Using GPU as Hardware Random Number Generator

Taeill Yoo, Yongjin Yeom

Department of Financial Information Security, Kookmin University, South Korea

Abstract

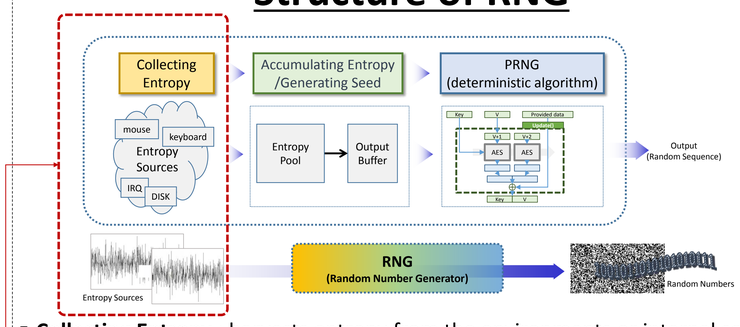
Random number generator (RNG) is expected to provide unbiased, unpredictable bit sequences. RNG collects entropy from the environments and accumulates it into the pool. Then from the entropy pool, RNG generates seed with high entropy as input to the cryptographic algorithm called pseudo-random bit generator.

Since the lack of entropy sources leads the output random bits predictable, it is important to harvest enough entropy from physical noise sources. Here, we show that we can harvest sufficient entropy from the race condition in parallel computations on GPU. According to the entropy estimations in NIST SP 800-90B, we measure the entropy obtained from NVIDIA GPU such as GTX690 and GTX780. Our result can be combined with high speed random number generating library like cuRAND. To sum up, GPU can be used as hardware random number generator with physical entropy source.

What is RNG?

- Random number generator (RNG) produces sequences of random numbers. (An ideal model is a fair coin toss)
- Pseudo-random number generator (PRNG) is a deterministic algorithm to generate random sequences (possibly a part of RNG).
- Applications: cryptography, computer simulation, game, etc.
- Requirements: unbiased (the same number of 0 and 1), unpredictable, etc.

Structure of RNG



- **Collecting Entropy** : harvests entropy from the environments or internal sources
- **Entropy Pool**: accumulate s and maintain entropy using the pool
- **PRNG**: outputs random sequences by the deterministic algorithm

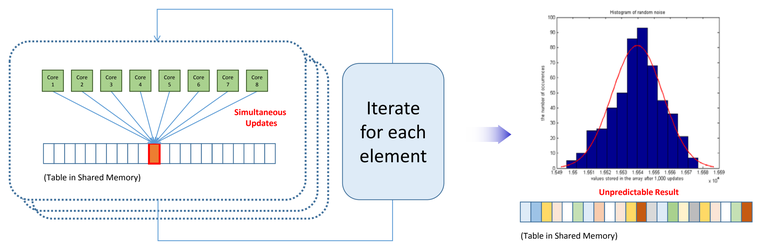
Practical RNG:

- PRNG can be chosen among ISO standard algorithms such as CTR_DRBG, HASH_DRBG, etc.
- **The main difficulty lies in collecting entropy (No standards available).**

Entropy Harvesting in GPU[1]

Race Condition during parallel computation on GPU:

- When two or more cores try to update shared resources, a race condition occurs inevitably and brings about an unexpected result.
- In general, it is important to avoid race conditions in parallel computing.
- However, we raise race condition intentionally so as to collecting entropy from the uncertainty.



CUDA Code for Collecting Entropy

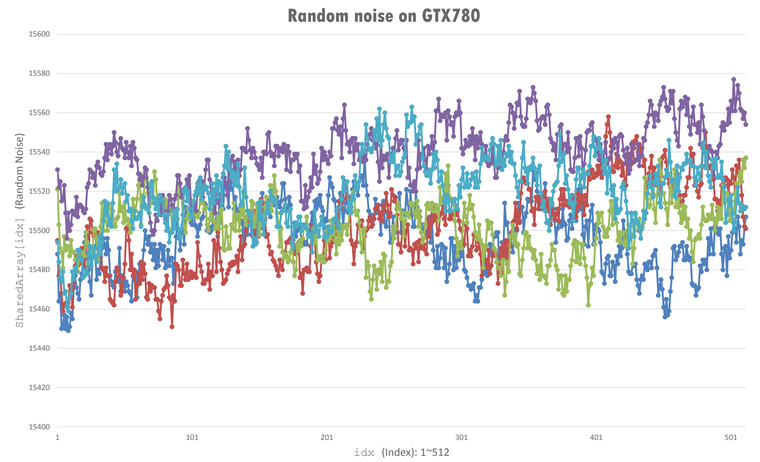
CUDA Source Code: Generating random noise using race conditions in GPU

```
/* Kernel function generating random noise */
__global__ void
RaceCondition(int *devArray, int nSize, int nIteration)
{
     int tid = threadIdx.x; //get thread ID
     devArray[tid] = 0; //initialization of array in global memory
      __shared__ int sharedArray[ARRAY_SIZE]; //(default)ARRAY_SIZE = 512
      sharedArray[tid] = devArray[tid]; //initialize shared memory
      __syncthreads(); //confirm the initialization

      /* Update shared memory that gives rise to Race condition */
     for(int i=0; i<nIteration; i++) {
       for(int j=0; j<nSize; j++) {
         sharedArray[j]++;
       }
     }
     __syncthreads();
     devArray[tid] = sharedArray[tid]; //copy to global memory
     __syncthreads();
}
```

Experiments

- **GPU:** GTX780 (We successfully run the same experiments on 610M & 690, too)
- **Array Size:** 512 elements
- **The number of iterations:** 1,000
- **The number of Threads:** 512
- **The number of experiments:** 5 (colored lines in the graph below)



Entropy Estimation by NIST 800-90B[2]

Entropy Source	Estimated Entropy	Sample Size	Data Range		Entropy Per Bit	Reference
			Min	Max		
Wireless(LQI)	0.47	8 bits	-	-	0.059	Hennebert et al. [3]
Packet Payload	2.8	320 bits	-	-	0.009	
Accelerometer X	0.22	9 bits	1	489	0.024	Hennebert et al. [4]
Accelerometer Y	0.42	9 bits	1	1024	0.047	
Accelerometer Z	0.36	9 bits	1	11	0.040	
Vibration sensor	0.17	16 bits	44	1018	0.011	
Magnetic Sensor	0.62	16 bits	9	216	0.039	
GTX 690	**0.50**	**4 bits**	**15222**	**15296**	**0.125**	**Our results**
GTX 780	**0.60**	**4 bits**	**15529**	**15614**	**0.150**	

Conclusion and Future Work

Collecting sufficient entropy for cryptographic module is challenging particularly for software module. We have shown that we can make use of GPU as entropy source and performed entropy estimation according to new methodology NIST 800-90B. We are planning to implement remaining parts of RNG on GPU including cryptographic algorithms so that GPU works as completely independent RNG containing entropy source.

References

[1] Y. Yeom, Generating Random Numbers for Cryptographic Modules Using Race Conditions in GPU, GDC 2012, Springer CCIS 351, pp. 96-102 (2012)

[2] E. Barker, J. Kelsey, Recommendation for the Entropy Sources Used for Random Bit Generation. NIST Draft Special Publication 800-90B (2012)

[3] C. Hennebert, H. Hossayni, C. Lauradoux, The entropy of wireless statistics, European Conference on Networks and Communications (2014)

[4] C. Hennebert, H. Hossayni, C. Lauradoux, Entropy harvesting from physical sensors, Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks, pp. 149-154 (2013)

Acknowledgements

1. This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (No. NRF-2014M3C4A7030648)
2. This research was partially supported by BK21PLUS through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (Grant No. 31Z20130012918)