Department of Information and Computer Science
Utrecht University

INFOB3CC: Final Exam

Trevor L. McDonell

Thursday, 31 January 2019, 11:00–13:00

Preliminaries

- The exam consists of 4 pages (including this page).
- Fill out the answers in a **separate answer booklet**. Clearly label each question.
- Write your **name** and **student number** at the top of every page you hand in.
- The maximum score is stated at the top of each question. The total number of marks available is $10 + 2$.
- Give **clear** and **concise** answers. The questions require only a short explanation of around one or two sentences each. Please write legibly.
- You may use diagrams to help explain your answers.
- Use a blue or black pen.
- Answer questions in English.

Good luck! (:

Question 1 (0.5 points). What is the difference between parallelism and concurrency?

Question 2 (0.5 points). What is the difference between task parallelism and data parallelism? Which of these types of parallelism are GPUs most suitable for?

Question 3 (1 points). For efficient code execution using SIMD instructions, the layout of data in memory must be considered. What is the difference between AoS and SoA? Which structure is better suited for SIMD code and why?

Question 4 (1 points). In OpenCL, how should the threads read and write data to global memory in order to maximise effective memory bandwidth? What is the ideal way for the threads of a warp to read and write to local memory? For each, explain what happens when the memory transfer is not in the ideal case.

Question 5 (1 points). Consider the following OpenCL kernel:

```
__kernel void question5(__global float* arr, const int n)
{
  int tid = get_global_id(0);

  if (tid < n) {
    arr[tid] = (float) tid;
  }

  barrier(CLK_GLOBAL_MEM_FENCE);

  if (tid >= 1024 && tid < n) {
    arr[tid] = arr[tid - 1024];
  }
}
```

The kernel is executed with a global work size of $(12 * 1024 =)$ 12288, which is also the size of the array `n`. The local work size is 1024. After executing the kernel, what value is stored in the array `arr` at index 1092?

Question 6 (2 points). Write a parallel, warp-wide, right-to-left inclusive plus-scan function in OpenCL, targeting the GPU. For example, given the following input, the function should produce the following output values:

```
input  = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
output = [32,31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,
          16,15,14,13,12,11,10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Question 7 (2 points). In image processing, the Gaussian blur is the result of convolving an image with a Gaussian function. How can a convolution be implemented in terms of parallel patterns? What is a separable convolution, and why would you want to implement a convolution like this?

Question 8 (2 points). The *segmented scan* generalises the scan pattern to perform separate parallel scans on arbitrary contiguous partitions (segments) of an input vector. For example, given these input values and segment descriptor, a segmented prefix sum will produce the following result:

```
input  = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
segd   = [4,4,8,1,3,1]
output = [1,2,3,4,1,2,3,4,1,2,3,4,5,6,7,8,1,1,2,3,1]
```

One way to implement segmented scan is to use a regular (non-segmented) scan, but where the combining function between elements, $\oplus$, has been transformed into a segmented version, $\oplus^s$, that takes an additional parameter indicating whether this value is at the start of a new segment:

$$(f_x, x) \oplus^s (f_y, y) = (f_x | f_y, \text{ if } f_y \text{ then } y \text{ else } x \oplus y)$$

For our previous example, this requires the following array to store the values of the flags f, where a 1 indicates the start of a new segment, and a zero otherwise.

```
flags  = [1,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,1]
```

Write down the step(s), in terms of parallel patterns, which can be used to produce the array **flags** from the segment descriptor **segd**.

Question 9 (bonus, 2 points). Write work-group wide parallel reduction kernel in OpenCL. The kernel has the following signature:

```
__kernel void reduce(
    __global float* odata,      // output array
    __global float* idata,      // input data
    __local  float* smem,       // pointer to local memory
    const int n                 // total number of elements in idata
);
```

The kernel targets the GPU, where the logical warp size is 32 and the local group size is 1024. The input array contains 1024 elements. The parameter `smem` is a pointer to $(48 * 33 * 4 =)$ 6336 bytes of local memory.

You may make use of the following function, which computes a warp-wide reduction: `float warp_reduce(__local *smem, float x)`. The first parameter `smem` is a pointer to a block of 48 elements of shared memory, unique to this warp, which can be indexed in the range $[0, 48)$. The second parameter `x` is the value this thread includes in the reduction. All threads of the warp must call this function in order to participate in the reduction. On completion, lane zero returns the final reduction value of this warp, while all other lanes return an undefined value.

THERE ARE NO MORE QUESTIONS

\o/