Module 1 – Introduction

These modules are targeting students with potentially very different backgrounds in OpenACC and parallel programming. As such, we do not assume that the students are coming into these modules with any parallel programming experience what-so-ever. Module 1 is meant to be a fast paced “catch-up,” whereas students are expected to gain a conceptual understanding of parallel programming, and learn how to implement these concepts using OpenACC.

Topics that will be covered are as follows:

- Introduction to parallelism
- The goals of OpenACC
- Basic parallelization of code using OpenACC

Our focus on OpenACC in this module is very narrow; we introduce parallelism in a very conceptual, graphical way, and show how we can relate these concepts to simple OpenACC directives. The goal with this is to ensure that students associate OpenACC with “general parallel programming” or “expressing parallelism” rather than thinking that OpenACC is a “multicore/GPU programming model.”

The module finishes with a graphical, step-by-step walkthrough of two basic codes; one is parallel, and the other is not. We show why we can parallelize one (safely) but not the other, and then show how we can express this parallelism with an OpenACC directive.

Each module will have a lab associated with it. Module 1’s lab cannot be code-focused, as the students simply do not have a great understanding of OpenACC at that point. This lab will not be a primary focus, and will be implemented at a later time.
Module 2 – Profiling

These modules are meant to be profiler-driven. Students will make small changes to their code, and re-profile it, comparing changes in the profiler vs. their expectations. Module 2 will have students profile the sequential Conjugate Gradient code to obtain baseline results. Afterwards, they will begin to implement a naïve parallelization of the code. The Conjugate Gradient code has 3 functions that are meant to be parallelized; students will add OpenACC directives to each of these functions, one-by-one, and view changes seen in the compiler. At this point, students are only expected to run their code for a multicore accelerator. The GPU implementation is more complex, and will be the focus of Module 3.

Topics that will be covered are as follows:

- Compiling sequential and OpenACC code
- The importance of code profiling
- Profiling sequential and OpenACC multicore code
- Technical introduction to the Laplace code

This module is meant to very customizable to the needs of the lecturer. The screenshots included use the PGI compiler, however, if the lecturer prefers to use a separate compiler, they have the option to switch out the screenshots with their own, or to profile the code live. Also, the lab for these modules can also be swapped out for any similarly complex lab code. Specific audiences may react better to specific codes, so provided that the code is sufficiently complex, any code can be substituted. However, this would require extra preparation by the lecturer, and would not realistically be able to be done using qwiklabs.

This module will also allow students to profile a sequential, and a multicore code to observe the differences. The profiling process is mostly a step-by-step tutorial, with regular pauses for explanation, or to allow students to explore the features of the profiler. The code that students are profiling is the Laplace code, which they will be working on for modules 2-6.

We choose to focus on multicore at first for two main reasons: multicore is a simpler platform to program for, and we want to emphasis that OpenACC is more than a “GPU programming model.” Students will begin working with GPUs in Module 4.
Module 3 – OpenACC Directives

Students will have already seen the parallel, kernels, and loop directive at this point. Now, this module will focus on teaching the specifics of each directive, the differences between them, and how to use them to parallelize our code.

Topics that will be covered are as follows:

- The Parallel directive
- The Kernels directive
- The Loop directive

Students will learn the key differences between the parallel and kernels directive, and can use either of them on the lab. It is recommended that they try both directives, however. This module includes a lot of code examples, and graphical representations of how the directives work with the hardware.

The lab section is designed for the students to achieve a working, near optimal version of a multicore Laplace program.
Module 4 – GPU Programming

This module is designed to teach students the key differences between GPUs and multicore CPUs. We also delve into GPU memory management, mostly from a conceptual level. We present CUDA Unified Memory as a reasonable solution to memory management, and then finish the module with a guide to GPU profiling using PGPROF. We also draw parallels between GPU architecture and our OpenACC general parallel model.

The lab section will allow students to play with basic data clauses, and managed memory. Then they will profile the code, and see how the changes they are making things affect how the GPU is running (by viewing things such as time spent on data transfers.)

Topics that will be covered are as follow:

- Definition of a GPU
- Basic OpenACC data management
- CUDA Unified Memory
- Profiling GPU applications
Module 5 – Data Management

In Module 4, we introduced students to a very basic solution to GPU data management. At the beginning of Module 5, we highlight the problems that this basic implementation has. The problem with our naïve solution is that there is far too many data transfers between the compute regions. Our program takes more time transferring data than it does computing our answer. We will have students remedy this by using explicit data management with the OpenACC data directive.

Topics that will be covered are as follows:

- OpenACC data directive/clauses
- OpenACC structured data region
- OpenACC unstructured data region
- OpenACC update directive
- Data management with C/C++ Structs/Classes

The bulk of this module will be code snippets and diagrams. We will use diagrams to represent CPU/GPU memory, and show the interaction between the two as we analyze the data directive/clauses. The lab section will allow students to experiment with both a structured and unstructured approach to data management in their Laplace code.
Module 6 – Loop Optimizations

Module 6 is the last “core” module. After Module 6, we expect students to be able to begin parallelizing their own personal code with OpenACC with a good amount of confidence. The remaining modules after this point are considered to be “advanced” modules, and are optional, and some may only be applicable to specific audiences. Module 6 is all about loop clauses. This module is meant be very visual, so that students can get a good sense of exactly how each clause is affecting the execution of their loop.

Topics that will be covered are as follows:

- Seq/Auto clause
- Independent clause
- Reduction clause
- Collapse clause
- Tile clause
- Gang Worker Vector

This module is “under construction,” meaning that it is the least polished. Explaining loop execution with words can be very confusing, and isn’t engaging for the students. So, the hope is to teach the loop clauses with visual diagrams, and step-by-step examples. Outside of the gang/worker/vector we take a step away from GPUs, and again focus on parallel programming conceptually, meaning that we are no longer worried about which hardware we are running on, but rather we are concerned with how we can optimize our loops for general parallelism.

We touch on each of the loop clauses, show how they look within code, and give a visual representation of it. The gang/worker/vector will most likely be the lengthiest section in this module, just because it is the most complex. Also, in the lab section of Module 6, we will make our final optimization to our Laplace code by utilizing loop optimizations and gang/worker/vector.

Module 7 – Asynchronous OpenACC

This module will primarily introduce the async clause and wait directive. The module will discuss the idea of making the host and device operate asynchronously. The primary example that will be used in this module is a pipelined image transformation filter.

Module 8 – OpenACC Interoperability

This module demonstrates how OpenACC is able to interoperate with other programming modules. The module will have examples for calling into accelerated libraries, interoperating with memory allocated outside of OpenACC, and how to mix OpenACC with MPI.
Module 9 – Multi-Device OpenACC
This module will build upon modules 7 and 8 to demonstrate how to divide work among multiple devices. The example will build upon the image filter used in module 7 and will reference MPI interoperability from module 8. The module will demonstrate managing multiple devices using a single process and the OpenACC runtime API and also using MPI to manage multiple devices via multiple processes.

Module 10 – Case Study?
At this point all of the basics needed to program in OpenACC have been covered. A real app case study would be useful here to demonstrate the big picture of taking a real, unaccelerated app and accelerating it using OpenACC.