CURAND USER GUIDE

3/30/12

CURAND LIBRARY

- provides facilities to generate high-quality pseudorandom and quasirandom numbers

TWO COMPONENTS

- host library /include/curand.h
  - similar to any other CPU library
  - random numbers can be generated on GPU or CPU

Device generation:
- calls to the library happen on the host
- actual work occurs on the device
- the resulting numbers are stored in global mem (on GPU)

Host generation - all work is done on the CPU

- device library /include/curand_kernel.h
  - includes device functions for setting random number generator states and generating sequences of rands.
  - user written kernels may call these device functions
  - it allows random numbers to be generated and immediately consumed by user kernels, without requiring the random numbers to be written to and then read from global device memory

HOST API OVERVIEW

The normal sequence of operations is as follows.

1. create a new generator of desired type with
   curandCreateGenerator()
2. Set the generator options (seed, offset, order)
3. allocate memory on the device with cudaMalloc()
4. generate random numbers with curandGenerate()
   (or other functions)
5. use the results
6. if needed, generate more random numbers
7. Clean up with curandDestroyGenerator()

Notes:

- to generate random numbers on the CPU, step 1 changes to curandCreateGeneratorHost() and step 3 requires memory allocation on the host.

- it is legal to create several generators at the same time. Multiple generators are independent.

GENERATOR TYPES

curandCreateGenerator()

Five types, two categories

CURAND_RNG_PSEUDO_XORWOW
CURAND_RNG_PSEUDO_MRG32K3A
CURAND_RNG_PSEUDO_MTGP32

CURAND_RNG_QUASI_SOBOL32 (64)
CURAND_RNG_QUASI_SCRAMBLED_SOBOL32 (64)

GENERATOR OPTIONS

SEED

- 64-bit integer that initializes the starting state

OFFSET

- parameter used to skip ahead in the sequence
  (not avail for MTGP)

ORDER

- parameter used to specify how the results are ordered in mem.

CURAND_ORDERING_PSEUDO_DEFAULT

- BEST
- SEEDED ???

FUNCTIONS

curandGenerate() – 32 bit unsigned int

curandGenerateUniform() ~ $U(0,1)$

curandGenerateNormal() ~ $N(\mu, \sigma^2)$

curandGenerateLogNormal()

curandGenerateUniformDouble()

curandGenerateNormalDouble()

etc.

```
int main(int argc, char *argv[])
{
    size_t n = 100;
    size_t i;
    curandGenerator_t gen;
    float *devData, *hostData;

    /* Allocate n floats on host */
    hostData = (float *)calloc(n, sizeof(float));

    /* Allocate n floats on device */
    CUDA_CALL(cudaMalloc((void **)&devData, n*sizeof(float)));

    /* Create pseudo-random number generator */
    CURAND_CALL(curandCreateGenerator(&gen,
                CURAND_RNG_PSEUDO_DEFAULT));

    /* Set seed */
    CURAND_CALL(curandSetPseudoRandomGeneratorSeed(gen,
                1234ULL));

    /* Generate n floats on device */
    CURAND_CALL(curandGenerateUniform(gen, devData, n));

    /* Copy device memory to host */
    CUDA_CALL(cudaMemcpy(hostData, devData, n * sizeof(float),
        cudaMemcpyDeviceToHost));
```

DEVICE API OVERVIEW

__device__ void curand_init (seed, sequence, offset, state)

- this function sets up an initial state allocated by the user. using the seed, seq. number and offset within sequence)

- same seed ⇒ same state and same sequence

Recommendations:

- each experiment should be assigned a unique seed.
- within an experiment, each thread of computation should be assigned a unique sequence number

FUNCTIONS

```
__device__ float curand_uniform (state)
__device__ float curand_normal (state)

__device__ double curand_uniform_double (state)
etc.
```

PERFORMANCE

- all these generators fail (occasionally) the BigCrush tests