SixTrack for GPU

R. De Maria

SixTrack Status

SixTrack: Single Particle Tracking Code [cern.ch/sixtrack].

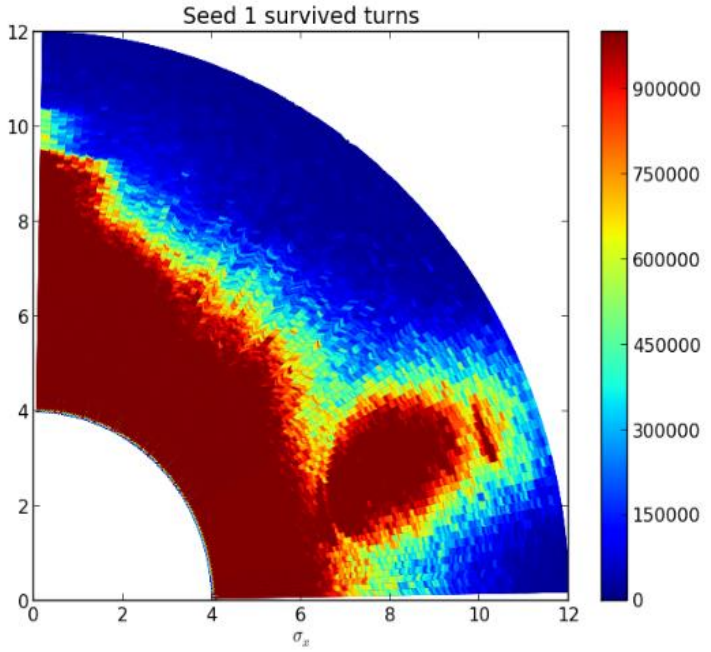
- 70K lines written in Fortran 77/90 (with few pre-processing steps).
- Numerically portable across OS and compilers.
- Used in the volunteer computing project LHC@Home with 200k registered users and about 20k cpus simultaneously running.

Example of an LHC simulation:

- 30k particles;
- 10^7 turns;
- 20k beam line elements.

Code speed:

- average 100 ns per particle per beam element
- 500 turns/(particle·sec) on serial code in recent hardware (LHC particles make 11245 turns/sec)



SixTrack GPU Status

GPU porting is being explored in the context of

- LHC@Home to use volunteer GPU:
  - heterogeneous hardware and software hard to test and fully deploy, many low-end GPU expected (low FP64 FLOPS count).
  - D. Mikushin (Applied Parallel Computing LLC) [indico/event/450856] demonstrated deploying with CUDA + additional compilation stages + code annotations + special compiler software (numerically ok without FMAC instructions, no benchmark available).
- Standalone tracking library (SixTrackLib) to be used with other codes (including SixTrack itself):
  - lightweight code being written in C/OpenCL for flexibility/portability (CERN&GSoC'14-'15).
  - speed-up of 250x w.r.t single i7 core with AMD-280X (~1TFLOPS FP64, ~300CHF) on first tests driven by pyopencl.
  - ongoing development: OpenCL not completed yet, pure python version benchmarked on components done for the LHC.

Hardware for single particle simulations:

- High FP64 FLOPS counts.
- Memory bandwidth and memory size less important.

Recent Hardware for FP64

19/02/2016

Name	TFLOPS (SP/DP)	Mem GB	Price
AMD Radeon 280	3.2/0.8	3	194CHF/175$
AMD Radeon 280X	4.1/1	3	no stock /220$
AMD FirePro W8100	4.2/2.1	8/ECC	1100CHF/1000$
AMD FirePro W9100	5.2/2.6	16/ECC	3200CHF/3000$
Nvidia Titan Black	4/1.3	6	~1000$ (not available)
Nvidia Titan Z	8/2.7	12	~1500$ (ebay)
Nvidia Tesla K40	4.2/1.4	12/ECC	4000CHF/3200$
Nvidia Tesla K80	8.7/2.9	24/ECC	5500CHF/5000$

10x cost difference for same (nominal) performance!

SixTrack: Model

Tracking: propagate *p* particles through *m* elements for *n* times

funset= list of functions
elememts = list of arrays
particles = array of arrays

Single Particle Loop:

```
for z in particles
for n times
for elem in elements
  f=funset[elem.type]
  z=f(elem,z)
```

Multi Particle Loop:

```
for n times
for elem in elements
  g=funset[elem.type]
  if  g is multiparticle
    particles=g(elem,particles)
  elif g is singleparticle_block
    for z in particles
      for elem in elements
        f=funset[elem.type]
        z=f(elem,z)
```

SixTrackLib: kernel implemented in OpenCL

```
z=particles[thread_id]
for elem in elements
   f=funset[elem.type]
   z=f(elem,z)
particles[thread_id]=z
```

SixTrackLib: implementation details

SixTrackLib: kernel implemented in OpenCL
z=particles[*thread_id*]
for *elem_id* in sequence
  elem=elements[*elem_id*]
  f=funset[*elem.*type]
  z=f(*elem*,*z*)
particles[*thread_id*]=z

funset= list of inlined functions in kernel (or in C particle loop for serial version)
=> switch case statements (no function pointer available in Open CL 1.2)
elements = list of arrays (prepared in CPU and copied once to global GPU memory)
=> flat array of union double/integers with structure indices
particles = array of arrays (prepared in CPU and copied once to global GPU memory)
=> flat array of union double/integers (row size fixed)
sequence = array of ints (in the elements array)

Code contains other complications (e.g. dynamic element manipulation, recursion and particle loss) not covered here.