

13th International Workshop on OpenCL and SYCL

IWOCL 2025



So You Want to Support SYCL: An OpenCL Perspective

Ben Ashbaugh, Intel




April 7-11, 2025 | Heidelberg, Germany | iwocl.org

KHRONOS
GROUP

Background

- SYCL 1.2.1 was developed as a “high level model” for OpenCL
- SYCL 2020 added support for multiple SYCL “backends”
 - Expanded SYCL beyond the OpenCL ecosystem
- OpenCL ❤️ SYCL
- OpenCL remains a popular implementation choice for SYCL





If I have an OpenCL
implementation,
what do I need to
support SYCL?

Good news!

- Not much!
- In theory, at least.
- You can get by with a fairly basic OpenCL 3.0 implementation.
- Remember: SYCL 1.2.1 targeted OpenCL 1.2!

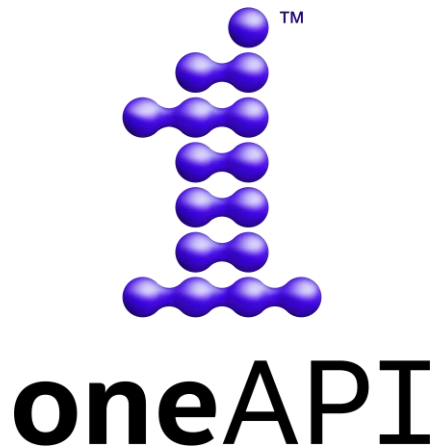


In practice....

- You need a bit more
 - Mostly in the form of optional OpenCL 3.0 features
 - Mostly on the device side (kernels, generated by the SYCL compiler)
 - A little on the host side (APIs, called by the SYCL runtime)

Caveats

- I am most familiar with the oneAPI DPC++ implementation
- Although, I will try to describe AdaptiveCPP also
- There may be bugs... in the presentation or in the SYCL implementations!



DPC++ Compilation Flow:

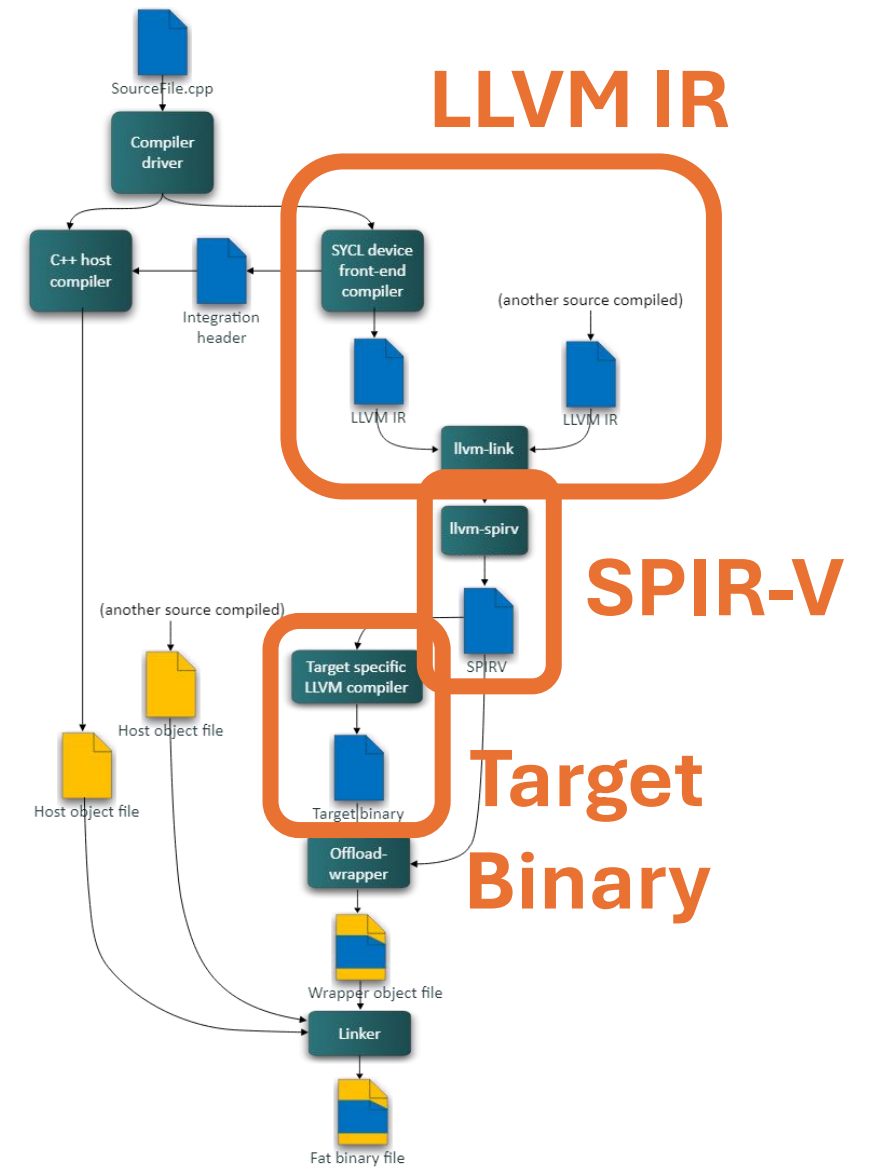
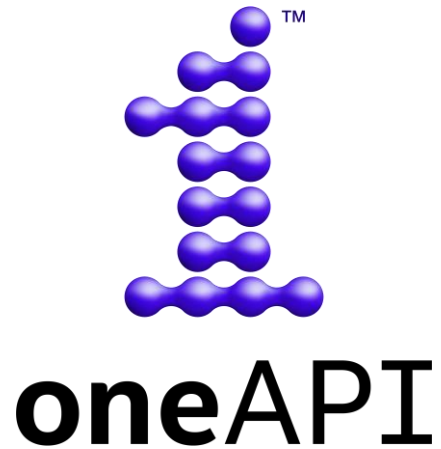
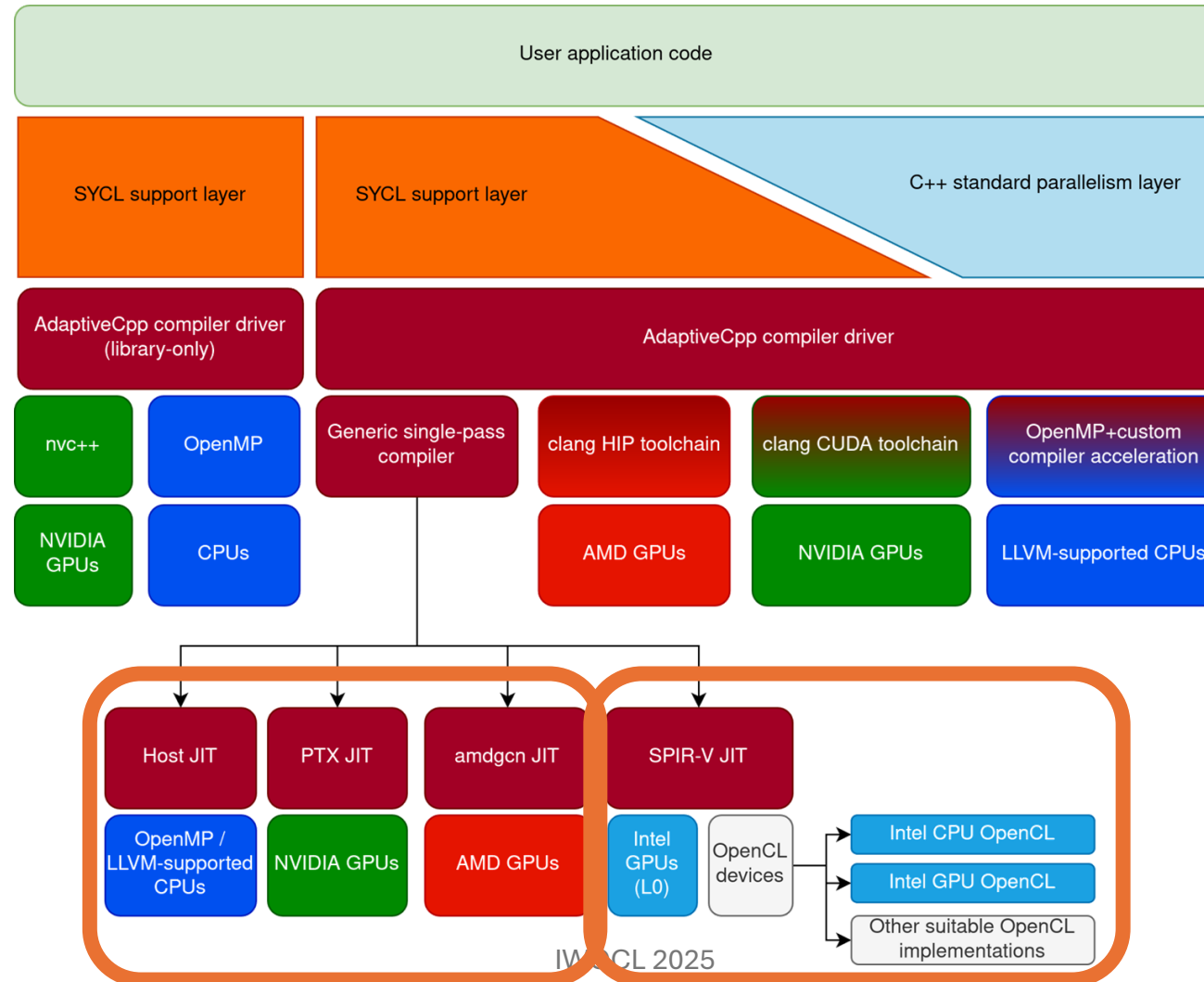


Diagram Source: <https://github.com/intel/llvm/blob/sycl/sycl/doc/design/CompilerAndRuntimeDesign.md>

AdaptiveCpp Compilation Flow:



Compilation Takeaways

- For JIT, you will *probably* want to support SPIR-V (kernel flavor):



- Don't support SPIR-V yet? Lots of open source tools can help!
 - <https://github.com/KhronosGroup/SPIRV-LLVM-Translator/>
- Not a strict requirement, but a strong recommendation

Compilation: Practical Requirements

- Support the Generic Address Space
 - Most SYCL pointers are not qualified with an address space
 - Address space inference is brittle and error-prone
- Support Sub-groups
 - Possible implementation: sub-group size equals work-group size (or one)
- Nice-to-Haves:
 - Work-group Scan and Reduction functions
 - Program Scope Global Variables
 - May need work if unsupported...

Host APIs

- Very little is *required**
- If you are OpenCL 3.0 conformant, then you should be good

* *Required* is doing a lot of work here! Hold that thought...

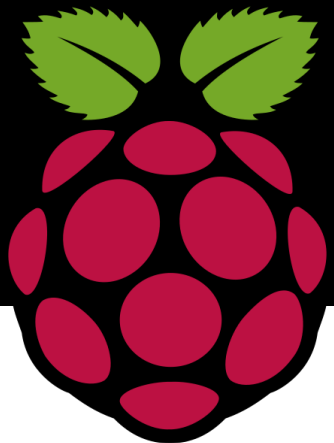
bashbaug@raspberrypi: ~/git/simple-sycl-samples/install/Release

File Edit Tabs Help

```
bashbaug@raspberrypi:~/git/simple-sycl-samples/install/Release $ uname -a
Linux raspberrypi 5.15.84-v8+ #1612 SMP PREEMPT Thu Jan 5 12:02:08 GMT 2023; root@raspb64: GNU/Linux
bashbaug@raspberrypi:~/git/simple-sycl-samples/install/Release $ sycl-ls
[opencl:cpu:0] Portable Computing Language, pthread-cortex-a72 1.2 [3.2-pre master-0-gcee84017]
bashbaug@raspberrypi:~/git/simple-sycl-samples/install/Release $ sycl-loader -c -g -julia
Running on SYCL platform: Portable Computing Language
Running on SYCL device: pthread-cortex-a72
Finished in 0.158837 seconds
Wrote image file julia.bmp
... done!
Total Enqueues: 16

bashbaug@raspberrypi:~/git/simple-sycl-samples/install/Release $ xdg-open julia.bmp
bashbaug@raspberrypi:~/git/simple-sycl-samples/install/Release $
```

julia.bmp (512x512) 100%



SYCL on Raspberry Pi via PoCL OpenCL 1.2 (CPU), Early 2023

BLOG

Using oneAPI Construction Kit to Enable Open Standards Programming for the Metis AIPU

Related products: [AI Software](#)

6 months ago • 0 replies • 9 views •



Manuel



Manuel Mohr | Staff Software Engineer at AXELERA AI

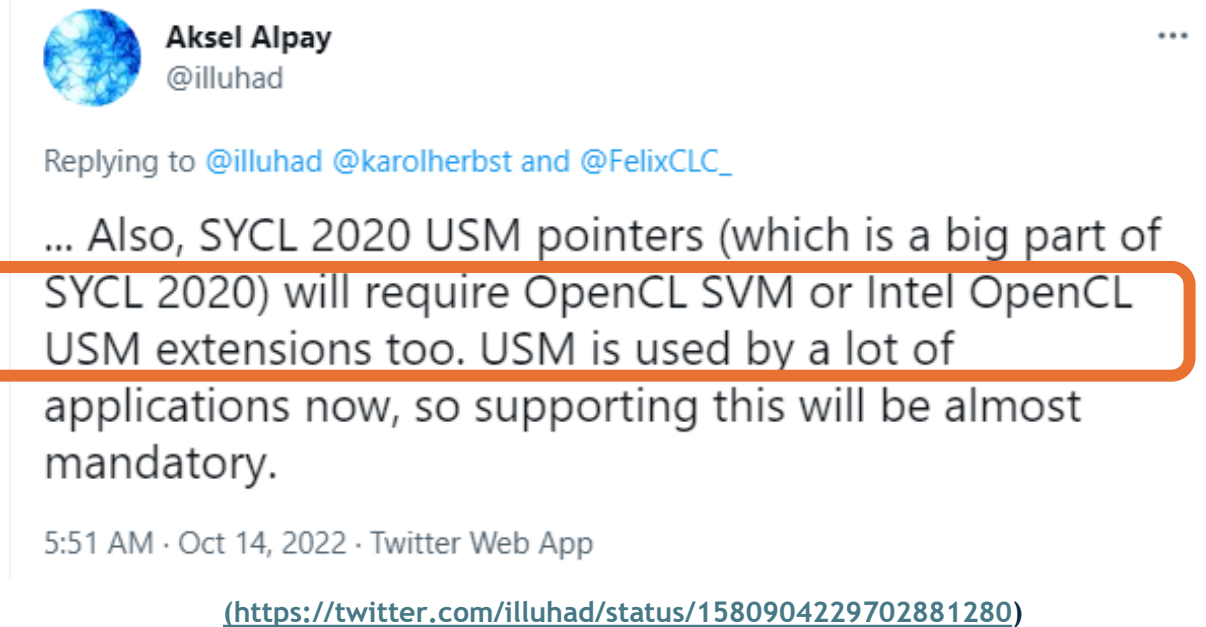


<https://community.axelera.ai/product-updates/using-oneapi-construction-kit-to-enable-open-standards-programming-for-the-metis-aipu-128>



Unified Shared Memory

SYCL and Pointers









cl_khr_unified_svm



Extension Goals:

- Support SYCL 2020 USM
- API Consistency with SVM
- Compatibility with Intel Extension
- Extensibility for the Future
- All Features Tested ([Test Plan](#))
- Many Implementations!

add cl_khr_unified_svm extension #1282

 bashbaug wants to merge 5 commits into [KhronosGroup:main](#) from [bashbaug:cl_khr_unified_svm](#) 

 Conversation 14  Commits 5  Checks 1  Files changed 3

 bashbaug commented on Nov 8, 2024 Member 

This PR contains the draft specification for `cl_khr_unified_svm`. It is intended to provide early access for review and community feedback, and it is not a ratified specification.

`cl_khr_unified_svm` adds additional types of Shared Virtual Memory (SVM) to OpenCL. Compared to Coarse-Grained and Fine-Grained SVM in OpenCL 2.0 and newer, the additional types of SVM added by this extension provides:

- Sufficient functionality to implement "Unified Shared Memory" (USM) in other APIs, such as SYCL.
- Additional control over the ownership and accessibility of SVM allocations, to more precisely choose between application performance and programmer convenience.
- A simpler programming model, by automatically migrating more SVM allocations between devices and the host, or by accessing more SVM allocations on the host without needing to map or unmap the allocation.

Specifically, this extension provides:

- Extensible interfaces to support many types of SVM, including the SVM types defined in core OpenCL, in this extension, and additional SVM types defined by other combinations of SVM capabilities.
- Explicit control over memory placement and migration by supporting device-owned SVM allocations for best performance, host-owned SVM allocations for wide visibility, and shared SVM allocations that may migrate between devices and the host.
- The ability to query detailed SVM capabilities for each SVM allocation type supported by a platform and device.
- Additional properties to control how memory is allocated and freed, including properties to associate an SVM allocation with both a device and a context.
- A mechanism to indicate that a kernel may access SVM allocations indirectly, without passing a set of indirectly accessed SVM allocations to the kernel, improving usability and reducing driver overhead for kernels that access many SVM allocations.
- A new query function to query properties of an SVM allocation.
- A new function to suggest an SVM allocation type for a set of SVM capabilities.

Because the interfaces defined by this specification are not final and are subject to change they are not intended to be used by shipping software products. If you are interested in using this feature in your software product, please let us know!

<https://github.com/KhronosGroup/OpenCL-Docs/pull/1282>

How cl_khr_unified_svm Works:

- Platform Query: Returns an array of SVM capabilities supported by devices in the platform

```
clGetPlatformInfo(  
    platform,  
    CL_PLATFORM_SVM_TYPE_CAPABILITIES_KHR,  
    numSVMTypes * sizeof(cl_svm_capabilities_khr),  
    allSVMTypes,  
    nullptr);
```

Index	Capabilities
0	Coarse Grain SVM
1	Fine Grain SVM
2	Device USM
3	Host USM
4	<some completely new SVM capabilities combination>
5	System SVM

How cl_khr_unified_svm Works:

- Device Query: Returns a parallel array of SVM capabilities supported by the specific device
 - May add to platform capabilities
 - Or be zero, if unsupported

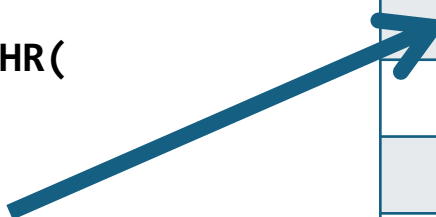
```
clGetDeviceInfo(  
    device,  
    CL_DEVICE_SVM_TYPE_CAPABILITIES_KHR,  
    numSVMTypes * sizeof(cl_svm_capabilities_khr),  
    allSVMTypes,  
    nullptr);
```

Index	Capabilities
0	Coarse Grain SVM
1	Fine Grain SVM (with atomics!)
2	Device USM
3	Host USM
4	<some completely new SVM capabilities combination>
5	0 (System SVM not supported)

How cl_khr_unified_svm Works:

- Allocation: Pass an SVM Type Index
 - Guaranteed to have the capabilities described in the platform array
 - May have additional capabilities on some devices

```
void* ptr = clSVMAllocWithPropertiesKHR(  
    context,  
    nullptr,    /* properties */  
    0,          /* SVM type index */  
    size,  
    &errorCode );
```



Index	Capabilities
0	Coarse Grain SVM
1	Fine Grain SVM (with atomics!)
2	Device USM
3	Host USM
4	<some completely new SVM capabilities combination>
5	0 (System SVM not supported)

How `cl_khr_unified_svm` Works:

- Additional Functionality:
 - SVM Type Helper: suggest an SVM type based on required capabilities
 - SVM Introspection: query properties of a pointer
 - SVM Free Properties: extensibility for the future
 - Improved SVM Usability: simplified indirect access

For everything else: Use existing SVM APIs!

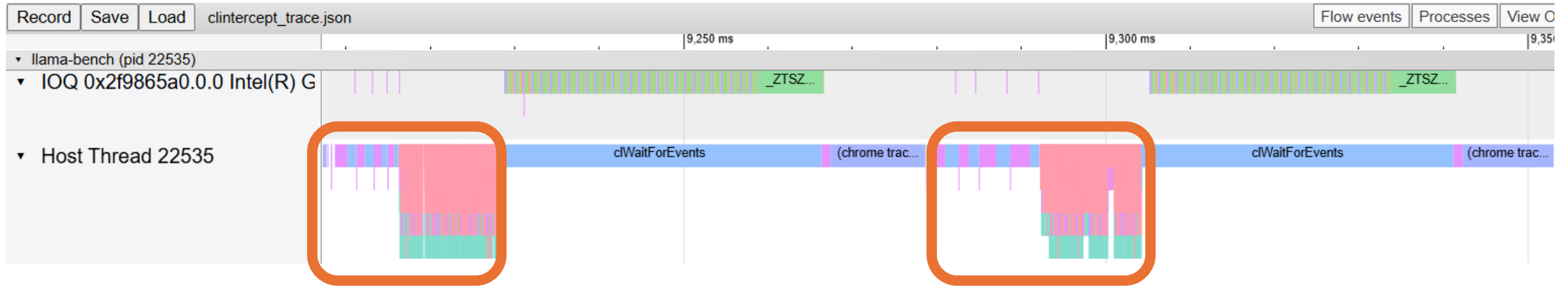
cl_khr_unified_svm Links for Reference:

- PR with the extension specification, best place to provide feedback:
<https://github.com/KhronosGroup/OpenCL-Docs/pull/1282>
- Draft PR with header changes:
<https://github.com/KhronosGroup/OpenCL-Headers/pull/269>
- Branch with extension loader changes:
https://github.com/bashbaug/opencl-extension-loader/compare/cl_khr_unified_svm
- Branch with emulation layer and some basic test apps:
https://github.com/bashbaug/SimpleOpenCLSamples/compare/cl_khr_unified_svm
 - (note, needs the updated headers and extension loader to build!)
- PR with CTS test plan:
<https://github.com/KhronosGroup/OpenCL-CTS/pull/2150>
- Development branch with CTS tests:
https://github.com/KhronosGroup/OpenCL-CTS/tree/cl_khr_unified_svm

Graphs and Command Buffers

https://github.com/intel/llvm/blob/sycl/sycl/doc/extensions/experimental/sycl_ext_oneapi_graph.asciidoc

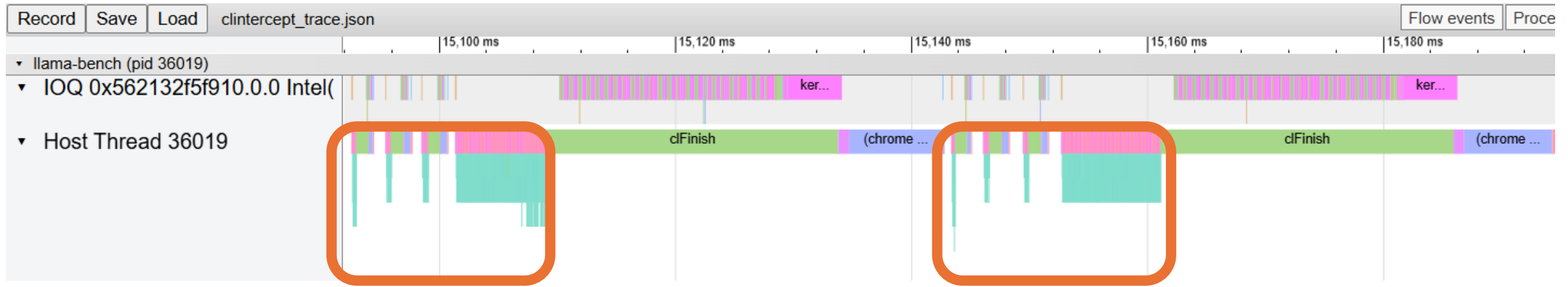
Problem Statement: llama.cpp (SYCL)



~15K OpenCL API Calls

... and, repeat...

Problem Statement: llama.cpp (OpenCL)



~11K OpenCL API Calls

... and, repeat...

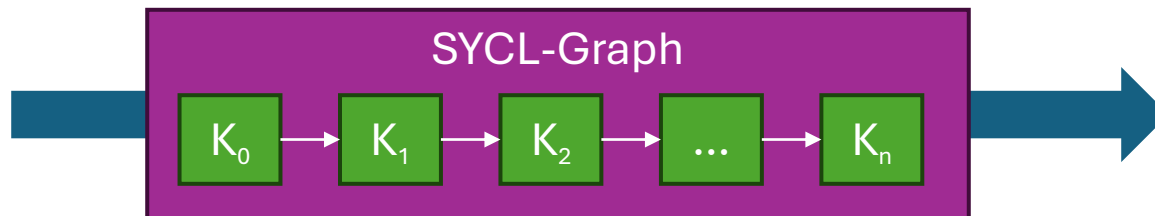
- Too Many Calls!
 - Idle Device, Too Much Power, Potential Host Bottleneck

SYCL-Graph

- Replace Individual Kernel Submissions



- With one SYCL-Graph Submission

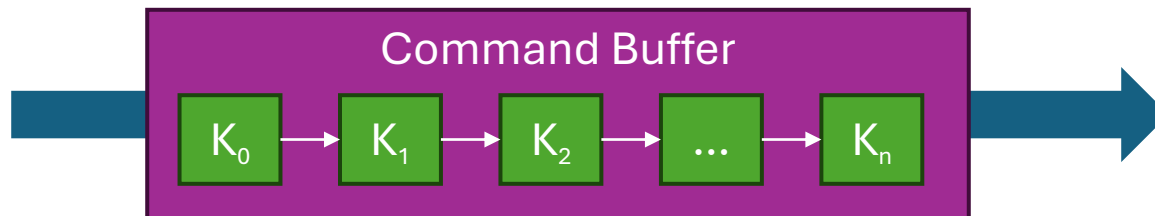


SYCL-Graph → OpenCL Command Buffer

- Replace Individual Kernel Enqueues



- With one Command Buffer Enqueue



Command Buffer Extension Status

- Two Command Buffer Extensions are *Provisionally* Released:
 - Note: Provisional extensions are subject to change!
 - [cl_khr_command_buffer](#): Base-level extension, supports creating and executing static command buffers
 - [cl_khr_command_buffer_mutable_dispatch](#): Layered extension, supports modifying command buffer dispatches (e.g. kernel args)

Command Buffer Extension Status

- Other Command Buffer Features in (Public) Development:
 - [cl_khr_command_buffer_mutable_memory_commands](#): Layered extension, supports modifying command buffer memory operands
 - Host access? Additional dispatch mutability (kernels)? Nested enqueue?
- CTS coverage is good
 - For both command buffers and mutable dispatch
- Implementations are starting to appear

Please provide feedback!
(For SYCL-Graph or Command Buffers!)

Summary and Wrap Up

Summary



- You don't need to implement SYCL to support SYCL!
 - All you need is OpenCL 3.0 with a few optional features.
 - Many open source resources exist to help out or get started!
- Two extensions are needed for many use-cases in practice:
 - Unified SVM: to support SYCL 2020 USM
 - Command Buffers: to support SYCL-Graph
 - Both extensions are still in development – please provide feedback!
- Thank you!

Backup

What about SVM?

- Application support for SVM is low, despite support from many devices:
 - **No SVM support:** Layered implementations (CLon12, clvk), (some mobile GPUs?)
 - **Coarse Grain SVM:** NVIDIA GPUs, many Intel GPUs, (some mobile GPUs, Mesa/rusticl?)
 - **Fine Grain SVM:** AMD GPUs
 - **Fine Grain SVM + Atomics:** Some Intel GPUs, Qualcomm GPUs
 - **System SVM:** CPUs

(Snapshot from April 9, 2025,
https://opencl.gpuinfo.org/displaydeviceinfo.php?name=CL_DEVICE_SVM_CAPABILITIES)

