SYCL – Introduction and Best Practices

(the slides are available under "Presentation Materials" in the above URL)
Date: June 10, 2020
Presented by: Thomas Applencourt (Argonne National Laboratory)

Q. I have a Windows 10 Workstation with a NVIDIA GPU. What compiler/toolchain should I use for SYCL development?

A. CodePlay support building for Windows but I'm not sure it supports targeting NVIDIA GPUs. (Kevin - ANL)
https://developer.codeplay.com/products/computecpp/ce/guides/platform-support/targeting-windows#bottom

Q. However much I would like to develop in C++, much (most?) of HPC code right now is in Fortran. Are there Fortran bindings for SYCL ?

A. SYCL uses significant C++ language features such as templates, lambdas, etc.. which makes it difficult to use in FORTRAN. As Thomas stated, either use OpenMP or a SYCL implementation of a core routine can be wrapped in a C wrapper and called from FORTRAN using ISO C bindings. (Kevin - ANL)
A. Intel Fortran will be able to offload to GPU. GPU support will be provided via OpenMP.

Q. Is there a performance penalty for USM vs. manual data management?

A. Performance penalties associated with USM and/or manual data management are functions of the hardware capabilities or runtime implementations. There is nothing specific in the SYCL specification that would cause USM to perform worse than manual data management. (Kevin - ANL)

A. USM provides mechanisms to control the data transfer. There is no penalty for the use of USM. It simplifies programming. However, a more experienced user who wants to be in charge of data transfer can write the code in the way that makes it clear when the runtime will initiate the data transfer between the host and device. Profiling the code by using VTune will show all data transfers.

Q. planning to port our CUDA kernels, our code uses multiple CUDA warp shuffle intrinsics (for sharing data between two CUDA threads), is there similar support for SYCL available?

A. There is currently no support for things like shuffle intrinsics in the SYCL specification, however, vendor specific extensions could support such a feature. (Kevin - ANL)

Q.I noticed that there is a SYCL-BLAS library. What about SYCL-FFT? *(There is mention here ~2018 that it is on the roadmap… any updates?)* If not, is there a way of using, say, cuFFT while writing a SYCL application that would work on NVIDIA machines in the meantime?

A I'm not aware of a public effort to create a general Sycl-FFT. Saying that, different vendors will provide different solutions. For example, Intel will provide SYCL port of various FFT libraries. If you are interested in Beta test them, please send us an email.
Due to the interabiliies between SYCL and OpenCL, one can also call https://github.com/clMathLibraries/clFFT in their code.
Also because sycl is C++, you can always call any C api in your code (for example cuFFT)

Q. Is there a library of SYCL accessors to cover the common needs of (ECP) developers? Or do I have to write my own specifically for my code?

A. Usual data-pattern (read, write, read/write) have assessors. If you have more specifics data-pattern of access we are general to HPC application, please send a email to us.

Q. Will a Fortran code that uses OpenMP be able to run assuming that the USM 'automatic data handling' mentioned above will take care of data transfers or does one explicitly need to do target data enter/exit/map?

A. Yes--In OpenMP, if you add "requires unified_shared_memory" and the hardware supports USM, you do not need to use data mapping like "target data enter/exit". (in this case, it is up to the hardware/driver to support USM. you can mark your code with "requires unified_shared_memory" to say that it's only valid if there is USM). So this is a bit different from the SYCL data transferring.

Q. Slightly off-topic and rambling, but "why C++"? What I am trying to understand is this: SYCL seems to provide a way of writing performant portable code that can target many platforms. I would really like to be able to take advantage of this, but prefer to write initial code in python, then write critical sections in an appropriate low-level language. (Much of our code uses libraries like BLAS or FFT for performance, so this currently works quite well.) Is there are good way of interacting with SYCL from other languages, or is it so reliant on C++ features that there is really not a good mapping from other languages to SYCL features.

For example, something that would be very useful would be to use SYCL code to allocate some sort of device array, then be able to wrap this in an object that could be passed back and used in a high-level language - possibly passed to existing CUDA or OpenCL code for some processing, then passed back. (Of course this would only work on specific targets).

As another example, it is really quite nice exploring things like CUDA and GPU programming from an interactive language like python since a developer can directly interact with the machine, query the hardware in real-time etc. (Great for teaching.)

A. (from another attendee): C++ and python codes can be written in a similar way (classes, inheritance, and constructors/destructors), but C++ is compiled and faster. Compilation provides type-checking (and compile-time code execution via templates) which can make code smaller than a stable python version. Some interesting examples of side-by-side C++ and python codes are in https://fenicsproject.org/ (python calls JIT-compile to C++).
A. See answers about Fortran. If you access to low level interface, SYCL is not the correct choice. You should use your favorite low-level interface (CUDA, OpenCL, Level-0).

Q. Why does example code use sycl::endl to print with std::cout?

A. We don't use std::cout but sycl::stream

Q. For an existing large application, how intrusive will it be to add SYCL? Is there experience how much code change is necessary?

A. This depends a lot on your application. But Using Unified Shared Memory, it should be self contained in the kernel part of your application.

Q. Is there any comparative performance data for a C++ code using SYCL versus using OpenMP to do exactly the same thing (much like the tests done in the RAJA performance test suite)?

A. Yes. Hope this help:
https://www.youtube.com/watch?v=5W6SsreZ3ew&feature=emb_logo
https://dl.acm.org/doi/pdf/10.1145/3388333.3388643
http://uob-hpc.github.io/2020/01/06/cloverleaf-sycl.html
https://sc19.supercomputing.org/proceedings/workshops/workshop_files/ws_p3hpc106s2-file1.pdf

Q. Is there a way to see the intermediate processing steps between source code and machine code? Specifically, it would be nice to see what is vectorized and where/how loops/array sizes are broken into tiles. Maybe there is an intermediate representation that has tile-prefetch information or something?

A. DPCPP can output its "SPRIV"(sp?) intermediate representation when given the appropriate flag. It's similar to LLVM IR. The later compile step to gen-smd on gen9 is also visible. It's easiest to ask Intel VTUNE for help with vectorization, etc.

Q. Does the CPU support of SYCL extend to KNL? Will this be usable on Theta?

A. Sycl uses the OpenCL backend by default. Intel doesn't support OpenCL on KNL. The open source OpenCL POCL might work

Q: Is there a way to see a log of the run-time decisions on data copies and device thread allocation / context switching?

A: You can use VTUNE. On JLSE (argonne system) you can use `iprof`.

Q: how could one express AVX 512 vectorization explicitly in sycl?

A: Try changing local workgroup size and adding -architecture- selection flags to force this. Inline assembly may be added to SYCL at some future date.

Q: I can't compile your stuff under intel devcloud

```
make
0_tiny_sycl_info.cpp:1:10: fatal error: CL/sycl.hpp: No such file or directory
 #include <CL/sycl.hpp>
```

A: solved by dpcpp -fsycl-unnamed-lambda 0_tiny_sycl_info.cpp -std=c++17 -o 0.out

Q: is it possible to use all the LLVM infrastructure for getting structured reports from opt passes (in dpcpp)?

A You can use traditional llvm flags to get the opt-report of the SPIR-V generation. The transformation of SPIRV into machine code (where the “vectorization” occurs in Intel platform) is not handled by LLVM but by another compiler (and at runtime by default). Hence I don’t think you can get an optimization report for this part yet.
This is an interesting remark, I will provide feedback to Intel on how we can improve your experience.

Q. We are also looking to use Summit, as well as Aurora. What SYCL support is available for the Power9 platform?

A. My understanding is that Codeplay (https://www.codeplay.com/) SYCL compiler supports Power9.