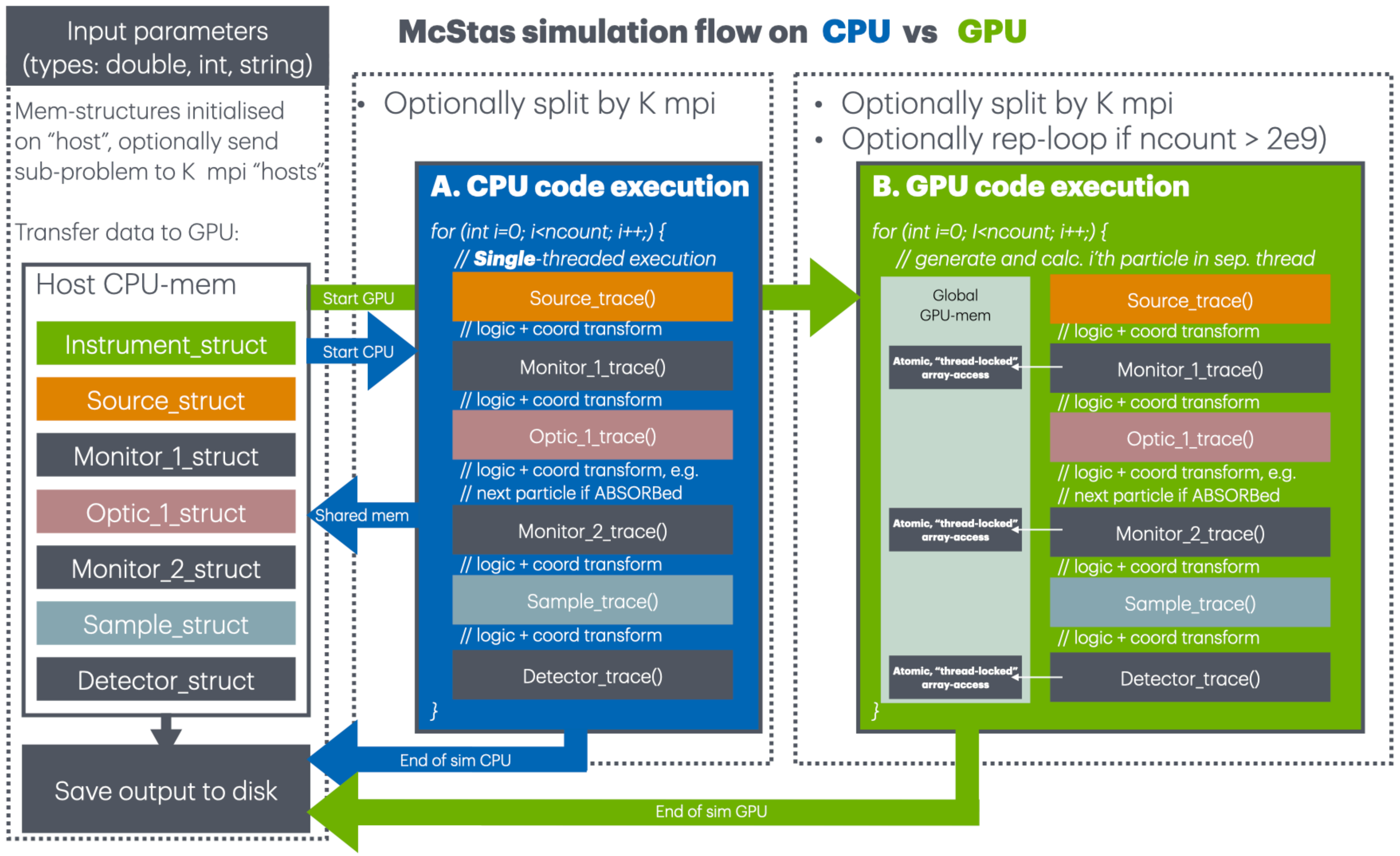
McStas simulation flow on CPU vs GPU


Input parameters
(types: double, int, string)
Mem-structures initialised on "host", optionally send sub-problem to K mpi "hosts"
Transfer data to GPU:
Host CPU-mem
Instrument_struct
Source_struct
Monitor_1_struct
Optic_1_struct
Monitor_2_struct
Sample_struct
Detector_struct
Save output to disk
Start GPU
Start CPU
Shared mem
End of sim CPU
End of sim GPU
• Optionally split by K mpi
• Optionally split by K mpi
• Optionally rep-loop if ncount > 2e9)
A. CPU code execution
B. GPU code execution
Global
GPU-mem
Atomic, "thread-locked" array-access
Atomic, "thread-locked" array-access
Atomic, "thread-locked" array-access

```
for (int i=0; i<ncount; i++;) {
   // Single-threaded execution
   Source_trace()
   // logic + coord transform
   Monitor_1_trace()
   // logic + coord transform
   Optic_1_trace()
   // logic + coord transform, e.g.
   // next particle if ABSORBed
   Monitor_2_trace()
   // logic + coord transform
   Sample_trace()
   // logic + coord transform
   Detector_trace()
}
```

```
for (int i=0; I<ncount; i++;) {
   // generate and calc. i'th particle in sep. thread
   Source_trace()
   // logic + coord transform
   Monitor_1_trace()
   // logic + coord transform
   Optic_1_trace()
   // logic + coord transform, e.g.
   // next particle if ABSORBed
   Monitor_2_trace()
   // logic + coord transform
   Sample_trace()
   // logic + coord transform
   Detector_trace()
}
```

‘Standard layout’ - 1

- 1 big parallel loop

```
#pragma acc parallel loop num_
  for (unsigned long pidx=0 ; pid
     …
     raytrace(_particle);
}
```

With

```
int raytrace(_class_particle* _particle) { /* single event propagation, called by
mccode_main for mini:TRACE */
  …
  _class_particle _particle_save=*_particle;
  /* the main iteration loop for one incoming event */
  while (!ABSORBED) { /* iterate event until absorbed */
    /* send particle event to component instance, one after the other */
    /* begin component arm=Arm() [1] */

   /* coordinate change pr. comp, trace fct. pr. comp until the end of comp list / ABS*/
   …
    /* begin component source=Source_simple() [2] */
    mccoordschange(_source_var._position_relative, _source_var._rotation_relative,
_particle);
    if (!ABSORBED && _particle->_index == 2) {
    …
    class_Source_simple_trace(&_source_var, _particle);
    /* restore-logic etc, then next comp */
    ….

  } /* while !ABSORBED */

  DEBUG_LEAVE()
  particle_restore(_particle, &_particle_save);
  DEBUG_STATE()

  return(_particle->_index);
} /* raytrace */
```

‘FUNNEL layout’
- potentially multiple kerne

- Particles traced in bunches of size ‘livebatchsize’:

With a split, minimum:

1. Initial kernel
2. Sorting kernel
3. Second kernel

Followed by ‘grouped’ components, e.g. between SPLIT’s

```
   #pragma acc parallel loop present(particles[0:livebatchsize])
   for (unsigned long pidx=0 ; pidx < livebatchsize ; pidx++) {
     _class_particle* _particle = &particles[pidx];
     _class_particle _particle_save;

     // arm
   if (!ABSORBED && _particle->_index == 1) {
       _particle->_index++;
     }
    // Comps up to monochromator in e.g. PSI_DMC
   …
  }

   // SPLIT with available livebatchsize
   long mult_foc_mono;
   livebatchsize = sort_absorb_last(particles, pbuffer, livebatchsize, gpu_innerloop, 1, &mult_foc_mono);
   //printf("livebatchsize: %ld, split: %ld\n",  livebatchsize, mult);

   #pragma acc parallel loop present(particles[0:livebatchsize])
   for (unsigned long pidx=0 ; pidx < livebatchsize ; pidx++) {
     _class_particle* _particle = &particles[pidx];
     _class_particle _particle_save;

     // foc_mono
   if (!ABSORBED && _particle->_index == 21) {
      …
        mccoordschange(_foc_mono_var._position_relative, _foc_mono_var._rotation_relative, _particle);
       _particle_save = *_particle;
       class_Monochromator_2foc_trace(&_foc_mono_var, _particle);
       if (_particle->_restore)
       particle_restore(_particle, &_particle_save);
       _particle->_index++;
     }

  …
}
```

dom macro

}

‘MULTIKERNEL layout’
- potentially multiple kernels

- Like FUNNEL but there will always be a kernel pr. comp

- Define to globally use / not use ‘absorption sort’ kernels in between comp kernels

- Rationale:
  - Allow more fine-grained profiling?
  - Alternative: Can we add pragmas for profiling / adding nvtx like points in acc regions?