Department of Information and Computer Science
Utrecht University

INFOB3CC: Accelerate (solutions)

Trevor L. McDonell

January 9, 2022

Introduction

Since it is the first week back after the break and we haven't covered any new theory yet, these are just some extra practical questions to get you familiar with the Accelerate, which you will use for the third practical.

Questions

1. Why does the Accelerate library make a distinction between `Acc` and `Exp`? How does this correspond to GPU programming in a language such as CUDA or OpenCL?

   > **Solution:** In Accelerate `Acc` is a parallel computation, and corresponds to a kernel function in CUDA or OpenCL which is executed in the GPU. The `Exp` sub-language corresponds to the code which each individual GPU thread executes in parallel.

2. The Mandelbrot set is generated by sampling complex numbers c in the complex plane and determining whether under iteration of the polynomial:

   $$z_{n+1} = c + z_n^2$$

   that the magnitude of z_n remains bounded however large n gets.

   (a) Follow this tutorial to implement a Mandelbrot set generator in Accelerate:
   `https://www.acceleratehs.org/examples/mandelbrot.html`

   Note: As explained in the Accelerate documentation as well, `lift` and `unlift` are very general functions that you need in specific situations, but require you to be very clear to the compiler about what types things has (and if you're not quite clear enough, you will get difficult-to-understand type errors). To combat this difficulty, pattern synonyms such as `T2`, `I1`, etc. were defined, which work more conveniently and give you better type errors when you make a mistake.

   The tutorial was written before the implementation of `T2`, `I1`, etc. in Accelerate. As such, it still uses `lift` and `unlift` to convert back and forth between e.g. `Exp (Z :. Int)` and `Exp Int`. In modern Accelerate code, as long as you're not defining your own data types (in this Mandelbrot tutorial, as well as in the Quickhull practical, you aren't), you should be able to program without using `lift` and `unlift` at all.

   In particular, instead of `(unlift -> x :+ y)`, use `(x ::+ y)` (search for "`pattern (::+)`" in the documentation for the `Data.Array.Accelerate.Data.Complex` module). And instead of `lift (x :+ y)`, use `x ::+ y`. Similarly, instead of `(unlift -> Z :. i :. j)`, use `(I2 i j)`, etc.

   (b) What are the problems with executing this kind of program on the GPU?

Solution: As with GPU programming in CUDA or OpenCL, each parallel operation (`Acc`) in Accelerate consists of many threads (`Exp`) executing in data-parallel. This means that the same considerations for parallel GPU code exist in Accelerate code. In this case, each thread might take a different number of iterations before deciding that point is in the set or not. On a device such as the GPU where the individual threads are mapped onto SIMD hardware where predicated execution is used to disable the SIMD lanes which do not all take the same branch, part of the hardware will thus not be used. This is known as *warp divergence*.