Peter Willendrup and Erik Knudsen DTU Physics

Adapting code for McStas 3 and GPUs

Agenda

- McStas on GPU via OpenACC
  (a “high-level” #pragma driven access to CUDA see https://www.openacc.org and https://developer.nvidia.com/hpc-sdk)
- How well (fast) does it work?
- Simulation flow
- What did we change?
- What needs doing on an instrument / component?
- What does not work

Warning:
1. Assumes previous experience with McStas
2. Does not introduce OpenACC

McStas 2.x -> McStas 3.0 main differences

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

Advantage of 3.0 also on CPU

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

OpenACC

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

McXtrace

McStas

The neutron and USERVARS in the instrument

v2.5: Global variables

```
double x, y, z, vx, vy, vz, t, sx, sy, sz, p;      double flag;
```

Can be probed using e.g. Monitor_nD with user1="flag" which uses the function

`double particle_getvar(_class_particle *p, char *name, int *suc)`

also works with e.g. "x"

v3.0: particle struct, including any USERVARS like flag.

```
struct _struct_particle {
  double x,y,z; /* position [m] */
  double vx,vy,vz; /* velocity [m/s] */
  double sx,sy,sz; /* spin [0-1] */
  unsigned long randstate[7];
  double t, p;    /* time, event weight */
  long long _uid;  /* event ID */
  long _index;     /* component index where to send this event */
  long _absorbed;  /* flag set to TRUE when this event is to be removed/ignored */
  long _scattered; /* flag set to TRUE when this event has interacted with the last component instance */
  long _restore;   /* set to true if neutron event must be restored */
  // user variables:
  double flag;
};
typedef struct _struct_particle _class_particle;
```

The neutron and USERVARS in the instrument

v2.5: Global variables

```
double x, y, z, vx, vy, vz, t, sx, sy, sz, p;      double flag;
```

v3.0: particle struct, including any USERVARS like flag.

```
struct _struct_particle {
  double x,y,z; /* position [m] */
  double vx,vy,vz; /* velocity [m/s] */
  double sx,sy,sz; /* spin [0-1] */
  unsigned long randstate[7];
  double t, p;    /* time, event weight */
  long long _uid;  /* event ID */
  long _index;     /* component index where to send this event */
  long _absorbed;  /* flag set to TRUE when this event is to be removed/
  long _scattered; /* flag set to TRUE when this event has interacted wi
  long _restore;   /* set to true if neutron event must be restored */
  // user variables:
  double flag;
};
typedef struct _struct_particle _class_particle;
```

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

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

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

Samples required a good deal of (detective) work…

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

Solutions:

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

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

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

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

Side-effect of thread-local vars:

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

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

McXtrace

McStas

New monitor-tools for debugging use

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

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

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

OpenACC

DTU EUROPEAN SPALLATION SOURCE

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

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

 t=0;
 z=0;

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

.... etc

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

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

Init and finalisation codes run purely CPU.

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

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

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

Contains component trace section

McXtrace

McStas

CROSSCHECK
Should give the SAME on CPU and GPU!

Porting an instrument to 3.0 and GPU

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

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

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

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

Common error-messages:

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

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

- At runtime, this indicates a GPU segfault

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

  - Often a symptom of "illegal access", colliding memory, something isn't thread-safe…

CROSSCHECK
Should give the SAME on CPU and GPU!

Porting a comp to GPU

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

Highlights of comps that work differently

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

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

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

The team, Nvidia mentors and Hackathon hosts :-)


Vishal Metha


Christian Hundt


Alexey Romanenko


Guido Juckeland


Sebastian von Alfthan

What code-parts got #pragma acc

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

Examples:

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

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

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

Sources: (TRACE-functions in SHARE)

sources/Source_Maxwell_3.comp
sources/Source_gen.comp

Samples: (TRACE-functions in SHARE)

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

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

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

All **monitors**:

#pragma acc atomic sections for arrays

Misc:
atomic capture for insertion in array of particle events

misc/MCPL_output.comp

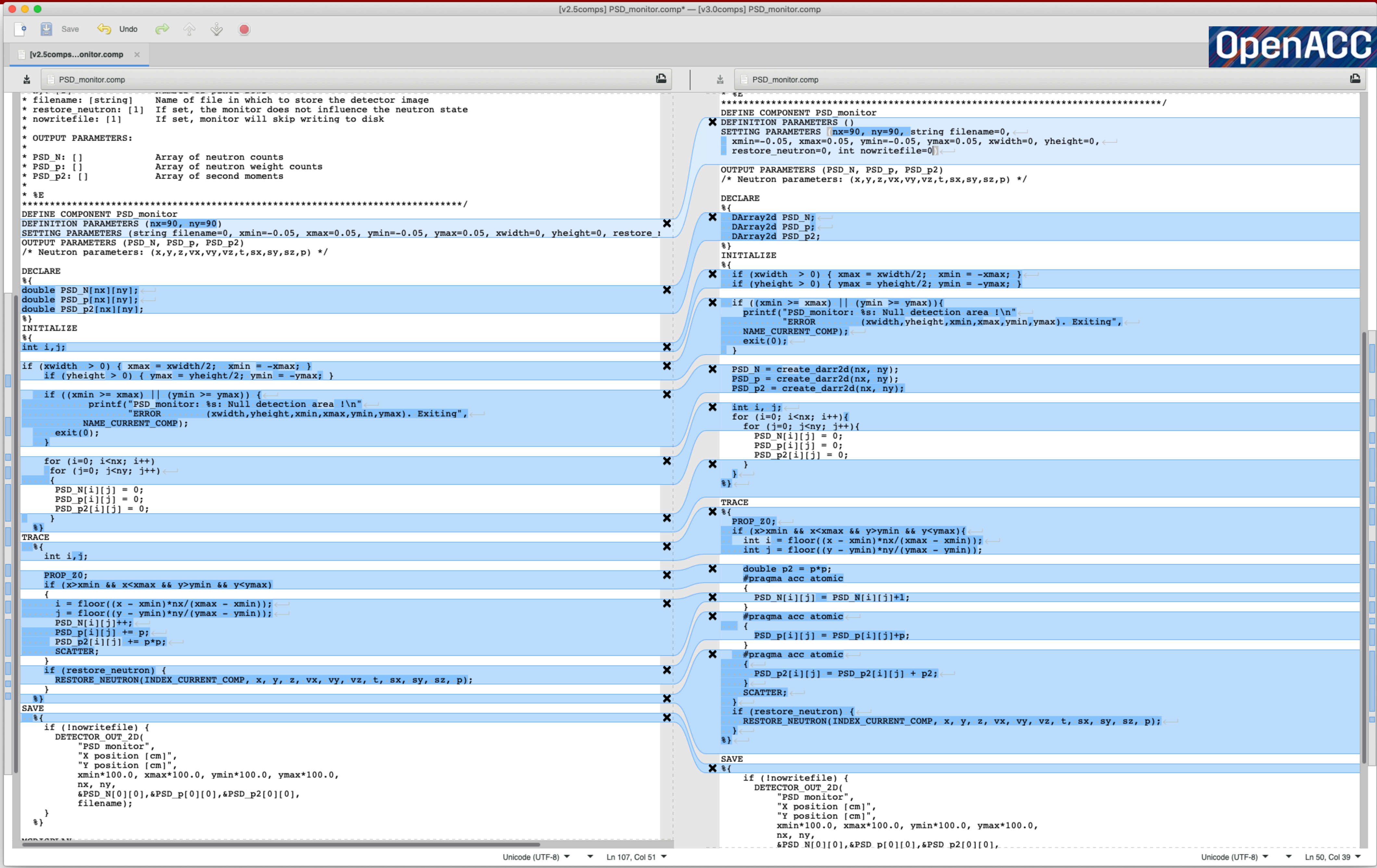
Contrib comps: (atomics in mon’s, #pragma acc routine for TRACE-funcs)

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

New RNG ‘KISS’

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

PSD lots of changes



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

Save   Undo

[v2.5comps...onitor.comp

PSD_monitor.comp

```
* filename: [string]    Name of file in which to store the detector image
* restore_neutron: [1]  If set, the monitor does not influence the neutron state
* nowritefile: [1]      If set, monitor will skip writing to disk
*
* OUTPUT PARAMETERS:
*
* PSD_N: []             Array of neutron counts
* PSD_p: []             Array of neutron weight counts
* PSD_p2: []            Array of second moments
*
* %E
*******************************************************************************/
DEFINE COMPONENT PSD_monitor
DEFINITION PARAMETERS (nx=90, ny=90)
SETTING PARAMETERS (string filename=0, xmin=-0.05, xmax=0.05, ymin=-0.05, ymax=0.05, xwidth=0, yheight=0, restore_
OUTPUT PARAMETERS (PSD_N, PSD_p, PSD_p2)
/* Neutron parameters: (x,y,z,vx,vy,vz,t,sx,sy,sz,p) */

DECLARE
%{
double PSD_N[nx][ny];
double PSD_p[nx][ny];
double PSD_p2[nx][ny];
%}
INITIALIZE
%{
int i,j;

if (xwidth  > 0) { xmax = xwidth/2;  xmin = -xmax; }
    if (yheight > 0) { ymax = yheight/2; ymin = -ymax; }

    if ((xmin >= xmax) || (ymin >= ymax)) {
            printf("PSD_monitor: %s: Null detection area !\n"
                   "ERROR        (xwidth,yheight,xmin,xmax,ymin,ymax). Exiting",
           NAME_CURRENT_COMP);
      exit(0);
    }

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

    PROP_Z0;
    if (x>xmin && x<xmax && y>ymin && y<ymax)
    {
      i = floor((x - xmin)*nx/(xmax - xmin));
      j = floor((y - ymin)*ny/(ymax - ymin));
      PSD_N[i][j]++;
      PSD_p[i][j] += p;
      PSD_p2[i][j] += p*p;
      SCATTER;
    }
    if (restore_neutron) {
      RESTORE_NEUTRON(INDEX_CURRENT_COMP, x, y, z, vx, vy, vz, t, sx, sy, sz, p);
    }
  %}
SAVE
  %{
    if (!nowritefile) {
      DETECTOR_OUT_2D(
          "PSD monitor",
          "X position [cm]",
          "Y position [cm]",
          xmin*100.0, xmax*100.0, ymin*100.0, ymax*100.0,
          nx, ny,
          &PSD_N[0][0],&PSD_p[0][0],&PSD_p2[0][0],
          filename);
    }
  %}
```

Unicode (UTF-8)   Ln 107, Col 51

PSD_monitor.comp

```
* %E
*******************************************************************************/
DEFINE COMPONENT PSD_monitor
DEFINITION PARAMETERS ()
SETTING PARAMETERS (nx=90, ny=90, string filename=0,
  xmin=-0.05, xmax=0.05, ymin=-0.05, ymax=0.05, xwidth=0, yheight=0,
  restore_neutron=0, int nowritefile=0)

OUTPUT PARAMETERS (PSD_N, PSD_p, PSD_p2)
/* Neutron parameters: (x,y,z,vx,vy,vz,t,sx,sy,sz,p) */

DECLARE
%{
  DArray2d PSD_N;
  DArray2d PSD_p;
  DArray2d PSD_p2;
%}
INITIALIZE
%{
  if (xwidth  > 0) { xmax = xwidth/2;  xmin = -xmax; }
  if (yheight > 0) { ymax = yheight/2; ymin = -ymax; }

  if ((xmin >= xmax) || (ymin >= ymax)){
    printf("PSD_monitor: %s: Null detection area !\n"
           "ERROR        (xwidth,yheight,xmin,xmax,ymin,ymax). Exiting",
    NAME_CURRENT_COMP);
    exit(0);
  }

  PSD_N = create_darr2d(nx, ny);
  PSD_p = create_darr2d(nx, ny);
  PSD_p2 = create_darr2d(nx, ny);

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

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

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

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

Unicode (UTF-8)   Ln 50, Col 39

Same seed and same # mpi nodes -> same output

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

Declare section

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

Initialise section

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

  /* call iteratively all components INITIALISE */

  class_Source_simple_init(&_source_var);

  class_Slit_init(&_coll2_var);

  class_PSD_monitor_init(&_detector_var);
```

Functions per component with related component structs

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

INITIALISE

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

Similar “host” update in FINALLY

EUROPEAN SPALLATION SOURCE

“Full” list of pragmas and accel-code used

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

“math.h on GPU”

Needed basic variables / flags

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

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

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

Ensure GLOBAL structs updated GPU-side end of INITIALISE

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

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

Loop V1

Loop V2

Ensure GLOBAL structs updated host-side start of FINALLY

McXtrace

McStas

Conclusions

- **It really does work nicely!**
- **Code changes** much **less invasive** than envisioned!
- It often gives a speedup of **1-2 orders** of magnitude over 1 cpu
- **Most things work**
  (we have workarounds or solutions in the pipe for the rest)
- **Documentation** comes in the form of the released code + this set of slides…
- McStas 3.0 is as of yet “ported” to GPU but **not fully “optimised” performance-wise**, we will try to go to another Hackathon
- **Union** needs a dedicated **Hackathon**