Rusticl: Compute for the Linux desktop?

Karol Herbst, Mesa, Red Hat
Who am I?

• Mesa contributor since 2016
• Works on Nouveau and OpenCL
• Part of Red Hat’s GPU team since 2018
• Added support for system SVM to mesa’s old CL driver
• Started rusticl in 2021
• Participates in the OpenCL WG
Who am I?

- Mesa contributor since 2016
- Works on Nouveau and OpenCL
- Part of Red Hat’s GPU team since 2018
- Added support for system SVM to mesa’s old CL driver
- Started rusticl in 2021
- Participates in the OpenCL WG
- Turns into a spec lawyer
The situation
The Linux desktop

• Compute hasn’t seen much adaption
• Lack of good runtimes/APIs?
• A lot of packaging work?
• What alternatives exist?
Applications

- CLBlast
- Darktable
- Davinci Resolve
- GIMP
- Libreoffice Calc
- OpenCV
- Couple of smaller ones
Applications

- CLBlast
- Darktable
- Davinci Resolve
- GIMP
- Libreoffice Calc
- OpenCV
- Couple of smaller ones
- There aren’t that many actually
What lead do this?

- Bad reputation of clover, AMD only
- Fragmented Eco system
- People think OpenCL is dead
- No out-of-the-box availability
Alternatives
Requirements

- It needs to run everywhere
- People still use GPUs from 15 years ago on modern desktops!
  - Got a recent report of Firefox crashing on GeForce 6000 GPUs :’)
- Users just want to use it
- App developers don’t want to become compute experts
- Want to use their favorite programming language
• Doesn’t see new features or concepts
• However it’s easier to use than Vulkan
• Limited compute capabilities
• GLSL
• SPIR-V support probably broken
- It has a lot of momentum
• It has a lot of momentum
• Vulkan evolves into a low-level API for drivers
• Hard to target for simple offloading
• We need something higher level
Vulkan

- It has a lot of momentum
- Vulkan evolves into a low-level API for drivers
- Hard to target for simple offloading
- We need something higher level
- Rusticl can run on top of Vulkan
• Has quite the momentum
• Actively developed
• Has quite the momentum
• Actively developed
• It’s not a runtime API
⇒ Runtime lock-in at compile time
• AdaptiveCPP might make SyCL more interesting?
• Alternatives generally vendor or language locked-in
• Desktop features don’t want to rely on it to function
  • What if an accessibility feature relies on e.g. CUDA?
  • What if you have to be a C++ developer to use it?
OpenCL

- Quite simple to use
- Inherited some of the time’s mindset
- Everything is abstracted
- Experiences a revival
- SPIR-V support
- So OpenCL it is
• Quite simple to use
• Inherited some of the time’s mindset
• Everything is abstracted
• Experiences a revival
• SPIR-V support
• So OpenCL it is (for now)
But does it work OOTB?

- No
But does it work OOTB?

- No
- Unless your distribution packages OpenCL runtimes
- Rusticl could fix this!
The solution?
• Rust based OpenCL 3.0 implementation in Mesa using Gallium
• Multi-threaded
• Mesa: FOSS cross-vendor GPU driver
• Gallium: driver abstraction
• Rust based OpenCL 3.0 implementation in Mesa using Gallium
• Multi-threaded
• Mesa: FOSS cross-vendor GPU driver
• Gallium: driver abstraction
• ... and my first Rust project
Why Rust?

- I wanted to learn about rust
- Is it feasible inside mesa?
- Is it feasible for an OpenCL impl?
- Removes entire classes of bugs
- Easy to write thread-safe code
- Arc, Mutex and closures help a lot!
Why Rust?

- I wanted to learn about rust
  - Is it feasible inside mesa?
  - Is it feasible for an OpenCL impl?

- Removes entire classes of bugs
- Easy to write thread-safe code
  - *Arc, Mutex* and closures help a lot!
Rusticl compiler stack

- Uses the LLVM-SPIRV-Translator for OpenCL C
- Optimized for SPIR-V input
  - causes issues, because everybody else uses LLVM
- NIR as mesa internal IR
  - optimization and lowering passes
- gets translated to Vk SPIR-V for Zink
But does it work OOTB?

- No
But does it work OOTB?

- No
- Unless you opt-in
The Journey so far
Making Mesa compute ready

- Mesa’s compiler is not LLVM based
  - Adding a structurizer (also used for NVK)
  - Supporting pointers
  - Supporting function calls (ongoing)

- Advanced compute features
  - SVM (also ongoing)
Use inside Mesa

• Shader lowering
  • Ray tracing
  • GS/TCS/TES for GPUs not supporting it
  • Internal shaders for Intel
• Shared code between host and GPU
• Memcpy
What about supporting other APIs?

• Implementing optional features for:
  • SyCL: DPCPP or AdaptiveCPP
  • CUDA/HIP: chipStar

• Level Zero: no concrete plans
So where are we going with this?
Things OpenCL might want to improve

- Performance pitfalls
  - $HOST_{PTR}$
  - API validation can be expensive
  - Almost no feedback on optimal usage
  - Multi device support feels too implicit

- SVM/USM/BDA

- Add explicit client VM management? Or sparse?
Ecosystem

- Early SPIR-V validation in
  - SPIRV-LLVM-Translator
  - DPCPP
  - LLVM SPIR-V backend
- Easier packaging
- Better debugging tools
  - need more tools like the opencl-intercept-layer
Where I want to see more Compute

• Video encoding/decoding?
  • AV1 is cool and all, but what about old hardware?
• Accessibility?
  • Disclaimer: I'm not an accessibility expert
Where I want to see more Compute

• Video encoding/decoding?
  • AV1 is cool and all, but what about old hardware?
• Accessibility?
  • Disclaimer: I’m not an accessibility expert
  • Speech-to-text
  • Automatic picture descriptions
  • Summaries of long texts
Where I want to see more Compute

• Video encoding/decoding?
  • AV1 is cool and all, but what about old hardware?
• Accessibility?
  • Disclaimer: I’m not an accessibility expert
  • Speech-to-text
  • Automatic picture descriptions
  • Summaries of long texts
• More High level APIs
  • Apple does this and are very successful with this?
More focus on SPIR-V
OpenCL C should not be the only way to write kernels!
Make others use it (e.g. HIP or openpm)
SPIR-V should become the de-facto compute IR
Other Considerations

- Laptops often have slow GPUs
  - requiring beefy GPUs for AI/ML is not great
- Unified memory often slow! (Apple solved this)
Questions?