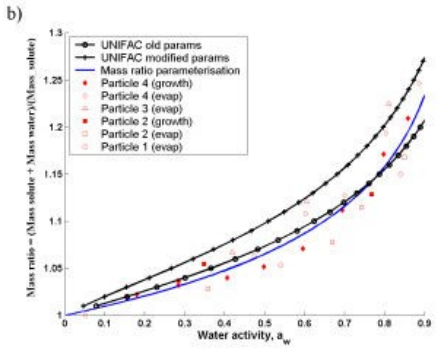
Porting Manchester's UNIFAC to OpenCL@XeonPhi: 160x speedup



Example of modelled versus measured water activity ('effective' concentration) for highly detailed organic chemical representation based on continental studies using UNIFAC

As we cannot use the performance results for most of our commercial projects because they contain sensitive data, we were happy that Dr. David Topping from the University of Manchester was so kind to allow us to share the data for the UNIFAC project. The goal for this project was simple: port the UNIFAC algorithm to the Intel XeonPhi using OpenCL. We got a total of 485x speedup: 3.0x for going from single-core to multi-core CPU, 53.9x for implementing algorithmic, low-level improvements and a new memory layout design, and 3.0x for using the XeonPhi via OpenCL. To remain fair, we used the 160x speedup from multi-core CPU in the title, not from serial code.

UNIFAC essentially calculates interactions between molecules in solution using what is known as the group contribution approach. Normally, it isn't used for say more than 10 compounds but in atmospheric science we can be interested in mixtures with up to 106 molecules. Problem is that one iteration with 800.000 molecules takes 33 seconds on a fast Intel CPU - as a simulation can take 10.000 iterations, the total time is put to 91.7 hours or 3.8 days.

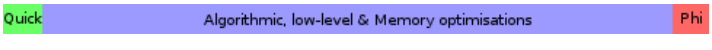
The speedups

As the code was single threaded, we had an easy start. A quick 3x speedup on a 4-core CPU was possible, using only OpenMP. A 4x speedup was not possible as the code in its current form did not scale well and needed work. So we started without OpenMP and focused on a single core first, having XeonPhi in mind.

Around 70% of our techniques were successful and we pushed the time from 33 seconds to 830 milliseconds on one CPU-core - a 39.8x speedup purely on algorithmic and low-level improvements! So the same simulation of 800.000 molecules went from 91.7 to 22.9 to 3.6 hours on a single CPU core. Using OpenMP, this got to about 207ms - indeed, the new code was capable of the full 4x speedup on the CPU - this explains the 53.9x in the first paragraph.

Here we got to the XeonPhi, the 31S1P to be exactly. We already focused on device specific optimisations, and rewrote the memory access. This brought the time on the XeonPhi to 68ms, putting the time for the whole experiment under 12 minutes.
What took 91.7 hours, now takes less than 12 minutes.
This was the first project we put XeonPhi first and you see that in the results, making the XeonPhi faster than a more powerful GPU (which needed 330 ms per iteration). The optimisations were also very effective on the CPU: using OpenMP on the CPU with XeonPhi specific optimisations took only 200ms. Using OpenCL, we got to even 130ms on the CPU.

The above shows that the magic doesn't happen by simply turning on OpenMP and making it run on a accelerator (XeonPhi, GPU, DSP or FPGA). Most performance increase was due to carefully crafting the code.



By far the largest part of the optimisations were due to

algorithmic, low-level and memory optimsations.

What can we do for you?

Below are the two main reasons for our customers to have hired us - they're both linked to the urge to out-compete and increase revenue.

Increase throughput

Often there are requirements to handle a certain amount of data within a given time. Think of a camera that needs to do some filters at 30 frames per second, or calculating the financial market exposure of a customer within a minute. What if you need to have a speedup of 800x to keep the computation within 30 minutes, as one customer needed? One option is to buy bigger hardware, but this has its limits. The other option is to make the code faster, as they did - our solution took less than a minute.

Most of our customers are first in market to have their software run on a GPU, making their software or service very competitive.

Creating new opportunities to out-compete

Say Dr. David Topping would share the optimised UNIFAC to one researcher and the other UNIFAC to another one. They would both get one week to get familiar with the software and one month to come up with some results. After that month the definition of a small experiment for one would be a huge project for the other. **What would take a full month (30 compute days) to compute for one of the researcher, would take the other one 14,550 days (39.9 years) and thus think of as impossible.** A simple test-setup of 10 minutes would take almost a full month for the other.

Because more things are getting possible, we often get a shift during projects. Like in two projects, where we had huge speedups (far over 100,000x), the goals changed overnight to include new requirements that were initially written off as "impossible". What would you do when you can process 100 elements per second initially and then are able to process over one million?

What software can be sped up?

We started with the algorithms that typical work fast on GPUs. Now we are confident that any type of software can be made faster - like in the above project, 90% of the speedup was without parallelisation. Just give us the needed speed-up and after a code-review we can tell you if that's possible.