Comparing SyCL data transfer strategies for tracking use cases

Sylvain Joube[1,2], Hadrien Grasland[1], David Chamont[1], Elisabeth Brunet[2]

[1]Université Paris-Saclay, CNRS/IN2P3, IJCLab, 91405 Orsay, France
[2]Télécom SudParis, Institut Polytechnique de Paris, Samovar, France

ACAT 2021
29 NOV - 03 DEC, 2021

SyCL[1] is a royalty-free, cross-platform abstraction layer that enables code for heterogeneous processors to be written using standard ISO C++ with the host and kernel code for an application contained in the same source file.

SyCL implementation & backend

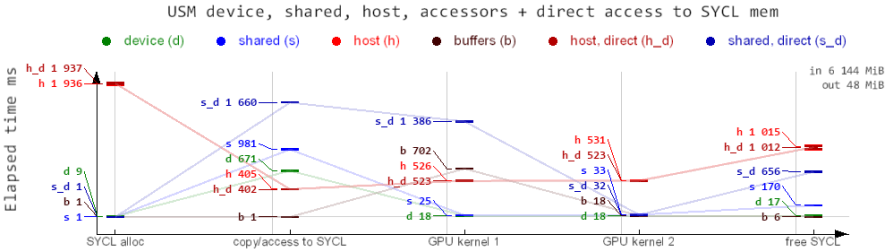
- hipSYCL[2] 0.9.1
- Cuda release 10.1, V10.1.243
- NVidia driver 460.73.01

Hardware

- 512 GB DDR4, 2400 MHz
- NVidia Quadro RTX 5000: 16 GB GDDR6, 448 GB/s
- Debian 11.0 stable
- PCIe 3.0 x16 (16 GB/s theoretical, 12 GB/s observed)
- AMD EPYC 7502, 32 cores

Microbenchmark

The aim of this microbenchmark is to give a first approximation of the relative performance of the SyCL memory models on a single hardware architecture, and to compare it with what is theoretically expected. The microbenchmark is a simple reduction on GPU, the sum of about 1.6 billion randomly generated 4-byte floats. Two kernels were launched each time, one after the other, to identify what is done in a lazy way.



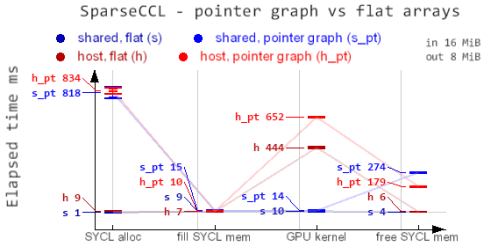
The first three curves (USM device, shared, host) are made with an explicit SyCL memory copy from host to device. The underlying copy for USM device is a simple explicit cuda memcpy, taking 671 ms, accounting for an observed speed of about 10 GB/s. It's kernel is processed in 18ms, leading to an observed memory bandwidth of about 360 GB/s, relatively close to the theoretical 448 GB/s of this GPU. Given that the USM host kernel only takes about 520 ms (12.4 GB/s), its usage seems to fit the single-access-to-data use-case, avoiding to pay for the copy overhead of USM device, granted that the SyCL host memory is heavily reused to make the allocation cost negligible.

The accessors-buffers curve has a very similar performance profile to USM device, except the memory copy is performed the first time data is accessed (GPU kernel run 1).

USM host and shared "direct" happens when one directly writes in the SyCL memory from the host, instead of calling a SyCL explicit copy from host memory to SyCL memory. USM host is not impacted while USM shared pays a huge overhead: SyCL does not know that we access it in a write-only mode, so does a huge number of page faults in an asynchronous way and only finishes during the first kernel run. The second run however has a very close performance profile to the explicit SyCL copy of USM shared.

SparseCCL[3]

SparseCCL is an algorithm specifically designed for HEP tracking. It is in charge of the very first step of the track reconstruction: the connected component labeling. Here is a performance analysis using the various SyCL memory models.

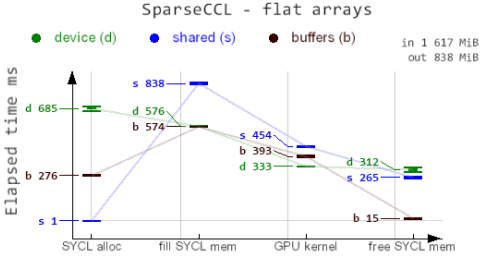


The allocation/free cost is huge when using pointer graphs (400 000 small allocations) compared to using regular flat arrays (4 big allocations) for USM shared and host. The kernel in itself does not seem to be heavily impacted.

However, USM shared is still much faster in kernel duration compared to USM host as data is used multiple times: the cost of passing through the PCIe bus is paid only once with USM shared, but each time data is accessed for USM host.

As seen before, USM shared does a number of things asynchronously. Memory allocation is only done when memory is accessed for the first time. Liberation however seems to be done in a synchronous way.

USM device and the accessors-buffers show a very similar performance profile, except the allocation is much more expensive for USM device: here the USM memory is initialized with specific flags to give the API more room to optimize its usage. The read_only, write_only and no_init flags does not yet exists in the USM world.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101004761.

[1]**SyCL**: https://www.khronos.org/registry/SYCL
[2]**hipSYCL**: https://github.com/hipSYCL
[3]**SparseCCL**: A. Hennequin, B. Couturier, V. V. Gligorov and L. Lacassagne, "SparseCCL: Connected Components Labeling and Analysis for sparse images," 2019 Conference on Design and Architectures for Signal and Image Processing (DASIP), 2019, pp. 65-70, doi: 10.1109/DASIP48288.2019.9049184.