SYCL memory models & SparseCCL

Master 2 internship -> PhD student

IJCLab - CNRS

David Chamont

Hadrien Grasland

SAMOVAR - Télécom SudParis

Elisabeth Brunet

Gaël Thomas

Sylvain Joube - 2021-09-17

ACTS Parallelization Meetings

Outline

Why SYCL & SparseCCL

Investigation with a micro-benchmark

- Explicit memory copy
- In-place filling

Investigation with SparseCCL

- Graph pointer
- Flat arrays

Conclusion & future work

Why SYCL ?

Similar to C++ standard

Compatible with many hardware chips

Based on an international consortium

Independent from any single manufacturer

Multiple memory models

SYCL USM memory models

Unified Shared Memory [USM, SYCL 2020]

- "USM device" : located on GPU, explicit transfer
- "USM host" : host-pinned, device-accessible [main memory]
- "USM shared" : implementation decides data location

Buffers and accessors [not tested here, SYCL 1.2]

- *Dependency graph between kernels*

Why SparseCCL?

Easy to understand and implement

First step in the track reconstruction

Only relies on csv input data

First results hard to understand...

Why a microbenchmark ?

Test SYCL memory on a simple memory-bound program

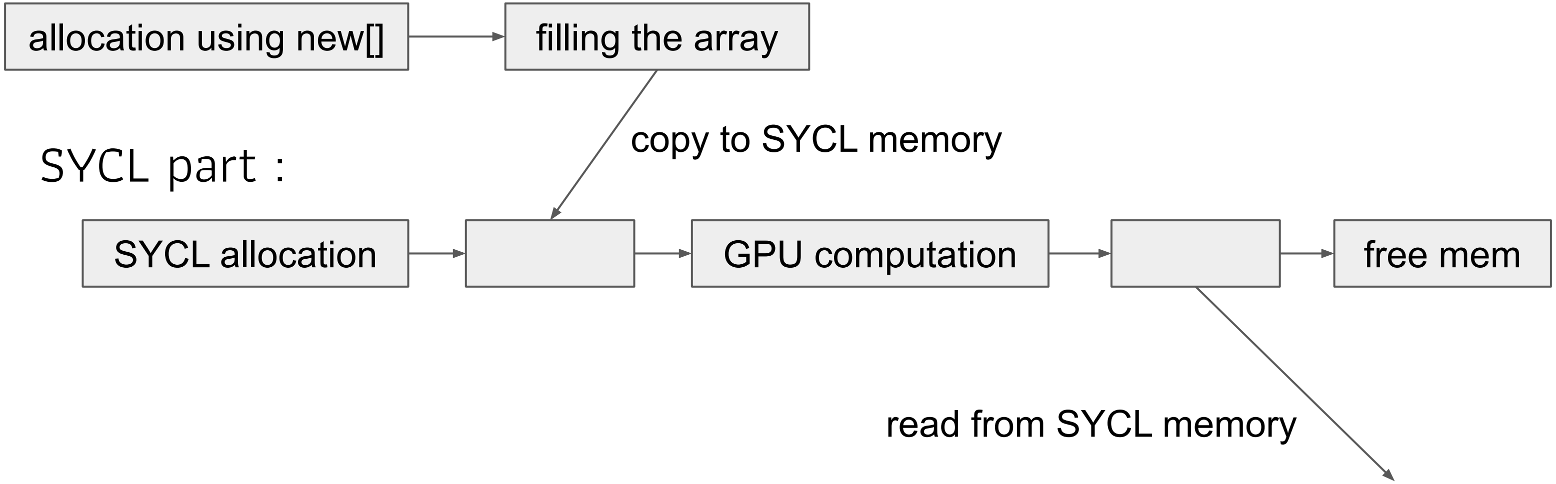
Understand what to expect from SparseCCL on SYCL

Data read only once

Microbenchmark : a simple reduction on GPU

Two main steps :

- Generic C++ code on CPU :



- SYCL part :

What will be measured here ?

SYCL memory behaviour

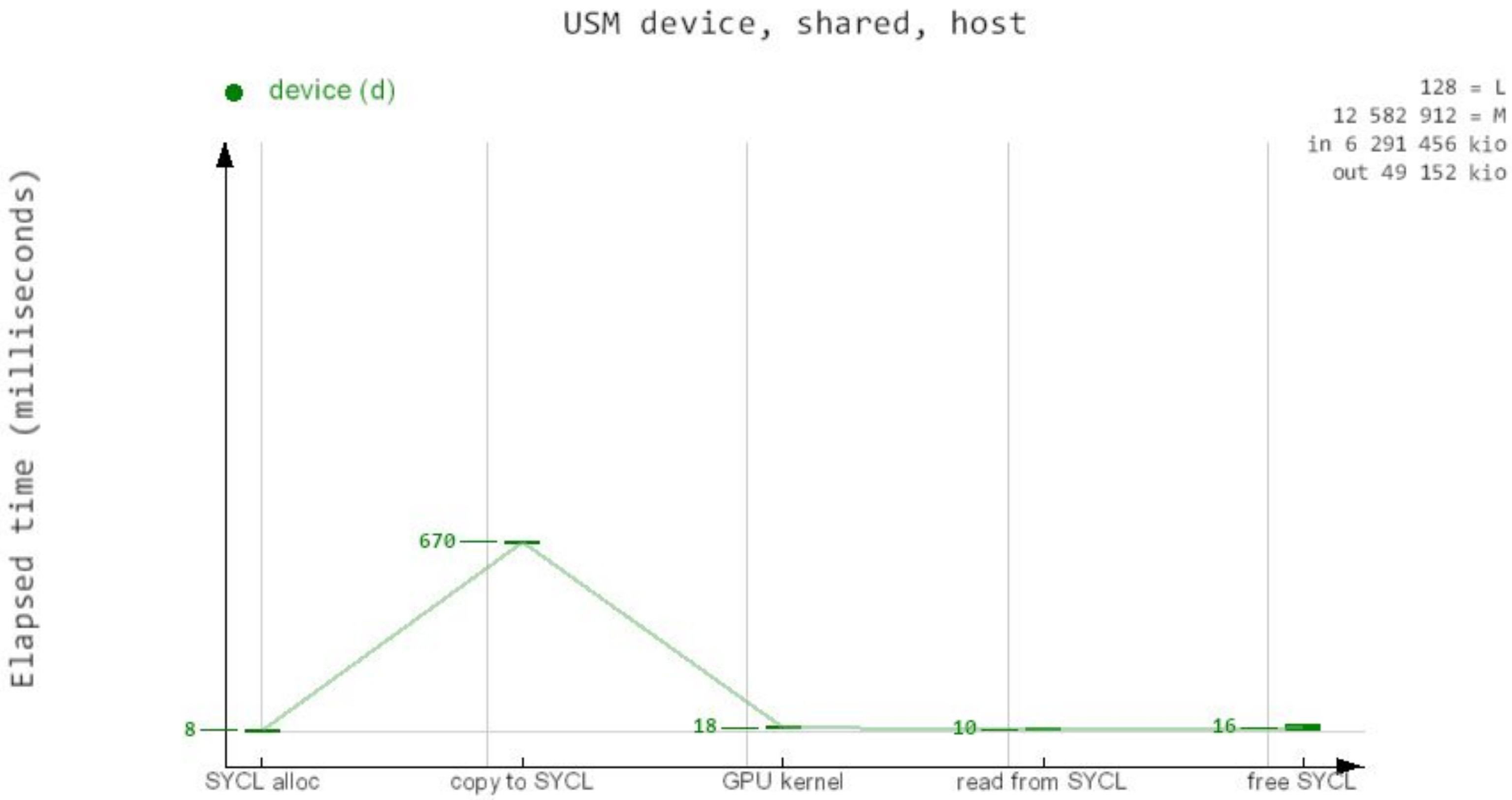
- With all USM models [device, shared, host]
- On a single server[1]
- Microbenchmark + SparseCCL

Dedicated server :

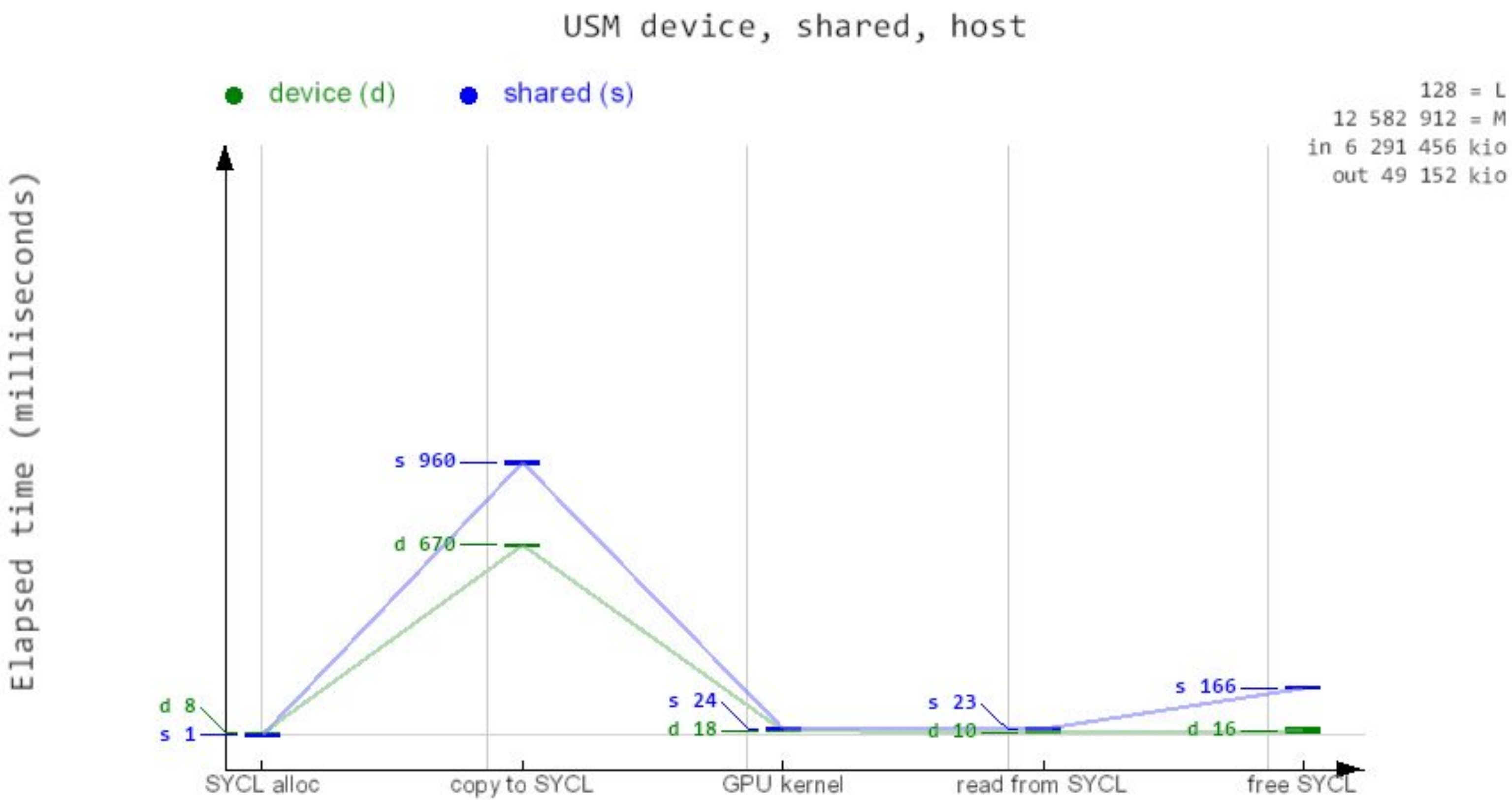
- 512 GB DDR4, 2400 MHz
- NVidia Quadro RTX 5000 : 16 GB GDDR6, 448 GB/s
- Debian 11.0 stable

[1]*singe server : but consistent quick cross-checked elsewhere with other hardware*

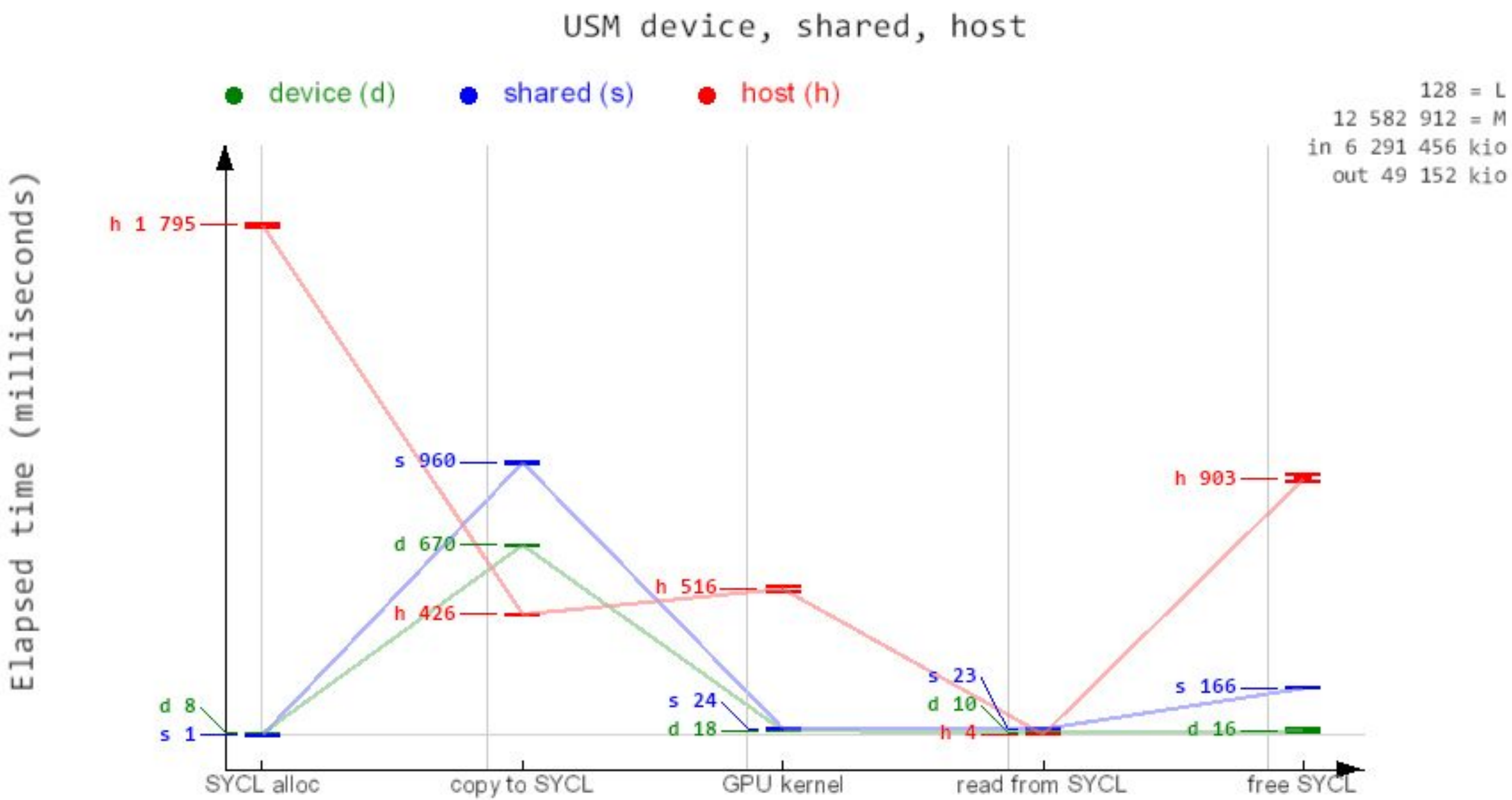
USM models comparison



USM models comparison



USM models comparison



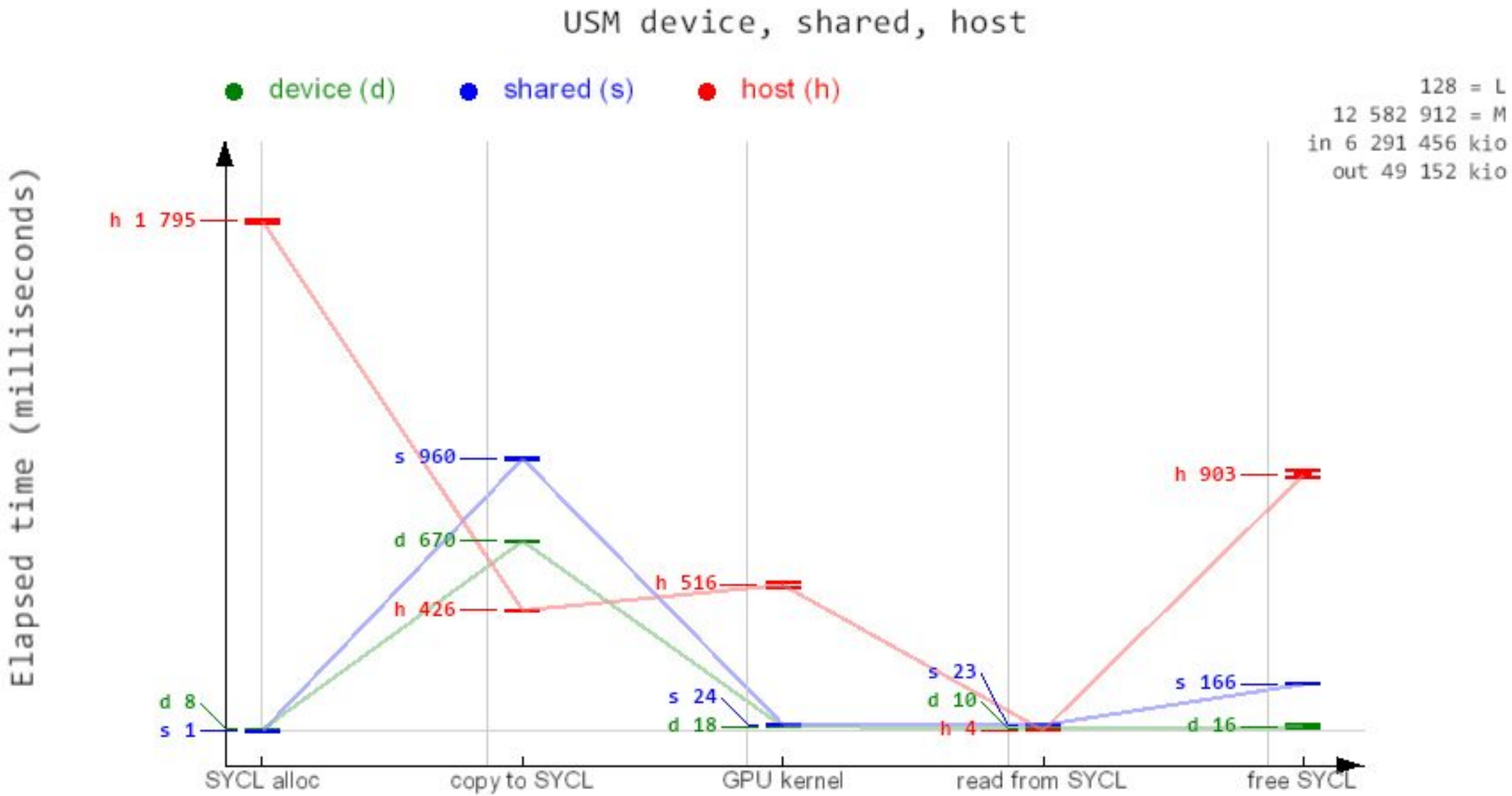
First findings

USM host :

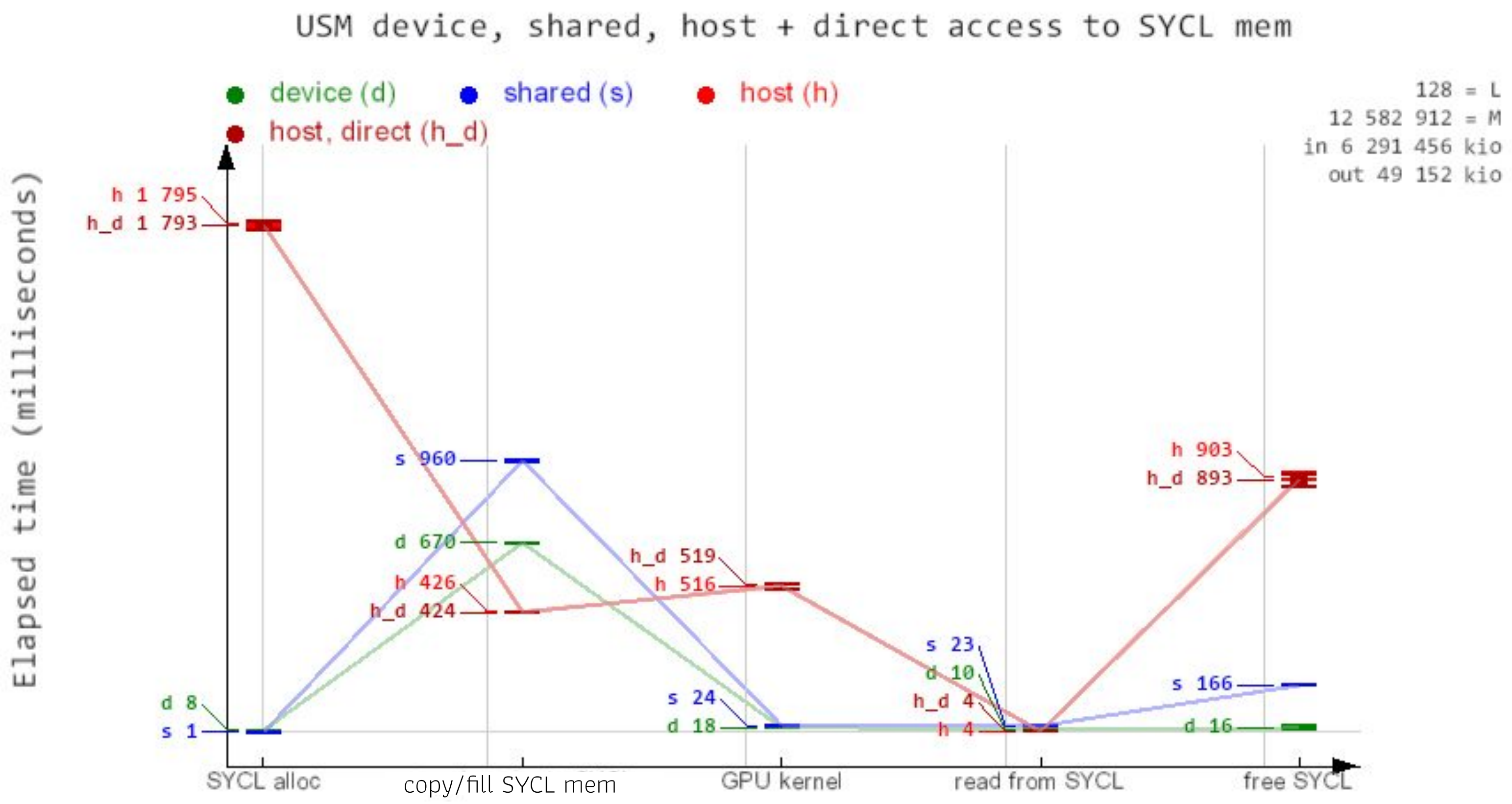
- for unique memory accesses
- should reuse allocated memory

USM host and shared : also for data bigger than GPU memory

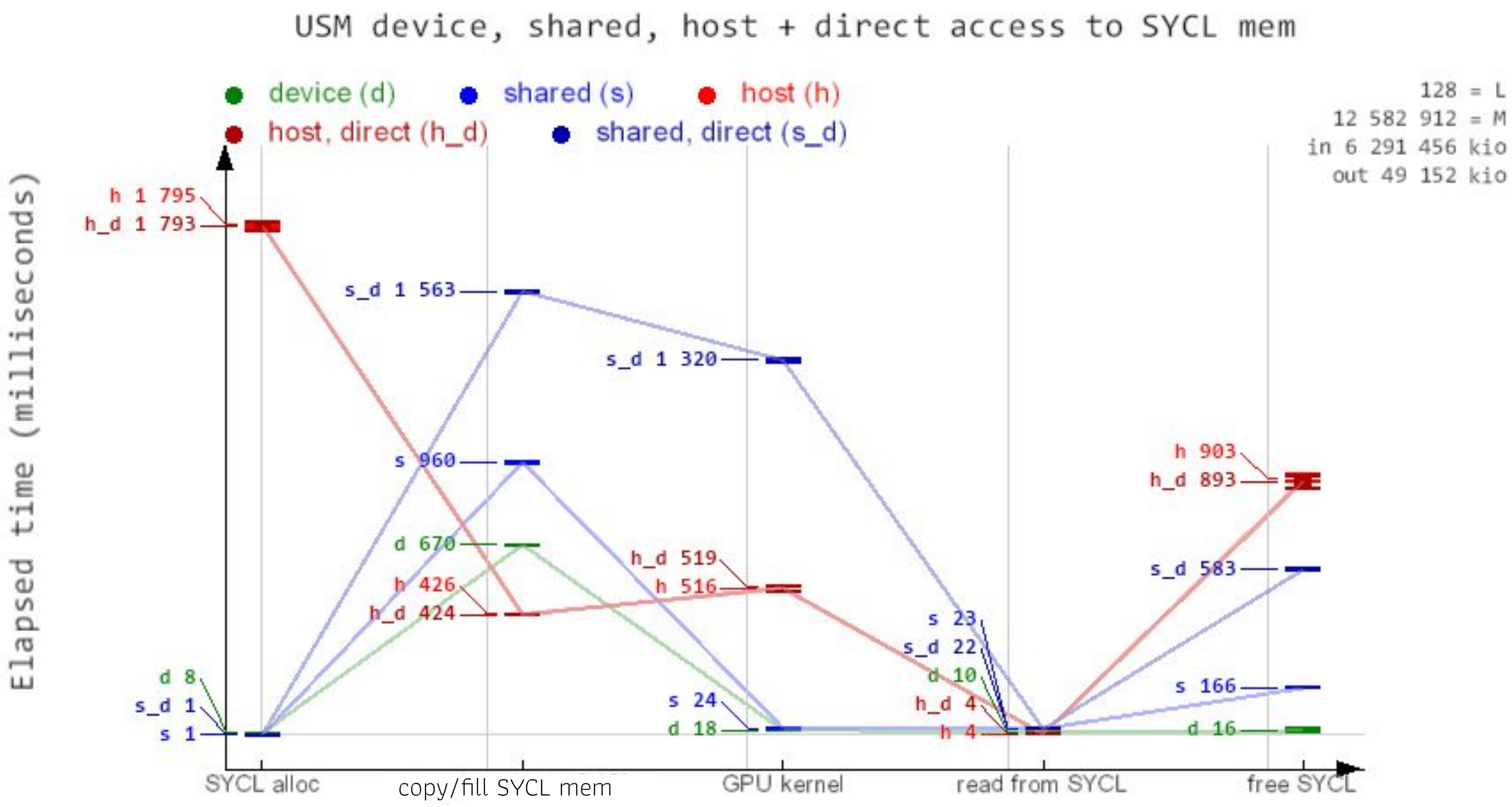
In-place SYCL memory filling ?



In-place SYCL memory filling ?



In-place SYCL memory filling



More findings

USM shared :

- very expensive in-place filling

USM host :

- can be used as a local memory on host, with no extra cost

SparseCCL

Questions :

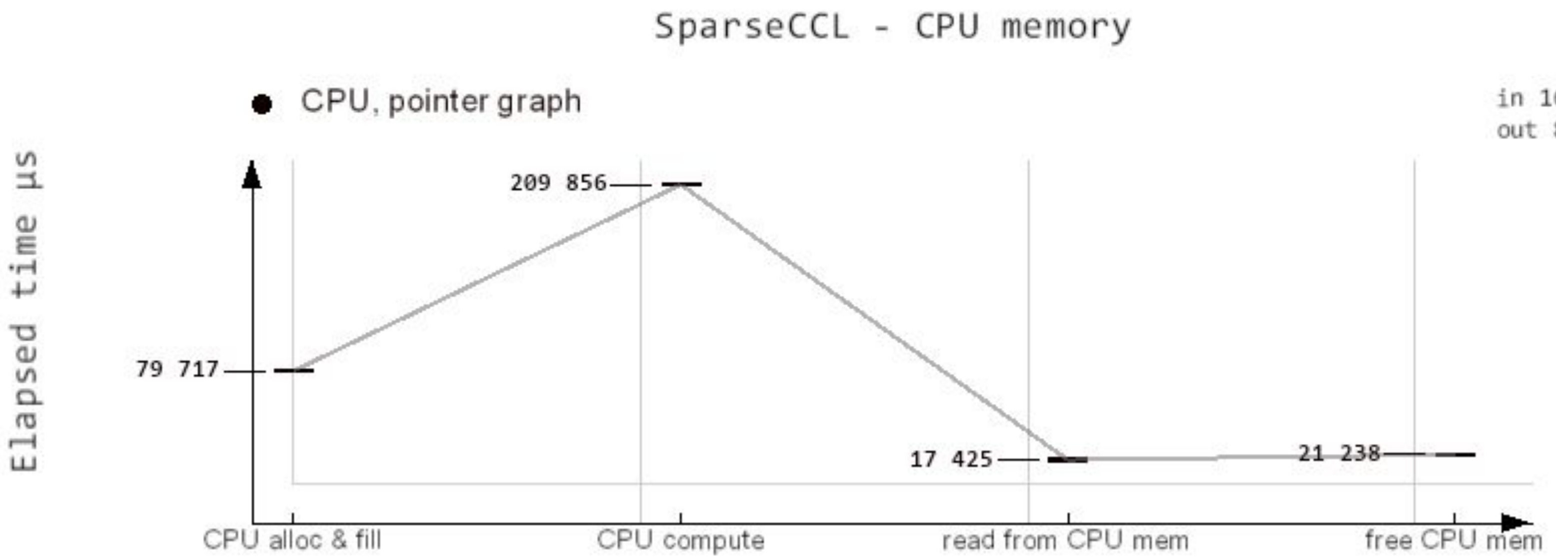
- Previous results still applies ?
- How does the USMs compare ?
- How hard is adapting existing code to SYCL ?

Data structures options :

1. Pointer graph : usual structures
2. Flat arrays : made for GPUs

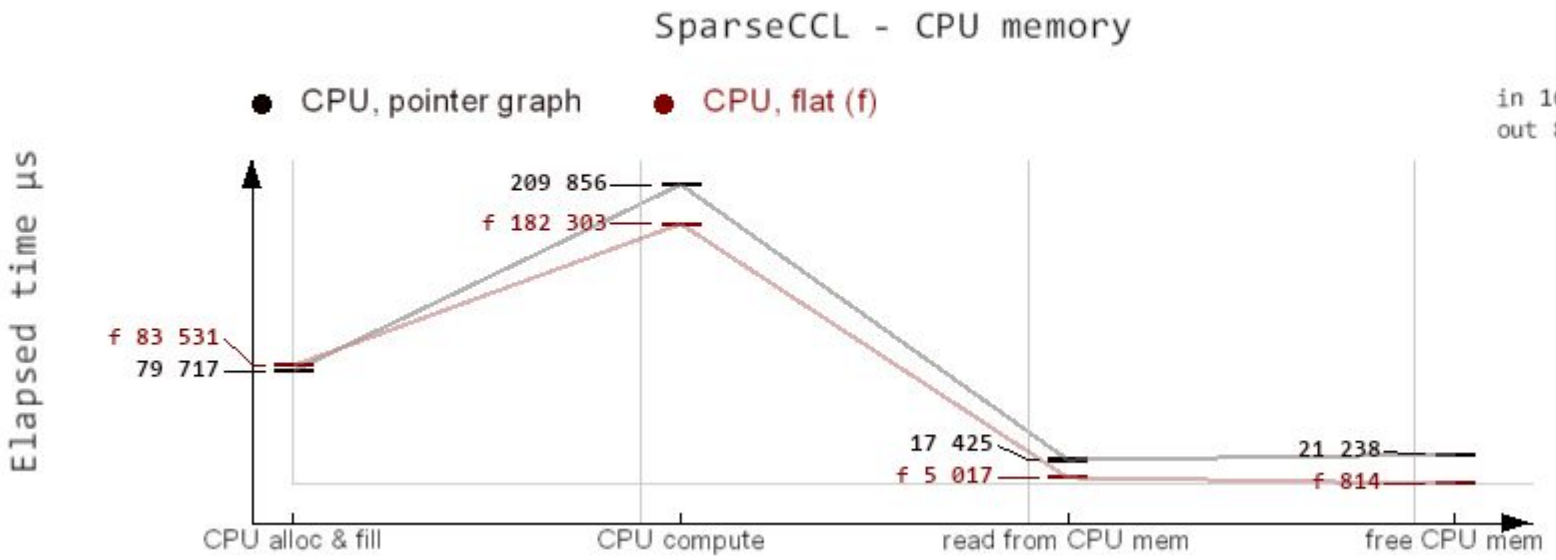
SparseCCL, CPU only

Grouping timing of allocation & fill : to avoid lazy allocation bias

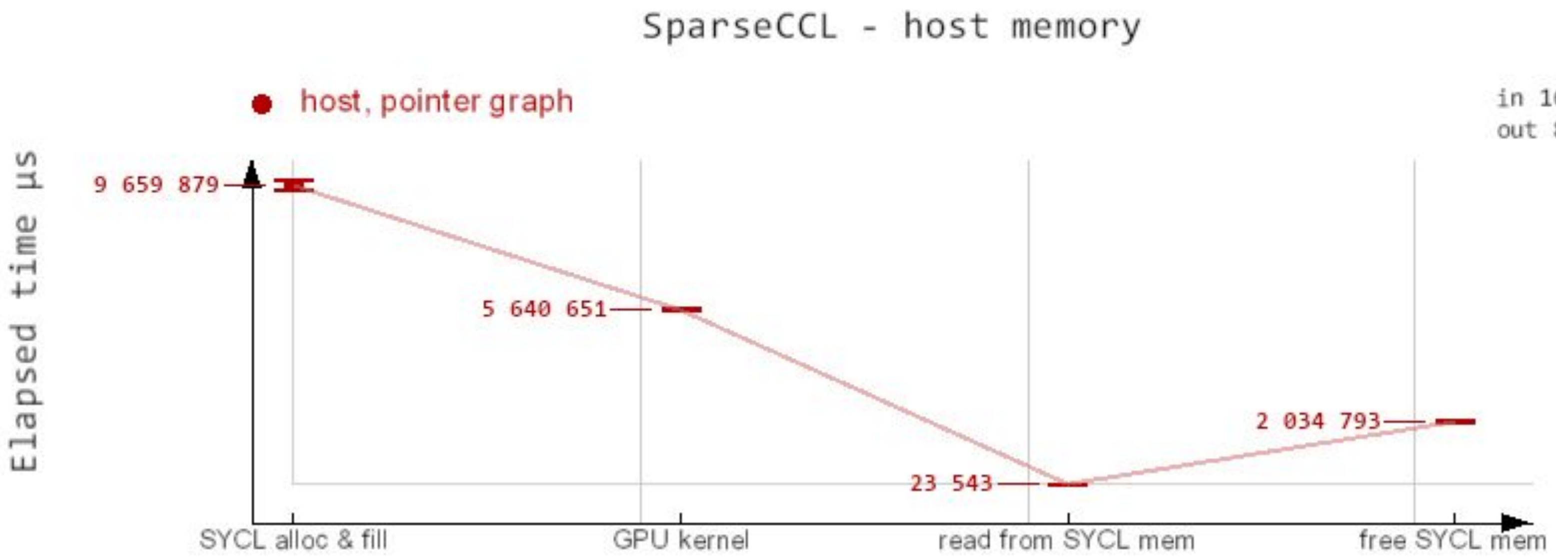


SparseCCL, CPU only : pointers vs flat

Similar between arrays and pointer graph. [10% on compute]



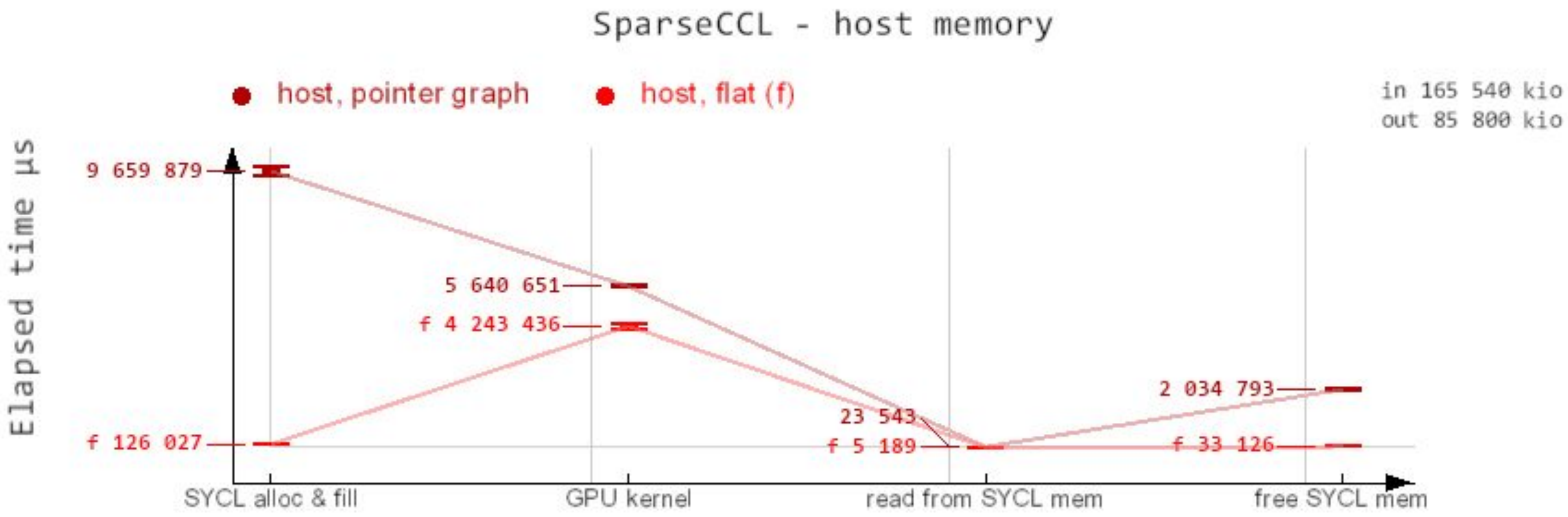
SparseCCL on GPU - USM host



SparseCCL on GPU - USM host

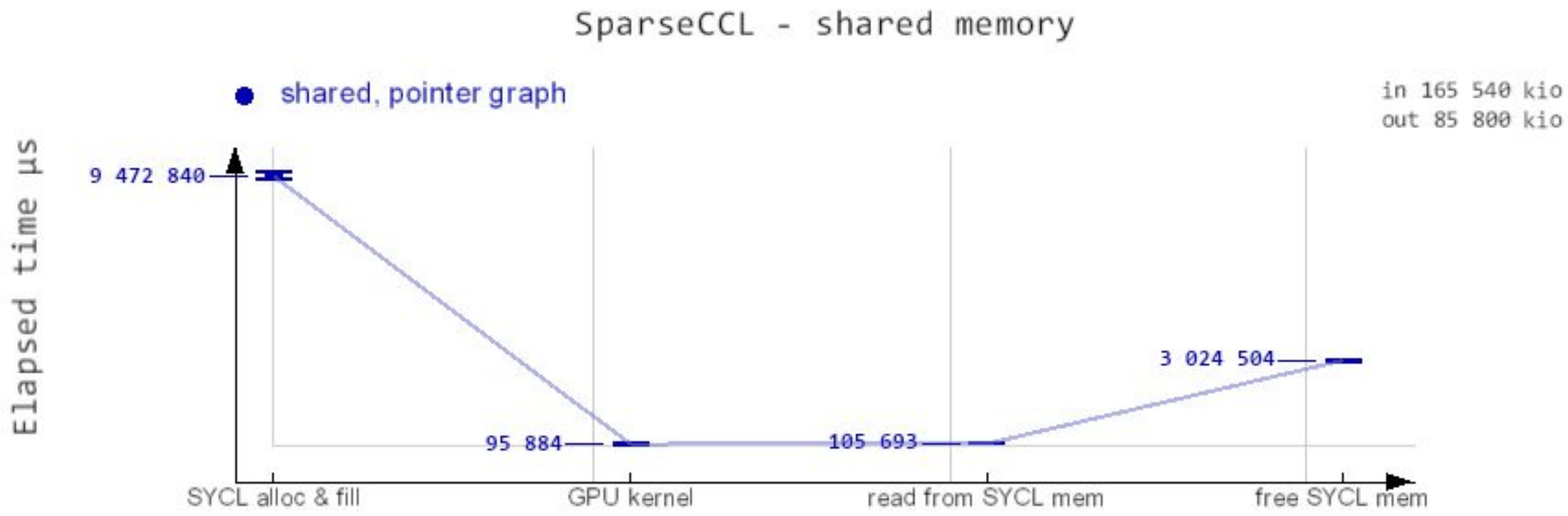
Allocation x80 ! Free x60 !

Global time x4



SparseCCL on GPU - USM shared

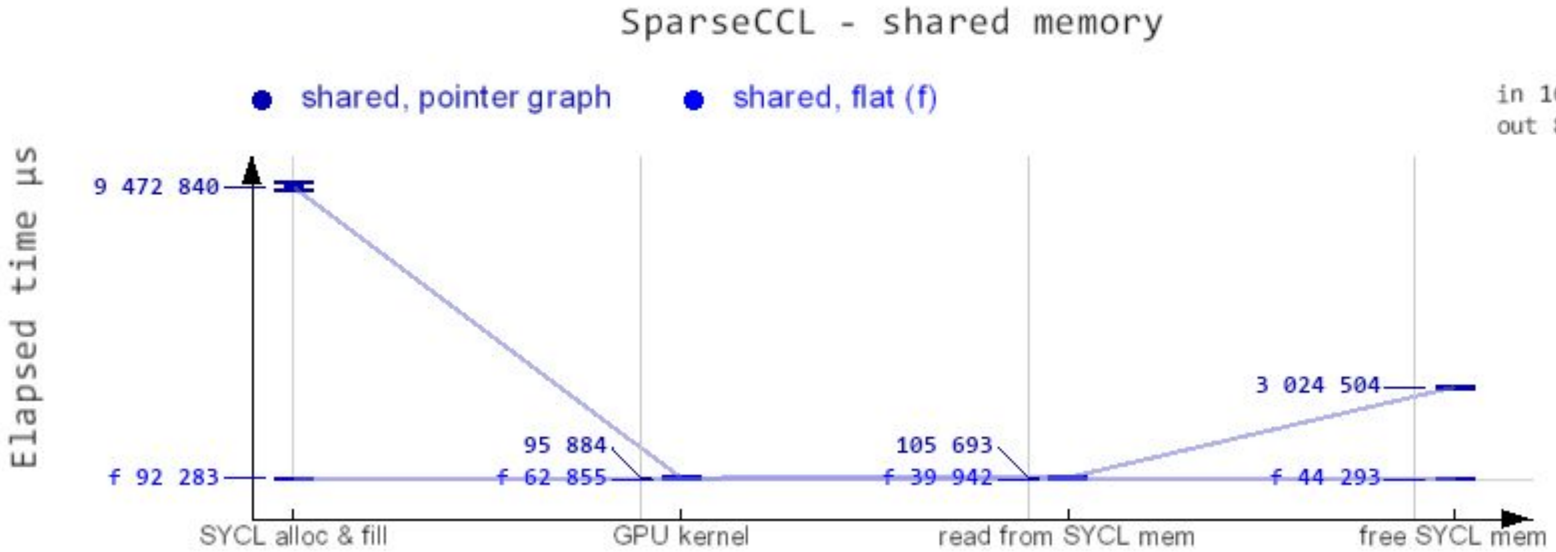
[same scale than before]



SparseCCL on GPU - USM shared

Faster kernel => much increased cost

Global time x50 !



Findings : pointer graph

Easy to replace malloc with malloc_[host/shared] but expensive

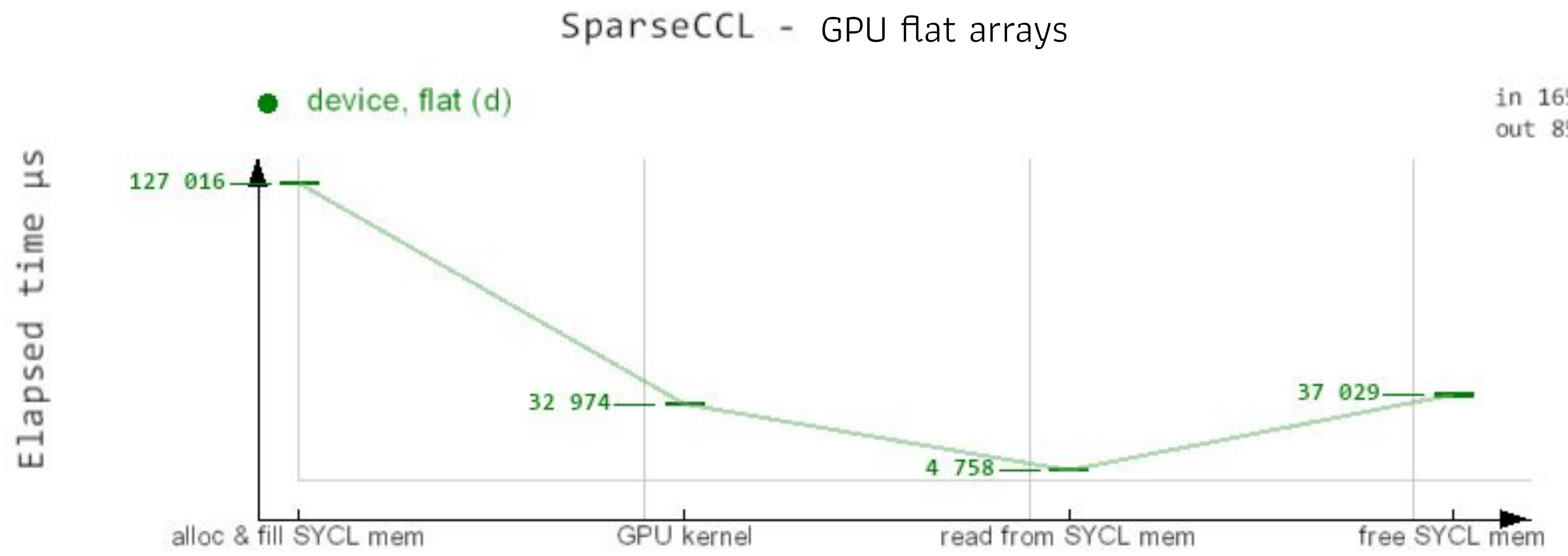
Needs structure adjustments to be efficient on GPU

USM device needs flat arrays

Now, what about flat arrays ?

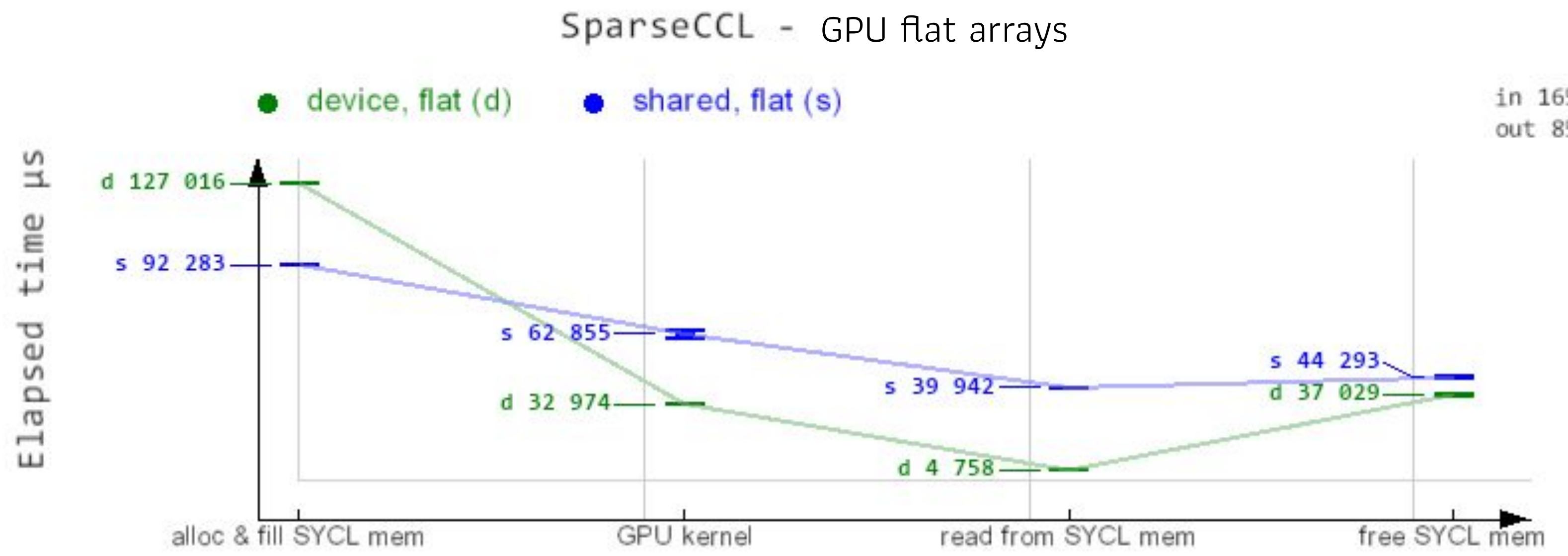
SparseCCL on GPU - flat arrays

USM device



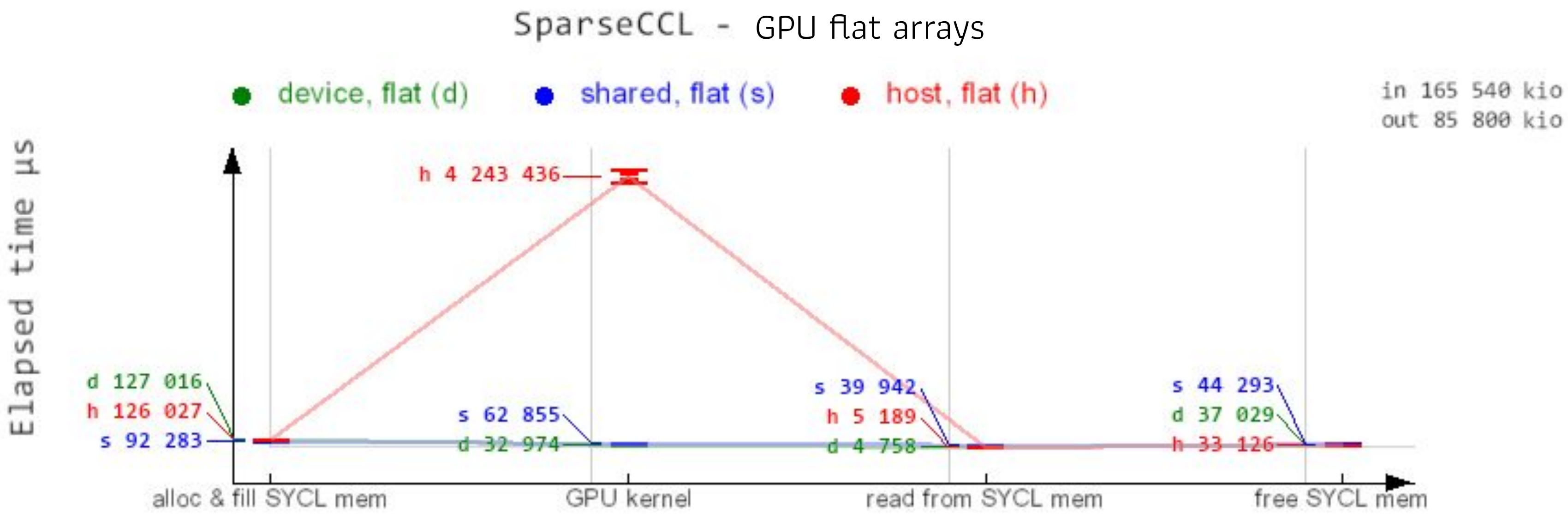
SparseCCL on GPU - flat arrays

USM device vs shared



SparseCCL on GPU - flat arrays

Multiple access to the same data => USM host kernel **very** expensive



Findings : flat arrays

USM host only appropriate for single data access

USM shared is an option

USM device still faster

General conclusions

Host & shared : Easy to reuse existing data structures but
most likely expensive.

USM host can *most likely* be used as a usual CPU-only memory.

USM device most efficient but needs flat data.

Limitations :

- *Only tried on SparseCCL and a microbenchmark.*
- *Only ran on a server and other smaller NVidia devices.*

Future work

Prefetch USM shared

Buffers and accessors

Evaluate USMs models for seeding

More devices : NVidia, AMD, Intel...

A deeper understanding of SYCL

Submitted abstract to ACAT'21[1]

[1] ACAT'21 : https://indico.cern.ch/event/855454/

Thank you ! Questions ?

Links

My [WIP] github : https://github.com/SylvainJoube/SYCL_tests

hipSyCL : https://github.com/illuhad/hipSYCL

SYCL : https://www.khronos.org/sycl

Data Parallel C++ : https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/dpc-compiler.html