Code Together Podcast
Episode 5: Collaborating to Build a Heterogeneous Future
Host: Nicole Huesman, Intel
Guests: Jeff Hammond, Intel; Ronan Keryell, Xilinx

Nicole Huesman: Welcome to *Code Together*, an interview series exploring the possibilities of cross-architecture development with those at the forefront. I'm your host, Nicole Huesman.

We are living through tumultuous times. Pivotal times. And like our world at large, research and industry face big challenges. Today, we'll talk about collaborative, open approaches as key strategies to solving some of these challenges. I'm joined by Jeff Hammond, principal engineer at Intel. Hi Jeff.

Jeff Hammond: Hi, Nicole.

Nicole Huesman: And Ronan Keryell, principal software engineer at Xilinx. Great to have you with us, Ronan.

Ronan Keryell: Hi, Nicole and Jeff.

Nicole Huesman: Ronan, why are open standards so important?

Ronan Keryell: Well, when you use a lot of computers, you have huge problems to solve. You will have a lot of researchers or a PhD student and so on to try to apply their questions into some code and to run those huge models such as for global warming or epidemiology. And at the end, you want to use these huge machines in the most efficient way. So, if you are able to use open standards and open language, for example, or open tools, then you will not waste your time reinventing the same wheels again and again. And if you can work with bigger communities, you will be able to solve bigger problems in a more efficient way. I think that's a way not to derail a lot of resource and to focus on solving problems.

Nicole Huesman: And Jeff, you've worked and spent a lot of time in open standards over the years. Is there anything you'd like to add?

Jeff Hammond: Well, there are a couple things. One of the most obvious places where standards made an impact on the scientific work that I was doing is, in the MPI forum, which is a standard layer for supercomputing, communication between nodes. Before we had a standard, people would write the code for every machine. And when the machine got turned on, then they would get on it and they would start to implement the thing that made it work. And in some cases, in the case I'm thinking of, it took a year before they actually got it working. And so that was one year out of the life of a six-year supercomputer that was wasted because it had to be ported to the proprietary or whatever, low-level layer. And when we got stuff into the standard that made that application run without machine-specific code, we were running on day one. That's a year of productivity just by being able to get stuff working. And it doesn't mean that you have to stop there. And this is, I think, a key misconception of standards. Standards don't mean that you're stuck with a code, write once, run everywhere with the same code. It just means that your code actually works the first day you use a machine, and it's up to you if you want to specialize.

Code Together Podcast
Episode 5: Collaborating to Build a Heterogeneous Future
Host: Nicole Huesman, Intel
Guests: Jeff Hammond, Intel; Ronan Keryell, Xilinx

Nicole Huesman: Let's shift a bit and talk about open source. Ronan why is it important for companies to participate in and contribute to open source projects and communities?

Ronan Keryell: Open source software I think it's good because it's an easy and efficient way to collaborate in an inclusive way across the various communities, and it is even easier nowadays, since we have social coding infrastructures such as GitLab, GitHub and so on. This open source community allows us to solve huge problems and to collaborate between different companies. Open source practices are well adopted by software companies, but the story is more complex for hardware vendors, probably because there is a difficult intellectual property story that they have to protect there. They use open source, but they keep it to make their products and then they do not contribute back to the community. On the long tail, they end up being separated from their original community—so, they have some old software internally that cannot benefit from the modern evolution of the main open source software they have used in the first place. If you compare it to a more collaborative, open source solution where you improve it for your product, then you contribute back, and then some people will take your improvement and will also improve it, and then you can benefit back. So that's a win-win situation that is, I would say today, a little bit difficult to understand for hardware companies, but, well, we are making progress.

Nicole Huesman: It's been amazing, hasn't it? The open source landscape is really changing so much. And it's amazing what folks may not realize who aren't as immersed in it, right? There are so many companies who, on one level, are really competitors, but in open source, they're collaborating side by side. Jeff, what do you think are some of the misperceptions or things that are misunderstood in open source?

Jeff Hammond: One of the things that's misunderstood is just how diverse the landscape of open source culture and licensing is. Ronan alluded to a number of those different options. One type of open source is like the right to change the oil in your car without going to the dealership. It's just being able to look under the hood and say, what do I got and I want to change it. So I've got the code. And if I decide that the code as provided doesn't do what I want, I can fix it. The other part about it is this feedback loop that comes when people collaborate together. I actually talk to Ronan much more often on GitHub than any other medium because he follows my project. I follow his project. We talk to each other there. I ask for things, he provides feedback, and it's a way of interacting with people and working together that is just fantastic. And if there are reasons that people can't share things, well fine, they don't have to do that feedback loop. But it certainly helps if they do. And I will say, you know, from the Intel perspective, a lot of the things we're doing with oneAPI and DPC++ are driven by the open source [community] and the first thing we ever did with DPC++ was announce to the LLVM community that we were planning on supporting SYCL, and we were putting [our contributions] on GitHub. That was the place we started. We didn't start with a product launch that was actually months later. And we continue to have a wonderful collaborative relationship with Codeplay and with Ronan, and a whole bunch of other folks who, you know, we have customers who get to provide feedback and discuss with us on

---

GitHub. We have other developers, you know, people who would nominally be considered competitors are still interacting with us because there's value in talking about problems even if people choose to solve them in different ways and in different forms.

Nicole Huesman: This is actually a great segue into talking more specifically about the ISO C++ and Khronos Group SYCL standards. So, Ronan, tell us how the C++ standard is evolving to address today's challenges.

Ronan Keryell: Yeah. So, C++ and C are very successful languages. I discovered recently the ISO C++ committee, and I found huge momentum there and a huge open source community to make it better every day with very smart people. So it's amazing what's happening there. More than 200 people are collaborating to make this C++ language better. And I think that's unique in the world. There are no other languages designed by committee. And compared to old C++, it allows you to express and to solve some complex problems in a simpler way. It's also interesting to see a global convergence between languages. If there are some great ideas in some languages, then the other languages will incorporate these features. Today in C++ you have this multiparty language where you have object-oriented generic programming, functional programming. And at the same time, it's still close to the metal from an efficiency perspective, which means that, even if you have this high-level concept, you will not throw away those efficiencies, which is the main goal to use languages such as C or C++. The idea with SYCL is that we've looked at this modern C++ as a first starting point because it didn't make any sense to start from an older standard. And we realized that all these new modern features, and at the same time this bare metal efficiency, was a great starting point for intelligent programming. So that's why we have started this SYCL initiative around 2011 and we've built on top of this and I think at the end, it's a good, trade-off.

Nicole Huesman: Jeff, can you talk a little bit about Intel's choice using SYCL as the basis for DPC++?

Jeff Hammond: The challenge when Intel decided to go big with GPUs, especially, was there's obviously some things out there that people use for programming GPUs. Everybody knows about CUDA obviously, and CUDA's proprietary, and that, has some obvious challenges. And we didn't want to do proprietary. That was one of the key things that we concluded early on. We didn't want to have the Intel GPU language or whatever it was. And everybody recognized that OpenCL was a thing that existed and had some limited success. OpenCL is not extremely widely used in HPC for a variety of reasons. We looked at OpenCL and we said, okay, there are some things about this that are right and some things about this that are not right. And one of the big things that we recognized was that modern C++ was really becoming the foundation of new code that was written in HPC and AI and a lot of other domains. C++ is rapidly becoming one of the most popular languages. As far as HPC, people who want to program high-performance workloads in whatever domain, C++ was really coming on strong. I did a survey. I looked at a lot of what our customers were doing at the Department of Energy and other places. And I saw a really strong trend toward modern

---

C++. So, C++ 11 with Lambdas and trending towards the Parallel STL that we know in C++ 17. But when we look at that, we see that there are some key pieces missing, and SYCL addresses those pieces very directly. SYCL is a heterogeneous-aware programming model. It's capable of understanding that there are multiple types of processors and multiple types of memory in a system in a way that today's standard C++ really doesn't do. And we also did a detailed study to ask, is SYCL sufficient in its current form? And we concluded that, no, it wasn't. It was very good, and it was the place we wanted to start, but SYCL, as it existed in 2018, or whenever we did the survey, was missing some key things. And some of those features we identified very early on were reductions, better expression of Atomics, unified shared memory, or pointers. And as I think everybody will see in the provisional spec and obviously all the work that's gone on in collaboration with Codeplay and Xilinx and other places, we were able to build those features into SYCL. So, you know, people like to criticize standards—well, standards don't have to stay where they are; they move just like everything else. And maybe they move a little bit slower than proprietary things because you can't have a, you know, by fiat model of what defines the language. But I believe we got a lot of value out of working together with the Khronos community to come up with a language design that multiple people thought was a good idea for a lot of different hardware.

Nicole Huesman: And Ronan, can you talk a little bit about why you find SYCL beneficial such that you built out triSYCL as a SYCL implementation?

Ronan Keryell: I started in the HPC world with my Master and PhD in the 80s, and I've seen a lot of dead languages along the road. A few years ago, I saw a presentation about modern C++, and said, oh, wow, we can do a lot of stuff with this C++! Actually, I was not aware of that, but with some people introducing me to this fancy stuff, I've seen the potential. And when I was working before at AMD, I discovered also Khronos in my team and I was introduced to this SYCL stuff and I realized, oh, wow, it's actually better than some other tools we are working on inside the company, so perhaps we could collaborate on these. So that's how I started this triSYCL implementation around 2014, first to help the SYCL committee. You'll have to understand, we thought there was a real potential to really improve the way to program computers nowadays in a very explicit way so that you can really control everything to be very efficient. And at the same time, since you rely on modern language, with SYCL perhaps we found a way to revolutionize the area of extreme computing and extreme embedded systems.

Nicole Huesman: You know, we're headed towards a really exciting milestone for the SYCL community with the SYCL 2020 provisional release. Jeff, can you talk a little bit about that?

Jeff Hammond: Yeah. The one big feature that everybody's excited about and I am really excited about is USM, unified shared memory. When SYCL was designed, they took a conservative approach in the language that allowed them to capture a lot of different hardware devices. There's some weird hardware out there that doesn't really like the concept of pointers. Pointers are kind of a foot gun and, some programmers do bad things

---

with them and they actually require the hardware to do some smart things with address translation and the like. So, the original SYCL has something called buffers and accesses, and this is a very elegant, opaque design. I think it's brilliant. It works really well. But because it's conservative from a language standpoint, it means that the programmers who can use it, find they get great results, it works, but it can be difficult for people to map existing code onto that abstraction, and existing code means CPU code means GPU code written in other languages, what not. And the USM feature set really expanded the flexibility of memory management in SYCL. Basically, it addressed a lot of key needs from our customers and internal purposes, and we've seen a lot of codes able to move to SYCL really, really easily by virtue of having that. It also allowed us to build some even better integration with modern C++. You can actually take a USM allocator, which was part of the design, and attach it into the object STL containers, and then you can use C++ just by the book in terms of ISO, and have it use the USM memory allocator behind the scene.

Nicole Huesman: Ronan, you want to chime in here? I'd love to hear your thoughts about the SYCL 2020 release as well.

Ronan Keryell: Yeah, I think there are a lot of new, new things and recently we got a lot of new people, new user groups participating in the standard. Intel and other companies have put a lot of effort into improving this standard. And what I find very exciting that's something we figured out when we were doing SYCL 1.2.1, which is a previous version, is that, okay, wow, there is some great potential. And at that time, we were just targeting OpenCL devices and then we realize, oh, SYCL is quite bigger than only focusing on one underlying API. So we have to be bold and proud and go further. And I think that's one very interesting aspect is this context of generic backend. So now with SYCL 2020, you can actually address a lot of different architectures. Now, with SYCL 2020, we can really be very, very heterogeneous.

Nicole Huesman: This has been really interesting. Is there anything that we haven't covered?

Jeff Hammond: The last thing I would want people to know about SYCL, if they haven't taken a look yet, is the diversity of the ecosystem already. Ronan has his implementation and I actually use his implementation on my laptop a lot of the time. And there's the Intel implementation, which of course comes in the product form, which, you know, you can get the binaries from oneAPI downloads or you can grab it from GitHub and build it yourself. And, you know, the Intel DPC++ open source compiler has support for both the Intel products and anything that supports the SPIR video OpenCL, but it also has the Nvidia PTX backend that's done by Codeplay. And then Codeplay, of course, has their own product, which is ComputeCpp, which supports Nvidia. And then, of course, their ComputeCpp supports the OpenCL sphere V landscape as well. And then there's also, hipSYCL, which is based on the HIP tool chain, which folks might know is AMD's HPC software stack for GPUs. And, the HIP stack supports AMD Radeon and, their GPUs, and then Nvidia GPUs, and then hipSYCL also supports OpenMP for CPUs. And what's really neat about that is you have four

different compilers that are all pretty high quality. I mean, none of them are perfect, but I've used them all. And they're all pretty good. And they support a tremendous diversity of hardware. So, you know, Intel stacks supports Intel FPGAs, Intel GPUs, Intel CPUs. All the CPUs are in there. Ronan could talk about, you know, FPGA support in triSYCL if he wants, but, there's a whole lot of hardware out there and it means that you have choices as a user. You can try multiple compilers and that's a really great way to debug code and identify whether you're dealing with undefined behavior, or a bug, or maybe it's just the implementation bug and it's not your fault. It's something you can only see when you have multiple compilers that implement the same language standard. And you only see that, of course, with standards; proprietary languages tend not to have multiple implementations because they're usually developed by one company. Why would one company do it twice? So, I just would encourage everybody to go look and see how neat of an ecosystem there is and how much there is to use today across a tremendous variety of hardware platforms.

Ronan Keryell: But yeah, actually there's another aspect we've not talked about. The fact that, since it's single source pure C++, a SYCL program can run on the CPU, so as Jeff said, on his laptop, which is amazing. For example, that's what we do with triSYCL is that we can run some code that would run on some hardware, such as an FPGA or CGRA, but you run it on your own CPU, on your laptop, in a kind of emulation mode. And then you can debug the same application, but in a quite easier way on your laptop without the real hardware, which means that you can even use SYCL to debug future architectures that do not exist yet. So you can really invent some new SYCL constructs and try them before deploying in some non-existing hardware yet. That's amazingly powerful.

Nicole Huesman: That is amazing. Ronan. Thanks so much for your insights. It's been such a great discussion.

Ronan Keryell: Thank you.

Nicole Huesman: And Jeff, it's always such a pleasure. Thanks for joining us.

Jeff Hammond: Thank you.

Nicole Huesman: For all of you listening, thanks so much for tuning in. Let's continue the conversation at oneapi.com. Until next time.