S4790: Numerical Integration in CUDA

Carl Ponder, Ph.D.
Climate / Weather / Ocean codes performance
NVIDIA Developer Technology
Topics

- Numerical approaches to Integral approximation
  - Grid Construction
  - Monte Carlo
  - Transforms (Gaussian Quadrature, Fast-Fourier Transform, Discrete Cosine Transform etc.)

- Using the cuRand library for random numbers

- Parallel sum-reduction of partial sums
  - Array folding
  - Atomic operations

- Language Abstractions
  - CUDA
  - OpenACC
  - Thrust
  - CUB
Connection information

- For all connection information for the GTC 2014 hands-on labs, please see 

bit.ly/gtc14labs
Getting access

- Go to [nvlabs.qwiklab.com](nvlabs.qwiklab.com), log-in or create an account
SelectGtc2014 link
Find lab and click start

Numerical Integration in CUDA (GTC 2014)
Connection information

- After about a minute, you should see

Lab Connection: Please follow the lab instructions to connect to your lab

Warning: Please do not transmit any data into the AWS resources used in this lab that are not related to qwikLABS™ or the hands-on lab you are taking.

Connection

- Password: 3z5XrL27c7w
- Endpoint: ec2-50-19-18-96.compute-1.amazonaws.com
Password to your GPU Instance
Address of your GPU Instance
Numerical Integration

- Measure Area / Volume of a region in N dimensions
  (as opposed to deriving an expression for the Indefinite Integral)

- Region may not have a closed-form integral
  - Solution to a system of differential equations
  - Inverse problem
  - Polytope
  - Other algorithmic description
Examples

- Gaussian function
- Estimating Pi by integrating 1 quadrant of a unit circle
- Volume of a symmetric octahedron
Various Approaches

- Fixed tilings
- Adaptive tilings
- Monte Carlo
- Elementary Function Interpolation
Example 1
Gaussian Function
\[
\int_{-\infty}^{a} e^{-x^2} \, dx
\]

- Elementary expression doesn't have an elementary integral
- Distribution function is basis of statistical analysis
- Indefinite integral has transcendental form and is contained in math libraries (including CUDA library) as “erf”.

Gaussian Function Properties

- Definite integral can be solved in certain cases

\[ \int_{-\infty}^{\infty} e^{-x^2} \, dx = \sqrt{\pi} \]

- Indefinite case can be split into convenient form

\[ \int_{-\infty}^{0} e^{-x^2} \, dx + \int_{0}^{a} e^{-x^2} \, dx \]

- So we don't have to deal with the infinite tail

\[ \int_{-\infty}^{0} e^{-x^2} \, dx = \frac{\sqrt{\pi}}{2} \]
Gaussian Function: Integration by Tiling

\[ \int_{0}^{1} e^{-x^2} \, dx \]

- Evaluate “height” of function over regular intervals
- Add up areas of corresponding rectangles
Gaussian Function: CUDA code for Rectangles

```cpp
__global__ void ratio ( const int COUNT, const float low, const float high )
{
    __shared__ float sum [ THREADS ];
    int i;
    float delta = ( high - low ) / ( blockDim.x * blockDim.x * COUNT );
    float x = delta * ( blockIdx.x * blockDim.x + threadIdx.x ) * COUNT;

    x += delta/2;                   // Half-step into interval.

    sum [ threadIdx.x ] = 0.0;
    for ( i=0; i<COUNT; i++ ) {
        sum [ threadIdx.x ] += expf ( - x*x ) * delta;
        x += delta;
    }
}```
K520 GPU

8 Streaming Multiprocessors (SM's)

- threads per block ≤ 1024
- threads per SM ≤ 48*32
- blocks per SM ≤ 16

Efficient organization is 1 block per SM and 1024 threads per block.

Have each thread iterate over multiple work items if they are quick.

Other issues with registers & local memory per thread.
Sum folding in CUDA

- Generating the per-thread data values is relatively easy.
- The complication is in combining the results in a sum-reduction.
- This is an issue for many kinds of algorithms.

It takes $\log_2(N)$ steps to fold all the results together.
CUDA code for sum folding

```
for ( i=THREADS/2; i>0; i=i/2 ) {
    if ( threadIdx.x >= i ) return;
    sum [ threadIdx.x ] += sum [ i + threadIdx.x ];
    syncthreads ( );
}
SCRATCH [ blockIdx.x ] = sum [ 0 ];
}
__device__ float SCRATCH[BLOCKS];
```

CUDA code for Atomic add

```
atomicAdd ( cumulative, sum [ 0 ] );
```
Sum-reduction Operations in Various Abstractions

**OpenACC:**

```c
#pragma acc loop vector reduction(+:val)
for( n = nstart; n < nend; ++n )
    val += m[n] * v[colndx[n]];
```

**THRUST:**

```c
int result = thrust::reduce(data, data + 6, 1);
```

**CUB:**

```c
   cub::DeviceReduce::Sum(d_temp_storage, temp_storage_bytes, d_in, d_sum, num_items);
```

Other kinds of reductions are available, max/min reduction, generalized inner-product

Numerical issue with adding long sequences of small numbers - best to use double-precision for intermediate summations, and round afterward.
CUDA Atomic Operations

static __inline__ __device__
int atomicAdd(int *address, int val)

static __inline__ __device__
unsigned int atomicAdd(unsigned int *address, unsigned int val)

static __inline__ __device__
unsigned long long int atomicAdd(unsigned long long int *address, unsigned long long int val)

static __inline__ __device__
float atomicAdd(float *address, float val)
Rectangle and Tiling Approaches

- This naïve construction shows the convergence as the rectangles get smaller.

- But it doesn't give hard upper- and lower- bounds on the final result

- Also runs into floating-point accuracy issues with too many small rectangles.

1 samples: Integral\([0,1]\) = 0.778801
2 samples: Integral\([0,1]\) = 0.754598
4 samples: Integral\([0,1]\) = 0.748747
8 samples: Integral\([0,1]\) = 0.747304
16 samples: Integral\([0,1]\) = 0.746944
32 samples: Integral\([0,1]\) = 0.746854
64 samples: Integral\([0,1]\) = 0.746832
128 samples: Integral\([0,1]\) = 0.746826
256 samples: Integral\([0,1]\) = 0.746825
512 samples: Integral\([0,1]\) = 0.746824
If the function is well-behaved, i.e. derivative has the same sign over the interval, then you can tile the regions inside and outside the curve and know how close the run is to converging.

Singularities cause trouble even if the integral is finite, the region needs to be bounded.

Also a complex region won't converge very quickly, this is even harder to manage in multiple dimensions.
Monte Carlo Integration

- Sample the points inside a bounding-box

- The expected fraction of points under the curve is the same as the proportion of the area under the curve

- Requires that there be a bounding-box, i.e. singularities are still an issue.

- Also requires an “easy” test of whether a point is inside the region or not.
Monte Carlo issues

- Assumes a reasonably good random-number generator.
- Doesn't exploit any particular structure of the function to be integrated
- Refinements exist – Importance Sampling, Stratified Sampling, Correlated Sampling etc.
Error Analysis

Each random sample is a Bernoulli Trial:

\[ x_n = \begin{cases} 
1 & \text{if point is inside region} \\
0 & \text{if point is outside region}
\end{cases} \]

\[ a = x_1 + x_2 + \ldots + x_b \]

\[ \Pr [ x_n = 1 ] = \frac{A}{B} \]

\[ E[x_n] = \frac{A}{B} \]

\[ \text{Var}[x_n] = \frac{A}{B}(1-\frac{A}{B}) \]

\[ \Pr [a=y] = \binom{b}{a} \left( \frac{A}{B} \right)^a \left( 1-\frac{A}{B} \right)^{b-a} \]

\[ E[a] = b\left( \frac{A}{B} \right) \]

\[ \text{Var}[a] = b\left( \frac{A}{B} \right)(1-\frac{A}{B}) \]

B = Area of box
A = Area inside region

b = total number of sample points
   = number of sample points inside box
a = number of sample points inside region
Error Analysis

\[ E[a] = b(A/B) \]
\[ E[a/b] = A/B \]

\[ \text{Var}[a] = b(A/B)(1-A/B) \]
\[ \text{Var}[a/b] = b(A/B)(1-A/B)/b^2 \]
\[ = (A/B)(1-A/B)/b \]

\[ \text{std}(a/b) = \sqrt{(A/B)(1-A/B)/b} \]

So the Standard Deviation is inversely proportional to the square root of the number of sample points, independent of the dimension of the problem.

\[ B = \text{Area of box} \]
\[ A = \text{Area inside region} \]

\[ b = \text{total number of sample points} \]
\[ = \text{number of sample points inside box} \]
\[ a = \text{number of sample points inside region} \]
Contrast with Grid approach

\[ \frac{B}{b^d} = \text{cell volume} \]

Volume of boundary region is proportional to

\[ (\frac{B}{b^d})b^{d-1} = \frac{B}{b} = \frac{B}{\text{samples}^{1/d}} \]

(assuming function is relatively smooth)

Error is proportional to the volume of this region.

Note that Monte Carlo error shrinks with \( \text{samples}^{1/2} \) while grid error shrinks more slowly with \( \text{samples}^{1/d} \).
Random Algorithms

- **Monte Carlo**: Number of steps is fixed but the quality of the result is variable
  - Integration
  - Stock market simulation
  - Molecular dynamics

- **Las Vegas**: Result is fixed but number of steps is variable
  - Quicksort with random partition selection
  - Various randomized search algorithms

- **Stochastic**: Uses random numbers but doesn’t perform independent trials
  - Randomized Cache simulation

(definitions vary)
NVIDIA cuRand Library

- Generation of independent random numbers
- Various distributions implemented: Uniform, Normal, log-Normal, Poisson, Histogram sampling
- Different generation algorithms: MTGP, MRG, Sobol, XORWOW
- 32-bit and 64-bit integer and floating-point variables
- Functions available from Host-side (CPU code) and Device-side (GPU code)
- ... and adding new capabilities!
cuRand Host-Side Interface (curand.h)

- Calls made in host-side (CPU) code.
- Array of random numbers is generated in the device memory.
- Subsequent kernel-calls can read the elements.
- Need to balance the random number storage space against the kernel execution time.
Host-side generation example

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

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

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

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

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

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

/* Show result */
for (i = 0; i < n; i++) {
    printf ("%.4f ", hostData [i]);
}
printf ("\n");

/* Cleanup */
curandDestroyGenerator (gen);
cudaFree (devData);
free (hostData);
```

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand.h>
```
cuRand Device-Side Interface

- Calls are made directly from CUDA code, analogous to RAND functions in Fortran and C/C++
- Each CUDA thread calls the generator concurrently
- It is critical that the generator produce numbers that are independent across threads
- Each thread uses extra parameters to generate a different sequence from other threads:

```c
__device__ void curand_init ( unsigned long long seed, unsigned long long sequence, 
                             unsigned long long offset, curandState_t *state)
```
Device-Side Generation Example

```c
__global__ void ratio ( const int COUNT, int * cumulative )
{
  __shared__ int sum [ THREADS ];
  int total = 0;
  int i;
  curandState state;

  /* Each thread gets same seed, a different sequence number, no offset. */
  curand_init ( 0, THREADS*blockIdx.x+threadIdx.x, 0, &state );

  for ( i=0; i<COUNT; i++ ) {
    float x = curand_uniform ( & state );
    float y = curand_uniform ( & state );
    if ( x*x + y*y < 1 ) total++;
  }

  sum [ threadIdx.x ] = total;
  syncthreads ( );

  for ( i=THREADS/2; i>0; i=i/2 ) {
    if ( threadIdx.x >= i ) return;
    sum [ threadIdx.x ] += sum [ i + threadIdx.x ];
    syncthreads ( );
  }

  atomicAdd ( cumulative, sum [ threadIdx.x ] );
}
```

#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand_kernel.h>

# include <stdio .h>
# include <stdlib .h>
# include <cuda .h>
# include <curand_kernel .h>
Function Definitions (curand_kernel.h)

Same function can be used with different generators

```c
__device__   float curand_uniform ( curandStateScrambledSobol64_t* state )
__device__   float curand_uniform ( curandStateSobol64_t* state )
__device__   float curand_uniform ( curandStateScrambledSobol32_t* state )
__device__   float curand_uniform ( curandStateSobol32_t* state )
__device__   float curand_uniform ( curandStateMtgp32_t* state )
__device__   float curand_uniform ( curandStateMRG32k3a_t* state )
__device__   float curand_uniform ( curandStateXORWOW_t* state )
```

Different distributions can be derived from the same generator

```c
__device__   unsigned int curand ( curandStateXORWOW_t* state )
__device__   float curand_uniform ( curandStateXORWOW_t* state )
__device__   unsigned int curand_poisson ( curandStateXORWOW_t* state, double lambda )
__device__   float curand_normal ( curandStateXORWOW_t* state )
__device__   float curand_log_normal ( curandStateXORWOW_t* state, float mean, float stddev )
```

Generators will differ in their initialization information

```c
__device__   void curand_init ( curandDirectionVectors32_t direction_vectors, unsigned int offset, curandStateSobol32_t* state )
__device__   void curand_init ( unsigned long long seed, unsigned long long subsequence, unsigned long long offset, curandStateMRG32k3a_t* state )
```
Example 2: Estimation of Pi

- Area of circle is $\pi r^2$
- $r=1$ in this case, so area of circle is $\pi$ and NE quadrant is $\pi/4$.
- Area of NE bounding box is 1.
- Sampling this region gives an estimate of $\pi/4$.

Sample files
- estimate_pi.cu
- estimate_pi.combined.cu
Example 3: Volume of a Polytope

- Best algorithms are exponential due to the number of corner regions.
- Bounding-box can be derived using 2d linear-programming problems.
- Monte Carlo formulation is remarkably simple.
- Sample file monte_carlo.XORWOW.atomic.cu
Transform Approaches

- Approximate the data as a sum of orthogonal functions
- Use orthogonal functions that have elementary integrals:
  - polynomials for Gaussian Quadrature
  - sines & cosines for FFT
- Parallel Operations:
  - Calculate points
  - Transform into series coefficients
  - Calculate series terms
  - Sum-reduction
Next Steps

cuRand & programming docs
http://thrust.github.io/
http://nvlabs.github.io/cub/

Monte Carlo algorithm & error-analysis
http://en.wikipedia.org/wiki/Monte_Carlo_integration
http://farside.ph.utexas.edu/teaching/329/lectures/node109.html

CUDA download and user community
http://www.nvidia.com/getcuda
http://developer.nvidia.com/join
Questions?

- Numerical approaches to Integral approximation
  - Grid Construction
  - Monte Carlo
  - Transforms (Gaussian Quadrature, Fast-Fourier Transform, Discrete Cosine Transform etc.)

- Using the cuRand library for random numbers

- Parallel sum-reduction of partial sums
  - Array folding
  - Atomic operations

- Language Abstractions
  - CUDA
  - OpenACC
  - Thrust
  - CUB