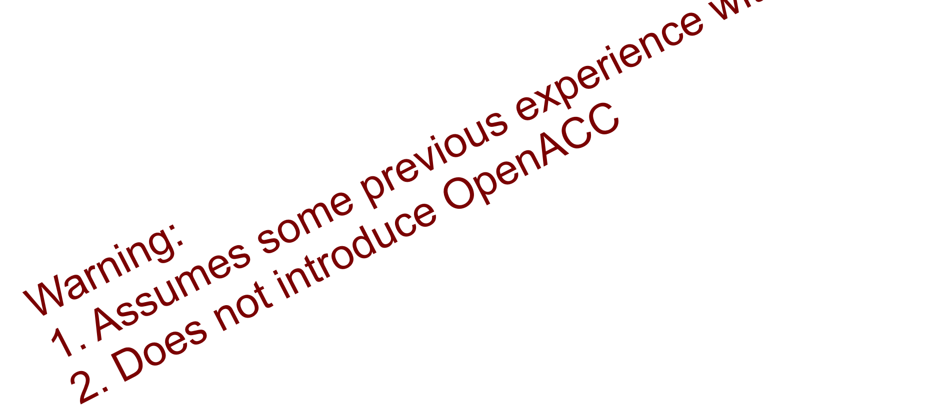
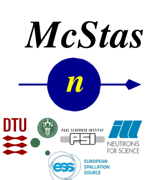
Peter Willendrup and Erik Knudsen DTU Physics

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

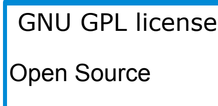
Agenda

- 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 needs doing on an instrument / component?
- What does not work

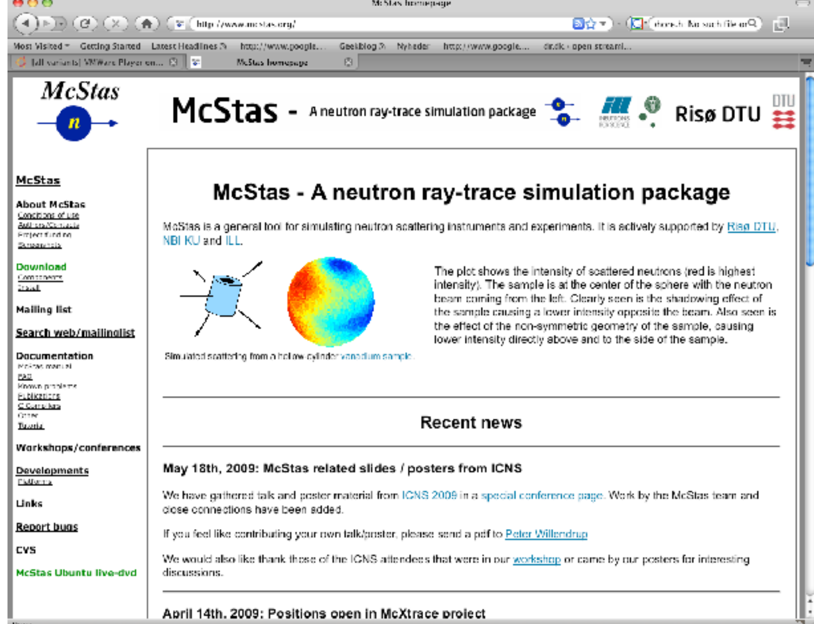


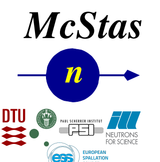


McStas Introduction



- Flexible, general simulation utility for neutron scattering experiments.
- Original design for Monte carlo Simulation of triple axis spectrometers
- Developed at DTU Physics, ILL, PSI, Uni CPH, ESS DMSC
- V. 1.0 by K Nielsen & K Lefmann (1998) RISØ
- Currently 2.5+1 people full time plus students





Project website at

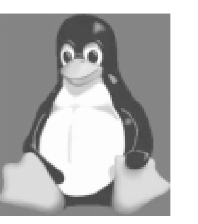
http://www.mcstas.org

mcstas-users@mcstas.org mailinglist

McStas overview

- Portable code (Unix/Linux/Mac/Windoze)
  - Ran on everything from iPhone to 1000+ node cluster!

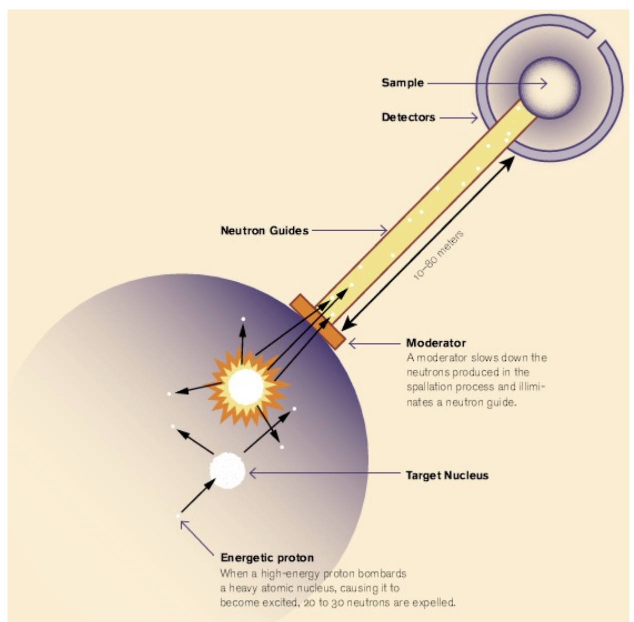
CPU's

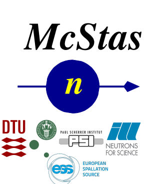


+ NVIDIA GPU's

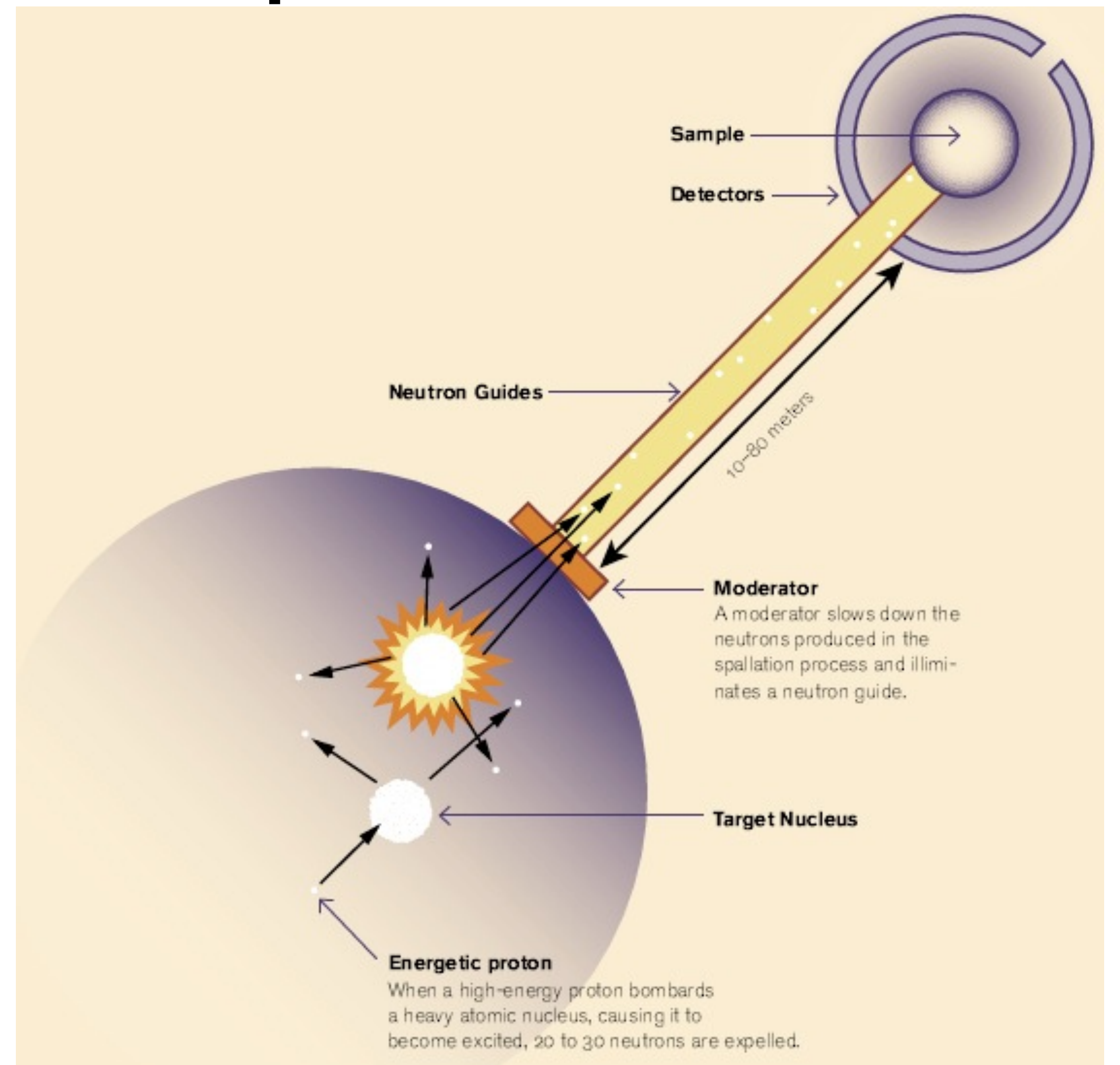
- 'Component' files (~200) inserted from library
  - Sources
  - Optics
  - Samples
  - Monitors
  - If needed, write your own comps

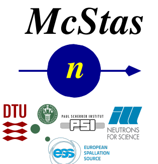
- DSL + ISO-C code gen.



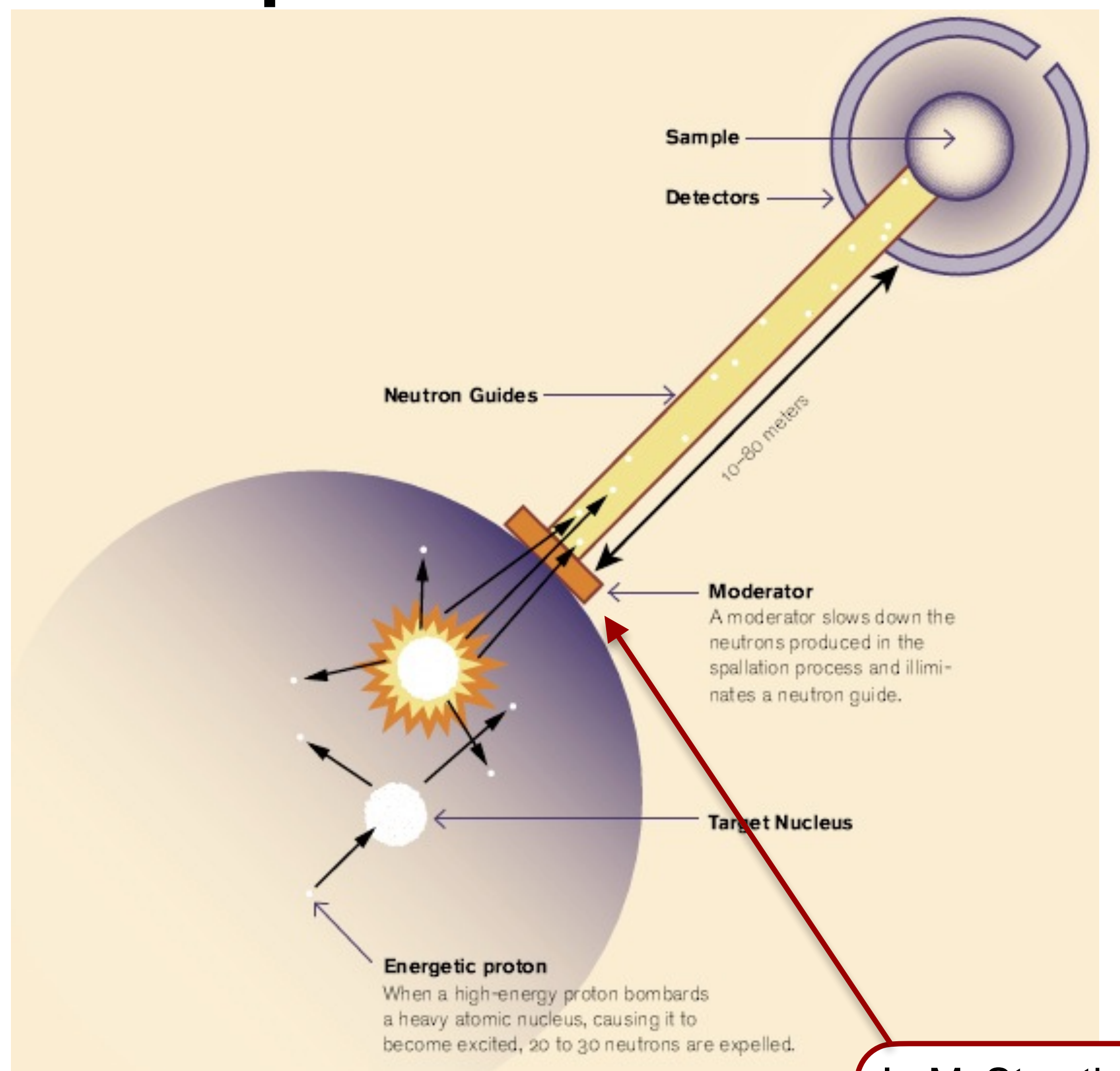


Components of neutron instruments

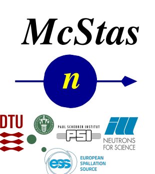




Components of neutron instruments

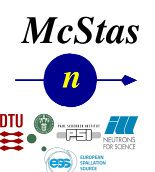


In McStas the moderator is the “source”



Components of neutron instruments

In McStas the moderator is the "source"



Components of neutron instruments

Sample
Detectors
Neutron Guides
10–80 meters
Moderator
A moderator slow
neutrons produce
spallation process
nates a neutron g
Target Nucleus
Energetic proton
When a high-energy proton bombards
a heavy atomic nucleus, causing it to
become excited, 20 to 30 neutrons are expelled.

The sample:

Crystalline, powders, liquids, micelles, structures to image, inelastic features like phonons…

In McStas the moderator is the “source”

Components of neutron instruments

Sample
Detectors
Neutron Guides
10–80 meters
Moderator
A moderator slow
neutrons produce
spallation process
nates a neutron g
Target Nucleus
Energetic proton
When a high-energy proton bombards
a heavy atomic nucleus, causing it to
become excited, 20 to 30 neutrons are expelled.

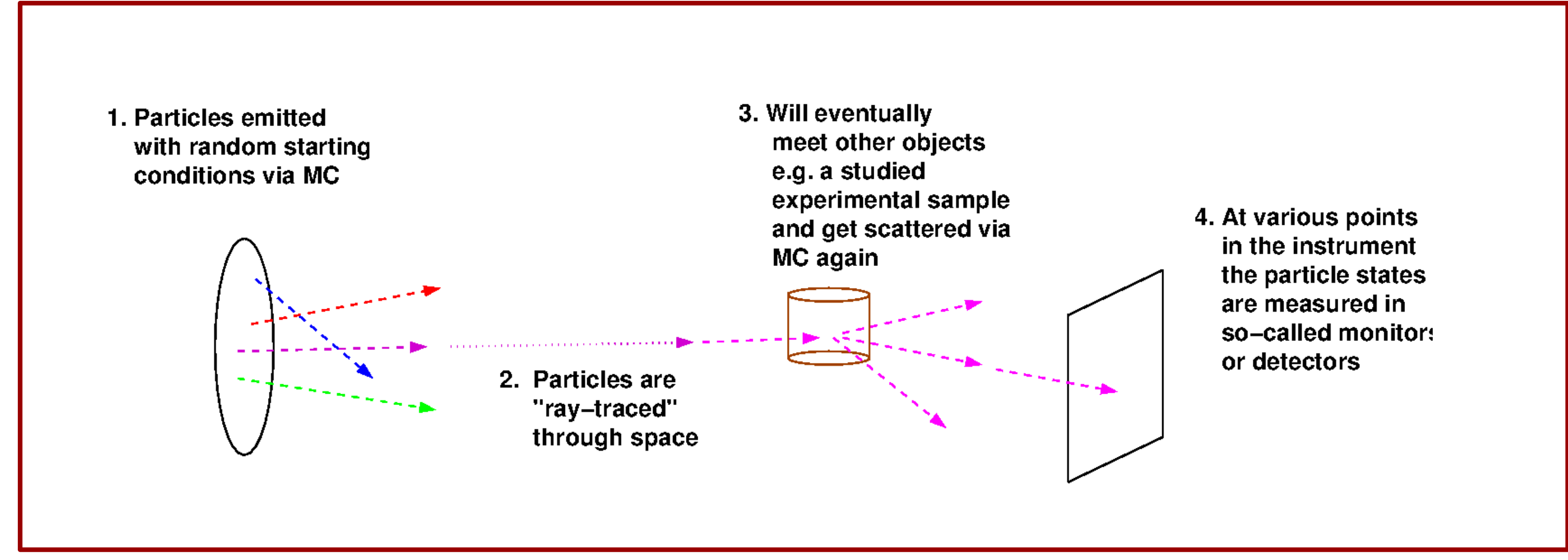
Detectors are “monitors” in McStas. Mostly they act as “perfect probes” and can be positioned thought your instrument gathering 1D/2D/ event lists…

The sample:

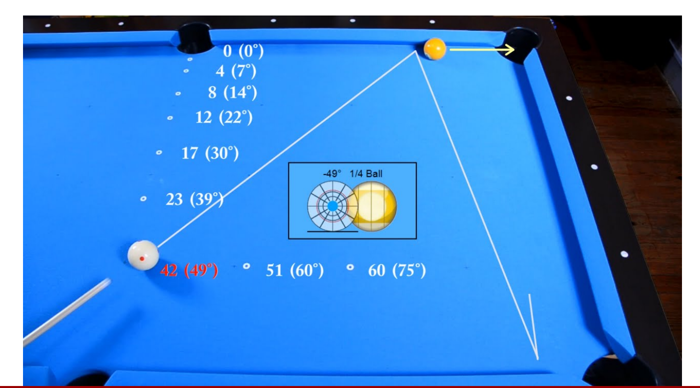
Crystalline, powders, liquids, micelles, structures to image, inelastic features like phonons…

In McStas the moderator is the “source”

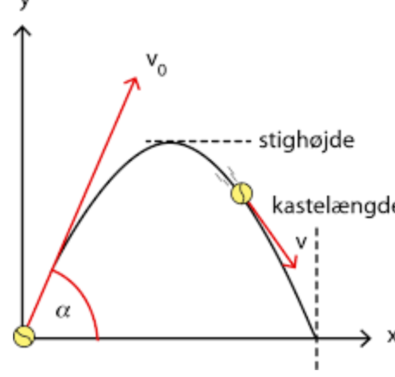
In the big picture, McStas is this…



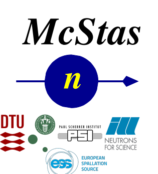
- Classical Newtonian mechanics, i.e.



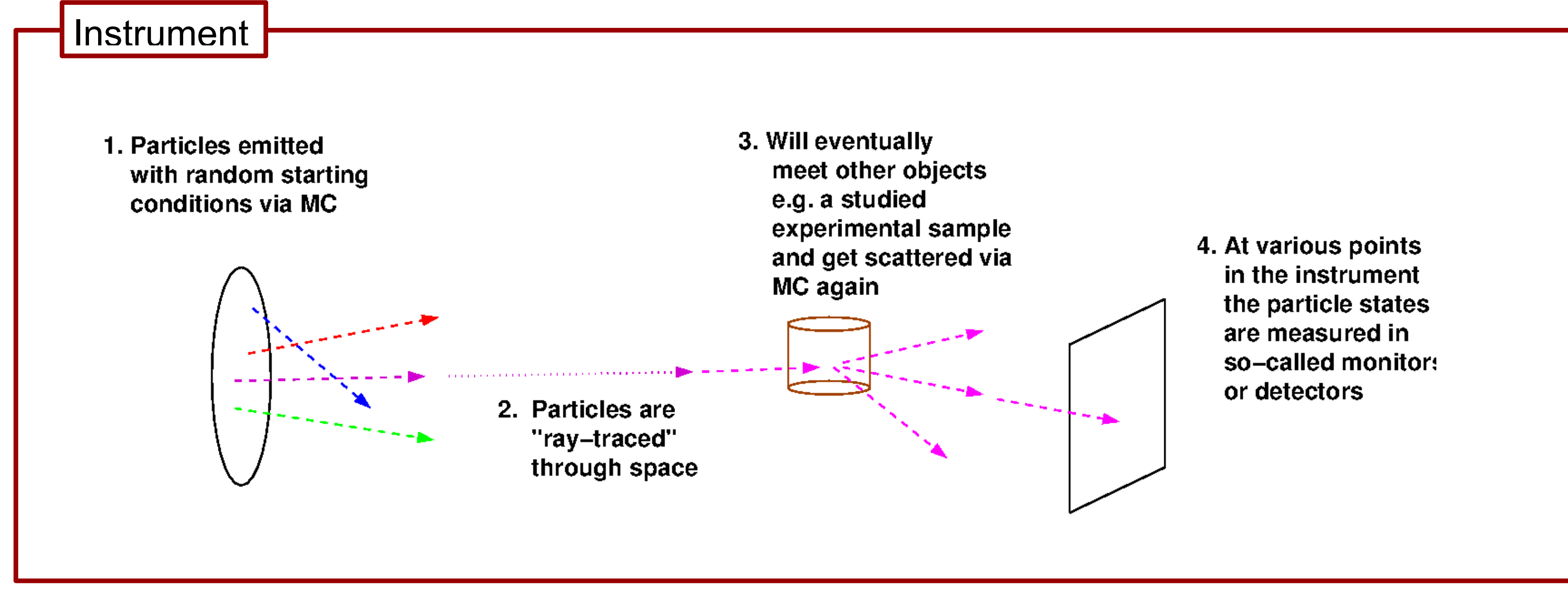
and



- (independent, particles though…)

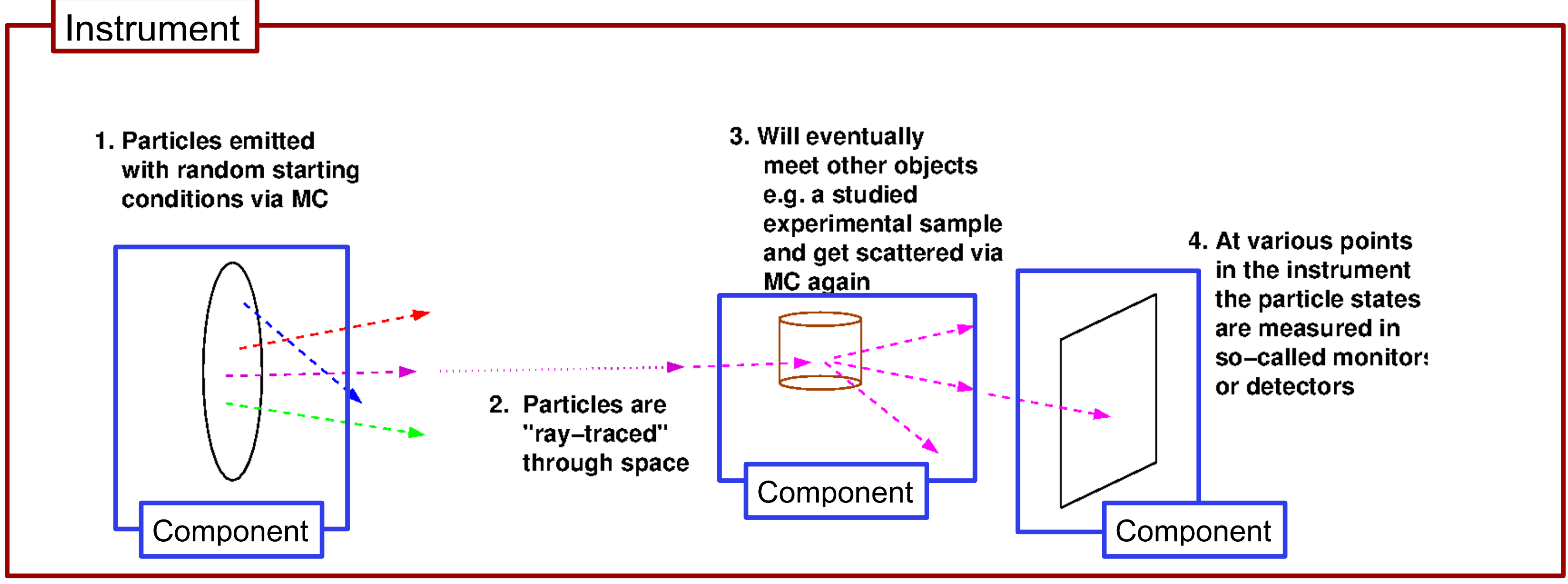


In the big picture, McStas is this…



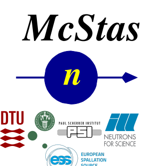
The instrument defines our “lab coordinate system”

In the big picture, McStas is this…

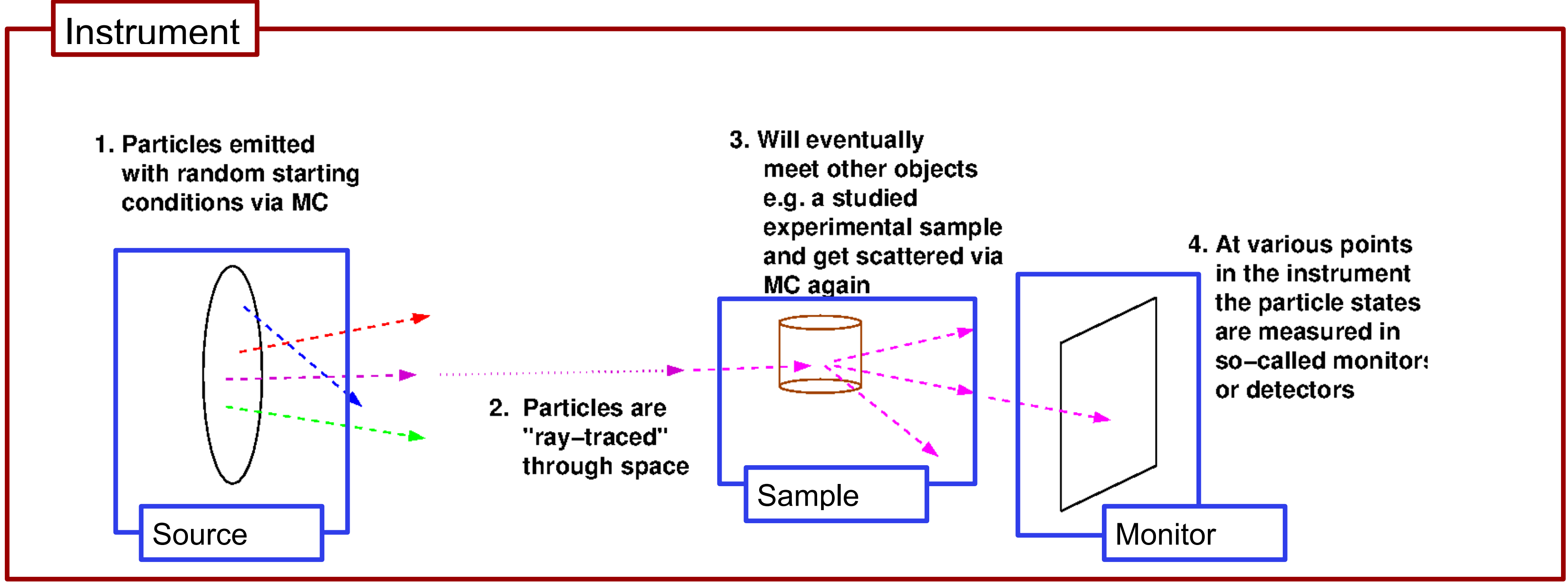


The instrument defines our “lab coordinate system”

The components define devices or features available in our instrument

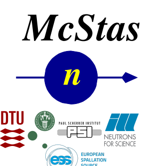


In the big picture, McStas is this…



The instrument defines our “lab coordinate system”

The components define devices or features available in our instrument - they have different function



In the big picture, McStas is this…

Instrument

1. Particles emitted with random starting conditions via MC

2. Particles are "ray–traced" through space

3. Will eventually meet other objects e.g. a studied experimental sample and get scattered via MC again

4. At various points in the instrument the particle states are measured in so–called monitors or detectors

Source

Sample

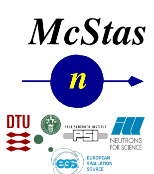
Monitor

Neutrons

The instrument defines our “lab coordinate system”

The components define devices or features available in our instrument - they have different function

Neutron particles are passed on from one component to the next, changing state under way



McStas is a Monte Carlo ray-tracer

Instrument

1. Particles emitted with random starting conditions via MC

2. Particles are "ray–traced" through space

3. Will eventually meet other objects e.g. a studied experimental sample and get scattered via MC again

4. At various points in the instrument the particle states are measured in so–called monitors or detectors

Source

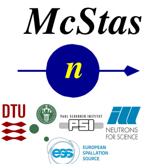
Sample

Monitor

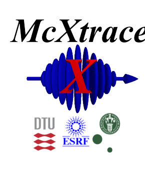
Neutrons

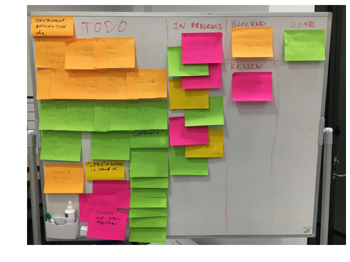
- Classical Newtonian mechanics, i.e.

- (independent, particles though…)



Main events on timeline of road toward GPU





2017: E. Farhi initial cogen modernisation

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

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

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!

McStas / McXtrace instrument simulation

CPU MPI 4 cores (95 % usage) — 16.12 s (Single core 56.0 s)

GPU 5% usage — 5.43 s

Comparison A vs B: Monitor: D1_SC1_Out.psd

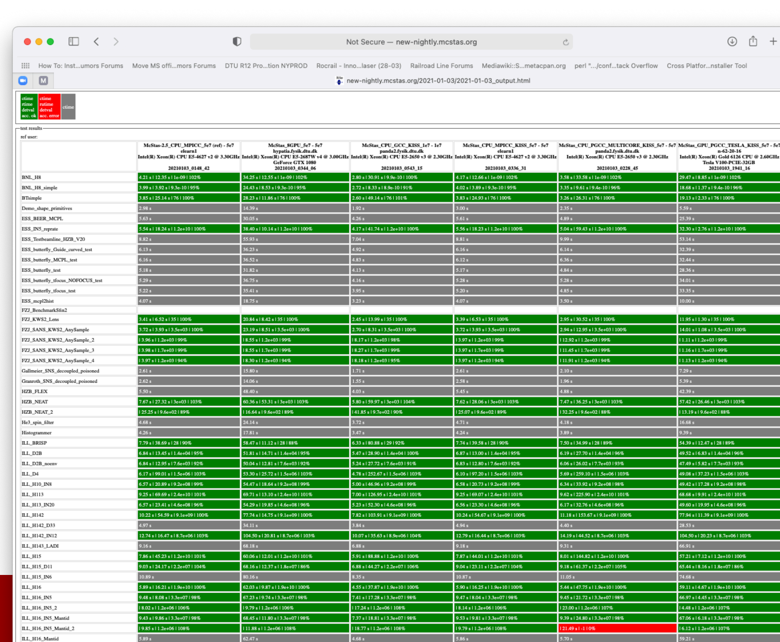
mentor: Vishal Metha — NVIDIA

hackathon org.: Guido Juckeland — HZDR, HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

mentor: Christian Hundt — NVIDIA

hackathon org.: Sebastian Von Alfthan — CSC

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



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.)

Figure 2

File Edit View Insert Tools Desktop Window Help

Performance in the PSI_source case - relative speedup

speed relative to gcc serial

Problem size / neutrons

PGCC Quadro T2000 doubles -O0
PGCC GeForce GTX 1080 -O0
PGCC Quadro T2000 floats -O0
PGCC GeForce GTX 1080 floats -O0
PGCC Volta V100 -O0
PGCC multicore -O2, 8 cores
PGCC -O2 cpu serial
gcc -O2 cpu serial
PGCC -O2 mpi, 8 cores

Speedup

Looks like a factor of ~600

10^3
~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 November 2019

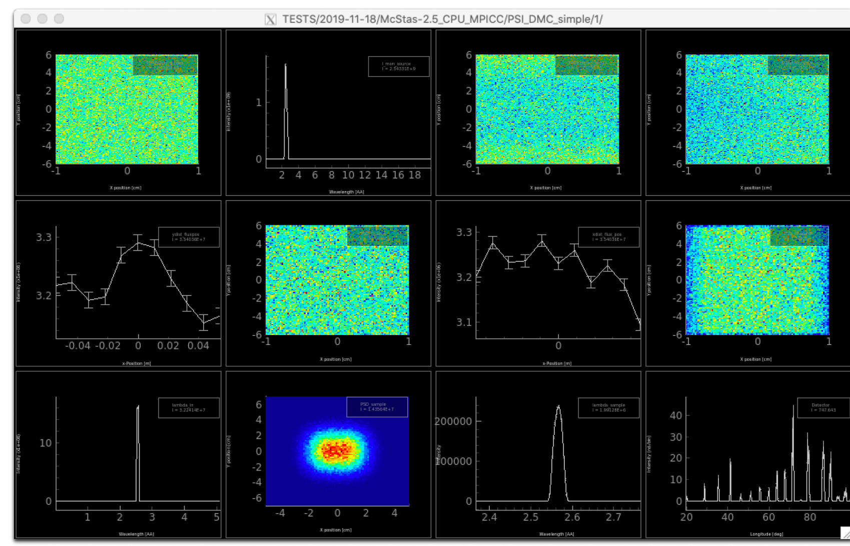
9 instruments fully ported, also realistic ones like PSI_DMC

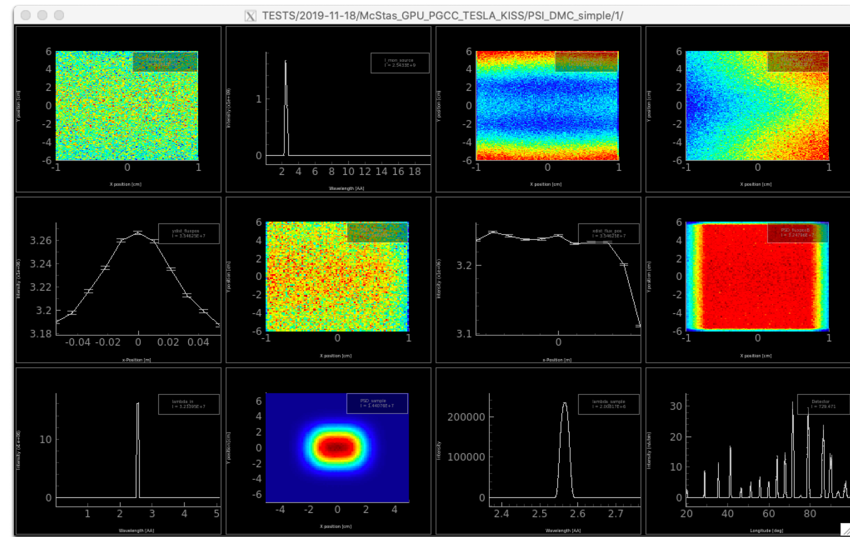
(Aug 2020: 99 instrs)

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

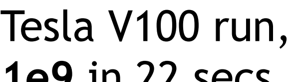
(1-core run, **1e9** would be 2000 secs)

VS.



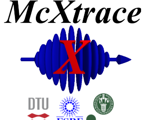


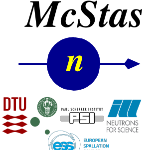
Tesla V100 run, **1e9** in 22 secs



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

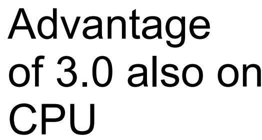
- If problem has the right size / complexity, GPU via OpenACC is great!





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**



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

- **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 randnom numbers, not billions)

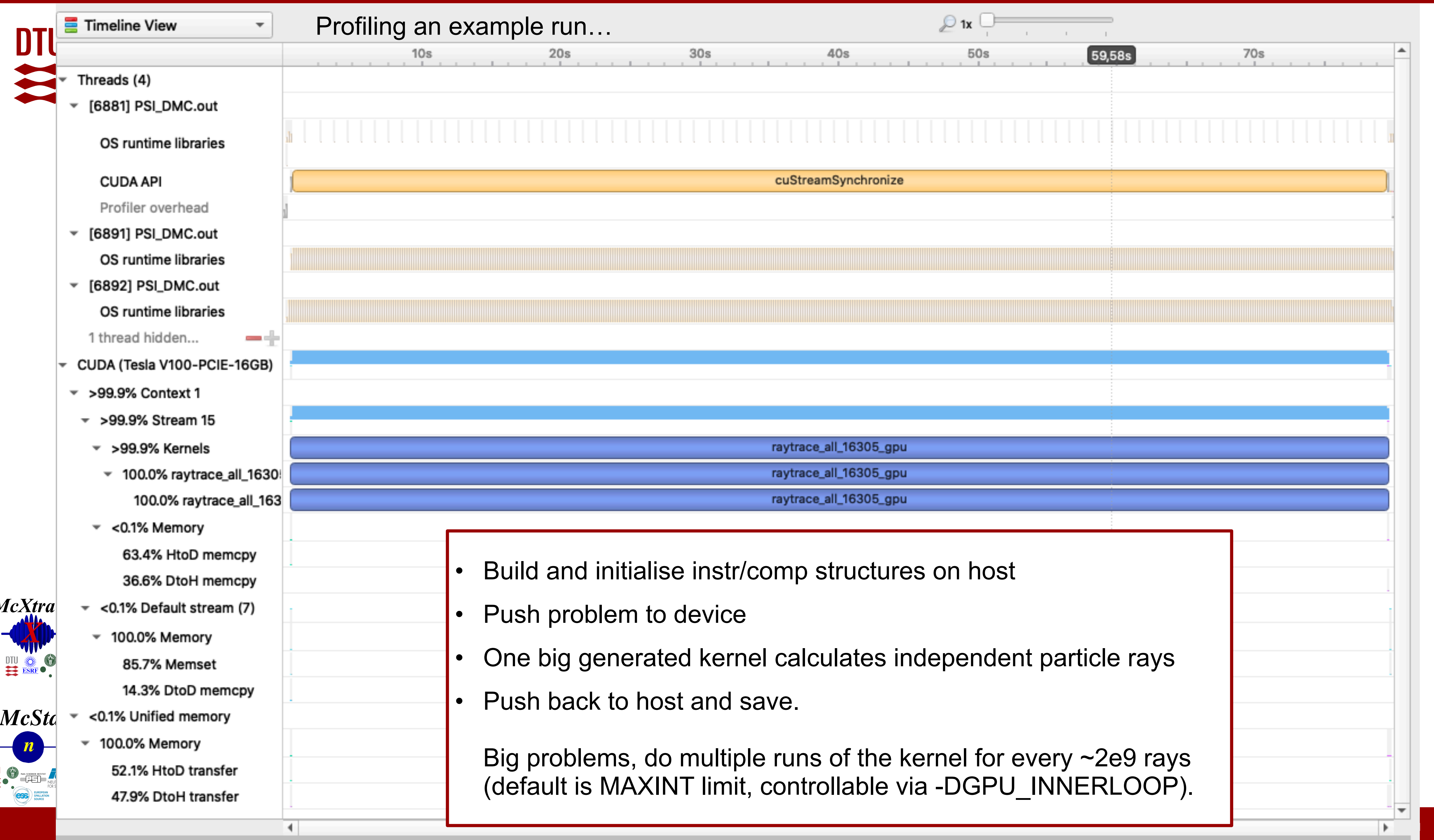
- Complete change to **dynamic** monitor-arrays

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)


Timeline View
Profiling an example run…
1x
10s
20s
30s
40s
50s
59,58s
70s
Threads (4)
[6881] PSI_DMC.out
OS runtime libraries
CUDA API
cuStreamSynchronize
Profiler overhead
[6891] PSI_DMC.out
OS runtime libraries
[6892] PSI_DMC.out
OS runtime libraries
1 thread hidden...
CUDA (Tesla V100-PCIE-16GB)
>99.9% Context 1
>99.9% Stream 15
>99.9% Kernels
raytrace_all_16305_gpu
100.0% raytrace_all_1630
raytrace_all_16305_gpu
100.0% raytrace_all_163
raytrace_all_16305_gpu
<0.1% Memory
63.4% HtoD memcpy
36.6% DtoH memcpy
<0.1% Default stream (7)
100.0% Memory
85.7% Memset
14.3% DtoD memcpy
<0.1% Unified memory
100.0% Memory
52.1% HtoD transfer
47.9% DtoH transfer
• 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).
DTU
McXtrace
McStas

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.

- 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;
```

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"

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;
```

McXtrace

McStas

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

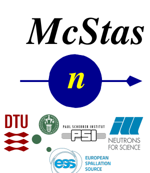
Side-effect:
Every function in TRACE that uses random numbers must have _particle in the footprint

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 wi
  long _restore;   /* set to true if neutron event must be restored */
  // user variables:
  double flag;
};
typedef struct _struct_particle _class_particle;
```

Also:

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



Source_simple minor changes

[Applications] Source_simple.comp — [Users] Source_simple.comp

Save Undo

[Application...imple.comp

Source_simple.comp

```
* centered at the beam (in order to improve MC-acceptance rate). The angular
* divergence is then given by the dimensions of the target.
* The neutron energy is uniformly distributed between lambda0-dlambda and
* lambda0+dlambda or between E0-dE and E0+dE.
* The flux unit is specified in n/cm2/s/st/energy unit (meV or Angs).
*
* This component replaces Source_flat, Source_flat_lambda,
* Source_flux and Source_flux_lambda.
*
* Example: Source_simple(radius=0.1, dist=2, focus_xw=.1, focus_yh=.1, E0=14, dE=2)
*
* %P
* radius: [m]                    Radius of circle in (x,y,0) plane where neutrons are generated
* yheight: [m]                   Height of rectangle in (x,y,0) plane where neutrons are generat
* xwidth: [m]                    Width of rectangle in (x,y,0) plane where neutrons are generate
* target_index: [1]              relative index of component to focus at, e.g. next is +1 this :
* dist: [m]                      Distance to target along z axis.
* focus_xw: [m]                  Width of target
* focus_yh: [m]                  Height of target
* E0: [meV]                      Mean energy of neutrons.
* dE: [meV]                      Energy half spread of neutrons (flat or gaussian sigma).
* lambda0: [AA]                  Mean wavelength of neutrons.
* dlambda: [AA]                  Wavelength half spread of neutrons.
* flux: [1/(s*cm**2*st*energy unit)]  flux per energy unit, Angs or meV if flux=0, the source emits :
* gauss: [1]                     Gaussian (1) or Flat (0) energy/wavelength distribution
*
* %E
*******************************************************************************/

DEFINE COMPONENT Source_simple
DEFINITION PARAMETERS ()
SETTING PARAMETERS (radius=0.1, yheight=0, xwidth=0,
dist=0, focus_xw=.045, focus_yh=.12,
E0=0, dE=0, lambda0=0, dlambda=0,
flux=1, gauss=0, int target_index=+1)
OUTPUT PARAMETERS (pmul,square,srcArea)
/* Neutron parameters: (x,y,z,vx,vy,vz,t,sx,sy,sz,p) */
DECLARE
%{
double pmul, srcArea;
int square;
double tx,ty,tz;
%}
INITIALIZE
%{
square = 0;
/* Determine source area */
if (radius && !yheight && !xwidth ) {
    square = 0;
    srcArea = PI*radius*radius;
  } else if(yheight && xwidth) {
    square = 1;
    srcArea = xwidth * yheight;
  }

  if (flux) {
    pmul=flux*1e4*srcArea/mcget_ncount();
    if (dlambda)
      pmul *= 2*dlambda;
    else if (dE)
      pmul *= 2*dE;
  } else {
    gauss = 0;
    pmul=1.0/(mcget_ncount()*4*PI);
  }

  if (target_index && !dist)
  {
    Coords ToTarget;
    ToTarget = coords_sub(POS_A_COMP_INDEX(INDEX_CURRENT_COMP+target_index),POS_A_CURRENT_COMP);
```

Source_simple.comp

```
* centered at the beam (in order to improve MC-acceptance rate). The angular
* divergence is then given by the dimensions of the target.
* The neutron energy is uniformly distributed between lambda0-dlambda and
* lambda0+dlambda or between E0-dE and E0+dE.
* The flux unit is specified in n/cm2/s/st/energy unit (meV or Angs).
*
* This component replaces Source_flat, Source_flat_lambda,
* Source_flux and Source_flux_lambda.
*
* Example: Source_simple(radius=0.1, dist=2, focus_xw=.1, focus_yh=.1, E0=14,
*
* %P
* radius: [m]                    Radius of circle in (x,y,0) plane where
* yheight: [m]                   Height of rectangle in (x,y,0) plane wh
* xwidth: [m]                    Width of rectangle in (x,y,0) plane whe
* target_index: [1]              relative index of component to focus at
* dist: [m]                      Distance to target along z axis.
* focus_xw: [m]                  Width of target
* focus_yh: [m]                  Height of target
* E0: [meV]                      Mean energy of neutrons.
* dE: [meV]                      Energy half spread of neutrons (flat or
* lambda0: [AA]                  Mean wavelength of neutrons.
* dlambda: [AA]                  Wavelength half spread of neutrons.
* flux: [1/(s*cm**2*st*energy unit)]  flux per energy unit, Angs or meV if fl
* gauss: [1]                     Gaussian (1) or Flat (0) energy/wavelen
*
* %E
*******************************************************************************

DEFINE COMPONENT Source_simple
DEFINITION PARAMETERS ()
SETTING PARAMETERS (radius=0.1, yheight=0, xwidth=0,
dist=0, focus_xw=.045, focus_yh=.12,
E0=0, dE=0, lambda0=0, dlambda=0,
flux=1, gauss=0, int target_index=+1)
OUTPUT PARAMETERS ()
/* Neutron parameters: (x,y,z,vx,vy,vz,t,sx,sy,sz,p) */
DECLARE
%{
double pmul;
double srcArea;
int square;
double tx;
double ty;
double tz;
%}
INITIALIZE
%{
square = 0;
/* Determine source area */
if (radius && !yheight && !xwidth ) {
    square = 0;
    srcArea = PI*radius*radius;
  } else if(yheight && xwidth) {
    square = 1;
    srcArea = xwidth * yheight;
  }

  if (flux) {
    pmul=flux*1e4*srcArea/mcget_ncount();
    if (dlambda)
      pmul *= 2*dlambda;
    else if (dE)
      pmul *= 2*dE;
  } else {
    gauss = 0;
    pmul=1.0/(mcget_ncount()*4*PI);
  }

  if (target_index && !dist)
```

Unicode (UTF-8)   Ln 56, Col 1   Unicode

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_n
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<ymax)
  {
    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<ymax){
    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<ymax){
  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;
```

Samples required a good deal of (detective) work…

Key issue: (mis-) Use of / component DECLARE variables in TRACE for storing particle-dependent information, e.g. reflection list in PowderN etc. must be avoided.

Solutions:

1) Make local thread-variables, e.g. →

```
// Variables calculated within thread for thread purpose only
char type = '\0';
int itype = 0;
double d_phi_thread = d_phi;
// These ones are injected back to struct at the end of TRACE in non-OpenACC case
int nb_reuses = line_info.nb_reuses;
int nb_refl = line_info.nb_refl;
int nb_refl_count = line_info.nb_refl_count;
double vcache = line_info.v;
double Nq = line_info.Nq;
double v_min = line_info.v_min;
double v_max = line_info.v_max;
double lfree = line_info.lfree;
double neutron_passed = line_info.neutron_passed;
long   xs_compute = line_info.xs_compute;
long   xs_reuse   = line_info.xs_reuse;
long   xs_calls   = line_info.xs_calls;
int    flag_warning = line_info.flag_warning;
double dq = line_info.dq;

#ifdef OPENACC
#ifdef USE_OFF
off_struct thread_offdata = offdata;
#endif
#else
#define thread_offdata offdata
#endif
```

2) Where meaningful, one could make atomic sections ala the monitors

Side-effect of thread-local vars:

Next neutron(s) in a SPLIT are no longer aware of e.g. available powder lines.

Potential, future solution: Mechanism to inject comp-specific code in the _particle

New monitor-tools for debugging use

- **Event_monitor_simple(nevents=1e6)**
  - basic non-Monitor_nD event monitor. Writes a “log” file, independent from detector_out macros

- **Flex_monitor_1D** , **Flex_monitor_2D**, **Flex_monitor_3D**,
  simple 1/2/3D “uservar” monitors tapping into the instrument USERVARS ala Monitor_nD

- Useful for **debugging** even **component** internals:
  On McStas 3, if same ncount, same seed, same level of MPI parallelisation, the output should be **identical** on CPU and GPU

```
#pragma acc routine seq
_class_Source_simple *class_Source_simple_trace(_class_Source_simple *_comp
  , _class_particle *_particle) {
  ABSORBED=SCATTERED=RESTORE=0;

  #define radius (_comp->_parameters.radius)
  #define yheight (_comp->_parameters.yheight)
  #define xwidth (_comp->_parameters.xwidth)
  #define dist (_comp->_parameters.dist)
  #define focus_xw (_comp->_parameters.focus_xw)
  #define focus_yh (_comp->_parameters.focus_yh)
  #define E0 (_comp->_parameters.E0)
  #define dE (_comp->_parameters.dE)
  #define lambda0 (_comp->_parameters.lambda0)
  #define dlambda (_comp->_parameters.dlambda)
  #define flux (_comp->_parameters.flux)
  #define gauss (_comp->_parameters.gauss)
  #define target_index (_comp->_parameters.target_index)
  #define pmul (_comp->_parameters.pmul)
  #define square (_comp->_parameters.square)
  #define srcArea (_comp->_parameters.srcArea)
  #define tx (_comp->_parameters.tx)
  #define ty (_comp->_parameters.ty)
  #define tz (_comp->_parameters.tz)
  SIG_MESSAGE("[_source_trace] component source=Source_simple() TRACE [Source_simple.comp:127]");
 double chi,E,lambda,v,r, xf, yf, rf, dx, dy, pdir;

 t=0;
 z=0;

 if (square == 1) {
   x = xwidth * (rand01() - 0.5);
   y = yheight * (rand01() - 0.5);
 } else {
   chi=2*PI*rand01();                          /* Choose point on source */
   r=sqrt(rand01())*radius;                    /* with uniform distribution. */
   x=r*cos(chi);
   y=r*sin(chi);
 }
 randvec_target_rect_real(&xf, &yf, &rf, &pdir,
                          tx, ty, tz, focus_xw, focus_yh, ROT_A_CURRENT_COMP, x, y, z, 2);
```

…. etc

Each component will correspond to a set of function. Trace is a GPU'ified function…

+ particle-loop and logic around, also running on GPU.

Init and finalisation codes run purely CPU.

"Scatter-gather" approach not far from what we do in MPI settings, i.e. :

GPU case:
N particles are calculated in parallel in N GPU threads. (Leave to OpenACC/device how many actually are running at one time)

CPU case:
N particles are calculated in M serial chunks over M processors.

Contains component trace section

(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

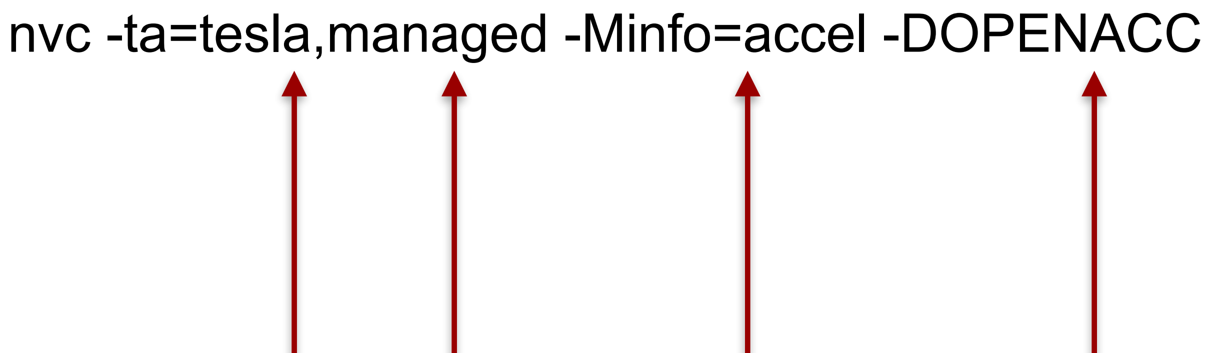
Give accel debug information

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.

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. printfs 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

Should give the SAME on CPU and GPU!

Porting an instrument to 3.0 and GPU

- **Instrument-level variables** that are to become particle-dependent "**flags**", e.g. for use in EXTEND WHEN must be put in the new section **USERVARS %{ double flag; %}**

- Use of **instrument input pars** in extend and WHEN must use **INSTRUMENT_GETPAR(varname)**

- **Non-flag instrument vars** to be used during TRACE / EXTEND / WHEN must be injected via **#pragma acc declare create(var)** and **#pragma acc update device(var)**

- **Declare-functions** to be used in **trace** (e.g. in an EXTEND) must have #pragma acc routine

Common error-messages:

- At compile-time, nvc is quite informative if using -Minfo:accel

```
PGC-S-0155-Invalid atomic expression
PGC-S-0000-Internal compiler error. Error: Detected unexpected atomic store opcode.
PGC-S-0155-External variables used in acc routine need to be in #pragma acc create() - flag
…
```

- At runtime, this indicates a GPU segfault

```
Failing in Thread:1
call to cuMemcpyDtoHAsync returned error 700: Illegal address during kernel execution
```

  - Often a symptom of “illegal access”, colliding memory, something isn’t thread-safe…

Should give the SAME on CPU and GPU!

Porting a comp to GPU

- **All pars must be setting parameters.**
  New types: string a="none" and vector b (either ={1,2,3,4} static init or via pointer.)
- **All function-declarations must be moved to SHARE**
- **DECLARE** must ***only*** contain variable **declarations**. All initialization resides in INITIALIZE
- **If the comp uses external libs** either
  - **Avoid use in component TRACE** (e.g. MCPL_input and output, handled in INIT/FINALLY)
  - **Use NOACC keyword** (e.g. Multilayer_sample use of GSL) - implies FUNNEL mode
- **Add #pragma acc routine** to functions to execute in TRACE
- **Functions that call rand01()** and friends **must include _class_particle *_particle in footprint.**
  (rand01() etc. are macros that carry thread-seed on _particle)s
- Generally, **don't store ANY** particle-derived vars on comp struct, **make local TRACE vars** instead.
  - Exception: Monitors, handle arrays in #pragma acc atomic clauses
- **Don't** use **RESTORE_NEUTRON** in TRACE to do a **local** restore, the macro only raises a flag

Highlights of comps that work differently

- Monitor_nD
  uservars are strings user1="flag", they use _particle_getvar to access instrument USERVARS

- MCPL_input and MCPL_output
  do most of their work in INIT/FINALLY - buffers transferred for TRACE use

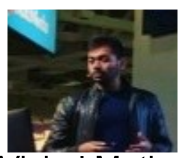
- PowderN + Single_crystal + Isotropic_sqw
  don't check for "same particle as before"
  - in SPLIT cases, no particle state info is kept
  (we could potentially use _particle and "USERVARS" injected from the comps…)

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)
- **Documentation** comes in the form of the released code + this set of slides…
- 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**
- **Compilation** with GCC 10 offloading support achieved May 2021 - but produces 0 on detectors…
  - hope for better GCC support and non-NVIDIA cards in 1-2 years

The team, Nvidia mentors and Hackathon hosts :-)

NVIDIA


Vishal Metha

Christian Hundt

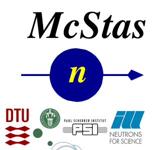
Alexey Romanenko

Jakob
Peter
Mads
Erik
Tobias
Torben
Gino - (RAMP)
Emmanuel

McStas McXtrace

Guido Juckeland

Sebastian von Alfthan



Backup slides…

What code-parts got #pragma acc

Obviously the **code-generator**…
mcstas/src/cogen.c.in

Examples:

25 **Instruments** that use various global vars in DECLARE that are neither input parameters or USERVARS

These **share/runtime** snippets:
(A good mix of everything)

share/adapt_tree-lib.c
share/interoff-lib.c
share/mccode-r.c
share/mccode-r.h
share/mccode_main.c
share/read_table-lib.c
share/r-interoff-lib.c
share/ESS_butterfly-geometry.c
share/ESS_butterfly-lib.c
share/cov-lib.c
share/monitor_nd-lib.c
share/mcstas-r.c
share/mcstas-r.h
share/pol-lib.c
share/ref-lib.c

Sources: (TRACE-functions in SHARE)

sources/Source_Maxwell_3.comp
sources/Source_gen.comp

Samples: (TRACE-functions in SHARE)

samples/Isotropic_Sqw.comp
samples/Magnon_bcc.comp
samples/Phonon_simple.comp
samples/PowderN.comp
samples/SANS_spheres2.comp
samples/Single_crystal.comp

Optics: (TRACE-functions in SHARE + declare create for Gauss structures)

optics/Elliptic_guide_gravity.comp
optics/FermiChopper.comp
optics/Guide_gravity.comp
optics/Monochromator_curved.comp
optics/Monochromator_flat.comp

All **monitors**:

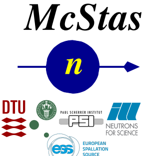
#pragma acc atomic sections for arrays

Misc:
atomic capture for insertion in array of particle events

misc/MCPL_output.comp

Contrib comps: (atomics in mon's, #pragma acc routine for TRACE-funcs)

contrib/FermiChopper_ILL.comp
contrib/Guide_honeycomb.comp
contrib/ISIS_moderator.comp
contrib/Lens.comp
contrib/Mirror_Elliptic.comp
contrib/Mirror_Parabolic.comp
contrib/PSD_Detector.comp
contrib/PSD_monitor_rad.comp
contrib/SNS_source.comp
contrib/SNS_source_analytic.comp
contrib/ViewModISIS.comp



New RNG ‘KISS’

- We couldn’t easily port Mersenne Twister
- Experimenting with curand showed huge overhead for our relative small number of random numbers
- An RNG ‘state’ carried with each particle - bonus: same seed gives same numbers even when comparing between CPU and GPU
- Required patching prototype of ALL functions making use of e.g. rand01()
- New RNG is simple, fast and “good enough”: Reproduces results of 2.7 over all of the example suite, see http://new-nightly.mcstas.org
-

OpenACC

PSD lots of changes

[v2.5comps] PSD_monitor.comp* — [v3.0comps] PSD_monitor.comp

Save Undo

[v2.5comps...onitor.comp

PSD_monitor.comp

```
* filename: [string]    Name of file in which to store the detector image
* restore_neutron: [1]  If set, the monitor does not influence the neutron state
* nowritefile: [1]      If set, monitor will skip writing to disk
*
* OUTPUT PARAMETERS:
*
* PSD_N: []             Array of neutron counts
* PSD_p: []             Array of neutron weight counts
* PSD_p2: []            Array of second moments
*
* %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_n
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
%{
int i,j;

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);
  }

  for (i=0; i<nx; i++)
   for (j=0; j<ny; j++)
   {
    PSD_N[i][j] = 0;
    PSD_p[i][j] = 0;
    PSD_p2[i][j] = 0;
   }
  %}
TRACE
  %{
    int i,j;

    PROP_Z0;
    if (x>xmin && x<xmax && y>ymin && y<ymax)
    {
      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);
    }
  %}
SAVE
  %{
    if (!nowritefile) {
      DETECTOR_OUT_2D(
          "PSD monitor",
          "X position [cm]",
          "Y position [cm]",
          xmin*100.0, xmax*100.0, ymin*100.0, ymax*100.0,
          nx, ny,
          &PSD_N[0][0],&PSD_p[0][0],&PSD_p2[0][0],
          filename);
    }
  %}
```

Unicode (UTF-8) Ln 107, Col 51

PSD_monitor.comp

```
* %E
*******************************************************************************/
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);

  int i, j;
  for (i=0; i<nx; i++){
    for (j=0; j<ny; j++){
      PSD_N[i][j] = 0;
      PSD_p[i][j] = 0;
      PSD_p2[i][j] = 0;
    }
  }
%}

TRACE
%{
  PROP_Z0;
  if (x>xmin && x<xmax && y>ymin && y<ymax){
    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);
  }
%}

SAVE
%{
  if (!nowritefile) {
    DETECTOR_OUT_2D(
        "PSD monitor",
        "X position [cm]",
        "Y position [cm]",
        xmin*100.0, xmax*100.0, ymin*100.0, ymax*100.0,
        nx, ny,
        &PSD_N[0][0],&PSD_p[0][0],&PSD_p2[0][0],
```

Unicode (UTF-8) Ln 50, Col 39

Same seed and same # mpi nodes -> same output

- Good for debugging
- If they don’t give the same CPU vs GPU, some comp(s) are not fully ported
- Use Event_monitor_simple to follow calculation pr. neutron

Declare section

```
/* User declarations from instrument definition. Can define functions. */
  double constant;
  double two_x_dummy;
```

Initialise section

```
  #define dummy (instrument->_parameters._dummy)
{
  constant=2;
  two_x_dummy=2*dummy;
}
  #undef dummy
  _arm_setpos(); /* type Arm */
  _source_setpos(); /* type Source_simple */
  _coll2_setpos(); /* type Slit */
  _detector_setpos(); /* type PSD_monitor */

  /* call iteratively all components INITIALISE */

  class_Source_simple_init(&_source_var);

  class_Slit_init(&_coll2_var);

  class_PSD_monitor_init(&_detector_var);
```

Functions per component with related component structs

Instrument and component structs built on CPU and transferred to GPU using OpenACC pragmas at the end of

INITIALISE

```
#ifdef USE_PGI
#  include <openacc.h>
acc_attach( (void*)&_arm_var );
acc_attach( (void*)&_source_var );
acc_attach( (void*)&_coll2_var );
acc_attach( (void*)&_detector_var );
#pragma acc update device(_arm_var)
#pragma acc update device(_source_var)
#pragma acc update device(_coll2_var)
#pragma acc update device(_detector_var)
acc_attach( (void*)&_instrument_var );
#pragma acc update device(_instrument_var)
#endif
```

Similar "host" update in FINALLY

“Full” list of pragmas and accel-code used

```
#include <accelmath.h>
#pragma acc declare create ( mcgravitation )
#pragma acc declare create ( mcseed )
#pragma acc declare create ( mcgravitation )
#pragma acc declare create ( mcMagnet )
#pragma acc declare create ( mcallowbackprop )
#pragma acc declare create ( mcncount )
#pragma acc routine seq
#pragma acc routine sequential
#pragma acc declare create ( _instrument_var )
#pragma acc declare create ( instrument )
#pragma acc declare create ( _arm_var )
#pragma acc declare create ( _source_var )
#pragma acc declare create ( _coll2_var )
#pragma acc declare create ( _detector_var )
#  include <openacc.h>
acc_attach( (void*)&_arm_var );
acc_attach( (void*)&_source_var );
acc_attach( (void*)&_coll2_var );
acc_attach( (void*)&_detector_var );
#pragma acc update device(_arm_var)
#pragma acc update device(_source_var)
#pragma acc update device(_coll2_var)
#pragma acc update device(_detector_var)
acc_attach( (void*)&_instrument_var );
#pragma acc update device(_instrument_var)
#pragma acc routine seq
#pragma acc atomic
#pragma acc parallel loop
#pragma acc declare device_resident(particles)
_class_particle* particles = acc_malloc(innerloop*sizeof(_class_particle));
#pragma acc enter data create(particles[0:innerloop])
#pragma acc parallel loop present(particles)
acc_free(particles);
#pragma acc update host(_arm_var)
#pragma acc update host(_source_var)
#pragma acc update host(_coll2_var)
#pragma acc update host(_detector_var)
#pragma acc update host(_instrument_var)
```

“math.h on GPU”

Needed basic variables / flags

GPUify all functions to be executed on GPU, i.e. in TRACE

Global instrument struct and component structs, including members like detector arrays etc.

OpenACC pure c-code, e.g. for the attaches (pointer-setup)

Ensure GLOBAL structs updated GPU-side end of INITIALISE

GPUify all functions to be executed on GPU, i.e. in TRACE

anything written to by multiple threads (detectors) should be “atomic”

Loop V1

Loop V2

Ensure GLOBAL structs updated host-side start of FINALLY