Prof. G. Zachmann
A. Srinivas

University of Bremen
School of Computer Science
CGVR Group
May 8, 2014

Summer Semester 2014

Assignment on Massively Parallel Algorithms - Sheet 4

Due Date 21. 05. 2014

Exercise 1 (Reduce operations, *8 Credits*)

The framework `reduce_max_sum` sets up a large array in global memory on the GPU.

The goal of this exercise is to write a program that computes the sum, the max, and the argmax for each block in *one* kernel call. Then these intermediate (per block) results are combined on the CPU. In this exercise you can assume that the input vector has a power-of-two length (and all elements are valid).

Example:
Input Array: 1, 3, -10, 0.2, 42.17, -0.1
Sum 36.27, Max: 42.17, Argmax 4:

Your task are the following:

a) Implement a version of the kernel that makes use of shared memory.

b) Implement another version of the kernel using global memory *only* for all intermediate results.

Note: CUDA does not support synchronization across different blocks of a kernel call.

c) Write are CPU reference implementation to compute the sum, max and argmax. Compare the running times of all three solutions.

c) Describe (in pseudo code) how you could change your kernel so that it can handle vectors of *arbitrary length*. (You don't have to implement this version.) What could be detrimental to the performance of this modified kernel?

Exercise 2 (Find the Synchronization Bug, *2 Credits*)

In the following kernel for the dot product, there is a bug that will cause erratic errors, which might look suspiciously like a race condition.

a) Find the bug. Hint: read the code carefully until the very last line.

b) Fix the bug by *only* adding one line of code. You don't have to implement anything.

```
__global__
void dotproduct( float *a, float *b, float *c, int N )
{
  __shared__ float cache[threadsPerBlock];
  int tid = threadIdx.x + blockIdx.x * blockDim.x;

  if ( tid < N)
      cache[threadIdx.x] = a[tid] * b[tid];

  // for reductions, threadsPerBlock must be a power of 2!
  int i = blockDim.x/2;
  while ( i != 0 )
  {
      __syncthreads();              // wait until all input data is ready
      if ( threadIdx.x < i )
          cache[threadIdx.x] += cache[threadIdx.x + i];
      i /= 2;
  }

  // last thread copies partial sum to global memory
  if ( threadIdx.x == (blockDim.x - 1) )
      c[blockIdx.x] = cache[0];
}
```