Title: Accelerated protein matching using modern graphics processing units
Author: Trevor L. McDonell, PhD student, UNSW
Email: trevor.mcdonell@gmail.com

Keywords: Functional programming; parallel computing; data parallelism; proteomics

Programming parallel computers is an extremely challenging task even for expert computer programmers, let alone for scientists in other disciplines whose computations often require the use of such high performance machines. Moreover, even the best parallel programmers can not do so without significant effort and a small amount of magic. While computers are certainly indispensable tools when it comes to solving complex problems, it undoubtedly complicates the life of a scientist to have to think about programming with concurrency as well as correctness, in order to investigate their true problem of interest.

Gaining increasing attention over the last few years has been the use of modern graphics cards for general purpose computation. These specialised coprocessors consist of a large number of simplified processor cores, and have the ability to manipulate large amounts of data at a rate which can be orders of magnitude faster than what is possible using traditional processors. Despite this overwhelming potential, however, we are left with the same basic problem; how can we tap into these latent powerhouses to help solve interesting problems?

One such application is the identification of proteins from the results of a mass spectrometry experiment. Due to the large number of proteins and their post-translational modified variants, the amount of data generated by a single experiment can easily exceed several gigabytes, and the time necessary to analyse and interpret the measurements often significantly exceeds the time spent on the experiment itself. In this work, we use the functional programming language Haskell and reimplement several commonly used functions (such as 'map', 'fold' and 'zipWith') so that internally the computation is executed using the graphics processor. Using these tools, we implement the SEQUEST algorithm, which is commonly used to analyse the results of a MS experiment. We investigate whether it is possible to implement an efficient, parallel version of a real-world algorithm, while still retaining the style of the Haskell language – that is, without becoming burdened with too many details of the parallel implementation or speciality nature of the graphics processor. We provide some preliminary results and comparison to the canonical (sequential) implementation.