CUDA for ARM Platforms is Now Available

by Mark Harris June 17, 2013

In 2012 alone, over 8.7 billion ARM-based chips were shipped worldwide. Many developers of GPU-accelerated applications are planning to port their applications to ARM platforms, and some have already started. I recently chatted about this with John Stone, the lead developer of VMD, a high performance (and CUDA-accelerated) molecular visualization tool used by researchers all over the world. But first … some exciting news.

To help developers working with ARM-based computing platforms, we are excited to announce the public availability of the CUDA Toolkit version 5.5 Release Candidate (RC) with support for the ARM CPU architecture. This latest release of the CUDA Toolkit includes support for the following features and functionality on ARM-based platforms.

- The CUDA C/C++ compiler (nvcc), debugging tools (cuda-gdb and cuda-memcheck), and the command-line profiler (nvprof). (Support for the NVIDIA Visual Profiler and NSight Eclipse Edition to come; for now, I recommend capturing profiling data with nvprof and viewing it in the Visual Profiler.)
- Native compilation on ARM CPUs, for fast and easy application porting.
- Fast cross-compilation on x86 CPUs, which reduces development time for large applications by enabling developers to compile ARM code on faster x86 processors, and then deploy the compiled application on the target computer.
- GPU-accelerated libraries including CUFFT (FFT), CUBLAS (linear algebra), CURAND (random number generation), CUSPARSE (sparse linear algebra), and NPP (NVIDIA Performance primitives for signal and image processing).
- Complete documentation, code samples, and more to help developers quickly learn how to take advantage of GPU-accelerated parallel computing on ARM-based systems.

At GTC 2013 we announced the Kayla development platform, a mini-ITX system with an NVIDIA Tegra 3 quad-core ARM processor and an NVIDIA GeForce GT640 GPU. This GPU has Compute Capability 3.5, meaning that it supports most of the same CUDA features of a high-end Tesla K20 GPU. Kayla is where mobile computing meets supercomputing, and is the ideal development platform for both next-generation low-power HPC and for mobile computing software targeting NVIDIA's future "Logan" SOC.

To understand how easy it is to port existing CUDA applications to ARM, I spoke to John Stone, the lead developer of VMD. John is also an NVIDIA CUDA Fellow and Associate Director of the CUDA Center of Excellence at University of Illinois at Urbana-Champaign.

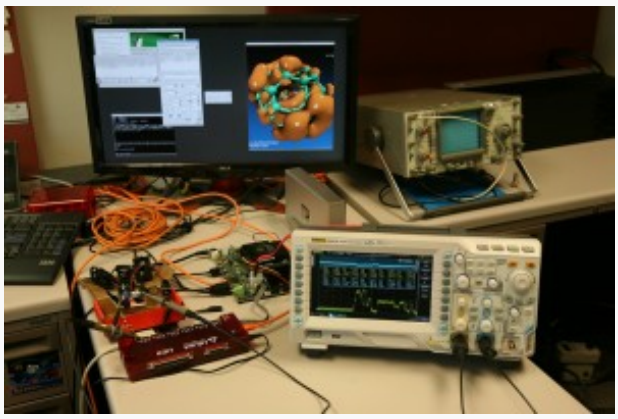
MH: John, why is CUDA on ARM interesting for your work?

JS: Having CUDA available on ARM is interesting to us for a number of reasons.

The ARM architecture is found in a very large fraction of mobile and embedded computing devices today, and we are currently developing mobile versions of our visualization and analysis tool VMD. One of the challenges in this work is to make the mobile version of VMD as feature-rich and interactive as possible, and some of that will hinge upon

achieving high performance in key VMD algorithms. Just as users of the desktop versions of VMD benefit from GPU-accelerated computing now, we hope that we will be able to do the same for mobile devices eventually. Going further, I expect that we could potentially see further proliferation of ARM into laptops or low-cost PC-like computing platforms in the future if it continues to gain popularity. All of this is a straightforward extension of what we've been doing with porting VMD to Android already.

Beyond our interest in mobile platforms, we have been very interested in using CUDA on ARM as a test bed for future GPU-accelerated energy-efficient supercomputing platforms. As you know, power consumption is a critical factor in the feasability of building the next generation of supercomputers, with so-called "exascale" computers being the next major long-term goal. In 2010 we evaluated the performance and energy efficiency benefits of GPU clusters for several scientific applications and found that beyond their well-known performance benefits, GPUs significantly improve energy efficiency.



A CUDA on ARM development board running VMD, with power usage displayed on an oscilloscope (courtesy John Stone).

Recently constructed petascale computers, such as Blue Waters at NCSA, have already begun to spur scientific advances that would not have been possible previously. Our laboratory recently solved the all-atom structure of the HIV-1 capsid on Blue Waters, where we benefited tremendously from the performance of the GPU-accelerated XK7 nodes. It would be wonderful if an exascale supercomputer (roughly 100 times the performance of Blue Waters) were available today; unfortunately, if one built such a machine with today's technology it would require at least one or two orders of magnitude more power. A large fraction of a nuclear power plant would have to be devoted to powering and cooling such a large system. There are many challenges to overcome before exascale computing becomes a reality, but the power consumption issue is fundamental, and the processors found in future supercomputers may have a great deal in common with the energy-efficient mobile platforms of today. Having CUDA available on ARM allows researchers like us to begin preparing our codes for future energy-effient HPC platforms, working past stumbling blocks, and developing new algorithms.

As GPUs continue to get faster and faster from one generation to the next, we have a need to stay ahead of the curve in terms of adapting our software so that we don't end up having CPU or PCI-e bottlenecks in our CUDA algorithms. The CARMA and Kayla platforms have taught us a lot about parts of our code where we'd made assumptions about the ratio of GPU performance to CPU performance, and/or the bandwidth of the PCI-e interconnect. In my view, algorithms that can achieve excellent performance on Kayla could be viewed as being well-prepared for future x86 CPUs paired with next-gen top-end CUDA GPUs such as Maxwell and Volta, and for future GPU hardware that closely couples ARM and GPU cores.

MH: What has been your experience with porting CUDA applications to ARM using the CUDA toolkit?

JS: We found it very easy to port our CUDA applications to the ARM platform.

We ported NAMD, our molecular dynamics simulation code, in two days, and some of that time also involved porting the Charm++ parallel runtime system that NAMD is based on. Similarly, it only took us a day or two to port our Lattice Microbes cell simulation software. Porting VMD was most involved due to the large number of libraries it depends on, but I ported VMD and all of its library dependencies in about four days.

The only CUDA code that we had to change when building for ARM was to disable the use of host-mapped memory (zero-copy) in some of our CUDA kernels, since it wasn't supported in the drivers yet. I would say that anyone that has ever worked with multi-platform software should have absolutely no difficulty getting their software running on ARM.

MH: Have you used the ARM versions of NVIDIA's GPU-accelerated libraries (cuFFT, cuRAND, cuBLAS, Thrust, etc.)?

JS: VMD uses the Thrust library for parallel prefix sum and sorting operations, and it has worked beautifully on ARM so far.

MH: Do you use native compilation on ARM, cross compilation for ARM on x86, or both? (and Why?)

JS: We chose to use the ARM-native CUDA compiler tool chain for our work, because this greatly simplified compilation of third party libraries that had build systems that were not well-prepared for cross compilation. We have found that the compile time for our larger applications (a half-million lines of code) using the native tool chain is perfectly reasonable. If we attach an SSD to the Kayla board, that goes a long way to eliminating compilation time concerns.

MH: What suggestions or lessons learned would you like to share with others getting started with CUDA on ARM processors?

JS: Our observation has been that the Tegra3 chip performs quite well on code that is cache-friendly. One area where traditional desktop CPUs will have a big advantage is in the on-chip cache sizes, so this is something to be keenly aware of when writing and testing your code on ARM. When developing CUDA algorithms for ARM, it is very important to avoid host-GPU communication wherever possible and to make efficient use of PCIe bandwidth, much moreso than on x86 platforms.

Download the CUDA 5.5 RC Toolkit with ARM support today!

‖∀