

Peter Willendrup and Erik Knudsen DTU Physics

mcstas-2.x vs. mcstas-3.0, status and elements of the 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

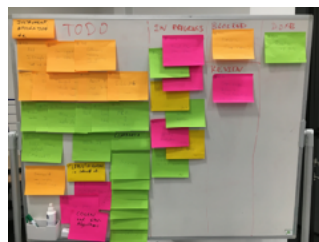
McXtrace



McStas



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



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

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

January 2020:
One-week local
hackathon @ DTU
with McCode & RAMP teams

November 2020
Virtual Hackathon,
setting release scope

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



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

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

December 15th 2020
McStas 3.0 release!

[illegible][illegible]

hackathon org.:
Guido Juckeland

 **HZDR**
 **HELMHOLTZ**
ZENTRUM DRESDEN
ROSSENDORF

hackathon org.:
Sebastian Von Alfthan

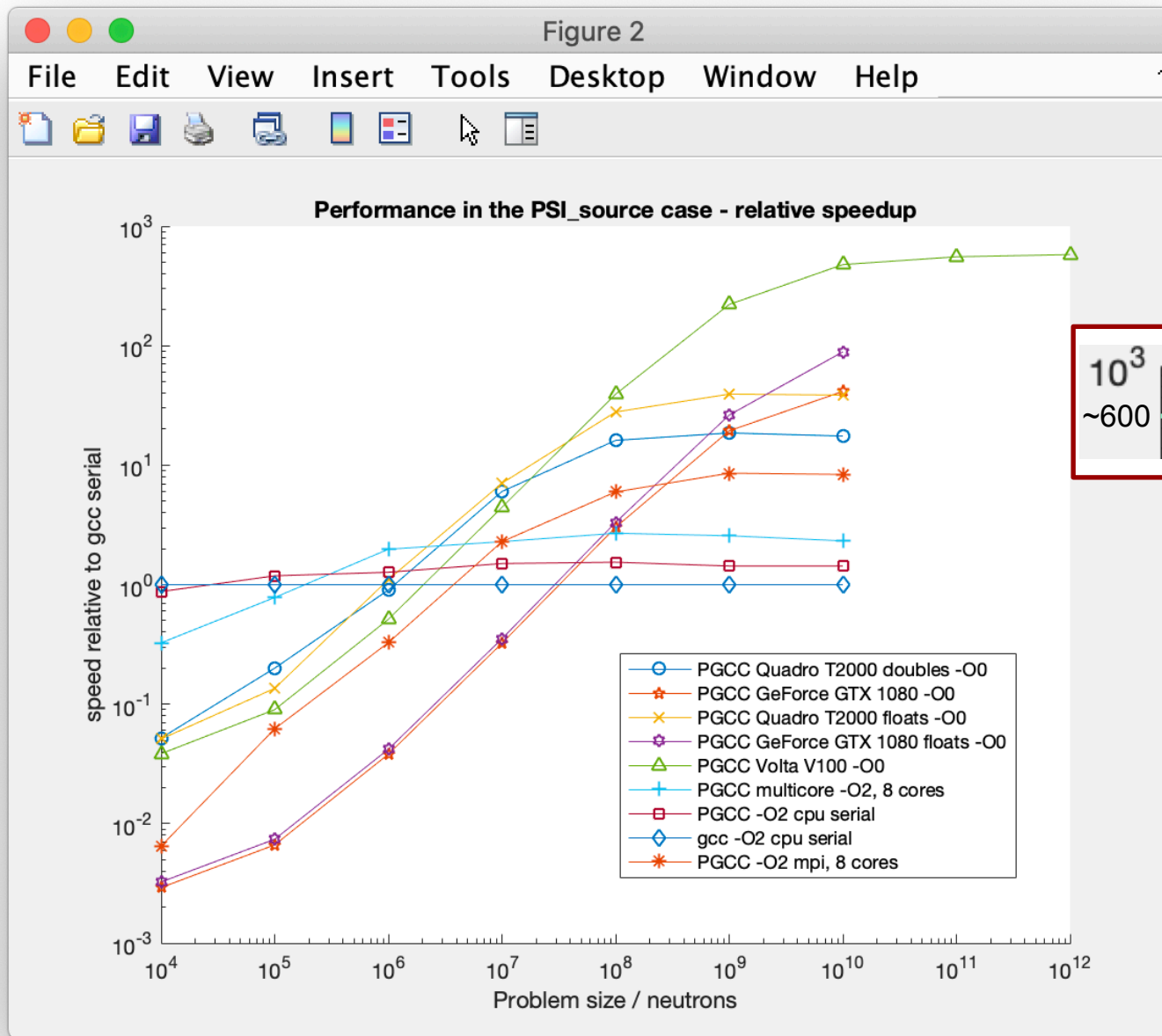
February 2020:
First release
McStas **3.0beta**
with GPU
support was
released
to the public

[illegible]

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**

McXtrace



McStas



McStas heading for the GPU... first benchmarking numbers from November 2019

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)

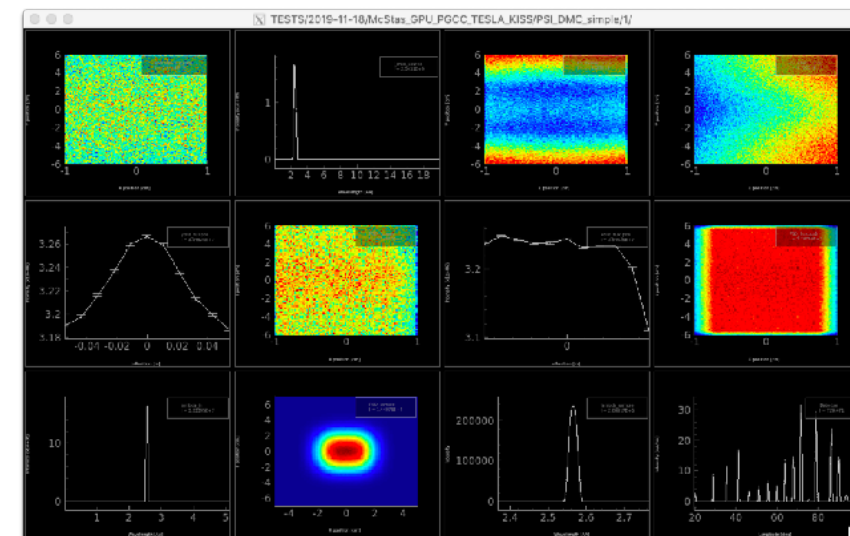
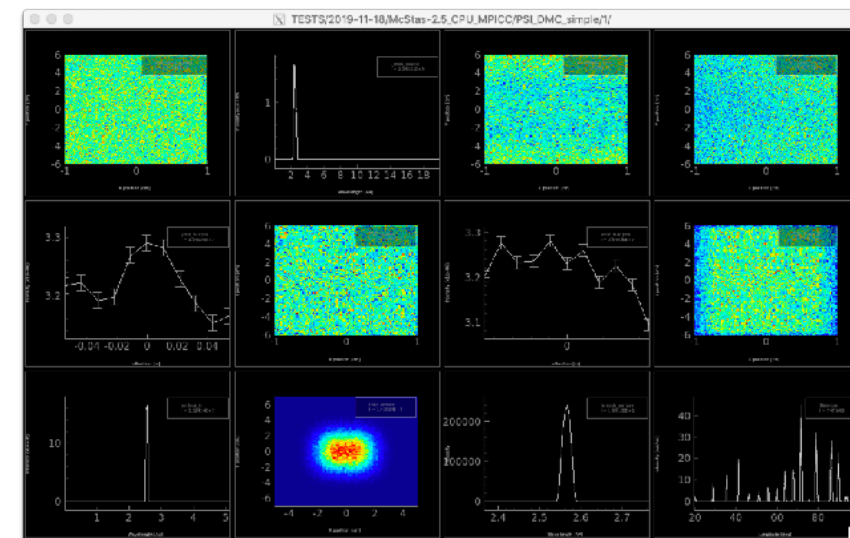
VS.

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!



McStas 2.x -> McStas 3.0 main differences

- **Rewritten** / streamlined simplified **code-generator** with
 - Much **less generated code**
 - **improved compile time and compiler optimizations**, esp. for large instrs
 - **Much less invasive use of #define**
 - **Component sections -> functions** rather than #define / #undef
 - Much **less global variables**, instrument, component and neutron reworked to be **structures**

Advantage
of 3.0 also on
CPU



- Use of **#pragma acc ...** in lots of places (**put in place by cogen** where possible)

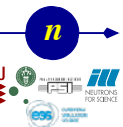
OpenACC

- **New random number generator** implemented
 - We couldn't easily port our legacy Mersenne Twister
 - Experimenting with curand showed huge overhead for our relative small number of random numbers
(we have hundreds or thousands of random numbers, not billions)
- Complete change to **dynamic** monitor-arrays

McXtrace



McStas



Anatomy of a McStas GPU run (*)

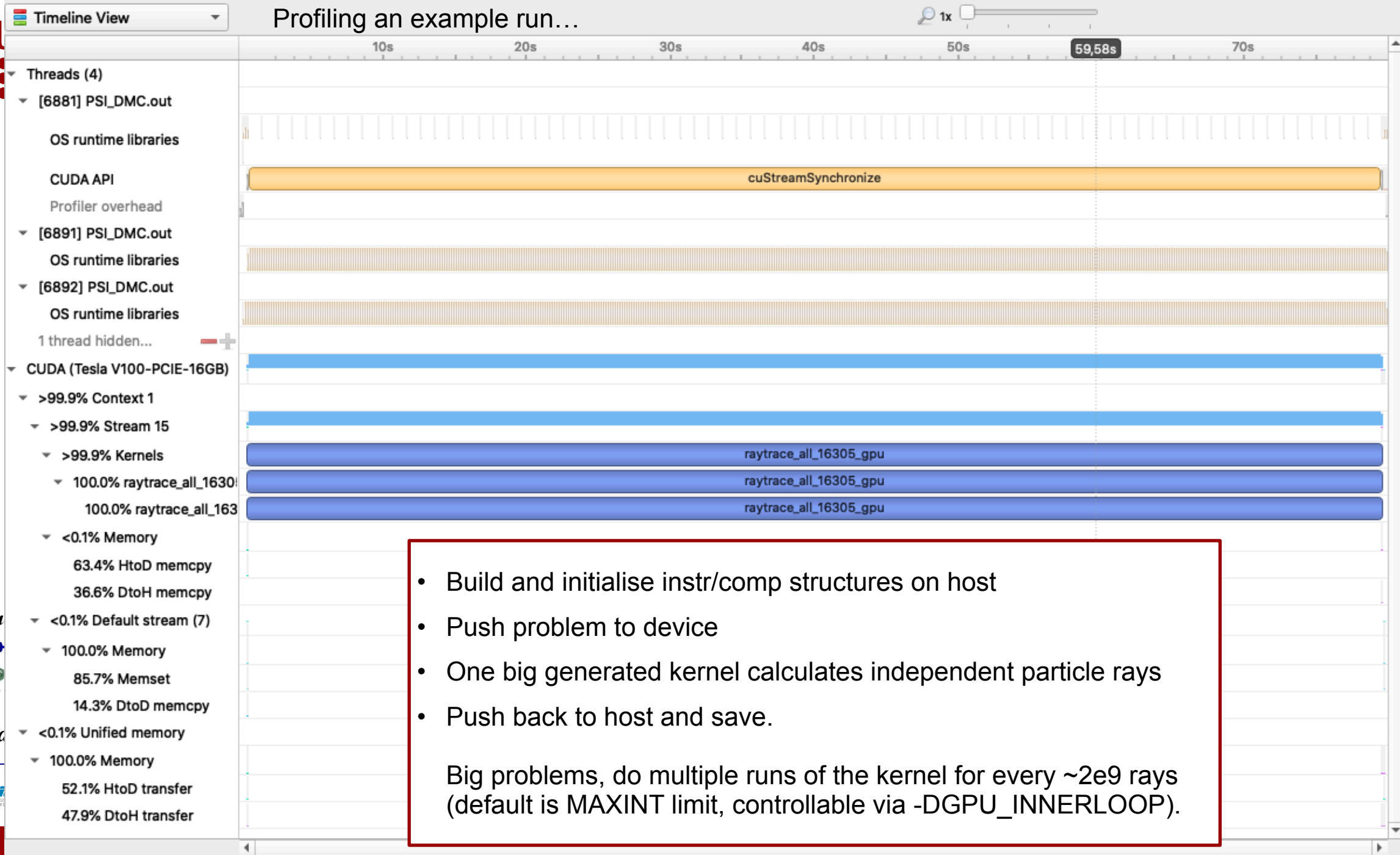
- Init, geometry, files etc. read on CPU
 - MPI if needed
- Memory-structures
 - Built on CPU
 - Marked for transfer to GPU (`#pragma acc declare create etc.`)
 - Initialised and synced across
 - Trace-loop is a `#pragma acc parallel loop`
 - Calculation performed entirely on GPU
 - Component structs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



OpenACC

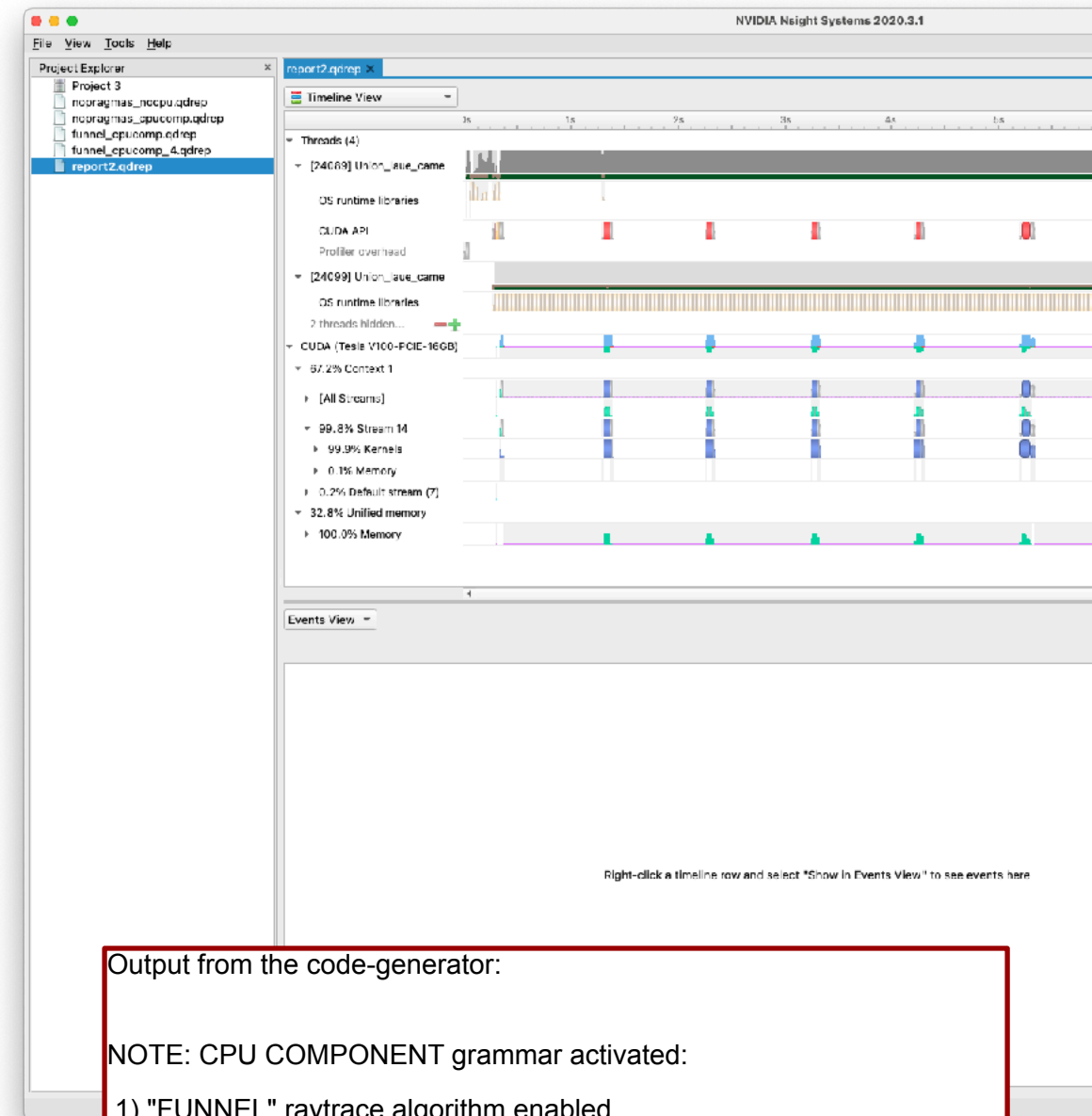
No printf's etc. available
on GPU, automatically
suppressed by `#defines`

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



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
 - Separation at SPLIT
 - Separation if CPUCOMPONENT in instrument file
(CPUCOMPONENT A=Comp(vars=pars...))
 - 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...



- Illustration, simple instr with
- Instr vars and “flag”
- Arm
- Source
- Slit
- PSD

```

example_v25.instr
/*
 * %Example: example.instr dummy=0 Detector: detector_I=345.995
 */
DEFINE INSTRUMENT Minimal(dummy=0)

DECLARE %{
double constant=2;
double two_x_dummy;
double flag;
%}

INITIALIZE %{
two_x_dummy=2*dummy;
%}

TRACE

COMPONENT arm = Arm()
AT (0, 0, 0) ABSOLUTE
EXTEND %{
flag=0;
%}

COMPONENT source = Source_simple(
radius = 0.02,
dist = 3,
focus_xw = 0.01,
focus_yh = 0.01,
lambda0 = 6.0,
dlambda = 0.05,
flux = 1e8)
AT (0, 0, 0) RELATIVE arm

COMPONENT coll2 = Slit(
radius = 0.01)
AT (0, 0, 6) RELATIVE arm
EXTEND %{
flag=SCATTERED;
%}

COMPONENT detector = PSD_monitor(
nx = 128,
ny = 128,
filename = "PSD.dat",
xmin = -0.1,
xmax = 0.1,
ymin = -0.1,
ymax = 0.1)
AT (0, 0, 9.01) RELATIVE arm

END

```

```

example_v30.instr
/*
 * %Example: example.instr dummy=0 Detector: detector_I=345.995
 */
DEFINE INSTRUMENT Minimal(dummy=0)

DECLARE %{
double constant;
double two_x_dummy;
%}

USERVARS %{
double flag;
%}

INITIALIZE %{
constant=2;
two_x_dummy=2*dummy;
%}

TRACE

COMPONENT arm = Arm()
AT (0, 0, 0) ABSOLUTE
EXTEND %{
flag=0;
%}

COMPONENT source = Source_simple(
radius = 0.02,
dist = 3,
focus_xw = 0.01,
focus_yh = 0.01,
lambda0 = 6.0,
dlambda = 0.05,
flux = 1e8)
AT (0, 0, 0) RELATIVE arm

COMPONENT coll2 = Slit(
radius = 0.01)
AT (0, 0, 6) RELATIVE arm
EXTEND %{
flag=SCATTERED;
%}

COMPONENT detector = PSD_monitor(
nx = 128,
ny = 128,
filename = "PSD.dat",
xmin = -0.1,
xmax = 0.1,
ymin = -0.1,
ymax = 0.1)
AT (0, 0, 9.01) RELATIVE arm

END

```

The neutron and USERVARS in the instrument

v2.5: Global variables

```
double x, y, z, vx, vy, vz, t, sx, sy, sz, p;    double flag;
```

v3.0: particle struct, including any USERVARS like flag.

```
struct _struct_particle {
    double x,y,z; /* position [m] */
    double vx,vy,vz; /* velocity [m/s] */
    double sx,sy,sz; /* spin [0-1] */
    unsigned long randstate[7];
    double t, p; /* time, event weight */
    long long _uid; /* event ID */
    long _index; /* component index where to send this event */
    long _absorbed; /* flag set to TRUE when this event is to be removed/ignored */
    long _scattered; /* flag set to TRUE when this event has interacted with the last component instance */
    long _restore; /* set to true if neutron event must be restored */
    // user variables:
    double flag;
};
typedef struct _struct_particle _class_particle;
```

Can be probed using e.g. Monitor_nD with user1="flag" which uses the function

```
double particle_getvar(_class_particle *p, char *name, int *suc)
```

also works with e.g. "x"

The neutron and USERVARS in the instrument

v2.5: Global variables

```
double x, y, z, vx, vy, vz, t, sx, sy, sz, p;    double flag;
```

RNG state is a thread-variable contained on the `_particle` struct. Was earlier a global state in CPU settings

v3.0: particle struct, including any USERVARS like flag.

```
struct _struct_particle {
    double x,y,z; /* position [m] */
    double vx,vy,vz; /* velocity [m/s] */
    double sx,sy,sz; /* spin [0-1] */
    unsigned long randstate[7];
    double t, p; /* time, event weight */
    long long _uid; /* event ID */
    long _index; /* component index where to send this event */
    long _absorbed; /* flag set to TRUE when this event is to be removed */
    long _scattered; /* flag set to TRUE when this event has interacted w
    long _restore; /* set to true if neutron event must be restored */
    // user variables:
    double flag;
};
typedef struct _struct_particle _class_particle;
```

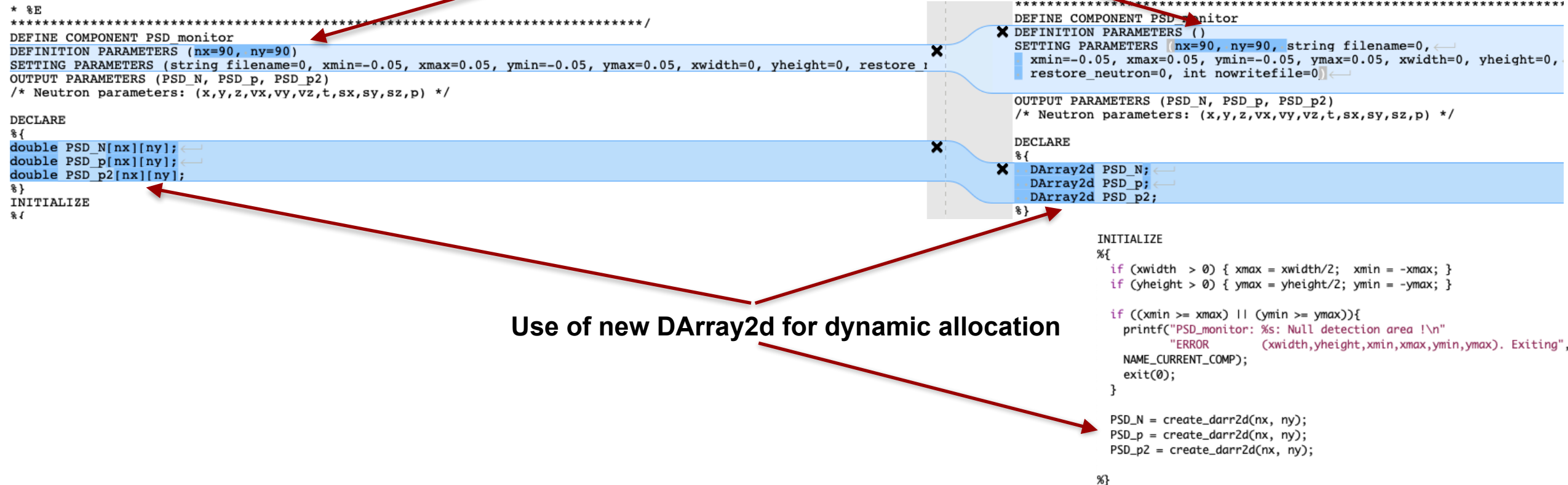
Side-effect:
Every function in TRACE that uses random numbers must have `_particle` in the footprint

Particle state data are not global:
Don't use `RESTORE_NEUTRON` in TRACE to do a **local** restore, the macro only raises a flag



PSD has several changes

No more DEFINITION PARAMETERS



```

* %E
*****
DEFINE COMPONENT PSD_monitor
DEFINITION PARAMETERS (nx=90, ny=90)
SETTING PARAMETERS (string filename=0, xmin=-0.05, xmax=0.05, ymin=-0.05, ymax=0.05, xwidth=0, yheight=0, restore_neutron=0, int nowritefile=0)
OUTPUT PARAMETERS (PSD_N, PSD_p, PSD_p2)
/* Neutron parameters: (x,y,z,vx,vy,vz,t,sx,sy,sz,p) */

DECLARE
%{
double PSD_N[nx][ny];
double PSD_p[nx][ny];
double PSD_p2[nx][ny];
%}
INITIALIZE
%{
*****
DEFINE COMPONENT PSD_monitor
DEFINITION PARAMETERS ( )
SETTING PARAMETERS (nx=90, ny=90, string filename=0,
xmin=-0.05, xmax=0.05, ymin=-0.05, ymax=0.05, xwidth=0, yheight=0,
restore_neutron=0, int nowritefile=0)
OUTPUT PARAMETERS (PSD_N, PSD_p, PSD_p2)
/* Neutron parameters: (x,y,z,vx,vy,vz,t,sx,sy,sz,p) */

DECLARE
%{
DArray2d PSD_N;
DArray2d PSD_p;
DArray2d PSD_p2;
%}

INITIALIZE
%{
if (xwidth > 0) { xmax = xwidth/2; xmin = -xmax; }
if (yheight > 0) { ymax = yheight/2; ymin = -ymax; }

if ((xmin >= xmax) || (ymin >= ymax)){
printf("PSD_monitor: %s: Null detection area !\n",
"ERROR (xwidth,yheight,xmin,xmax,ymin,ymax). Exiting",
NAME_CURRENT_COMP);
exit(0);
}

PSD_N = create_darr2d(nx, ny);
PSD_p = create_darr2d(nx, ny);
PSD_p2 = create_darr2d(nx, ny);

%}

```

Use of new DArray2d for dynamic allocation

PSD lots of changes

```
TRACE
%{
  int i,j;

  PROP_Z0;
  if (x>xmin && x<xmax && y>ymin && y<ymin){
  {
    i = floor((x - xmin)*nx/(xmax - xmin));
    j = floor((y - ymin)*ny/(ymax - ymin));
    PSD_N[i][j]++;
    PSD_p[i][j] += p;
    PSD_p2[i][j] += p*p;
    SCATTER;
  }
  if (restore_neutron) {
    RESTORE_NEUTRON(INDEX_CURRENT_COMP, x, y, z, vx, vy, vz, t, sx, sy, sz, p);
  }
}
```

```
TRACE
%{
  PROP_Z0;
  if (x>xmin && x<xmax && y>ymin && y<ymin){
    int i = floor((x - xmin)*nx/(xmax - xmin));
    int j = floor((y - ymin)*ny/(ymax - ymin));

    double p2 = p*p;
    #pragma acc atomic
    {
      PSD_N[i][j] = PSD_N[i][j]+1;
    }
    #pragma acc atomic
    {
      PSD_p[i][j] = PSD_p[i][j]+p;
    }
    #pragma acc atomic
    {
      PSD_p2[i][j] = PSD_p2[i][j] + p2;
    }
    SCATTER;
  }
  if (restore_neutron) {
    RESTORE_NEUTRON(INDEX_CURRENT_COMP, x, y, z, vx, vy, vz, t, sx, sy, sz, p);
  }
}
```

Enabling atomic writes on the detector arrays

```
PROP_Z0;
if (x>xmin && x<xmax && y>ymin && y<ymin){
  int i = floor((x - xmin)*nx/(xmax - xmin));
  int j = floor((y - ymin)*ny/(ymax - ymin));

  double p2 = p*p;
  #pragma acc atomic
  {
    PSD_N[i][j] = PSD_N[i][j]+1;
  }
  #pragma acc atomic
  {
    PSD_p[i][j] = PSD_p[i][j]+p;
  }
  #pragma acc atomic
  {
    PSD_p2[i][j] = PSD_p2[i][j] + p2;
  }
  SCATTER;
}
```

Compiler settings used for GPU:

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



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

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

Give accel debug information

Main “enable GPU”/OpenACC switch

(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`

Defaults for “GPU neutron loops”:

1) non-funnelled

GPU_INNERLOOP=2147483647

2) funnelled

GPU_FUNNEL_INNERLOOP=1024*1024

For CPU/threading, use below settings, with `-DMULTICORE` e.g. printf's are not nullified in TRACE

`nvc -ta=multicore -Minfo=accel -DOPENACC (-DMULTICORE)`

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...)

Highlights of comps that work differently

- Monitor_nD
uservars are strings user1="flag", they use `_particle_getvar` to access instrument USERVERS
- MCPL_input and MCPL_output
do most of their work in INIT/FINALLY - buffers transferred for TRACE use
- PowderN + Single_crystal + Isotropic_sqw
don't check for "same particle as before"
- in SPLIT cases, no particle state info is kept
(we could potentially use `_particle` and "USERVERS" injected from the comps...)

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



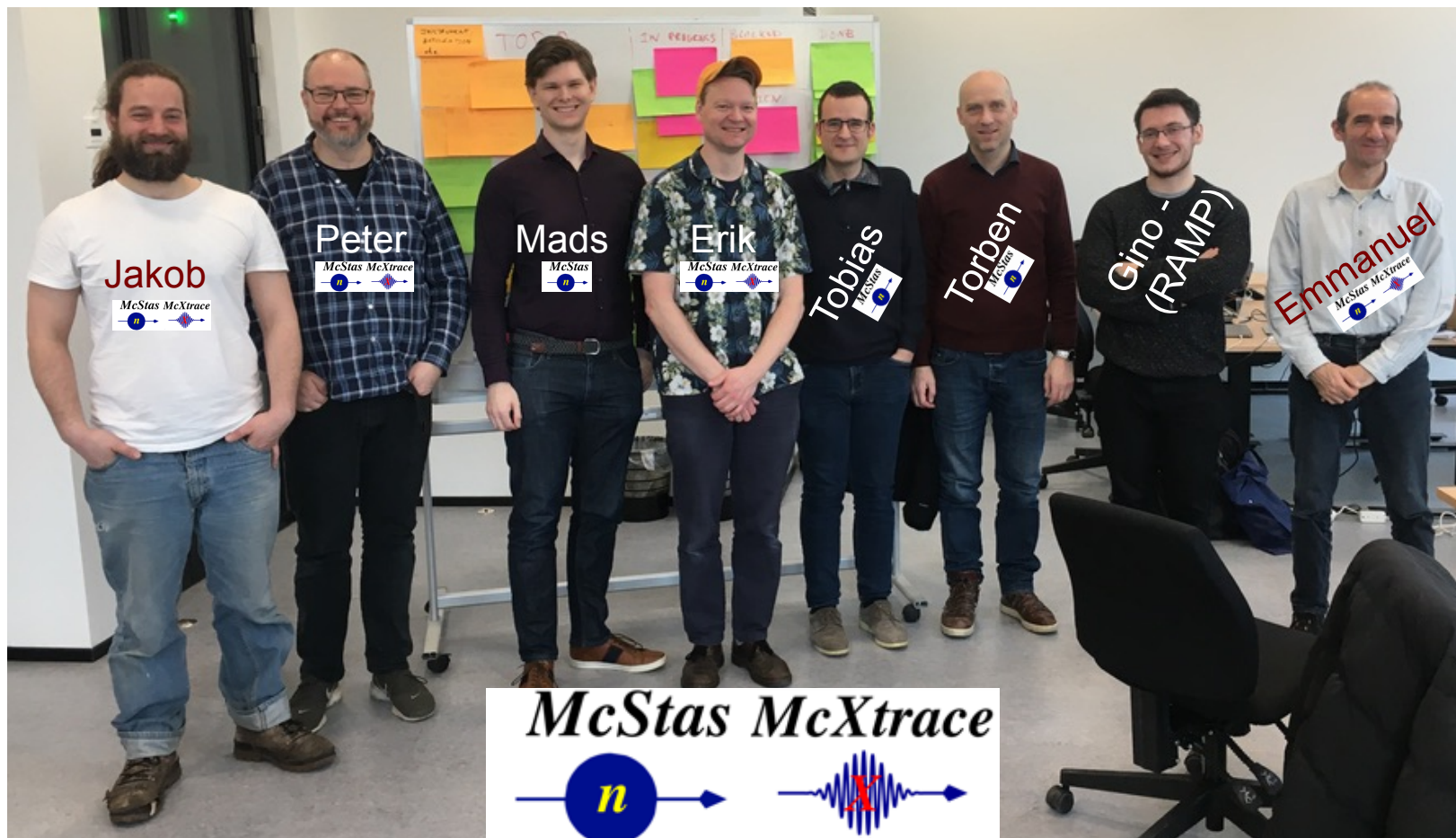
Alexey Romanenko



Guido Juckeland



Sebastian von Alfthan



McXtrace



McStas

