Mersenne Twister Implementation on a GPU

By Rohit Bal & Sagar Keer
CSE 704 Spring 2011
Instructor: Dr. Russ Miller
What is the Mersenne Twister?

- A pseudo-random number generator
- Developed by Matsumoto & Nishimura in 1997
- For a k-bit word length, produces uniform distribution in the range $[0, 2^k - 1]$
Properties:
- Long Period
- Efficient use of memory
- Good distribution properties
- High Performance
Mersenne Twister focuses on having an almost perfectly uniform distribution.

- Designed for statistical simulations like the Monte-Carlo simulations where uniformity plays a key role.
- In its canonical form, it is not suitable for cryptographic applications as future values can be predicted from a limited set of outputs.
Parameters:
- $w$ - word size
- $n, m$ - degree of recursion, middle term
- Also $n > m > 1$
- $r$ - separation point in $x_k^{upper} \mid x_{k+1}^{lower}$
- $a$ - bit vector, lower row of Matrix $A$
- $l, u, s, t$ - tempering shift parameters
- $b, c$ - tempering masks, bit vectors
Bit vectors are given by the recurrence relation:

\[ x_{k+n} = x_{k+m} + (x_k^{\text{upper}} \ | \ x_{k+1}^{\text{lower}}) \oplus A \]

where:

\[ x_k^{\text{upper}} \ | \ x_{k+1}^{\text{lower}} \] is the concatenation of

\( r \) most significant bits in \( x_k \) and \( w-r \) least significant bits in \( x_{k+1} \)
Matrix $A$ is a $w \times w$ matrix of the form

\[
\begin{array}{cccc}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
a_{w-1} & a_{w-2} & & \\
\end{array}
\]

\[
\begin{array}{cccc}
& & & 1 \\
& & & \ldots \\
& & & 1 \\
a_1 & a_0 & & \\
\end{array}
\]
For better distribution, transformation is applied using the tempering vector T to each bit vector, with the operations:

\[
\begin{align*}
    z &= x \\
    z &= z \oplus (z \gg u) \\
    z &= z \oplus (z \ll s) \& b \\
    z &= z \oplus (z \ll t) \& c \\
    z &= z \oplus (z \gg l)
\end{align*}
\]

where \(b, c, l, u, s, t\) are the tempering components as defined earlier.
For a position $k \geq n$, $x_k$ is the function of three preceding sequence elements $i.e.$:

$$x_k = f(x_{k-n}, x_{k-n+1}, x_{k-n+m})$$

To sum up, we get an almost perfectly uniform distribution using $n$ initial seeds.
C with CUDA
Used MTGP libraries
MAGIC system
Worked with 32-bit word size
Produced integral values
Algorithm maps well to CUDA because:

- Uses bitwise arithmetic
- Arbitrary amount of memory writes
Parameter ‘sets’ determine period

Total number of parameter sets is 128

In other words, 128 pseudorandom sequences can be generated for each period.

Maximum period is $2^{44497} - 1$ !!

However, we used only one period: $2^{23209} - 1$
Each thread uses its thread_id as a parameter (like a seed)

This parameter is used to calculate parameters defined by the MT algorithm

This guarantees randomization at thread level
Results

![Graph showing running time vs. data size for sequential and parallel algorithms. The graph plots running time in seconds on the y-axis and data size in millions (x10^6) on the x-axis. The sequential line shows a significant increase in running time as data size increases, while the parallel line remains relatively flat, indicating better performance for larger data sizes.](image-url)
References

- Makoto Matsumoto, Keio University/Max-Planck_Institut_fur_Mathematik; TakujiNishimura, Keio University.
- NVIDIA Sample Code for MT
Thank you!