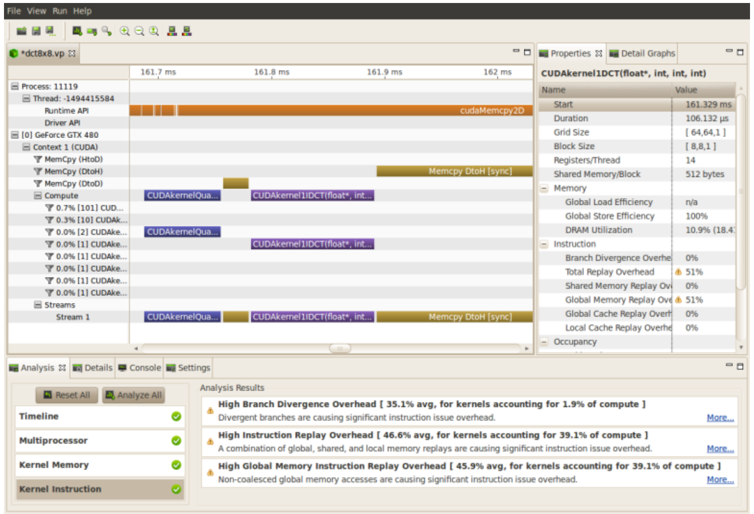
GPU Computing with CUDA

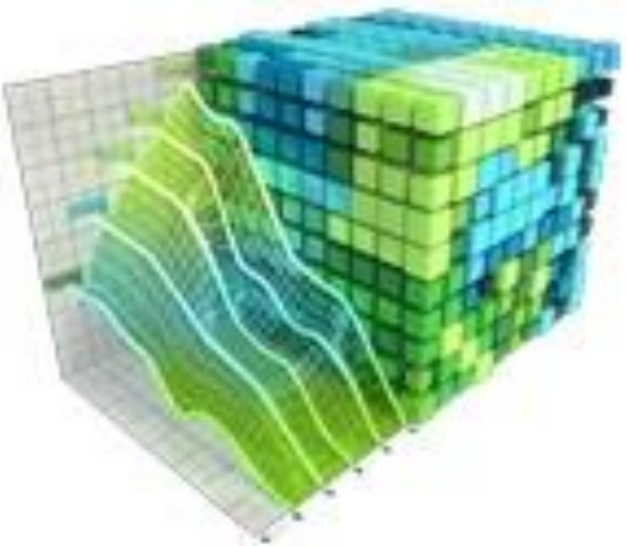
Hands-on: CUDA Profiling, Thrust

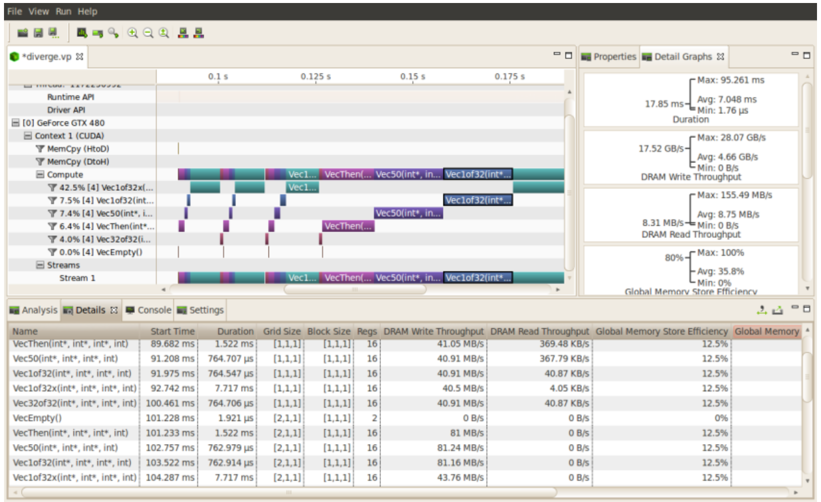
Dan Melanz & Andrew Seidl
Simulation-Based Engineering Lab
Wisconsin Applied Computing Center
Department of Mechanical Engineering
University of Wisconsin-Madison

Portland, Oregon
August 4, 2013

CUDA Profiling







CUDA Code Profiling

- We will be using the CUDA Visual Profiler to profile a matrix addition problem:

$$C = A + B$$

Programming Demo #1

CUDA Programming w/ Thrust

CUDA Programming w/ Thrust

- Thrust is a parallel algorithms library which resembles the C++ Standard Template Library (STL):
  - High-level
  - Enhances productivity
  - Allows for interoperability
  - Helps develop high-performance applications

CUDA Programming w/ Thrust

- Remember, CUDA programs have a basic flow:
  1) The host initializes an array with data.
  2) The array is copied from the host to the memory on the CUDA device.
  3) The CUDA device operates on the data in the array.
  4) The array is copied back to the host.

- This is true for Thrust, too!

Dot Product Example...

Example 2: Vector Dot Product

- Recall the dot product example from last time:
  - Given vectors a and b each with size N, store the result in scalar c

$$c = \mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \ldots + a_N b_N$$

Purpose of the exercise: use `thrust` to get it done

Example 2: Vector Dot Product

- Stage 1: The host initializes the array with data, the code looked like

```
 // Allocate host data
    float *h_A = (float *) malloc(size);
    float *h_B = (float *) malloc(size);
    float *h_C = (float *) malloc(size);
    float *dotProd_h = (float *)malloc(sizeof(float));

// Initialize the host input vectors
    for (int i = 0; i < numElements; ++i) {
        h_A[i] = rand()/(float)RAND_MAX;
        h_B[i] = rand()/(float)RAND_MAX;
    }
```

Example 2: Vector Dot Product

- Stage 1: Using Thrust, we can change the code to:

```
// Allocate the host vectors
thrust::host_vector<float> h_A;
thrust::host_vector<float> h_B;
thrust::host_vector<float> h_C;

// Initialize the host input vectors
for (int i = 0; i < N; ++i)  {
       h_A.push_back(rand()/(float)RAND_MAX);
       h_B.push_back(rand()/(float)RAND_MAX);
       h_C.push_back(0.f);
}
```

Example 2: Vector Dot Product

- Stage 2: Data copied from host to device memory; code looked like

```
// Allocate memory for the device data
   float *d_A = NULL;
   float *d_B = NULL;
   float *d_C = NULL;
   float *dotProd_d = NULL;

   cudaMalloc((void **)&d_A, size);
   cudaMalloc((void **)&d_B, size);
   cudaMalloc((void **)&d_C, size);
   cudaMalloc((void **)&dotProd_d, sizeof(float));

   // Copy the host input vectors A and B in host memory
   // to the device input vectors in device memory
   cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
   cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
```

Example 2: Vector Dot Product

- Stage 2: Using **`thrust`**, the code can be simplified:

```
// Allocate the device vectors
thrust::device_vector<float> d_A = h_A;
thrust::device_vector<float> d_B = h_B;
thrust::device_vector<float> d_C = h_C;
```

- Keep in mind that what happens under the hood is the same copy of data from the host to the device; i.e., it’s still an expensive operation

Example 2: Vector Dot Product

- Stage 3: The CUDA device operates on the data in the array, we originally have the following code:

```
// Launch the Vector Dot Product CUDA Kernel
   int threadsPerBlock = numElements;
   int blocksPerGrid =(numElements + threadsPerBlock - 1) / threadsPerBlock;
   vectorDot<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, dotProd_d, numElements);
```

Example 2: Vector Dot Product

- Stage 3: Using Thrust, we can change the code to:

```
// compute d_C = d_A * d_B (element-wise)
thrust::transform(
       d_A.begin(),
       d_A.end(),
       d_B.begin(),
       d_C.begin(), thrust::multiplies<float>());

// sum the values in d_C and put into the variable dotProd_d
double dotProd_d = thrust::reduce(
       d_C.begin(), d_C.end());
```

- Note that we do not need to specify the execution configuration

Example 2: Vector Dot Product

- Stage 4: The value is copied back to the host, code looks like:

```
// Copy the device result vector in device memory to the
// host result vector in host memory.
cudaMemcpy(dotProd_h, dotProd_d, sizeof(float), cudaMemcpyDeviceToHost);
```

- We can completely remove this step since `thrust::reduce(...)` copies this for us
- Thrust also cleans up after itself, no need to include `free(...)` or `cudaFree(...)`

Example 2: Vector Dot Product

- To compile this code:

```
$ nvcc dotProductThrust.cu
```

- To run this code:

```
$ qsub submit_example.sh
```

Programming Demo #2