Peter Willendrup and Erik Knudsen DTU Physics

McStas-3.0 GPU port

Agenda

- McStas on GPU via OpenACC

  (a “high-level” #pragma driven access to CUDA see https://www.openacc.org and https://developer.nvidia.com/hpc-sdk)

- How well (fast) does it work?

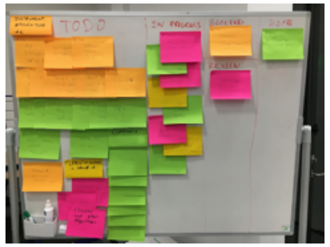
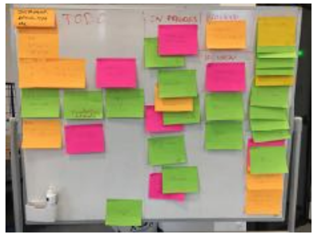
- Simulation flow

- What did we change?

- What does not work

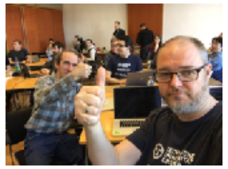
Warning:
1. Assumes previous experience with McStas
2. Does not introduce OpenACC

Main events on timeline of road toward GPU

2017: E. Farhi initial cogen modernisation

March 2018: Participation at **Dresden** Hackathon. 1st "null" instrument prototype runs.



Fall 2018 onwards: J. Garde further cogen modernisation and restructuring

October 2019: Participation at **Espoo** Hackathon. First meaningful data extracted. Work on cogen and realising we need another RNG.

October 2019 onwards: J. Garde & P. Willendrup: New RNG, test system, multiple functional instruments.

November-December 2019: First good look at benchmarks and overview of what needs doing for first release with limited GPU support.

January 2020: One-week local hackathon **@ DTU**

with McCode & RAMP teams

February 2020: **First release** McStas **3.0beta** with GPU support was **released** to the public

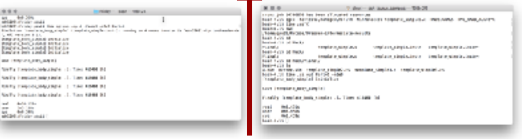
2020 1st **Corona** lockdown P. Willendrup & E. Knudsen continue work on comp and cogen

November 2020 **Virtual** Hackathon, setting release scope

December 15th 2020 McStas 3.0 release!

McStas / McXtrace instrument simulation

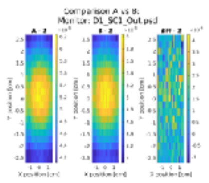
CPU MPI 4 cores (95 % usage) | GPU 5% usage



16.12 s (Single core 56.0 s) | 5.43 s

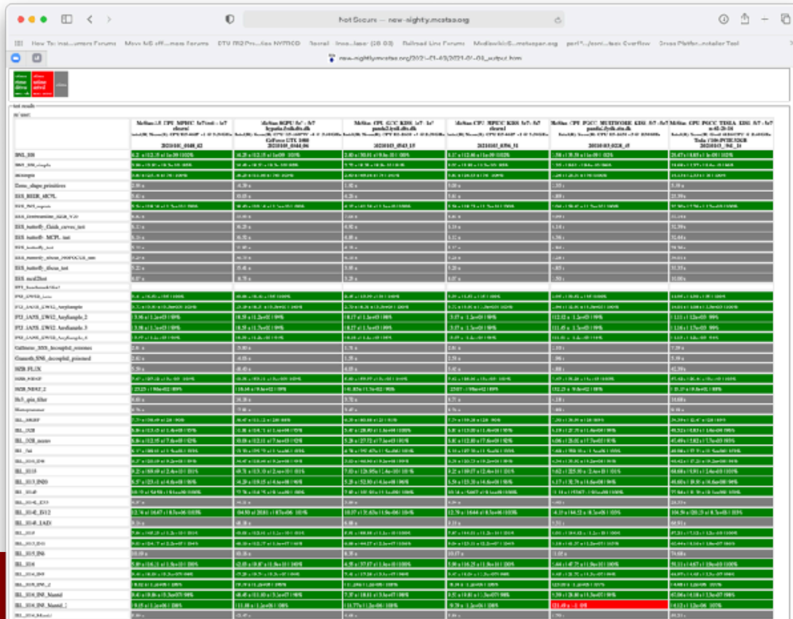
mentor: Vishal Metha — NVIDIA

hackathon org.: Guido Juckeland — HZDR, Helmholtz Zentrum Dresden Rossendorf



mentor: Christian Hundt — NVIDIA

hackathon org.: Sebastian Von Alfthan — CSC

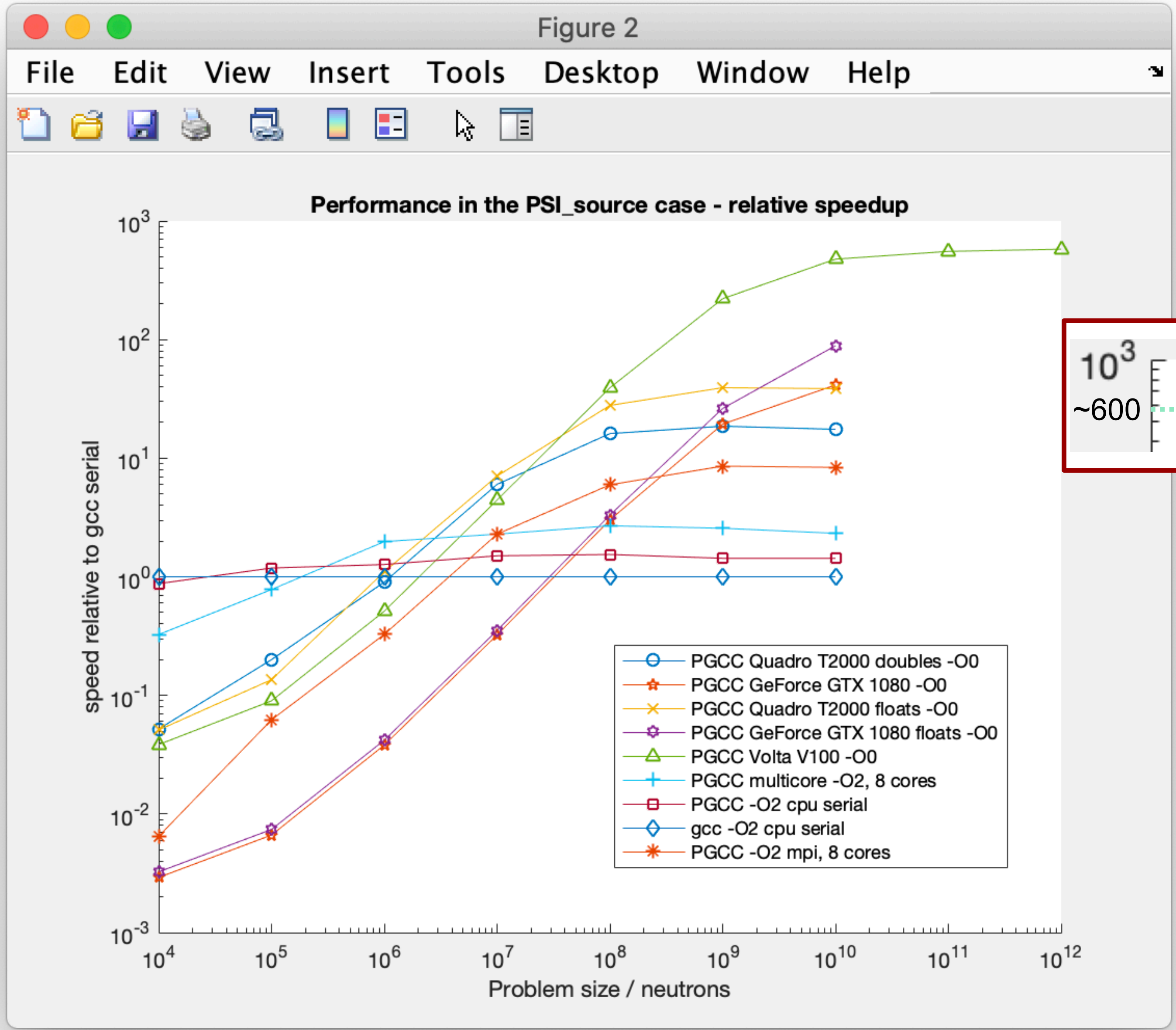


OpenACC NVIDIA

McStas heading for the GPU… November 2019 - first good look at performance.

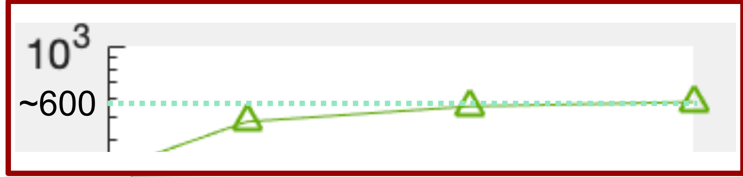
Idealised instrument with source and monitor only - i.e. without any use of the ABSORB macro.

(Likely a good indication of maximal speedup achievable.)



Speedup

Looks like a factor of ~600



V100 execution speedups renormalised to wall-clock of single-core gcc standard simulation,

V100 run is 600 times faster than a single-core CPU run

McStas heading for the GPU… first benchmarking numbers from <u>November</u> <u>2019</u>

9 instruments fully ported, also realistic ones like PSI_DMC

(Aug 2020: 99 instrs)

10-core MPI run, **1e9** in 200 secs

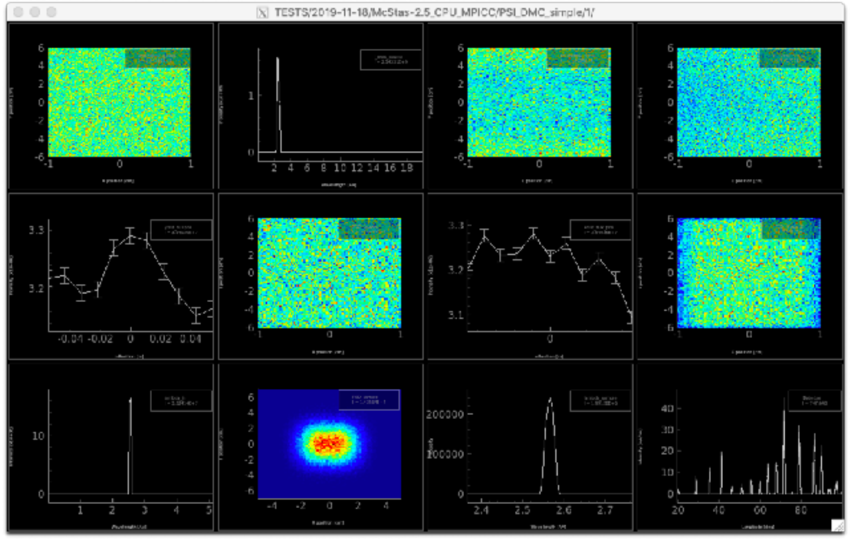
(1-core run, **1e9** would be 2000 secs)

VS.

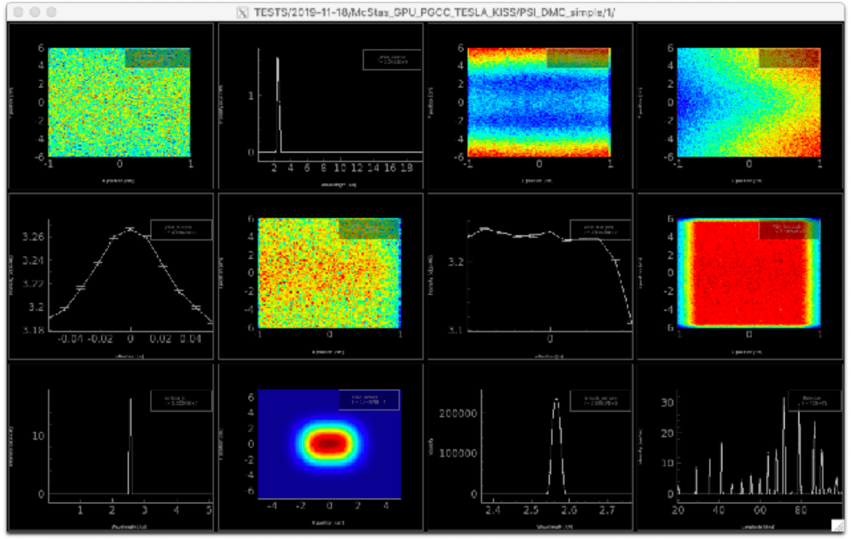
Tesla V100 run, **1e9** in 22 secs

~ i.e. 2 orders of magnitude wrt. a single, modern CPU core

- If problem has the right size / complexity, GPU via OpenACC is great!



TESTS/2019-11-18/McStas-2.5_CPU_MPICC/PSI_DMC_simple/1/



TESTS/2019-11-18/McStas_GPU_PGCC_TESLA_KISS/PSI_DMC_simple/1/

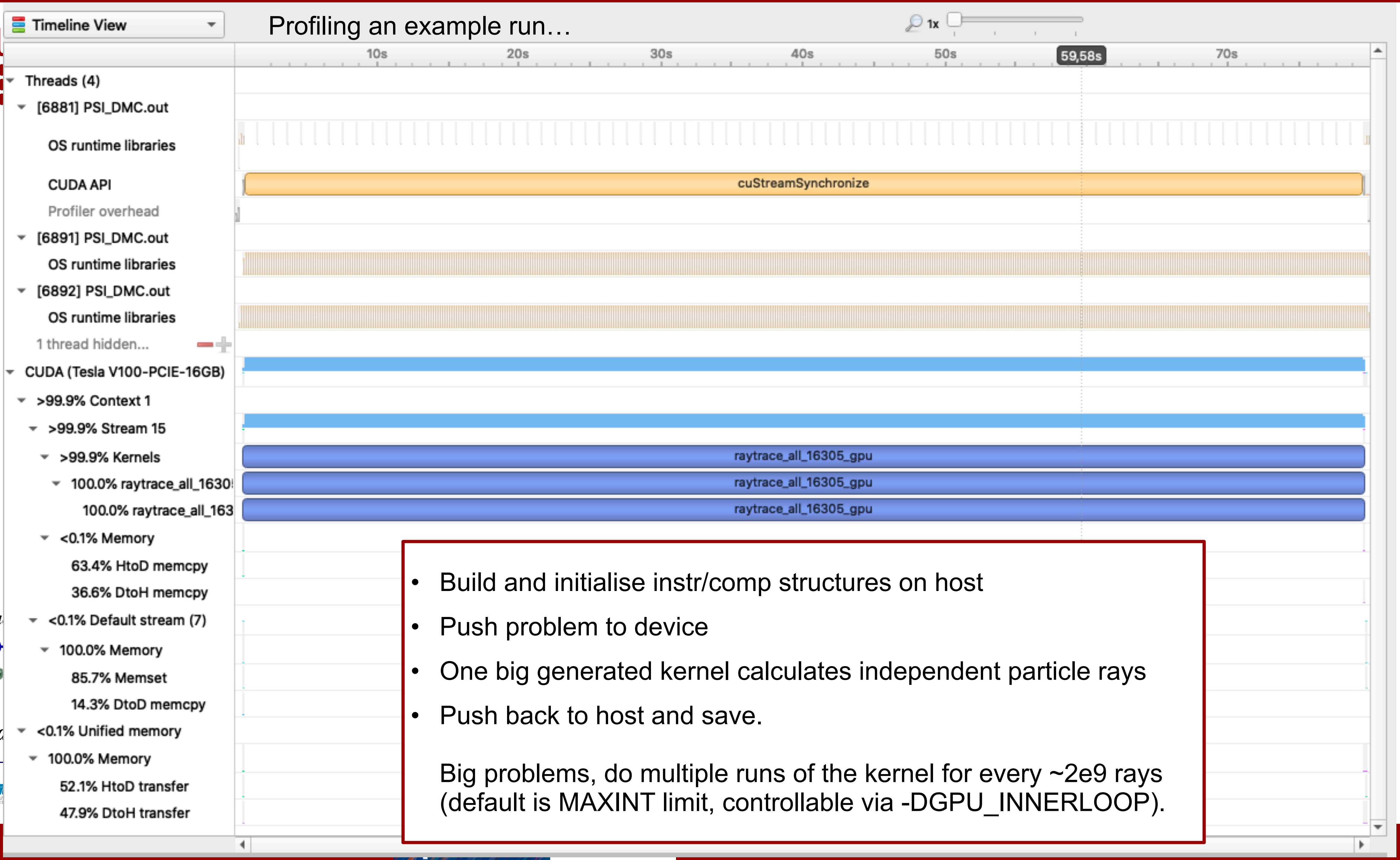
Anatomy of a McStas GPU run (*)

- Init, geometry, files etc. read on CPU
  - MPI if needed
- Memory-structures
  - Built on CPU
  - Marked for transfer to GPU (#pragma acc declare create etc.)
  - Initialised and synced across
  - Trace-loop is a #pragma acc parallel loop
    - Calculation performed entirely on GPU
  - Component structs (incl. e.g. monitor-arrays) synced across
- Finally and Save runs on CPU
  - MPI merge if needed

No printfs etc. available on GPU, automatically suppressed by #defines

(* Alternative layout via FUNNEL mode, see next 2 slides)

Profiling an example run…



- Build and initialise instr/comp structures on host
- Push problem to device
- One big generated kernel calculates independent particle rays
- Push back to host and save.

  Big problems, do multiple runs of the kernel for every ~2e9 rays (default is MAXINT limit, controllable via -DGPU_INNERLOOP).

FUNNEL mode

- Activated **explicitly** using -DFUNNEL or **implicitly** using CPUCOMPONENT in instrument or NOACC in comp header

- Has N kernels / calculation zones instead of one

    1. Separation at SPLIT

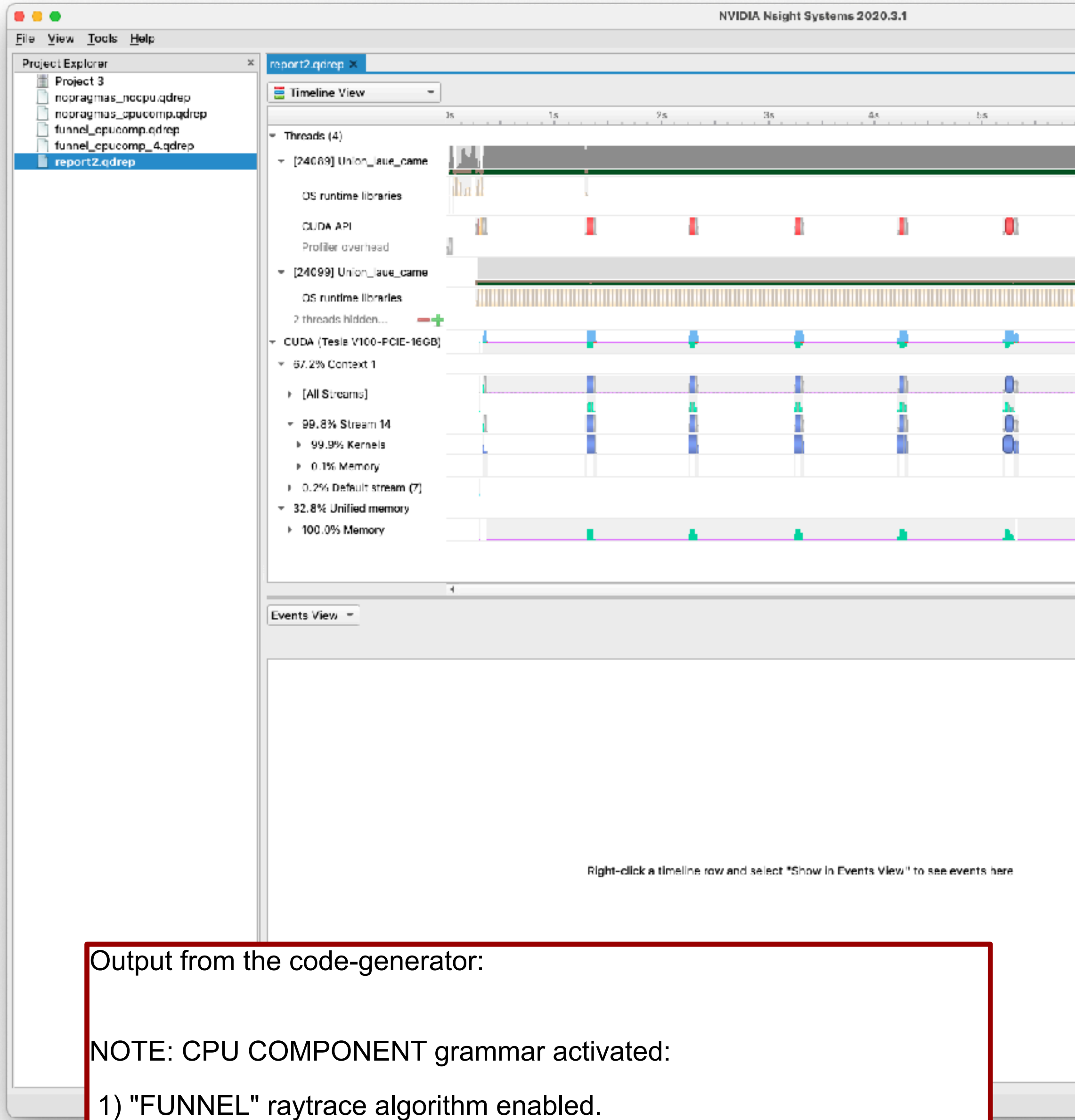
    2. Separation if CPUCOMPONENT in instrument file
       (CPUCOMPONENT A=Comp(vars=pars…))

    3. Separation if a component has NOACC in the header
       (See e.g. Multilayer_sample, Union_master)

- Each of these “calculation zones” is finalised before the next one initiated.

- Example:
  Union: Instrument up to Union_master can be GPU, then CPU, then GPU again

  - Can be as slow as single cpu…
  - Copying back and forth to/from GPU is costly…



Output from the code-generator:

NOTE: CPU COMPONENT grammar activated:

1) "FUNNEL" raytrace algorithm enabled.

2) Any SPLIT's are dynamically allocated based on available buffer size.

(McStas 3.0 mcrun is preconfigured on Linux - excluding -Minfo=accel, simply use mcrun --openacc when compiling, can also combine with e.g. --mpi=N)

Compiler settings used for GPU:

```
nvc -ta=tesla,managed -Minfo=accel -DOPENACC
```

Main “enable GPU”/OpenACC switch

Give accel debug information

Use CUDA shared memory for host-device-host allocation. Needed for our 2D-arrays at present, may include penalty, we could get rid.

Generate Tesla code. “compute capability” e.g. tesla:cc70 may be specified to indicate specific card.

What doesn’t work

- **Function pointers are not available on GPU**
  - Solutions:
    - Code around if possible (integration routine pr. specific function to be integrated…)
    - Mark the component NOACC

- **Variadic functions are not available on GPU**

- **Anonymous structs as comp pars are not available on GPU**
  - Unfold into comp struct

- **User-defined fieldfunctions for polarisation had to be abandoned**
  - No solution yet, may become handled via grammar

- **External libs generally can not be used in TRACE** (“#pragma….” hard to add on 3rd party codes)
  - Handle in INIT / FINALLY (MCPL)
  - NOACC (GSL etc.)

- **Union master is for now NOACC**, will eventually become supported on GPU

- (Looks like we may have implemented a BUG in the NeXus/Mantid stuff…)

Conclusions

- **It really does work nicely!**
- **Code changes** much **less invasive** than envisioned!
- It often gives a speedup of **1-2 orders** of magnitude over 1 cpu
- **Most things work**
  (we have workarounds or solutions in the pipe for the rest)
- McStas 3.0 is as of yet “ported” to GPU but **not fully “optimised” performance-wise**, we will try to go to another Hackathon
- **Union** needs a dedicated **Hackathon**

The team, Nvidia mentors and Hackathon hosts :-)

Vishal Metha

Christian Hundt

Alexey Romanenko

Guido Juckeland

Sebastian von Alfthan