Peter Willendrup DTU Physics & ESS DMSC

McStas + McXtrace: Monte Carlo Ray-Tracers + GPU Using ISO-C Code-Generation and OpenACC #pragmas

Agenda

- Background, science: → Keywords: Monte Carlo ray-traces for particle transport
  - Neutrons, X-rays and scattering techniques
  - What are the McStas and McXtrace codes used for?

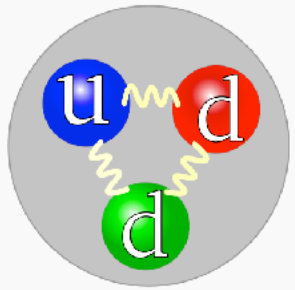
- Background, code:
  - Technical foundations of the codes and why we chose OpenACC
  - Our timeline toward the GPU
  - What did we change for OpenACC and what does not (yet) work?

- GPU status:
  - How well (fast) does it work?
  - Simulation flow
  - What to investigate / improve?

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^0, N^0
Antiparticle	Antineutron
Theorized	Ernest Rutherford[1][2] (1920)
Discovered	James Chadwick[1] (1932)
Mass	1.674 927 351(74) ×10⁻²⁷ kg[3] 939.565 378(21) MeV/c²[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 ×10⁻²⁶ e·cm
Electric polarizability	1.16(15) ×10⁻³ fm³

Life time: $\tau_{1/2} = 890s$

Mass: $m = 1.675 \times 10^{-27} kg$

Charge: $Q = 0$

Spin: $s = \hbar/2$

Magnetic moment: $\mu/\mu_n = -1.913$

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

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

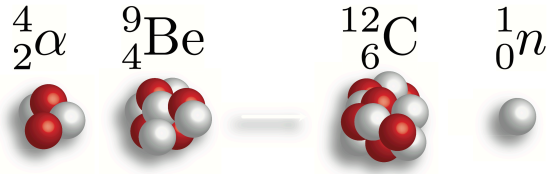
	Energy	Wavelength	n-Wavevector	Velocity	Frequency
cold neutrons:	E = 1 meV	l = 9.0446 Å	k = 0.6947 1/Å	v = 437 m/s	v = 0.2418 THz
	E = 5 meV	l = 4.0449 Å	k = 1.5534 1/Å	v = 978 m/s	v = 1.2090 THz
thermal neutrons:	E = 25 meV	l = 1.8089 Å	k = 3.4734 1/Å	v = 2187 m/s	v = 6.045 THz
	E = 50 meV	l = 1.2791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

McXtrace

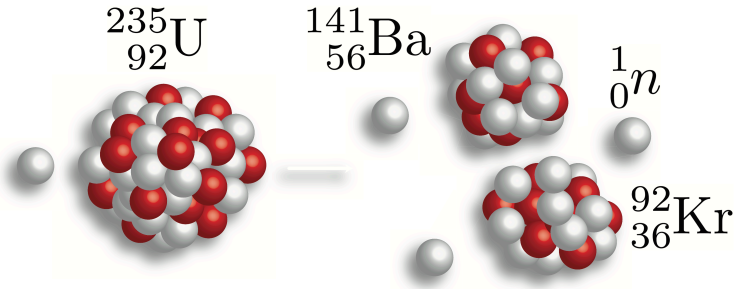
McStas

DTU ess EUROPEAN SPALLATION SOURCE

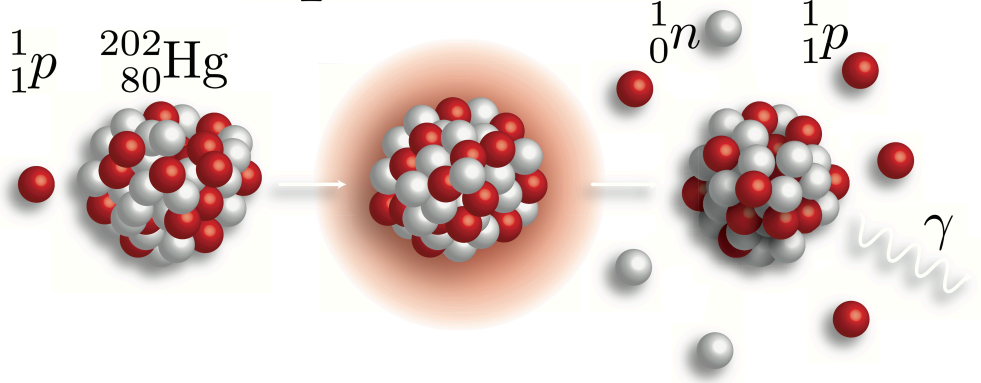
Chadwick

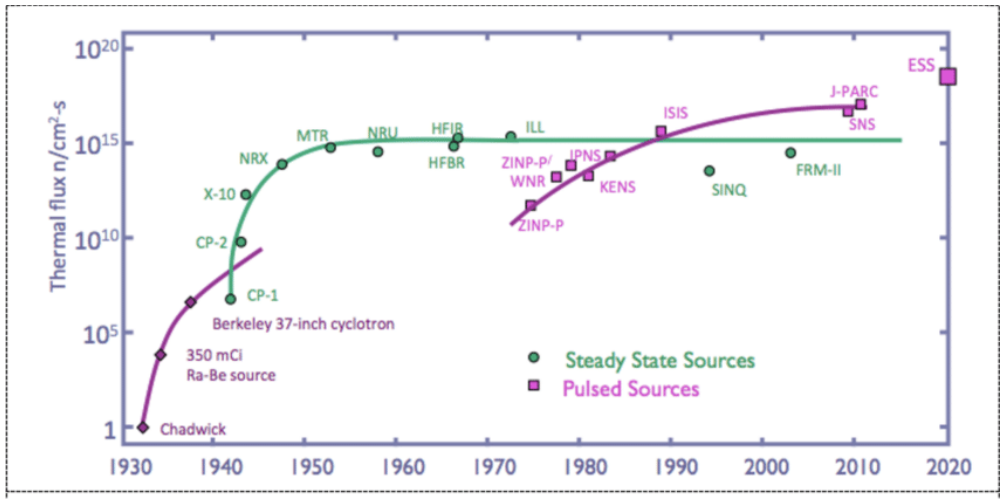


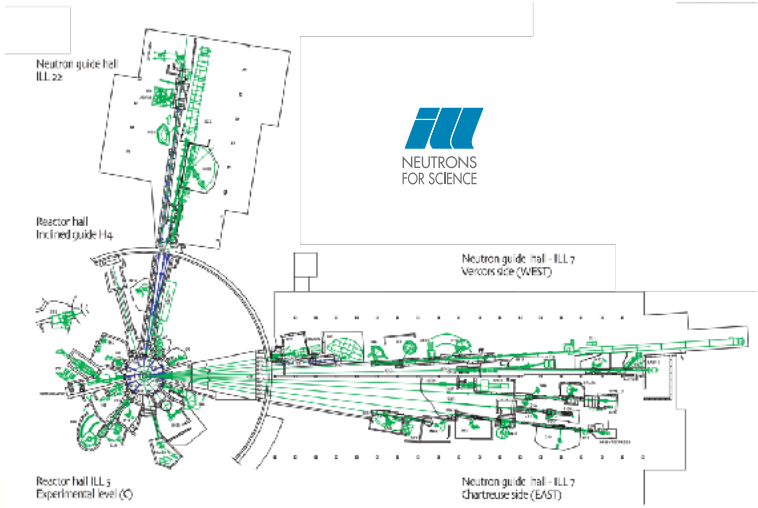
Fission

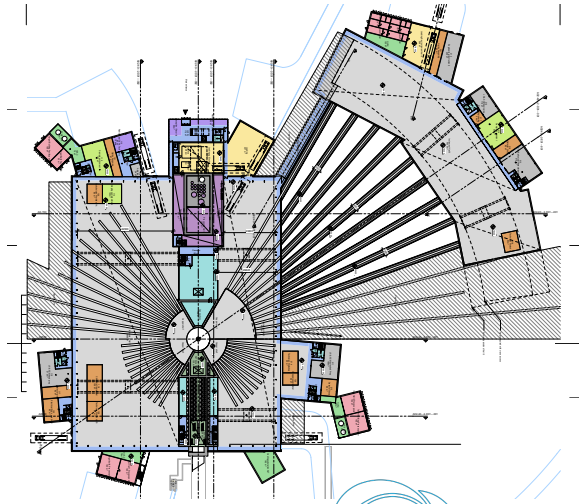


Spallation









McXtrace

DTU ESRF

McStas

DTU ILL NEUTRONS FOR SCIENCE ess EUROPEAN SPALLATION SOURCE

ess EUROPEAN SPALLATION SOURCE

1994 Nobel Prize in Physics

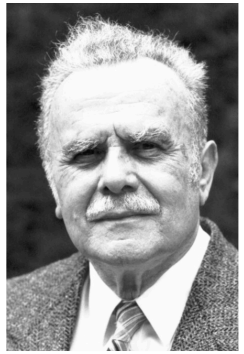


Photo from the Nobel Foundation archive.
Bertram N. Brockhouse
Prize share: 1/2

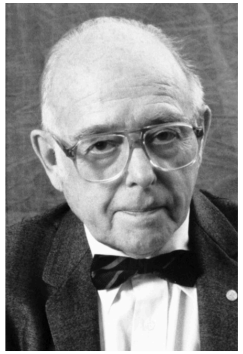
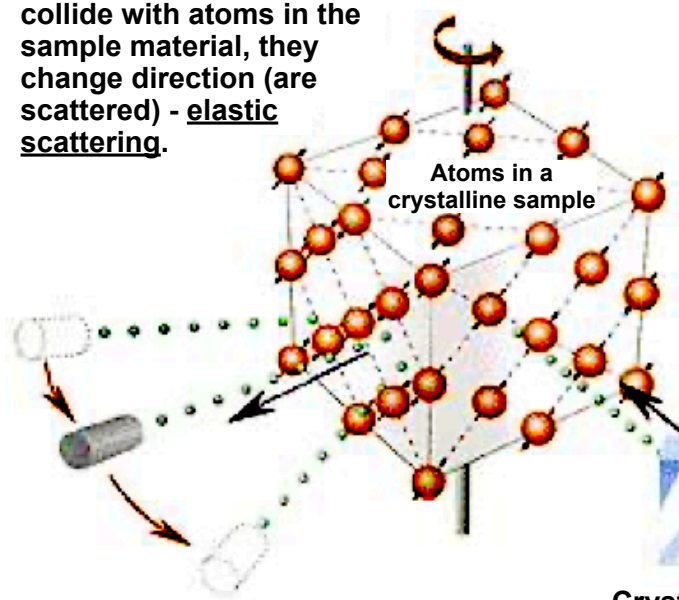


Photo from the Nobel Foundation archive.
Clifford G. Shull
Prize share: 1/2

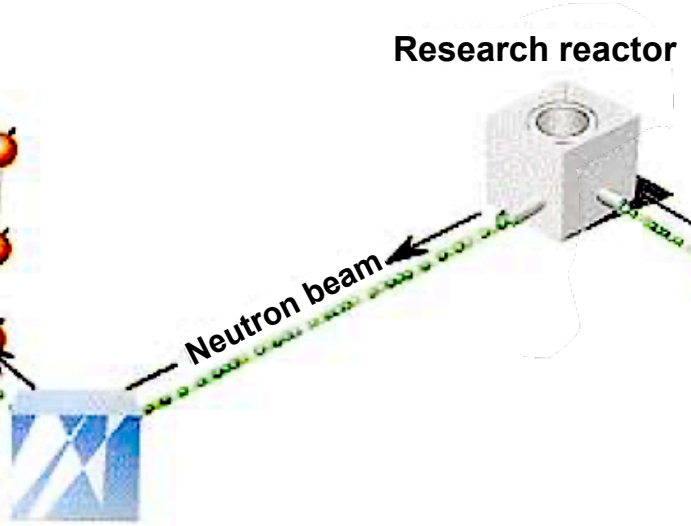
'… In simple terms, Clifford G. Shull has helped answer the question of where atoms "are" and Bertram N. Brockhouse the question of what atoms "do".

Diffraction

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



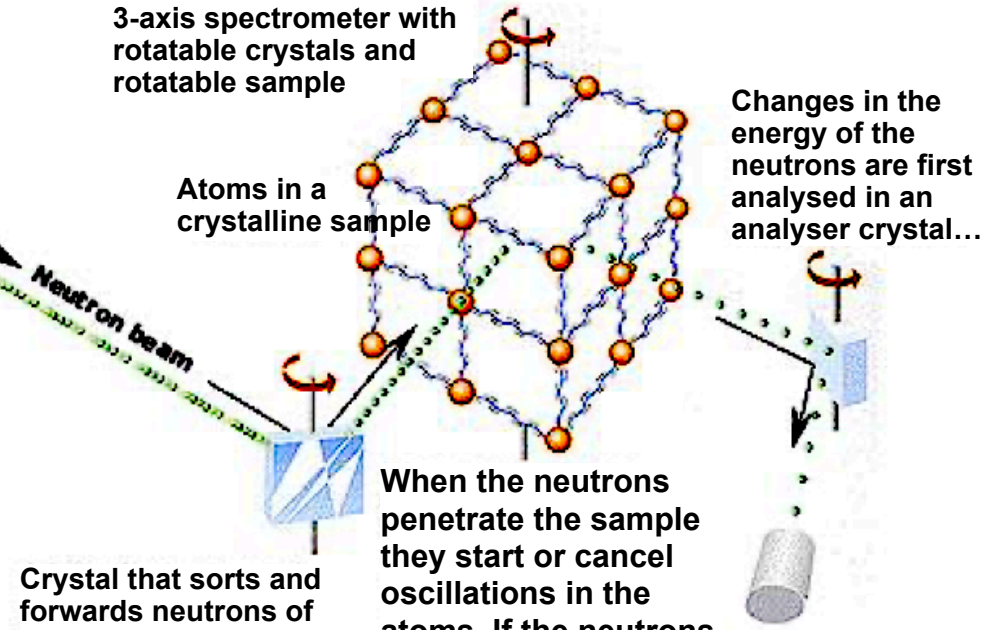
Detectors record the directions of the neutron and a diffraction pattern is obtained.
The pattern shows the positions of the atoms relative to one another.



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

Spectroscopy

3-axis spectrometer with rotatable crystals and rotatable sample



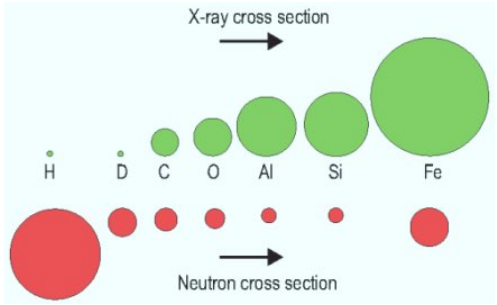
Crystal that sorts and forwards neutrons of a certain wavelength (energy) - mono-chromatized neutrons

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 - <u>inelastic scattering</u>.

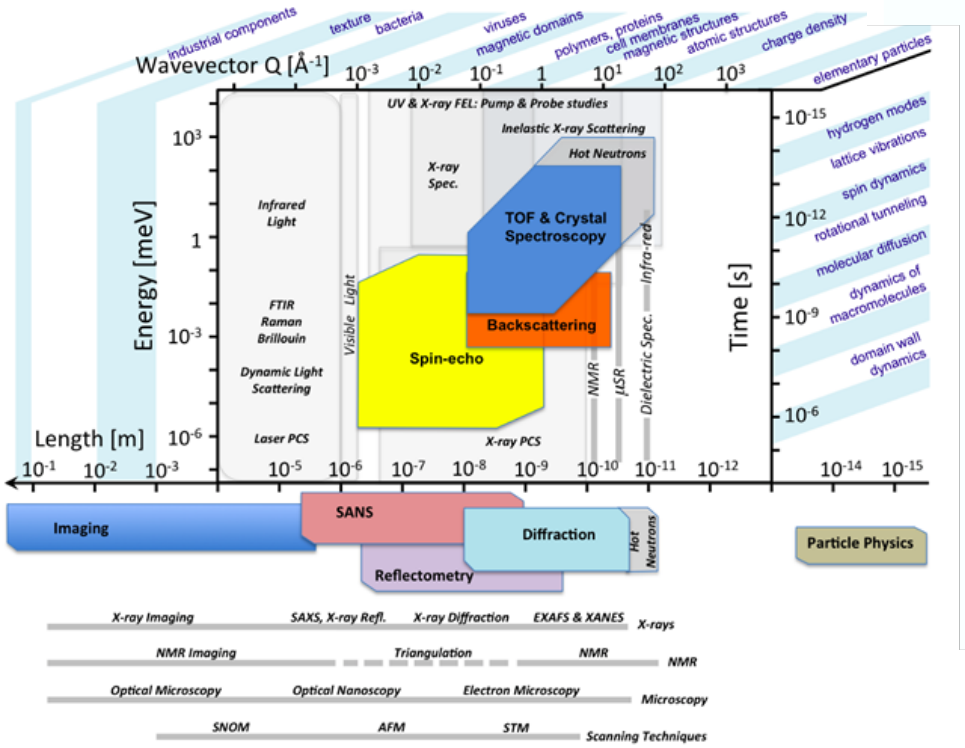
Changes in the energy of the neutrons are first analysed in an analyser crystal…

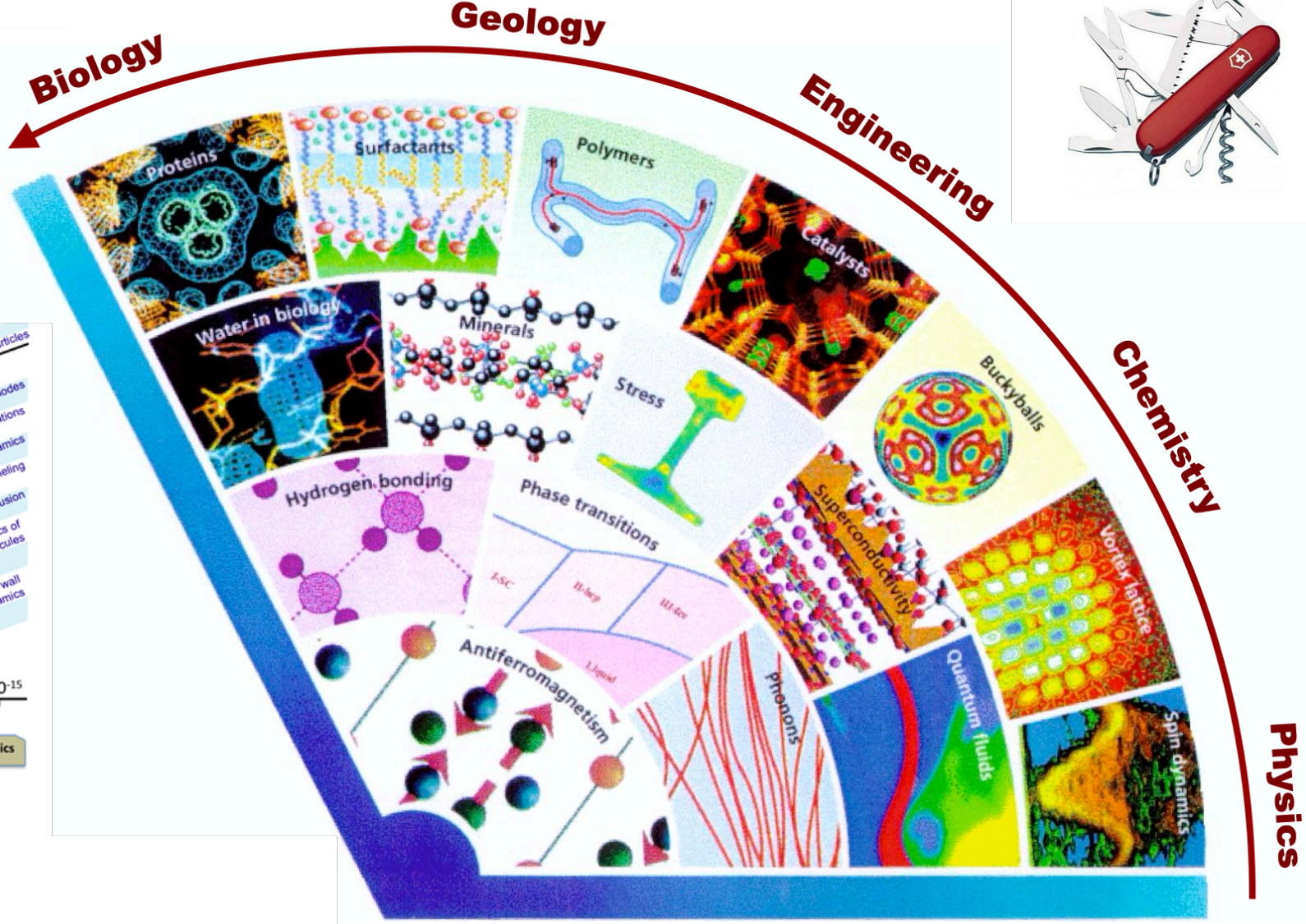
.. and the neutrons hen counted in a letector.

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

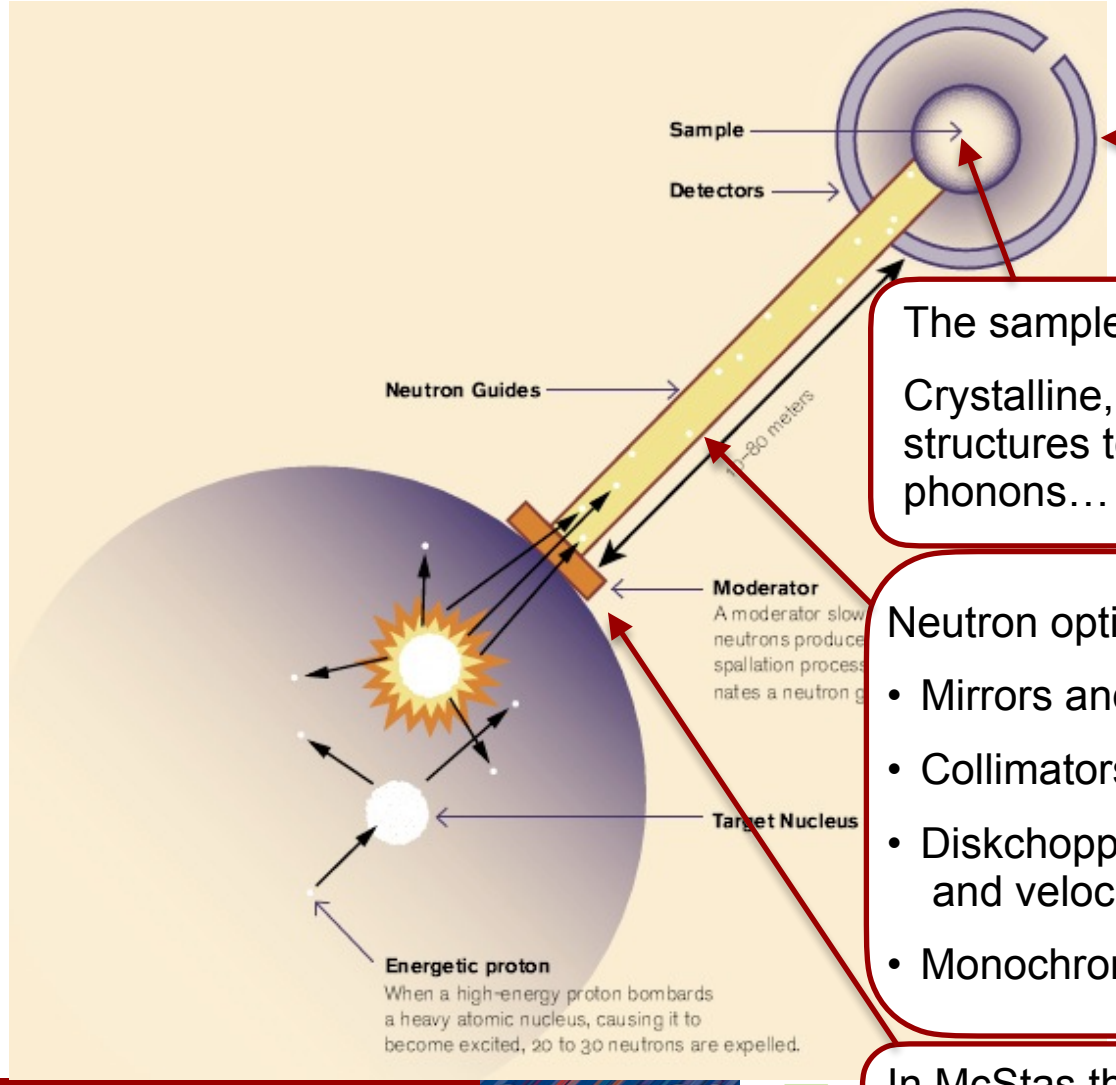


- complementary to X-rays

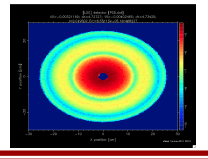


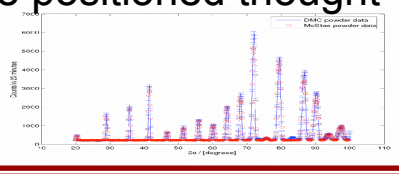


Components of neutron instruments



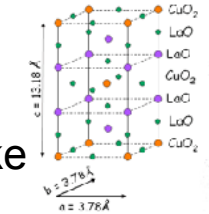
Detectors are “monitors” in McStas. Mostly they act as “perfect probes” and can be positioned thought your instrument gathering 1D/2D/ event lists…

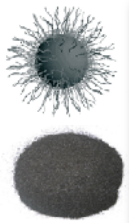




The sample:

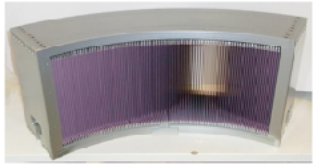
Crystalline, powders, liquids, micelles, structures to image, inelastic features like phonons…

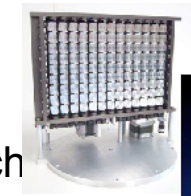


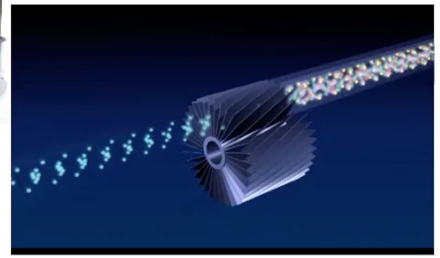


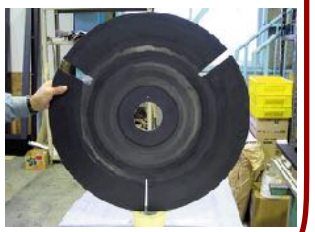
Neutron optics include things like:

- Mirrors and guides
- Collimators and slits
- Diskchoppers, Fermi ch and velocity selectors
- Monochromators/Analysers









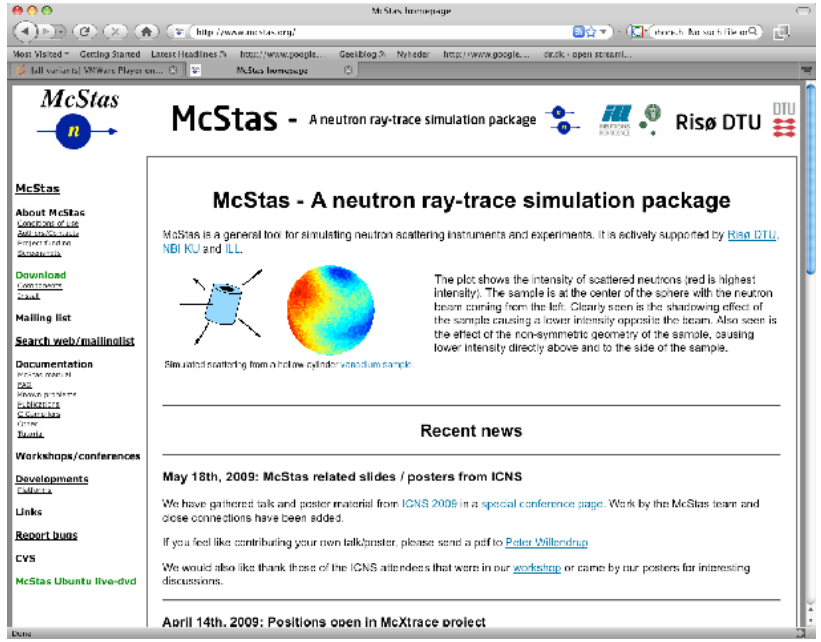
In McStas the moderator is the “source”

McStas Introduction

GNU GPL license

Open Source

- 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

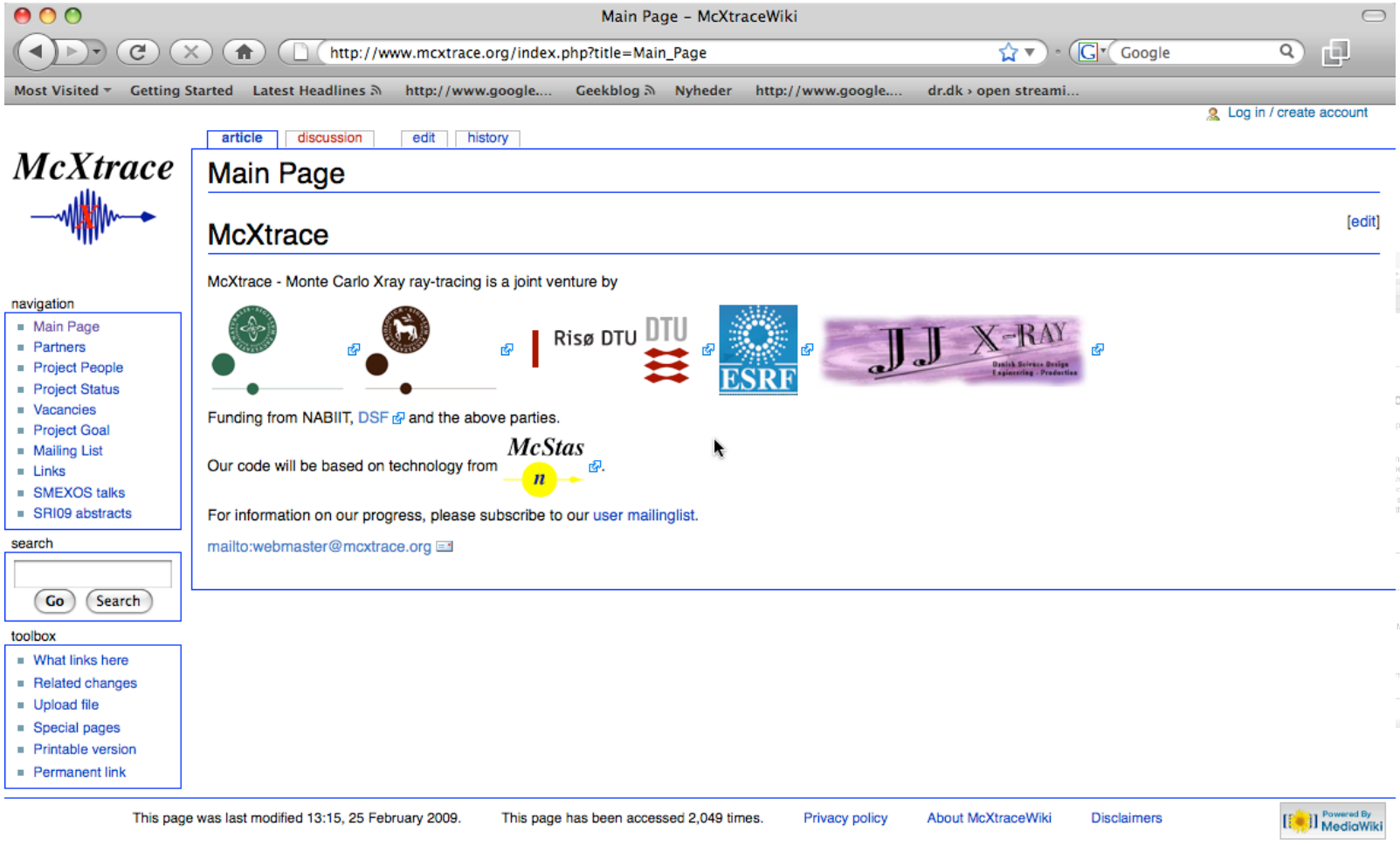


Project website at

http://www.mcstas.org

mcstas-users@mcstas.org mailinglist

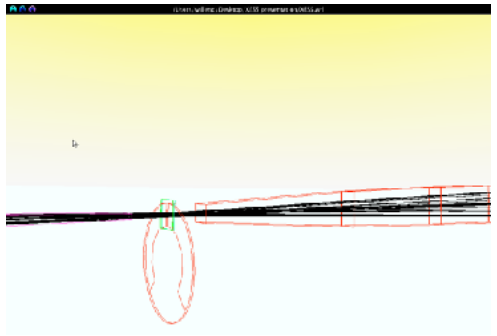
McXtrace - since jan 2009 similar for X-rays

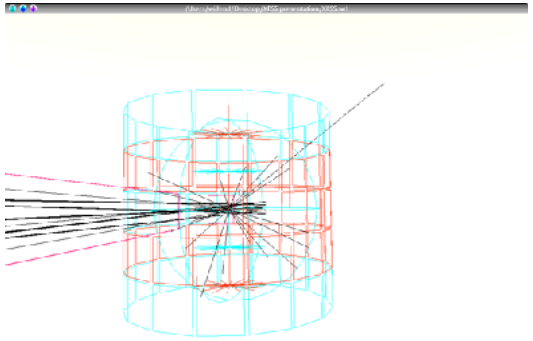


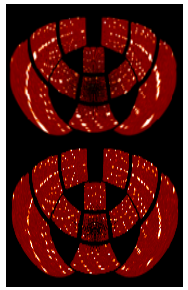
- Synergy, knowledge transfer, shared infrastructure and codebase on GitHub

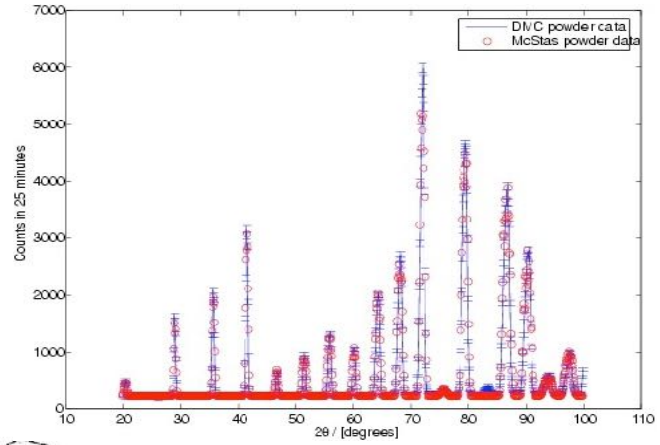
What is McStas used for?

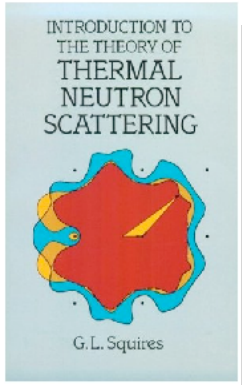
- Instrumentation
- Planning
- Construction
- Virtual experiments
- Data analysis
- Teaching
  (KU, DTU)

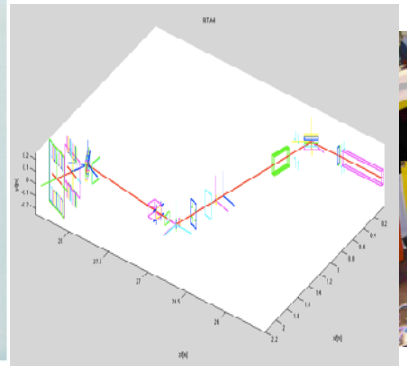


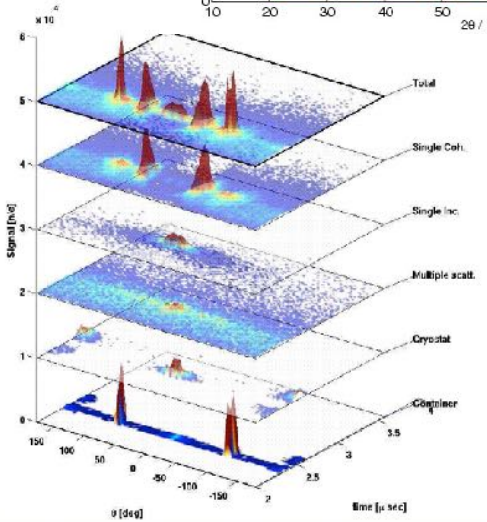




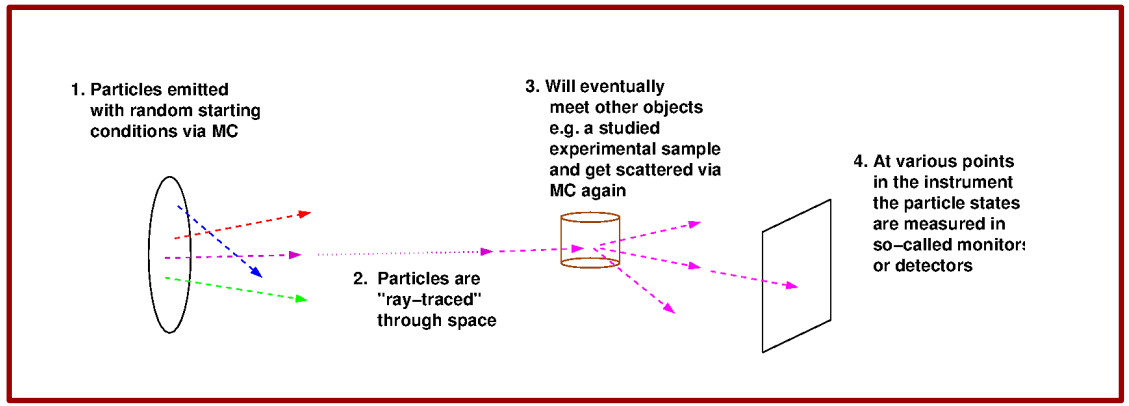


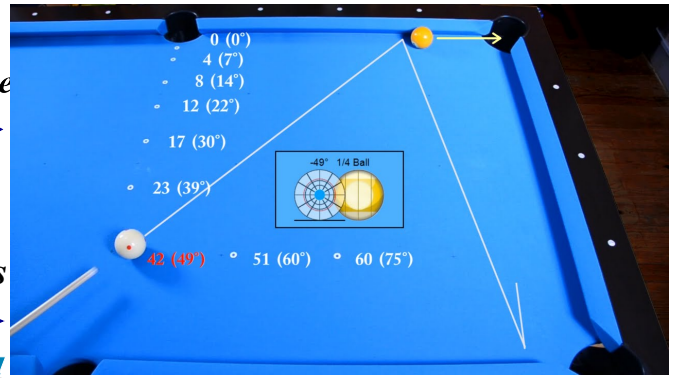


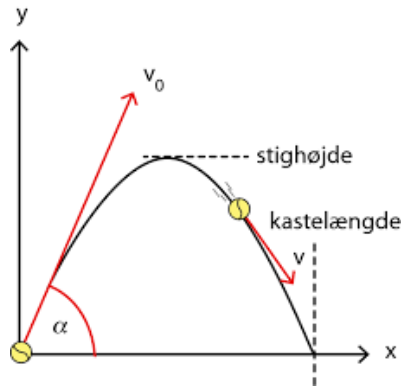


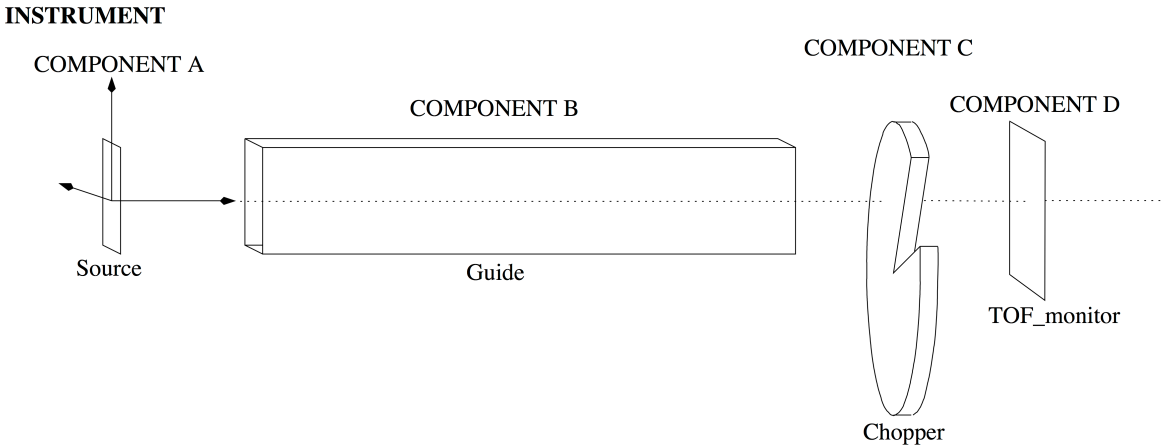


McStas and McXtrace Monte Carlo ray-tracers









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

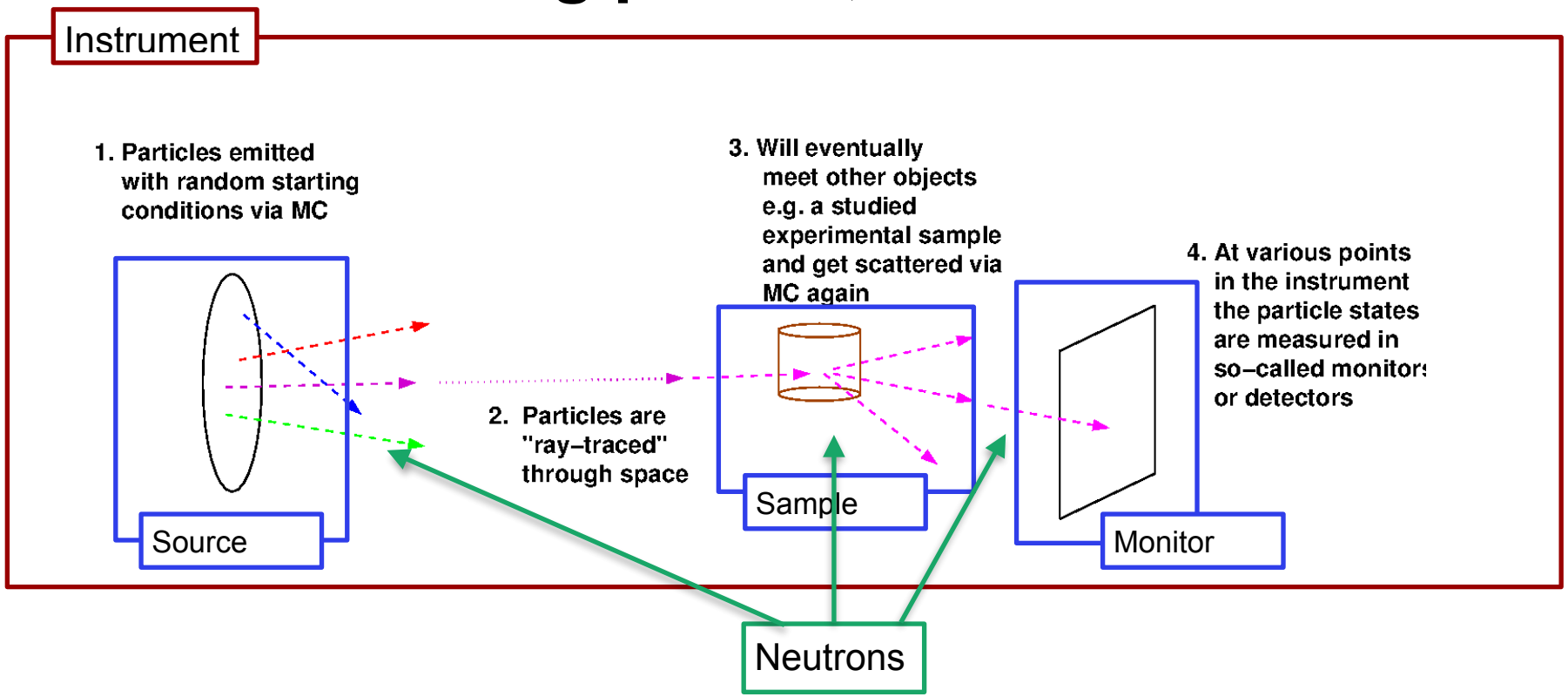
Random numbers	I/O	Physical consts.

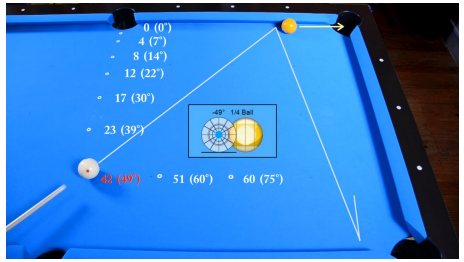
"Kernel and runtime c–code"

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

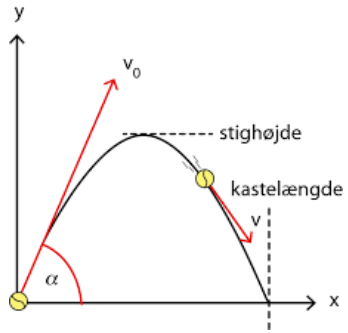
The simulation executable produces data output which can be visualized using the mcplot and mcdisplay tools

In the big picture, McStas is this…





Classical Newtonian mechanics (independent, particles though…)



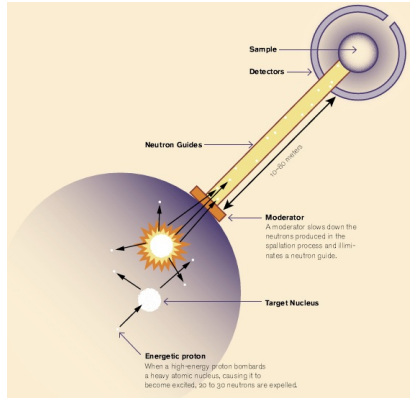
The instrument defines our “lab coordinate system”

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

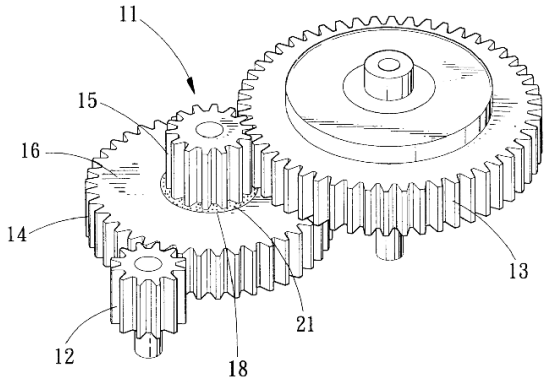
Neutron particles are passed on from one component to the next, changing state under way

McStas 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

DTU ess EUROPEAN SPALLATION SOURCE

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

Stats and information on the codebase

Languages

Total Lines :	2,132,114	Code Lines :	1,832,884	Percent Code Lines :	86.0%
Number of Languages :	23	Total Comment Lines :	203,852	Percent Comment Lines :	9.6%
		Total Blank Lines :	95,378	Percent Blank Lines :	4.5%

Language Breakdown

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
Postscript	901,336	9,202	1.0%	948	911,486	42.8%
AMPL	356,420	27,344	7.1%	4,012	387,776	18.2%
Fortran (Free-format)	216,230	86,604	28.6%	34,678	337,512	15.8%
C	126,112	31,224	19.8%	20,646	177,982	8.3%
JavaScript	100,016	1,116	1.1%	2,596	103,728	4.9%
Python	45,988	33,214	41.9%	18,278	97,480	4.6%
TeX/LaTeX	37,440	4,208	10.1%	5,940	47,588	2.2%
shell script	11,870	2,836	19.3%	2,954	17,660	0.8%
CMake	10,746	2,522	19.0%	2,492	15,760	0.7%
HTML	6,712	0	0.0%	238	6,950	0.3%
XML	4,486	58	1.3%	68	4,612	0.2%
Matlab	3,362	782	18.9%	550	4,694	0.2%
Perl	3,120	3,392	52.1%	48	6,560	0.3%
C++	2,968	448	13.1%	696	4,112	0.2%
TypeScript	2,718	12	0.4%	312	3,042	0.1%
NSIS	898	316	26.0%	380	1,594	0.1%
DOS batch script	696	236	25.3%	108	1,040	0.0%
CSS	662	8	1.2%	96	766	0.0%
Make	344	136	28.3%	122	602	0.0%
Fortran (Fixed-format)	340	0	0.0%	122	462	0.0%
IDL/PV-WAVE/GDL	232	134	36.6%	58	424	0.0%
Vim Script	160	36	18.4%	14	210	0.0%
R	28	24	46.2%	22	74	0.0%
Totals	1,832,884	203,852		95,378	2,132,114	

Project Summary : Factoids

Mature, well-established codebase

Large, active development team

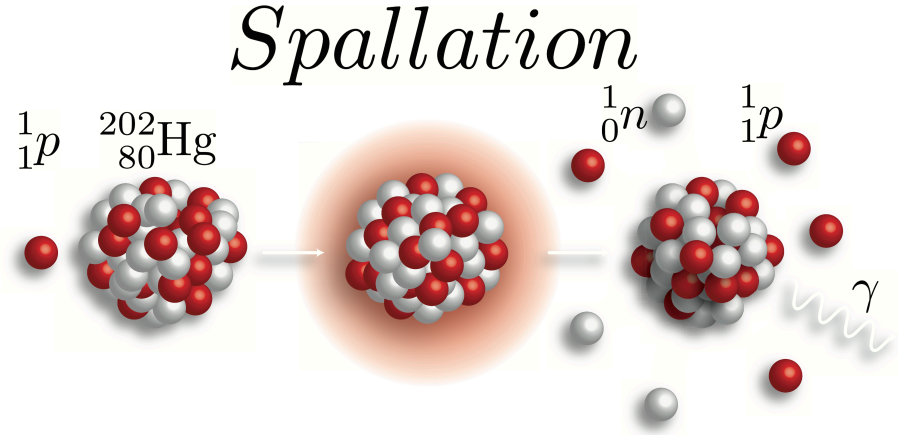
Stable Y-O-Y development activity

Very few source code comments

McCode is written mostly in AMPL.

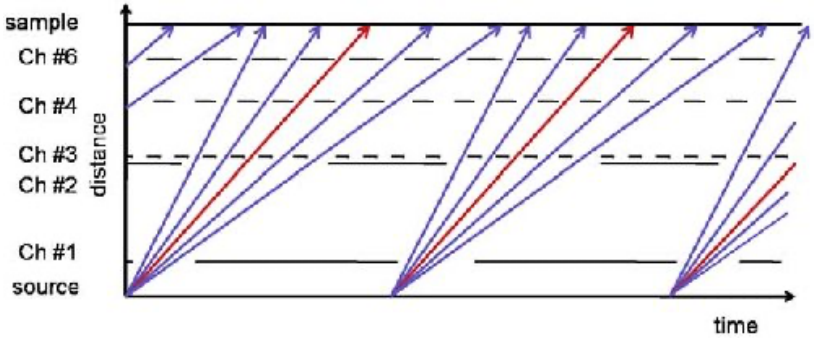
Nope, that's our DSL and grammar. :-) Which is close to "English".

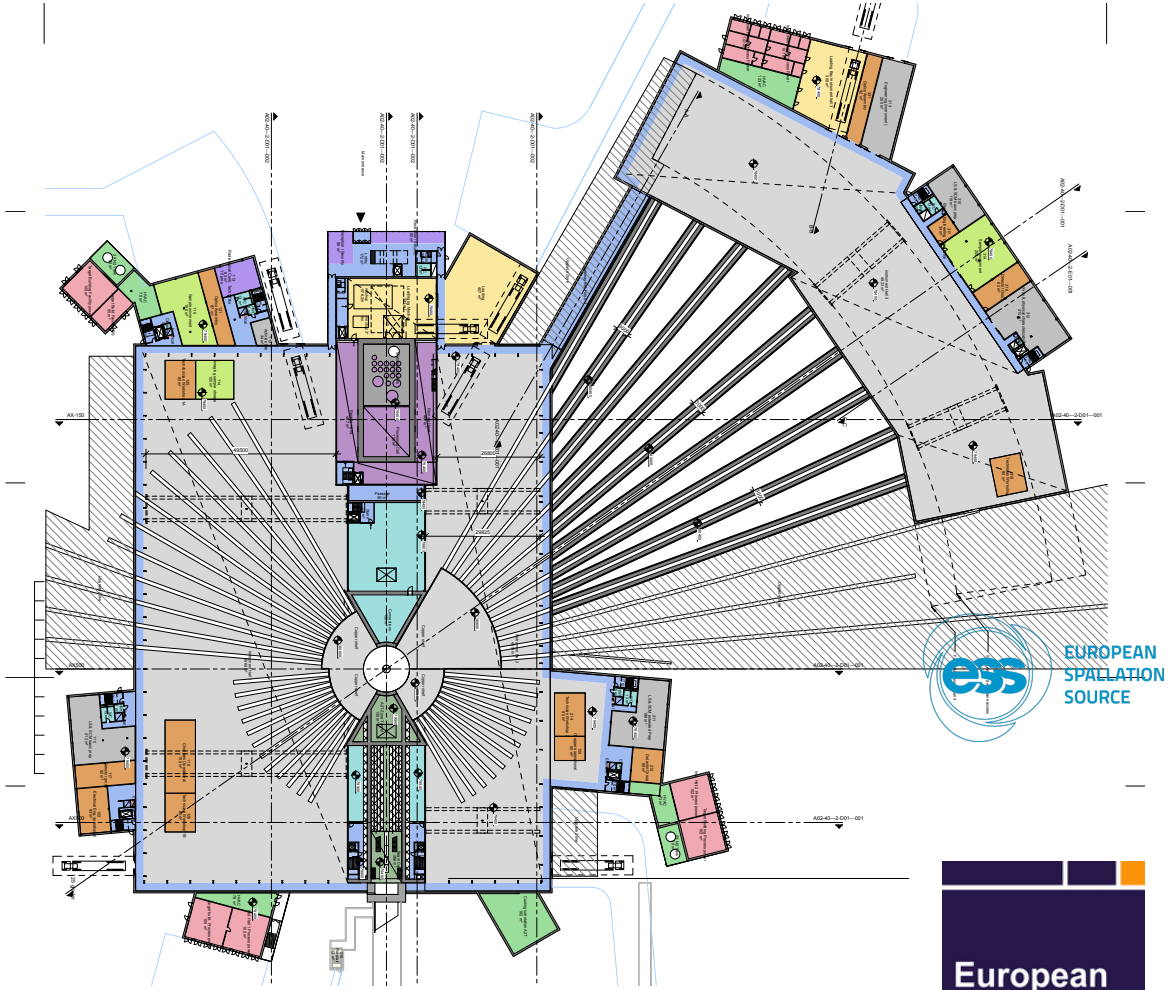
If it is so bloody good and “final”,why bother looking at GPU?



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!

GPU-computing in McStas - McXtrace

- LOTS of slowish processor-cores
- Limited bandwidth in CPU / GPU exchange

Perfect for highly-parallel calculations that have compact input / output, e.g. McStas.
(embarrassingly parallel)

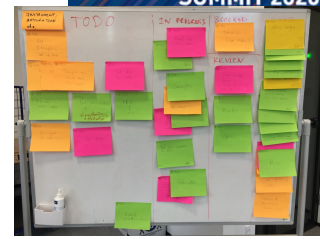
- Main HW providers today are NVIDIA, AMD, to some extent intel
- Software-frameworks
  - CUDA (NVIDIA-specific)
    - Accessible in C, ++, Fortran, shared library
  - OpenCL (hw-agnostic)
    - C-like programming language with own syntax
  - OpenACC (almost NVIDIA-specific, but extending
    - #pragma pre-compiler mechanism, accessible in C, ++, Fortran
  - OpenMP is picking up GPU-support…
  - Intel oneAPI
    - Claims to be a unified approach to CPU/GPU/MPI/… I didn't get around to really try it yet… ;-)

Prototype made by Brian Vinter students in 2010:
* Good speed-up of ~2 orders found
* Considered specific problem only
* Was float-only solution in hw at the time
* Would be too invasive on the code-base

Prototype made by Emmanuel Farhi + Post Doc in 2014:
* Would be too invasive on the code-base
* Effectively duplication of algorithm-code for CPU/GPU version
* Lots of locks / mutex'es needed placement everywhere
* Limited speedups found, e.g. ~ 15% if using GPU for RNG

What we eventually have gone for, with good results!

Main events on timeline of road toward GPU



2017: E. Farhi initial cogen modernisation

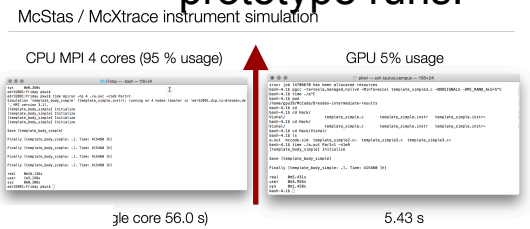
Fall 2018 onwards:
J. Garde further cogen modernisation and restructuring

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

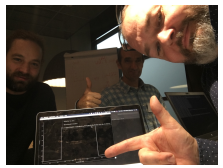
March 2018:
Participation at Dresden Hackathon.
1st "null" instrument prototype runs.

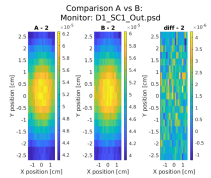


NVIDIA mentor: Vishal Metha

hackathon org.:
Guido Juckeland

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

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

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

McStas heading for the GPU… numbers from November 2019

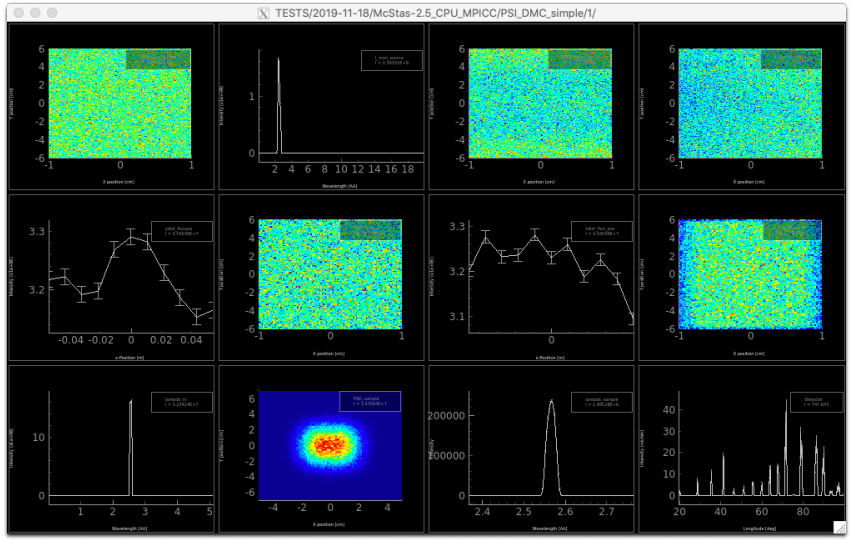
9 instruments fully ported, also realistic ones like PSI_DMC

(Aug 2020: 99 instrs)

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

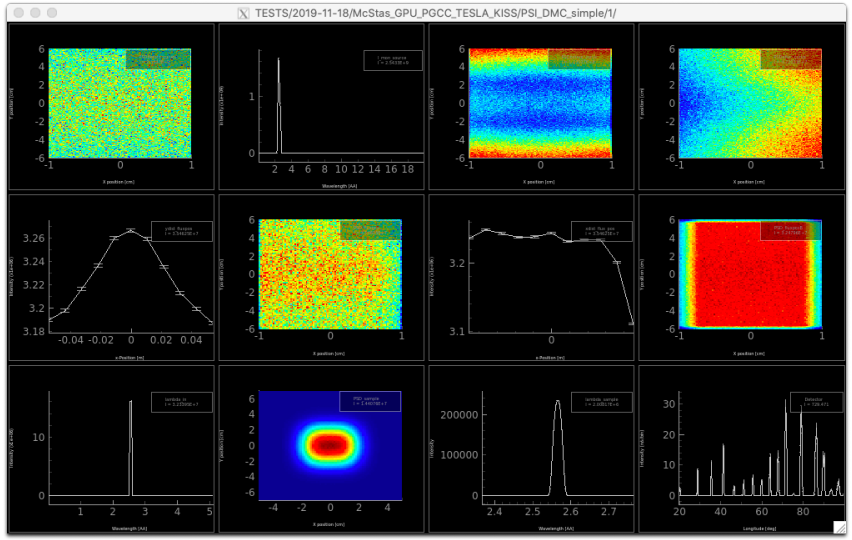
(1-core run, **1e9** would be 2000 secs)

VS.



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

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

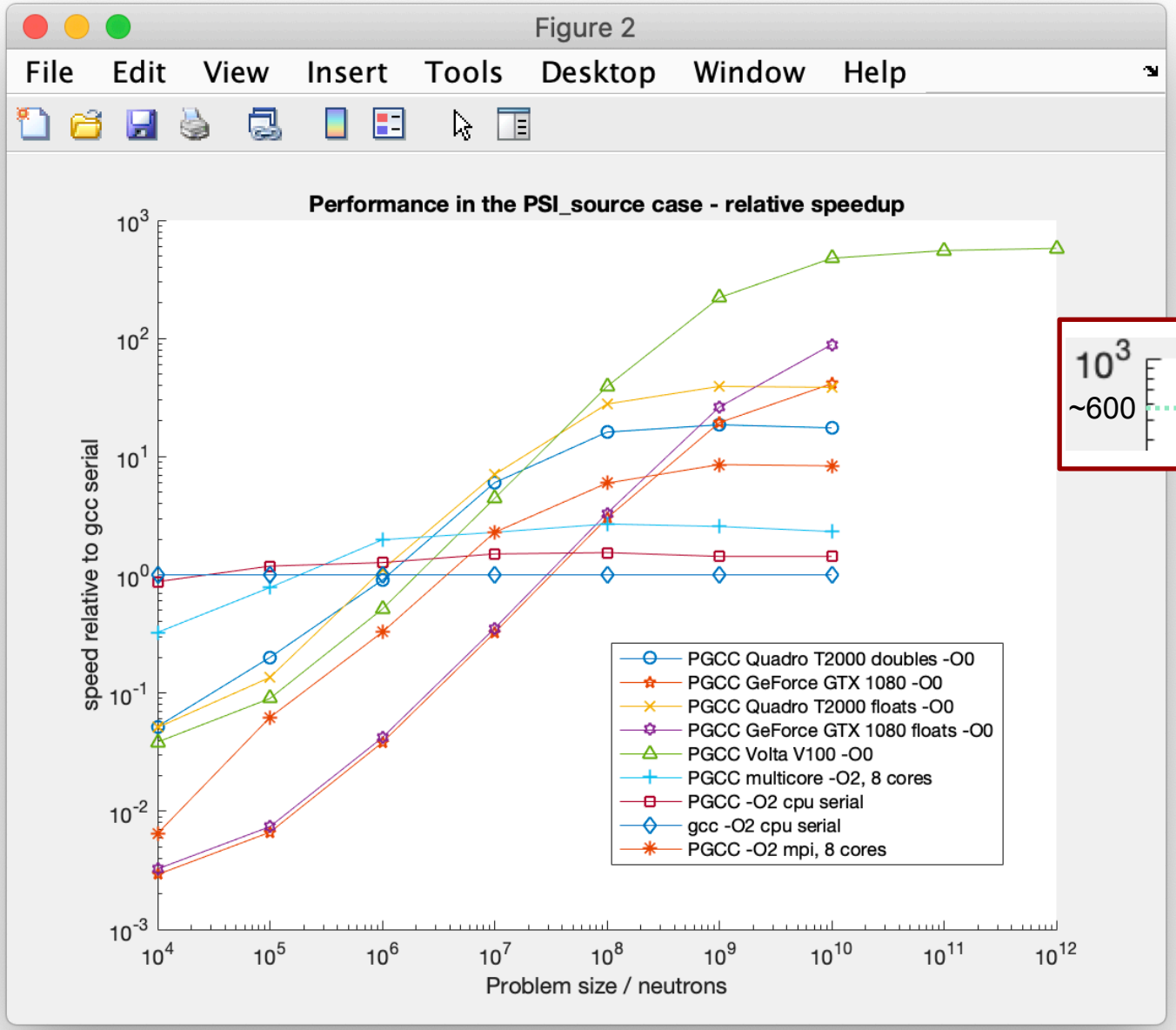


- If problem has the right size / complexity, GPU via OpenACC is great!

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

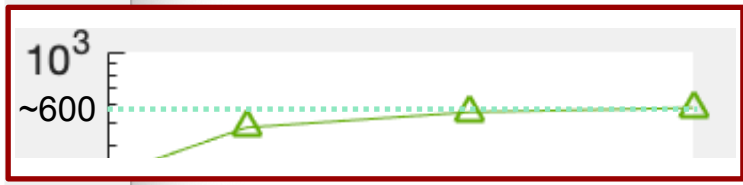
Idealised instrument with source and monitor only - i.e. without any use of the ABSORB macro.

(Likely a good indication of maximal speedup achievable.)



Speedup

Looks like a factor of ~600



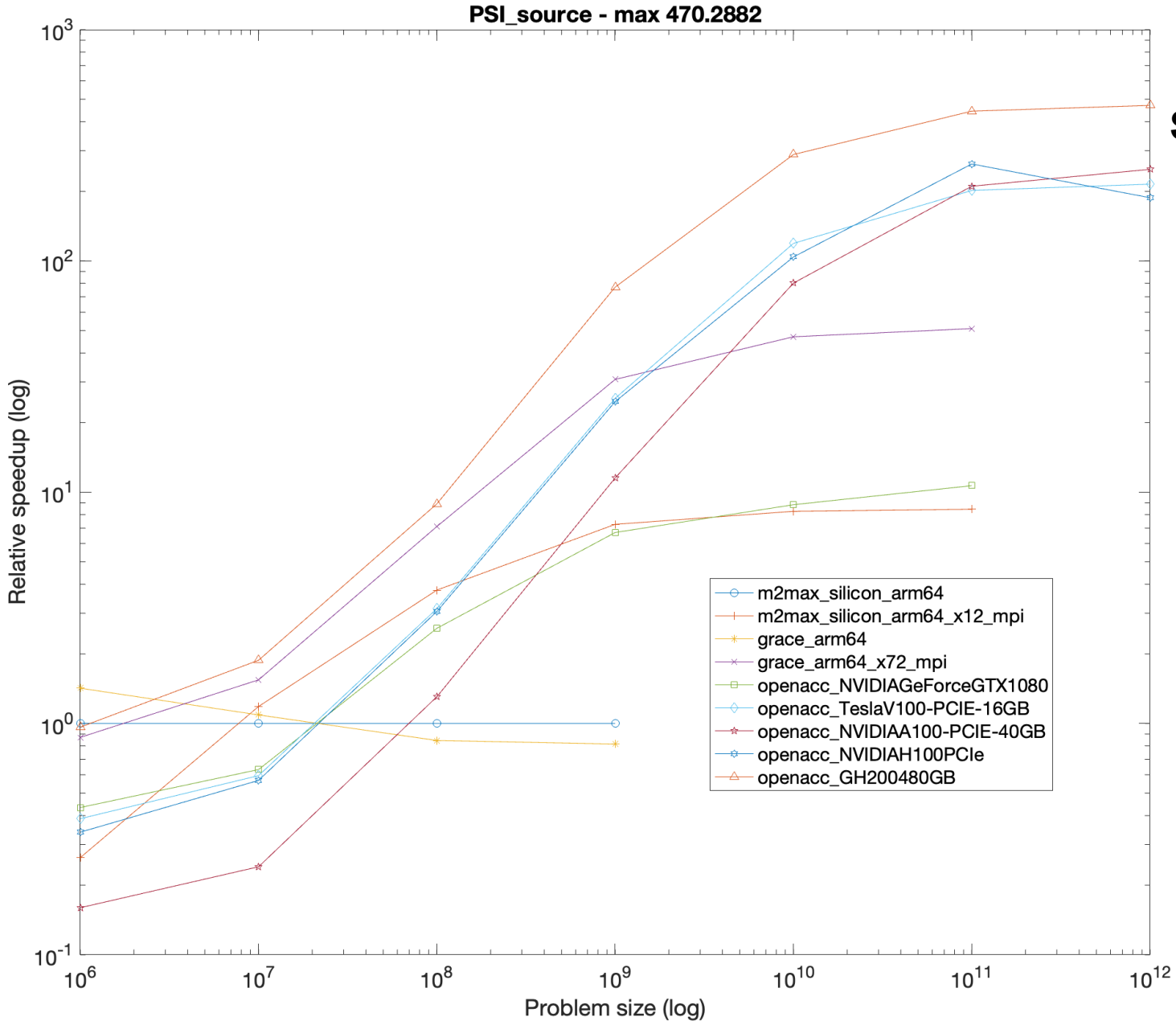
V100 execution speedups renormalised to wall-clock of single-core gcc standard simulation,

V100 run is 600 times faster than a single-core i7 CPU run

2025 measurement vs. arm64 cores in MacBook Pro / Grace Hopper

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

Factor of ~500 GH200 vs single arm64 core

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

DTU EUROPEAN SPALLATION SOURCE

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

McXtrace

McStas

DTU

ESS EUROPEAN SPALLATION SOURCE

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

Contains component trace section

"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.

McXtrace

McStas

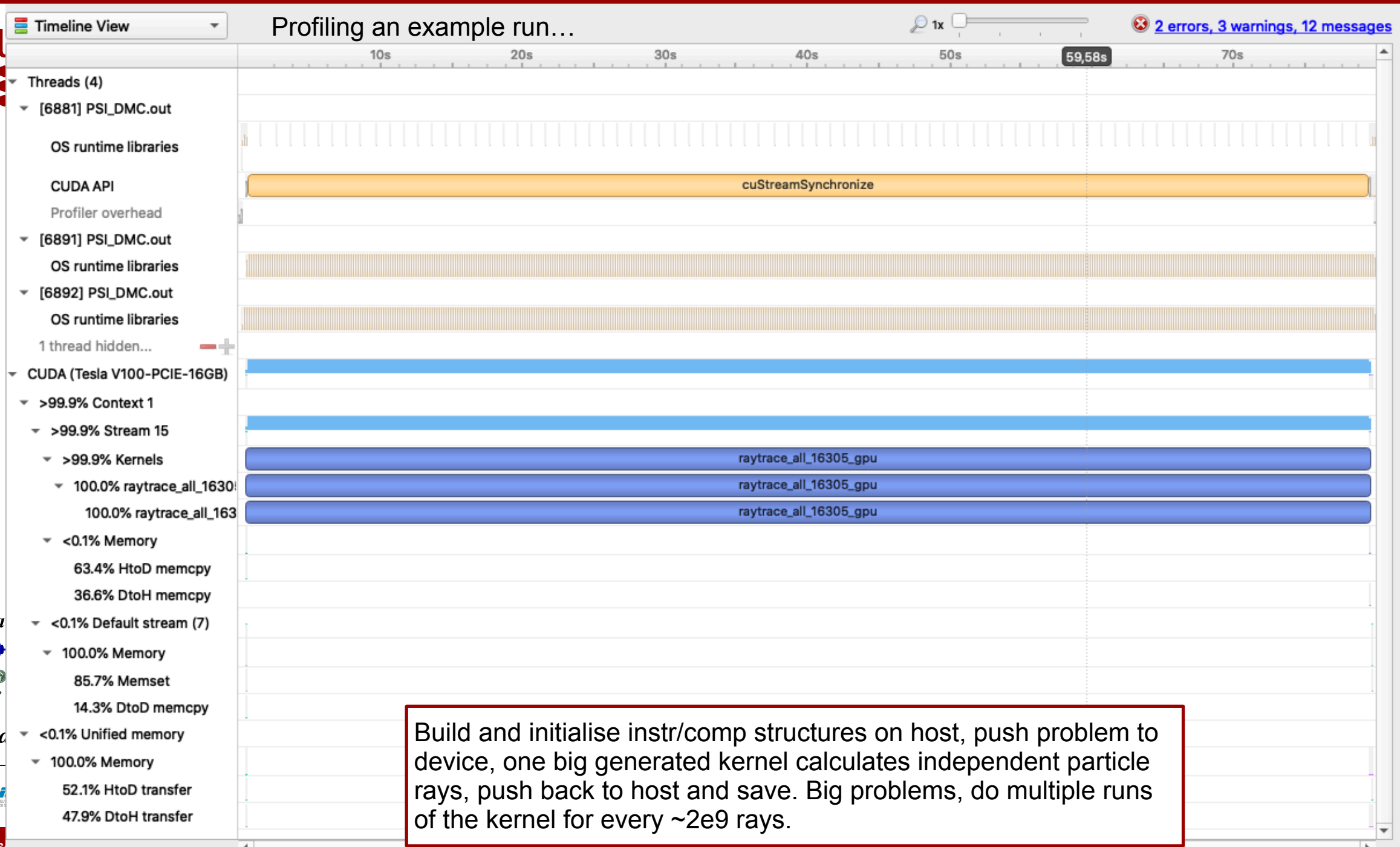
Anatomy of a McStas GPU run (*)

- 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

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

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

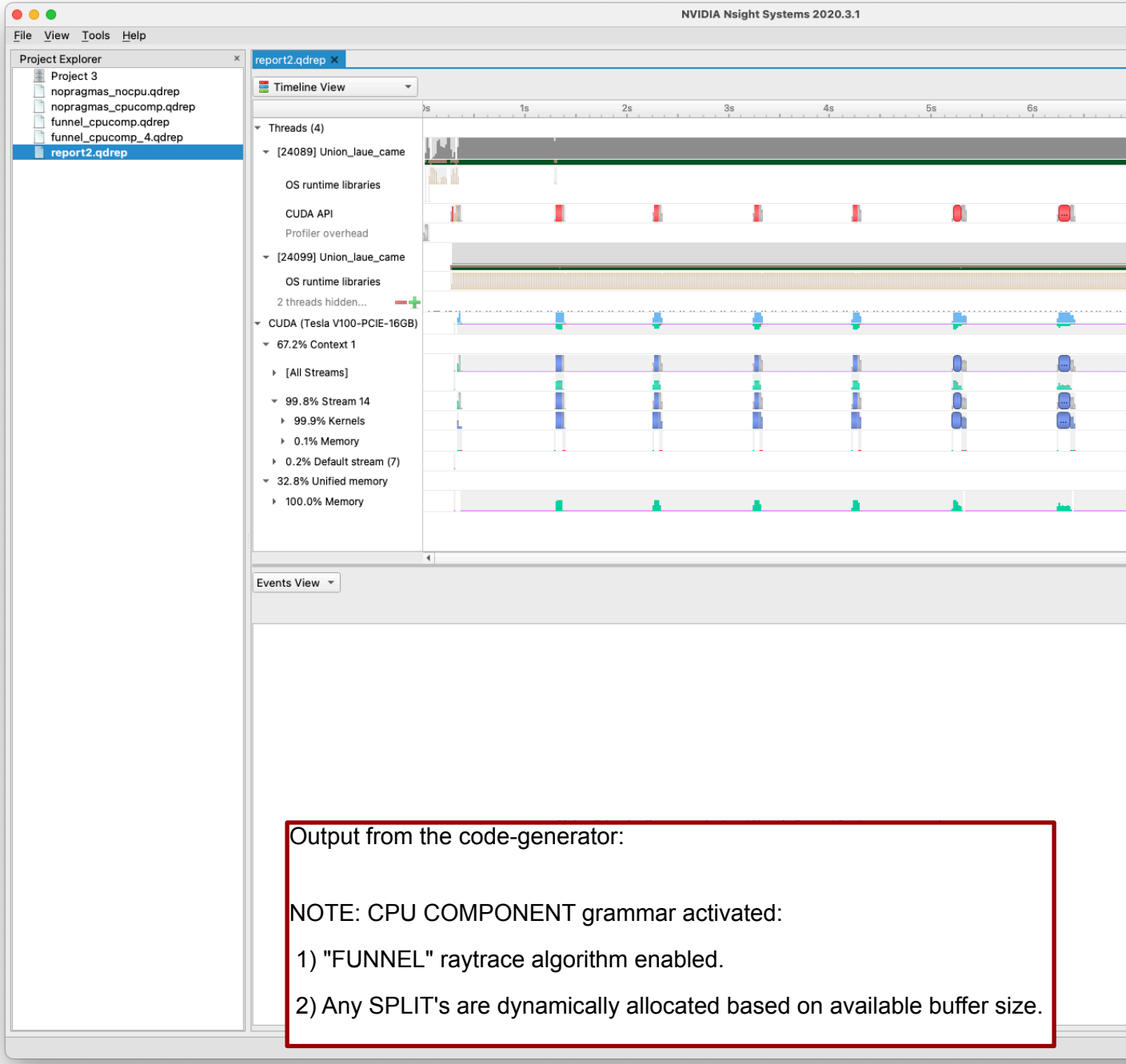
Profiling an example run…



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.

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
    - As slow as single-core cpu… (multicore??)
    - 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.

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". 😐

Compiler settings used for GPU:

(McStas mcrun is preconfigured on Linux, simply use mcrun --openacc when compiling, can also combine with e.g. --mpi=N for N GPU-exec)

nvc --gpu=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 NVIDIA GPU code, multicore also possible with ~ close to mpi performance

Compiler settings used for GPU:

(McStas mcrun is preconfigured on Linux, simply use mcrun --openacc when compiling, can also combine with e.g. --mpi=N for N GPU-exec)

nvc --gpu=managed -Minfo=accel -DOPENACC (-DFUNNEL)

Defaults for “GPU neutron loops”:
1) non-funnelled
   GPU_INNERLOOP=2147483647
2) funnelled
   GPU_INNERLOOP=1024*1024
3) control using --gpu_innerloop on binary

For CPU/threading, use below settings, with -DMULTICORE e.g. printfs are not nullified in TRACE

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

Pragmas in play…

We also use
* openacc.h e.g. for “attaching device pointers"
* accelmath.h for a math.h GPU replacement

+ some self-made replacements for e.g. string handling that are (otherwise) not available for GPU.

- Data need to be transferred to the GPU, we use
  - #pragma acc declare create(VAR) // put at VAR declaration
    double VAR;
  - #pragma acc update device(VAR) // after assignment

- Main particle loop has a
  - #pragma acc parallel loop
    for (unsigned long pidx=0 ; pidx < innerloop ; pidx++) {

- Any function to be evaluated on GPU needs. Put in place by code-generator whenever we can…
  - #pragma acc routine // on fct. prototype or on actual function def.
    _class_Source_div *class_Source_div_trace(_class_Source_div *_comp , _class_particle *_particle) {

- If writing to VAR is necessary, this can be done *atomically* using e.g.
  - #pragma acc atomic
    VAR = VAR + 1; // ++ operators etc. is too complex

- We pull data back using this mechanism
  - #pragma acc update host(VAR)

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 (means: "trigger FUNNEL mode and run comp on CPU" with acc sections around, see later slide)
- **Variadic functions are not available on GPU**
- **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

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 almost all of the rest)
- **Documentation** on McCode Wiki…
- McStas 3.0 is as of yet “ported” to GPU but **not fully “optimised” performance-wise**, we will try to **go to another Hackathon < This one!**
- **Union** needs a dedicated **Hackathon**

Ideas to investigate / work on?

- Investigate **getting rid of 'managed mode'**
  - (2D array-vars > 1D, add #pragma acc shape and make simpler structs?)

- Get **"multicore" functional on "NOACC" FUNNEL sections**?

- **64bit vs. 32bit Floats** (often only ~ factor 2 speed up)
  - MCNUM where applicable, use a define to decide

- **Optimise a set of selected instrument models**, for instance these:
  - Source + monitor ideal case
  - simple optics model from McStas (guide system?)
  - instrument with nested SPLITs (templateTAS?)
  - Powder diffractometer from McXtrace
  - OFF-oriented instrument, e.g. Airport_Scanner McXtrace
  - Laue instrument using Single crystal - NMX?

- Wanted outcomes:
  - **Learn to practically use the Nvidia profilers** (ncu / nvprof)
  - Get a better **understanding of the limitations** of the current implementation
  - Try to **identify obvious bottlenecks**

- Other
  - share/ snippets > **proper shared libs**? CPU conda libs, tooling for GPU libs maybe?

The team, Nvidia mentors and Hackathon hosts :-)

Vishal Metha

Christian Hundt

Alexey Romanenko

Guido Juckeland

Sebastian von Alfthan