Wrong Way: Lessons Learned and Possibilities for Using the "Wrong" Programming Approach on Leadership Computing Facility Systems
Date: February 16, 2022
Presented by: Philip C. Roth (Oak Ridge National Laboratory)

Q. Does anyone know whether there is development for OpenACC on Intel Level Zero? For comparison, for NVIDIA GPUs, one can use NVHPC, and it is possible to run OpenACC on AMD GPUs with GCC >10 (if configured properly).

A. I don't know of an effort specifically for OpenACC with the Level Zero runtime. It might be interesting to see if the scenario could be accomplished using Clacc (https://csmd.ornl.gov/project/clacc) to convert OpenACC code into OpenMP code, and then use Intel oneAPI's support to run the OpenMP code.

C. Questions about binary analysis, its capabilities to understand performance, or what Dyninst is? I'm a Dyninst dev, so feel free to contact me (thaines@cs.wisc.edu) or submit a question at github.com/dyninst/dyninst.

Q. Does Dyninst support GPUs? Something that project Freud did by instrumenting PTX and NVIDIA assembly?

A. Dyninst currently supports NVIDIA and AMD GPUs. It can do rewriting on NVIDIA binaries directly. We chose not to go through the textual representations like PTX or assembly because the compiler can mess up our instrumentation code- we mostly use trampolines which do not like being inlined. We have parsing of instructions and DWARF for AMD. We don't yet support their extended DWARF. We are working on instrumentation there. We have very rudimentary support for Intel GPUs. We just haven't had access to hardware to work on that.

Q. Stupid question: how to use SYCL/DPC++ from Fortran? Are there solutions such as mixed language programming a la HIPfort (where kernels are in HIP, and one needs to write ISO_C_BINDING interfaces for the Fortran part of the code)?

A. I don't know of anything like hipfort for SYCL/DPC++. That may be because HIP has an API with C bindings and represents kernels as separate functions, whereas SYCL and DPC++ are single-source C++-based languages. If not using directives like OpenMP or OpenACC, any solution I can think of involves calling from Fortran code to a C++ function with a C binding, and then launching the kernel from the C++ stub. The trickiest part seems to be making sure that the C++ code receives the interprets the data from the Fortran side correctly. The ISO_C_BINDING and C_PTR seem to be essential to this.

While it isn't SYCL, Matt Norman from ORNL and some colleagues have been developing a framework called YAKL (Yet Another Kernel Launcher, https://github.com/mrnorman/YAKL) that serves a similar purpose to portability layer software like Kokkos or RAJA, but also is designed to ease the calling of kernels from Fortran code. It demonstrates the approach above of Fortran code calling into C++ functions that launch the kernels. Although it does not use Kokkos, it has

some Kokkos interoperability as described in the YAKL wiki (https://github.com/mrnorman/YAKL/wiki/YAKL-Interoperation-with-Kokkos).

Q. Is Fortran (or C++) the right way to program on HPC machines? (given there is no parity in support across compilers and vendors)

A. This is a touchy subject! There are many reasons to like Fortran, and a substantial amount of the cycles on our production systems at OLCF are consumed executing Fortran code. But there are also many reasons to like C++, and it ***feels*** like these days the people providing compilers and productivity tools (e.g., debuggers, profilers/tracers) have stronger support for C++ before they have strong support for Fortran.

Q. Both mpich and openmpi are CUDA aware. Is there anyone out there working on openCL, HIP and SYCL aware MPI?

A. To me, this is more a question of whether an MPI implementation is "GPU aware" (i.e., can transfer data to/from buffers in GPU memory) rather than specifically "CUDA-aware" or "HIP-aware." MPICH 4.0.1 looks like is GPU aware, and can use the CUDA, Level Zero, or HIP runtime. (See the README in the MPICH 4.0.1 source code release.) Certainly, the Cray MPICH implementation available on HPE systems like OLCF's Crusher (and eventually Frontier) is also GPU aware.

Q. Why hasn't anyone worked on something like "write once, run everywhere" for GPU?

A. My guess is that the developers of software like HIP, Kokkos, RAJA, OCCA, OpenMP, etc. feel like they *are* working on that "write once, run everywhere" solution. If only people could agree on which one had the right perspective and convince everyone else to convert to their approach. I'm joking, of course. It may just be too early for there to be a "winner" in terms of mind share, and it may also be that there will *never* be a "winner" because of the disparate goals/priorities of the projects that develop software to run on GPU.

Q. Have you tested MAGMA on Crusher/Frontier?

A. Yes, several of the projects from the OLCF Center for Accelerated Application Readiness (CAAR) program and the Exascale Computing Project use MAGMA on Crusher.

Q. Are there SYCL implementations of the SHOC benchmarks?

A. Zheming Jin from ORNL has produced SYCL versions of (most? all?) the SHOC benchmarks, available at https://github.com/zjin-lcf/HeCBench. E.g., search that page for maxflops-sycl, the SYCL version of the SHOC MaxFlops benchmark.

Q. Any experience with accelerating via OMP-target?

A. I confess, I "naturally" think in C++ and prefer an approach that can represent kernels as lambdas. I myself have done minimal testing/performance measurement of code that uses OpenMP target to run code on a GPU. My understanding from others who have worked with applications using OpenMP target offload is that the compiler support is maturing, but that there

might still be a performance gap between the OpenMP target version of a code and one that uses a different approach to programming for the GPU.

Q. How about ISO C++ Parallel vs SYCL?

A. I love the idea that this problem could be addressed for C++ code within an ISO standard. Right now, the only implementation of C++17 parallel algorithms for a GPU that I know of is in the C++ compiler of the NVIDIA HPC SDK, which only supports NVIDIA GPUs. One might argue that it would be best for the community overall if "someone" were to invest time and effort and funding into adding GPU capability to a Standard Library implementation like libc++, but it's not easy to see who that someone should be. Applications targeting performance portability from GPUs are not (yet?) written to use C++ parallel algorithms because of the lack of implementations, and people who might provide the implementations aren't prioritizing it because of the lack of applications using it.

Lacking more than one implementation of the ISO C++ Standard Library for GPUs, other approaches like SYCL are (at least slightly) more portable.

Q. What about reproducibility? Scientists need to be able to reproduce analysis on specific software versions/options.

A. While I agree with the sentiment, I'm not entirely sure what the asker had in mind relative to the "wrong way" topic. The topic of reproducibility seems to be orthogonal to whether someone's GPU code is written in one of the target system's natural ways, or an "off the beaten path" way. In either case, for reproducibility it is important to capture information about the software and hardware used, e.g., by noting the modules loaded and the OS version. As noted in the talk, it might be possible to improve reproducibility across systems compared to a list of module names and versions if the software used is captured in a container and the systems support running containerized software.

Q. Which space(s) do we watch for updates on the "right way?"

A. Someone helpfully entered the URL to a blog post about NVIDIA's HPC SDK for its C++ standard library parallel implementation (https://developer.nvidia.com/blog/developing-accelerated-code-with-standard-language-parallelism). But as noted in the ISO C++ Parallel question above, it's a pretty constrained solution until there are more implementations or support for more than one vendor's GPUs. My best suggestion is to look to the computing facilities for their system-specific user guides and training resources, their annual user meetings and their monthly user telecons. There's a lot of good information available, and a lot of experience on offer.