Peter Willendrup and Jakob Garde DTU Physics

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

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
  - changes to engine API for component developers
  - definition parameters no longer supported in components (strings, arrays etc. as replacement)
  - much easier to hack/experiment directly on the generated code
  - new USERVARS feature which enriches the particle struct to enable “per particle” flags
  - 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
- Complete change to dynamic monitor-arrays
- Various tricks relating to GPU

Main events on timeline of current developments

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.

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

NVIDIA mentor: Vishal Metha

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

NVIDIA mentor: Christian Hundt

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

McStas heading for the GPU… March 2018

McCode on GPU?

1st prototype, “null”-instrument with only one component.

Based on NVIDIA compiler technology,

PGCC and OpenACC

pragmas

More to come in 2019!

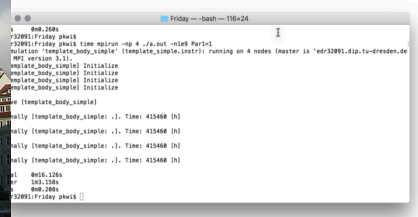
McStas / McXtrace instrument

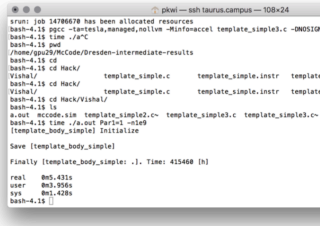
Port heavy part of component to GPU (Shows some potential)

Port neutron loop to GPU
- Lots of code
- Code generation
- Compiles! Speed up today

CPU MPI 4 cores (95 % usage)

GPU 5% usage





16.12 s (Single core 56.0 s)

5.43 s

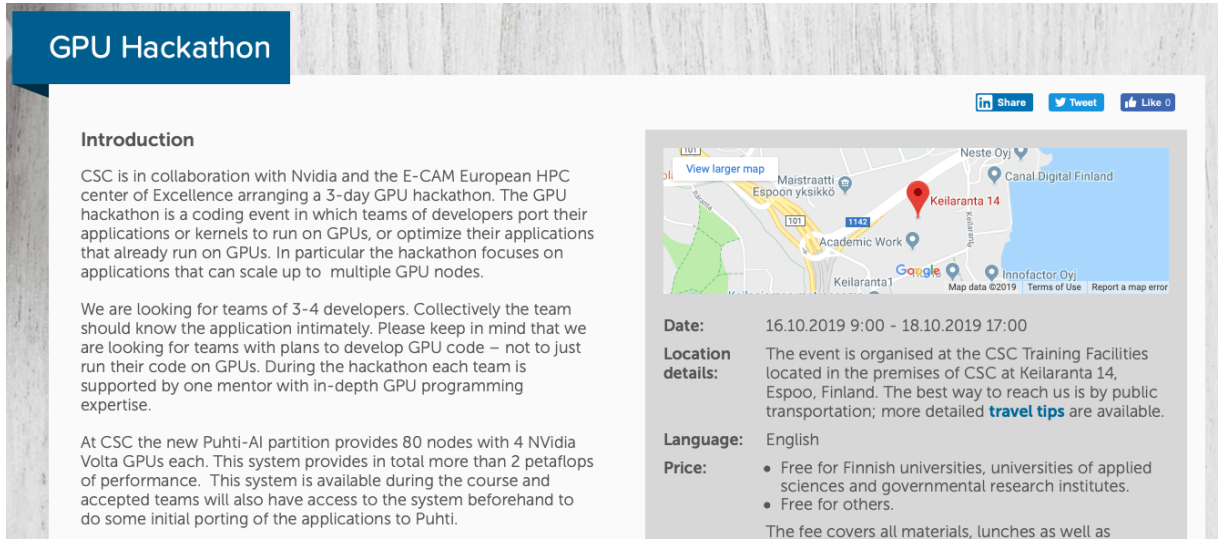
McStas heading for the GPU… October 2019

Rewritten code-generation with automated additions of OpenACC pragmas.

Quite transparent wrt. CPU vs. GPU

First simulations with meaningful output

Speed on DELL with Quadro-card ~ on par with running on CPU with MPI



GPU Hackathon

Introduction

CSC is in collaboration with Nvidia and the E-CAM European HPC center of Excellence arranging a 3-day GPU hackathon. The GPU hackathon is a coding event in which teams of developers port their applications or kernels to run on GPUs, or optimize their applications that already run on GPUs. In particular the hackathon focuses on applications that can scale up to multiple GPU nodes.

We are looking for teams of 3-4 developers. Collectively the team should know the application intimately. Please keep in mind that we are looking for teams with plans to develop GPU code – not to just run their code on GPUs. During the hackathon each team is supported by one mentor with in-depth GPU programming expertise.

At CSC the new Puhti-AI partition provides 80 nodes with 4 NVidia Volta GPUs each. This system provides in total more than 2 petaflops of performance. This system is available during the course and accepted teams will also have access to the system beforehand to do some initial porting of the applications to Puhti.

Share Tweet Like 0

View larger map

Maistraatti Espoon yksikkö Neste Oyj Canal Digital Finland Keilaranta 14 Academic Work Keilaranta1 Innofactor Oyj Google Map data ©2019 Terms of Use Report a map error

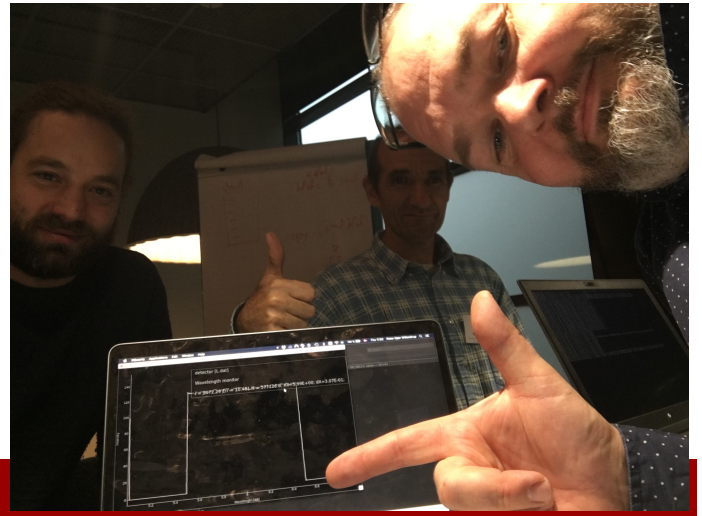
Date: 16.10.2019 9:00 - 18.10.2019 17:00

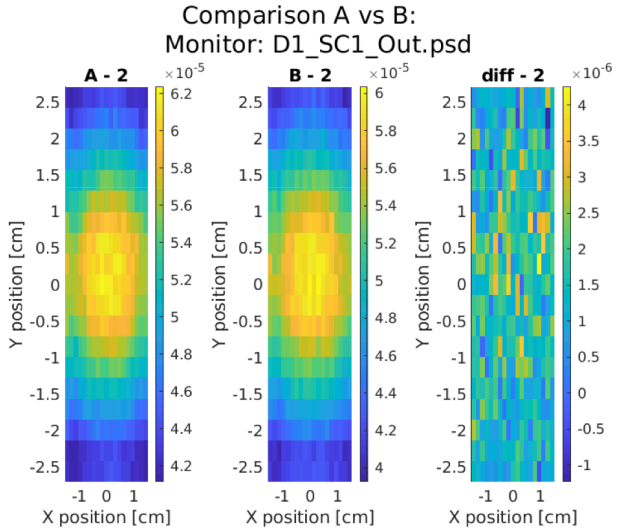
Location details: The event is organised at the CSC Training Facilities located in the premises of CSC at Keilaranta 14, Espoo, Finland. The best way to reach us is by public transportation; more detailed travel tips are available.

Language: English

Price:
- Free for Finnish universities, universities of applied sciences and governmental research institutes.
- Free for others.

The fee covers all materials, lunches as well as





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

example_v25.instr

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

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

INITIALIZE %{
  two_x_dummy=2*dummy;
%}

TRACE

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

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

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

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

END
```

example_v30.instr

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

DECLARE %{
  double constant;
  double two_x_dummy;
%}

USERVARS %{
  double flag;
%}

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

TRACE

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

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

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

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

END
```

example_v30.instr

```
/*
* %Example: example.instr
*/
DEFINE INSTRUMENT Minimal(

DECLARE %{
  double constant;
  double two_x_dummy;
%}

USERVARS %{
  double flag;
%}

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

All the stuff usually put in DECLARE earlier like

- double var=3.0;
- function declarations

are actually OK to here, but not so in the components.

For symmetry we suggest initialisation in initialise only.

Becomes part of the _particle struct, i.e. can change with each particle

Arm unchanged

[v2.5comps] Arm.comp — [v3.0comps] Arm.comp

Save Undo

[v2.5comps]...] Arm.comp

Arm.comp

Files are identical Hide

```
/*******************************************************************
*
* McStas, neutron ray-tracing package
*         Copyright 1997-2002, All rights reserved
*         Risoe National Laboratory, Roskilde, Denmark
*         Institut Laue Langevin, Grenoble, France
*
* Component: Arm
*
* %I
*
* Written by: Kim Lefmann and Kristian Nielsen
* Date: September 1997
* Version: $Revision$
* Release: McStas 1.6
* Origin: Risoe
*
* Arm/optical bench
*
* %D
* An arm does not actually do anything, it is just there to set
* up a new coordinate system.
*
* %P
* Input parameters:
*
* %E
*******************************************************************

DEFINE COMPONENT Arm
DEFINITION PARAMETERS ()
SETTING PARAMETERS ()
OUTPUT PARAMETERS ()
/* Neutron parameters: (x,y,z,vx,vy,vz,t,sx,sy,sz,p) */
TRACE
%{
%}

MCDISPLAY
%{
  /* A bit ugly; hard-coded dimensions. */

  line(0,0,0,0.2,0,0);
  line(0,0,0,0,0.2,0);
  line(0,0,0,0,0,0.2);
%}

END
```

Unicode (UTF-8) Ln 1, Col 1

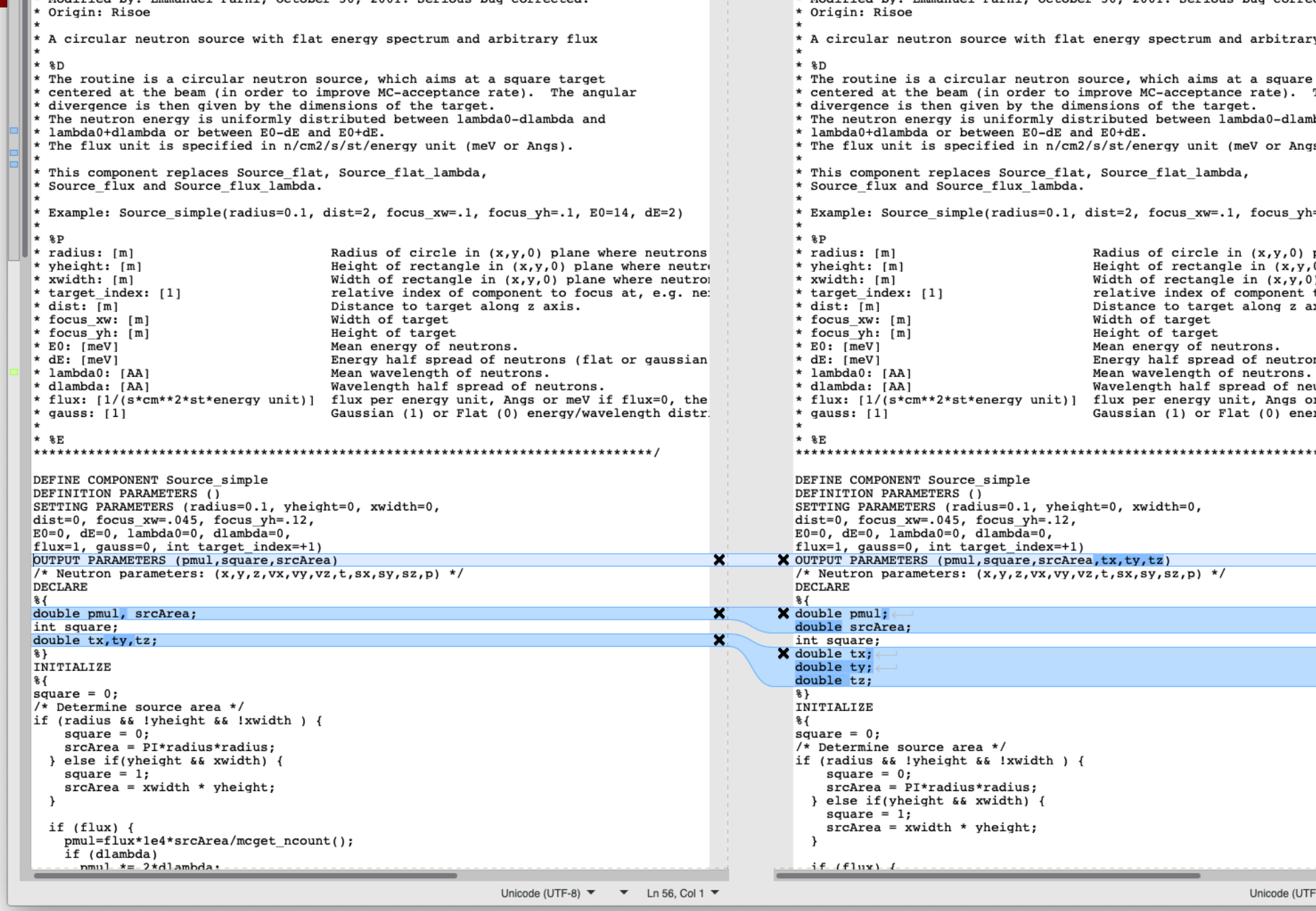
Arm.comp

Files are identical Hide

```
/*******************************************************************
*
* McStas, neutron ray-tracing package
*         Copyright 1997-2002, All rights reserved
*         Risoe National Laboratory, Roskilde, Denmark
*         Institut Laue Langevin, Grenoble, France
*
* Component: Arm
*
* %I
*
* Written by: Kim Lefmann and Kristian Nielsen
* Date: September 1997
* Version: $Revision$
* Release: McStas 1.6
* Origin: Risoe
*
* Arm/optical bench
*
* %D
* An arm does not actually do anything, it is just there to set
* up a new coordinate system.
*
* %P
* Input parameters:
*
* %E
*******************************************************************

DEFINE COMPONENT Arm
DEFINITION PARAMETERS ()
SETTING PARAMETERS ()
OUTPUT PARAMETERS ()
/* Neutron parameters: (x,y,z,vx,vy,vz,t,sx,sy,sz,p) */
TRACE
%{
%}

MCDISPLAY
%{
  /* A bit ugly; hard-coded dimensions. */

  line(0,0,0,0.2,0,0);
  line(0,0,0,0,0.2,0);
  line(0,0,0,0,0,0.2);
%}

END
```

Unicode (UTF-8) Ln 48, Col 4

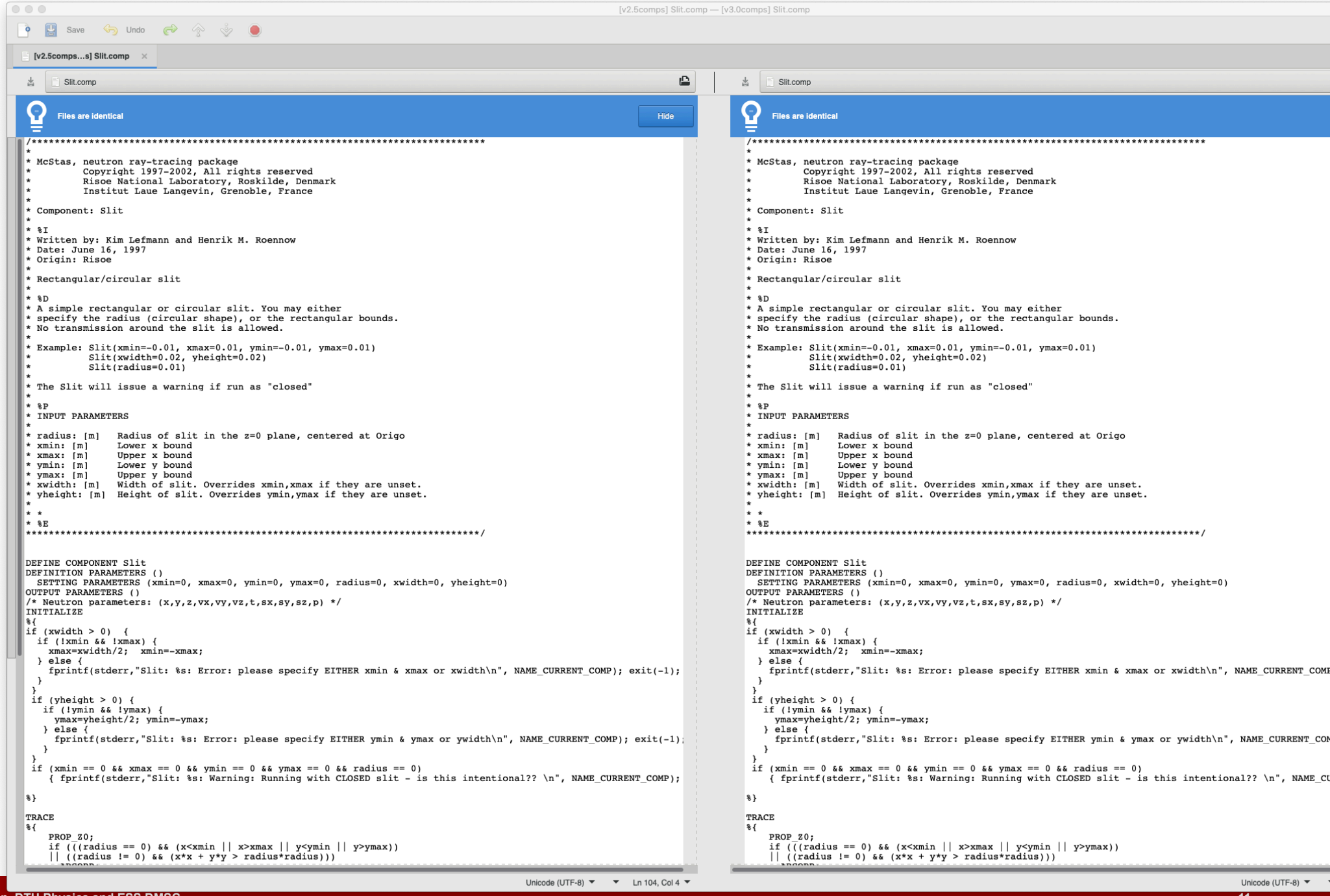
Source_simple minor changes



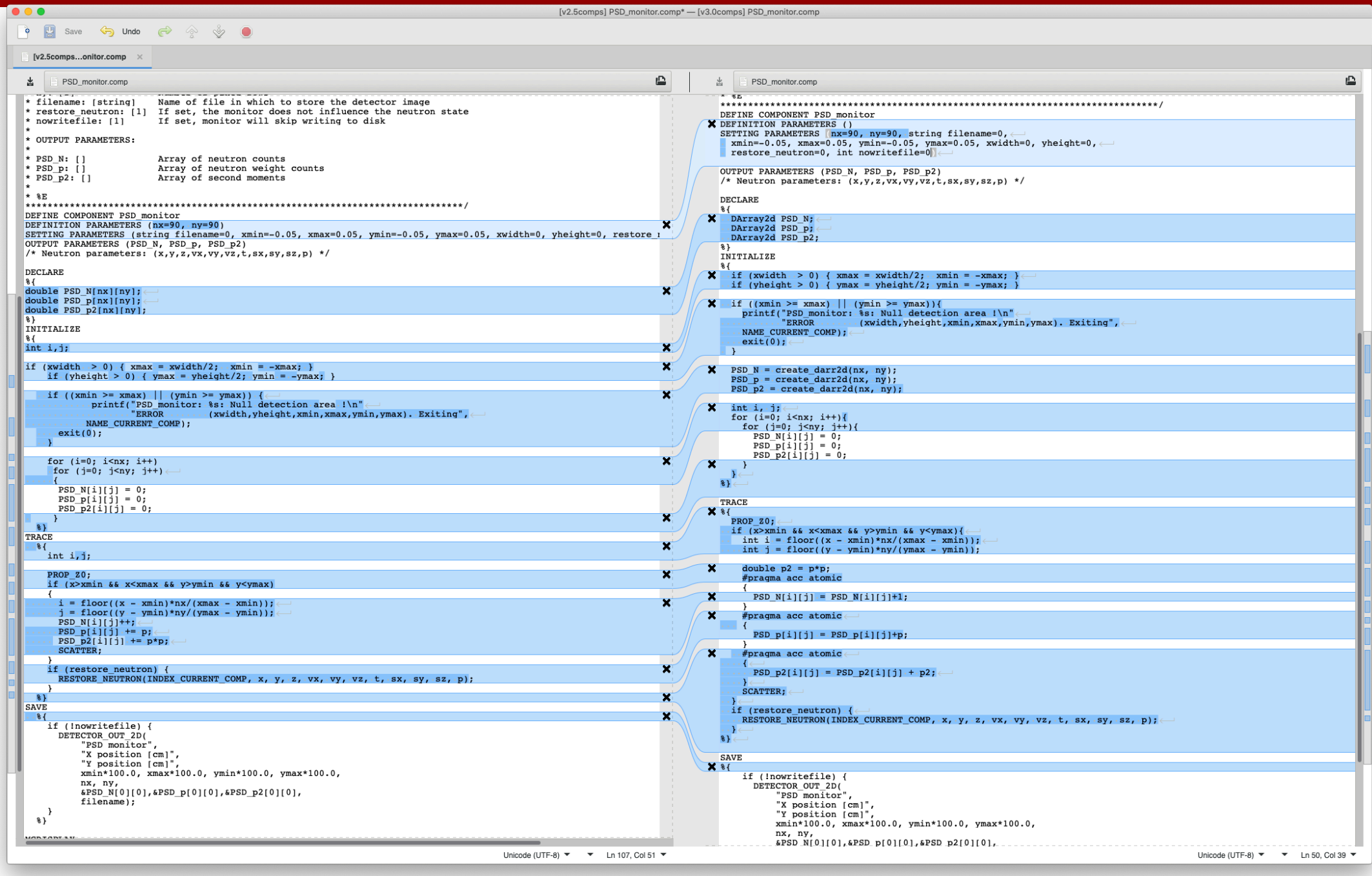
```
* Origin: Risoe
*
* A circular neutron source with flat energy spectrum and arbitrary flux
*
* %D
* The routine is a circular neutron source, which aims at a square target
* centered at the beam (in order to improve MC-acceptance rate).  The angular
* divergence is then given by the dimensions of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

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

  if (flux) {
    pmul=flux*1e4*srcArea/mcget_ncount();
    if (dlambda)
      pmul *= 2*dlambda;
```

```
* Origin: Risoe
*
* A circular neutron source with flat energy spectrum and arbitrary
*
* %D
* The routine is a circular neutron source, which aims at a square
* centered at the beam (in order to improve MC-acceptance rate).
* divergence is then given by the dimensions of the target.
* The neutron energy is uniformly distributed between lambda0-dlamb
* lambda0+dlambda or between E0-dE and E0+dE.
* The flux unit is specified in n/cm2/s/st/energy unit (meV or Angs
*
* This component replaces Source_flat, Source_flat_lambda,
* Source_flux and Source_flux_lambda.
*
* Example: Source_simple(radius=0.1, dist=2, focus_xw=.1, focus_yh=
*
* %P
* radius: [m]                       Radius of circle in (x,y,0) p
* yheight: [m]                      Height of rectangle in (x,y,0
* xwidth: [m]                       Width of rectangle in (x,y,0)
* target_index: [1]                 relative index of component t
* dist: [m]                         Distance to target along z ax
* focus_xw: [m]                     Width of target
* focus_yh: [m]                     Height of target
* E0: [meV]                         Mean energy of neutrons.
* dE: [meV]                         Energy half spread of neutron
* lambda0: [AA]                     Mean wavelength of neutrons.
* dlambda: [AA]                     Wavelength half spread of neu
* flux: [1/(s*cm**2*st*energy unit)] flux per energy unit, Angs or
* gauss: [1]                        Gaussian (1) or Flat (0) ener
*
* %E
*******************************************************************

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

  if (flux) {
```

Slit unchanged



PSD lots of changes



PSD lots of changes

No more DEFINITION PARAMETERS

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

DECLARE
%{
double PSD_N[nx][ny];
double PSD_p[nx][ny];
double PSD_p2[nx][ny];
%}
INITIALIZE
%{
```

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

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

Use of new DArray2d for dynamic allocation

PSD lots of changes

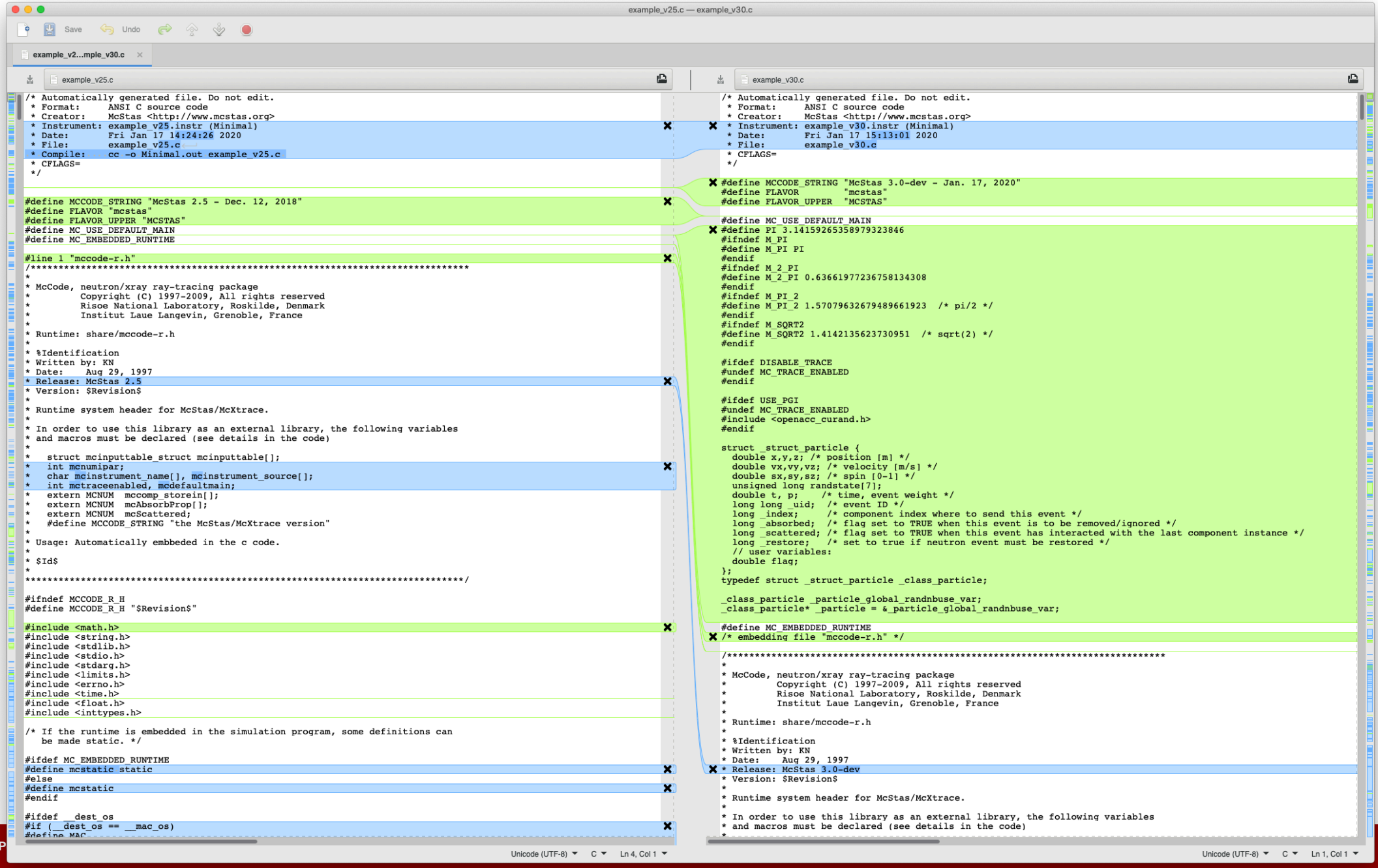
```
TRACE
%{
  int i,j;

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

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

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

Enabling atomic writes on the detector arrays

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

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

Generated code: LOTS of changes



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;
```

Input parameters and the instrument struct

```
struct _struct_instrument_parameters {
  MCNUM _dummy;
};
typedef struct _struct_instrument_parameters _class_instrument_parameters;

struct _instrument_struct {
  char   _name[256]; /* the name of this instrument e.g. 'Minimal' */
/* Counters per component instance */
  double counter_AbsorbProp[6]; /* absorbed events in PROP routines */
  double counter_N[6], counter_P[6], counter_P2[6]; /* event counters after each component instance */
  _class_particle _trajectory[6]; /* current trajectory for STORE/RESTORE */
/* Components position table (absolute and relative coords) */
  Coords _position_relative[6]; /* positions of all components */
  Coords _position_absolute[6];
  _class_instrument_parameters _parameters; /* instrument parameters */
} _instrument_var;
struct _instrument_struct *instrument = & _instrument_var;
```

… in e.g. an EXTEND section one should now use the macro INSTRUMENT_GETPAR(dummy) which translates the bare dummy into instrument->_parameters._dummy

```
#define INSTRUMENT_GETPAR(par) (instrument->_parameters._ ## par)
```

Declare section

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

Initialise section

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

  /* call iteratively all components INITIALISE */

  class_Source_simple_init(&_source_var);

  class_Slit_init(&_coll2_var);

  class_PSD_monitor_init(&_detector_var);
```

Functions per component with related component structs

Component struct example

```
/* component detector=PSD_monitor() [4] DECLARE */
/* Parameter definition for component type 'PSD_monitor' */
struct _struct_PSD_monitor_parameters {
  /* Component type 'PSD_monitor' setting parameters */
  MCNUM nx;
  MCNUM ny;
  char filename[16384];
  MCNUM xmin;
  MCNUM xmax;
  MCNUM ymin;
  MCNUM ymax;
  MCNUM xwidth;
  MCNUM yheight;
  MCNUM restore_neutron;
  long nowritefile;
  /* Component type 'PSD_monitor' private parameters */
  /* Component type 'PSD_monitor' DECLARE code stored as structure members */
  DArray2d PSD_N;
  DArray2d PSD_p;
  DArray2d PSD_p2;
}; /* _struct_PSD_monitor_parameters */
typedef struct _struct_PSD_monitor_parameters _class_PSD_monitor_parameters;
```

Built from component definition

```
/* Parameters for component type 'PSD_monitor' */
struct _struct_PSD_monitor {
  char      _name[256]; /* e.g. detector */
  char      _type[256]; /* PSD_monitor */
  long      _index; /* e.g. 2 index in TRACE list */
  Coords    _position_absolute;
  Coords    _position_relative; /* wrt PREVIOUS */
  Rotation  _rotation_absolute;
  Rotation  _rotation_relative; /* wrt PREVIOUS */
  int       _rotation_is_identity;
  _class_PSD_monitor_parameters _parameters;
};
typedef struct _struct_PSD_monitor _class_PSD_monitor;
_class_PSD_monitor _detector_var;
#pragma acc declare create ( _detector_var )
```

OpenACC clause to define variable in device scope.

Component init function example

```
_class_Source_simple *class_Source_simple_init( *_comp ) {
  #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_init] component source=Source_simple() INITIALISE [Source_simple.comp:68]");
square = 0;
/* Determine source area */
if (radius && !yheight && !xwidth ) {
    square = 0;
    srcArea = PI*radius*radius;
  } else if(yheight && xwidth) {
    square = 1;
    srcArea = xwidth * yheight;
  }

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

Contains component initialise section

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

Main trace Loop (v1)

```
/* loop to generate events and call raytrace() propagate them */
void raytrace_all(unsigned long long ncount, unsigned long seed) {

  /* CPU-loop */
  unsigned long long loops;
  long innerloop=2147483647;
  loops = ceil((double)ncount/innerloop);
  if (ncount>innerloop) {
    printf("Defining %llu CPU loops around kernel and adjusting ncount\n",loops);
    mcset_ncount(loops*2147483647);
  } else {
    loops=1;
    innerloop = ncount;
  }

  for (unsigned long long cloop=0; cloop<loops; cloop++) {
    if (loops>1) fprintf(stdout, "%d..", (int)cloop); fflush(stdout);

    #pragma acc parallel loop
    for (unsigned long pidx=0 ; pidx < innerloop ; pidx++) {
      _class_particle particleN = mcgenstate(); // initial particle
      _class_particle* _particle = &particleN;
      particleN._uid = pidx;

      long seq = pidx + seed;
      srandom(_hash(pidx + seed));

      raytrace(_particle);
    } /* inner for */
    seed = seed+innerloop;
  } /* CPU for */
  printf("\n");
} /* raytrace_all */
```

CPU may execute multiple GPU loops

Main parallel pragma that define kernel

Cogen-constructed function with calls to component trace functions

Main trace Loop (v1)

```
#pragma acc routine seq
int raytrace(_class_particle* _particle) { /* single event propagation, called by mccode_main for Minimal:TRACE */

  /* init variables and counters for TRACE */
  #undef ABSORB0
  #undef ABSORB
  #define ABSORB0 do { DEBUG_ABSORB(); MAGNET_OFF; ABSORBED++; return(ABSORBED);} while(0)
  #define ABSORB ABSORB0
  DEBUG_ENTER();
  DEBUG_STATE();
  /* the main iteration loop for one incoming event */
  while (!ABSORBED) { /* iterate event until absorbed */
    _class_particle _particle_save;
    /* send particle event to component instance, one after the other */
    char flag_nocoordschange=0;
    if (!ABSORBED && _particle->_index == 1) {
      /* begin component arm=Arm() [1] */
      if (!flag_nocoordschange) { // flag activated by JUMP to pass coords change
        if (_arm_var._rotation_is_identity) {
          coords_get(coords_add(coords_set(x,y,z), _arm_var._position_relative),&x, &y, &z);
        } else
          mccoordschange(_arm_var._position_relative, _arm_var._rotation_relative, _particle);
      } else flag_nocoordschange=0;
      _particle_save = *_particle;
      DEBUG_COMP(_arm_var._name);
      DEBUG_STATE();
      class_Arm_trace(&_arm_var, _particle); /* contains EXTEND code */
      if (_particle->_restore)
        *_particle = _particle_save;
      _particle->_index++;
      if (!ABSORBED) DEBUG_STATE();
    } /* end component arm [1] */
    if (!ABSORBED && _particle->_index == 2) {
      /* begin component source=Source_simple() [2] */
      if (!flag_nocoordschange) { // flag activated by JUMP to pass coords change
        if (_source_var._rotation_is_identity) {
          coords_get(coords_add(coords_set(x,y,z), _source_var._position_relative),&x, &y, &z);
        } else
          mccoordschange(_source_var._position_relative, _source_var._rotation_relative, _particle);
      } else flag_nocoordschange=0;
      _particle_save = *_particle;
      DEBUG_COMP(_source_var._name);
      DEBUG_STATE();
      class_Source_simple_trace(&_source_var, _particle);
      if (_particle->_restore)
        *_particle = _particle_save;
      _particle->_index++;
      if (!ABSORBED) DEBUG_STATE();
    } /* end component source [2] */
```

Comp 1

Comp 2

Main trace Loop (v2) potentially “funnelled”

```
void raytrace_all_funnel(unsigned long long ncount, unsigned long seed) {

  // set up outer (CPU) loop / particle batches
  unsigned long long loops;
  long innerloop=1024*1024; // <-- tune by memory capacity here (max threads: 2147483647);
  loops = ceil((double)ncount/innerloop);
  if (ncount>innerloop) {
    printf("Defining %llu CPU loops around kernel and adjusting ncount\n",loops);
    mcset_ncount(loops*2147483647);
  } else {
    loops=1;
    innerloop = ncount;
  }

  // outer loop / particle batches
  for (unsigned long long cloop=0; cloop<loops; cloop++) {
    if (loops>1) fprintf(stdout, "%d..", (int)cloop); fflush(stdout);

    // create particles memory block and pointer array (buffer and sorted)
    #ifdef USE_PGI
    _class_particle* particles = acc_malloc(innerloop*sizeof(_class_particle));
    #else
    _class_particle* particles = malloc(innerloop*sizeof(_class_particle));
    #endif
    #pragma acc enter data create(particles[0:innerloop])
    // TODO: _class_particle** psorted = malloc(innerloop*sizeof(_class_particle*));
    // TODO: _class_particle** pbuffer = malloc(innerloop*sizeof(_class_particle*));

    // TODO: do we need a GPU data section for the above buffers here?

    // set up, generate particles
    #pragma acc parallel loop present(particles)
    for (unsigned long pidx=0 ; pidx < innerloop ; pidx++) {
      // generate particle state, set loop index and seed
      particles[pidx] = mcgenstate();
      _class_particle* _particle = particles + pidx;
      _particle->_uid = pidx;
      srandom(_hash(pidx + seed)); // _particle->state usage built into srandom macro
    }

    // iterate components

    //TODO: innerloop = sort_absorb_last(particles, innerloop);
```

Smaller “innerloop” due to memory needs for the array of “particles” etc.

Allocation of particle array

“fill” the array and make it available to GPU

Initialise / generate particles GPU side

Main trace Loop (v2) potentially “funnelled”

```
// arm
#pragma acc parallel loop present(particles)
for (unsigned long pidx=0 ; pidx < innerloop ; pidx++) {
  _class_particle* _particle = particles + pidx;
  _class_particle _particle_save;
  if (!ABSORBED) {
    if (_arm_var._rotation_is_identity)
      coords_get(coords_add(coords_set(x,y,z), _arm_var._position_relative),&x, &y, &z);
    else
      mccoordschange(_arm_var._position_relative, _arm_var._rotation_relative, _particle);
    _particle_save = *_particle;
    class_Arm_trace(&_arm_var, _particle); /* contains EXTEND code */
    if (_particle->_restore)
      *_particle = _particle_save;
    _particle->_index++;
  }
}

//TODO: innerloop = sort_absorb_last(particles, innerloop);

// source
#pragma acc parallel loop present(particles)
for (unsigned long pidx=0 ; pidx < innerloop ; pidx++) {
  _class_particle* _particle = particles + pidx;
  _class_particle _particle_save;
  if (!ABSORBED) {
    if (_source_var._rotation_is_identity)
      coords_get(coords_add(coords_set(x,y,z), _source_var._position_relative),&x, &y, &z);
    else
      mccoordschange(_source_var._position_relative, _source_var._rotation_relative, _particle);
    _particle_save = *_particle;
    class_Source_simple_trace(&_source_var, _particle);
    if (_particle->_restore)
      *_particle = _particle_save;
    _particle->_index++;
  }
}

//TODO: innerloop = sort_absorb_last(particles, innerloop);
```

Comp 1 in independent kernel

One would then get rid of ABSORB’ed particles, do SPLITS here

Comp 2 in independent kernel

Both trace approaches execute the normal component trace function,

example:

```
#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);
```

Contains component trace section

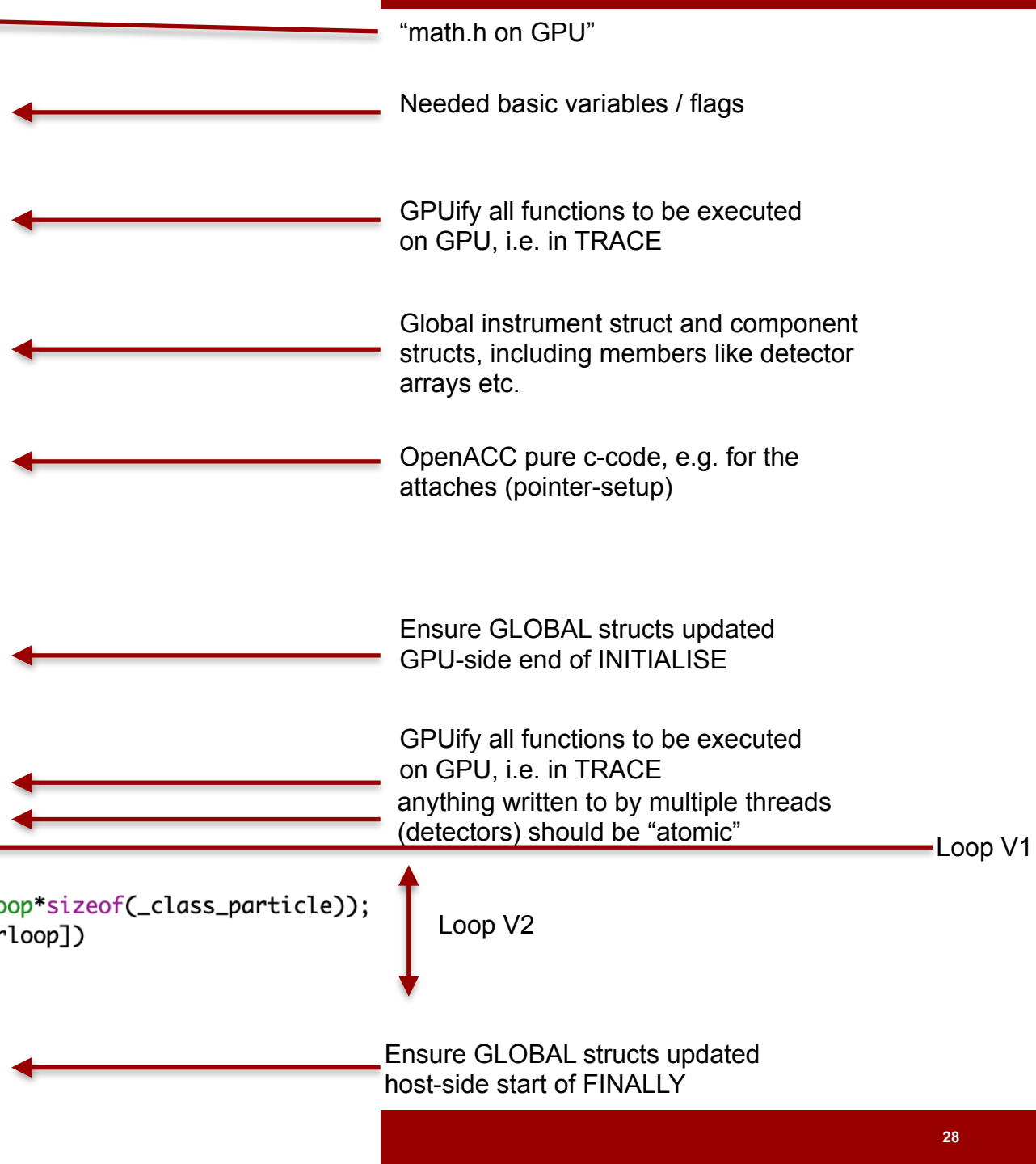
…. etc

Pro’s and cons

- V1 in general parallelises well, but SPLITS may not be easy to do
- V2 is slower due to multiple kernels and related memory transfer overhead (?). Allows SPLIT
- We are thinking if we should do V3 “best of both worlds”, i.e. multiple kernels only when SPLIT is introduced, each SPLIT infers a problem-reduction and another kernel
- Underlying component code identical, only changes in the code generation needed

Pragmas and libs used

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

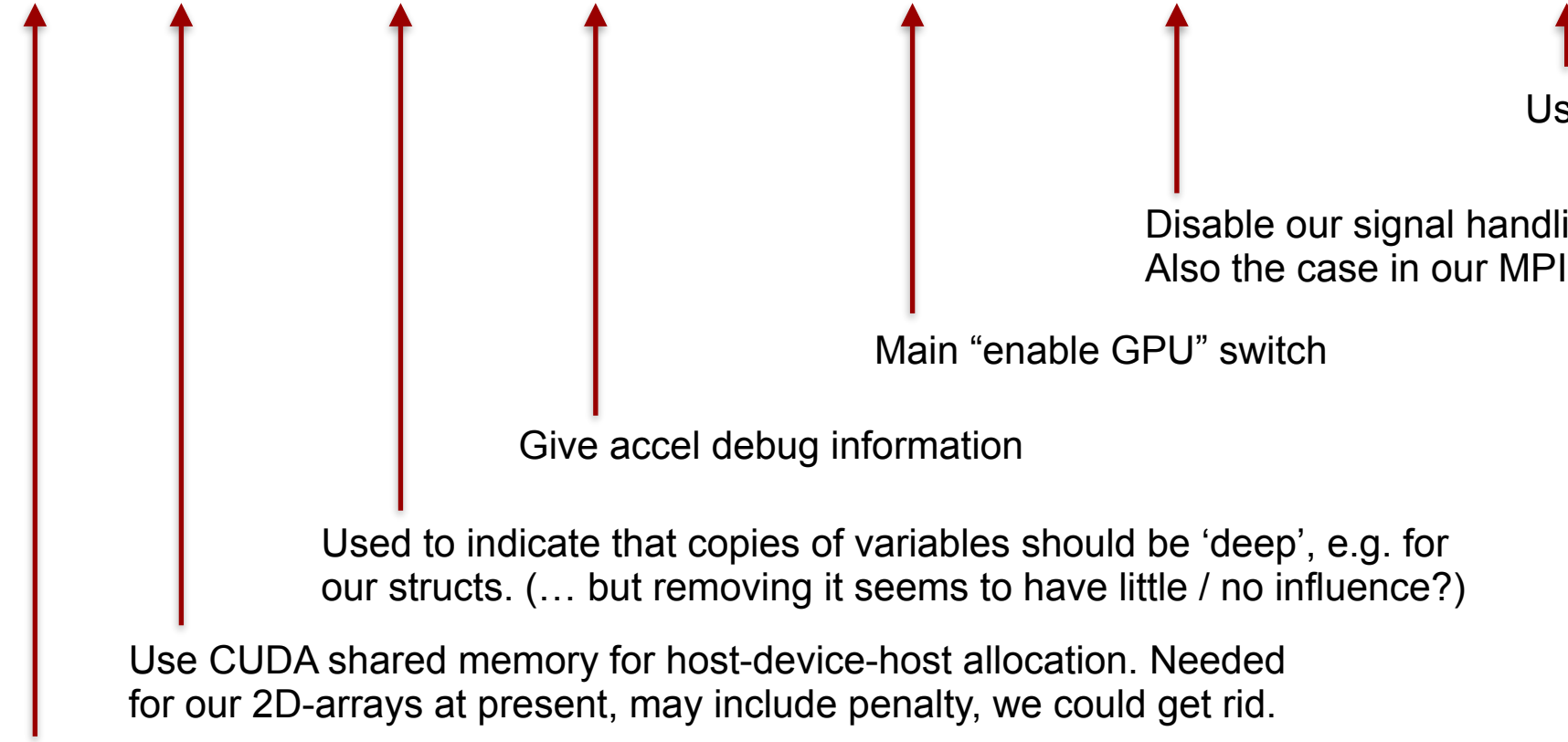


New RNG ‘KISS’

- We couldn’t easily port Mersenne Twister
- Experimenting with curand showed huge overhead for our relative small number of random numbers
- An RNG ‘state’ carried with each particle - bonus: same seed gives same numbers even when comparing between CPU and GPU
- Required patching prototype of ALL functions making use of e.g. rand01()

Compiler settings used

pgcc -ta=tesla,managed,deepcopy -Minfo=accel -DUSE_PGI -DNOSIGNALS -DRNG_ALG=2



Use our new KISS rng

Disable our signal handling, e.g. USR2 for save.
Also the case in our MPI implementation.

Main “enable GPU” switch

Give accel debug information

Used to indicate that copies of variables should be ‘deep’, e.g. for our structs. (… but removing it seems to have little / no influence?)

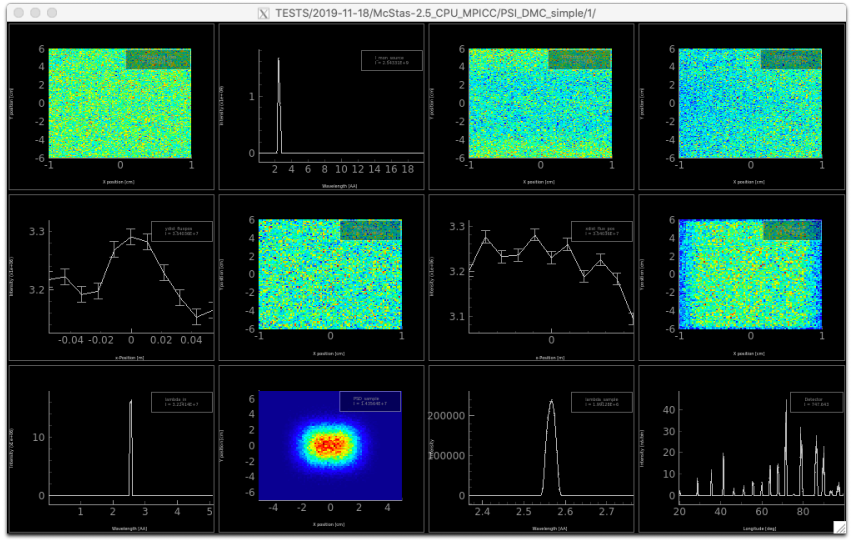
Use CUDA shared memory for host-device-host allocation. Needed for our 2D-arrays at present, may include penalty, we could get rid.

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

McStas heading for the GPU… November 2019

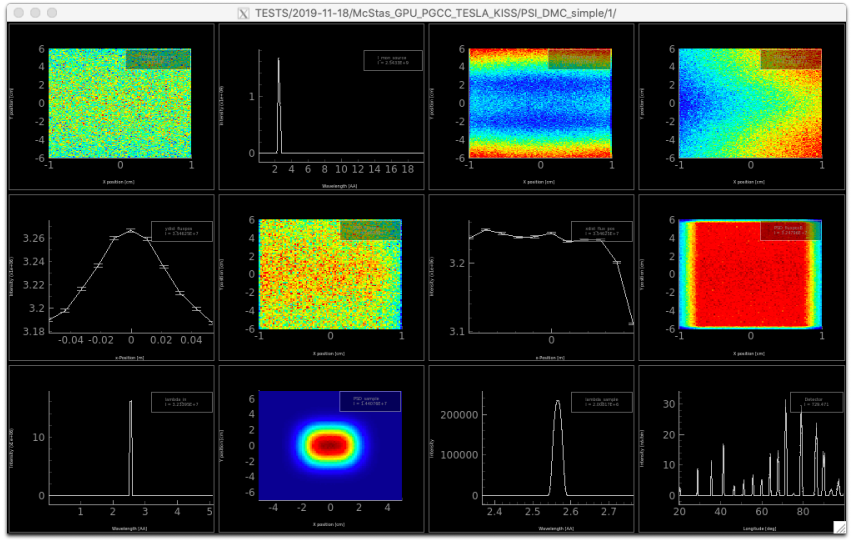
9 instruments fully ported, also realistic ones like PSI_DMC*

10-core MPI run, **1e7** in 2 secs



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

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



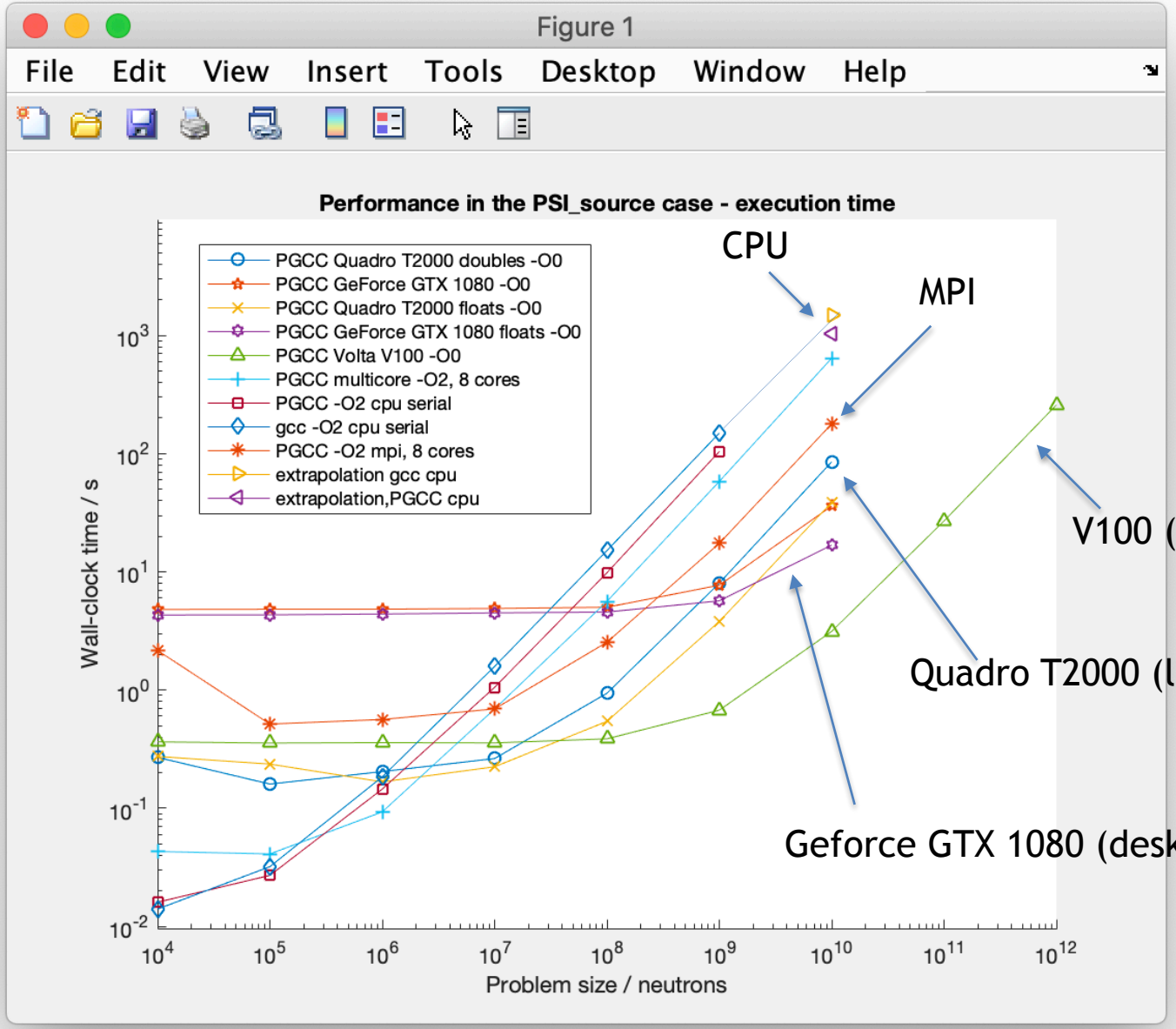
*Guide component without reflection-file support, SPLIT disabled, OFF geometry disabled

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

Wallclocks:

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

(Likely a good indication of maximal speedup achievable.)

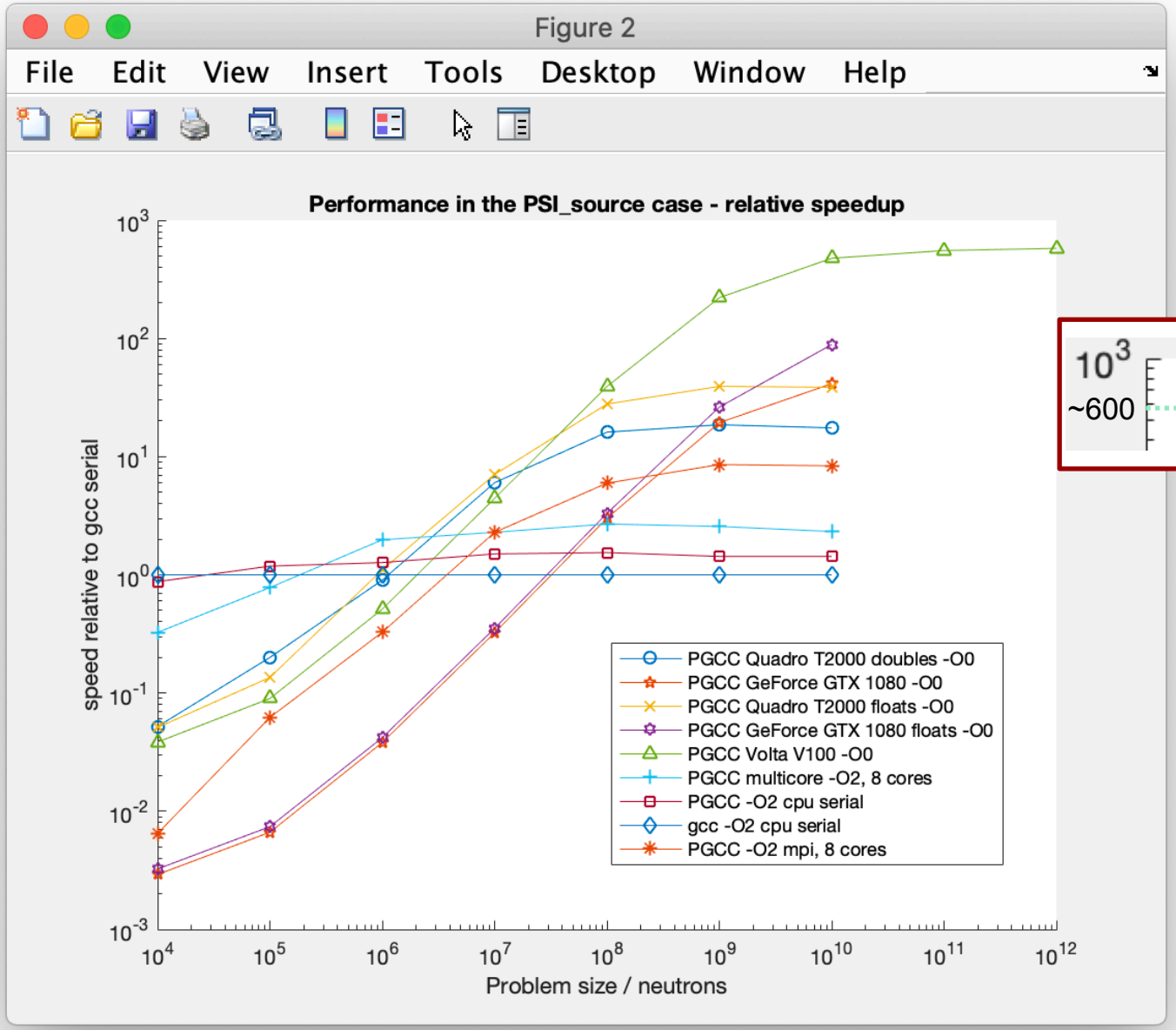


McStas heading for the GPU… November 2019 - first good look at performance.

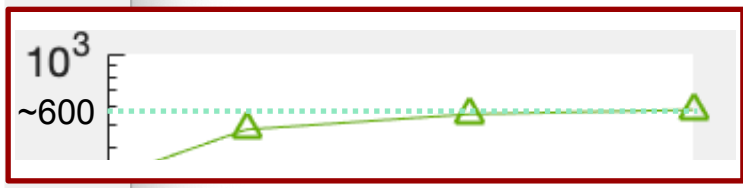
Speedup:

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

(Likely a good indication of maximal speedup achievable.)



Looks like a factor of ~600



Renormalised to wall-clock of single-core gcc standard simulation

McStas heading for the GPU… December 2019 - today's compilation status:

Numerical output with graphics:
http://new-nightly.mccode.org/2019-12-06/2019-12-06_output.html

Statistics:

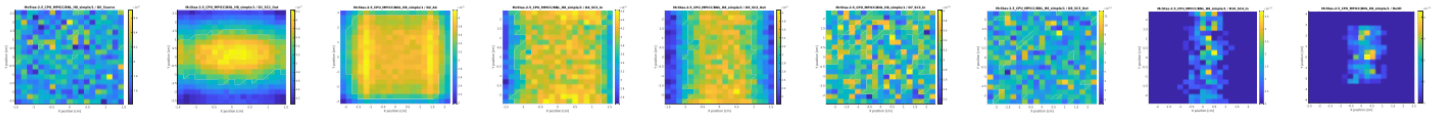
http://new-nightly.mccode.org/2019-12-06/stats.txt

(38 of 142 instruments, 62 of 207 components

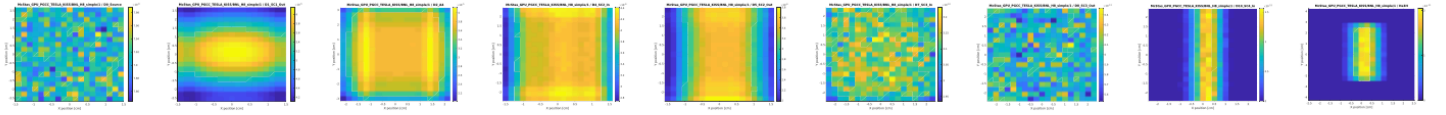
BNL_H8_simple/1 - comparison McStas-2.5_CPU_MPICC vs McStas_GPU_PGCC_TESLA_KISS

(Click to access files)

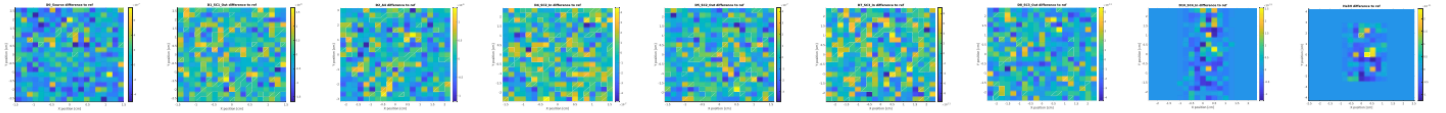
McStas-2.5_CPU_MPICC (reference)



McStas_GPU_PGCC_TESLA_KISS



Difference



test results

ref user: pkwi

	McStas-2.5_CPU_MPICC (ref) - 1e7 n-62-23-6 Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz 20191120_0127_38	McStas_CPU_GCC_KISS - 1e6 n-62-21-99 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz 20191205_0058_36	McStas_CPU_GCC_MT - 1e6 n-62-21-99 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz 20191205_0122_14	McStas_CPU_MPICC_KISS - 1e7 n-62-23-9 Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz 20191205_0034_53	McStas_GPU_PGCC_TESLA_KISS - 1e9 n-62-20-6 Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz Tesla V100-PCIE-16GB 20191205_0012_04
BNL_H8_simple	5.80 s | 1.99 s | 9.7e-10 | 99%	4.32 s | 1.03 s | 1.3e-09 | 136%	4.35 s | 0.99 s | 1e-09 | 106%	4.11 s | 1.50 s | 1.1e-09 | 110%	17.59 s | 8.16 s | 9.7e-10 | 98%

McStas 3.0 - next generation code generator - release plans

- Limited-functionality “beta” release to be made public soon (jan-mar) after **2.6 (january)**
  - Expect bugs!
  - Only a subset of components / instruments
  - Event interchange with 2.6 possible via MCPL

- Main purpose: **get this working in ‘the wild’**
  - Your instruments will likely **require (limited) rewriting**
    - Your instruments will need USERVARS for flags that change with each neutron
  - Your own components will **likely require rewriting**
    - Support for DEFINITION PARAMETERS is deprecated. Use the new string, vector, … instead
    - New macro MC_GETPAR2 needed to access other component scopes (NOT via defines)
    - E.g. the declare section **cannot include assignments**
    - Arrays must be declared/initialized using a new set of functions (i.e. not double PSD_I[nx][ny] with definition parms)

- Hence some backward compatibility is lost and we need **to increment major release #**

McStas 3.0 - next generation code generator - code camp January 27th-31st in Copenhagen

- **Code-camp participants:**
  - **McStas-McXtrace team** (bold means confirmed):
    - **Peter Willendrup, Emmanuel Farhi, Erik Knudsen, Jakob Garde, Tobias Weber, Torben R Nielsen, Mads Bertelsen**
  - **RAMP team from the UK**
    (RAMP is relatively young "McStas equivalent" with GPU-support from the beginning via C++ and OpenCL.)
    - **Gino Cassella, Göran Nilsen**
  - **We didn't attract someone from Nvidia, but we have access to Guido Juckeland in Dresden**
- **Code-camp focal points:**
  - Port as many remaining instruments / components as possible to GPU
  - Experiment with "telescopic flow"/V2/V3, i.e. a different approach to handling ABSORB's and SPLIT's
  - Experiment with simulation flow between CPU and GPU (a few things can not be done GPU)
  - Comparisons with RAMP
  - Initiate port of GPU work to McXtrace tree
  - Have fun! :-)