HPC03
Programming Perspectives for Pre-exascale Systems
F. Courteille* (NVIDIA) & J. Eaton (NVIDIA)

SUMMARY
Reservoir simulation of large scale projects is becoming increasingly complex, requiring more than simple black oil models and vertical well models to capture the behaviour of unconventional, fractured and highly heterogeneous production zones. Nvidia provides an array of accelerated linear algebra libraries to deal with the equations that must be solved in these situations. Accelerating sparse linear algebra on the latest GPU architectures has real potential for performance gains of hundreds of percent over carefully tuned multi-core CPU-only implementations, but at what cost in complexity? This talk will address the programming approaches needed to utilize GPUs at scale for today’s most challenging problems, and give a glimpse of the path forward to pre-exascale applications.
**Introduction**

Recent advances in hardware technology have made the GPGPU a strong addition to CPUs for HPC workloads, especially when considering price/performance or FLOPS/Watt metrics. One aspect that has held back deployment of this hardware for HPC has been the difficulty of programming these devices for the average engineer. To get the most out of the hardware has in the past required users to learn a GPU-specific language called CUDA (Compute Unified Device Architecture), in addition to having good parallel programming knowledge as well.

In this talk, we will emphasize the many successful ways to tap into the power of GPUs without learning or writing a single line of CUDA.

First, with zero changes to an existing code using BLAS routines, it is possible to use nvBLAS to intercept BLAS calls and re-route them to the GPU for ‘free’ acceleration of those kernels.

Second, using C libraries from the CUDA toolkit enables many programmers to avoid using CUDA directly, but make calls to CUDA libraries as a regular C library.

Third, the emerging OpenACC standard for directive based programming allows a user to add a few lines of directives to existing codes to inject GPU acceleration of critical loops.

This talk will show examples of the benefits of each approach, and demonstrate that CUDA programming knowledge is not strictly required to derive benefit from GPGPU hardware.
Conclusions

Learning CUDA is perceived as a necessary step to deriving value from GPGPU hardware, but in fact there are several existing methods to accelerate an existing code without using a single line of CUDA. These methods allow keeping a single code base for ease of maintenance, and deliver strong performance increases for less effort than taking a full CUDA approach.