Colby Morrison

Article Summary

In their article, Sutter and Laurs describe the incoming concurrency revolution. Moore’s law is dead. As a result, programs no longer will run faster simply because processors are getting faster. The authors describe a shift to multicore machines as the primary method of increasing hardware power. To take advantage of these multicore computers, programs will have to be written concurrently, allowing multiple tasks to execute in parallel on the separate cores. However, programmers are used to writing sequential programs and algorithms and many modern languages are suited better for sequential programming than parallel programming. The authors argue that a new outlook is needed to allow for the easier development of concurrent applications.

Concurrent programming as it currently stands is much more difficult than sequential programming. Though concurrency is already working well in some places, for example in web servers that utilize concurrent databases, in many other areas concurrency is difficult and convoluted. The authors describe the degree of coupling and granularity as important factors in a concurrent program. Coupling refers to the communication between tasks and their shared data while granularity refers to the scope of the task at hand and the difficulties in simply creating and executing parallel tasks that may result. Furthermore, hardware support is almost always needed, such as the xchg instruction in x86\_64. The authors describe 3 types of parallelism with respect to the data and computation. The first is independent parallelism where members of a collection are manipulated in parallel by the same routine. Regular parallelism is similar, but it implies dependencies between the members, so more care is needed. The final type is unstructured parallelism, where the parallel computations themselves differ. This is the most frequent and most difficult flavor. Furthermore, managing shared state is a critical issue. Locks are difficult to keep track and rely on the programmer to implement correctly. Even the simplest programs are vulnerable to the deadlock’s deadly embrace. Though attempts have been made to recover from deadlock automatically, none are foolproof.

To overcome these issues, the authors argue that a higher level of abstraction than locks and threads are needed to allow for more fluid concurrent programming. This is analogous to how object-oriented programming provided a higher level of abstraction over strictly imperative code or assembly. They note functional languages’ higher-level features such as map and map-reduce are a fantastic example of this type of abstraction. Asynchronous code, with callback functions and the like, is another promising option. This can be seen today in platforms such as Node.js. Also discussed are various tools that can make parallel code seem sequential to ease programming.

Finally, the authors discuss the need for better debugging and testing tools and practices for concurrency. They note better logging and reverse execution as good examples. Overall, the concurrency revolution is here, so we must be ready to adapt.