Timing CPU and GPU Kernels

CISC 879 – Advanced Parallel Programming

William Killian

2013 March 05

Outline

- Timing CPU Code
  - `gettimeofday`
  - `clock_gettime`
- Timing GPU Code
  - Profiling
  - Use CPU Timers
  - Use Events

Using **`gettimeofday`** for Timing

- `gettimeofday` uses the system time for timing
- Accurate within 10us on average

Example:

```
#include <time.h>
#include <sys/time.h>

struct timeval begin, end;

gettimeofday (&begin, NULL);
// execute some arbitrary code/kernel
gettimeofday (&end, NULL);

int time = 1e6 * (end.tv_sec - begin.tv_sec) + (end.tv_usec - begin.tv_usec);
```

Using `clock_gettime` for Timing

- `clock_gettime` uses the number of cycles that have passed on the CPU for timing
- Accurate within 1ns on average (clock rate of the CPU)
- Must link with the realtime library when compiling (`-lrt`)

Example:

```
#include <time.h>
struct timespec begin, end;

clock_gettime (CLOCK_PROCESS_CPUTIME_ID, &begin);
// execute some arbitrary code/kernel
clock_gettime (CLOCK_PROCESS_CPUTIME_ID, &end);

uint64_t time = 1e9 * (end.tv_sec - begin.tv_sec) + (end.tv_nsec - 
begin.tv_nsec);
```

Using Profiling for Timing

- When running on GPUs, we can profile our code to see the total execution time of every kernel or memory transfer
- Enabled by setting `COMPUTE_PROFILE` environment variable to 1
  - `export COMPUTE_PROFILE=1 # bash`
  - `setenv COMPUTE_PROFILE 1 # csh`
- Execute your code normally
  - `./matrixMultiply`
- One or more profile logs will be generated
  - i.e. `cuda_profile_0.log` or `opencl_profile_0.log`
- Under these log files there are a few different columns:
  - Method – kernel invocation
  - GPUtime – time to run on the GPU (what we care about)
  - CPUtime – time to run on the CPU (sometimes interesting)
  - Occupancy – how much of the GPU was used for the given kernel

Using Events for Timing

- Events are special kernels that can be invoked for precise timing on the GPU
- OpenCL and CUDA have their own respective events
- On the next two slides are specific instances for OpenCL and CUDA
- Requires specific API calls to each (not generic)

OpenCL Events for Timing

Each enqueue call optionally returns an event object that uniquely identifies the enqueued command. The event object of a command can be used to measure its execution time if as detailed in Section 5.9 and illustrated in Listing 2.1. Profiling can be enabled by setting the CL_QUEUE_PROFILING_ENABLE flag in properties argument of either clCreateCommandQueue or clSetCommandQueueProperty.

```
cl_ulong start, end;

clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_END,
                        sizeof(cl_ulong), &end, NULL);
clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_START,
                        sizeof(cl_ulong), &start, NULL);

float executionTimeInMilliseconds = (end - start) * 1.0e-6f;
```

Listing 2.1 How to time code using OpenCL events

Note that the timings are measured on the GPU clock, and so are operating system–independent. The resolution of the GPU timer is approximately half a microsecond.

CUDA Events for Timing

```
cudaEvent_t start, stop;
float time;
cudaEventCreate (&start);
cudaEventCreate (&stop);
cudaEventRecord (start, 0);
kernel <<<grid, threads>>> (d_odata, d_idata, size_x, size_y, NUM_REPS);
cudaEventRecord (stop, 0);
cudaEventSynchronize (stop);
cudaEventElapsedTime (&time, start, stop);
cudaEventDestroy (start);
cudaEventDestroy (stop);
```

- Here cudaEventRecord() is used to place the start and stop events into the default stream, stream 0.
- The device will record a timestamp for the event when it reaches that event in the stream.
- The cudaEventElapsedTime() function returns the time elapsed between the recording of the start and stopevents.
- return value is expressed in milliseconds (with resolution of 0.5 us)
- timing resolution is operating-system-independent.

Questions / Comments

Perhaps a Demo?