Eighty-five percent of processors shipped today include an integrated GPU. While this GPU can be used to significantly improve application performance and energy use, existing GPU programming frameworks such as OpenCL, CUDA, and C++ AMP require too much knowledge about the specific GPU’s architecture to produce good application performance, and that performance isn’t portable to other GPUs. We advocate the use of familiar multi-core programming models for GPU programming.

This tutorial describes Concord, a heterogeneous C++ programming framework for processors with integrated GPUs that allows general-purpose data-parallel programs to take advantage of GPU execution. It can be used to accelerate both regular programs such as matrix-multiply and irregular programs such as those doing graph-processing. Concord supports most C++ features including virtual functions and reductions on the GPU. It supports seamless sharing of data between the CPU and GPU with an efficient software implementation of shared virtual memory. Concord also includes a number of compiler optimizations to improve the effectiveness of GPU execution.

We present results for nine realistic irregular C++ applications running on both a high-end desktop and an ultrabook. The results show that Concord’s GPU acceleration improves energy efficiency by up to 6.04× on the ultrabook and 3.52× on the desktop over a multi-core CPU.

The outline of our tutorial is as follows:

- Background on GPUs and GPGPU programming
  - Integrated and discrete GPUs
  - GPGPU frameworks: OpenCL 2.0, CUDA, C++ AMP, HSA, & Renderscript
- Concord C++ heterogeneous programming framework
  - Adapts familiar data-parallel C++ constructs from TBB
  - Efficient software implementation of shared virtual memory (SVM)
  - Compiler optimizations for SVM and improved GPU performance
  - Support virtual functions and reductions on GPU
  - Demonstrate performance and energy benefits of SVM