

## Scientific Software for the 21<sup>st</sup> Century

Anybody working in the software industry will be aware of the changes in hardware that are causing us to re-factor and, in many cases, rewrite our software. As the number of cores increases and their clock speeds decrease a great deal of attention has focussed on the technologies that we can use to address these developments. Should we stick with tried and well-understood technologies such as OpenMP or MPI, or switch to one of the many new languages that are springing up? Perhaps we should drop our procedural and object-oriented programmes and write everything in a functional language such as Haskell or F#?

Maybe we should think less about the technologies that we use and focus more on the algorithms that we employ. Many of our existing algorithms are inherently sequential or, even if they do parallelise, they don't scale up to exploit the tens and hundreds of cores which we will have at our disposal in a few years time. For example, a few years ago NAG implemented a suite of software for solving global optimisation problems, based on an algorithm called *multi-level coordinate search*. This algorithm splits the region under consideration into boxes which are either discarded because they can't hold the optimal value, or are further sub-divided because they might do. Attempts to parallelise this algorithm failed because all threads have to continually update each other on their progress, or lack of it, and the communication overhead swamps the computation time. Instead, we have been forced to investigate a different algorithm known as *particle swarm*. This is a stochastic algorithm where "particles" search part of the region under consideration and occasionally exchange information about their findings with one of their "neighbours". Information propagates through the "swarm" of particles, but the overall communication overhead is very low and the algorithm appears to scale extremely well to larger numbers of cores.

A few years ago we wouldn't have considered using an algorithm like this because it performs quite poorly on a single processor. Moreover it is fundamentally non-deterministic, making it hard to test and to benchmark. However it's performance on modern multi-core processors is streets ahead of our traditional approach and so we're going to have to adapt our software development processes to incorporate it.

In a similar vein, we're involved in a project developing software for performing Monte-Carlo simulations in CUDA. One of our early adopters is using this code as part of an application studying the volatility of financial instruments by solving a variant of the Black-Scholes equation. They report that this code runs over 200 times faster on the GPU (a Tesla card with 240 cores) than a sequential version running on a conventional CPU. Once again a stochastic approach has been taken over a more conventional method (such as finite differences or finite elements) because it is easy to program and scales extremely well.

This illustrates another challenge that will face many of us soon – that of hybrid architectures where different kinds of processors co-exist in the same machine. Software that makes the most of such hardware must itself be a hybrid, and ideally should be able to adapt to the particular platform that it is running on and choose the best algorithm to use given the resources available at any given time. An algorithm that works well in a cache-rich environment such as a cluster of multi-core x86

processors may run very slowly on an accelerator, so it is not sufficient simply to have multiple implementations of the same basic approach tuned for different hardware. Such software is probably going to be quite complex and will certainly require a lot of specialised expertise to design and implement.

The good news is that there are a number of research and community activities which are seeking to develop high-performance components which can be incorporated into applications, improving software performance and programmer productivity. Examples include the PLASMA project based at the University of Tennessee which is developing a library of high-performance linear algebra components for multi-core platforms and the related MAGMA project which is doing a similar job for multi-core platforms where a GPU is also present. Such activities deserve the community's active support.

Of course, even with suites of tuned, adaptable components at our disposal, rewriting so much of the scientific community's software to exploit modern hardware is going to be a long, expensive effort. It's going to take time, and will require developers to learn new skills and adopt new tools and programming environments. On the other hand computer hardware has never been cheaper, so maybe now is the time to switch priorities and to invest in the future by investing in software.

Mike Dewar

Chief Technical Officer

Numerical Algorithms Group (NAG)

Article written for publication in Scientific Computing World Dec 09 / Jan 10 issue