Fast, high-quality PRNG for heterogeneous computing

Marco Barbone, Georgi Gaydadjiev, Alexander Howard, Wayne Luk, George Savvidy, Konstantin Savvidy, Andy Rose, Alexander Tapper

m.barbone19@imperial.ac.uk
Monte Carlo simulations

**Figure 2:** Breakdown of estimated compute workloads in 2028 for ATLAS

Over 50% is required by Monte Carlo related workloads
Motivation

GPUs/FPGAs RNGs are:
- Slow and high-quality
- Fast and low-quality
- In the GPU case, closed-source

There is no middle-ground
MixMax RNG

It is a high-quality generator suitable for MC simulations

Quality, speed tunable based on state-size

Even small state size offers high quality numbers

It offers a seeding mechanism that guarantees no collisions between streams

Used in CLHEP & Geant4 (default)
Goal

Accelerate MixMax and compare the performance against state-of-the-art RNGs

Provide a reliable RNG for Monte Carlo simulations
FPGA-VHDL design

- VHDL-2008
- 62 SLOC
- Clear interface
- Fully pipelined, 100% duty-cycle
Experimental Setup

• Xilinx Ultrascale+ FPGA
• Vivado 2020.2
• Default settings
Performance

- Achieves 300 MHz
- Requires 550 LUTs
  - c.f. 523k available in KU15P, 1.75M available in VU13P
- This is 50% of the Mersenne Twister resources
- or 25% of the Mersenne Twister resources per output bit
GPU C++/CUDA design

- Each thread contains an RNG, state is not shared
- GPU is seeded with 128 bits, the thread id is concatenated to generate different streams
- Only 128 bits needs to be transferred to the GPU
- The user can change this behavior
- Clear C++ interface
Experimental setup

- NVIDIA 3090 Ti
- nvcc 11.8
- g++ 9.4.0
Validation

Compared against the original implementation

Test took ~3 months (server crashed due to blackout....)

Results never diverged
## Performance

### Parameters:

<table>
<thead>
<tr>
<th>Threads</th>
<th>Blocks</th>
<th>Parallelism</th>
<th>Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>984</td>
<td>125,952</td>
<td>generating 125,952 numbers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>time (ms)</th>
<th>time/it (ns)</th>
<th>% vs mixmax8</th>
<th>Throughput (GB/s)</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>Philox4_32_10</td>
<td>23144</td>
<td>10.78</td>
<td>-7.03</td>
<td>87074</td>
<td>[ref]</td>
</tr>
<tr>
<td>MRG32k3a</td>
<td>15203</td>
<td>7.08</td>
<td>-38.93</td>
<td>132555</td>
<td>[ref]</td>
</tr>
<tr>
<td>XORWOW</td>
<td>23206</td>
<td>10.81</td>
<td>-6.78</td>
<td>86841</td>
<td>[ref]</td>
</tr>
<tr>
<td>MixMaxGPU&lt;240&gt;</td>
<td>55294</td>
<td>25.75</td>
<td>122.11</td>
<td>36446</td>
<td>[ref]</td>
</tr>
<tr>
<td>MixMaxGPU&lt;17&gt;</td>
<td>30800</td>
<td>14.34</td>
<td>23.72</td>
<td>65430</td>
<td>[ref]</td>
</tr>
<tr>
<td>MixMaxGPU&lt;8&gt;</td>
<td>24895</td>
<td>11.59</td>
<td>0</td>
<td>80949</td>
<td>[ref]</td>
</tr>
</tbody>
</table>
Against Mersenne twister for GPU (MTGP32)

**Parameters**

<table>
<thead>
<tr>
<th>Threads</th>
<th>Blocks</th>
<th>Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>256.00</td>
<td>128.00</td>
<td>32,768</td>
</tr>
</tbody>
</table>

iteration = generating 32768 numbers

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>time (ms)</th>
<th>time/it (ns)</th>
<th>% vs mixmax8</th>
<th>speedup</th>
<th>Throughput (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTGP32</td>
<td>81161</td>
<td>37.79</td>
<td>1341.84</td>
<td>1</td>
<td>6459</td>
</tr>
<tr>
<td>Philox4_32_10</td>
<td>7803</td>
<td>3.63</td>
<td>38.62</td>
<td>10</td>
<td>67190</td>
</tr>
<tr>
<td>MRG32k3a</td>
<td>5092</td>
<td>2.37</td>
<td>-9.54</td>
<td>16</td>
<td>102963</td>
</tr>
<tr>
<td>XORWOW</td>
<td>7731</td>
<td>3.60</td>
<td>37.34</td>
<td>10</td>
<td>67816</td>
</tr>
<tr>
<td>MixMaxGPU&lt;240&gt;</td>
<td>15996</td>
<td>7.45</td>
<td>184.17</td>
<td>5</td>
<td>32776</td>
</tr>
<tr>
<td>MixMaxGPU&lt;17&gt;</td>
<td>6075</td>
<td>2.83</td>
<td>7.92</td>
<td>13</td>
<td>86302</td>
</tr>
<tr>
<td>MixMaxGPU&lt;8&gt;</td>
<td>5629</td>
<td>2.62</td>
<td>0</td>
<td>14</td>
<td>93140</td>
</tr>
</tbody>
</table>
Conclusion

MixMax is 14 times faster than Mersenne Twister on GPU

25% of the Mersenne Twister resources per output bit on FPGA

Seeding is easier and faster (128 bits, compared to O(KB))

MixMax has the potential to become de-facto the standard RNG on any platform

Suitable for ML & MC due to parallelism, efficiency and quality

Future work: Delivered via CLHEP or standalone repository
Backup
What is MixMax?

The MIXMAX generator is a family of pseudorandom number generators (PRNG) and is based on Anosov C-systems (Anosov diffeomorphism) and Kolmogorov K-systems (Kolmogorov automorphism).
What is MixMax?

- The MIXMAX generator is a family of pseudorandom number generators (PRNG) and is based on Anosov C-systems (Anosov-diffeomorphism) and Kolmogorov K-systems (Kolmogorov automorphism).
- It is a very good PRNG
- Used in CLHEP & Geant4 (default)