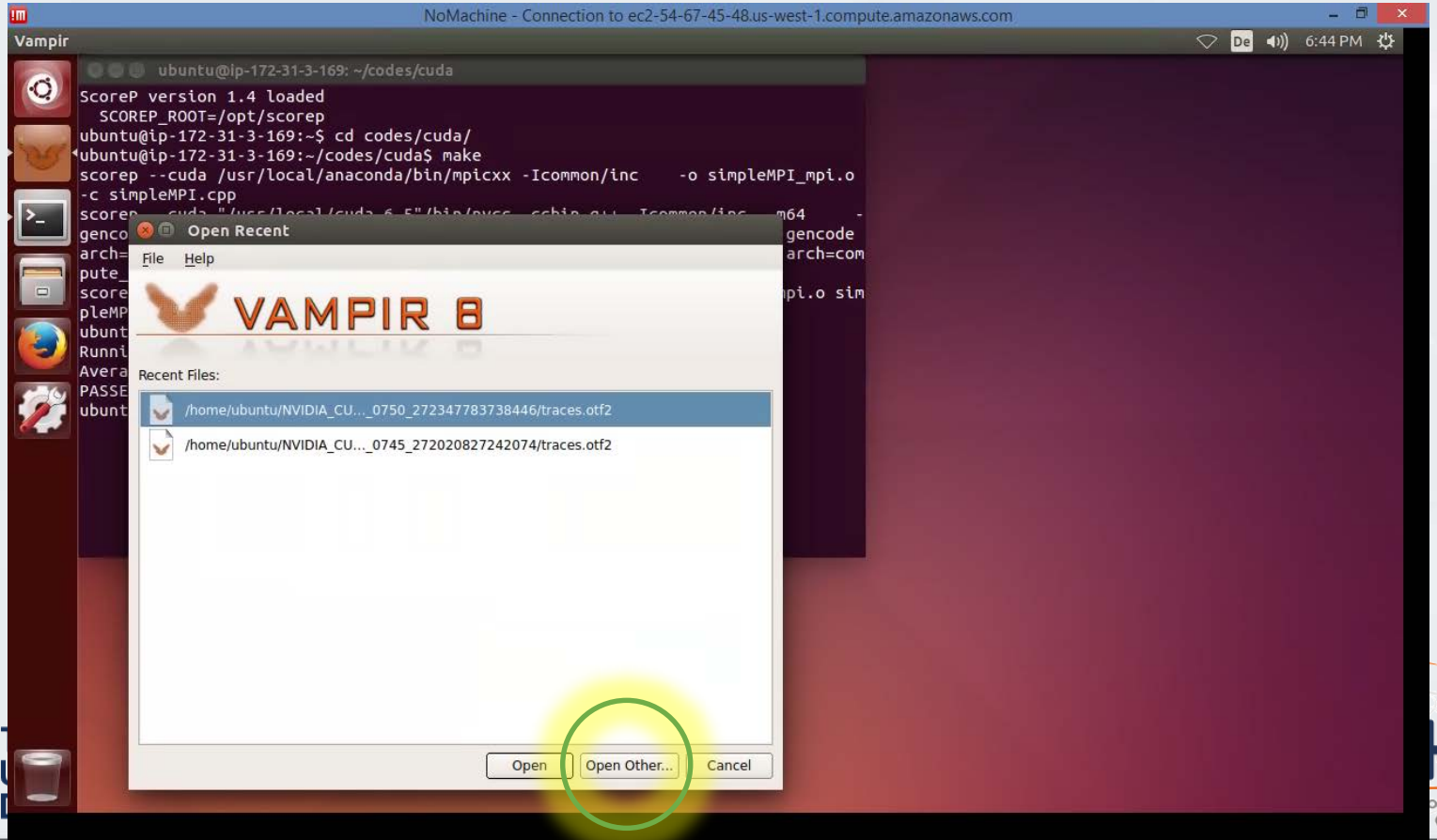
Let's Open Your Tracefile

Start Vampir



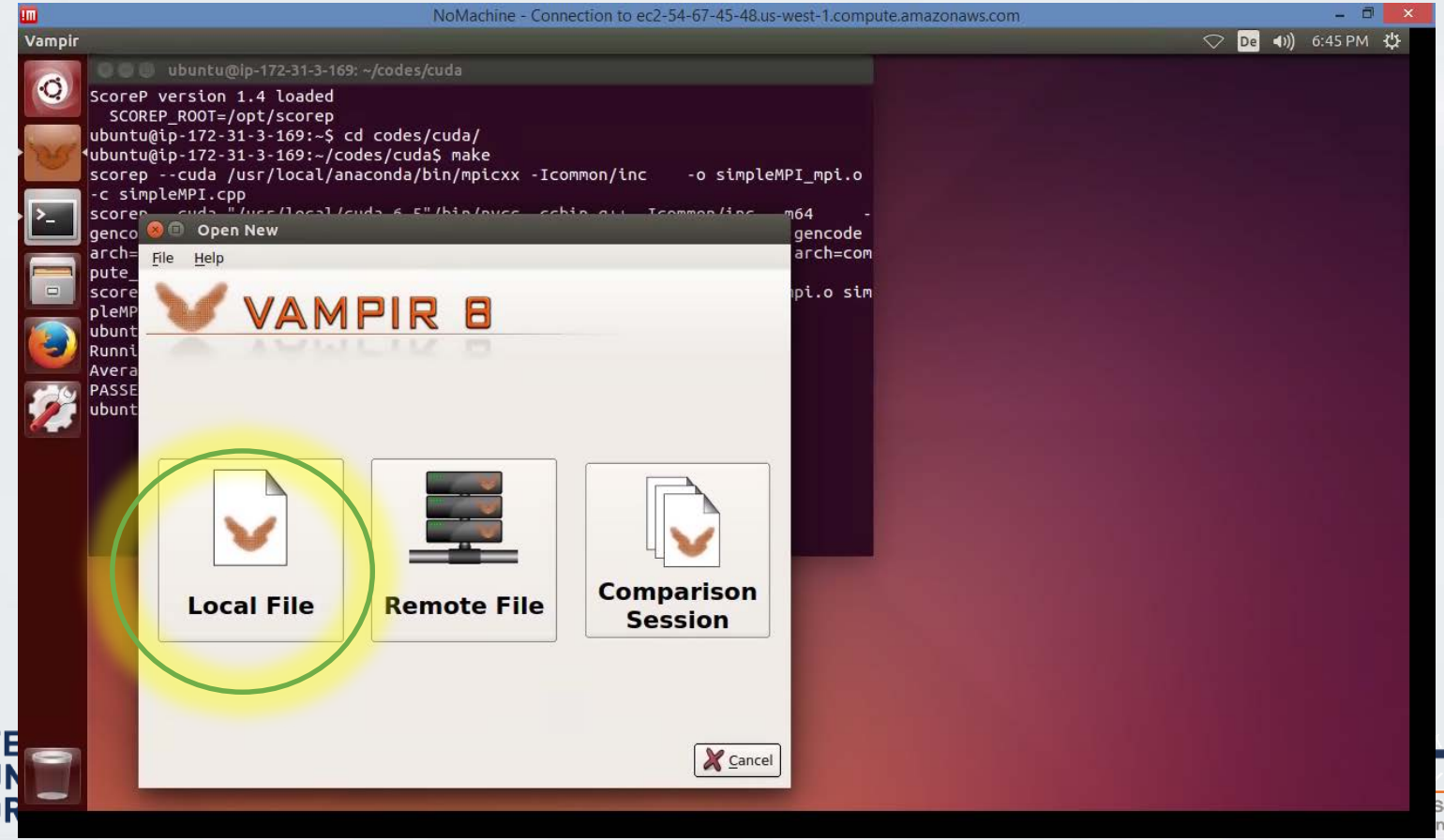
Let's Open Your Tracefile (2)

Click on "Open Other"



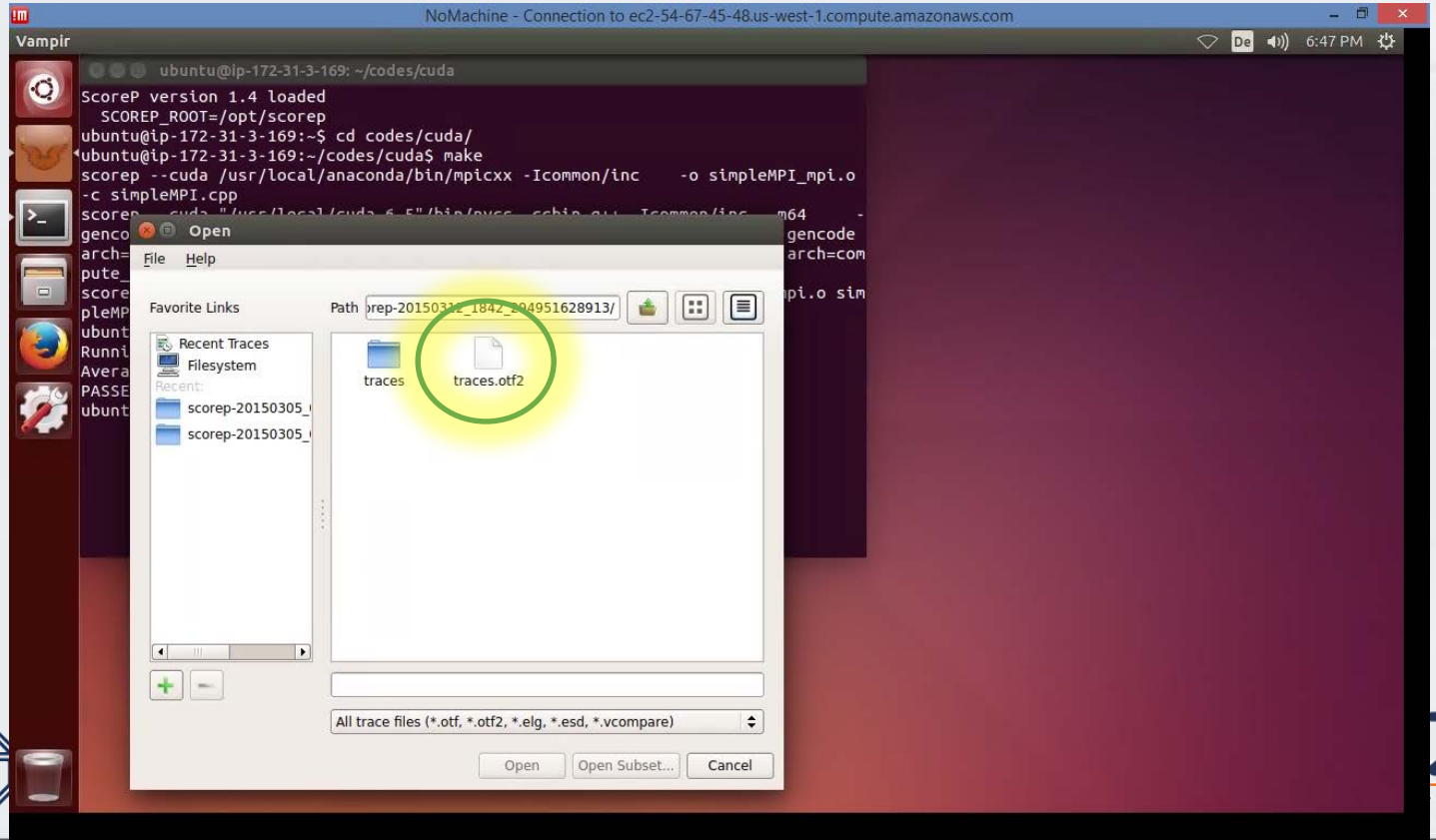
Let's Open Your Tracefile (3)

Select "Local File"



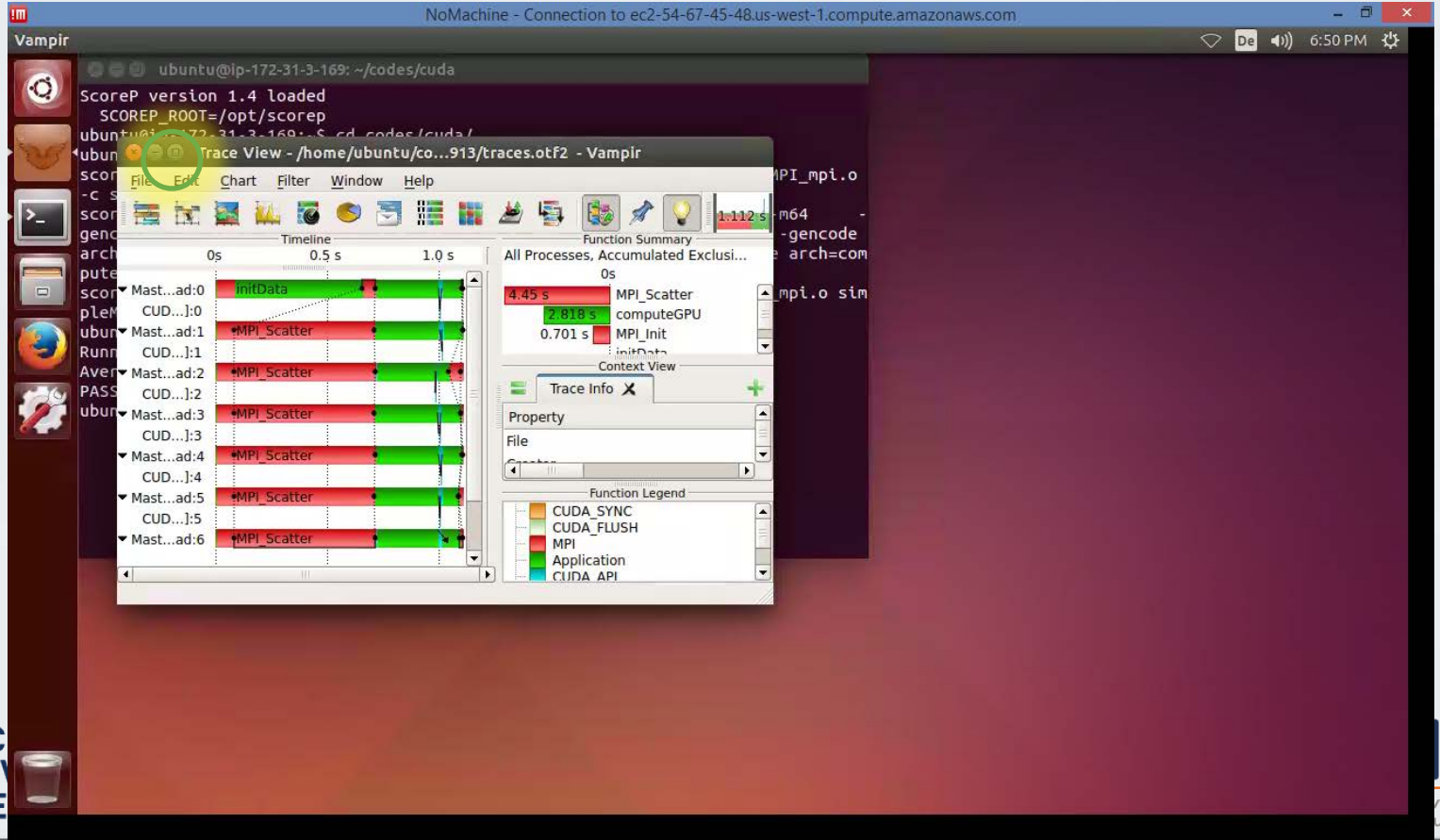
Let's Open Your Tracefile (4)

Navigate to "home", "ubuntu", "codes", "cuda", "scorep*", Open "traces.otf2"

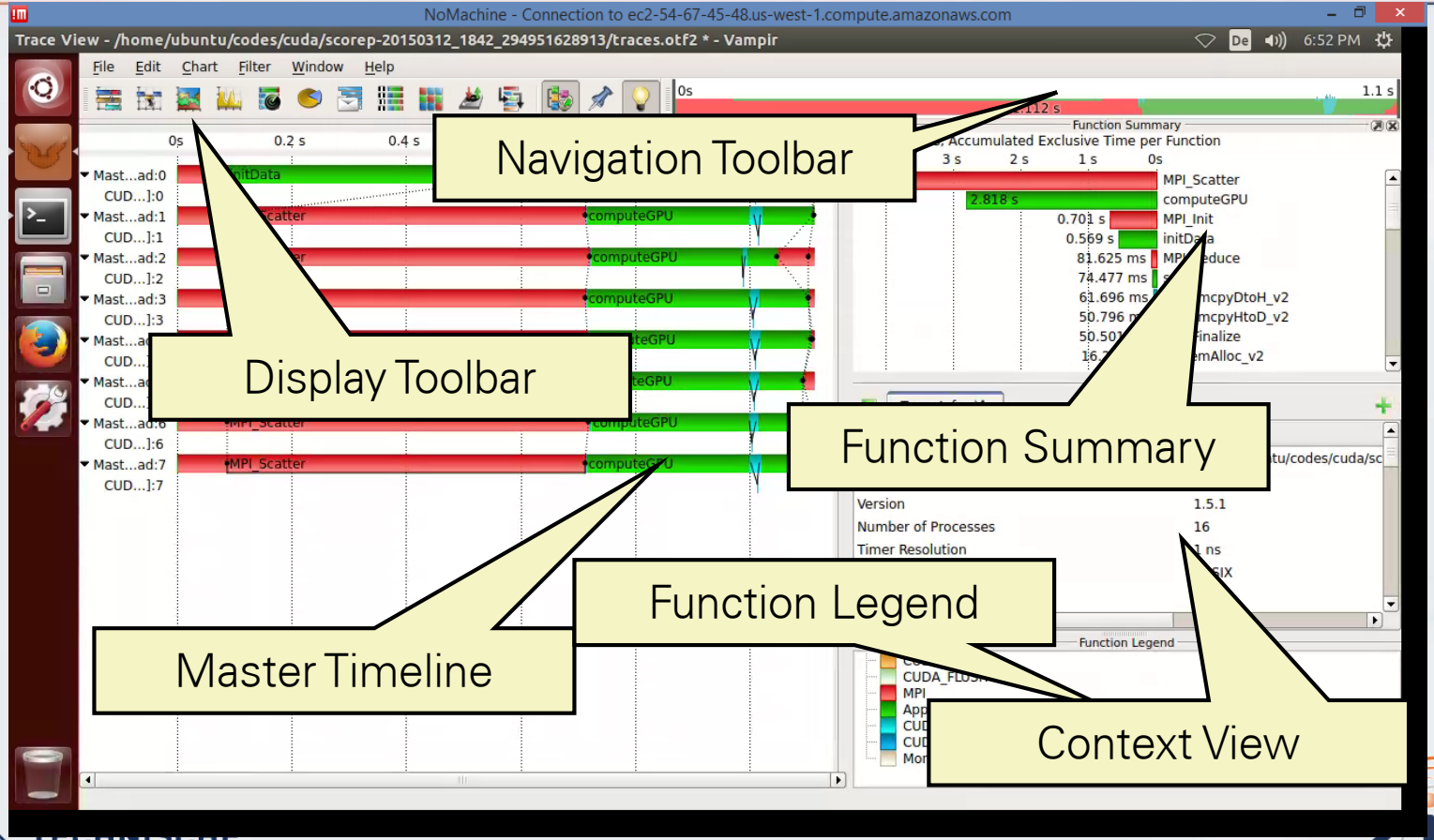


Let's Open Your Tracefile (5)

Maximize the Vampir window



What Do You See?



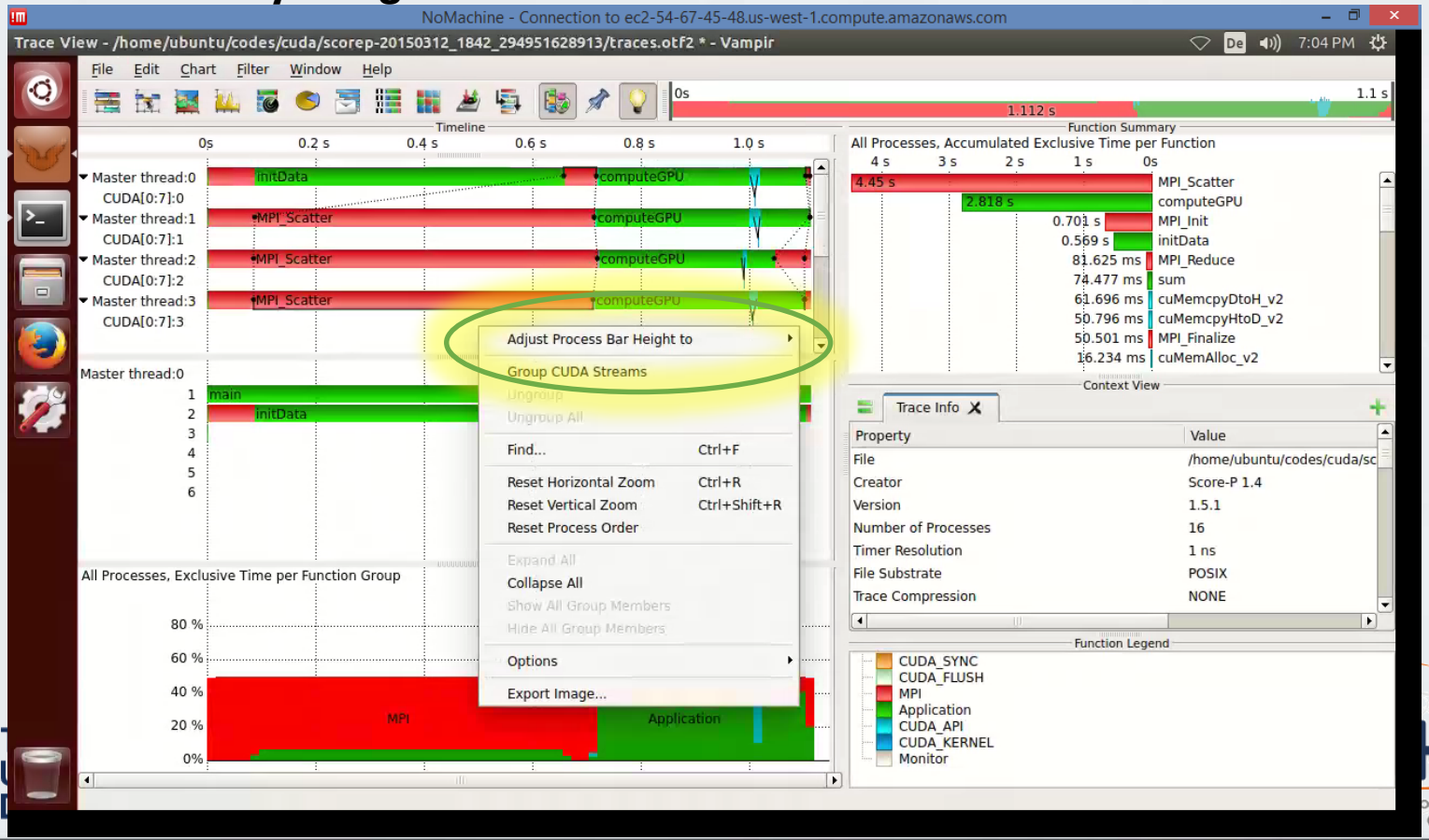
Demo

- Clicking on anything provides details in the context view
- Zooming is done by click, hold, release
  - Horizontal (Undo: Ctrl+Z, Reset: Ctrl+R)
  - Vertical (Undo: Ctrl+Z, Reset: Ctrl+Shift+R)
- Navigation Toolbar provides ways of sliding and zooming
- Adding more displays via display toolbar
- Moving displays around, dock to any border

Now you go ahead!

Changing displays

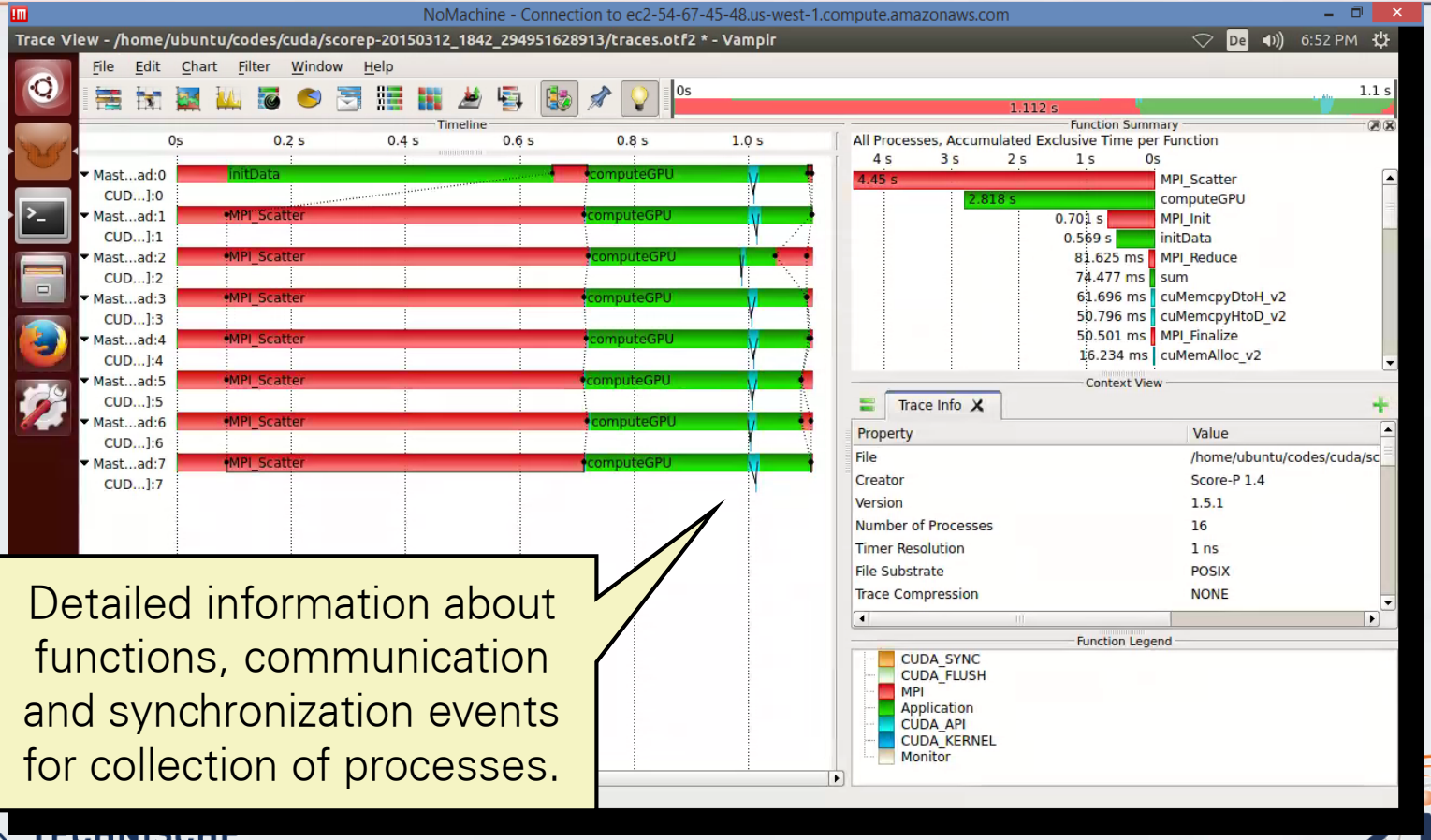
Right click on anything



Tasks

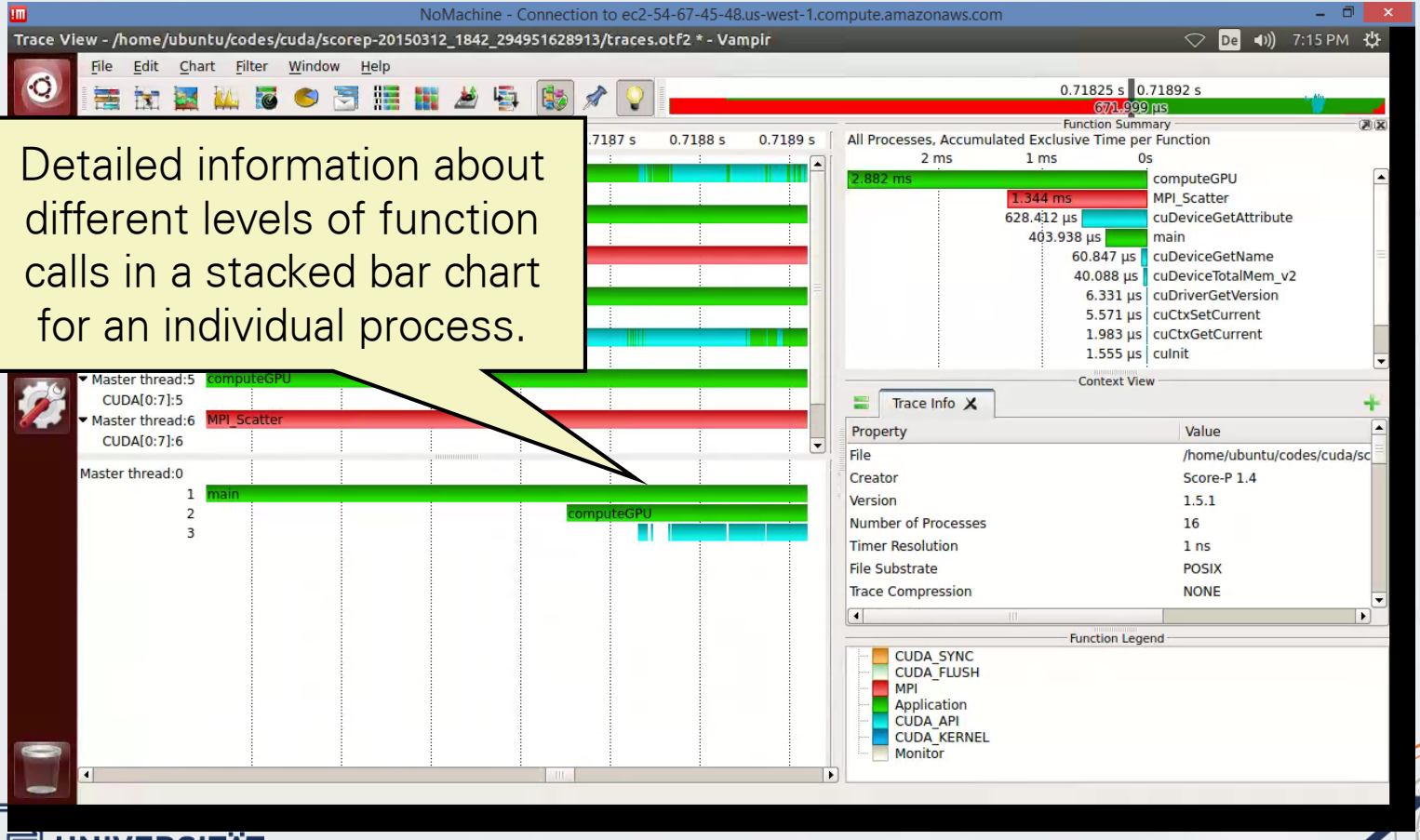
- Right click into Master Timline
- Adjust Process Bar Height to fit Chart Height
- Determine length of initialization phase
- Determine length of compute phase
- Determine kernel runtime
- Determine message sizes

Displays: Master Timeline



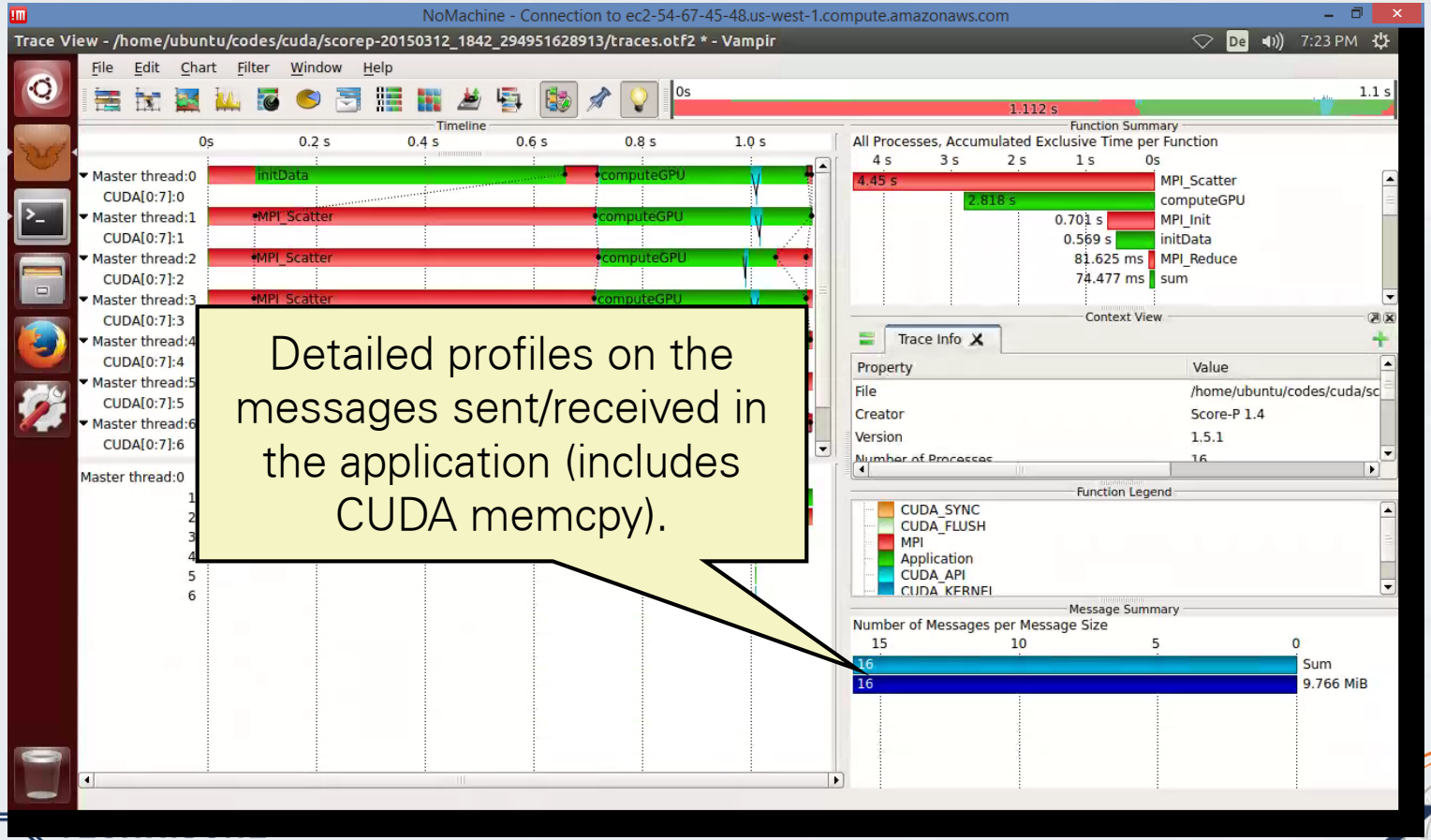
Detailed information about functions, communication and synchronization events for collection of processes.

Displays: Process Timeline



Detailed information about different levels of function calls in a stacked bar chart for an individual process.

Displays: Message Summary



Profiling At Its Best

- All displays are updated to the currently zoomed time interval
- Function Summary
  - Include/exclude functions
  - Change metric
  - Select processes used for profile
- Message Summary
  - Change metric
  - Select only specific senders/receivers

There Is an Example Trace to Play With

Go and look under /home/ubuntu/traces/cuda for more traces

Now go and play with your or my trace –
tell me how to improve the application

TECHNISCHE UNIVERSITÄT DRESDEN
Center for Information Services and High Performance Computing (ZIH)

A Look Ahead: OpenACC Tracing

Guido Juckeland (guido.juckeland@tu-dresden.de)

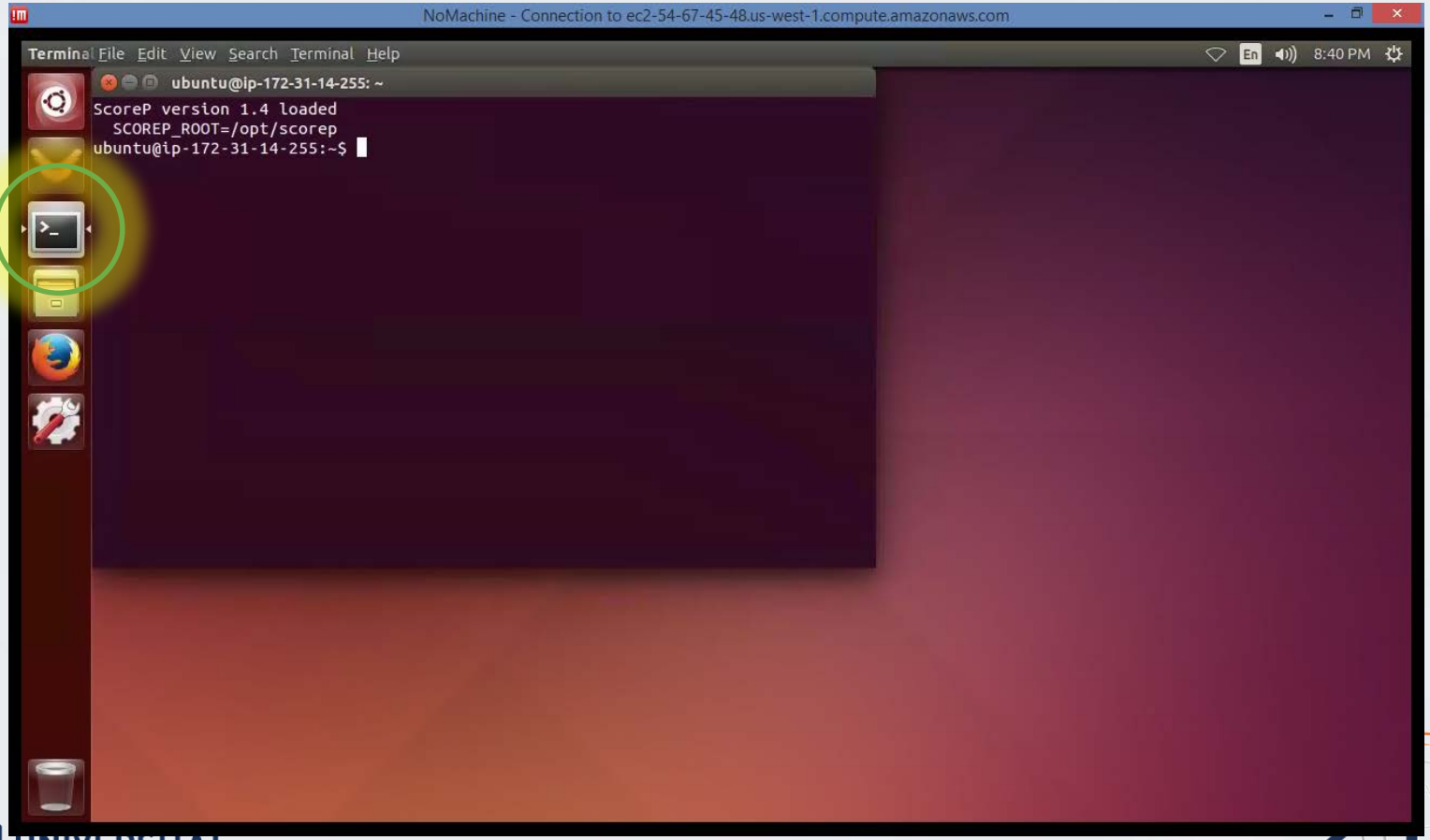
ZIH
Center for Information Services & High Performance Computing

Disclaimer

- Your are looking at a prototype
- Only works with PGI compilers and developer version of Score-P

- If you find it cool – talk to your OpenACC compiler vendor ☺

Start a Terminal



Switch to developer version of Score-P

```
% ubuntu@ip-172-31-3-169:~$ module purge
ScoreP version 1.4 unloaded

% module av
------------------------ /usr/share/modules/versions ------
3.2.10
------------------------ /usr/share/modules/modulefiles ----
dot                  modules              scorep/dev-openacc
module-git           null                 use.own
module-info          scorep/1.4(default)

% module load scorep/dev-openacc

ScoreP version openacc loaded
  SCOREP_ROOT=/opt/scorep-openacc
```

Go to OpenACC Example and Compile

- Go to OpenACC Example

```
% cd codes/openacc
```

- Compile

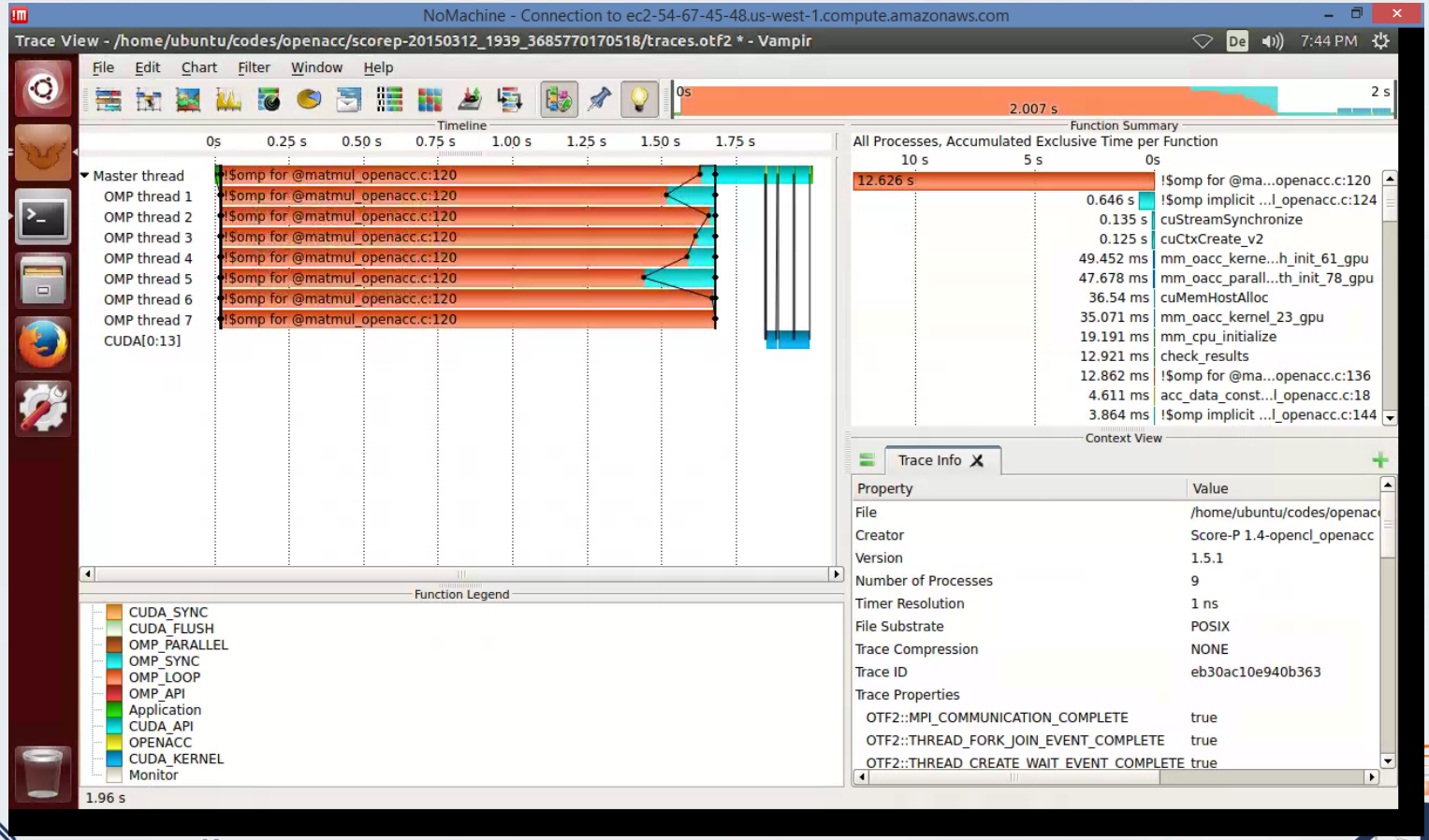
```
% make
scorep --cuda pgcc -mp -ta=nvidia matmul_openacc.c -o matmul_openacc
```

Run Example

- Run

```
% export OMP_NUM_THREADS=8
% ./matmul_openacc
CPU MM with 8 threads
MM on CPU: 1.658984 sec
mm_oacc_kernel(): 0.207447 sec
OpenACC matrix multiplication test was successful!
mm_oacc_kernel_with_init(): 0.052948 sec
OpenACC matrix multiplication test was successful!
mm_oacc_parallel_with_init(): 0.051797 sec
OpenACC matrix multiplication test was successful!
Total runtime: 0.325640 sec
```

Open the New Tracefile



There Is an Example Trace to Play With

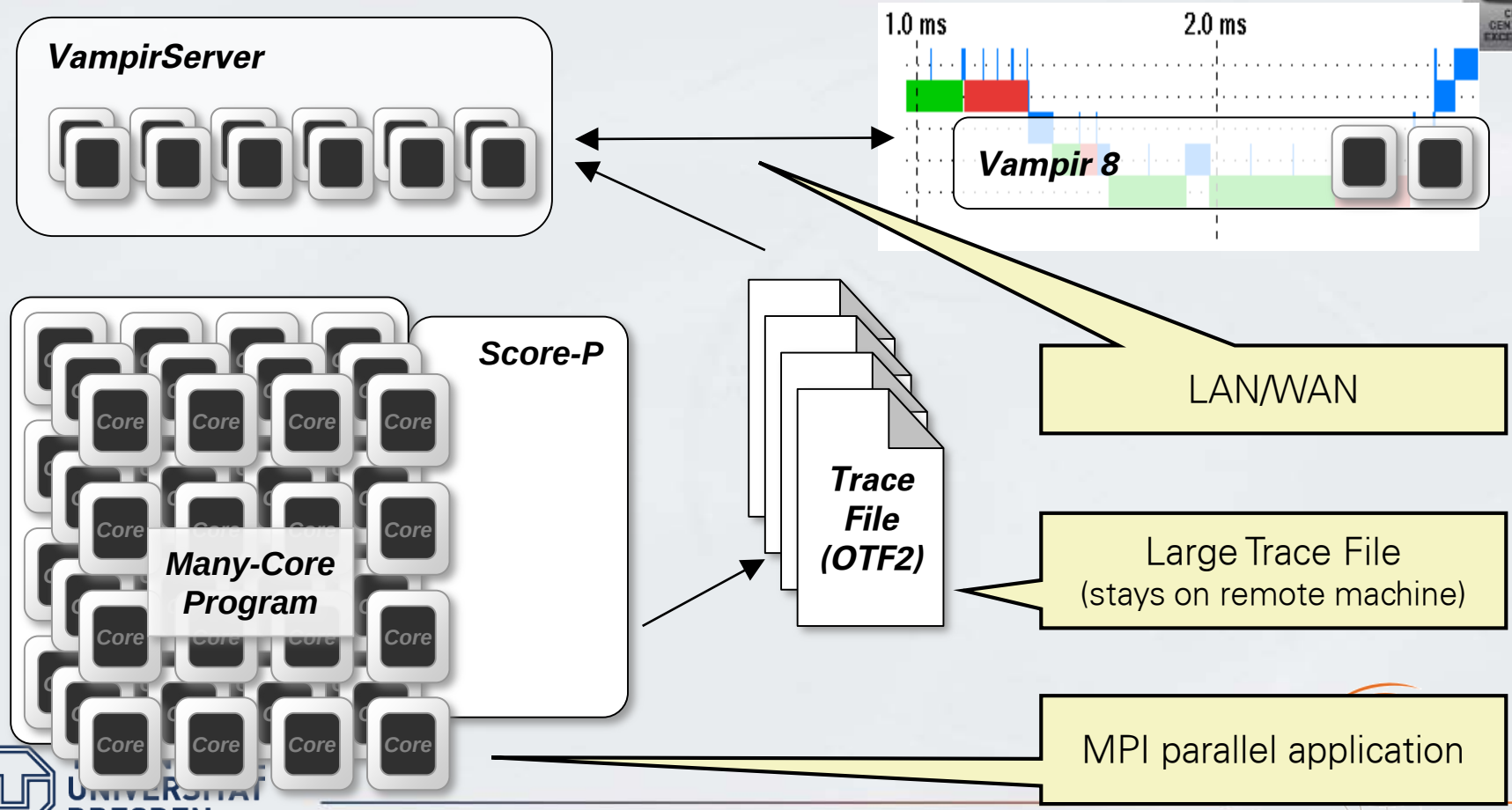
Go and look under /home/ubuntu/traces/openacc for more traces

Now go and play with your or my trace –
tell me how to improve the application

Looking a Little Further

Guido Juckeland (guido.juckeland@tu-dresden.de)

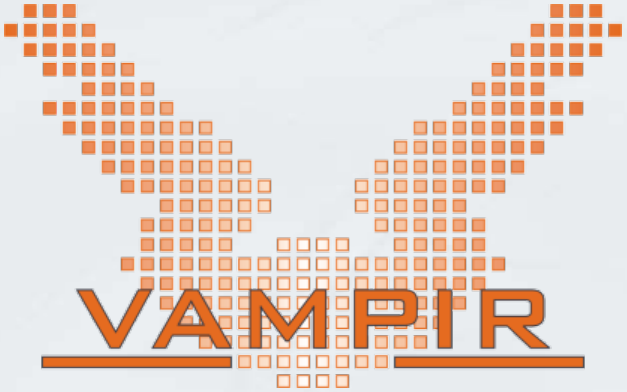
How About Very Large Tracefiles?



Wrap Up

- Performance Analysis is valuable
- Use “easy” tools first
- Score-P can record any concurrent activity
- Vampir can visualize all that activity

- The rest is experience and up to you ☺



Vampir is available at http://www.vampir.eu,

get support via vampirsupport@zih.tu-dresden.de