CUDA

More GA, Events, Atomics

GA Revisited

- Speedup with a more computationally intense evaluation function
- Parallel version of the crossover and mutation operators
  - In this version of CUDA, no random in GPU
  - Used rand.cu, linear congruent; fairly poor random number generator
  - New kernel invocation based on POP_SIZE
    - <<<128,32>>>
    - See file online for code

Events

- Can use CUDA events to time how long something takes

```
cudaEvent_t   start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);

…
cudaEventRecord(start, 0);

// Do some work on the GPU, stream 0

cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);

float elapsedTime;
cudaEventElapsedTime(&elapsedTime, start, stop));
printf("Time: %3.2f milliseconds\n",elapsedTime);
```

Texture Memory

- Just Mentioning – requires use of some graphics-like functions for non-graphics use
  - E.g. tex1Dfetch(textureIn, offset);
  - Designed for OpenGL rendering pipeline
  - Read-Only memory but cached on-chip
  - Good for memory access patterns using spatial locality (e.g. access other items near a 2D coordinate)

Atomics

- Requires Compute Capability 1.2 or higher
  - Our Tesla supports 1.3
  - GeForce GTX TX 580 and Tesla S2070 support 2.0
- Atomics are used to manage contention from multiple processors or threads to shared resources
  - Primitives discussed earlier; e.g. Exchange instruction to implement barrier synchronization, keep memory consistent

Example: Histogram (CPU version 1)

```
#include <stdio.h>
#include <string.h>

#define SIZE 26

int main()
{
    char *buffer = "THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS";
    int histo[SIZE];  // Count of letters A-Z in buffer, where [0]=A's,
                      // [1]=B's, etc.

    for (int i = 0; i < SIZE; i++)
        histo[i] = 0;

    for (int i = 0; i < strlen(buffer); i++)
        histo[buffer[i]-'A']++;

    for (int i = 0; i < SIZE; i++)
    {
        printf("%c appears %d times.\n", ('A'+i), histo[i]);
    }

    return 0;
}
```

Longer Histogram

```
#include <stdio.h>
#include <string.h>

#define STRSIZE 26*20000      // 520000 chars
#define SIZE 26

int main()
{
    char *buffer;
    int histo[SIZE];

    printf("Creating data buffer...\n");
    // Allocate buffer
    buffer = (char *)malloc(STRSIZE);
    for (int i = 0; i < 26*20000; i++)
        buffer[i]= 'A' + (i % 26);

    for (int i = 0; i < SIZE; i++)
        histo[i] = 0;

    printf("Counting...\n");
    for (int i = 0; i < strlen(buffer); i++)
        histo[buffer[i]-'A']++;

    for (int i = 0; i < SIZE; i++)
    {
        printf("%c appears %d times.\n", ('A'+i), histo[i]);
    }

    free (buffer);

    return 0;
}
```

First Attempt – GPU Version

- Have a thread operate on each character
- 520,000 chars
  - Let's use 16250 blocks and 32 threads/block
  - Index into the string is the usual
    - blockIdx.x * blockDim.x + threadIdx.x

```
__global__ void histo_kernel(char *dev_buffer, int *dev_histo)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    char c = dev_buffer[tid];
    dev_histo[c-'A']++;
}

histo_kernel<<<16250,32>>>(dev_buffer, dev_histo);
```

FAIL – WHY?

Atomics to the Rescue

- Guarantee only one thread can perform the operation at a time
- Must compile with

  nvcc –arch=sm_11

  or

  nvcc –arch=sm_12

atomicAdd, atomicSub, atomicMin, atomicMax, atomicExch, atomicInc
See http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf

Modified Kernel

```
__global__ void histo_kernel(char *dev_buffer, int *dev_histo)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    char c = dev_buffer[tid];
    atomicAdd(&dev_histo[c-'A'],1);
}
```

Shared Memory Atomics

- Compute Capability 1.2+ allows atomics in shared memory; using 16250 blocks, 32 threads

```
__global__ void histo_kernel(char *dev_buffer, int *dev_histo)
{
    __shared__ int block_histo[SIZE];
    if (threadIdx.x < 26)
        block_histo[threadIdx.x] = 0;
    __syncthreads();

    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    char c = dev_buffer[tid];
    atomicAdd(&block_histo[c-'A'],1);

    __syncthreads();
    if (threadIdx.x < 26)
        atomicAdd(&dev_histo[threadIdx.x],block_histo[threadIdx.x]);
}
```

Zero histogram per block

Add block histogram

Add to global histogram