Accelerating an Excel Sheet with OpenCL

One of the world's most used software is far from performance optimised and there is hardly anything we can do about it. I'm talking about Excel.

There are various engine replacements which promise higher speeds, but those have the disadvantage that they're still not fast enough with really heavy calculations. Another option is to use much faster LibreOffice, but companies prefer ribbons over new software. The last option is to offer performance-optimised modules for the problematic parts. We created a demo a few years ago and revived it recently.

Excel's bits and bytes

External DLLs

Excel can be combined with an external DLL, which can be an OpenCL program. Data from the cells can be sent and received, such that interaction is seamless.

Precision

There are problems with precision. Excel works with 8-byte numbers (64 bit floats), VBA has a variety of data types. The Double data type is 8 bytes, the Integer data type is 2 bytes. This means that Doubles in VBA are equal to Floats in C++ and programming needs to be done carefully.

The precision-problems are with all spreadsheet softwares, as Excel alternative Gnumeric discusses it nicely.

The demo

We wanted to make clear that OpenCL is much faster and integrated OpenCL in Excel - this can be any acceleration language from CUDA to HCC. The latest version of our Excel-demo did a financial calculation, namely Monte Carlo. It was inspired by the various Monte Carlo computations in the OpenCL SDKs. The speedup was quite big - partly because Excel is not fully using the hardware, but also because many financial algorithms do well on GPUs. In the demo it's quite obvious that OpenCL is used for the computations, but the Excel sheets could be designed such nobody even notices GPUs were used, with a fall-back to native code.

The speedup was at least 7x, depending on the exact hardware being used.

https://streamhpc.com/wp-content/uploads/2016/09/OpenCLExcelMonteCarlo.mp4

CPU versus GPU for this Monte Carlo computation was 2 minutes versus a few seconds. The video was 2m10 seconds and cut back

to 15 seconds. The mouse move to the center is due to clicking away a message box, which did not get recorded - the moment the mouse moves, the computation was finished.

Advancing the demo

The DLL can also send data to a central server (private compute cloud) to fully offload the computations. This is possible because interfacing and presentation is kept fully separated from the compute. Several other algorithms can be implemented and optimised for the server hardware, while the user works on a thin client. The server could have numerous dedicated GPUs, but also FPGAs.

I hope you learnt a bit on how to circumvent performance problems like those in Excel. Many other closed softwares have the possibility to be extended, and thus be made faster. It might look impossible to work around preferred supplier software at first, but we did find ways like here in Excel.

Make it work for you

First read the follow-up blog post, which describes the steps of speeding up Excel sheets.

We can speed up your Excel-sheets by building a backend that does the heavy lifting while leaving the interface (the sheets) the same. Get in touch to discuss your goals.