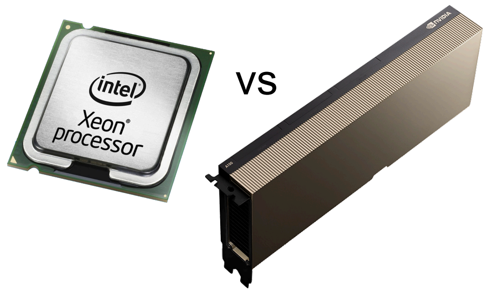
<u>Peter Kjær Willendrup</u>[1,2], Emmanuel Farhi[3], Mads Bertelsen[2], Torben Roland Nielsen[2], Tobias Weber[4], Erik Bergbäck Knudsen[1,5], Jakob Garde[6] , Tobias Weber[4]

[1]Technical University of Denmark / Danmarks Tekniske Universitet (*Department of Physics*) [2]European Spallation Source ERIC (*Data Management and Software Center*) , [3]Synchrotron SOLEIL, [4]Institut Laue Langevin, [5]Copenhagen Atomics, [6]Technical University of Denmark / Danmarks Tekniske Universitet (*Department of Electrical Engineering*)

Speeding up legacy:

GPU-accelerating the McStas instrument simulation code using OpenACC.

McStas



Agenda

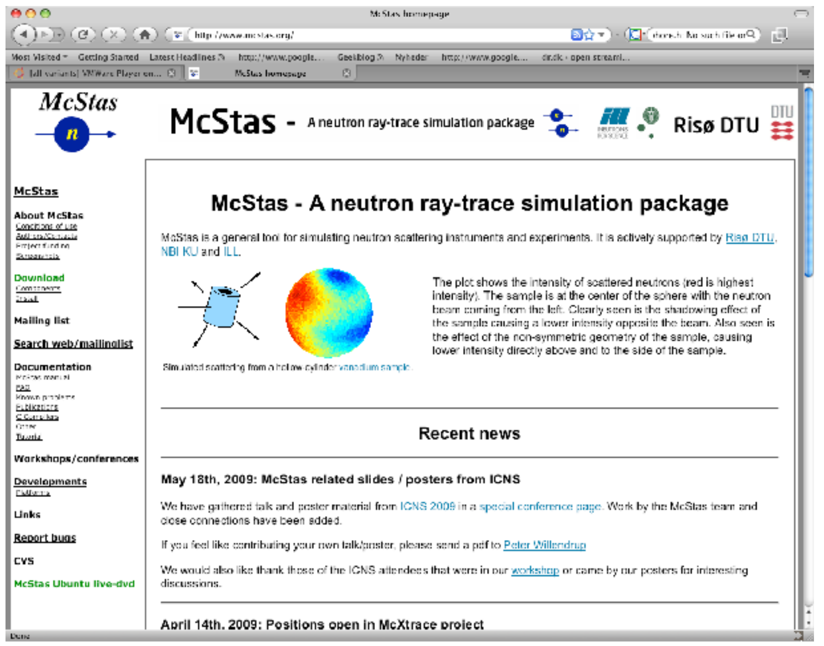
- What is McStas in two slides
- Why and how was McStas ported to GPU’s
- How well (fast) does it work?
- HOWTO steps:
  Porting an instrument / component
- Conclusions

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
  (work initiated in 1997, 25 year project anniversary, 2023 anniversary release)
- Small, dedicated team, many contributions from users, students
- Similar for X-rays, see http://www.mcxtrace.org - we share code base, tools and infrastructure.



McStas - A neutron ray-trace simulation package

McStas is a general tool for simulating neutron scattering instruments and experiments. It is actively supported by Risø DTU, NBI KU and ILL.

Simulated scattering from a hollow cylinder vanadium sample.

The plot shows the intensity of scattered neutrons (red is highest intensity). The sample is at the center of the sphere with the neutron beam coming from the left. Clearly seen is the shadowing effect of the sample causing a lower intensity opposite the beam. Also seen is the effect of the non-symmetric geometry of the sample, causing lower intensity directly above and to the side of the sample.

Recent news

May 18th, 2009: McStas related slides / posters from ICNS

We have gathered talk and poster material from ICNS 2009 in a special conference page. Work by the McStas team and close connections have been added.

If you feel like contributing your own talk/poster, please send a pdf to Peter Willendrup

We would also like thank those of the ICNS attendees that were in our workshop or came by our posters for interesting discussions.

April 14th, 2009: Positions open in McXtrace project

Project website at

http://www.mcstas.org

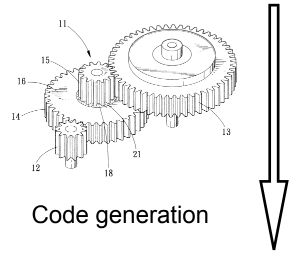
mcstas-users@mcstas.org mailinglist

Software facts

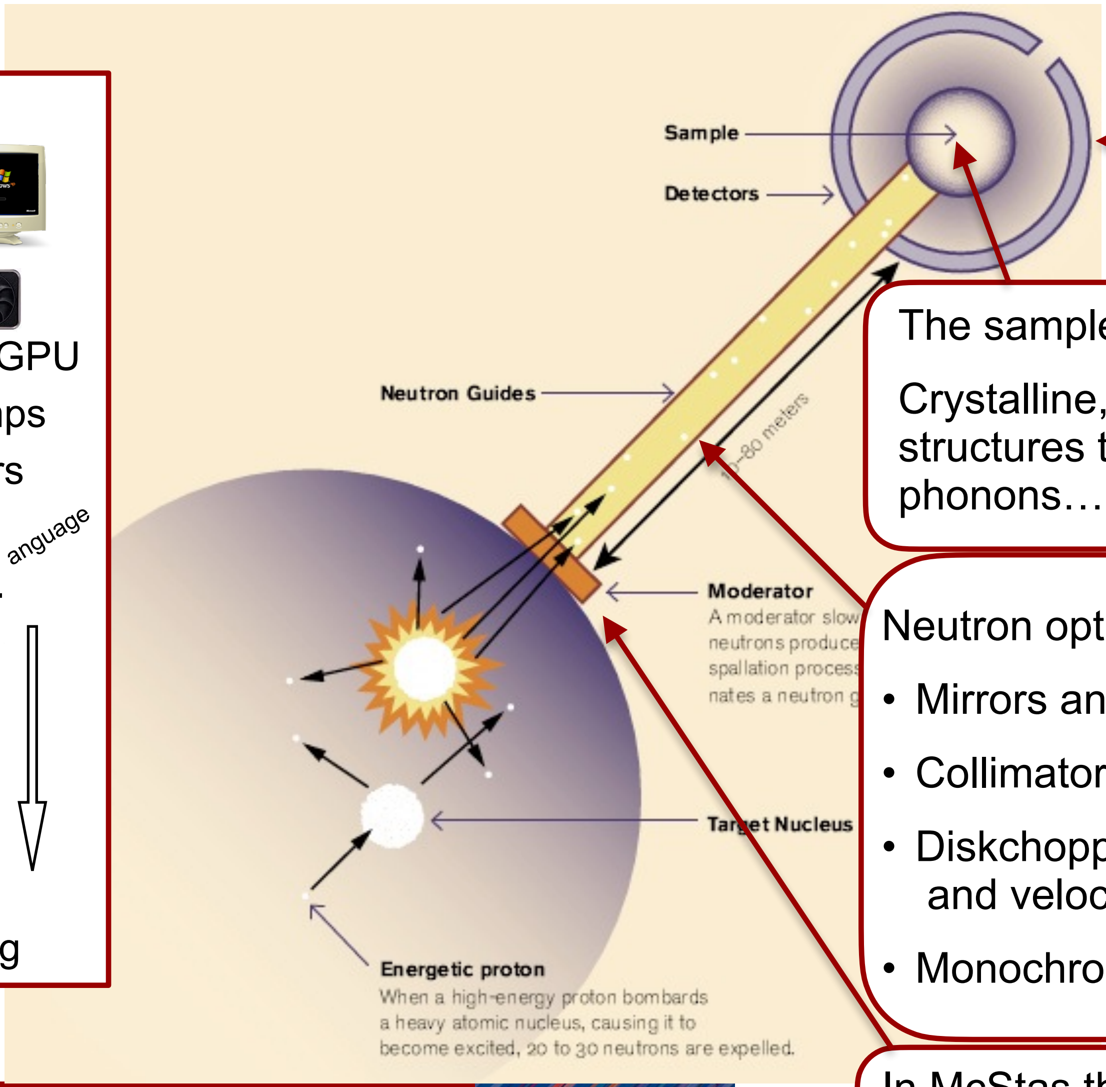
Components of neutron instruments

- Portable

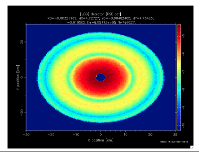
- CPU/MPI/GPU
- ~ 250 comps
- ~ 250 instrs
- D.omain S.pecific L.anguage

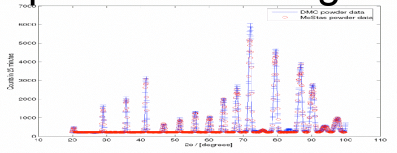


- C-code
- Binary prog



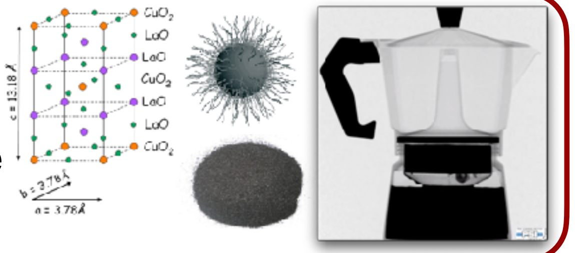
Detectors are "monitors" in McStas. Mostly they act as "perfect probes" and can be positioned thought your instrument gathering 1D/2D/ event lists…





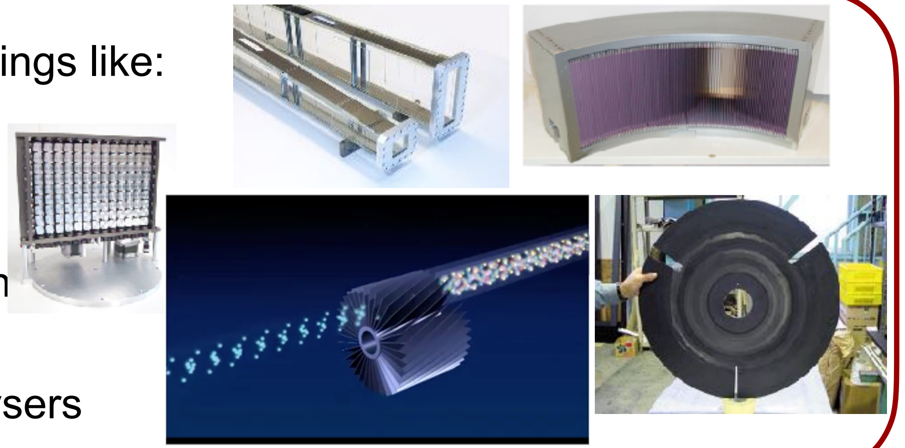
The sample:

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

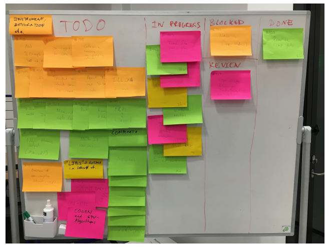


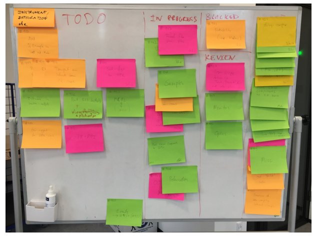
In McStas the moderator is the "source"

Main events on timeline of road toward GPU

McStas

McXtrace





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.

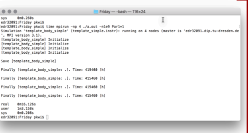
2020 1st **Corona** lockdown
P. Willendrup & E. Knudsen continue work on comp and cogen

December 15th 2020
McStas 3.0 release!

November 24th, 2021
McStas 3.1 release!

McStas / McXtrace instrument simulation

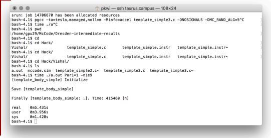
CPU MPI 4 cores (95 % usage)



```
Friday — -bash — 116×24
sys     0m0.260s
edr32091:Friday pkwi$
edr32091:Friday pkwi$ time mpirun -np 4 ./a.out -n1e9 Par1=1
Simulation 'template_body_simple' (template_simple.instr): running on 4 nodes (master is 'edr32091.dip.tu-dresden.de
', MPI version 3.1).
[template_body_simple] Initialize
[template_body_simple] Initialize
[template_body_simple] Initialize
[template_body_simple] Initialize

Save [template_body_simple]

Finally [template_body_simple: .]. Time: 415460 [h]

Finally [template_body_simple: .]. Time: 415460 [h]

Finally [template_body_simple: .]. Time: 415460 [h]

Finally [template_body_simple: .]. Time: 415460 [h]

real    0m16.126s
user    1m3.158s
sys     0m0.208s
edr32091:Friday pkwi$
```

16.12 s (Single core 56.0 s)

GPU 5% usage

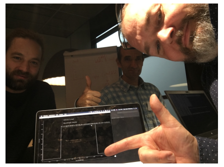


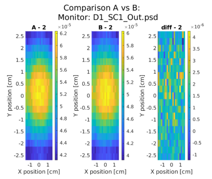
```
pkwi — ssh taurus.campus — 108×24
srun: job 14706670 has been allocated resources
bash-4.1$ pgcc -ta=tesla,managed,nollvm -Minfo=accel template_simple3.c -DNOSIGNALS -DMC_RAND_ALG=5^C
bash-4.1$ time ./a^C
bash-4.1$ pwd
/home/gpu29/McCode/Dresden-intermediate-results
bash-4.1$ cd
bash-4.1$ cd Hack/
Vishal/              template_simple.c      template_simple.instr   template_simple.instr~
bash-4.1$ cd Hack/
Vishal/              template_simple.c      template_simple.instr   template_simple.instr~
bash-4.1$ cd Hack/Vishal/
bash-4.1$ ls
a.out  mccode.sim  template_simple2.c~  template_simple3.c  template_simple3.c~
bash-4.1$ time ./a.out Par1=1 -n1e9
[template_body_simple] Initialize

Save [template_body_simple]

Finally [template_body_simple: .]. Time: 415460 [h]

real    0m5.431s
user    0m3.956s
sys     0m1.428s
bash-4.1$
```

5.43 s

mentor: Vishal Metha

hackathon org.:
Guido Juckeland

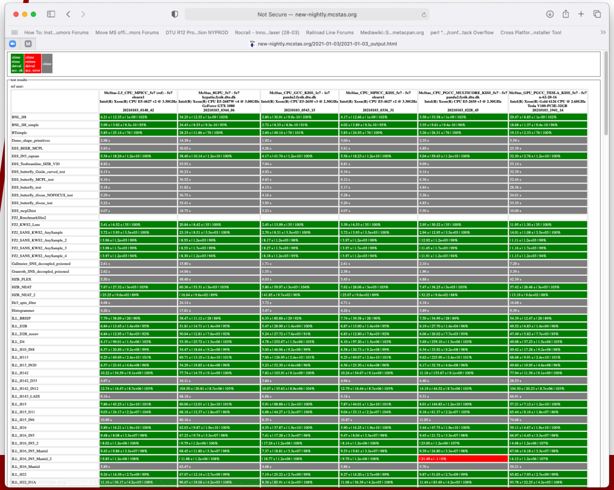




mentor: Christian Hundt

hackathon org.:
Sebastian Von Alfthan

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



Not Secure — new-nightly.mcstas.org

new-nightly.mcstas.org/2021-01-03/2021-01-03_output.html

🟩	🟥	▨
ctime rtime detval acc. ok	ctime rutime detval acc. error	ctime

test results

ref user:

	McStas-2.5_CPU_MPICC_5e7 (ref) - 5e7 elearn1 Intel(R) Xeon(R) CPU E5-4627 v2 @ 3.30GHz 20210103_0148_42	McStas_8GPU_5e7 - 5e7 hypatia.fysik.dtu.dk Intel(R) Xeon(R) CPU E5-2687W v4 @ 3.00GHz GeForce GTX 1080 20210103_0344_06	McStas_CPU_GCC_KISS_1e7 - 1e7 panda2.fysik.dtu.dk Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz 20210103_0543_15	McStas_CPU_MPICC_KISS_5e7 - 5e7 elearn1 Intel(R) Xeon(R) CPU E5-4627 v2 @ 3.30GHz 20210103_0336_31	McStas_CPU_PGCC_MULTICORE_KISS_5e7 - 5e7 panda2.fysik.dtu.dk Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz 20210103_0228_45	McStas_GPU_PGCC_TESLA_KISS_5e7 - 5e7 n-62-20-16 Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz Tesla V100-PCIE-32GB 20210103_1941_16
BNL_H8	4.21 s | 12.35 s | 1e-09 | 102%	34.25 s | 12.55 s | 1e-09 | 102%	2.80 s | 30.91 s | 9.9e-10 | 100%	4.17 s | 12.66 s | 1e-09 | 102%	3.58 s | 33.58 s | 1e-09 | 102%	29.47 s | 8.85 s | 1e-09 | 102%
BNL_H8_simple	3.99 s | 3.92 s | 9.3e-10 | 95%	24.43 s | 8.53 s | 9.3e-10 | 95%	2.72 s | 8.33 s | 8.9e-10 | 91%	4.02 s | 3.89 s | 9.3e-10 | 95%	3.35 s | 9.61 s | 9.4e-10 | 96%	18.68 s | 1.37 s | 9.4e-10 | 96%
BTsimple	3.85 s | 25.14 s | 76 | 100%	28.23 s | 11.86 s | 76 | 100%	2.60 s | 49.14 s | 76 | 101%	3.83 s | 24.93 s | 76 | 100%	3.26 s | 26.31 s | 76 | 100%	19.13 s | 2.33 s | 76 | 100%
Demo_shape_primitives	2.98 s	14.39 s	1.92 s	3.00 s	2.35 s	5.59 s
ESS_BEER_MCPL	5.63 s	30.05 s	4.26 s	5.61 s	4.89 s	25.39 s
ESS_IN5_reprate	5.54 s | 18.24 s | 1.2e+10 | 100%	38.40 s | 10.14 s | 1.2e+10 | 100%	4.17 s | 41.74 s | 1.2e+10 | 100%	5.56 s | 18.23 s | 1.2e+10 | 100%	5.04 s | 59.43 s | 1.2e+10 | 100%	32.30 s | 2.76 s | 1.2e+10 | 100%
ESS_Testbeamline_HZB_V20	8.82 s	55.93 s	7.04 s	8.81 s	9.99 s	53.14 s
ESS_butterfly_Guide_curved_test	6.13 s	36.23 s	4.92 s	6.16 s	6.14 s	32.39 s
ESS_butterfly_MCPL_test	6.16 s	36.52 s	4.83 s	6.12 s	6.36 s	32.44 s
ESS_butterfly_test	5.18 s	31.82 s	4.13 s	5.17 s	4.84 s	28.36 s
ESS_butterfly_tfocus_NOFOCUS_test	5.29 s	36.75 s	4.16 s	5.28 s	5.28 s	34.01 s
ESS_butterfly_tfocus_test	5.22 s	35.41 s	3.95 s	5.20 s	4.85 s	33.35 s
ESS_mcpl2hist	4.07 s	18.75 s	3.23 s	4.07 s	3.50 s	10.00 s
FZJ_BenchmarkSfin2						
FZJ_KWS2_Lens	3.41 s | 6.52 s | 35 | 100%	20.84 s | 8.42 s | 35 | 100%	2.45 s | 13.99 s | 35 | 100%	3.39 s | 6.53 s | 35 | 100%	2.95 s | 30.52 s | 35 | 100%	11.95 s | 1.30 s | 35 | 100%
FZJ_SANS_KWS2_AnySample	3.72 s | 3.93 s | 3.5e+03 | 100%	23.19 s | 8.51 s | 3.5e+03 | 100%	2.70 s | 8.31 s | 3.5e+03 | 100%	3.72 s | 3.93 s | 3.5e+03 | 100%	2.94 s | 12.95 s | 3.5e+03 | 100%	14.01 s | 1.08 s | 3.5e+03 | 100%
FZJ_SANS_KWS2_AnySample_2	| 3.96 s | 1.2e+03 | 99%	| 8.55 s | 1.2e+03 | 99%	| 8.17 s | 1.2e+03 | 98%	| 3.97 s | 1.2e+03 | 99%	| 12.92 s | 1.2e+03 | 99%	| 1.11 s | 1.2e+03 | 99%
FZJ_SANS_KWS2_AnySample_3	| 3.98 s | 1.7e+03 | 99%	| 8.55 s | 1.7e+03 | 99%	| 8.27 s | 1.7e+03 | 99%	| 3.97 s | 1.7e+03 | 99%	| 11.45 s | 1.7e+03 | 99%	| 1.16 s | 1.7e+03 | 99%
FZJ_SANS_KWS2_AnySample_4	| 3.97 s | 1.2e+03 | 94%	| 8.30 s | 1.2e+03 | 94%	| 8.18 s | 1.2e+03 | 95%	| 3.97 s | 1.2e+03 | 94%	| 11.91 s | 1.2e+03 | 94%	| 1.13 s | 1.2e+03 | 94%
Gallmeier_SNS_decoupled_poisoned	2.61 s	15.80 s	1.71 s	2.61 s	2.10 s	7.29 s
Granroth_SNS_decoupled_poisoned	2.62 s	14.06 s	1.55 s	2.58 s	1.96 s	5.39 s
HZB_FLEX	5.50 s	48.40 s	4.03 s	5.45 s	4.88 s	42.39 s
HZB_NEAT	7.67 s | 27.32 s | 3e+03 | 103%	60.36 s | 53.31 s | 3e+03 | 103%	5.80 s | 59.97 s | 3e+03 | 104%	7.62 s | 28.06 s | 3e+03 | 103%	7.47 s | 36.25 s | 3e+03 | 103%	57.42 s | 26.46 s | 3e+03 | 103%
HZB_NEAT_2	| 25.25 s | 9.6e+02 | 89%	| 16.64 s | 9.6e+02 | 89%	| 41.85 s | 9.7e+02 | 90%	| 25.07 s | 9.6e+02 | 89%	| 32.25 s | 9.6e+02 | 88%	| 13.19 s | 9.6e+02 | 88%
He3_spin_filter	4.68 s	24.14 s	3.72 s	4.71 s	4.18 s	16.68 s
Histogrammer	4.26 s	17.81 s	3.47 s	4.24 s	3.89 s	9.39 s
ILL_BRISP	7.79 s | 38.69 s | 28 | 90%	58.47 s | 11.12 s | 28 | 88%	6.33 s | 80.88 s | 29 | 92%	7.74 s | 39.58 s | 28 | 90%	7.50 s | 34.99 s | 28 | 89%	54.39 s | 12.47 s | 28 | 89%
ILL_D2B	6.84 s | 13.45 s | 1.4e+04 | 95%	51.81 s | 14.71 s | 1.4e+04 | 95%	5.47 s | 28.90 s | 1.4e+04 | 100%	6.87 s | 13.00 s | 1.4e+04 | 95%	6.19 s | 27.70 s | 1.4e+04 | 96%	49.52 s | 6.83 s | 1.4e+04 | 96%
ILL_D2B_noenv	6.84 s | 12.95 s | 7.6e+03 | 92%	50.04 s | 12.81 s | 7.6e+03 | 92%	5.24 s | 27.72 s | 7.6e+03 | 91%	6.83 s | 12.80 s | 7.6e+03 | 92%	6.06 s | 26.02 s | 7.7e+03 | 93%	47.49 s | 5.82 s | 7.7e+03 | 93%
ILL_D4	6.17 s | 99.01 s | 1.5e+06 | 103%	53.30 s | 25.72 s | 1.5e+06 | 103%	4.78 s | 252.67 s | 1.5e+06 | 103%	6.10 s | 97.20 s | 1.5e+06 | 103%	5.69 s | 259.10 s | 1.5e+06 | 103%	49.08 s | 37.23 s | 1.5e+06 | 103%
ILL_H10_IN8	6.57 s | 20.89 s | 9.2e+08 | 99%	54.47 s | 18.64 s | 9.2e+08 | 99%	5.00 s | 46.96 s | 9.2e+08 | 99%	6.58 s | 20.73 s | 9.2e+08 | 99%	6.34 s | 33.92 s | 9.2e+08 | 98%	49.42 s | 17.28 s | 9.2e+08 | 98%
ILL_H113	9.25 s | 69.69 s | 2.4e+10 | 101%	69.71 s | 13.10 s | 2.4e+10 | 101%	7.00 s | 126.95 s | 2.4e+10 | 101%	9.25 s | 69.07 s | 2.4e+10 | 101%	9.62 s | 225.90 s | 2.4e+10 | 101%	68.68 s | 9.91 s | 2.4e+10 | 101%
ILL_H13_IN20	6.57 s | 23.41 s | 4.6e+08 | 96%	54.29 s | 19.85 s | 4.6e+08 | 96%	5.23 s | 52.30 s | 4.6e+08 | 96%	6.56 s | 23.30 s | 4.6e+08 | 96%	6.17 s | 32.76 s | 4.6e+08 | 96%	49.60 s | 19.95 s | 4.6e+08 | 96%
ILL_H142	10.22 s | 54.59 s | 9.1e+09 | 100%	77.74 s | 14.75 s | 9.1e+09 | 100%	7.82 s | 103.91 s | 9.1e+09 | 100%	10.24 s | 54.67 s | 9.1e+09 | 100%	11.18 s | 153.67 s | 9.1e+09 | 100%	77.94 s | 11.39 s | 9.1e+09 | 100%
ILL_H142_D33	4.97 s	34.11 s	3.84 s	4.94 s	4.40 s	28.53 s
ILL_H142_IN12	12.74 s | 16.47 s | 8.7e+06 | 103%	104.50 s | 20.81 s | 8.7e+06 | 103%	10.07 s | 35.63 s | 8.9e+06 | 104%	12.79 s | 16.44 s | 8.7e+06 | 103%	14.19 s | 44.52 s | 8.7e+06 | 103%	104.50 s | 20.23 s | 8.7e+06 | 103%
ILL_H143_LADI	9.16 s	68.18 s	6.88 s	9.18 s	9.31 s	66.91 s
ILL_H15	7.86 s | 45.23 s | 1.2e+10 | 101%	60.06 s | 12.01 s | 1.2e+10 | 101%	5.91 s | 88.88 s | 1.2e+10 | 100%	7.87 s | 44.01 s | 1.2e+10 | 101%	8.01 s | 144.82 s | 1.2e+10 | 100%	57.21 s | 7.12 s | 1.2e+10 | 100%
ILL_H15_D11	9.03 s | 24.17 s | 2.2e+07 | 104%	68.16 s | 12.37 s | 1.8e+07 | 86%	6.88 s | 44.27 s | 2.2e+07 | 106%	9.04 s | 23.11 s | 2.2e+07 | 104%	9.18 s | 61.37 s | 2.2e+07 | 105%	65.44 s | 8.16 s | 1.8e+07 | 86%
ILL_H15_IN6	10.89 s	80.16 s	8.35 s	10.87 s	11.05 s	74.68 s
ILL_H16	5.89 s | 16.21 s | 1.9e+10 | 100%	62.03 s | 9.87 s | 1.9e+10 | 100%	4.55 s | 37.87 s | 1.9e+10 | 100%	5.90 s | 16.25 s | 1.9e+10 | 100%	5.44 s | 47.75 s | 1.9e+10 | 100%	59.11 s | 4.67 s | 1.9e+10 | 100%
ILL_H16_IN5	9.48 s | 8.08 s | 3.3e+07 | 98%	67.23 s | 9.74 s | 3.3e+07 | 98%	7.41 s | 17.28 s | 3.3e+07 | 98%	9.47 s | 8.04 s | 3.3e+07 | 98%	9.45 s | 21.72 s | 3.3e+07 | 98%	66.97 s | 4.45 s | 3.3e+07 | 98%
ILL_H16_IN5_2	| 8.02 s | 1.2e+06 | 106%	| 9.79 s | 1.2e+06 | 106%	| 17.24 s | 1.2e+06 | 108%	| 8.14 s | 1.2e+06 | 106%	| 23.00 s | 1.2e+06 | 107%	| 4.48 s | 1.2e+06 | 107%
ILL_H16_IN5_Mantid	9.43 s | 9.86 s | 3.3e+07 | 98%	68.45 s | 11.80 s | 3.3e+07 | 98%	7.37 s | 18.81 s | 3.3e+07 | 98%	9.53 s | 9.81 s | 3.3e+07 | 98%	9.39 s | 24.80 s | 3.3e+07 | 98%	67.06 s | 6.18 s | 3.3e+07 | 98%
ILL_H16_IN5_Mantid_2	| 9.85 s | 1.2e+06 | 108%	| 11.88 s | 1.2e+06 | 108%	| 18.77 s | 1.2e+06 | 108%	| 9.79 s | 1.2e+06 | 108%	| 21.49 s | -1 | 0%	| 6.12 s | 1.2e+06 | 107%
ILL_H16_Mantid	5.89 s	62.47 s	4.68 s	5.86 s	5.70 s	59.21 s
ILL_H22	9.24 s | 14.39 s | 2.7e+09 | 99%	67.07 s | 12.14 s | 2.7e+09 | 99%	7.19 s | 25.22 s | 2.7e+09 | 99%	9.27 s | 14.20 s | 2.7e+09 | 99%	9.87 s | 31.03 s | 2.7e+09 | 99%	65.82 s | 7.93 s | 2.7e+09 | 99%
ILL_H22_D1A	11.16 s | 36.17 s | 4.2e+05 | 108%	90.67 s | 19.08 s | 4.2e+05 | 108%	8.38 s | 85.91 s | 4.2e+05 | 108%	11.08 s | 36.39 s | 4.2e+05 | 108%	11.44 s | 83.49 s | 4.2e+05 | 108%	93.78 s | 22.25 s | 4.2e+05 | 108%

McStas 3.2 release is expected in the fall of 2022

GPU-computing “101”

- LOTS of slowish processor-cores
  (aka. massively parallel”)
- BUT: Limited bandwidth in CPU / GPU exchange
  (PCIe…)
- Main HW providers today are NVIDIA , AMD , 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**
    - **“High-level, compiler-driven CUDA”**
  - 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… ;-)

Why consider GPU’s for McStas in the first place?

- GPU’s have great potential for speedups within “intrinsically parallel” problems
- McStas is already “embarrassingly parallel”: every neutron is independent
- TCO or “FLOP / energy” can be greener / cheeper than for CPU’s

Why OpenACC?

- Was retrofitted elegantly to our “old” sw framework”
- **Identical code** runs now on both **CPU** and **GPU**

Foreseen use-case?

- Pre- or in-experiment help tool needs to be “as fast as the experiment”. (Modern day spallation sources with event-mode and sample sim. are a challenge.)

NVIDIA

GPU support in McStas, so far is effectively NVIDIA-specific!

OpenACC

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

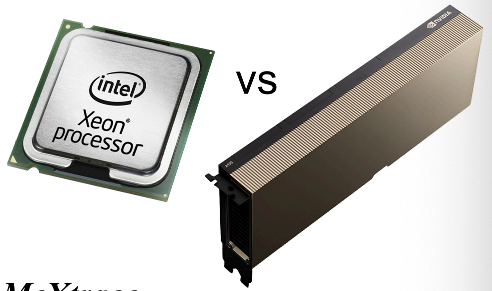
No printfs etc. available on GPU, automatically suppressed by #defines

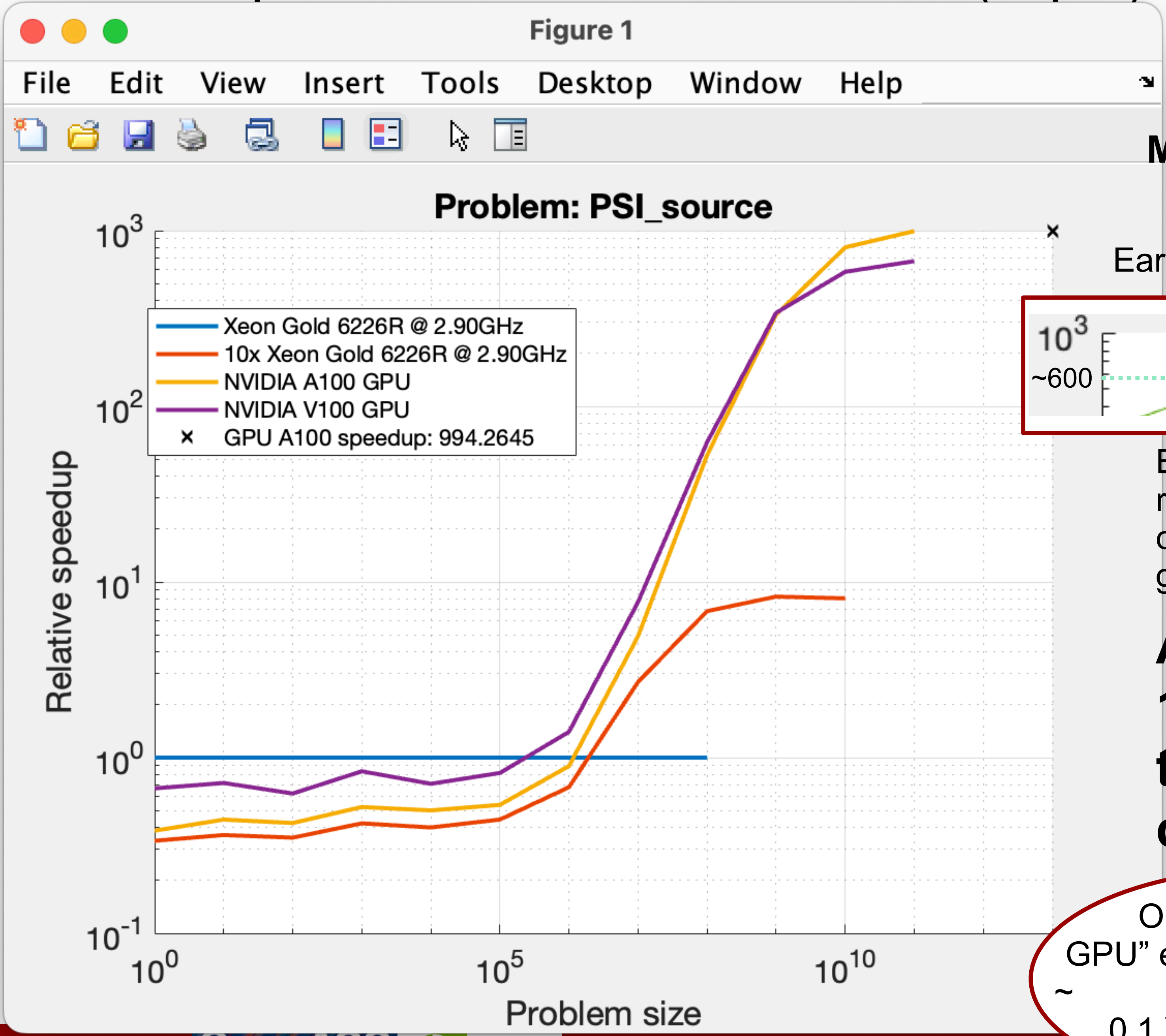
The GPU user experience is almost identical to that of running on CPU!

Maximum performance indication on NVIDIA A100 (Ampere)

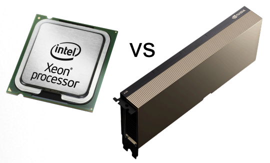
Idealised instrument with source and monitor only - i.e. without any use of the ABSORB macro.

(Good indication of maximal speedup achievable.)



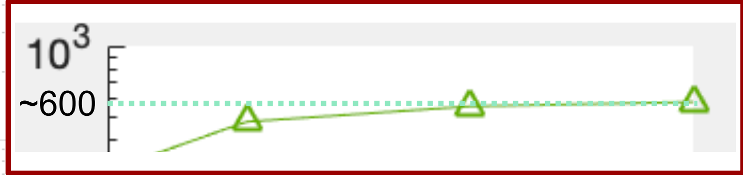


—	Xeon Gold 6226R @ 2.90GHz
—	10x Xeon Gold 6226R @ 2.90GHz
—	NVIDIA A100 GPU
—	NVIDIA V100 GPU
×	GPU A100 speedup: 994.2645



Maximal speedup: ~1000

Earlier dataset from V100 ~600



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

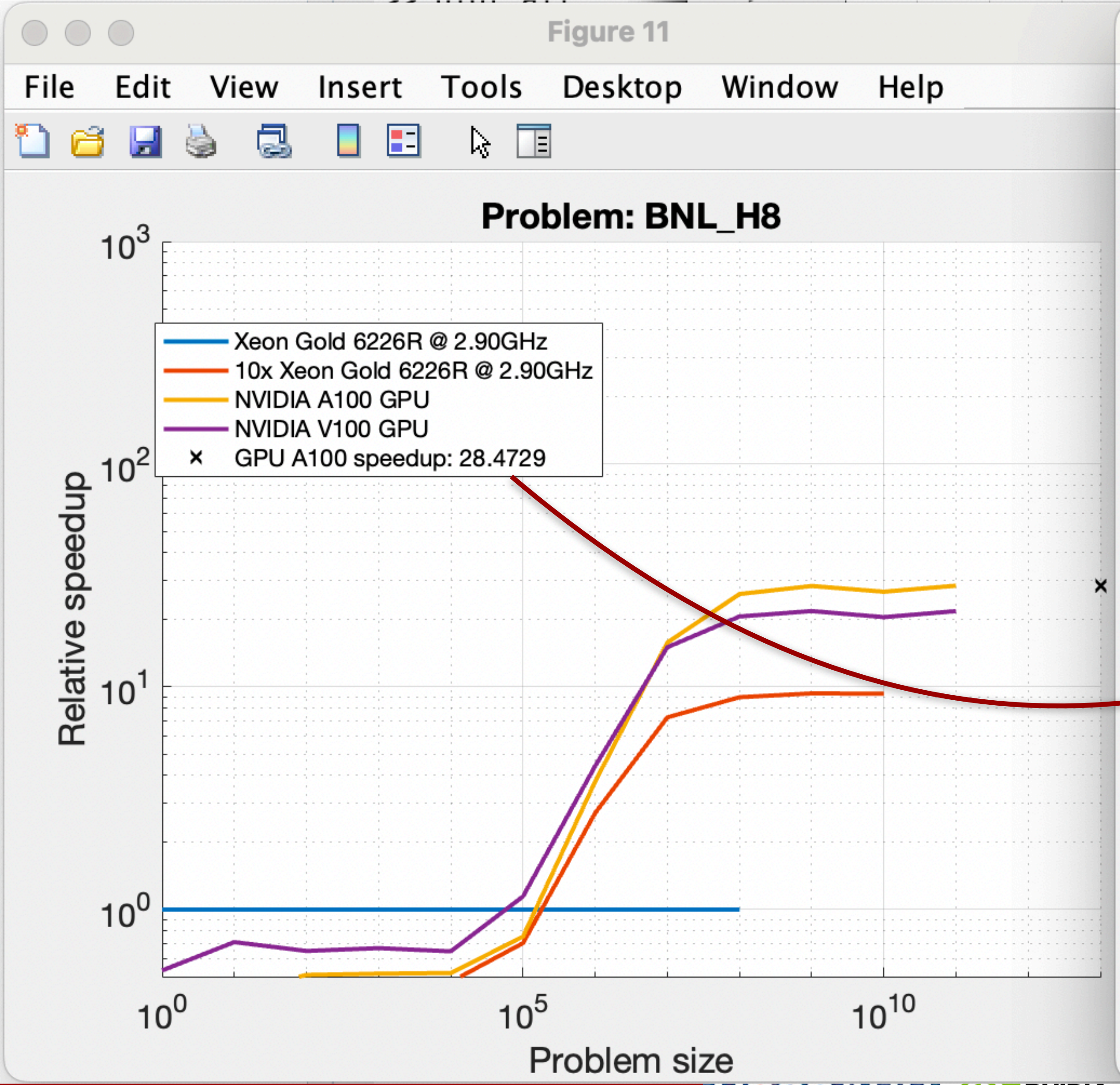
A100 run is ~ 1000 times faster than a single-core CPU run

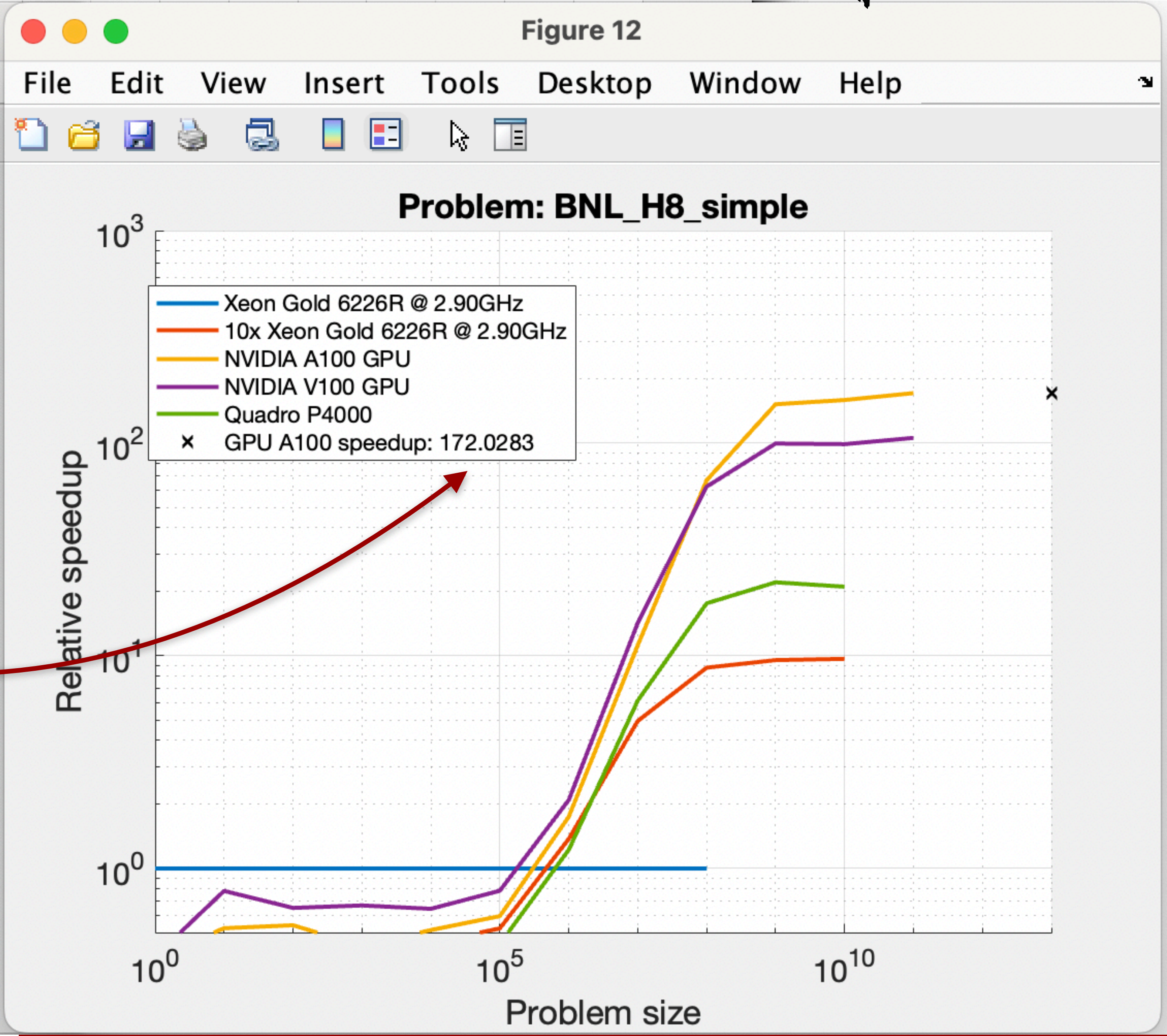
Older "Gamer-GPU" e.g. GeForce 1030 is ~ 0.1 V100 or 0.05 A100

Real-world problem 1: BNL_H8 TAS models with / without SPLIT: factor of **28-172** speedup

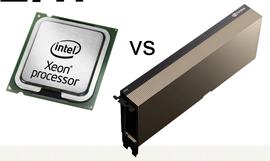
(SPLITs makes the instrument less ‘intrinsically’ parallel)

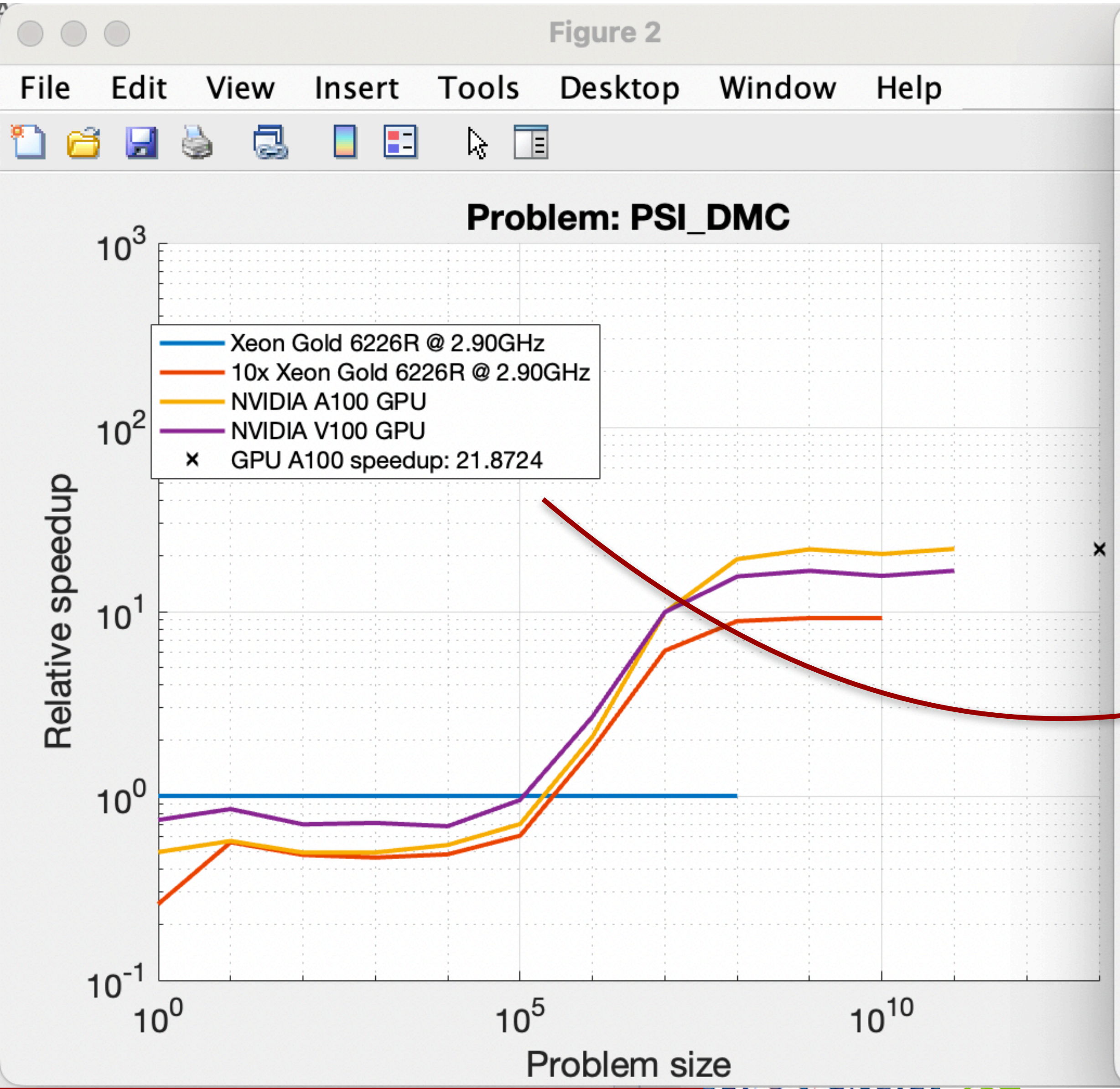
vs

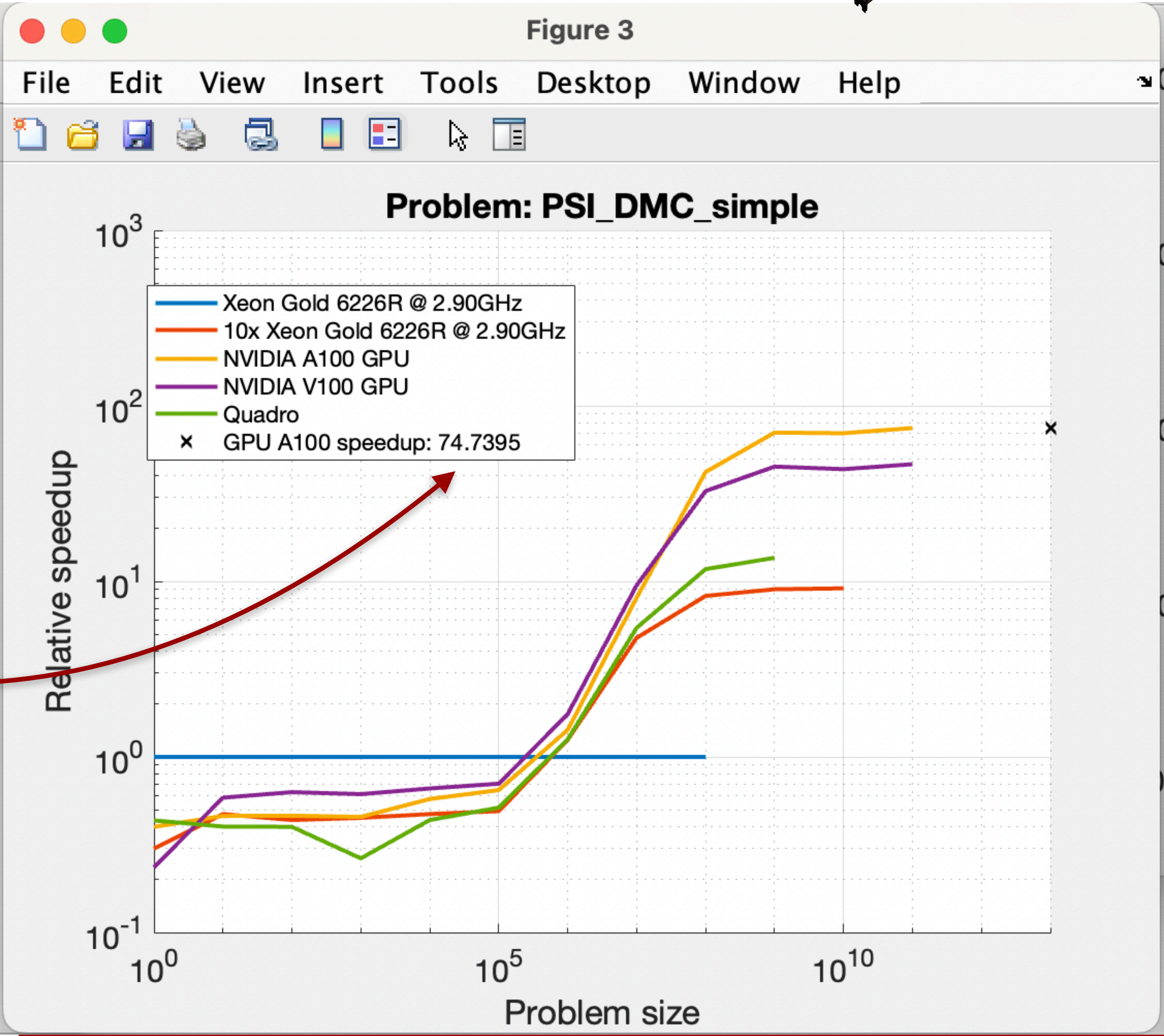




Real-world problem 2: PSI_DMC diff models with / without SPLIT: factor of **21-74** speedup

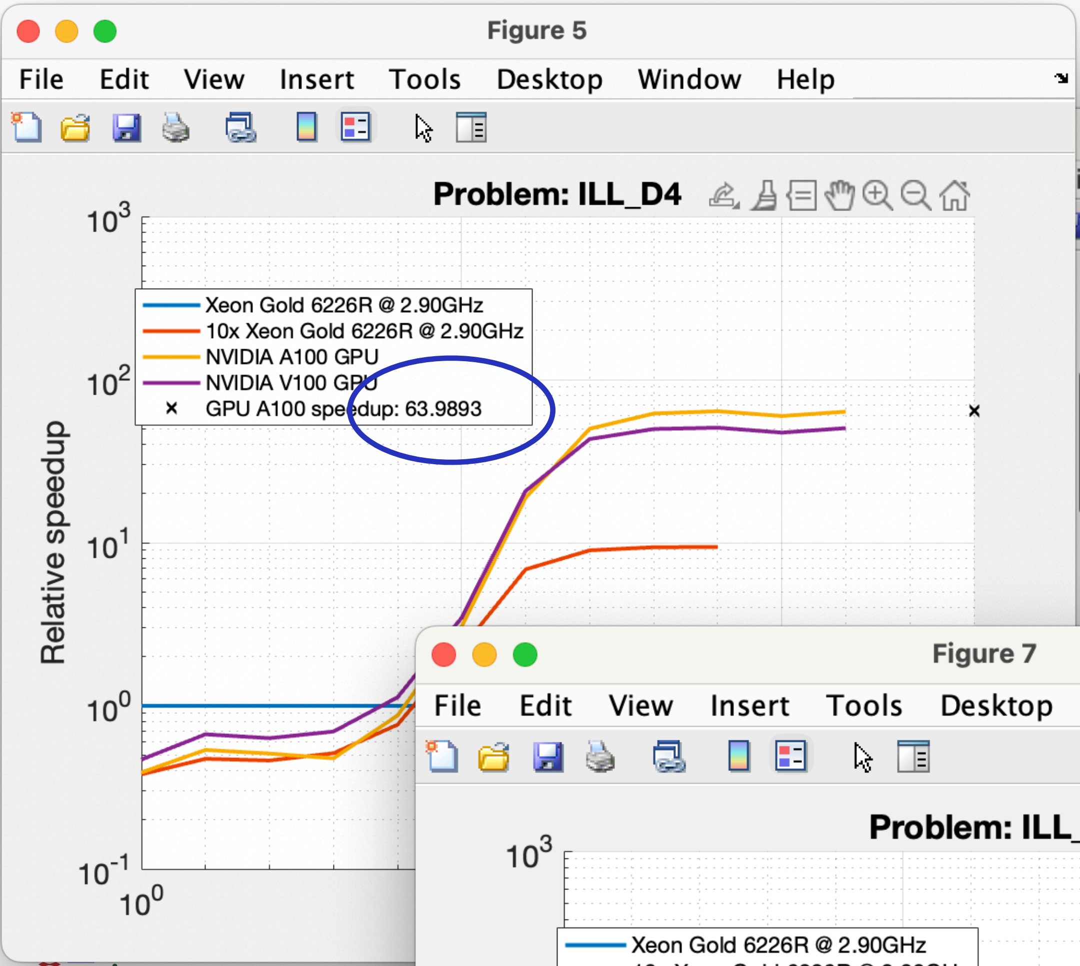


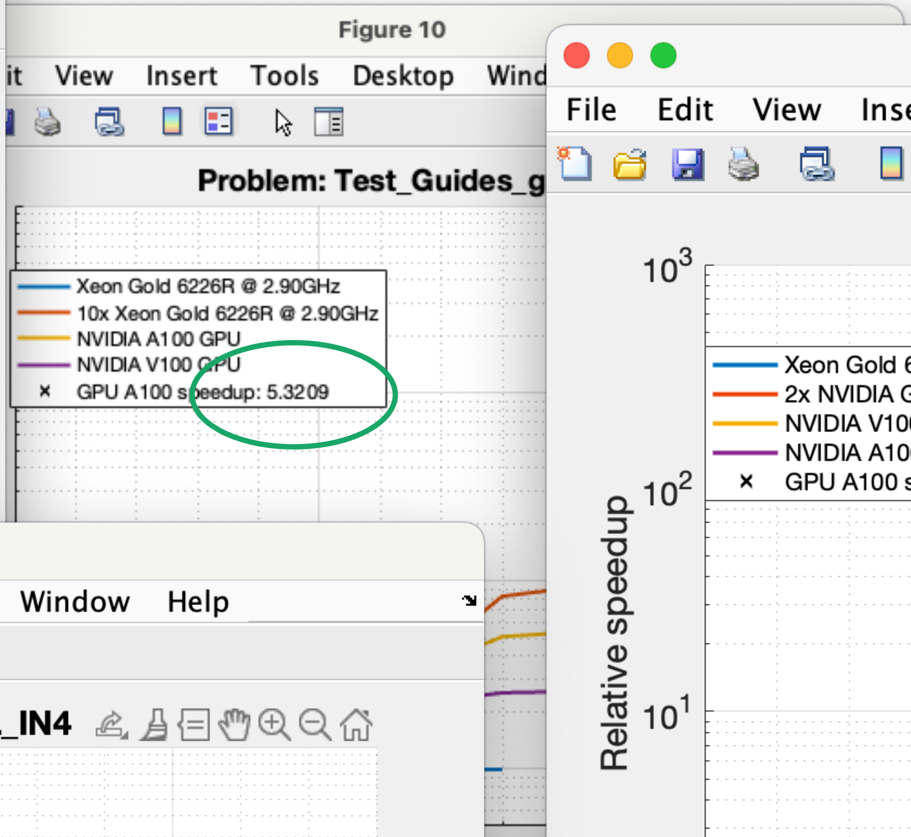


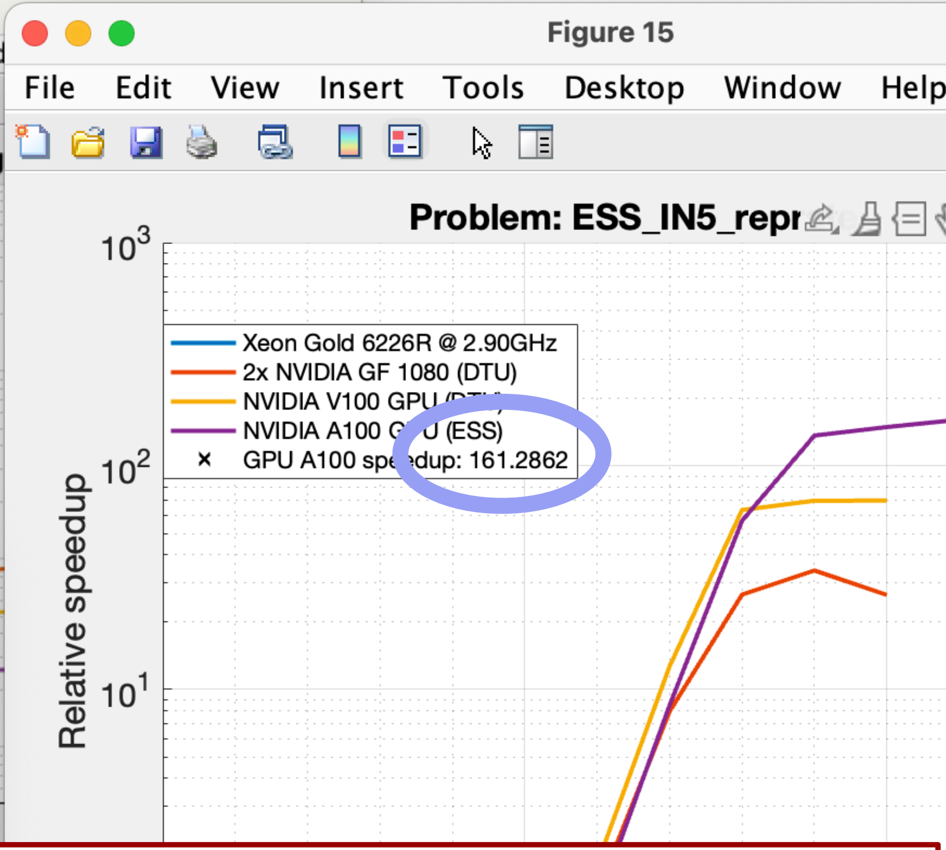


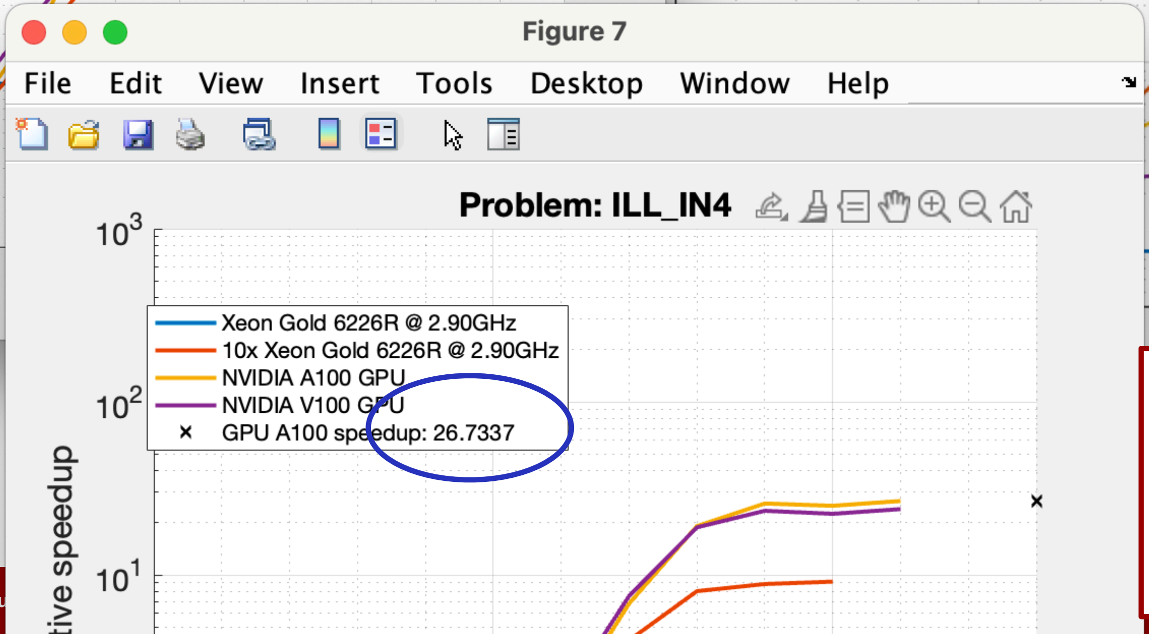
Real-world problems 3:

"sample-only" sims often ~ factor of 50
"full instruments" ~ factor of 20-30-60, but **160 seen**
"optics-only" surprisingly ~ factor of 5 ??









Main message:
Here is room for more, optimisation ongoing!!!
* handle splits better
* investigate "key" components
* structural code-changes (comp USERVARS etc)

First user applications of McStas on GPU are coming in:

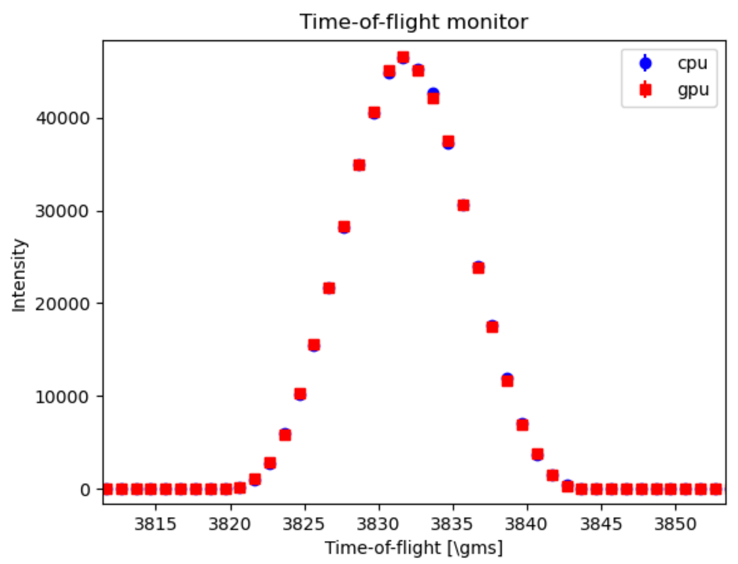
Incident Beam Simulations with GPUS (to be presented in full at JCNS workshop)

Team: Garrett E. Granroth, Fahima Islam, Thomas Huegle, Jiao Lin, Peter Willendrup

- Overall project goal: Deliver realistic instrument simulations to users on the same time scale as a neutron scattering measurement.
- Currently the most time is spent simulating up to the sample (The Incident beam)
- Beam line simulations for SNAP, GPSANS, and ARCS have been updated for McStas 3.X. (They can now run on NVIDIA gpus as well as cpus.)
- The GPSANS update was straight forward as it leveraged updated McStas components
- The SNAP, SEQUOIA and ARCS simulations have custom components so they were more effort.
- The openacc implementation in McStas 3.X streamlined the development process.
- Simulations run more than 100x faster on an NVIDIA A100 than a cpu.
- With all 8 gpus on an NVIDIA A100, simulations can run much faster than the duration of a single neutron scattering measurement

Details of Simulation Speed increase

Instrument	Time on 1 CPU (s)	Time on 1 GPU (s)	Speed up (x)
ARCS	103920	58	1791
GPSANS	3380	16	211
SNAP	20592	60	343

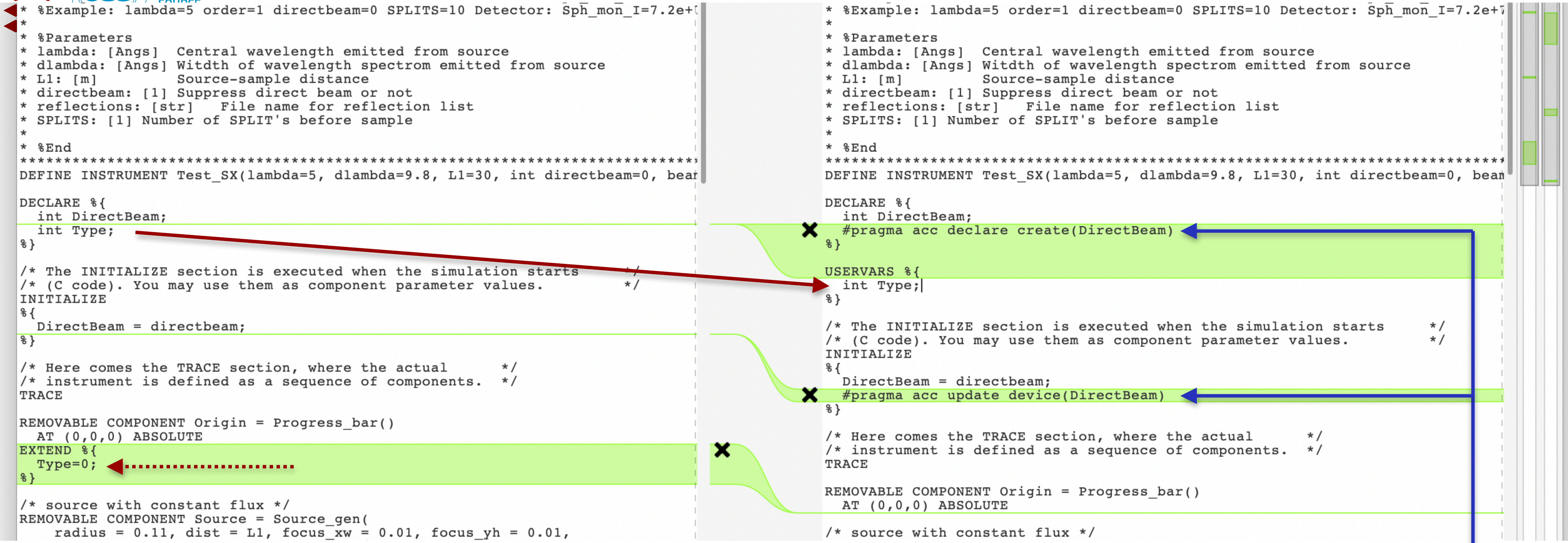


Beam monitor from ARCS simulation shows no statistically significant difference between cpu and gpu simulations

Slide from Garrett Granroth, ORNL/SNS

Porting an instrument - HOWTO

see also https://github.com/McStasMcXtrace/McCode/wiki/HOWTO%3A-Modifying-a-McStas-2-instrument-for-use-under-McStas-3



```
* %Example: lambda=5 order=1 directbeam=0 SPLITS=10 Detector: Sph_mon_I=7.2e+
*
* %Parameters
* lambda: [Angs]  Central wavelength emitted from source
* dlambda: [Angs] Witdth of wavelength spectrom emitted from source
* L1: [m]         Source-sample distance
* directbeam: [1] Suppress direct beam or not
* reflections: [str]   File name for reflection list
* SPLITS: [1] Number of SPLIT's before sample
*
* %End
*******************************************************************************
DEFINE INSTRUMENT Test_SX(lambda=5, dlambda=9.8, L1=30, int directbeam=0, bear

DECLARE %{
  int DirectBeam;
  int Type;
%}

/* The INITIALIZE section is executed when the simulation starts     */
/* (C code). You may use them as component parameter values.         */
INITIALIZE
%{
  DirectBeam = directbeam;
%}

/* Here comes the TRACE section, where the actual      */
/* instrument is defined as a sequence of components.  */
TRACE

REMOVABLE COMPONENT Origin = Progress_bar()
  AT (0,0,0) ABSOLUTE
EXTEND %{
  Type=0;
%}

/* source with constant flux */
REMOVABLE COMPONENT Source = Source_gen(
    radius = 0.11, dist = L1, focus_xw = 0.01, focus_yh = 0.01,
```

```
* %Example: lambda=5 order=1 directbeam=0 SPLITS=10 Detector: Sph_mon_I=7.2e+7
*
* %Parameters
* lambda: [Angs]  Central wavelength emitted from source
* dlambda: [Angs] Witdth of wavelength spectrom emitted from source
* L1: [m]         Source-sample distance
* directbeam: [1] Suppress direct beam or not
* reflections: [str]   File name for reflection list
* SPLITS: [1] Number of SPLIT's before sample
*
* %End
*******************************************************************************
DEFINE INSTRUMENT Test_SX(lambda=5, dlambda=9.8, L1=30, int directbeam=0, bear

DECLARE %{
  int DirectBeam;
  #pragma acc declare create(DirectBeam)
%}

USERVARS %{
  int Type;
%}

/* The INITIALIZE section is executed when the simulation starts     */
/* (C code). You may use them as component parameter values.         */
INITIALIZE
%{
  DirectBeam = directbeam;
  #pragma acc update device(DirectBeam)
%}

/* Here comes the TRACE section, where the actual      */
/* instrument is defined as a sequence of components.  */
TRACE

REMOVABLE COMPONENT Origin = Progress_bar()
  AT (0,0,0) ABSOLUTE

/* source with constant flux */
```

* Particle “flags” must go in USERVARS (automatically initialised to 0 in 3.x)

* DECLARE-vars used within instrument TRACE need #pragma’s

* Monitor_nD user-variables-identifiers are string type   user1=flag $\longrightarrow$ user1=“flag”

If you can’t make it work, please write mcstas-users@mcstas.org or define a GitHub issue

Porting a component - HOWTO

see also https://github.com/McStasMcXtrace/McCode/wiki/HOWTO%3A-Modifying-a-McStas-2-component-for-use-under-McStas-3

1. DEFINITION PARAMETERS is not supported. Vars must become SETTING PARAMETERS, specifically:
   - Simply move string vars
   - Lists/array-pointers need the vector type e.g. vector a={1,2,3,4} or b=c where c is an instrument-level array/pointers. (Base type in vector is a double.)
2. DECLARE must have "simple" content, i.e. vars without initialisation each alone on a line:

```
DECLARE %{
  double a,b;
%}
```

$\longrightarrow$

```
DECLARE %{
  double a;
  double b;
%}
```

   - and initialise them in INITIALIZE

3. SHARE-based TRACE-functions that pick random numbers MUST include the "particle" in the footprint:
   - double my_function(double a, int b, double*c, _class_particle* _particle);
   - this forwards the RNG state (carried with each particle)
4. DECLARE-parameters should not be used to store particle-derived information...
   - Use a local TRACE-scope var instead
   - Ensure this by checking that CPU and GPU runs are identical if run with the same seed.
5. If you are using external libs, e.g. GSL or function pointers, your code can not run on GPU.
   - You may put the NOACC keyword in the component header, this forces execution on CPU only.

If you can't make it work, please write mcstas-users@mcstas.org or define a GitHub issue

All instruments (>250) distributed with McStas 3.x can utilise NVIDIA GPUs.
- please use as inspiration!!

1. You need an NVIDIA card in your machine
2. Use Linux ;-) (or WSL 2 on windows, including relevant driver and kernel…)
3. Install the NVIDIA hpc sdk https://developer.nvidia.com/nvidia-hpc-sdk-downloads
4. Your McStas 3.x is preconfigured with reasonable defaults if the nvc compiler is on the PATH, i.e.

A. Single-core CPU compilation by
```
mcrun -c Instrument.instr
```

B. Enable MPI by
```
mcrun -c --mpi=8 Instrument.instr
```

C. Enable GPU by
```
mcrun -c --openacc Instrument.instr
```

D. Combined MPI and GPU run can be achieved via
```
mcrun -c --mpi=8 --openacc Instrument.instr
```

… Or use similar settings from your mcgui.

"Do almost as usual".

Output data should look "as usual", an instrument compiled for GPU can (currently) not output mcdisplay graphics.

If you can't make it work, please write mcstas-users@mcstas.org or define a GitHub issue

Conclusions

- **It really does work nicely!**
- **Code changes** much **less invasive** than envisioned!
- Use is transparent, fully integrated in mcgui / mcrun utils (on Linux or through WSL)
- It often gives a speedup of **1-2 orders** of magnitude over 1 cpu core
- Used for pre / in experiment simulations at ORNL, runs at least “real-time” wrt. experiments
- Most things work already
  (we have workarounds or solutions in the pipe for the rest)
- McStas 3.1 is as of yet “fully ported” to GPU but **not fully “optimised” performance-wise**, we will try to go to another Hackathon
- Basic **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 :-)

Vishal Metha

Christian Hundt

Alexey Romanenko

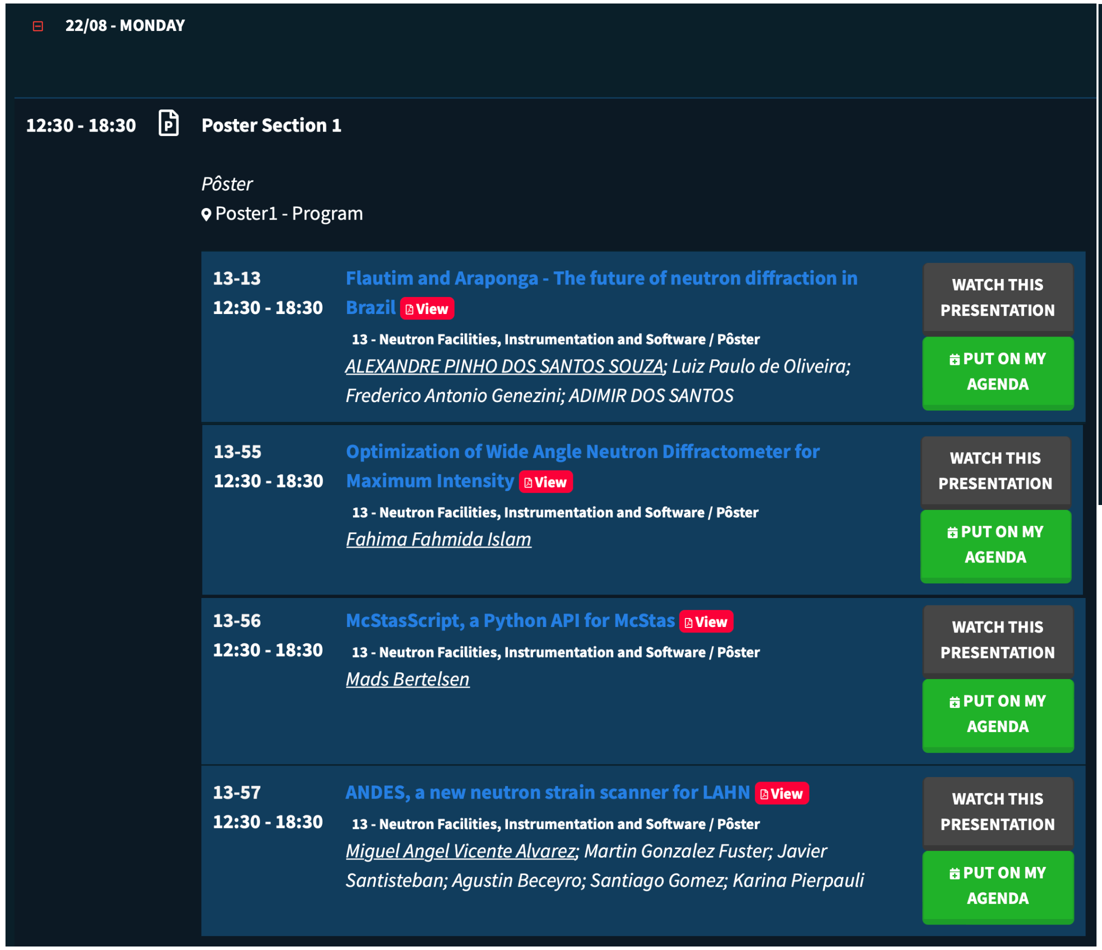
Guido Juckeland

Sebastian von Alfthan

Datacenters:

- HZDR Dresden
- CSC Espoo
- DTU Lyngby
- DTU RISØ
- ESS DMSC

— Other McStas contributions —



22/08 - MONDAY

12:30 - 18:30 **Poster Section 1**

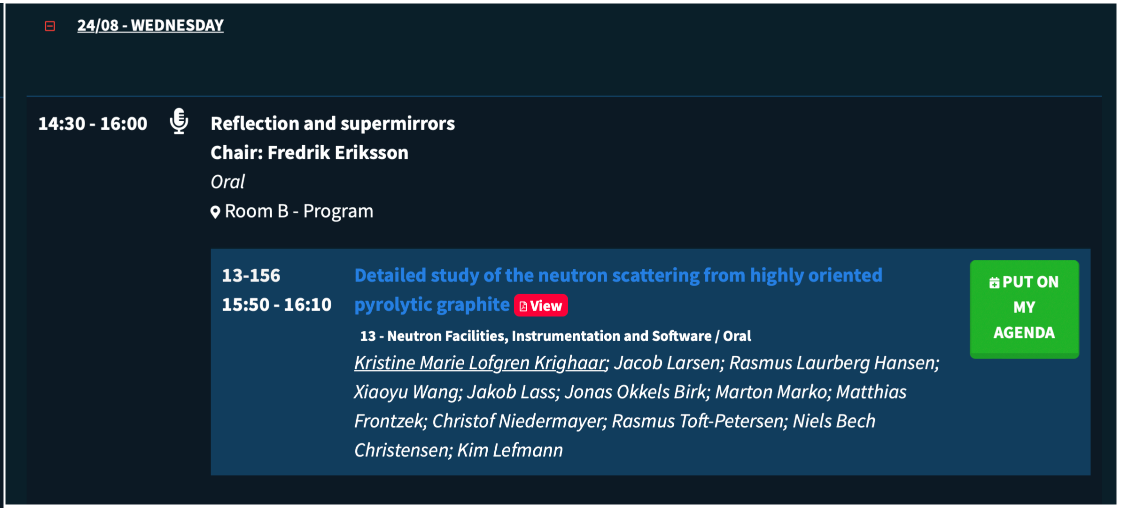
Pôster
Poster1 - Program

13-13
12:30 - 18:30
Flautim and Araponga - The future of neutron diffraction in Brazil View
13 - Neutron Facilities, Instrumentation and Software / Pôster
ALEXANDRE PINHO DOS SANTOS SOUZA; Luiz Paulo de Oliveira; Frederico Antonio Genezini; ADIMIR DOS SANTOS
WATCH THIS PRESENTATION
PUT ON MY AGENDA

13-55
12:30 - 18:30
Optimization of Wide Angle Neutron Diffractometer for Maximum Intensity View
13 - Neutron Facilities, Instrumentation and Software / Pôster
Fahima Fahmida Islam
WATCH THIS PRESENTATION
PUT ON MY AGENDA

13-56
12:30 - 18:30
McStasScript, a Python API for McStas View
13 - Neutron Facilities, Instrumentation and Software / Pôster
Mads Bertelsen
WATCH THIS PRESENTATION
PUT ON MY AGENDA

13-57
12:30 - 18:30
ANDES, a new neutron strain scanner for LAHN View
13 - Neutron Facilities, Instrumentation and Software / Pôster
Miguel Angel Vicente Alvarez; Martin Gonzalez Fuster; Javier Santisteban; Agustin Beceyro; Santiago Gomez; Karina Pierpauli
WATCH THIS PRESENTATION
PUT ON MY AGENDA



24/08 - WEDNESDAY

14:30 - 16:00 **Reflection and supermirrors**
Chair: Fredrik Eriksson
Oral
Room B - Program

13-156
15:50 - 16:10
Detailed study of the neutron scattering from highly oriented pyrolytic graphite View
13 - Neutron Facilities, Instrumentation and Software / Oral
Kristine Marie Lofgren Krighaar; Jacob Larsen; Rasmus Laurberg Hansen; Xiaoyu Wang; Jakob Lass; Jonas Okkels Birk; Marton Marko; Matthias Frontzek; Christof Niedermayer; Rasmus Toft-Petersen; Niels Bech Christensen; Kim Lefmann
PUT ON MY AGENDA

— Backup slides follow —

McStas 2.x -> McStas 3.x main differences

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

Advantage of 3.0 also on CPU

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

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

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

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)`

Things that couldn't be done

- **Function pointers / abstract functions are not available on GPU**
  - Solutions:
    - Code around if possible (e.g. integration routine pr. specific function to be integrated…)
    - Use mechanism to do this calculation CPU-side before/after/at cost of transfers

- **Variadic functions are not available on GPU**
    - Special case: printf() and friends

- **Anonymous structs as comp pars are not available on GPU**
  - Declare struct explicitly

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