Steven Diviney
08462267
CS3015

1 Why parallel programming is hard

There are a few key issues that make parallel programming difficult. The first is the simple fact that programmers have been trained to think sequentially. Even today when parallel programming has been identified as the next big thing, introductory and even intermediate courses still teach programming in a strictly sequential way. This permeates every aspect of development, from analysis, development, debugging and testing. This methodology is not confined to programming however. People naturally break up problems into a series of linear tasks to be completed. The human brain is an inherently parallel machine. Perhaps if are thought to think in parallel from the outset the problem would be significantly reduced.

Interestingly we approach parallel programming with an asynchronous model. This is to account for differences in timing with different threads completing and so on. Our most familiar parallel machine, our brain, however is synchronous. The synapses fire at a set rate. Even in hardware parallel tasks are done in time with a common clock. Most implementations of parallel programming languages do not easily allow a synchronous model.
The common logic today seems to be "take a sequential program and break it up into multiple parts, then assign them to separate cores". The problem with this "decomposition" approach is we are using sequential elements to construct parallel programs. If we use sequential elements to build sequential programs, why shouldn't we use parallel elements to build parallel programs?
Another problem inherent with parallel programming is the sheer volume of additional code required to parallelise code. The programmer must dedicate a substantial amount of time to housekeeping. More often than not, the parallel code and the code to manage the parallel code is ambiguous at best. This leaves the programmer with the impression that parallel programming is a hugely complicated task. If we are to simplify parallel programming, better tools and models need to be implemented
The interleaving of instructions in parallel streams introduces an element of non-determinism into any program. Successive execution might interleave the instructions in different ways, depending on factors such as how many processors are be used, the initial state of the machine, where the data is stored physically in memory and so forth. All of these extra, seemingly ran-

dom factors, makes testing of code questionable. Programming is a high level task, built upon several layers of abstraction. It is taken for granted that the various "layers" will work as expected. When an element of random chance, the interleaving of instructions, is introduced it invalidates such abstractions. The solution thus far has been to enforce determinacy. However, this requires significant overhead and additional work by the programmer.