Overview

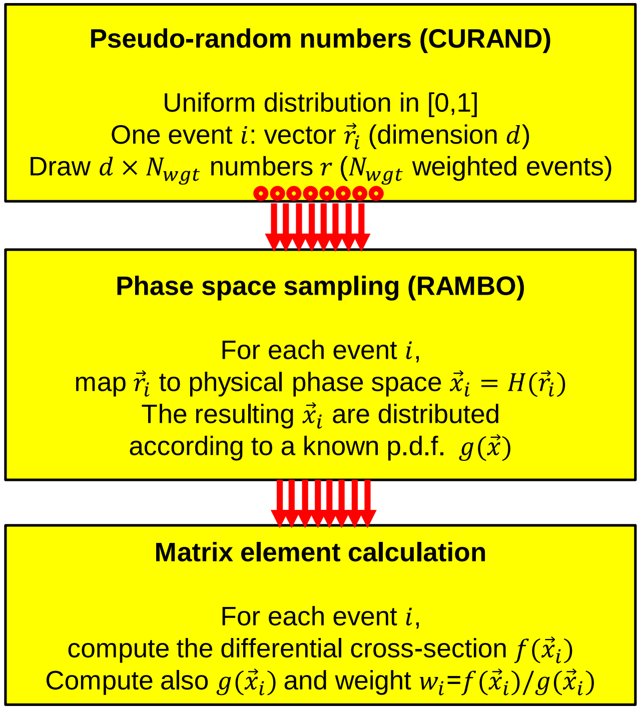
- My code is in https://gitlab.cern.ch/roiser/madgraph4gpu/-/tree/master/examples/gpu/eemumu_AV
  - Integrates my developments with some (not all…) of others' developments
    - Olivier's simplified IXX/OXX
    - Curand for random numbers
    - Not there yet: cucomplex, streams, graphs…

- General idea: single source, try to keep options open with #ifdefs
  - Cuda and c++ (symlink xxx.cc files as gxxx.cu files)
  - AOSOA (ASA), AOS, SOA memory layouts
  - Local/global/shared memory for the w[5][6] intermediate wavefunctions
  - Double and single precision (FPTYPE = double or float, CXTYPE = complex)
  - Curand generation on host and on device

Overall data flow

- Streamlined use of C-style arrays for inputs and outputs
  - Removing std::vector by itself is a large speedup in both c++ and cuda
  - Fix dimensions in advance
    - Not hardcoded: events per each iteration = ndim (=nthr*nblk i.e. #threads*#blocks)
    - Hardcoded: npar=4 (e+ e- mu+ mu-), np4=4 (E px py pz)
    - Each event also needs np4 random numbers

- Three main phases
  - Random number generation
    - Output: FPTYPE rnarray [ndim * npar * np4]
    - Need np4=4 random numbers per particle
      - (even if only 3 degrees of freedom)
  - Rambo: map random numbers to momenta
    - Input: rnarray [ndim * npar * np4]
    - Output: FPTYPE allmomenta[ndim * npar * np4]
    - Output: FPTYPE weights[ndim]
  - Sigmakin: from momenta to MEs
    - Input: allmomenta[ndim * npar * np4]
    - Output: FPTYPE MEs[ndim]
  - Physics validation: mean and stddev of MEs
    - Missing: proper computation with weights



Pseudo-random numbers (CURAND)

Uniform distribution in [0,1]
One event i: vector $\vec{r}_i$ (dimension d)
Draw $d \times N_{wgt}$ numbers r (N_{wgt} weighted events)

Phase space sampling (RAMBO)

For each event i,
map $\vec{r}_i$ to physical phase space $\vec{x}_i = H(\vec{r}_i)$
The resulting $\vec{x}_i$ are distributed
according to a known p.d.f. $g(\vec{x})$

Matrix element calculation

For each event i,
compute the differential cross-section $f(\vec{x}_i)$
Compute also $g(\vec{x}_i)$ and weight $w_i = f(\vec{x}_i)/g(\vec{x}_i)$

Memory layout for allmomenta

- Current default: AOSOA (ASA): allmomenta[npag][npar][np4][nepp]
  - Partition ndim events per iteration into npag pages of nepp events per page
    - i.e. ndim = npag * nepp
  - Currently nepp=32 is hardcoded (#threads per GPU warp)
    - Eventually? Use nepp = #threads per block i.e. allmomenta[nblk][npar][np4][nthr]
- Alternative SOA: allmomenta[npar][np4][ndim]
- Alternative AOS: allmomenta[ndim][npar][np4]

- *Idea was to try and exploit memory coalescing (and SIMD?) in the GPU*
  - *BUT: no obvious performance benefit/penalty in any of these choices* ☹

- *Probably best to keep AOSOA anyway?*
  - *allmomenta[nblk][npar][np4][nthr]?*

Local memory and shared memory

- This refers to the intermediate wavefunctions w[5][6] in each event
  - Where w[0:3] are the npar=4 particles, w[4] is a tmp for internal particles

- Current default: LOCAL w[5][6] on the stack in each event
  - “local” is actually “thread-local” global memory
- Alternative SHARED: sw[5][6][nthr] for all events in one GPU block
  - BUT 32 threads is 32*5*6*2(complex)*8(double)=15kb, GPU has only 48kb
  - Probably non scalable, and worse performance than local (to be understood)
- Alternative GLOBAL: gw[5][6][ndim] for all events in all GPU blocks
  - BUT this soon exhausts global memory (malloc and/or curand fail)

- *Idea was to try and exploit memory coalescing (and SIMD?) in the GPU*
  - *And also to reduce the number of registers and improve performance*
  - *BUT: LOCAL seems to have better performance without added issues*

- *Clearly best to keep LOCAL at the moment*
  - *Keep options in the code to do other things? (may also help in CPU SIMD?)*
  - *En passant: LOCAL “-p 16384 32 12” seems faster than “-p 2048 256 12”?*

Random number generation

- Curand can be used both on the CPU (host) and on the GPU (device)
  - It produces the same numbers in both options
  - Useful to get strict reproducibility of physics results
    - NB: the seeds are reinitialized at every new iteration of ndim events

- Currently: using the fastest curand generator, not the best for physics
  - Fastest: CURAND_RNG_PSEUDO_MTGP32
  - Best for physics (Lorenzo Moneta): CURAND_RNG_PSEUDO_MRG32K3A
  - A factor 500 in speed between the two options
  - The point is only to get the GPU machine going, and same results on CPU

Current CPP baseline

```
time ./check.exe -p 16384 32 12
***************************************
NumIterations             = 12
NumThreadsPerBlock        = 32
NumBlocksPerGrid          = 16384
---------------------------------------
FP precision              = DOUBLE (nan=0)
Momenta memory layout     = AOSOA[32]
Curand generation         = HOST (C++ code)
---------------------------------------
NumberOfEntries           = 12
TotalTimeInWaveFuncs      = 1.741161e+01 sec
MeanTimeInWaveFuncs       = 1.450968e+00 sec
StdDevTimeInWaveFuncs     = 1.619446e-03 sec
MinTimeInWaveFuncs        = 1.449295e+00 sec
MaxTimeInWaveFuncs        = 1.449451e+00 sec
---------------------------------------
ProcessID:                = 17797
NProcesses                = 1
NumMatrixElementsComputed = 6291456
MatrixElementsPerSec      = 3.613368e+05 sec^-1
***************************************
NumMatrixElements(notNan) = 6291456
MeanMatrixElemValue       = 1.394735e-02 GeV^0
StdErrMatrixElemValue     = 3.034488e-06 GeV^0
StdDevMatrixElemValue     = 7.611337e-03 GeV^0
MinMatrixElemValue        = 6.071582e-03 GeV^0
MaxMatrixElemValue        = 3.374925e-02 GeV^0
***************************************
```

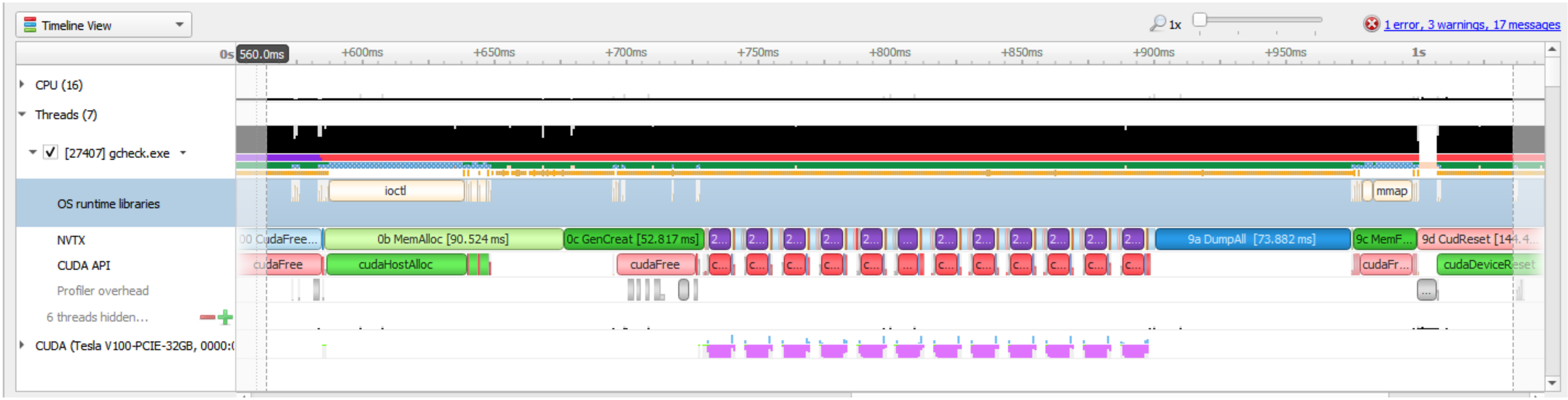
```
***************************************
 0a ProcInit : 0.000451 sec
 0b MemAlloc : 0.053021 sec
 0c GenCreat : 0.001038 sec
 1a GenSeed  : 0.000020 sec
 1b GenRnGen : 0.359951 sec
 2a RamboIni : 0.130261 sec
 2b RamboFin : 2.077701 sec
 3a SigmaKin : 17.411612 sec
 4a DumpLoop : 0.015827 sec
 9a DumpAll  : 0.046769 sec
 9b GenDestr : 0.000112 sec
 9c MemFree  : 0.001842 sec
      TOTAL : 20.098604 sec
***************************************
real    0m20.108s
user    0m20.060s
sys     0m0.045s
```

Current CUDA baseline

Our “historical” throughput is 3.6E5 against 5.0E8 :
GPU ~1500 faster than one CPU core
Here “throughput” is ONLY sigmakin + copy MEs to host:
For unweighted evt generation should add copy of momenta!

```
time ./gcheck.exe -p 16384 32 12
****************************************
NumIterations            = 12
NumThreadsPerBlock       = 32
NumBlocksPerGrid         = 16384
---------------------------------------
FP precision             = DOUBLE (nan=0)
Momenta memory layout    = AOSOA[32]
Wavefunction GPU memory  = LOCAL
Curand generation        = DEVICE (CUDA code)
---------------------------------------
NumberOfEntries          = 12
TotalTimeInWaveFuncs     = 1.257645e-02 sec
MeanTimeInWaveFuncs      = 1.048037e-03 sec
StdDevTimeInWaveFuncs    = 2.003851e-05 sec
MinTimeInWaveFuncs       = 1.035607e-03 sec
MaxTimeInWaveFuncs       = 1.040581e-03 sec
---------------------------------------
ProcessID:               = 17851
NProcesses               = 1
NumMatrixElementsComputed = 6291456
MatrixElementsPerSec     = 5.002570e+08 sec^-1
****************************************
NumMatrixElements(notNan) = 6291456
MeanMatrixElemValue      = 1.394735e-02 GeV^0
StdErrMatrixElemValue    = 3.034488e-06 GeV^0
StdDevMatrixElemValue    = 7.611337e-03 GeV^0
MinMatrixElemValue       = 6.071582e-03 GeV^0
MaxMatrixElemValue       = 3.374925e-02 GeV^0
****************************************
```

```
****************************************
 00 CudaFree : 0.192427 sec
 0a ProcInit : 0.000582 sec
 0b MemAlloc : 0.062740 sec
 0c GenCreat : 0.017667 sec
 1a GenSeed  : 0.000017 sec
 1b GenRnGen : 0.007964 sec
 2a RamboIni : 0.000124 sec
 2b RamboFin : 0.000062 sec
 2c CpDTHwgt : 0.008801 sec
 2d CpDTHmom : 0.106680 sec
 3a SigmaKin : 0.000106 sec
 3b CpDTHmes : 0.012470 sec
 4a DumpLoop : 0.022217 sec
 9a DumpAll  : 0.046581 sec
 9b GenDestr : 0.000321 sec
 9c MemFree  : 0.022609 sec
 9d CudReset : 0.061855 sec
    TOTAL : 0.563224 sec
****************************************
real    0m0.578s
user    0m0.211s
sys     0m0.344s
```


Timeline View
1x
1 error, 3 warnings, 17 messages
0s
560.0ms
+600ms
+650ms
+700ms
+750ms
+800ms
+850ms
+900ms
+950ms
1s
CPU (16)
Threads (7)
[27407] gcheck.exe
OS runtime libraries
NVTX
CUDA API
Profiler overhead
6 threads hidden...
CUDA (Tesla V100-PCIE-32GB, 0000:(
ioctl
mmap
00 CudaFree...
0b MemAlloc [90.524 ms]
0c GenCreat [52.817 ms]
2...
9a DumpAll [73.882 ms]
9c MemF...
9d CudReset [144.4...
cudaFree
cudaHostAlloc
c...
cudaFr...
cudaDeviceReset

Miscellanea

- Various other items in random order on the todo list
  - Software issues
    - Namespaces
    - Variable names and where to hardcode them (also need short names)
    - General cleanup
  - Physics validation
    - Full chain: use weights for cross section and for unweighted event generation
  - C++ code
    - Complete new memory layouts, try vectorization?
  - Complex arithmetics and memory
    - Fewer registers with cucomplex?
    - Use RRRRIIII layout rather than RIRIRIRI?
  - Helicities
    - Helicity loop as extra parallelsm dimension?
    - Helicity masking (I implemented a very simple version in cuda)