Peter Willendrup DTU Physics & ESS DMSC

Porting Legacy Monte Carlo Ray-Tracing to GPU Using ISO-C Code-Generation and OpenACC #pragmas

Agenda

- Neutrons, X-rays and scattering techniques
- What are the McStas and McXtrace codes used for?

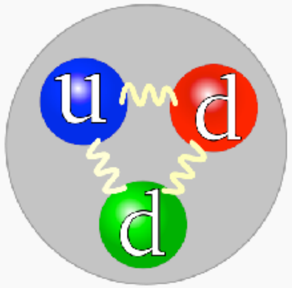
→ Keywords: Monte Carlo ray-traces for particle transport

- Technical foundations of the codes and why we chose OpenACC
- Our timeline toward the GPU
- Conclusions

The neutron and its basic properties

Subatomic particle discovered by Sir James Chadwick in 1932

Neutron



The quark structure of the neutron. (The color assignment of individual quarks is not important; what is important is that all three colors are present.)

Classification	Baryon
Composition	1 up quark, 2 down quarks
Statistics	Fermionic
Interactions	Gravity, Weak, Strong, Electromagnetic
Symbol	n, n^0, N^0
Antiparticle	Antineutron
Theorized	Ernest Rutherford[1][2] (1920)
Discovered	James Chadwick[1] (1932)
Mass	$1.674\,927\,351(74)\times10^{-27}$ kg[3] 939.565 378(21) MeV/c^2[3] 1.008 664 916 00(43) u[3]
Mean lifetime	881.5(15) s (free)
Electric charge	0 e 0 C
Electric dipole moment	$<2.9\times10^{-26}$ e⋅cm
Electric polarizability	$1.16(15)\times10^{-3}$ fm^3

Life time:	$\tau_{1/2} = 890s$
Mass:	$m = 1.675 \times 10^{-27} kg$
Charge:	$Q = 0$
Spin:	$s = \hbar/2$
Magnetic moment:	$\mu/\mu_n = -1.913$

$$E = \frac{1}{2}mv^2 = \frac{\hbar^2 k^2}{2m} \qquad \lambda = 2\pi/k$$

$$E = 81.81 \cdot \lambda^{-2} = 2.07 \cdot k^2 = 5.23 \cdot v^2$$

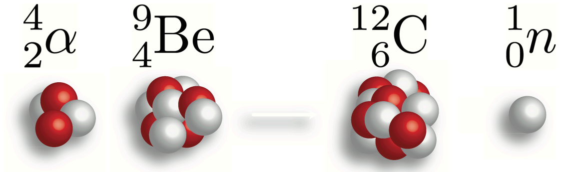
	Energy	Wavelength	n-Wavevector	Velocity	Frequency
cold neutrons:	E = 1 meV	l = 9.0446 Å	k = 0.6947 1/Å	v = 437 m/s	v = 0.2418 THz
	E = 5 meV	l = 4.0449 Å	k = 1.5534 1/Å	v = 978 m/s	v = 1.2090 THz
thermal	E = 25 meV	l = 1.8089 Å	k = 3.4734 1/Å	v = 2187 m/s	v = 6.045 THz
neutrons:	E = 50 meV	l = 1.2791 Å	k = 4.9122 1/Å	v = 3093 m/s	v = 12.090 THz

McXtrace

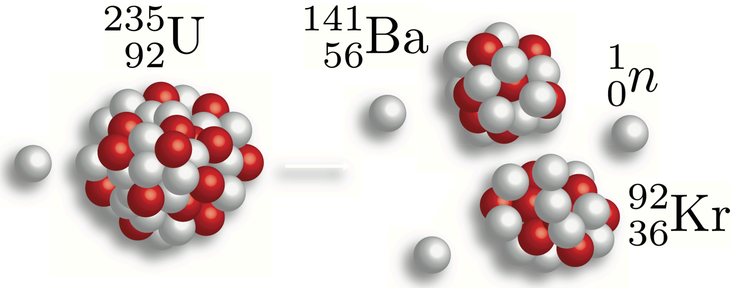
McStas

Wavelengths and energies compatible with structure and dynamics in condensed matter

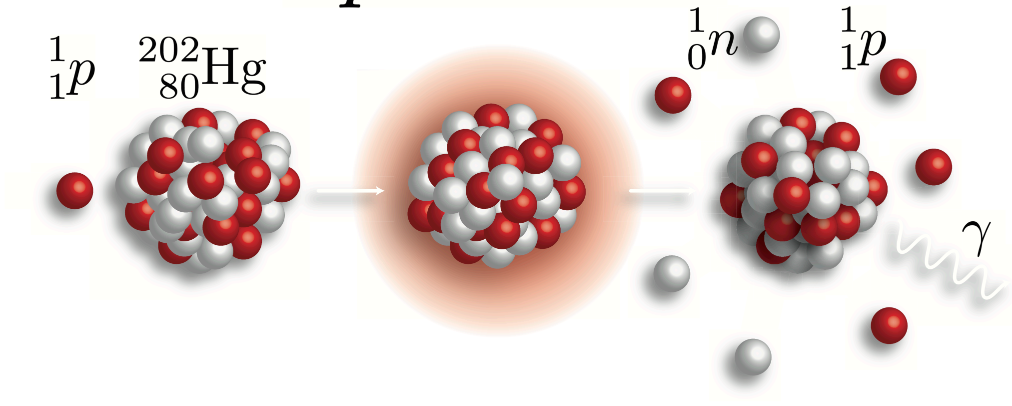
Chadwick

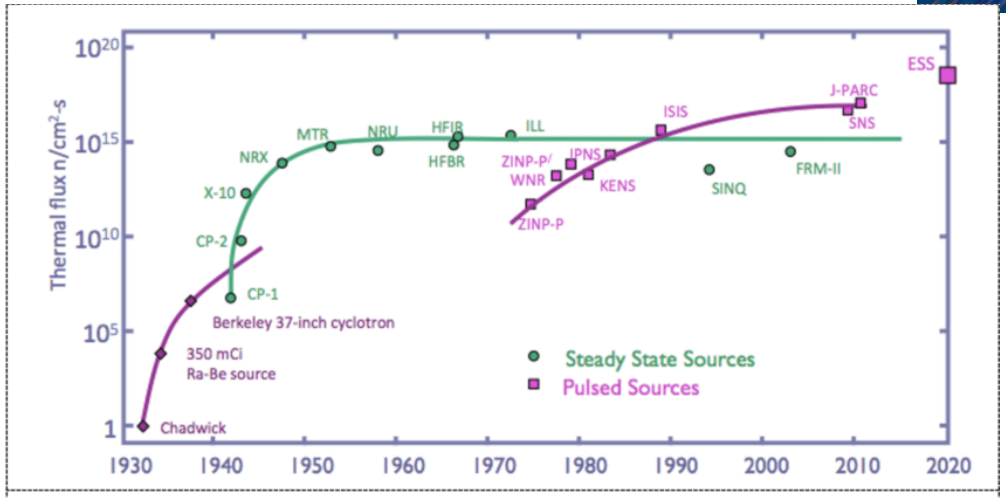


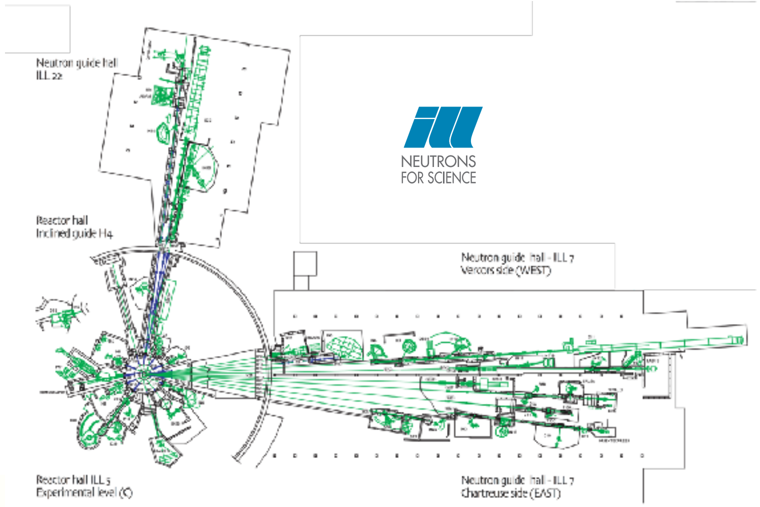
Fission

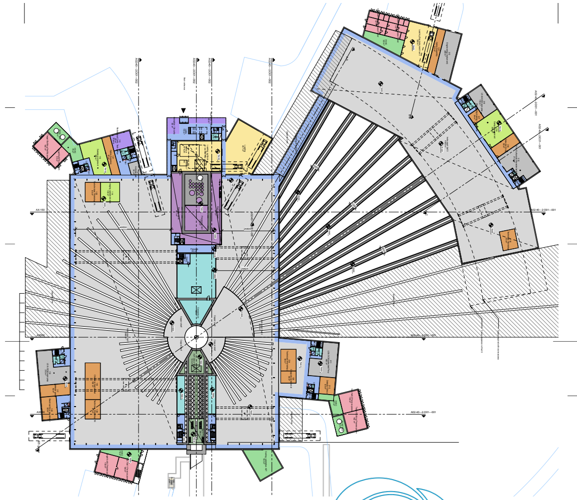


Spallation









McXtrace

McStas

1994 Nobel Prize in Physics

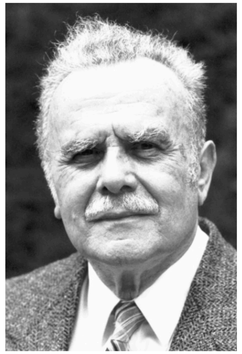


Photo from the Nobel Foundation archive.
Bertram N. Brockhouse
Prize share: 1/2

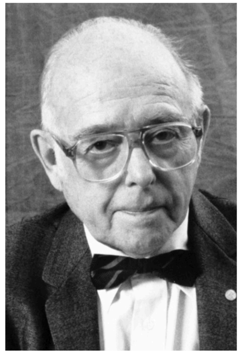
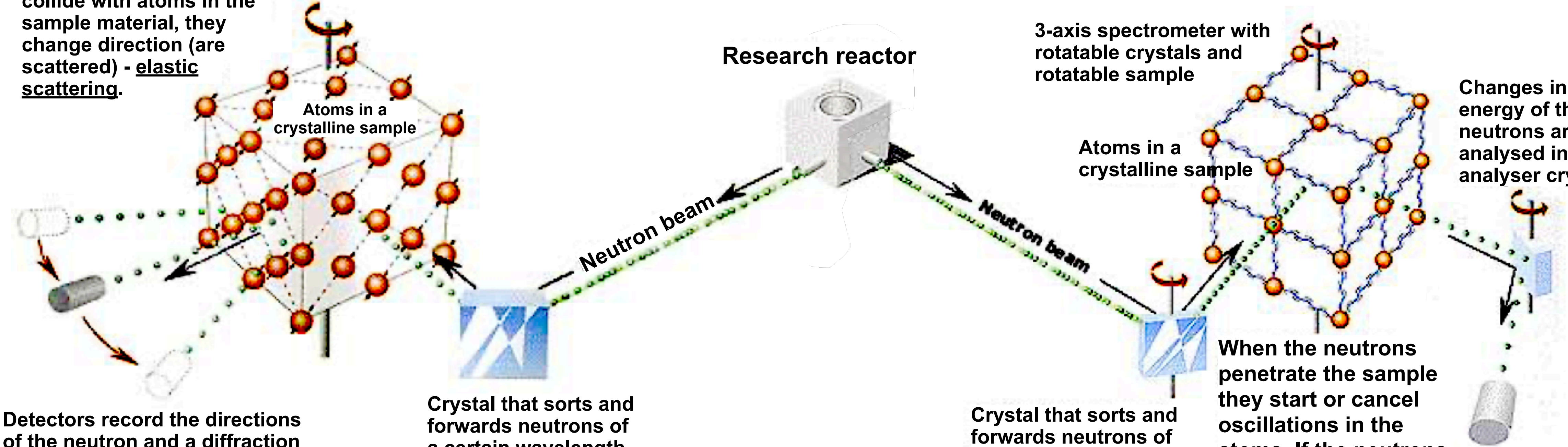


Photo from the Nobel Foundation archive.
Clifford G. Shull
Prize share: 1/2

'… In simple terms, Clifford G. Shull has helped answer the question of where atoms "are" and Bertram N. Brockhouse the question of what atoms "do".

Diffraction

Spectroscopy



When the neutrons collide with atoms in the sample material, they change direction (are scattered) - elastic scattering.

Detectors record the directions of the neutron and a diffraction pattern is obtained.
The pattern shows the positions of the atoms relative to one another.

Crystal that sorts and forwards neutrons of a certain wavelength (energy) - mono-chromatized neutrons

3-axis spectrometer with rotatable crystals and rotatable sample

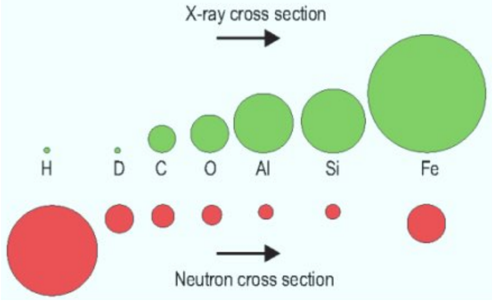
Changes in the energy of the neutrons are first analysed in an analyser crystal…

Crystal that sorts and forwards neutrons of a certain wavelength (energy) - mono-chromatized neutrons

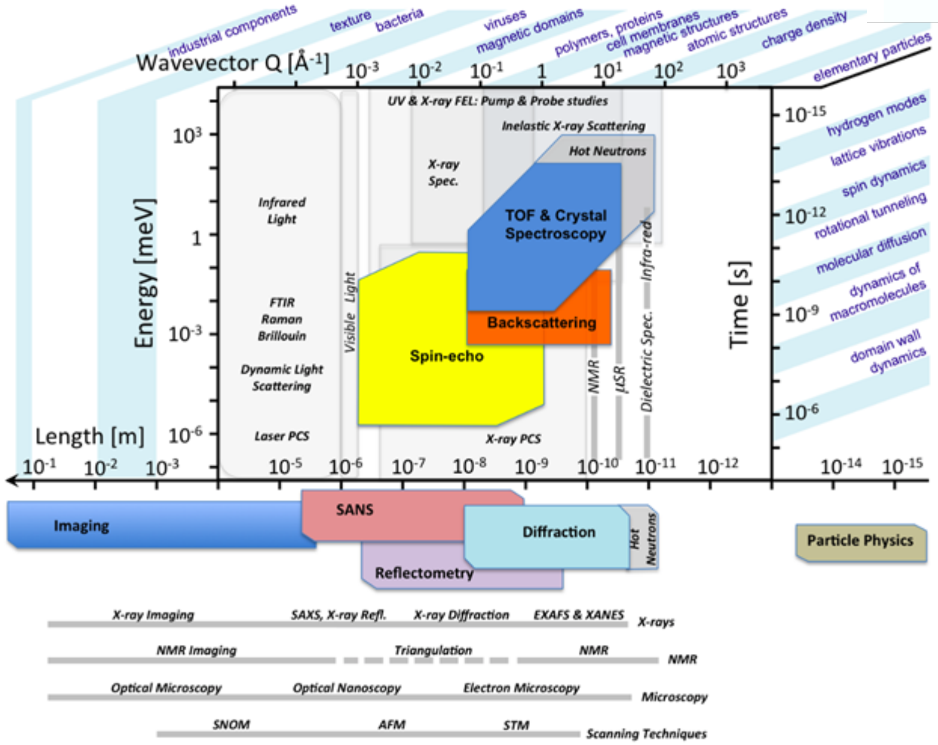
When the neutrons penetrate the sample they start or cancel oscillations in the atoms. If the neutrons create phonons or magnon they themselves lose the energy these absorb - inelastic scattering.

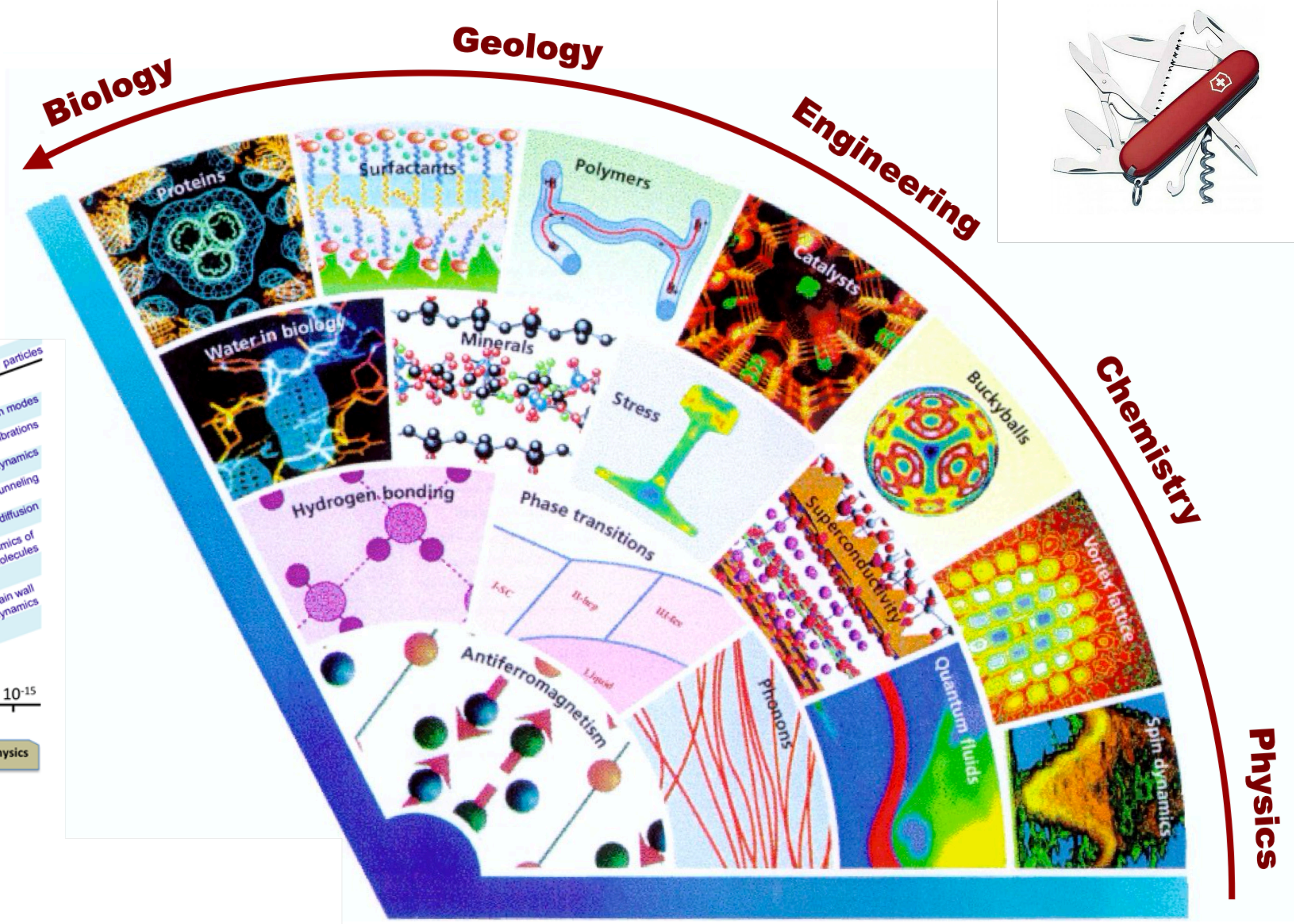
.. and the neutrons hen counted in a letector.

The neutron is a multidisciplinary probe for structure and dynamics of condensed matter systems - ‘Swiss army knife’



- complementary to X-rays



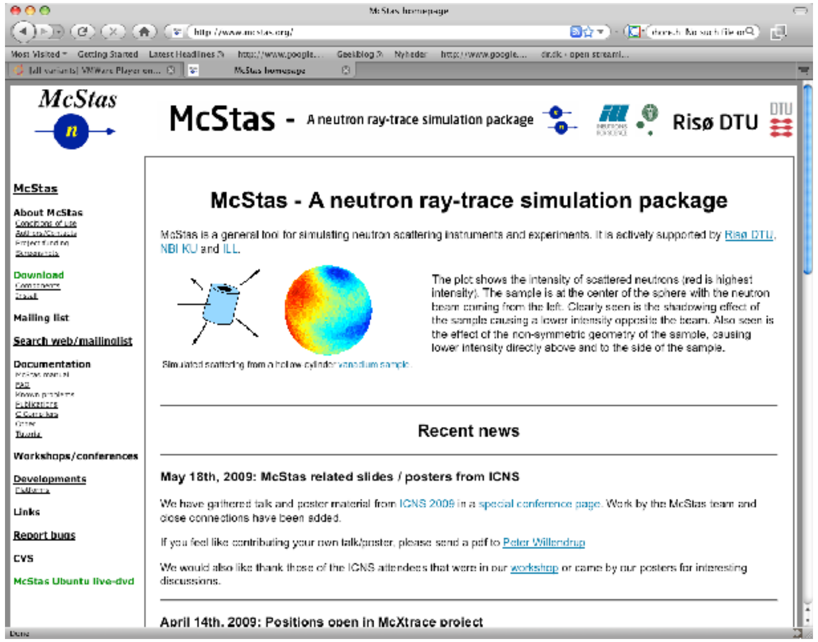


McStas Introduction

- 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 ~6 people on joint McStas-McXtrace team but only 2 full time, based at DTU

GNU GPL license

Open Source



McXtrace

McStas

Project website at

http://www.mcstas.org

mcstas-users@mcstas.org mailinglist

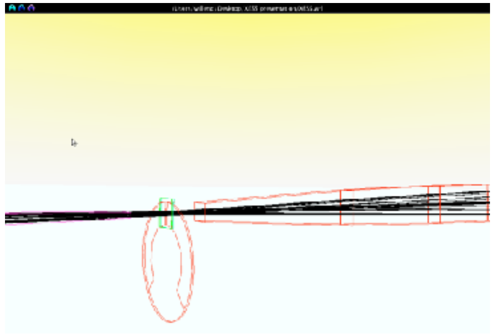
McXtrace - since jan 2009 similar for X-rays

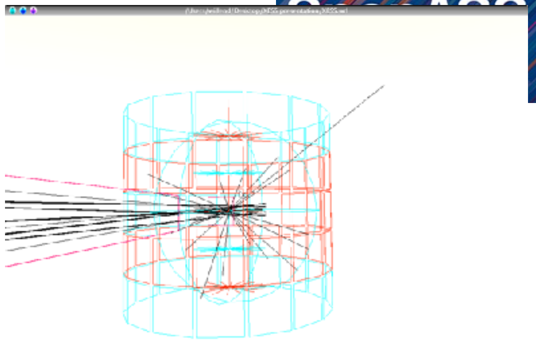
- Synergy, knowledge transfer, shared infrastructure and codebase on GitHub

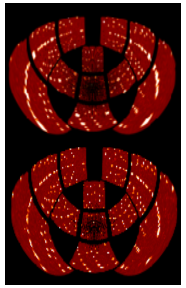
What is McStas used for?

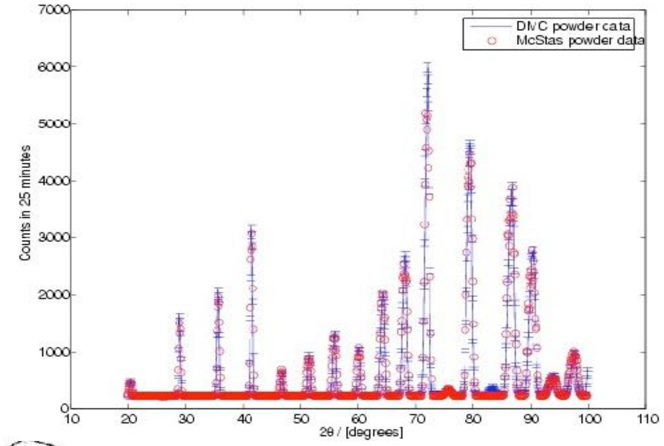
- Instrumentation
- Planning
- Construction
- Virtual experiments
- Data analysis
- Teaching

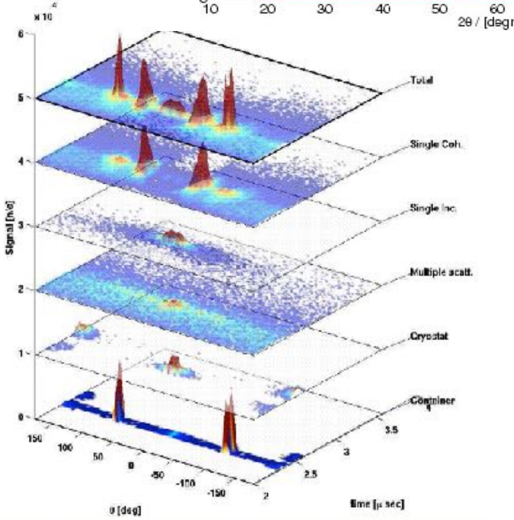
(KU, DTU)

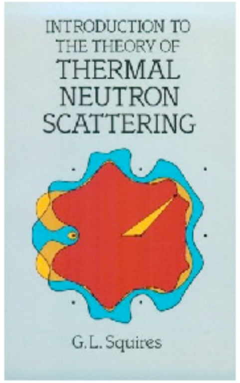


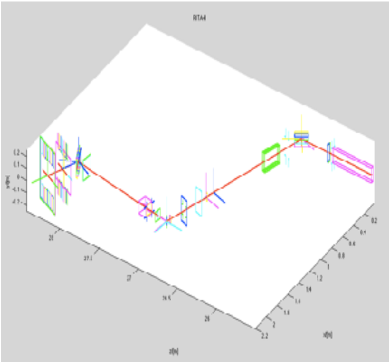








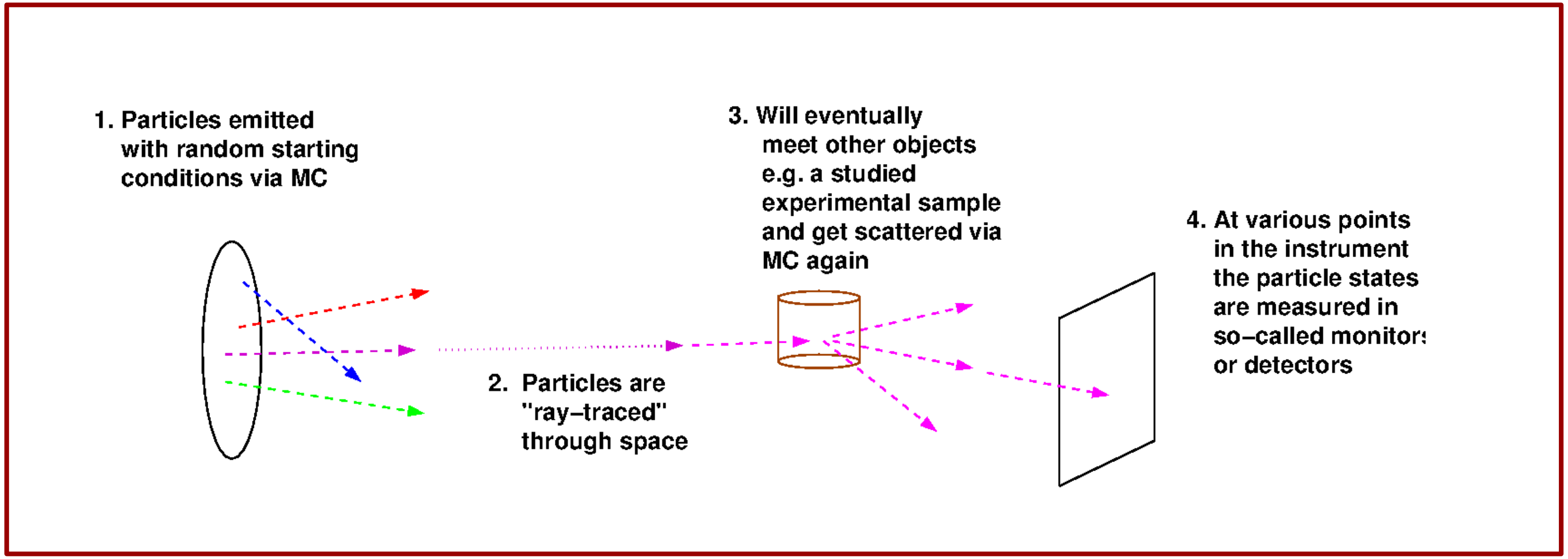




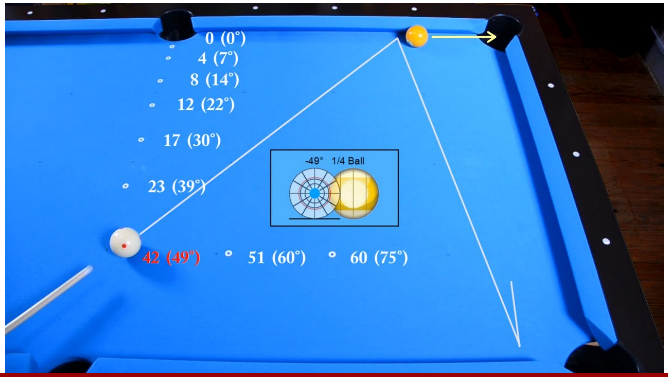
McXtrace

McStas

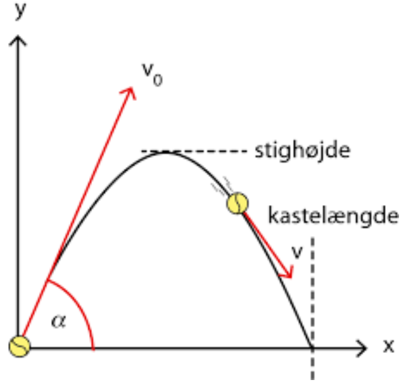
McStas and McXtrace are Monte Carlo ray-tracers



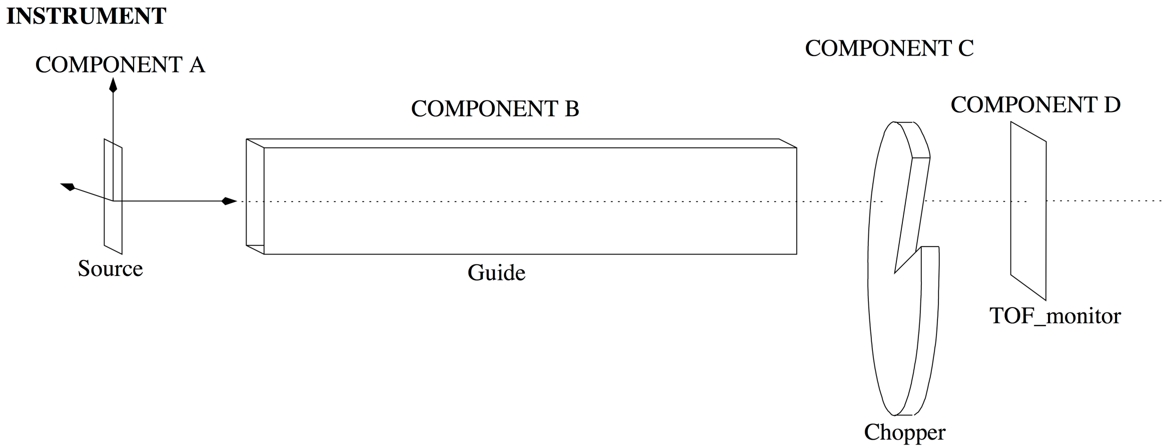
- For the neutrons, gravity kicks in… A cold neutron falls ~10cm over 150m!
- Classical Newtonian mechanics, i.e.
- (independent, particles though…)

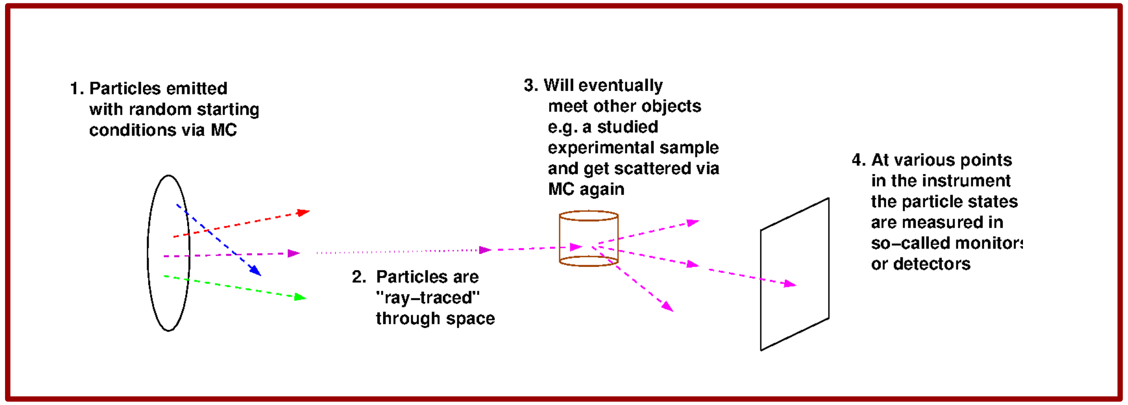


and



McStas and McXtrace Monte Carlo ray-tracers





The "tool layer" consists of programs manipulated by the McStas user:

mcgui, graphical user interface	mcplot, visualize histogram outp.	mcdisplay, visualize instrument

mcgui is used to assemble an instrument file, which is taken over by the McStas system

```
DEFINE INSTRUMENT Example(Param1=1, string Param2="two", ...)

COMPONENT A = Source(Parameters...)
AT (0, 0, 0) ABSOLUTE

COMPONENT B = Guide(Parameters...)
AT (0, 0, 1) RELATIVE A

COMPONENT C = DiskChopper(Parameters...)
AT (0, 0, 1) RELATIVE B

COMPONENT D = TOF_monitor(Parameters, filename="Tof.dat")
AT (0, 0, Param1) RELATIVE PREVIOUS
```

DSL

"Instrument file"

- **Source.comp – c–code**
- **Guide.comp – c–code**
- **DiskChopper.comp – c–code**
- **TOF_monitor.comp – c–code**

Component library

Random numbers	I/O	Physical consts.

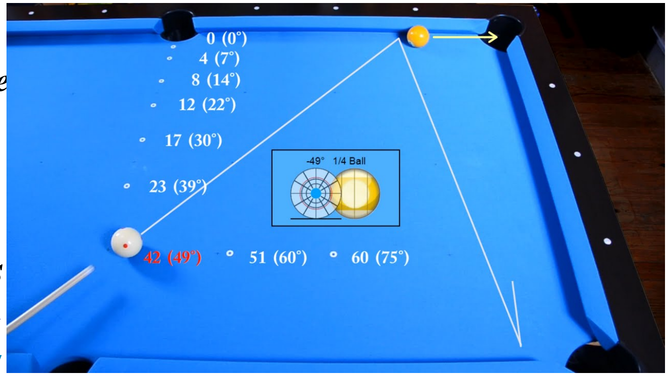
"Kernel and runtime c–code"

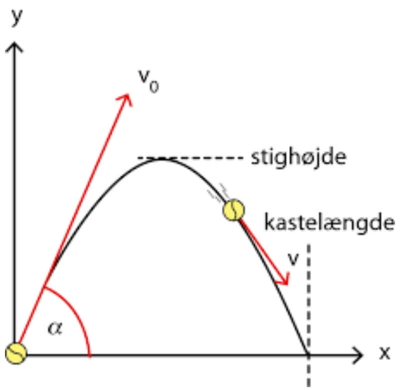
The McStas system generates an "ISO C file" and an executable from instrument file and c–codes

The simulation executable produces data output which can be visualized using the mcplot and mcdisplay tools

McXtrace

McStas

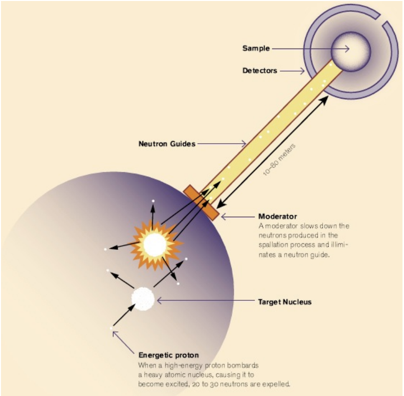




McStas tech overview

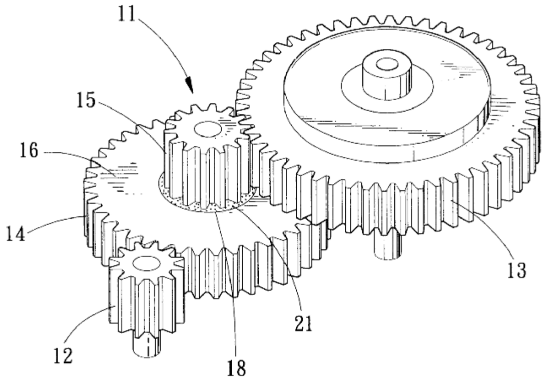


- Portable code (Unix/Linux/Mac/Windoze)
  - On the CPU-side, ran on everything from iPhone to 1000+ node cluster, intel, Alpha, PA-RISC etc.



- ‘Component' files (~100) inserted from library
  - Sources
  - Optics
  - Samples
  - Monitors
  - If needed, write your own comps - **many are USER developments ~200-line “physicist” codes**

- DSL + ISO-C code-gen. (compiler technology / LeX+Yacc)
  - Simple Instrument language → (Code generation) → ISO C



- Component codes realizing beamline parts (including user contribs)

- Library of common functions
  - I/O
  - Random numbers
  - Physical constants
  - Propagation
  - Precession in fields
  - ...

User experience:
- Write instrument
- Launch simulation (generates binary and runs simulation)
- Look at output data

https://www.openhub.net/p/mccode

Stats and information on the codebase

Languages

Total Lines :	266,097	Code Lines :	204,053	Percent Code Lines :	76.7%
Number of Languages :	19	Total Comment Lines :	36,001	Percent Comment Lines :	13.5%
		Total Blank Lines :	26,043	Percent Blank Lines :	9.8%

Language Breakdown

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
AMPL	62,577	-642	-1.0%	1,718	63,653	23.9%
C	36,382	9,549	20.8%	6,415	52,346	19.7%
Fortran (Free-format)	36,215	14,692	28.9%	5,075	55,982	21.0%
TeX/LaTeX	17,048	1,920	10.1%	2,790	21,758	8.2%
Python	15,354	4,842	24.0%	4,195	24,391	9.2%
Perl	13,833	1,642	10.6%	1,198	16,673	6.3%
shell script	5,583	1,240	18.2%	1,372	8,195	3.1%
CMake	4,341	1,138	20.8%	1,133	6,612	2.5%
JavaScript	3,945	374	8.7%	1,001	5,320	2.0%
HTML	2,732	36	1.3%	90	2,858	1.1%
XML	2,638	10	0.4%	277	2,925	1.1%
Matlab	1,837	735	28.6%	378	2,950	1.1%
C++	735	95	11.4%	183	1,013	0.4%
CSS	273	9	3.2%	48	330	0.1%
Ruby	177	282	61.4%	69	528	0.2%
Fortran (Fixed-format)	170	0	0.0%	61	231	0.1%
DOS batch script	140	53	27.5%	12	205	0.1%
R	72	26	26.5%	28	126	0.0%
IDL/PV-WAVE/GDL	1	0	0.0%	0	1	0.0%
Totals	204,053	36,001		26,043	266,097	

Project Summary : Factoids

Mature, well-established codebase

Large, active development team

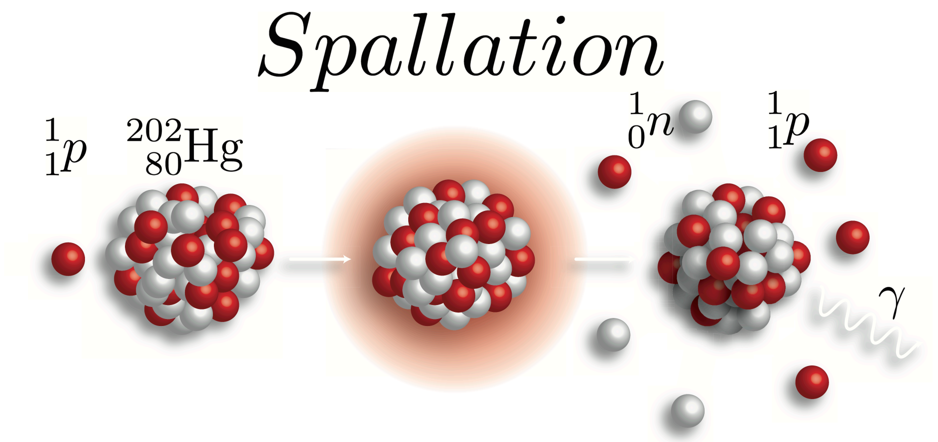
Stable Y-O-Y development activity

Very few source code comments

McCode is written mostly in AMPL.

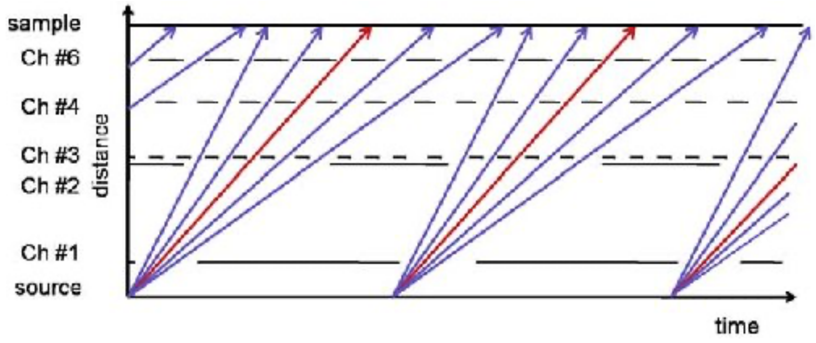
Nope, that's our DSL and grammar. :-) Which is close to "English".

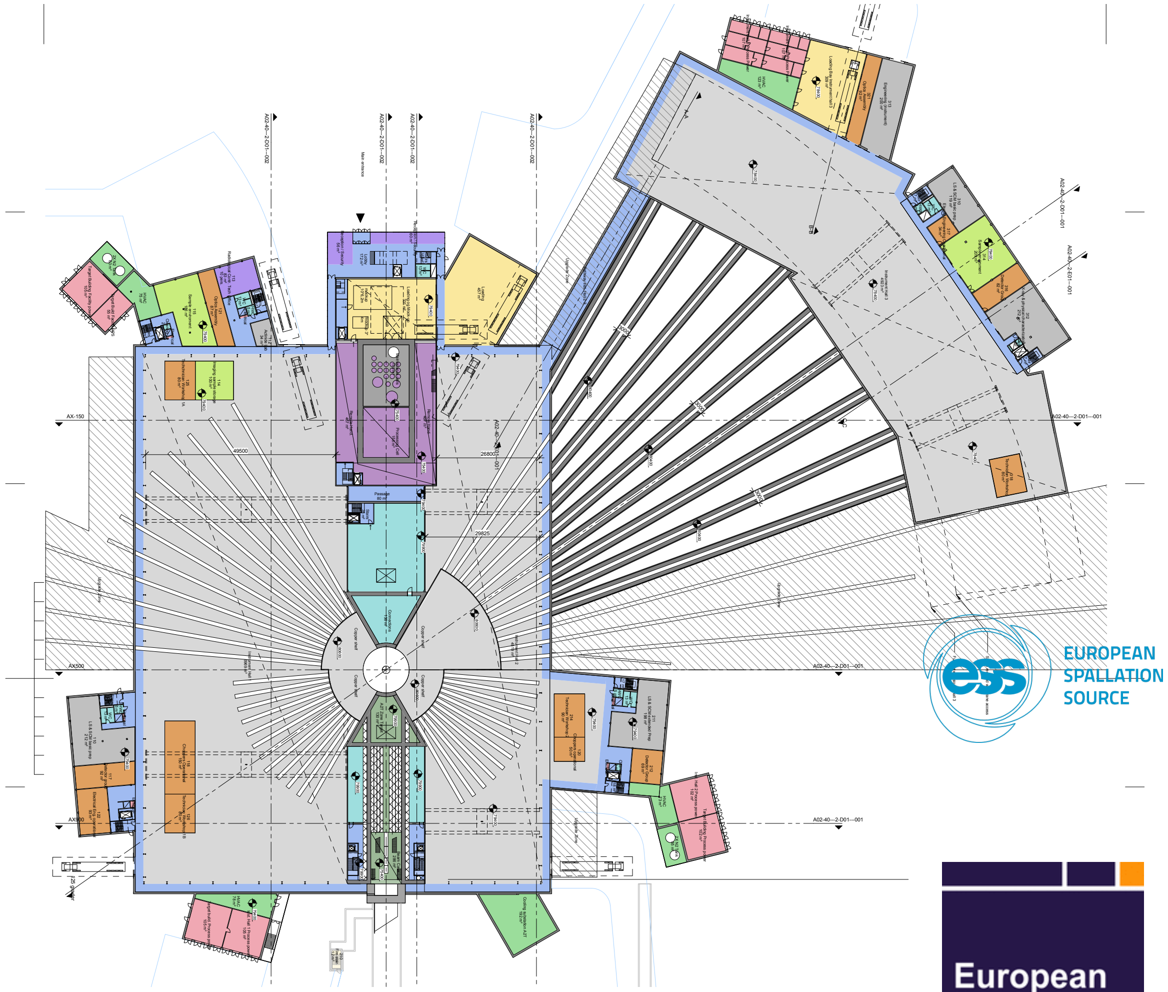
If it is so bloody good and “final”,why bother looking at GPU?



Next-generation, long-pulse spallation facilities are complex to construct and model since they use

- ‘rep-rate multiplication’
- event-mode experiments



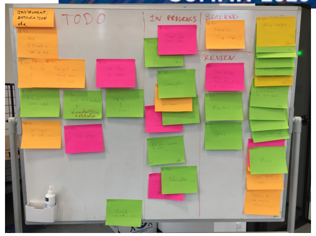


Even more so for e.g. X-ray Free-Electron Lasers…

At reactor and short-pulse, the **McStas run-time** keeps up with experimental time, **not the the case at ESS**…

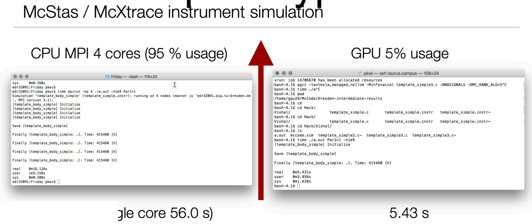
A ~2 order of magnitude would be great - and we believe we found it!

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.

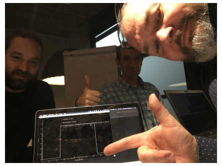


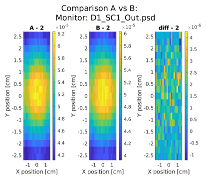
NVIDIA mentor: Vishal Metha

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

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.





NVIDIA mentor: Christian Hundt

hackathon org.:
Sebastian Von Alfthan (CSC)

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

McStas heading for the GPU… 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

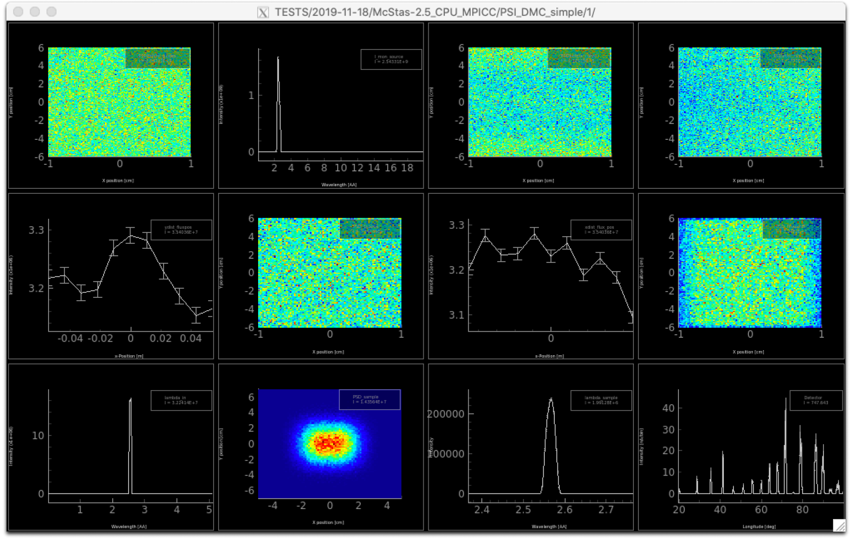
(1-core run, **1e9** would be 2000 secs)

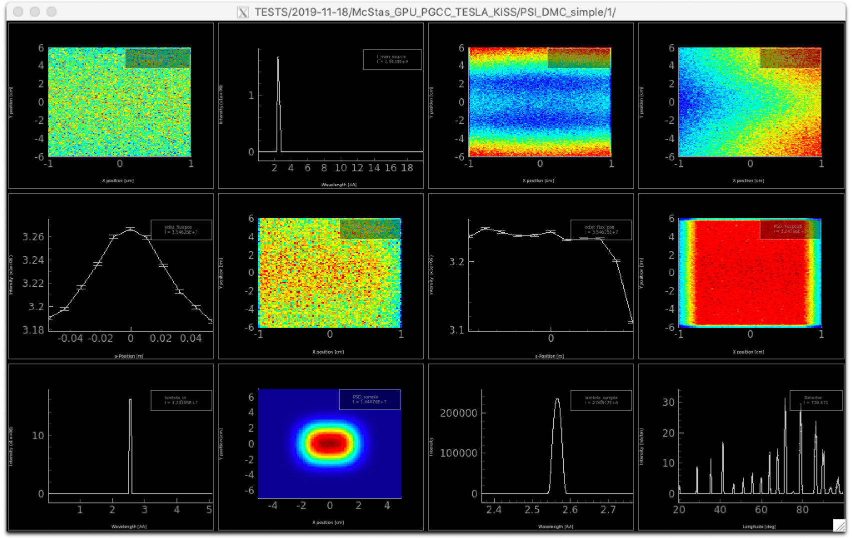
VS.

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!

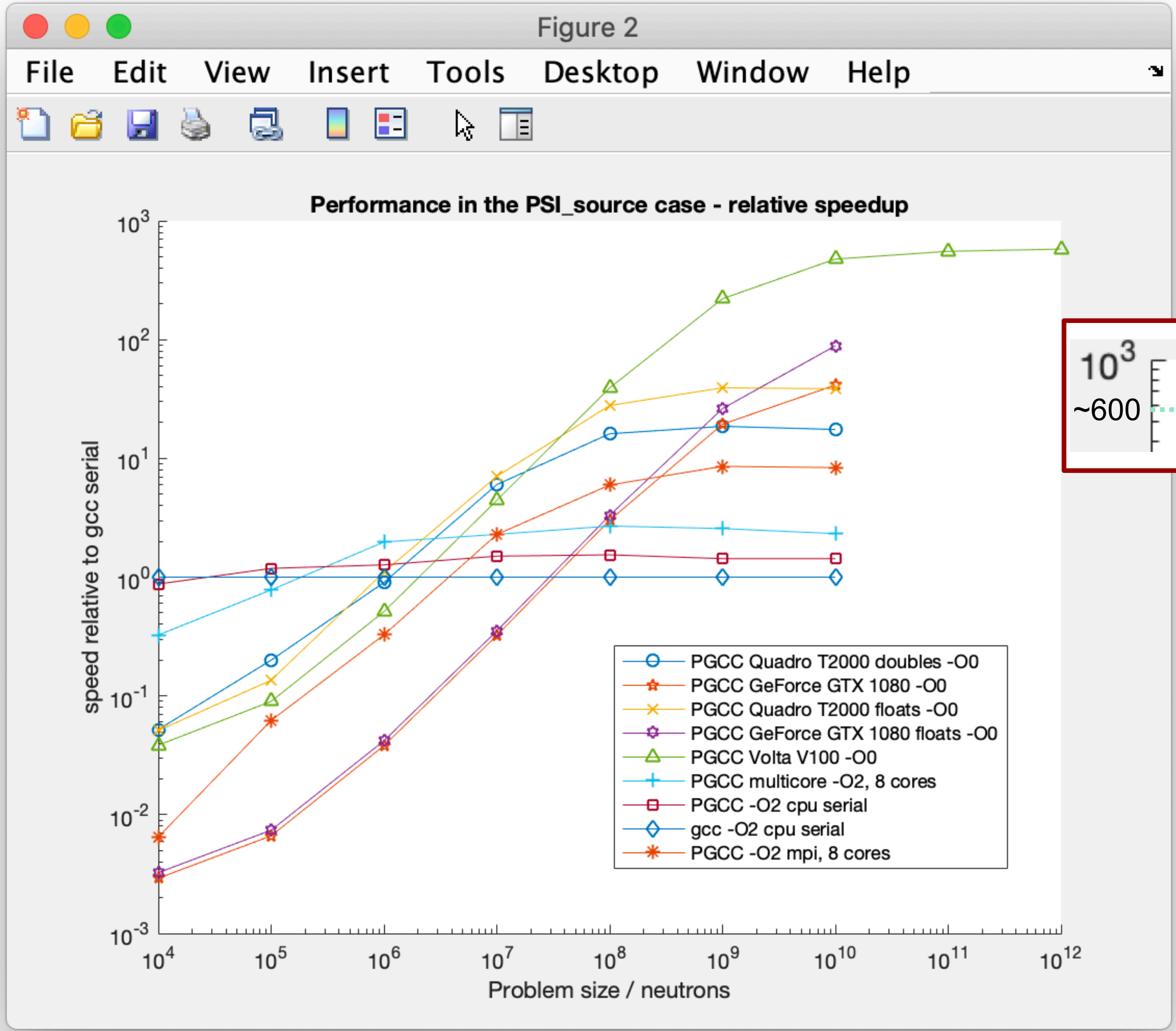




McStas heading for the GPU… November 2019 - first good look at performance.

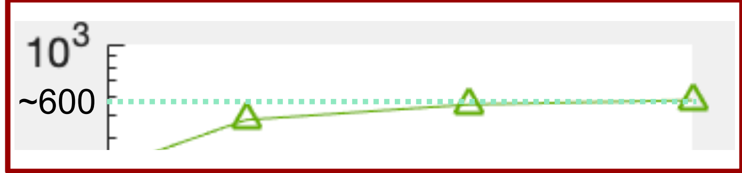
Idealised instrument with source and monitor only - i.e. without any use of the ABSORB macro.

(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 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

The neutron and “state-flags” 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;
```

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

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

Each component will correspond to a GPU'ified function…

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

Init and finalisation codes run purely CPU.

```
#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

"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

“Full” list of pragmas and accel-code used

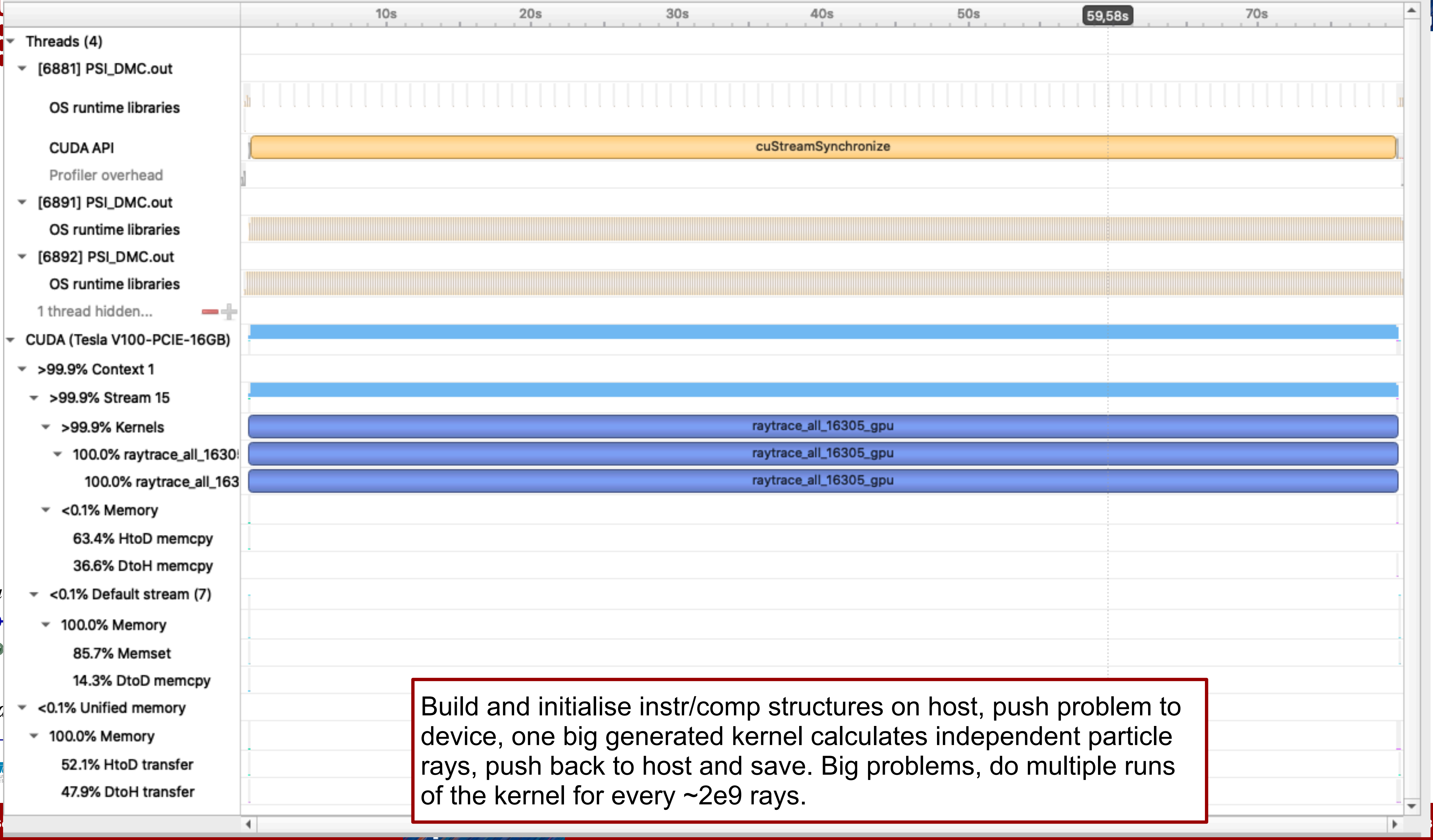
```c
#include <accelmath.h>
#pragma acc declare create ( mcgravitation )
#pragma acc declare create ( mcseed )
#pragma acc declare create ( mcgravitation )
#pragma acc declare create ( mcMagnet )
#pragma acc declare create ( mcallowbackprop )
#pragma acc declare create ( mcncount )
#pragma acc routine 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

McXtrace

McStas

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.

Conclusion / remaining key questions

1. Why are you using OpenACC? What is your favorite feature?
   - We could **keep** most of the **core of our legacy code base intact**, some structural changes were needed. ☺ (Attempt done with OpenCL 10 y ago, no real success)

2. Did you have to use 2 diff. code bases for CPU and GPU?
   - **Nope**, just one code base, a few generated **#pragmas** and **#define's** ☺

3. Tell us about a feature you are looking for, that OpenACC lacks, why do you need it?
   - A few niche cases use function **pointers/polymorphism** on CPU, which is not available in OpenACC. Current workaround with case-hierarchy is a bit **ugly** 😐
   - **deepcopy** of 2-dimensional arrays as struct members does not seem to work correctly in our case without **managed** mode. Does not "feel" transparent wrt. possible performance penalties. 😐 We **could** fully move to 1D arrays, but this would require too much work. 😐

4. Any additional bugs, features, feedback to share?
   - We did a Hackathon together with "competing" **Python+OpenCL code RAMP** which is written from scratch. Where we tested, **we were ~40% faster**. ;-) ☺
   - In our niche setting, **curand() turned out quite slow** ☹ 😐
   - **Better high-level OpenACC docs needed**. Choice of exact pragma often trial and error. 😐
   - **CUDA-mindset** / thinking seems required for **deep/full** understanding of **OpenACC**. Somewhat **contradictory** to the idea of "more science, less programming". 😐

McXtrace

McStas

Thank you for the attention - thanks to the team, Nvidia mentors and Hackathon hosts :-)

Vishal Metha

Christian Hundt

Alexey Romanenko

Guido Juckeland

Sebastian von Alfthan