GPU Programming Problems

Scott Grauer-Gray

Project 1

- Sample loop for project 1:

```
for (int nl = 0; nl < 4*ntimes; nl++)
{
     for (int i = 0; i < LEN; i++)
     {
          s = b[i] + c[i] * d[i];
          a[i] = s * s;
     }
     dummy(a, b, c, d, e, aa, bb, cc, 0.);
}
```

- Should not parallelize outer 'nl' loop (only 'i' loop)
  - Purpose is for 'i' loop to run multiple times for accurate timing
    - Same kernel will be called multiple times on GPU
  - dummy function on CPU is to ensure that outer loop is actually run 4*ntimes
    - May not be necessary on GPU

Project 1

- Sample loop for project 1:

```
for (int nl = 0; nl < 4*ntimes; nl++)
{
    for (int i = 0; i < LEN; i++)
    {
        s = b[i] + c[i] * d[i];
        a[i] = s * s;
    }
    dummy(a, b, c, d, e, aa, bb, cc, 0.);
}
```

- OpenCL Code:

```
//outer loop still on CPU
for (int nl = 0; nl < 4*ntimes; nl++)
{
    //run 'i' loop in parallel on GPU
    clEnqueueNDRangeKernel(command_queue, iKernel, ...)

    //synchronize so kernel completes before beginning next iteration
    clFinish(command_queue);
}
```

Program Parallelization on GPU

- Need to ensure that program can be parallelized for computation on GPU
- Cannot have dependencies between loop iterations

Program Parallelization on GPU

- i-loop below not parallelizable on GPU
  - aa[i-1] must be computed before aa[i] for each iteration
  - All iterations are parallel on GPU (in theory)
  - Not parallelizable on GPU as a result

```
for (int i= 1; i < N; i++)
{
   aa[i] = aa[i-1] + b[i] * c[1];
}
```

Program Parallelization: Data Races

- Given summation loop:
  - Sum all values from i=0 to N-1 in array 'A' w/ result in A[N]

```
a[N] = 0.0f;
for (int i= 0; i < N; i++)
{
   a[N] += a[i];
}
```

Program Parallelization: Data Races

```
a[N] = 0.0f;
for (int i= 0; i < N; i++)
{
   a[N] += a[i];
}
```

- If parallelized on GPU w/ every iteration in parallel...
  - All iterations read the initial value of a[N] in parallel
  - All iterations i add a[i] to initial a[N] and write updated value to a[N] in parallel
  - Data race between all i iterations
  - Only one a[i] value actually added to a[N]

Program Parallelization: Data Races

- Still possible to use GPU
  - One option: using atomics...GPU kernel code becomes:

```
int iVal = blockIdx.x*blockDim.x + theadIdx.x;
if ((iVal >= 0) && (iVal < N))
{
   atomicAdd(&(a[N]), a[i]);
}
```

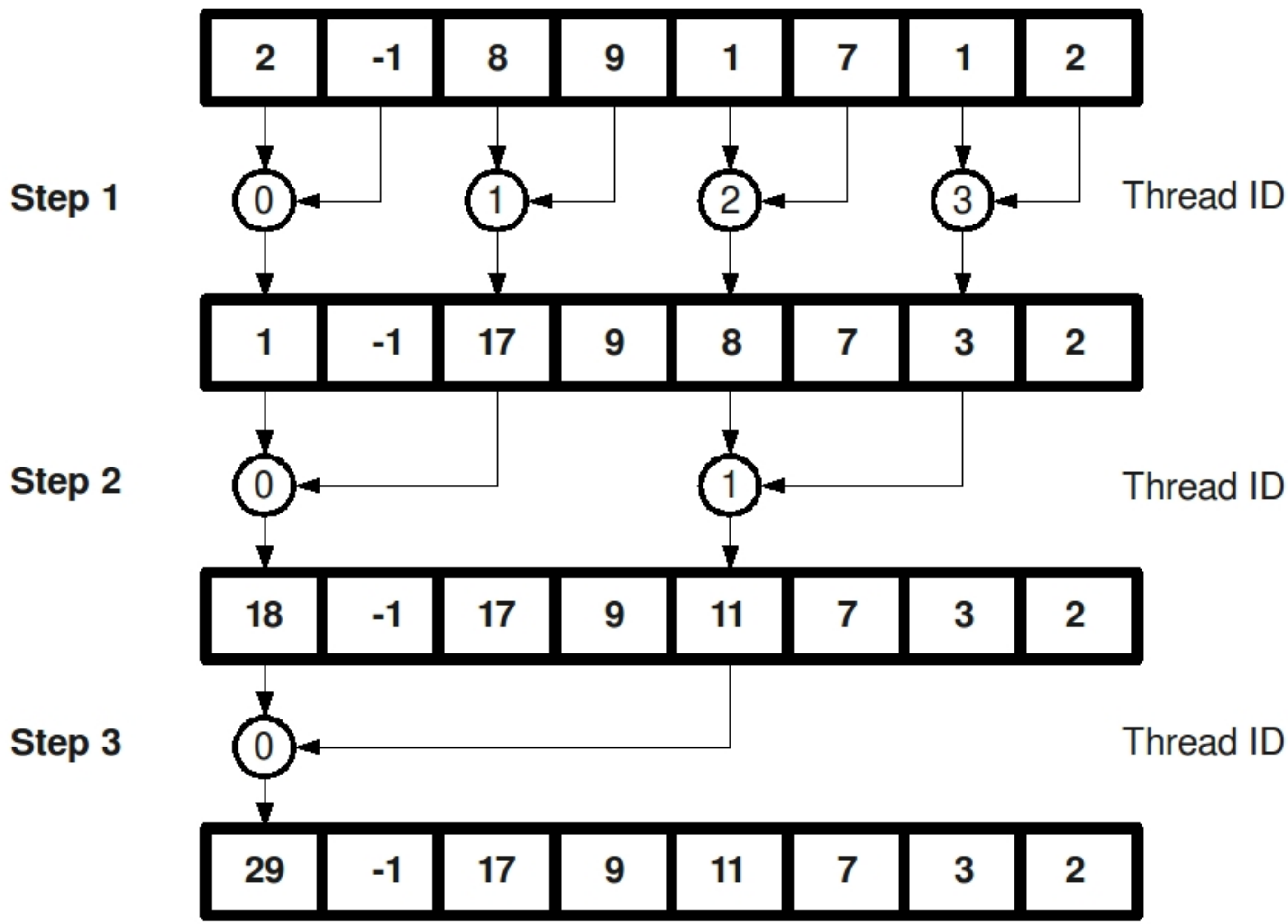
  - Atomic operation forces each atomicAdd operation to be sequential, defeating purpose of using GPU
    - Will likely be slower than on CPU

Program Parallelization: Data Races

- Still possible to use GPU
  - Better option: reduction with multiple steps
  - More difficult to parallelize than embarrassingly parallel loop
  - May be able to find reduction as part of a programming library
    - Implemented/optimized in thrust (CUDA library) and OpenACC compilers
    - Likely better performance using other people's optimized code than writing own implementation

Program Parallelization: Reduction

- Illustration of reduction on GPU on 8-element array



GPU Problems and Detection

- GPU Problem Categories:
    - Intra-group Data Races
    - Inter-group Data Races
    - Barrier Divergence

- **GPUVerify** is a program that can be used to help verify the integrity and check for the above conditions in GPU kernels

GPU Problems and Detection

- Intra-group Data Races
  - OpenCL – Data Race between work items within the same work group
  - CUDA – Data Race between threads in the same thread block

Example from GPUVerify documentation:

Suppose the following OpenCL kernel is executed by a single work group consisting of 1024 work items:

```
__kernel void foo(__global int *p) {
  p[get_local_id(0)] = get_local_id(0);
  p[get_local_id(0) + get_local_size(0) - 1] = get_local_id(0);
}
```

GPU Problems and Detection

Example from GPUVerify documentation - Explanation:

An *intra-group* data race can occur between work items 0 and 1023. If we run GPUVerify on the example:

```
gpuverify --local_size=1024 --num_groups=1 intra-group.cl
```

then this intra-group race is detected:

```
intra-group.cl: error: possible write-write race on ((char*)p)[4092]:

intra-group.cl:3:23: write by thread (0, 0, 0) group (0, 0, 0)
 p[get_local_id(0) + get_local_size(0) - 1] = get_local_id(0);

intra-group.cl:2:5:  write by thread (1023, 0, 0) group (0, 0, 0)
 p[get_local_id(0)] = get_local_id(0);
```

GPU Problems and Detection

- Inter-group Data Races
  - OpenCL – Data Race between work items in *different* work groups
  - CUDA – Data Race between threads in a *different* thread block

Example from GPUVerify documentation:

Suppose the following CUDA kernel is executed by 8 thread blocks each consisting of 64 work items:

```
#include <cuda.h>

__global__ void foo(int *p) {
  p[threadIdx.x] = threadIdx.x;
}
```

GPU Problems and Detection

Example from GPUVerify documentation - Explanation:

The kernel is free from intra-group data races, but *inter-group* data race can occur between threads in different blocks that have identical intra-block thread indices. If we run GPUVerify on the example:

```
gpuverify --blockDim=64 --gridDim=8 inter-group.cu
```

then an inter-group race is detected:

```
inter-group.cu: error: possible write-write race on ((char*)p)[0]:

inter-group.cu:4:3: write by thread (0, 0, 0) group (0, 0, 0)
 p[threadIdx.x] = threadIdx.x;

inter-group.cu:4:3: write by thread (0, 0, 0) group (1, 0, 0)
 p[threadIdx.x] = threadIdx.x;
```

GPU Problems and Detection

- Barrier Divergence
    - When a barrier occurs the threads within a thread block/work group should evaluate the barrier condition uniformly. If they do not, a single thread could diverge (or skip)

Example from GPUVerify documentation:

GPUVerify rejects the following OpenCL kernel, executed by a single work group of 1024 work items, because work items will execute different numbers of loop iterations, breaking the barrier synchronization rules:

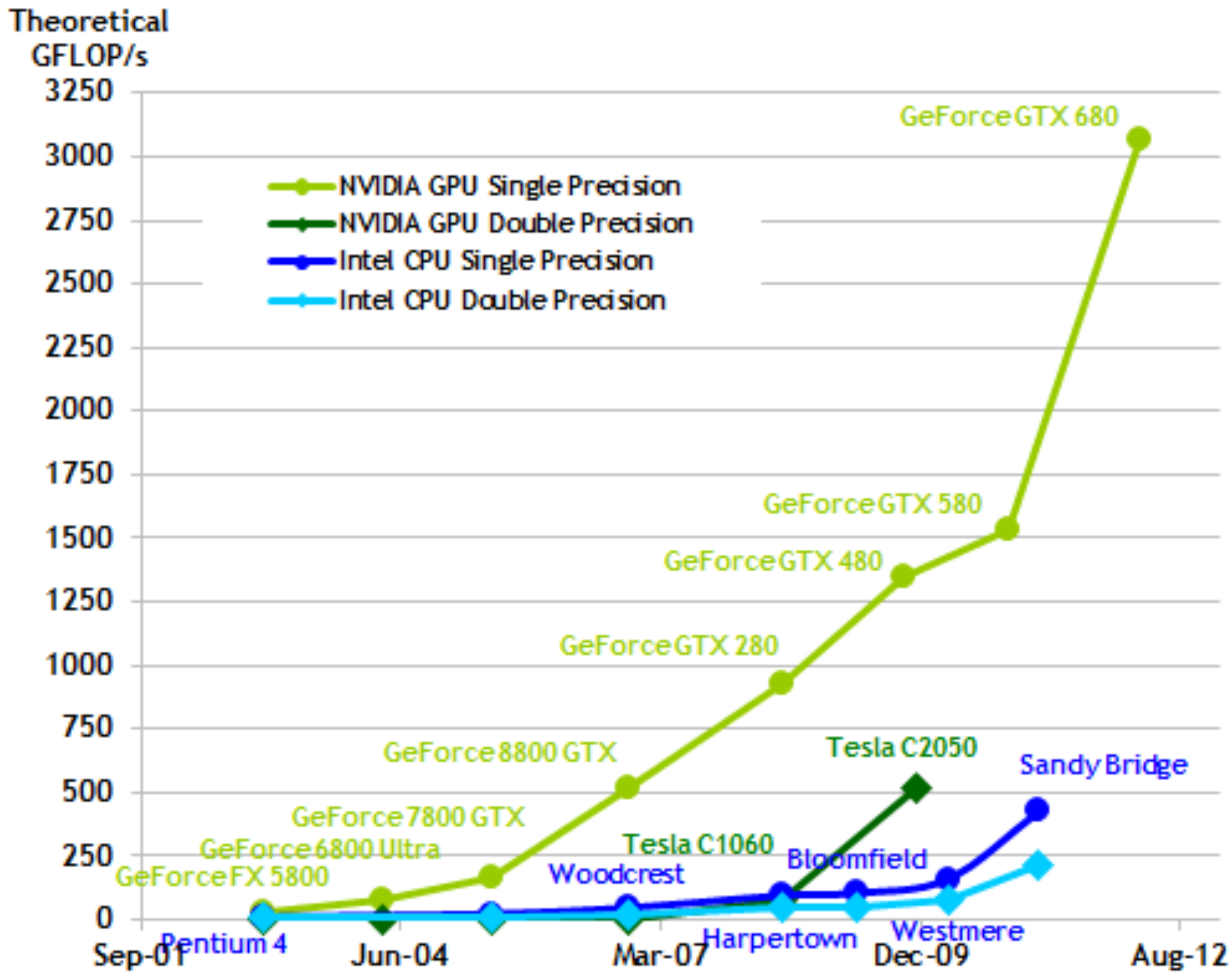
```
__kernel void foo(__global int *p) {
  for(int i = 0; i < get_global_id(0); i++) {
    p[i + get_global_id(0)] = get_global_id(0);
    barrier(CLK_GLOBAL_MEM_FENCE);
  }
}
```

```
gpuverify --local_size=1024 --num_groups=1 barrier-div-opencl.cl
```

```
barrier-div.cl:4:5: error: barrier may be reached by non-uniform
control flow
   barrier(CLK_GLOBAL_MEM_FENCE);
```

GPU Overhead

- Max GPU performance much better than CPU



GPU Overhead

- Theoretical performance doesn't account for overhead of GPU computing
  - Time to set up GPU environment
  - Time to compile GPU program at run-time (in OpenCL)
  - Time to set up and free memory on GPU
  - Time to transfer data from CPU to GPU and vice versa
  - Possible resource overhead of dedicating portion of workforce to GPU programming

GPU Overhead

- Overhead often not accounted for in work showing GPU speedup
  - GPU programmer wants to show as large a speedup as possible
  - Overhead time may differ across systems

GPU Overhead - Timing Considerations

- Overhead time may differ across systems
  - Transfer time may differ depending on file system and RAM configuration
- Programmer may intend to keep data on GPU for possible further processing
- Need full application/use cases to be able to measure influence of transfer time

GPU Overhead

- "Fixing" Overhead issue
  - Initial GPU implementation doesn't show speedup when using GPU due to overhead
  - Solution: make the problem space larger!

GPU Overhead - increased problem size

- Increase parallelism, makes overhead lower portion of overall computation
- Often valid solution in academic work
- Interesting to show how speedup vary across various problem sizes
- Feel free to adjust loop sizes in project 1 and show results for different configurations

Advertised vs. Actual Speedup

- Speedup shown in GPU papers often are compared to un-optimized CPU code
    - NVIDIA admits that most of the 100x+ speedups are from academia and compared to un-optimized CPU code

Advertised vs. Actual Speedup

- **Quote from GM of NVIDIA's Tesla business**: "Most people we find who have optimized CPU code, and really you'll only find optimized CPU code in the HPC world, get between 5x to 10x speed up, that's the average speed up that people get. In some cases it's even less, we've seen people getting speed ups of 2X but they are delighted with 2x because there is no way for them to get a sustainable 2X speed up from where they are today"

Advertised vs. Actual Speedup

- Paper from Intel: "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU"
    - **Authors of paper** "find that after applying optimizations appropriate for both CPUs and GPUs the performance gap between an Nvidia GTX280 processor and the Intel Core i7 960 processor narrows to only 2.5x on average" on a set of common benchmarks

Advertised vs. Actual Speedup

- Maximum GPU speedup after applying optimizations is 14.9x

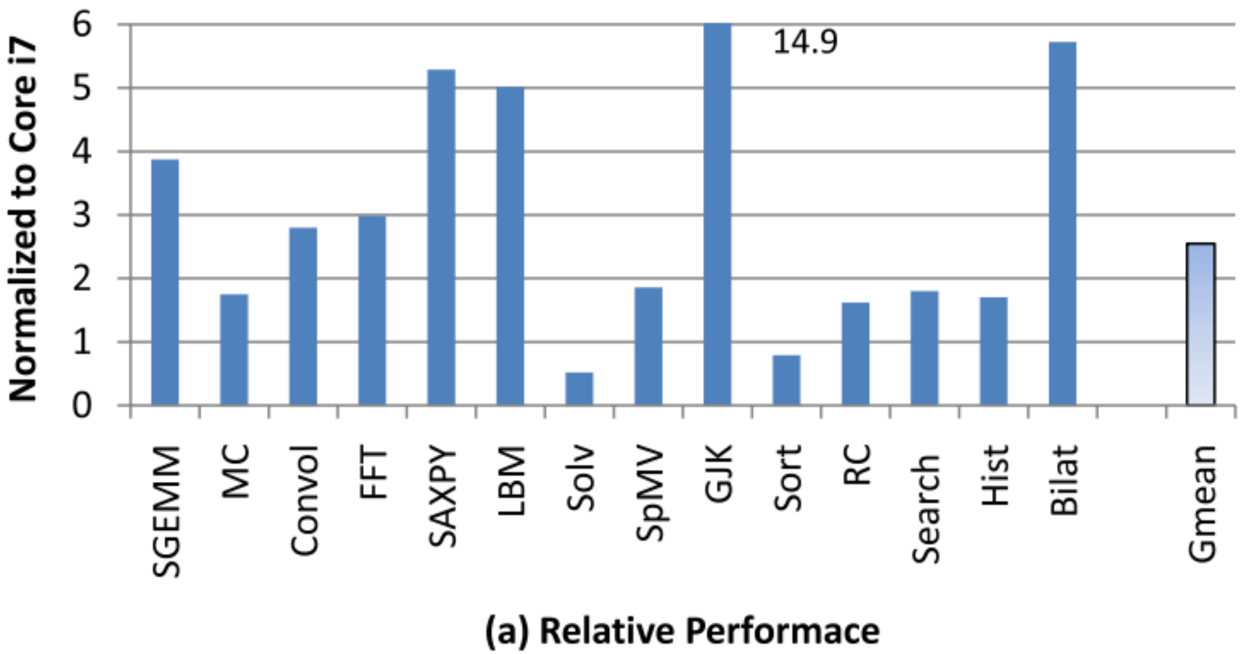


Figure 1: Comparison between Core i7 and GTX280 Performance.

Advertised vs. Actual Speedup

- **NVIDIA's response**: "It’s a rare day in the world of technology when a company you compete with stands up at an important conference and declares that your technology is *only* up to 14 times faster than theirs. In fact in all the 26 years I’ve been in this industry, I can’t recall another time I’ve seen a company promote competitive benchmarks that are an order of magnitude slower."
- Also claim easier to code/optimize on GPU using CUDA than on multi-core CPU