SYCL Programming Model for Aurora

ECP Annual Meeting
Feb. 3-7, 2020
Argonne Leadership Computing Facility and Intel Co.

Yasaman Ghadar, Kevin Harms, Jeff Hammond, Thomas Applencourt, Brian Homerding
Outline

10:30a  Aurora Overview
10:45a  Motivation for SYCL/DPC++
11:00a  oneAPI Overview
11:15a  SYCL Structure and Examples
11:45a  Q & A

Yasaman Ghadar
Kevin Harms
Jeff Hammond
Thomas Applencourt
All
Aurora Overview: Hardware

- Intel-Cray machine arriving at Argonne in 2021 with sustained Performance > 1Exaflops
- Intel Xeon processors and Intel Xe GPUs
  - 2 Xeons (Sapphire Rapids)
  - 6 GPUs (Ponte Vecchio [PVC])
- Greater than 10 PB of total memory
- Cray Slingshot network and Shasta platform
- Filesystem
  - Distributed Asynchronous Object Store (DAOS)
    - Greater 230 PB of storage capacity and bandwidth of > 25 TB/s
  - Lustre
    - 150 PB of storage capacity and bandwidth of ~1TB/s
Aurora Overview: Software

- Software (Intel oneAPI umbrella):
  - Intel compilers (C, C++, Fortran)
  - Programming models: DPC++, SYCL, OpenMP, OpenCL
  - Libraries: oneMKL, oneDNN, oneDAL
  - Tools: VTune, Advisor
  - Python

- Overview of the Argonne Aurora Exascale System, Wed. @ 2:30 Legend Ballroom
- DAOS (Distributed Asynchronous Object Storage) for Applications Thu. @ 8:30am Discovery A
Node-level Hardware
The Evolution of Intel GPUs

Source: Intel
Intel Discrete Graphics 1 (DG1):

- Intel Xe DG1: Software Development Vehicle (SDV) Shown At CES 2020

- Xe DG1 GPU is part of the low power (LP) series and is **not** a high-end product.

- Chip will be used in laptops with GPU integrated on to the motherboard PCIe form factor is only for the SDV and not a product

- The SDV is shipping worldwide to integrated software vendors (ISVs) to enable broad software optimization for the Xe architecture.
Aurora Compute Node

- 2 Intel Xeon (Sapphire Rapids) processors
- 6 Xe Architecture based GPUs (Ponte Vecchio)
  - All to all connection
  - Low latency and high bandwidth
- 8 Slingshot Fabric endpoints
- Unified Memory Architecture across CPUs and GPUs

Overview of the Argonne Aurora Exascale System
2:30pm - 3:30pm, Feb 5
Legends Ballroom
**Intel GPUs**

- Recent and upcoming integrated generations:
  - **Gen 9** – used in Skylake based processors
  - **Gen 11** – used in Ice Lake based processors

- Gen 9: Double precision peak performance: 100-300 GF
  - Low by design due to power and space limits

- Future **Intel Xe** GPU series will be:
  - Integrated
  - Discrete GPUs

---

Layout of Architecture components for an Intel Core i7 processor 6700K for desktop systems (91 W TDP, 122 mm)
GPU Architectures

- GPU architectures have hardware hierarchies
  - Built out of smaller scalar units each level shares a set of resources
<table>
<thead>
<tr>
<th>Nvidia/CUDA Terminology</th>
<th>AMD Terminology</th>
<th>Intel Terminology</th>
<th>Description (pulled almost word-for-word from AMD slides)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Streaming Multiprocessor (SM)</td>
<td>Compute Unit (CU)</td>
<td>SubSlice (SS)</td>
<td>One of many independent parallel vector processors in a GPU that contain multiple SIMD ALUs.</td>
</tr>
<tr>
<td>Kernel</td>
<td>Kernel</td>
<td>Kernel</td>
<td>Functions launched to the GPU that are executed by multiple parallel workers on the GPU.</td>
</tr>
<tr>
<td>Warp</td>
<td>Wavefront</td>
<td>Vector thread, issuing SIMD instruction</td>
<td>Collection of operations that execute in lockstep, run the same instructions, and follow the same control-flow path. Individual lanes can be masked off.</td>
</tr>
<tr>
<td>Thread block</td>
<td>Workgroup</td>
<td>Workgroup</td>
<td>Group of warps/wavefronts/vector threads that are on the GPU at the same time. Can synchronize together and communicate through local memory.</td>
</tr>
<tr>
<td>Thread</td>
<td>Work item/Thread</td>
<td>Work item/Vector lane</td>
<td>Individual lane in a warp/wavefront/vector thread</td>
</tr>
<tr>
<td>Global Memory</td>
<td>Global Memory</td>
<td>GPU Memory</td>
<td>DRAM memory accessible by the GPU that goes through some layers of cache</td>
</tr>
<tr>
<td>Shared memory</td>
<td>Local memory</td>
<td>Shared local memory</td>
<td>Scratchpad that allows communication between warps/wavefronts/vector threads in a threadblock/workgroup</td>
</tr>
<tr>
<td>Local memory</td>
<td>Private memory</td>
<td>GPRF</td>
<td>Per-thread private memory, often mapped to registers.</td>
</tr>
</tbody>
</table>
Motivation for SYCL/DPC++

Kevin Harms
Why SYCL?

- What are the benefits?

- Creating a greenfield application or complete rewrite and chose C++ as your base language
- SYCL is a high performance solution for Aurora supported by Intel
- Coming from CUDA or HIP and you want to embrace an open standard
- Multiple SYCL implementations supporting various backend hardware
- High performance and portable across implementations
Mapping of existing programming models to Aurora

- **OpenMP w/o target**
- **OpenMP with target**
- **OpenACC**
- **OpenCL**
- **CUDA/HIP**
- **DPC++/SYCL**
- **Kokkos**
- **Raja**
DPC++ (Data Parallel C++) and SYCL

- **SYCL**
  - Khronos standard specification
  - SYCL is a C++ based abstraction layer (standard C++11)
  - Builds on OpenCL concepts (but single-source)
  - *SYCL is standard C++17*

- **Current Implementations of SYCL:**
  - ComputeCPP™ (www.codeplay.com)
  - Intel SYCL (github.com/intel/llvm)
  - triSYCL (github.com/triSYCL/triSYCL)
  - hipSYCL (github.com/illuhad/hipSYCL)

- **Runs on today’s CPUs and nVidia, AMD, Intel GPUs**
DPC++ (Data Parallel C++) and SYCL

- **SYCL**
  - Khronos standard specification
  - SYCL is a C++ based abstraction layer (standard C++11)
  - Builds on OpenCL concepts (but single-source)
  - *SYCL is standard C++17*

- **Current Implementations of SYCL:**
  - ComputeCPP™ (www.codeplay.com)
  - Intel SYCL (github.com/intel/llvm)
  - triSYCL (github.com/triSYCL/triSYCL)
  - hipSYCL (github.com/illuhad/hipSYCL)

- **Runs on today’s CPUs and nVidia, AMD, Intel GPUs**

- **DPC++**
  - Part of Intel oneAPI specification
  - Intel extension of SYCL to support new innovative features
  - Incorporates SYCL 1.2.1 specification and Unified Shared Memory
  - Add language or runtime extensions as needed to meet user needs

---

### Extensions

<table>
<thead>
<tr>
<th>Extensions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified Shared Memory (USM)</td>
<td>defines pointer-based memory accesses and management interfaces.</td>
</tr>
<tr>
<td>In-order queues</td>
<td>defines simple in-order semantics for queues, to simplify common coding patterns.</td>
</tr>
<tr>
<td>Reduction</td>
<td>provides reduction abstraction to the ND-range form of parallel_for.</td>
</tr>
<tr>
<td>Optional lambda name</td>
<td>removes requirement to manually name lambdas that define kernels.</td>
</tr>
<tr>
<td>Subgroups</td>
<td>defines a grouping of work-items within a work-group.</td>
</tr>
<tr>
<td>Data flow pipes</td>
<td>enables efficient First-In, First-Out (FIFO) communication (FPGA-only)</td>
</tr>
</tbody>
</table>

SYCL Ecosystem

SYCL source code

- **clang** (Intel SYCL)
  - Any CPU
  - OpenCL + SPIR-V
    - Intel CPUs
    - Intel GPUs
    - AMD GPUs (depending on driver stack)
    - ARM Mali
    - Renesas R-Car

- **ComputeCpp**
  - Any CPU
  - PTX devices

- **triSYCL**
  - Experimental!

- **hipSYCL**
  - Any CPU
  - OpenCL + SPIR-V
    - NVIDIA GPUs

- **sycl-gtx**
  - OpenCL 1.2
    - Pretty much anything 😊

- **Any CPU**

- **OpenCL + SPIR-V**
  - Intel CPUs
  - Intel GPUs
  - AMD GPUs (depending on driver stack)
  - ARM Mali
  - Renesas R-Car

- **OpenCL + SPIR-V**
  - NVIDIA GPUs

- **OpenCL + SPIR-df**
  - ROCm
  - AMD GPUs
  - NVIDIA GPUs

- **CUDA**
  - NVIDIA GPUs

https://github.com/lllahad/hipSYCL/blob/master/doc/img/sycl-targets.png
SYCL Performance Across Architectures - LCALS (RAJA Perf)
Intel oneAPI and DPC++

Jeff Hammond
oneAPI Industry Initiative

Optimized Applications

Optimized Middleware & Frameworks

oneAPI Industry Specification

Direct Programming

Language

Host Interface

API-based Programming

Libraries

Low-Level Hardware Interface

CPU

GPU

FPGA

Specialized Accelerators

https://www.oneapi.com/
oneAPI Industry Initiative

The DPC++ language is:
- ISO C++17
- SYCL 1.2.1
- Extensions proposed to SYCL-Next
- Other extensions (rare)

DPC++ language extensions are documented and implemented as open-source here:

https://github.com/intel/llvm/tree/sycl/

oneAPI libraries include:
- oneDPL: parallel STL plus extensions that work with DPC++ programs.
- oneMKL: MKL subset for DPC++ programs (i.e. device support)
- oneDNN: deep learning kernels
- oneCCL: collectives library for AI (descended from Intel MLSL)
- Level Zero: low level runtime that supports both C and C++

https://www.oneapi.com/
DPC++ Extensions: What and Why?

Unified Shared Memory (USM)
What: Pointer-based memory management.
Why: (1) Compatibility with C-oriented legacy software designs.
     (2) Hard to use accessors inside of Kokkos and RAJA.

Optional lambda name
What: SYCL requires named lambdas in kernels.
Why: (1) Many implementations don’t need named kernels.
     (2) Hard to generate unique names in some contexts (K&R C++).

In-order queues
What: SYCL queues are out-of-order, unlike OpenCL.
Why: Makes it easier to write some programs, particularly when USM is used.
DPC++ Extensions: What and Why?

Reductions
What: Reductions.
Why: (1) Every sane HPC programming model has reductions.
(2) Application programmers should not be required to roll their own.

Subgroups
What: Similar to OpenCL 2.1 extensions (Khronos and Intel)
Why: More control over hardware resources.

Data flow pipes
What: Producer-consumer objects for pipelined execution.
Why: FPGA programs make heavy use of pipes.
SYCL References

Spec
https://www.khronos.org/registry/SYCL/specs/sycl-1.2.1.pdf

Examples
https://github.com/ParRes/Kernels
https://github.com/kevin-harms/sycltrain/tree/master/9_sycl_of_hell
https://github.com/codeplaysoftware/computecpp-sdk/tree/master/samples
https://github.com/homerdin/RAJAPerf

Documentation
https://github.com/jeffhammond/dpcpp-tutorial

Tools

Training
https://www.colfax-intl.com/training/intel-oneapi-training
Question and Answer Session

- Who is using SYCL/DPC++ for their project?
- What is/would prevent you from using SYCL?
- What do you want from SYCL?
- What is missing from SYCL?
- What would you want to know about in future sessions?
- How do you want to use library (MKL, etc.) routines, as kernels or within kernels?
Acknowledgements

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation’s exascale computing imperative.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.