Sampling Secondary Particles in High Energy Physics Simulation on the GPU

Soon Yung Jun
Fermilab
for the Geant Vector/Coprocessor R&D Team

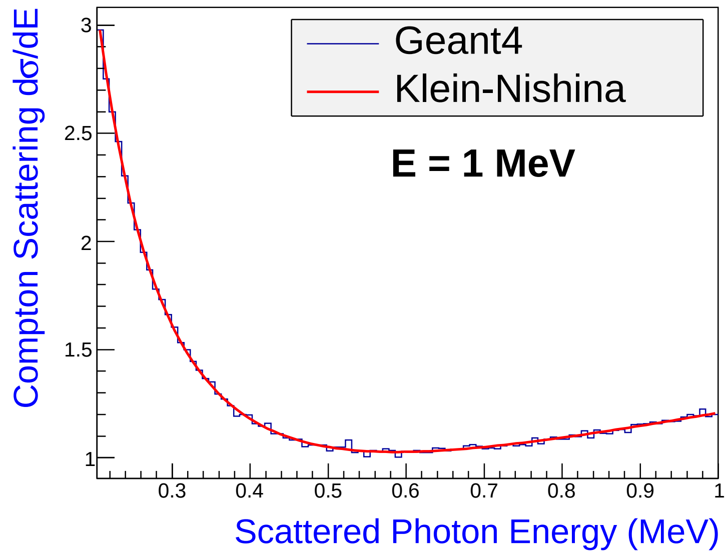
GPU Computing in High Energy Physics

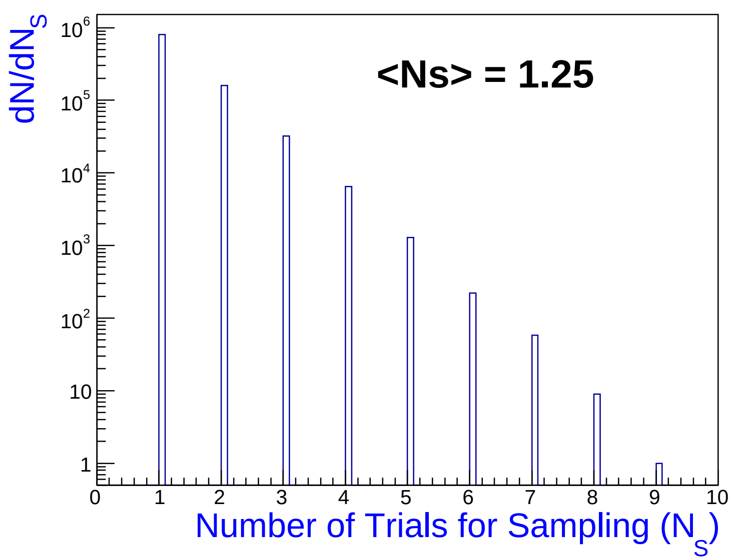
Sept. 10-12, 2014

Pisa, Italy

Introduction

- MC methods are often the only practical way to sample random variables governed by complicated probability distributions
- MC methods that are widely used in high energy physics and detector simulation
  - inverse transform (analytical solution)
  - acceptance and rejection
  - composition and rejection (Geant4)
- Compton scattering: Formula vs. Geant4
  - Klein-Nishina $d\sigma[\gamma(E,0) \to \gamma'(E', \sin\theta)]$
  - average number of trials $N_s = 1.25$, sampling efficiency $= 0.80$ at $E = 1$ MeV using a composition and rejection
- What is a problem?





Simulation with Coprocessors

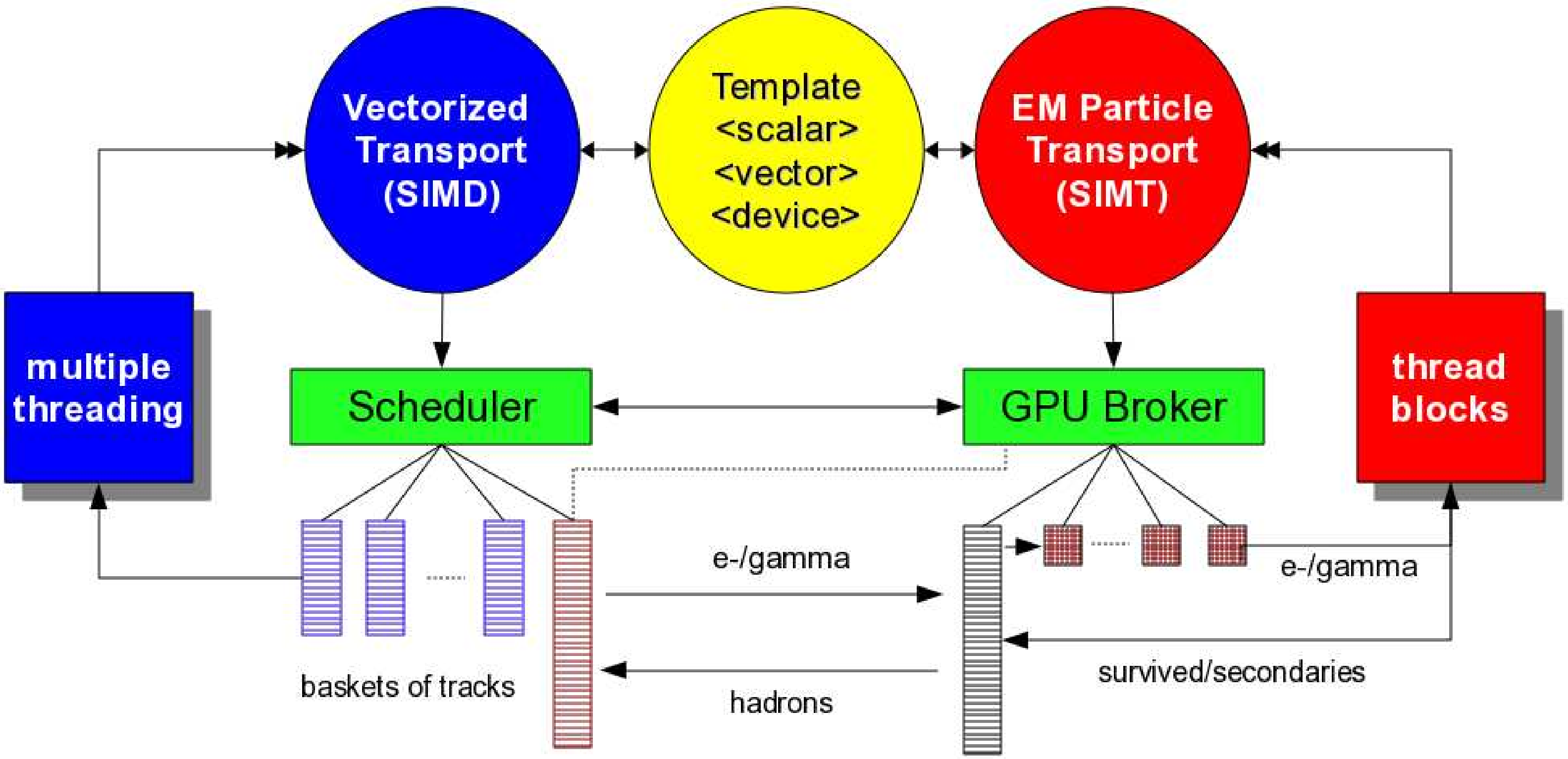
- Optimal algorithms for coprocessors (GPUs, many cores)
  - maximal instruction throughput and data locality
  - minimal branch and memory access
- Problems of sampling with "acceptance/composition and rejection" methods
  - non-deterministic loop counter (the number of trials in sampling)

```
//sample x and y for a given yo
do {
  x = f(r1);
  y = g(x,r2);
} while (y<yo);
```

  - implicit (do-while) loops: divergence, not vectorizable
  - adopt/change/develop algorithms for parallel or vector architectures

Problem Statement and Background

- Explore physics algorithms for HEP detector simulation suitable for SIMD/SIMT
- A part of R&D efforts for Geant Vector/Coprocessor Simulation
  - concurrent framework (support heterogeneous computing models)
  - vectorized geometry (bucketized particle transport)
  - physics models (tabulated/vectorized/massively parallel)



Review: Sampling Techniques

- Inverse transform (cumulative distribution): $F(a) = \int_{-\infty}^{a} f(x)dx$ for $\forall f(x)$ (p.d.f)

$$u[0,1] = F(x) \rightarrow \exists x = F^{-1}(u)$$

- Acceptance and rejection (Von Neumann): try x, accept if

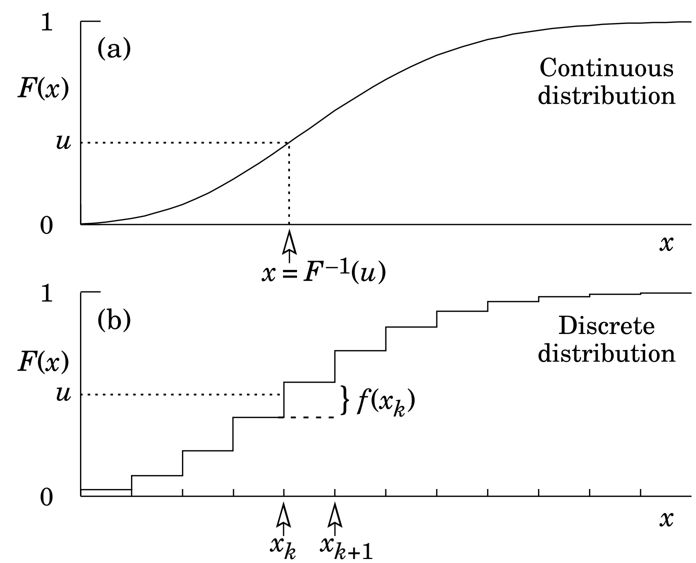
$$u[0,1] \leq \frac{f(x)}{Ch(x)}$$

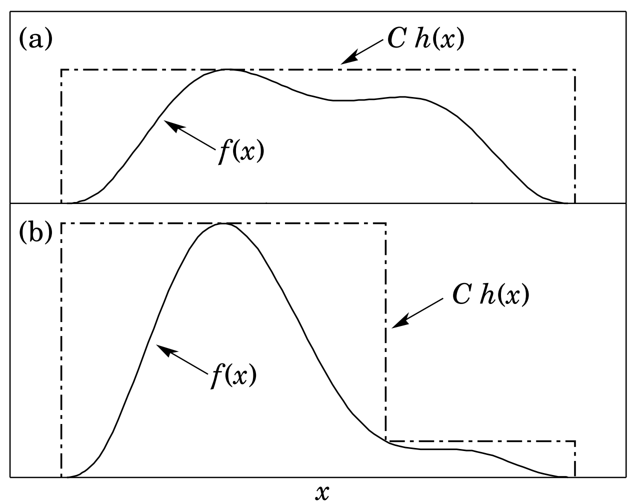
efficient if proportionality const. $C \rightarrow 1$

- Composition and rejection: decompose with density functions, $f_i(x)$ and rejection functions, $0 < g_i(x) < 1$

$$f(x) = \sum_{i=1}^{n} \alpha_i f_i(x) g_i(x)$$

- Alias method (A.J Walker, 1974)



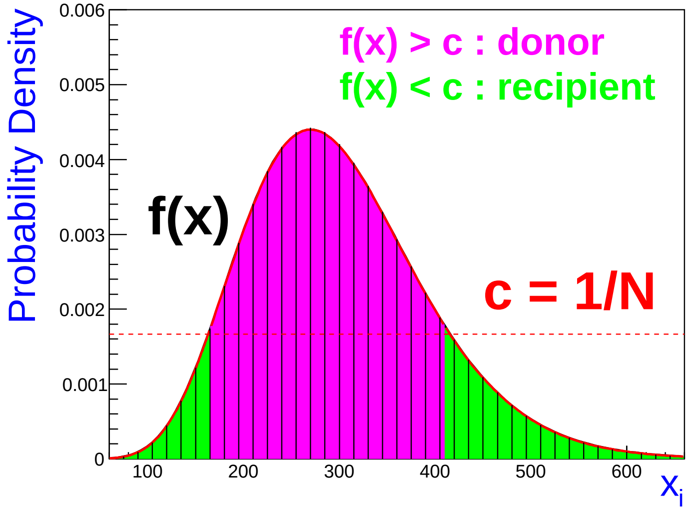


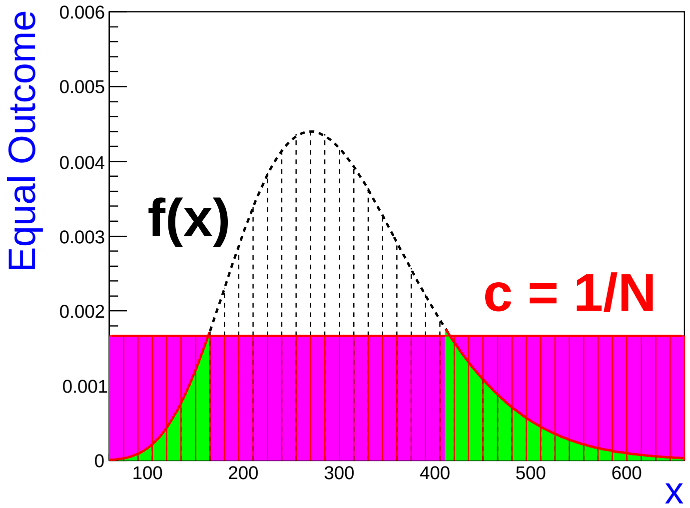
Alias Method for N Discrete Outcomes

- Recast p.d.f $f(x) \to c$, N equal probable events, each with likelihood $1/N = c$
  - alias, a[recipient]= donor
  - non-alias probability, q[i]
  - pdf, $p_j = f(x_j)$ for interpolation
- Sample x_j with random $u[0.1)$
  - uniform (Alias): $N \cdot u = i + \alpha$
    select $j = (u \leq q[i])\ ?\ i : a[i]$
    $x_d = j\Delta x$, $x_u = (j+1)\Delta x$

$$x_j = (1-\alpha)x_d + \alpha x_u$$

  - linear interpolation (Alias2): test
    if $u'(p_j + p_{j+1}) \leq (1-u)p_j + u \cdot p_{j+1}$
    then $x = x_j$
    else $x = \alpha x_d + (1-\alpha)x_u$
- Effectively vectorized





Implementation

- Sampling methods

MC methods	Sampling	Table
Composition and rejection	do-while loop	on-the-fly calc.
Inverse transform	uniform (InverseCDF) linear (InverseCDF2)	q[double] q[double], p[double]
Alias	uniform (Alias) linear (Alias2)	a[int], q[double] a[int], q[double], p[double]

- Physics models studied and size of sampling tables

Process	Model	Secondaries	N_S	Bin Scale	Table Size
γ Compton	Klein-Nishina	e^-	1.012	linear	100×100
e^- Bremsstrahlung	Seltzer-Berger	γ	1.943	log	100×1000
e^- Ionization	Moller	e^-	2.162	linear	100×100
e^+ Ionization	Bhabha	e^-	4.934	linear	100×100

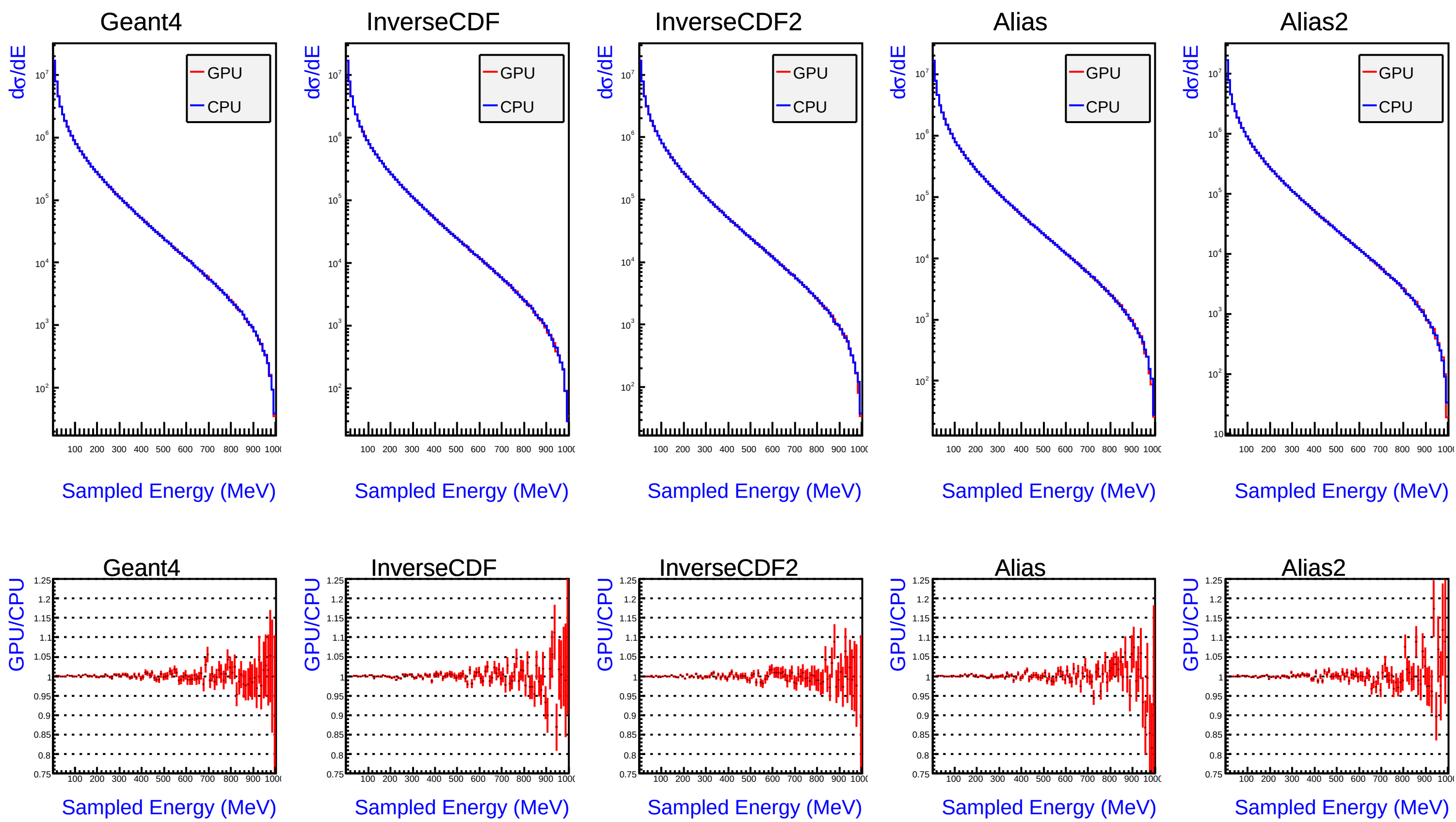
Table 1: "Composition of rejection methods" used in Geant4 EM physics models and tested for this talk with primary particle $E = \exp(-x/200)$ in $x[1, 1001]$ MeV, where N_S = average number of trials, $\epsilon = N_{\mathrm{events}}/N_{(\mathrm{total\ trials})}$

Test Setups

- Build 2-dimensional sampling tables for each physics models on CPU
  - size of double$[m][n]$ (ex: m=energy bins, n=sampling bins)
  - convert to 1D array: inverseCDF $(p,q)_{m \times n}$ and alias $(a,p,q)_{m \times n}$
  - remove/refine valley/tail ambiguity for inverseCDF (see backup, p26)
- Data transfer to GPU
  - sampling tables and random states: one time
  - primary tracks (particles): recurrent per task
- Kernel set-up (performance measurement): sample secondary particles including
  - task1: read energy of particles and write back the updated momenta
  - task2: create secondary particles and store them to the global memory (GST)
- Copy secondaries to CPU for validations
- Other considerations: impact of conditional loops, texture memory, data layout (AoS vs. SoA)

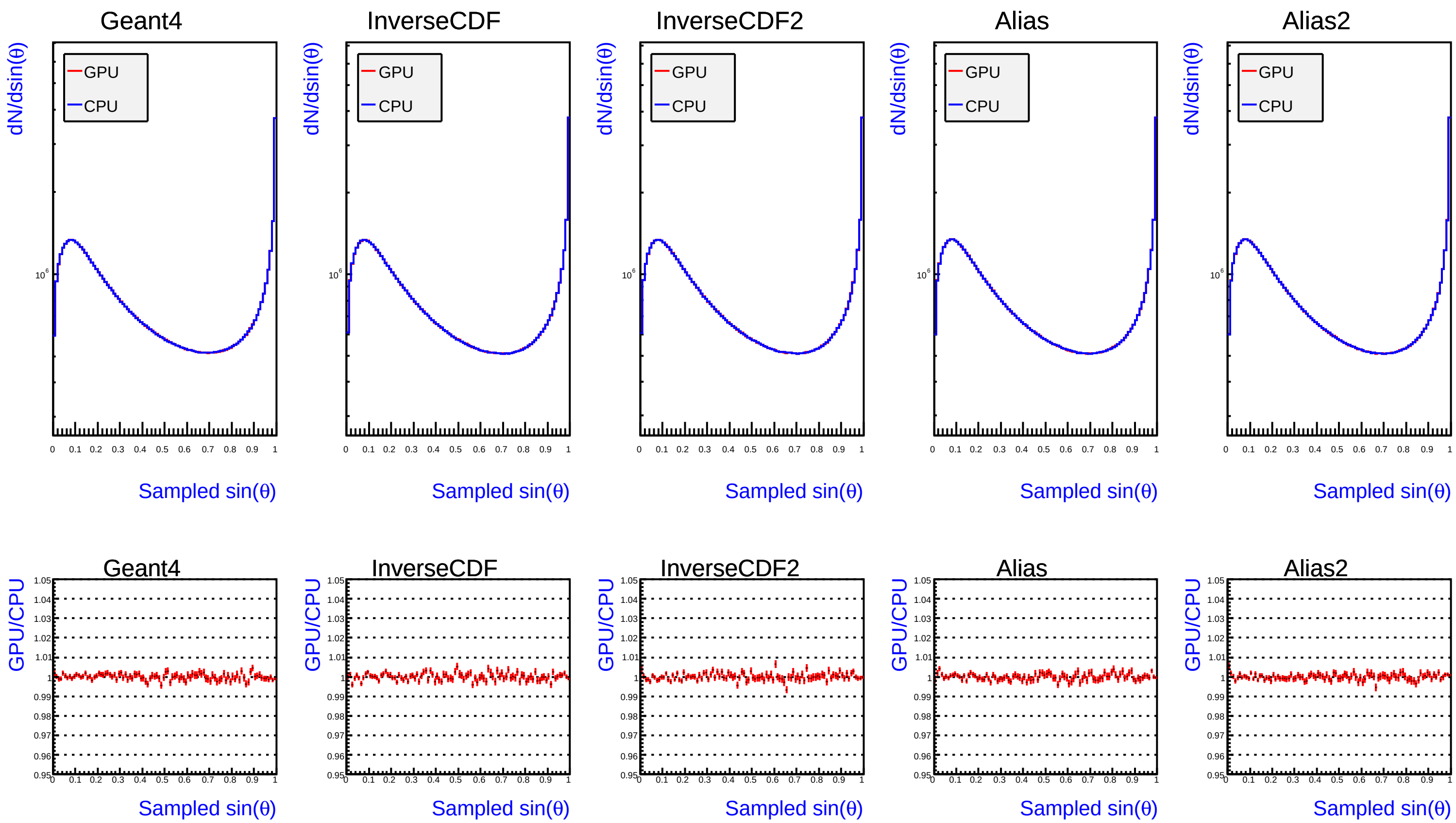
Validation: GPU vs. CPU

- Comparisons: should be identical except the random sequence
  - store secondaries on GPU and copy them to CPU (D2H)
  - compare entries of secondaries produced on GPU to those on CPU
  - ex: Compton - scattered energy (difference is a statistical fluctuation?)



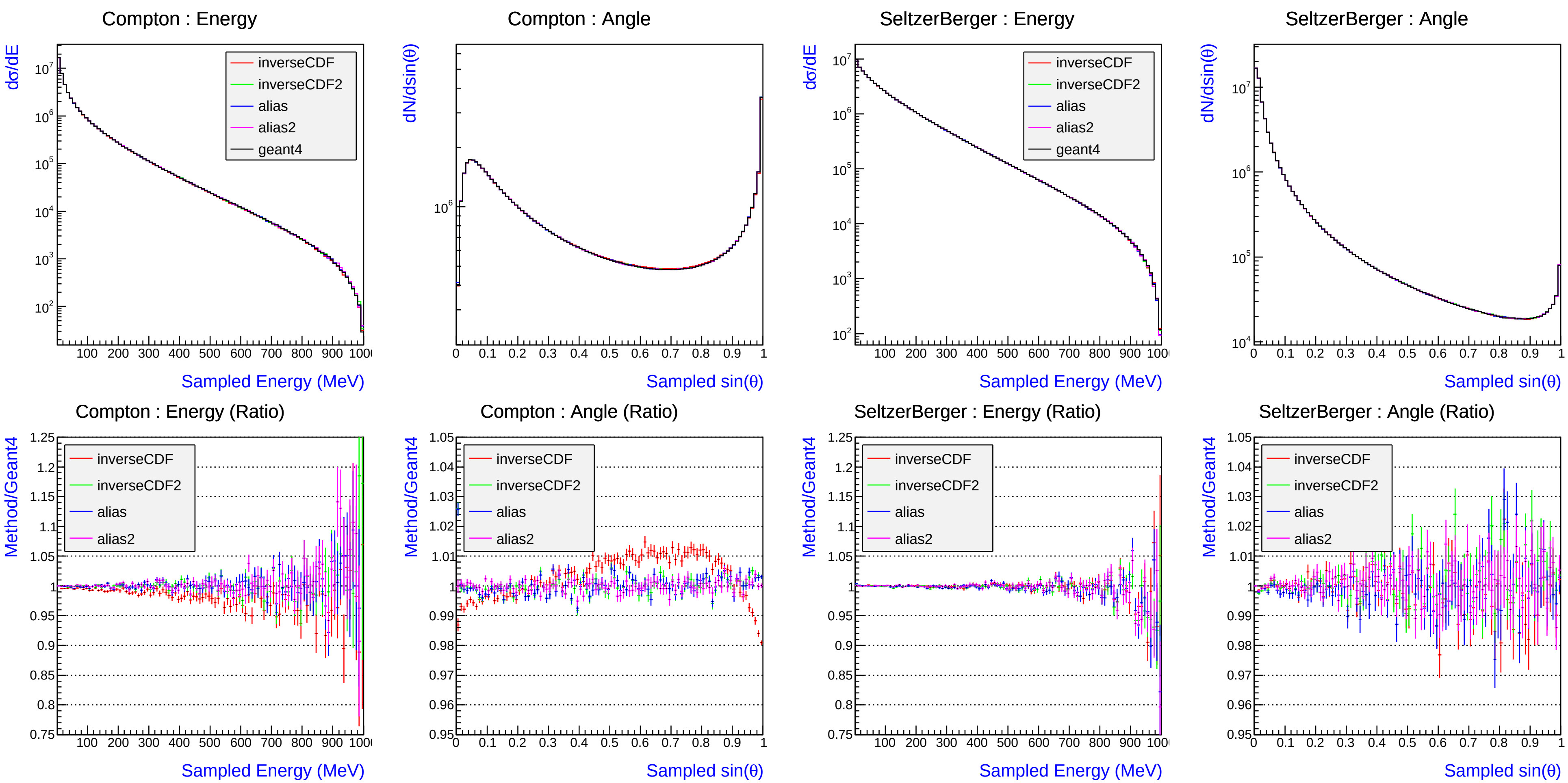
Validation: GPU vs. CPU

- Compton - scattered angle (rotated to the incident γ direction)
  - the angle is (100%) correlated to the scattered energy, $\frac{E'}{E} = \frac{m_e}{m_e + E(1-\cos\theta)}$
  - difference is less than 1.0% for the given sample (a ruler of measurement)



Method Validation: Comparisons on CPU

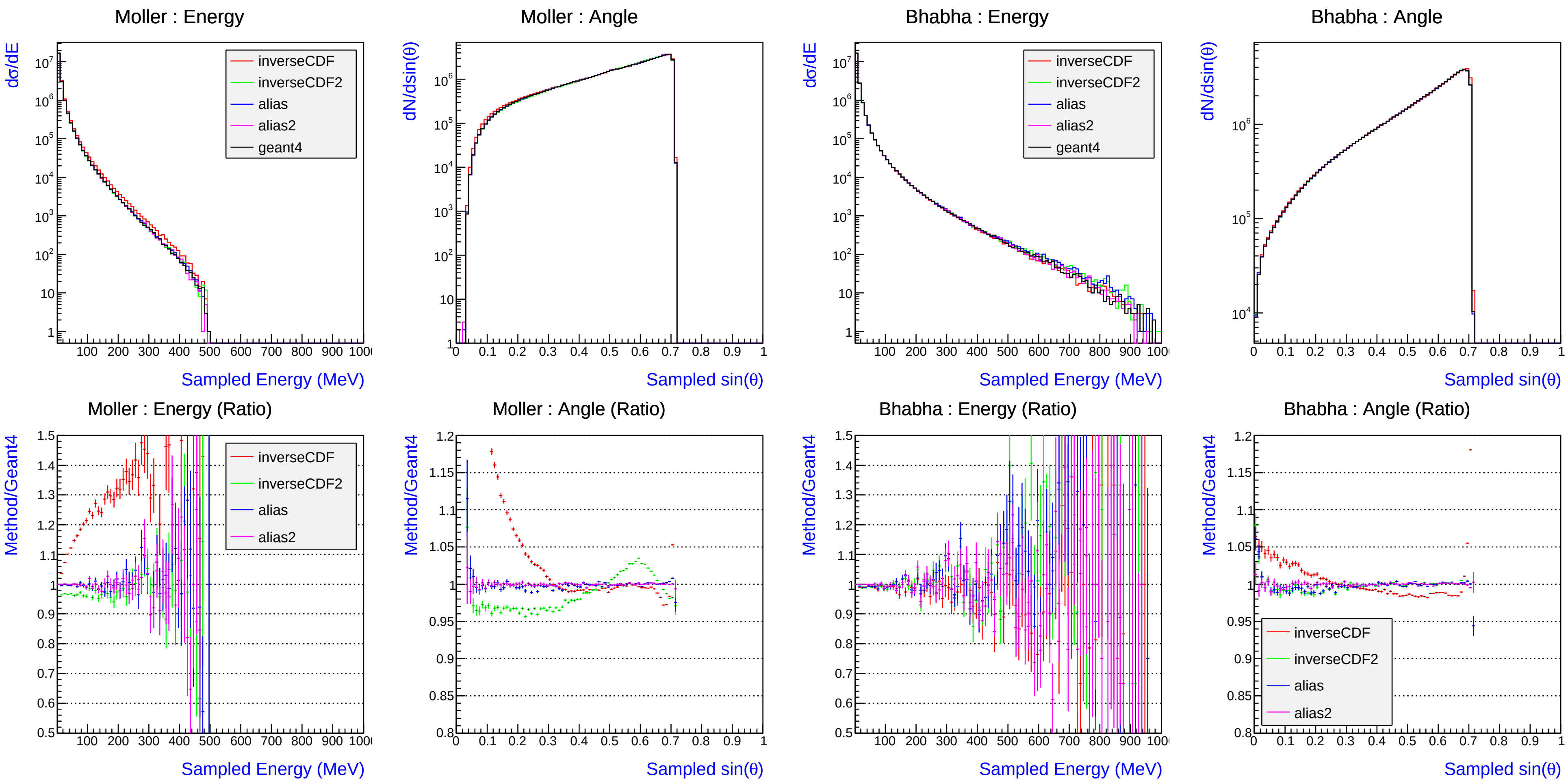
- Energy and angle of scattered particles (4 models × 5 methods)



- The InverseCDF (uniform) shows a bias in Compton Scattering

Method Validation: Comparisons on CPU

- Energy and angle of scattered particles (4 models × 5 methods)



- Alias and Alias2 show good agreements compared with Geant4 for all models

Performance Measurements

- Hardware

Host (CPU)	Device (GPU)
Intel® Xeon® E5-2620 12 cores @ 2.0 GHz	Nvidia K20(Kepler) 2496 cores @0.7 GHz

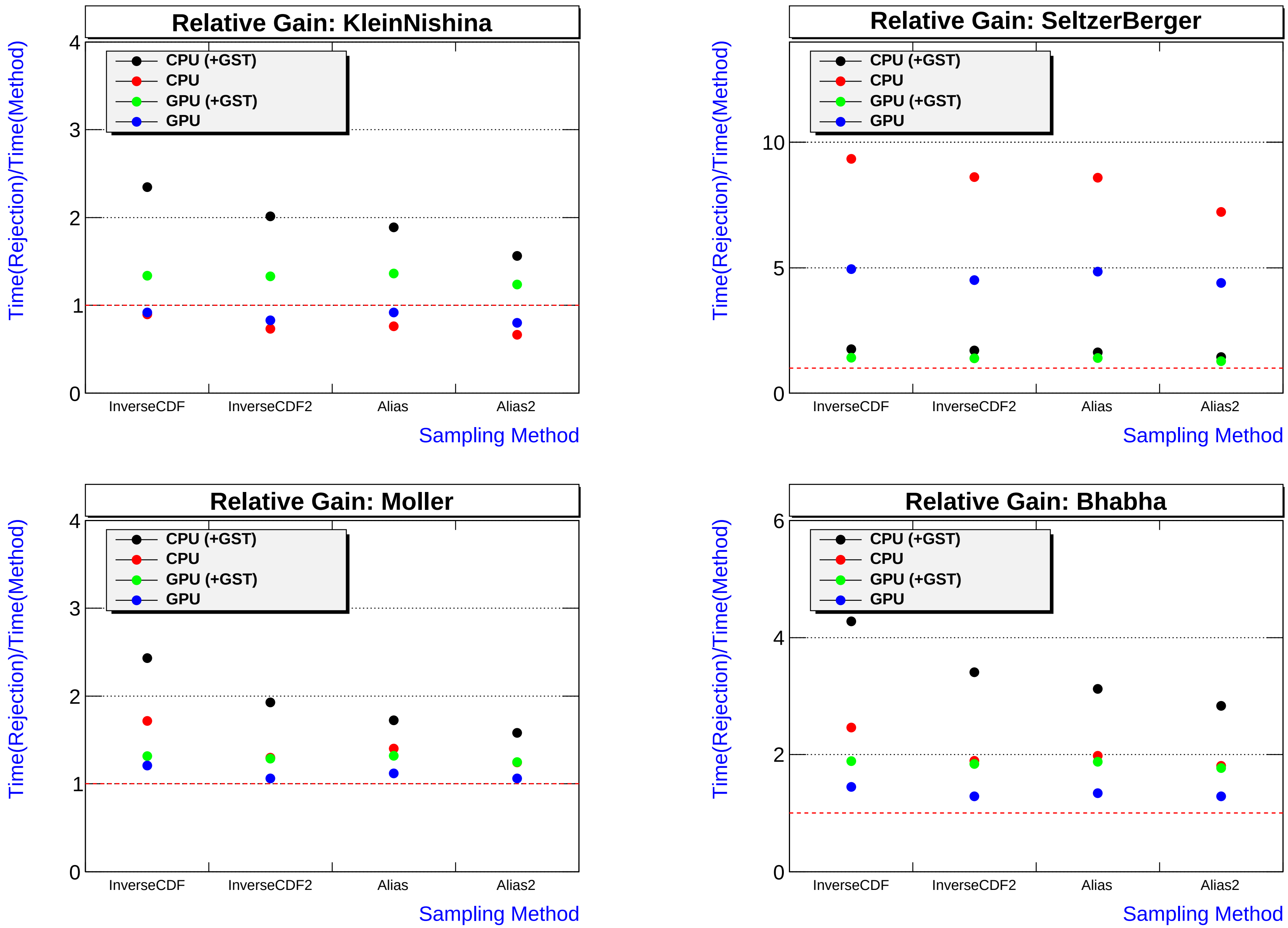
  - compilation: cuda 6.0, nvcc -arch=sm_35 –optimize 3 –use_fast_math

- Initialization
  - build tables on CPU and allocate them to GPU : $\sim$ 3 msec for q$[100 \times 100]$
  - set up random states: (blocks $\times$ threads) curandState

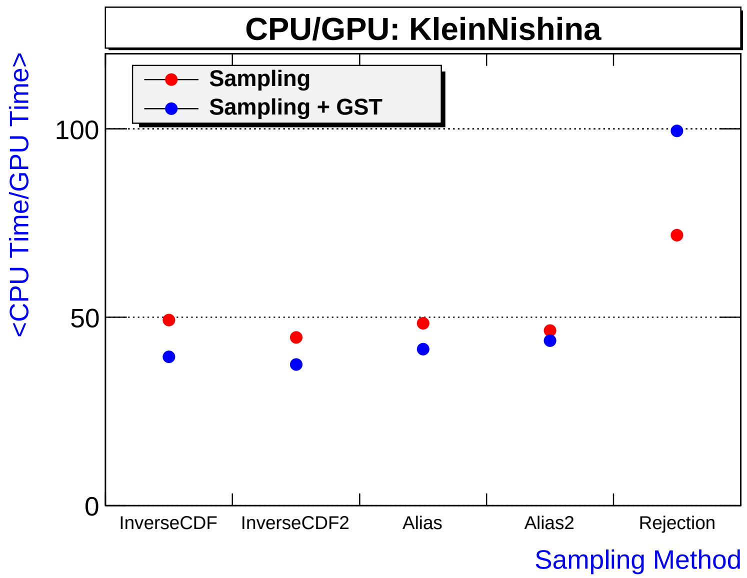
- Measurement
  - 1000 events, 79872 tracks/event, sample one secondary particle/track
  - thread execution: blocks per grid = 26, threads per block = 192
  - time in [msec]: CPU Time (1 core) vs. GPU Time (all cores)

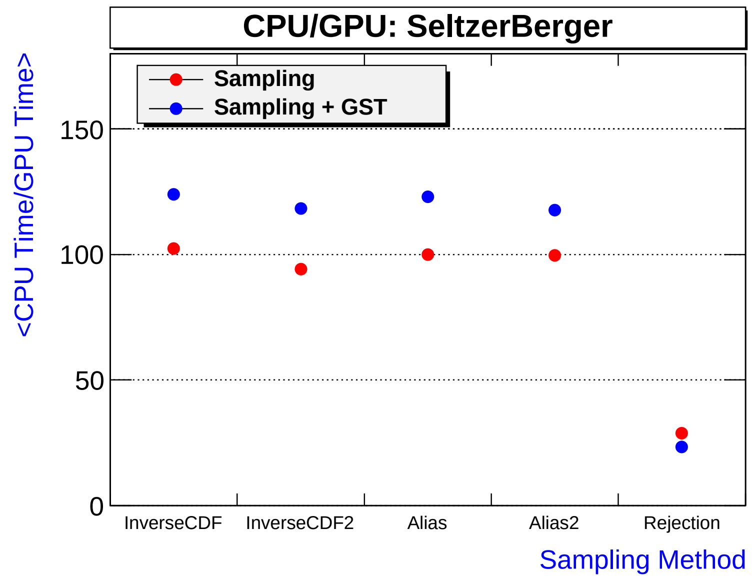
- Comparisons: relative gain, speedup (CPU/GPU) with RMS errors
  - 5 sampling methods and 4 physics models
  - 2 kernels: task1, task2 (GST)

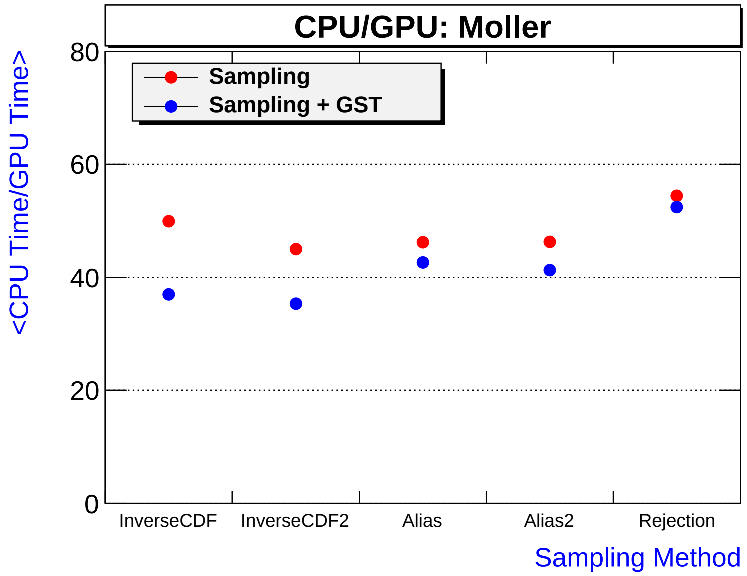
Relative Gain w.r.t Composition and Rejection

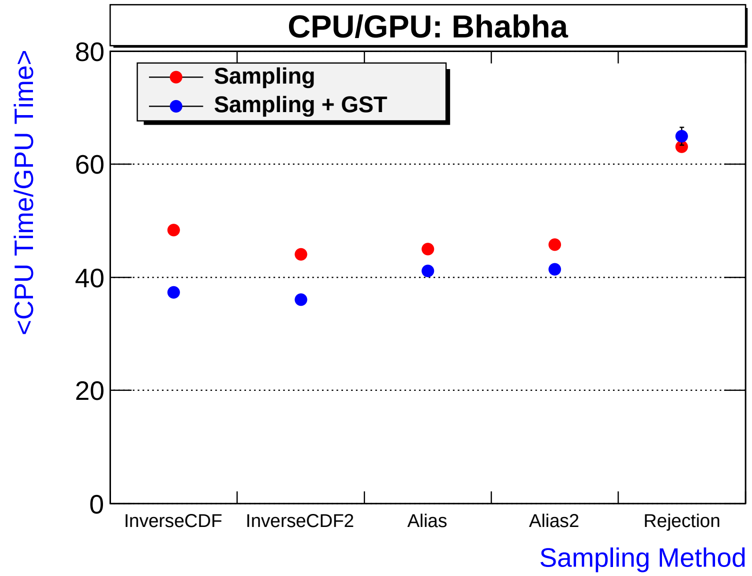


Performance: CPU/GPU



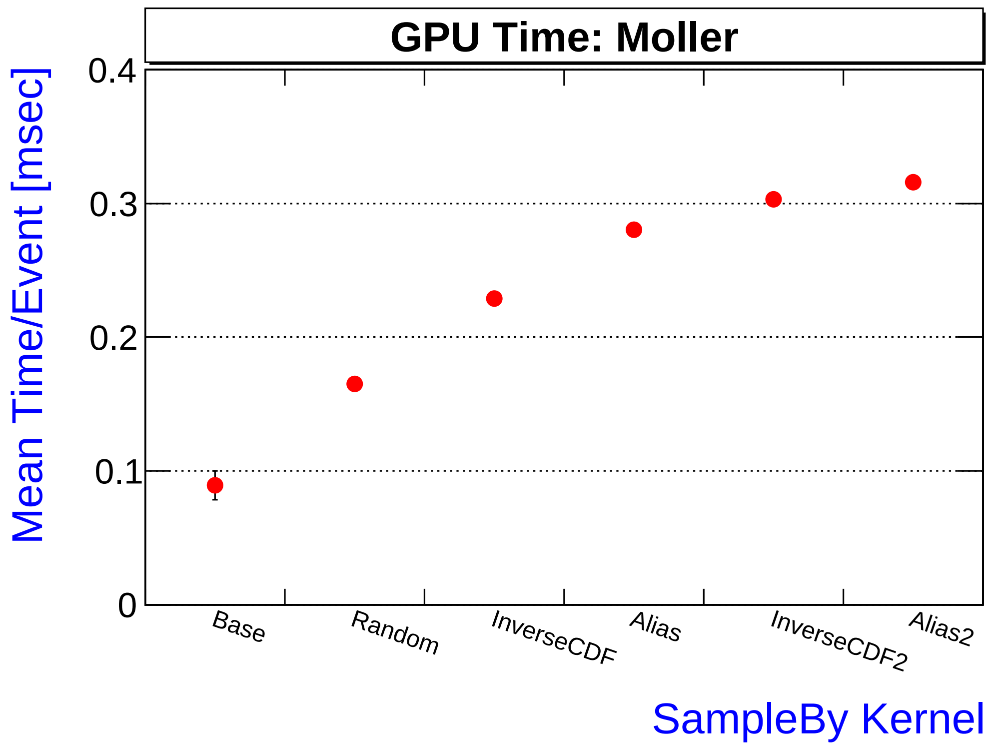






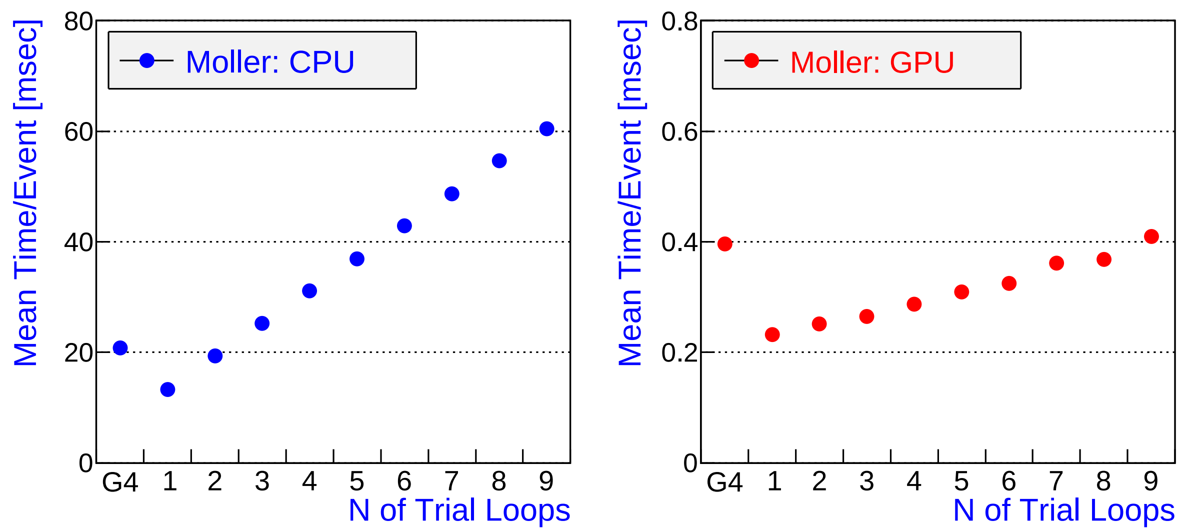
GPU Time by Kernel Components

- GPU time contribution by each step (reference: Moller InverseCDF)
  - Base: invoke objects, read/write track energy (39%)
  - Random: invoke random states (33%)
  - InverseCDF: bin search and memory access for CDF (28%)
  - Alias (+22%), InverseCDF2 (+32%), Alias2 (+38%)



Impact of Implicit Loops: Divergence

- Degradation due to divergence is not proportional to $N_{\max} \sim O(\langle N_s \rangle \log N_{\text{total}})$
  - cost for an additional trial is relatively inexpensive in GPU
  - impact of implicit loops is relatively weak when $\langle N_S \rangle$ or $N_{\max}$ is large



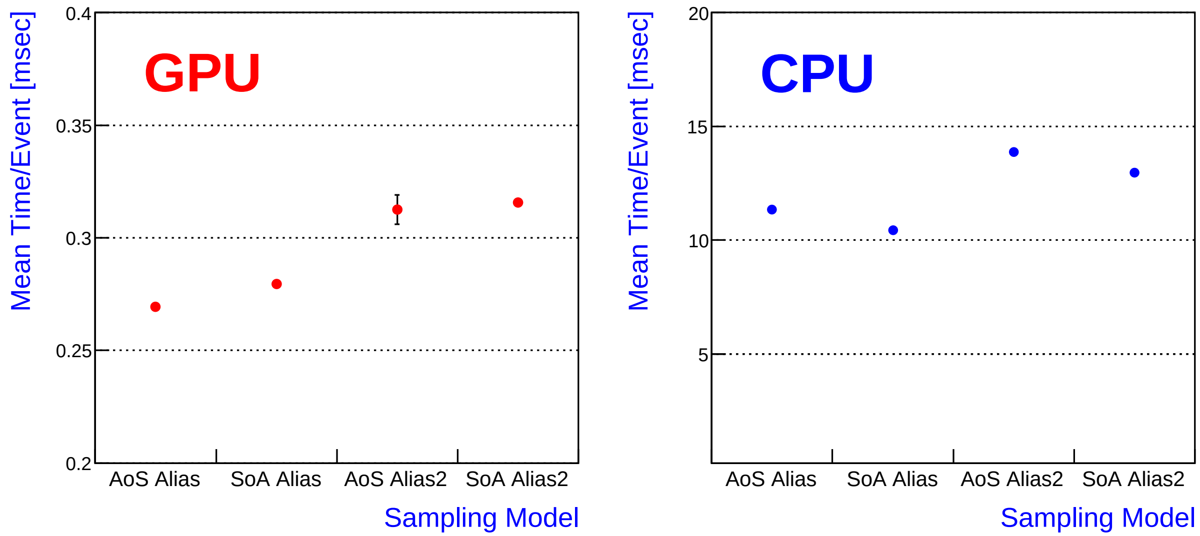
Model	G4 N_S	G4 GPU [ms]	Fixed N_S	Fixed GPU [ms]	G4/Fixed
Klein-Nishina	1.012	0.214 ± 0.002	1.0	0.209 ± 0.002	1.03
Seltzer-Berger	1.943	3.675 ± 0.061	2.0	1.866 ± 0.011	1.97
Moller	2.162	0.393 ± 0.007	2.0	0.243 ± 0.002	1.62
Bhabha	4.934	0.552 ± 0.014	5.0	0.273 ± 0.001	2.03

SoA vs. AoS: Alias and Alias2

- AoS vs. SoA:

```
struct AoS {
  G4int    a;
  G4double p;
  G4double q;
};
AoS  *aos;
```

```
struct SoA {
  G4int    *a;
  G4double *p;
  G4double *q;
};
SoA  soa;
```

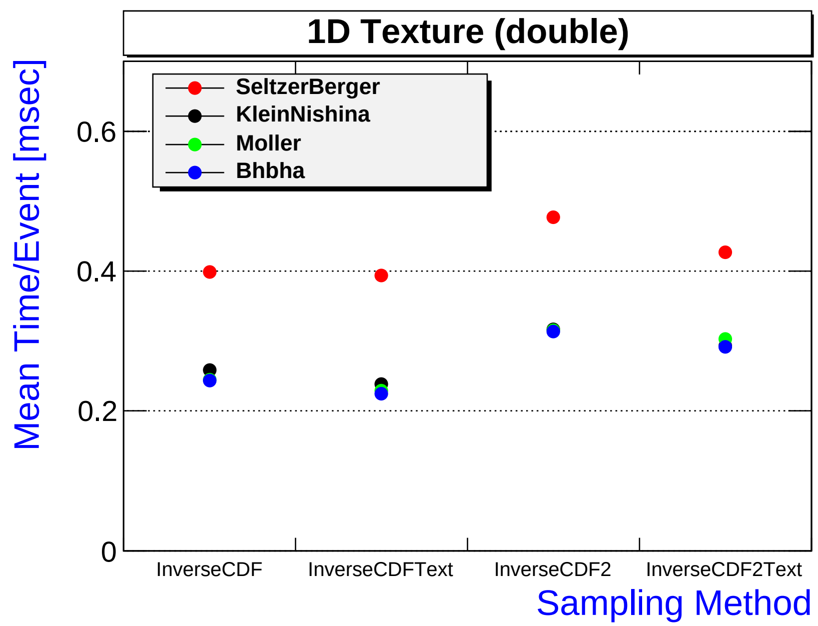


  – GPU: SoA is worse than AoS (uncoalesced memory access by each thread for random array lookups)
  – CPU: SoA is better than AoS ($\sim 20\%$)

Texture

- 1D Texture (double) for sampling (inverse CDF, PDF) tables

```
texture<int2, 1> texInverseCDF;
static __inline__ __device__ G4double FetchToDouble(texture<int2, 1> T, int i)
{
  int2 v = tex1Dfetch(T,i);
  return __hiloint2double(v.y, v.x);
}
```



  – 10-20% improvement (use int2 and cast it to double, no linear filtering)

Conclusion

- Evaluated performance of different MC methods to sample secondary particles produced by EM physics processes
  - performance degradation due to the thread divergence (with composition and rejection method) is not as significant as expected
  - inverse transform (inverse CDF method) or alias method improve computing performance in both CPU and GPU and may replace the conventional composition and rejection method
  - alias methods show no bias in sampled distributions for EM models tested and are suitable for both parallel and vector architectures

- Perspective
  - adopt and develop HEP algorithms for HPC
  - GPU prototype as a task-driven (high throughput applications)
  - vector prototype as a framework-driven (vector pipeline)
  - portable codes for different architectures (code abstraction)

Acknowledgment

- Geant Vector/Coprocessor R&D Team: http://geant.cern.ch
  - John Apostolakis, Rene Brun, Federico Carminati, Andrei Gheata, Mihaly Novak, Sandro Wenzel (CERN)
  - Philippe Canal, Victor Daniel Elvira, Soon Yung Jun, Guilherme Lima (Fermilab, US)
  - Marilena Bandieramonte (Universita e INFN, IT)
  - Georgios Bitzes (University of Athens, GR)
  - Johannes Christof De Fine Licht (University of Copenhagen, DK)
  - Laurent Duhem (Intel)
  - Raman Sehgal (Bhabha Atomic Research Centre, IN)
  - Oksana Shadura (National Technical Univ. of Ukraine)

- Members of the US ASCR-HEP Collaboration
  - Azamat Mametjanov (ANL, US)
  - Boyana Norris (Univ. of Oregon, US)

Backup

Composition and Rejection Method

- Combination of the "composition" and rejection methods for the case that
  - $f(x)$ is a complicated function
  - acceptance and rejection is inefficient
- Widely used in Geant4 to sample secondary particles for a given interaction
  - to sample x from the distribution $f(x)$ which can be written as

$$f(x) = \sum_{i=1}^{n} \alpha_i f_i(x) g_i(x)$$

  - where $\alpha_i > 0$ (fraction), $f_i(x)$ (PDFs), $0 \leq g_i(x) \leq 1$ (rejection functions)
  - select i based on α_i, try x' from $f_i(x)$ and calculate $g_i(x')$
  - $x = x'$ if $g_i(x') < u$ (random $u[0,1)$), otherwise try again
- Very powerful method for sampling x based on the distribution $f(x)$ if
  - f_i/g_i can be sampled/evaluated easily
  - $f(x)$ may not be normalized and the mean number of trials is $\sum_i \alpha_i / C$ where $\int f(x)dx = C \approx 1$

Klein-Nishina Compton Scattering

- Klein-Nishina Differential Cross Section: $\gamma(E,0) \rightarrow \gamma'(E',\sin\theta)$

$$\begin{aligned} \frac{d\sigma}{d\epsilon} &= \frac{X_o n \pi r_o^2 m_e}{E^2}[\frac{1}{\epsilon}+\epsilon][1-\frac{\epsilon\sin^2\theta}{1+\epsilon^2}] \\ \epsilon &= \frac{E'}{E} = \frac{m_e}{m_e+E(1-\cos\theta)} \end{aligned}$$

- Composition and Rejection (Geant4)

$$\begin{aligned} f(\epsilon) &= [\frac{1}{\epsilon}+\epsilon] = \sum_{i=1}^{2}\alpha_i f_i(\epsilon) \\ g(\epsilon) &= [1-\frac{\epsilon\sin^2\theta}{1+\epsilon^2}] \end{aligned}$$

decompose $f(\epsilon)$ with $\epsilon_o = 1/(1+2E/m_e)$ for the minimum γ energy as

$$\alpha_1 \times f_1(\epsilon) = \ln(1/\epsilon_o) \times \frac{1}{\epsilon\ln(1/\epsilon_o)}, \quad \alpha_2 f_2(\epsilon) = \frac{1}{2}(1-\epsilon_o^2) \times \frac{2\epsilon}{(1-\epsilon_o^2)}$$

Inverse Probability Distribution (Technical Details)

- Building Inverse PDF, $\{F^{-1}(u)[k] \mid k = 0, N\}$
  - find the bin index j_k for $F^{-1}(u)[k]$ searching $F(x)[k]$ until $\Delta u \times k < F(x)[j]$
  - build $F^{-1}(u)[k] = \Delta x[(j_k - 1) + \delta j]$ taking into account the spread in $[j_k - 1, j_k]$ (linear interpolation for the same j over k)

```
//linear interpolation within the interval with the same index
unsigned int last, cnt, lcnt;
last = cnt = lcnt = 0;

for(int i = 0; i < n ; ++i) {
  if(ip[i] > last) {
    for(int j = 0 ; j < (cnt-lcnt) ; ++j ) {
      q[j+lcnt] = dy*(last + (1.0*j/(cnt-lcnt)));
    }
    last = ip[i];
    lcnt = cnt;
  }
  ++cnt;
}
//for the last bin
q[n-1] = 1.0;
```

Alias Method (Technical Details)

- Building the alias table ($a[n]$) and the non-alias probability ($q[n]$)
  - build the discrete probability density function, $p[k] = f(x_k)$ with $x_k = k\Delta x$
  - calculate the average likelihood per equal probable event;

$$c = \frac{1}{n-1}$$

  - pick a pair $(i,j) =$(recipient,donor) such that non-zero $p[i] \leq c$ and $p[j] > c$
  - evaluate alias and non-alias probability for recipient

```
a[recipient] = donor;
q[recipient] = n*p[recipient];
```

  - update pdf

```
p[donor] = p[donor] - (c-p[recipient]);
p[recipient] = 0.0;
```

  - repeat this for O(n) iterations

Performance: Time