Introductions

Simon McIntosh-Smith:

I am going to move on to the live panel session in just a moment. I'm going to put a slide up so that you can see who's going to be on the panel. Neil and Michael are going to stay around so they can be on the panel too, but there's a whole bunch of other people. I'm going to ask each one of them to introduce themselves briefly so that you know who's speaking.

Also, I am going to remind each panellist that you will need to unmute to introduce yourselves and then you can re-mute afterwards. So if we start with Alastair, please, introduce yourself.

Alastair Murray:

Hi, there. My name is Alastair Murray. I work at Codeplay where I lead a team of people implementing OpenCL and including recently, we've been prototyping OpenCL 3.0. And within the OpenCL working group, I do some of the work on the specifications.

Simon McIntosh-Smith:

Thank you, Alastair. Ben, could you go next?

Ben Ashbaugh:

Sure. I'm Ben Ashbaugh. I am a software architect at Intel. Within Khronos, I edit some of the OpenCL specifications. I also participate in the SPIR working group and I feed some of the features into SYCL and oneAPI.

Simon McIntosh-Smith:

Thank you, Ben. Dennis, over to you.

Dennis Adams:

Hi. Thank you. I am Dennis Adams, Director of Technology at Sony Creative Software. We've been using OpenCL for about 10 years. First, to retrofit GPU image processing into a large video editing application and then to build a new cross-platform video application that exclusively uses OpenCL for all of its image processing.

Simon McIntosh-Smith:

Thank you, Dennis. Eric, you're up next.

Eric Berdahl:

Hi, I'm Eric Berdahl. I'm a senior engineering manager at Adobe's Digital Video and Audio business unit. We're the part of the company that makes products like Premiere Pro, Premiere Rush and After Effects and many others, products that you would use to create video. I manage the group at Adobe that builds the rendering pipelines and the GPU runtimes that make all of Adobe's video products run. We have been using OpenCL like Sony for more than a decade as one of the prime GPU runtimes that let our software run on multiple platforms. I also represent Adobe to Khronos and I'm personally actively involved in several of the working groups, notably, the OpenCL working group and the Vulkan working group.

Simon McIntosh-Smith:

Awesome. Thank you, Eric. Hal, you're next.

Hal Finkel:

Hi, I'm Hal Finkel from Argonne National Laboratory. I lead our compiler technology and programming languages team there. I'm the vice-chair of the C++ standards committee and I'm also involved in the SYCL working group. I am excited to be here, and we are excited to be exploring SYCL and also Intel's DPC++ for our upcoming supercomputing systems.

Simon McIntosh-Smith:

Thank you, Hal. Jeremy, can you introduce yourself?

Jeremy Kemp:

Hi there. My name is Jeremy Kemp, senior software engineer at Imagination Technologies where I'm responsible for our implementation of OpenCL. And my capacity within Khronos is to represent Imagination within the OpenCL working group.

Simon McIntosh-Smith:

Great. Thank you. Kevin, you're next.

Kevin Petit:

Hello. I'm Kevin Petit. I am a software architect at Arm and amongst other things, I work on the OpenCL implementation that is used on many GPUs.

Simon McIntosh-Smith:

Thank you, Kevin. Martin, over to you. Are you there, Martin? I can see you're unmuted, but I can't hear you. We might be having some audio issues with Martin. We'll come back to you if we can, Martin. And Martin is the local chair for IWOCL 2020 in Munich. So thanks for being here, Martin. Last but not least, Ronan, can you introduce yourself, please?

Ronan Keryell:

Hi, I'm Ronan Keryell. I work at Xilinx Research Labs. I'm mainly on the higher-level programming models for FPGAs and CGRAs based on modern C++ and I have worked on SYCL since 2014, mainly on triSYCL and also inside Khronos as the editor of the SYCL specification.

Panel Discussion - Part 1

Simon McIntosh-Smith:

Thank you very much. And thank you to all our panellists for that. If I can remind all the panellists, if you make sure you're muted and then when questions come up, I'll steer them to you or if you'd like to join in, then unmute yourself at that point. But thank you all for joining us. So this is the point where we'd like the audience to start asking questions. It can be about anything to do with OpenCL or SYCL or anything related to those things.

I can see there are questions starting to come through. I will kick off with one which I think is a fairly broad, one that anyone might be able to answer. Maybe we'll start with an OpenCL answer and then move on to a SYCL one. But with all the announcements that have just been made by Neil and by Michael with OpenCL 3.0 and with the next generation SYCL stuff, what are the most important changes that are coming in those new versions of the standards?

So, we're going to start for OpenCL 3. First of all, what are the most important changes for OpenCL 3. Somebody on the panel can take that one.

Eric Berdahl:

Well, I'll lead off since nobody else wants to jump in. I'll say there are three things that really jump out at me about OpenCL 3 that I'm excited about. One of them is … actually the point is that OpenCL 3 is not that big a step codewise. We can update very easily from OpenCL 2 to OpenCL 3.

Neil made a point of this in the presentation. So there's not a lot of work you need to do to roll up to you start using OpenCL 3 implementations and then start incrementally adopting features. One of the things that I really like about the spec itself is also the fact that it's really opening up for the possibility of layered implementations.

That trend that Neil talked about in the presentation about layers becoming more prevalent in the industry is one that we are watching very carefully at Adobe and that I have been subtly influencing for several years. I think it's a very important direction for the future of GPU runtimes and GPU programming models.

And the third thing is in some ways a minor thing, but as a user, it's one of the most exciting pieces is that the OpenCL 3 specification is now a universal specification. So there isn't just an OpenCL 3 spec, it's an OpenCL spec and I can go and find information about every version of OpenCL ever shipped in that spec makes my ability to find information about OpenCL significantly easier. And I've very much enjoyed using that specification.

Simon McIntosh-Smith:

Great. Thank you, Eric. Did anyone else want to chip in on what are the biggest changes for OpenCL 3? Go ahead, Alastair, then maybe we will come to Dennis.

Alastair Murray:

So speaking as someone who, until recently, had only implemented OpenCL 1.2 because of the fact that hardware we target can't support some 2.X features, it's pretty good to be able to … if we had the features that we can support without having to try and emulate the features we can't.

Simon McIntosh-Smith:

Thank you. And, Dennis, did you want to add something to this question?

Dennis Adams:

Yeah. I would actually agree very much with Eric on his second point, the layered implementations. We, like them, have a large base of existing OpenCL C and are looking for essentially deployment flexibility on run times other than the OpenCL runtime. So the layered implementation approach opens up a lot of avenues for getting onto Vulkan runtimes or on top of Metal runtimes or other things like that.

And the new initiative by Microsoft on top of DX are all very exciting. OpenCL C is a very expressive language for high-performance image processing, allowing us to optimize using shared local memory and workgroups and things like that. And we would hate to have to rewrite those into some flavour of the day compute language every time there is a new platform out there. So the layered approach and being able to retain our investment in over 300 kernels is very important to us.

Simon McIntosh-Smith:

Great.

Ben Ashbaugh:

I was going to mention the new version of OpenCL C also. I mean, that is one thing that we haven't revved since OpenCL 2.0, I believe. So we've got a much better and complete kernel programming language now in OpenCL 3.

Simon McIntosh-Smith:

Thank you, Ben. How about the more SYCL-oriented folks on the panel? Anyone from the sequel side? What are the biggest changes coming in in what Michael was just talking about?

Michael Wong:

I can start off and then I'm going to ask Ronan to jump in on some of these ones.

Simon McIntosh-Smith:

Okay.

Michael Wong:

So I think certainly, I mentioned generalizations, but what I didn't mention is why they help us. And you're going to see that in some of the talks about that online. So with generalization, which actually ended up being a bit of a bigger change, but it gives us essentially independence from the single backend. But we're still deeply tied to OpenCL, but it can now also allow us to adopt a Vulkan, Cuda, OpenMP backends or whatever you can put your imagination to.

Michael Wong:

The other big one that I urge people to look at, things that unify shared memory. This enabled codes would point us in data structures to work naturally without translation or buffers or accessors. And it helps us to port from lower-level languages or from Cuda, for instance. I think that's a big, big sell. That's a big improvement on these things. The other one is modules. I talked about it and not those C++ modules. Unfortunately, that name is heavily overloaded.

Michael Wong:

What SYCL modules are, what they do is they essentially embed multiple objects into archives even for different backends. And so, this massively reduces ambiguity in the spec and remove. So I'm going to ask Ronan to jump in so that I don't end up talking about all. Maybe Ronan can talk a little bit about reductions, which was a top request from IWOCL DHPC++ in 2019 or specialization constants and other things. Ronan?

Ronan Keryell:

Yes. So in HPC, having some efficient reductions to take advantage of the real hardware features is very important because if you're using accelerators, it's to reach the maximum performance. So that's something which is coming. For specialization constants, the idea is actually similar to something pushed by Hal Finkel in ISO C++ about JITting some C++ templates.

Often, for example, in machine learning, or things like that, you have a huge amount of machine learning kernels. But often the difference is just in a few coefficients. So, if you can fix these coefficients in some way so that they are compiled or JITted just before run time, you can achieve a more efficient execution and you can basically lower your carbon footprint by these optimizations. So I think that's pretty exciting to see the gain in efficiency we can get from that, for example.

Michael Wong:

And if I might also ask Hal Finkel who gives us the user's point of view as to what he thinks is useful, things like … there's improvements to auto-space, in order queues, removing standard layer restrictions and interop. Hal, do you want to say anything then from your user point of view?

Hal Finkel:

I mean, I'll just quickly add that there are a large number of features like that that we've now been working on for a while and have been inspired by a lot of our experience working on other programming models such as Kokkos and RAJA and the standard of C++ itself. And I think that it's really going to improve the user experience.

Simon McIntosh-Smith:

Great. Thank you. Thanks to all of you. That's great. I'm going to move on to one of the questions that's been coming live. It was quite long, but I think I'd summarize it as, is there a timeline for the OpenCL 3.0 to Vulkan interop? Could somebody answer that, OpenCL 3.0 to Vulkan interop, how is that going? Is there any timeline for that?

Ben Ashbaugh:

Broadly, we've been working on this for a while now. It is rounding up into shape. I can't give a target date, but certainly, I would hope that this will come out in 2020 and hopefully, sooner in 2020 rather than later.

Simon McIntosh-Smith:

Brilliant. Thank you. Was someone else going to add to that?

Neil Trevett:

I was going to say, Ben, did you want to just briefly mention the way we're doing it? It's layered over a generic memory sharing interface.

Ben Ashbaugh:

Yeah. The high-level way of working with it a lot like the external memory in Vulkan itself. So the good news here is that this won't be specific to Vulkan. We should be able to extend this to support other APIs as well. We've heard requests for interop with DX 12 and that should be able to use a similar mechanism. We're solving Vulkan first, but we do think that this is a general mechanism that can work for other APIs too.

Simon McIntosh-Smith:

Brilliant. Thank you. One of the next questions we've had from the live audience is, is there any widely available mobile device support, OpenCL or SYCL in the pipeline that the panel's aware of? So anything in the mobile space, OpenCL and SYCL, that anyone can comment on?

Jeremy Kemp:

Yeah. So, Imagination will be shipping OpenCL 3.0 implementation. We expect to have that once the spec has been finalized and once it's ready, but from that point on, we'll be shipping on mobile GPUs.

Simon McIntosh-Smith:

Thank you, Jeremy. You're fading out to be a little bit quiet some of the times. Some of the time, it's actually fine, but I think we got most of the gist. Thank you. Any other comments on the mobile space for OpenCL and SYCL?

Neil Trevett

Qualcomm, but also the mobile space supports OpenCL quite extensively. Quite a few of the mobile chip vendors do actually ship OpenCL drivers. It's the low level foundation for the stack, particularly for applications like video and imaging and increasingly inferencing acceleration. Google has not put the OpenCL into the native platform definition for Android, but this is where the layering kicks in again - Google has been instrumental in the clspv project, which layers OpenCL over Vulkan… and Google now are bringing OpenCL apps and libraries over into Android using OpenCL through this layered implementation. And now, we have Kevin from Arm here. I mean, Kevin, do you want to comment on that?

Kevin Petit:

I can say, for OpenCL 3.0, that Arm is committed to implementing OpenCL 3.0. And as you said, there's a bit of a disconnect between what's vendor support and what is widely accessible to developers. Most mobile vendors implement OpenCL and ship OpenCL, but it is not an official API on Android, which makes it hard to access sometimes.

Simon McIntosh-Smith:

Thank you.

Eric Berdahl:

And to actually follow on to what Kevin was saying as an example of that. In one of our Adobe apps, we are actually using OpenCL C kernels in our Android application despite the fact that there's no OpenCL runtime. We use the SPIR-V compiler and a custom Vulkan hosted code that we use, but all of our shaders are still the same OpenCL kernels that we run on desktop.

Simon McIntosh-Smith:

Brilliant. Thank you. I just had a little message from Tim Lewis reminding me that two of the submissions this year that are actually in the program you're going to find on the website were from Qualcomm as well. So even though they're not on the panel, you can go and see some of what they're up to there. So thank you for that.

Audience Poll: #1 - OpenCL over Metal

Now, before we go onto one of the next questions we've had, I'm going to launch a poll.

And this is not something we have tried with this size of audience, so this is slightly experimental. So everyone in the audience, we're going to launch a poll for you. We've got two questions we're going to ask you. We're going to do one of them now. And the first one is, given that we just been talking about layering, it's related to that. So let's see how this works.

You should all see a poll popup, or at least those of you using the desktop app or the web app, you should hopefully see a poll and it is asking about layering. Would you be interested in a layered OpenCL over Metal to guarantee OpenCL will always be available on Apple platforms? So you get to pick yes or no. So just click one of those and click submit.

And I will leave that open just for a moment and once everyone has had a chance to do that, I'll close the poll and then you'll get to see the results. It should be quite interesting. Obviously, Apple was one of the main backers of OpenCL in the early days and in more recent times, they've been focusing on Metal. This would be quite interesting to see. Well, there's lots of people voting. Thank you.

So just to make sure you are paying attention, but the answers are actually really interesting to us and it might affect what's going on. I think nearly two-thirds of people on the meeting have voted already, which is really good. Cool. So I think I'm going to close the poll and hopefully, you get to see the results. Here we go. I'm going to close this poll and it should change to being share results. Here we go.

Hopefully, you get to see what I just saw, which is quite interesting. Here we go. You can all see that. So about two thirds of everyone voted. Would you be interested in a layered OpenCL over Metal to guarantee OpenCL would always be available on Apple? 70% said yes, 30% said no. So that's pretty interesting. And anyone on the panel, are you surprised by that? Is that going to change anything we're doing? Is that really interesting?

Neil Trevett:

That is really interesting. Thank you, everyone, for that feedback. It seems like an obvious thing going forward and obviously, Apple have deprecated. OpenCL is still available right now, but the layering is I think going to be a vital piece to ensure that OpenCL remains available where everyone wants to run their applications.

And, Apple is a significant platform. So this is encouraging and hopefully, we will be able to get this going as a project pretty soon and then encourage people to get involved.

Panel Discussion - Part 2

Simon McIntosh-Smith:

Great. Thank you, Neil. Awesome. Well, we've got more questions coming in. The next one hopefully is going to be quite a quick one and it's … does OpenCL 3 support unified shared memory? OpenCL 3 and unified shared memory. Somebody comment on that.

Ben Ashbaugh:

We have an Intel extension for unified shared memory that we're proving out first before we would add this to the core specification or even as a standard extension. But if this is something that you are interested in, definitely let us know at Intel and also at the OpenCL working group. And I think this is a great candidate for a future extension in 2020.

Simon McIntosh-Smith:

Great, thank you. Don't forget to keep sending in more questions for the panel. We're filtering through them now and asking people what's going on. Maybe while we've got Ben on the line, one of the other questions that I'm being asked quite a lot is what's the relationship between SYCL and oneAPI, which Intel has been driving? There's been a lot more interest in SYCL recently, but could you just comment on how SYCL and oneAPI are related?

Ben Ashbaugh:

Yeah, it's a great question. I'll give a bit of a brief answer and we can go into more detail later. So oneAPI is a broad initiative. It covers things like tools and libraries. It also covers what we're calling direct programming, which is where you would write code and kernels that would run on accelerators. So the component that we call direct programming is data-parallel C++.

It is a specification and an implementation that includes C++ and SYCL, and it also includes extensions like the USM extension. So I think the short answer is that the real question is about the relationship between data-parallel C++ and SYCL, and SYCL is an important component of data-parallel C++.

Michael Wong:

Can I ask a question even though I am on the panel?

Ben Ashbaugh:

Go ahead.

Michael Wong:

I think I know the answer, Ben, but maybe not everyone on the group is aware. But is oneAPI strictly Intel or is it open to anyone? So this probably will help other people.

Ben Ashbaugh:

Yeah, great clarification. It is absolutely not strictly for Intel. It's an industry initiative. Anybody is welcome to join and implement parts of oneAPI. We actually have support for other devices in our data parallel C++ compiler today. So, you can use data parallel C++ to compile for Nvidia GPUs and you can find all about this on the oneAPI website, oneAPI.com.

Simon McIntosh-Smith:

Yeah, thanks, Ben. There's actually been a follow up question from your previous answer, Ben, which is, is the Intel unified shared memory just for CPUs and GPUs or will it also apply to the FPGA stack as well?

Ben Ashbaugh:

So I can't really comment on product plans, but it's certainly intended to be used for other devices also. And there's nothing that I know of that would prevent it from being supported on other devices also.

Michael Wong:

I will try to remember this question for tomorrow's SYCL live panel and have hopefully, one of the experts to prepare a good answer.

Simon McIntosh-Smith:

Right. Good. Another question that's come in is quite good. Is there something specific you're doing to minimize performance degradation between using SYCL and directly using the backend? So this is probably a question more for the SYCL guys and commenting on performance overheads and things like that.

Michael Wong:

Ronan, do you want to … can you say the question again? I missed the first part of it?

Simon McIntosh-Smith:

Yeah, it is basically what's happening in terms of trying to minimize performance overheads using SYCL compared to using OpenCL or something like that directly.

Ronan Keryell:

Yes. So from a theoretical point of view, I can see any real difference and actually, with modern C++, you can express even more details if you want, because of our new extensions and a lot of things like constexpr in C++. And so I think theoretically, you should be able to write more efficient code and more adaptable code because it's easier to write in modern C++ some code that we really take into account various data types that you cannot do, for example, in plain OpenCL right now.

Michael Wong:

So I'll quickly add my perspective on that. We're not here to … we don't want to get into any flame war between C and C++ performance. But I will say that SYCL is being modelled after C++ and C++ has traditionally been shown to have reduced a lot of the overheads that people have been concerned about.

So without any concrete data, nobody can really say for sure, but we have seen that in some cases with C++, code with the improvement or on Compiler optimizations over the years by many compiler groups, they have been able to reduce overheads wherever it's unnecessary so that it's comparable to inline C code.

Simon McIntosh-Smith:

Okay. Thank you. Is there anyone else on the panel who'd like to comment on performance of SYCL compared to other lower level things or the backends before we move on? I know we have … It's something we've been doing quite a lot in the Bristol Research Group and Tom Deakin's been doing a lot of this work recently.

Simon McIntosh-Smith:

And it's actually pretty close to OpenCL and other lower level things. So I'm pleasantly surprised and it's something that keeps improving all the time. But generally, with all these things, the more people use things, the better they get. So I'll just encourage everyone to give these things a try.

Hal Finkel:

I'll just add quickly that one of the things that we working on at Argonne as far as our compiler work goes, is taking advantage of single source program models in order to specifically enable optimizations inside of kernels based on data that's outside the kernel. And that's something that you can't do in a model like sickle and can't do in a model like OpenCL where the kernels are compiled separately.

Hal Finkel:

So there actually are cases where you can get even better performance with a single source program model than you can with separate source program models.

Simon McIntosh-Smith:

Great. Thank you, Hal. Alastair, did I see you unmute? Did you want to add anything, Alastair?

Alastair Murray:

Yeah. So, at Codeplay, we implement both OpenCL and SYCL and much like other panellists have already said, it's very rare that SYCL provides any disadvantages. In fact, often, it's faster because you can write better code. The only real case where we never end up clearly winning with OpenCL is if you're able to directly export the [inaudible 00:28:41] feature that is not yet enabled in SYCL.

But really, that's just a matter of time and it's really just a function and you have to implement it in the lower level before you can do it in a higher level. So really, it is about exposing the hardware features in terms of advantages.

Ronan Keryell:

Yeah. I would precise that what is also very important with the new backend interface coming in SYCL 2020 is that you have a very strong interoperability layer with the underlying implementations. That means that if you're not happy with your SYCL implementation, you can still very easily use, for example, OpenCL kernel from inside SYCL programs, so you don't lose a lot here.

Simon McIntosh-Smith:

Thank you.

Michael Wong:

It's great that OpenCL and SYCL have these different characteristics. I think they play to the strength of their specific communities in that way.

Hal Finkel:

Yeah. And I think one area where we have seen OpenCL performance superior to that from other program models are specific cases where you're able to take advantage of just-in-time compilation from OpenCL C code. So you can dynamically compose code that does just what you want as opposed to pre-compiling a more generic version.

And I think that over time, this is something that we'll see improve in other program models as well, whether it's specialization or other capabilities that will provide some of the same kinds of functionality. And I think that will be important in part because it'll provide that functionality in a way which is cleaner and more usable and maintainable than just constructing source code as strings inside your program.

Simon McIntosh-Smith:

Thank you, Hal. There was an ultimate example of that a couple of years ago with James Price, who is one of the former members of our group in Bristol and now at Google. It was basically generating kernels, statement by statement on the fly to putting together the optimal configuration for a certain platform. So, you can take this quite a long way.

Very good. This is a really good follow-on question about this discussion, which is, would it be possible to merge SYCL features to future C++ standards? And I guess we can open that up a bit further too. Where do we see SYCL and C++ and oneAPI and Kokkos? Where are they all going? Are they going to merge? Is one informing the other? What's happening there?

Michael Wong:

I've watched this space very closely for three, four, five years now. And for all the authors of those frameworks you mentioned, Kokkos, RAJA, SYCL and I imagine, oneAPI have all stated that their wish is to push their features into C++, ISO C++. And that has already happened. Things like MD Span and Span came from Kokkos. We're pushing upwards into C++ affinity, and we've tried pushing other things in because it's not going to be exactly the way it looks in these frameworks where they originated.

And now, I would say … and I've given talks in the past pointing out that there's this quiet cooperation happening within C++ where a number of people who are interested in heterogeneous C++ are collaborating to add this into ISO C++ slowly. So I think for a long time, frameworks like SYCL, Kokkos, RAJA will lead C++ and will have to because things will settle in the C++. It's an ISO standard. So it moves like a battleship.

Even as fast as C++ moves, it still needs to bring along 200 people, 500 companies, 20 countries along with it. So I think there will always be a space for forward-looking frameworks that are trying to take advantage of the latest hardware, the latest programming innovations and models. And by the same token, I think a lot of these frameworks is also downloading features from ISO C++.

For instance, we definitely intend to download features like futures and executors and core routines and modules and C++ ranges into SYCL future. Other people? I think Hal is also another person that has good answers on this, and Ronan.

Hal Finkel:

So I'll just quickly add two things. So first to follow up on Michael's coming to a quiet cooperation, I'll also note that a lot of the same people are involved in all of these spaces. So we have good participation from a lot of people who are on the SYCL working group within standard C++ itself. The same thing is true of developers of Kokkos and RAJA and other similar frameworks.

So, there's really a way in which there's a lot of learning going on between the different implementations, but also a lot of direct cross-pollination because it's the same people who are involved in these different areas. I think one of the interesting questions going forward from the standard C++ side of things has to do with the way that we evolve the memory model and how we adopt features that are proven out by SYCL and Kokkos and other frameworks for dealing with different memory spaces and execution and data locality concerns into standard C++.

And right now, we are learning how to best do this in the context of these other frameworks and those learnings are then going to be moved forward into the C++ standard.

Simon McIntosh-Smith:

Very good. Thank you. If there's nothing more to add to that one, there's one follow-up question on SYCL, which is, what toolchains are required to run SYCL applications on Xilinx FPGAs today? And are any of those tools freely available? That might be a question for Ronan.

Ronan Keryell:

Well, for Xilinx FPGA triSYCL is a research project, so I would say it's very hard for normal people to use it. I cannot say anything about the roadmap for SYCL support at Xilinx. But if you want to try, I can give you some support, but it will be difficult today. That's what I can say.

Simon McIntosh-Smith:

Okay. So it sounds like that's early days at the moment. But you are saying people can get in touch with you if they are interested in SYCL on Xilinx FPGA.

Ronan Keryell:

Yes.

Audience Poll: #2 - SPIR-V Benefits

Simon McIntosh-Smith:

Brilliant. Thank you. Okay, good. Now, before we move on to the next question, we've got another poll. This is, again, making sure you're all still awake and also listening. I'm going to launch another poll. This one's a little bit different. It's the last one and it's about SPIR and SPIR-V. So I'm going to launch the poll now. Hopefully, you'll see this. It should be coming up at any moment. I've got a few more options for you this time.

So, if SPIR-V were widely available, it would improve developer and user experience because … click all of these things that are relevant to you. So which of these things might help you in terms of to help secure IP inside your kernels? Because you don't have to ship the kernels to these strings anymore. They let you use non-OpenCL C language kernels, offline kernel compilation, leveraging tooling around spear V.

There are lots of more tools starting to support SPIR-V or none of the above. So it'll be really useful, really interesting to get you to comment on this. If you're using SPIR-V, you're interested in SPIR-V. Lots of people have voted. Nearly, half of you have all voted now, which is great. Thank you.

Simon McIntosh-Smith:

Lots of people have voted. I'm going to close the poll and then I'm going to share the results. I'm going to share the results with you all so you can see what you said. And again, I think it makes it pretty interesting reading. Here we go. So what was the most popular? Offline kernel compilation for speed or portability?

What is the most popular? Leverage tooling around SPIR-V. That's really interesting. Confidence in using non-OpenCL C languages is important. And then not so much interest in securing IP and side kernels. I thought that might've been one of the more important ones. That's quite interesting.

Michael Wong:

I'm shocked that that's near the bottom of the list.

Neil Trevett

Probably as it is not truly encryption, more obfuscation, I think. It helps, but it's not a bulletproof solution. So maybe that's okay. But overall interesting results.

Simon McIntosh-Smith:

Yeah. So thank you very much everyone for voting on that. I think we had maybe 60% voted, which is an interesting size of sample. Very good. Thank you. Right. Let's hide that again and then we'll move on. Don't forget, you can keep asking questions. If you've got more questions, keep posting them. We've had some really good ones. Thank you so much.

Panel Discussion - Part 3

So other questions we've had come in beforehand was, will we ever be able to use OpenCL or SYCL with other languages such as Fortran or Python or OpenMP? That's what we get asked a lot, so if you can comment on that.

Michael Wong:

I would just say we would if we have more of those experts in the Khronos community because you need the local experts there to be able to make a good interface there. I'll stop there and allow other people to jump in. I will say that there's nothing that prevents us, I think, from working with OpenMP. This has been asked around a lot in terms of language framework interoperability.

Obviously, when you have two concurrent … two parallel programming languages going together, you're going to run into things like scheduling issues, overloading your scheduling space and your split space, but … And there's nothing that seems to be able to create a topmost overload governor on that. So that's just a standard thing that you would have when you're interfacing any multiple parallel programming languages together.

Ben Ashbaugh:

In all seriousness, I think there are a couple of Python bindings for OpenCL at least, so give it a try if you haven't.

Simon McIntosh-Smith:

Yeah, I think, Hal, you wanted to say something.

Hal Finkel:

Yeah. I was just going to add that we are dealing with these interoperability issues on a regular basis. And my expectation is that as we explore solutions in this space, as we figure out the things that seem to work best, we'll definitely bring those to the relevant standards committee.

Michael Wong:

That's true. You guys are probably the one that will … where this collision will happen in very serious workloads right away since you still use OpenMP and OpenCL and now, SYCL.

Hal Finkel:

That's exactly right. But part of the issue here is that in many of these cases, it's not obvious that there's a right answer for what the interoperability semantic should be. And so a lot of this ends up being somewhat empirical based on looking at the applications and what they want to do in the various use cases in trying to fashion some set of solutions that practically work. And so again, I think as we get that experience, we'll be able to bring that back to the standards body.

Simon McIntosh-Smith:

Thank you. We're having a piece of follow-on question and in particular, about Fortran. So has anyone looked at Fortran with OpenCL or SYCL? And in particular, one person raised the issue of Fortran front end streaming into LLVM, whether that makes this easier, more tractable, et cetera. So Fortran support for SYCL and OpenCL, is that possible? Is it getting easier? Is it something that-

Hal Finkel:

Yeah, I'll speak to this a little bit. We have experience mixing Fortran with a number of different other programming models. Now, sometimes, this is done in this trivial sense, right, where you have some kernels that you're going to write in some other language and then you're going to call those from Fortran. And practically speaking, we have a number of users who do precisely that.

With the implementation of Fortran's front end for LVM, one of the things that that does allow us to do is to use that front end in order to directly generate LVM and perhaps SPIR-V code from Fortran. Now, there are a lot of questions you have to answer about what it means when you call Fortran built-in functions and the like.

But in the context of OpenMP, for example, this is an active area. We intend to support OpenMP offload in Fortran. And many of the same challenges and questions that come up there will come up in adaptations of other programming models for Fortran.

Michael Wong:

So for us, I think if there is such thing as what Hal alluded to, a Fortran front end for LLVM and someone were energetic and interested in doing it, it is entirely possible for us to pair it with SYCL in some future way. And we would welcome that work.

Michael Wong:

And I understand where this question is coming from because in a high-performance computing domain, with the research, a lot of kernels are still purely written in Fortran. Okay? And they might be accessed from the outside by C or C++ or something like that. And people are just not going to spend all the time tuning the kernels from Fortran to some other language.

Ben Ashbaugh:

Yeah. I was going to ask for that clarification too. Is this more about writing Fortran kernels, so the code that runs on the device or now, Fortran API bindings for the things that run on the host code?

Simon McIntosh-Smith:

I think it's a wide range of things. I think the question largely comes from Fortran and isn't going away. So I certainly want to go with, I think, on the UK national machines, 70% of all core hours are running codes written in Fortran. So how do we get those codes running on GPUs, things like that?

And the moment you have to rewrite the code in something else first, C or C++ and then you put it to kernels, it might be much easier to get more code through if you didn't have to go through that extra step. And it could be Fortran in SYCL or Fortran in OpenCL or anything like that.

Michael Wong:

Yeah. Actually, rewriting it a generally introduces of possible large number of bugs which they don't want to live through because this Fortran has been debugged over 20 years.

Neil Trevett:

Yeah. And a general comment from the OpenCL side and following on what Ben was saying earlier, I mean, we have the community resource page and it always amazes me going by how much there is there. I've just gone and searched on the community resource page and there's about a dozen hits on Fortran there for various tools doing various things with Fortran.

So, yeah. There's actually quite a lot of depth in the OpenCL ecosystem. So going to check out the community resource page. And if you know stuff that is not there, you can do a pull request and add your own favourite tools to let the community know. So I encourage people to do that.

Simon McIntosh-Smith:

Thank you, Neil. And that's on the Khronos web pages. Yeah?

Neil Trevett:

That's right.

Dennis Adams:

Just in a really general sense, speaking to bridges and interop and things like that, you mentioned these Fortran routines were written 20 years ago and fully debugged. And nobody wants to have to put an intern on it for a summer to rewrite it in a different language because it works just fine in the language it's in, which speaks towards the investment even in OpenCL C kernels between, for example, Adobe and us.

Dennis Adams:

They still apply to the image processing that we want to do on new platforms. So having a really clean bridge to new runtimes like Vulkan and things like that and even bridging into SYCL and other … you can certainly write new things in the new expressive language, but you do need a way to bring along the old things, whether that's OpenCL C or Fortran or what.

Simon McIntosh-Smith:

Thank you. Good point. We've got a question that's coming. It's marked as being Nvidia-specific. Well, I think we'll start it that way, but it does open up to everyone else as well. So the question was really about, with the recent announcements for OpenCL 3 and for SYCL, is this going to make a difference with how much Nvidia is going to be able to support in terms of how much of OpenCL it can officially support?

Is that going to go up? And then also, a specific question about SPIR-V. So maybe this is for Neil initially, but then maybe we'll wipe this out to everyone else as well. So will the changes mean that Nvidia can officially support more of OpenCL, for example, and more of SPIR?

Neil Trevett:

Yeah, I think it will definitely help. NVIDIA has publicly stated that we are going to implement OpenCL 3.0. And I think the fact that we can now begin to have much more flexibility in shipping the functionality that is relevant to our customers will help us implement more than we are currently able to do for our customers going forward.

Personally, I would love Nvidia to support SPIR-V in OpenCL. I loved the results of that poll. I'm going to take that back to Nvidia and see if we can support it. Of course, we already support SPIR-V for Vulkan. So now, the gap is potentially not too big, but as always, it comes down to what our customers are asking for.

But the amount of goodness in the SPIR-V ecosystem that we have been talking about now is beginning to get, I think, pretty persuasive. So, yeah, hopefully … I can't commit to it, but hopefully, yes, I can see things going in the right direction.

Simon McIntosh-Smith:

Thanks, Neil. Maybe I'll open that to a few others maybe. It's nice to have Arm here. We don't always have Arm in some of these panels. So if you asked Arm, is this going to make it easier for Arm to support more of OpenCL and maybe also answer about SYCL. Is there going to be any official support for SYCL on Arm in the future?

Kevin Petit:

So Arm already supports OpenCL 2.1 on GPU, so including SPIR-V. We started shipping that, I think it was last year.

Simon McIntosh-Smith:

Cool.

Kevin Petit:

On SYCL, it's an interesting technology that we're watching. We don't have anything to say publicly about whether we will bring that support to SYCL at this stage directly. But I should add that via SPIR-V, it is possible to take a third party SYCL implementation and run on ARM GPUs.

Simon McIntosh-Smith:

Very good. Is there anyone else on the panel that would like to address this.

Michael Wong:

Yeah. support SPIR-V or something like that, right? Yeah. It is theoretically possible. Yeah.

Simon McIntosh-Smith:

Good. Anyone else? How about Imagination? How about Jeremy? Any comments on SYCL or more of OpenCL.

Michael Wong:

So Simon, Jeremy did type some answers in on the chat.

Simon McIntosh-Smith:

Oh, thank you. Good. Short answer. Yes. Very good. 1.2 devices will support an overwhelming majority of features. I guess that's something that's going to [inaudible 00:52:05] with the new approach in OpenCL 3, so that's really good. And 2.0 devices retain all features. Awesome. Thank you, Jeremy. Good on text in the end. Yeah. So one of the questions that's been asked, it's a bit more future-looking.

We are in our last 10 minutes now, so we've almost run out of time. But it is: what do the panellists think are the biggest missing features still for both OpenCL and SYCL? So anyone can have a go at answering that. Well, what's still to come, is I guess that question is about. Who'd like to have a go at answering that? Dennis?

Dennis Adams:

I'll jump in real quick. I think we've talked about the real positive experiences around SPIR-V and what it brings to the table. And so having SPIR-V either required or at least very widely supported across all of the platforms that a given company wants to use would be very beneficial.

Simon McIntosh-Smith:

Great. Thank you. Eric?

Eric Berdahl:

Yeah, I completely echo that. The phrase that I like to use is that SPIR-V should be the common shading language. And then I would like to see a renaissance of shader coding languages, all generating SPIR-V. And once the runtimes start all consuming SPIR-V, you don't have to care what I write in. And that's good for me. That lets me be more expressive.

The other thing I like to see is continue the direction of having additional layers on top of the dominant GPU runtimes. The things that the GPU runtimes give us are ubiquity and stability and quality, but the things that the layers give us is programming expressiveness and workflow, things that are very difficult to do down at the very low levels.

So that is why I'd like to see things like OpenCL and SYCL continue as high-level programming interfaces interfacing down into lower-level GPU runtimes via this layering strategy and pattern.

Simon McIntosh-Smith:

Brilliant. Thank you, Eric. Anyone else? What are the missing things? What kinds of things might come in OpenCL 3 to one or next-gen SYCL? Other thoughts?

Michael Wong:

Right. I think some of the things that we are definitely looking forward to are things like subgroups. We want it to simplify the accessors. Accessors are great and they certainly serve a certain community for implicit … aid the movement very well. But there's a community that don't care about that that much, especially if you're runtime and you want more explicit control.

We also have things like a better way, a more formalized way of handling extensions. Address spaces, I think, is another thing that we want to serve. And we've already talked about a few other things in that direction.

Simon McIntosh-Smith:

Thanks, Michael. Alastair?

Alastair Murray:

For OpenCL, obviously 3.0 is all about flexibility and that's great, but I think [inaudible 00:55:22] will be doing is, or I hope we'll be doing is more in that direction making OpenCL even better for a wider range of devices. The ones I care about are things like DSPs and AI accelerators, but any heterogeneous device.

Simon McIntosh-Smith:

Cool. Thank you. Anyone else that had been unmuted at one point? Did you want to add anything, Ben?

Ben Ashbaugh:

Yeah. I was just going to mention that Neil's slides had a list of extensions or possible future features in OpenCL, so that's a great list to start with. One of the items that wasn't on that list though, that we've heard requested for is some global barrier or a synchronization across workgroups. That's not something that's been on the list before, but we've heard that request, I think even from IWOCL audiences in the past. I think that's a good one to investigate for a future version of OpenCL and SYCL.

Simon McIntosh-Smith:

Yeah. When I teach OpenCL to undergraduates, that's the number one thing that tends to trip them up. And they always look a bit puzzled at me when I explain that situation. They don't understand why it's not possible, but they get it in the end. But that would make their lives a lot easier, I think, if we had something like that. All right, good.

I think we have got maybe time for one last question. And maybe the one that's the most appropriate is what does the panel think are the biggest opportunities in terms of increased adoption for OpenCL and SYCL for the next few years? So where do you see the growth areas for OpenCL and SYCL over the next couple of years? Who'd like to have a go at that?

Eric Berdahl:

Mobile. Clearly, the place where OpenCL is least available right now is in mobile devices. And despite the excellent work that the vendors have done to make OpenCL SDKs available for their individual parts, none of the mobile ecosystems really support OpenCL. And I think there's a lot of things going on that have an opportunity to change that over the next one, two and three years.

Michael Wong:

And I would echo that for SYCL. SYCL has made surprisingly large inroads in high-performance computing and ADAS, advanced driving assistance code for self-driving cars. We've made some inroads into FPGAs, AI processors and machine-learning processors and custom processors. And as Eric said I, when you asked that question about SYCL and mobile, I realized that we haven't really made much inroad in that direction, so we could certainly do better there.

Michael Wong:

I think all the time, SYCL direction will certainly be augmenting things like … of course, this depends on how many people we have in the working group towards things like more C++ features. And I mentioned that before. More safety-critical features. We view that … and that's an underpinning of Khronos as well with our safety staff counsel is to make the language safer to be used in various future applications that we anticipate rapidly coming. And I think OpenCL also has that same safety direction as well.

Neil Trevett:

Yeah, that's right. I mean, the flexibility of OpenCL 3 is a good foundation for that. If we push forward to a flexible profile, I think that is probably numerically OpenCL's biggest processor adoption opportunity to bring in billions of embedded processors into the OpenCL ecosystem. And of course reducing API surface area, which the flexibility will enable us to do, is vital to safety-critical.

And it's interesting, the embedded space applications hopping from embedded system to embedded system isn't so important. So the flexibility makes a lot of sense. We need though to use the profiles so, in the desktop space where portability is more important to know, we can retain more portability between different vendors. So the profiling and the flexibility that OpenCL 3 gives us is going to enable us to reach into different markets in very tuned ways, optimal ways for each of those markets in turn.

Dennis Adams:

That would extend mobile to then move into the edge. On the desktop, you have some great choices and open standards and things like that. But as you move closer to edge, you basically have a vendor API that you have to deal with. And so being able to get these open standards into edge would give us the same flexibility that we have on desktop everywhere.

Simon McIntosh-Smith:

Thank you. We're almost at time. But before we wrap up, I just want to remind everyone on the call. Don't forget, you can get involved in OpenCL and SYCL. This is the great thing about open standards. You can be directly involved, you can submit ideas, you can really get involved in where things are going. So go and have a look at the Khronos website. Actually, you're encouraged to get involved, so please, do.

If you are interested, if this has whetted your appetite, go and find some way of getting involved. That'd be great. Also, to remind you that all of the materials for the IWOCL and SYCL conference are now on the IWOCL website. You can find all of the presentations, all the videos, all the papers and proceedings on that are now available there for free, so go and check out all of those.

Even though we are about to finish, we are not all going to disappear. There's going to be ongoing chat on the Slack channel. You want to find the live Slack channel, you can go to the IWOCL website. Again, you'll find that. Many of the authors on the papers are hanging around, so you can ask questions of the people who submitted things. Many of us will go and be on

Slack after this as well. If you want to ask more questions, come find us on Slack.

So that is all of that. And just before we finish, I'd like to thank everyone who's been involved. So many thanks to our two speakers, Neil and Michael, today. Thanks also to the panellists coming along. It's been a really good discussion. Thanks to all of you who've joined us. We've had a lot more people on the call today than we normally get in person. So that's been one silver lining to the cloud of running a digital event.

So, thank you all very much for joining us. And also, you see people like Tim Lewis, James Jordan and Jeff Phillips, Tom Deakin and several others who've all been doing all the organizing behind the scenes to make this possible. So I know we can't get around applause them all, but just to let you all know that we really appreciate your time. I mean, it's been fun doing this as a digital event. It's the first time we've done it.

But hopefully, we will see many of you again face to face when we make it to Munich this time next year. So thank you all for joining us. We're hoping that the recording from this panel will go on the IWOCL website. But thank you for joining yesterday.

Enjoy all the great content that's now on the IWOCL website. Thank you all for joining us. Stay safe and hopefully, speak to some of you soon on Slack. Thank you, everyone.

--- ends ---