Episode 21: Driving Innovation Further Faster through Open Standards
Host: Nicole Huesman, Intel
Guests: Dr. Ruymán Reyes Castro, Codeplay Software; Kevin Harms, Argonne Lab

---

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

We've continually talked about the increasing adoption of SYCL for heterogeneous computing, and March brought exciting news. The National Energy for Research Scientific Computing Center (NERSC) at Lawrence Berkeley Lab and the Argonne Leadership Computing Facility, or ALCF, are working with Codeplay Software to enhance the LLVM-based DPC++ open source compiler, based on the SYCL standard, to support Nvidia GPUs. (Refer to the announcement here.)

Today, we're catching up with Dr. Ruymán Reyes Castro at Codeplay Software and Kevin Harms at Argonne Lab about this collaboration.

As CTO at Codeplay, Ruymán has deep experience in High Performance Computing with a focus in compiler and runtime development. He has been one of the major contributors to the SYCL programming model and has led the research and implementation teams for ComputeCpp and the DPC++ CUDA backend within Codeplay. Thanks for joining us, Ruymán!

Ruymán Reyes Castro: Thank you.

As Senior Software Developer at Argonne, Kevin leads the ALCF's Performance Engineering team. His research interests include Parallel I/O and File Systems, High Performance Computing and Storage, and Platform Analysis and Benchmarking. Great to have you with us, Kevin!

Kevin Harms: Thanks.

Nicole: So let's dive in. Kevin, I'm sure some folks may be wondering why does Argonne want to enable Perlmutter at Berkeley Lab?

Kevin: Well, if I get started off with a little bit longer answer, I mean, the labs have a long history of working with each other. So, we at ALCF have always worked closely with NERSC on various aspects. And so of course, you know, we're always interested in working with them. But, more specifically for ALCF's upcoming exascale system, Aurora, it'll be featuring Intel GPUs using DPC++ as one of our key programming models. And of course we'd like to enable portability across the various big compute systems in DOE. And so, we wanted to work with NERSC because they were also interested in enabling SYCL on their platform. And so, we kind of worked together with them and Codeplay to come up with some arrangement to help enable the SYCL DPC++ work that will work on their upcoming Perlmutter system.

Ruymán: I think that's very exciting. So, globally we have been working in SYCL since, well, before it was called SYCL, probably even since 2017. And one of the main things we have always been looking forward is to make sure that we have the right programming model for scientists and for developers that want to write code that is portable across different architectures. And once we get to the point that people in the HPC environment started to use it, including national labs, then we quickly realized that we needed to tackle the elephant in the room, which is CUDA, right? So, there's a lot of code written in CUDA and we need to be able to lift that code up to an open standard and make sure we can port it to all their

platforms. And that's why we got involved with DPC++ in the CUDA back-end, and that's why we decided, well, we get SYCL code, right, and CUDA, and we can make this ecosystem much more open and much more accessible to everyone. And then once we did, then the first prototype that we contributed back to the DPC++ repo, then we got the ball rolling, we got the SYCL spec now supporting things beyond OpenCL. And that's when we got engaged with national labs here and with Kevin and others, and we realized that, well, let's make these something that scientists can actually use. So let's make this something that people can write code today on their platforms and have the code ready for the future, for whatever architecture comes later on. Like, I mean, ALCF has the case of the Intel GPUs coming, but there might be others coming in other labs that will be supported with the single programming model. And that's very exciting. And we were very, very keen on contributing to this effort.

Kevin: You touched on one of the things that is very important to, I think, Department of Energy and the various labs, is portability. A lot of the users who will run on NERSC will also run on Argonne's Aurora machine or Oakridge's Frontier machine, and we want to enable abilities for those codes to be run in multiple places without needing to re-engineer them all. And the Department of Energy has always been out in the forefront in regard to portability. I mean, there are key players in the MPI standard, key players in OpenMP. And so, we've kind of gotten a bit behind the eight ball with CUDA becoming such a large platform, but only supported on a single system. And so, we really want to push the envelope and bring SYCL and DPC++ to the forefront because those are open standards and also open source and develop an ecosystem around those.

Ruymán: Yeah, I think that's really interesting. One of the main things that is there is you have all of this CUDA code, right, and you have all these investments in CUDA, and you may end up, for the sake of argument, CUDA tomorrow came and says, well, now it's an open and anyone can implement it, right? Even if you had that, there is still the fact that who is controlling that environment, who's controlling the features, who's controlling what you can do with that programming model, right? The best thing about having this open standard like SYCL is that these different companies with different interests and different hardware platforms that all want to work together to make sure that we drive the standards towards an environment that can be portable for different systems. So when the national labs decide to say, I'm going to write this code, they'll have to think, 'Oh, if I go down this path, I'm going to be locking myself accidentally to specific hardware architecture.' Whereas, if you go down [the path] with an open standards, you say, 'Well, I'm going to use the code of the open standard today, then it can also work with the standard body and influence a standard body to make sure that the vendors can work together and give me the features I need,' and I have much more ways to convey what and have them supply my requirements, which is a very important thing for long-term existing code bases, right? It's not something you are going to be only maintained for two, three months. Some of these code bases can be maintained for years.

Kevin: Yeah, completely agreed. One of the major things that's important for DOE is obviously our main focus is high performance computing, but we also recognize high performance computing is a small market in the overall market of computing, and we want to have an open system where people can contribute and collaborate and innovate that's not just in HPC. Like, we need to have a wider community than just HPC. And so, like, having an open standards body where people can come and join and submit feature requests or come up with new ideas is very important.

Ruymán: Yeah. I think this collaboration between industry, academia, developers, and just people in the open is very useful to improve the quality of the standards, improve the quality of implementations and just to drive the science and everything around it forward. It's better to have these things in an open community.

Kevin: Yeah. So, one of the big pushes over the last, I would say, three to four years within the Department of Energy is the Exascale Computing Program. And a lot of scientists and development teams have been working on modernizing their codes. Now, there's obviously a big class of people still running 30-year-old Fortran code, but because of the drive to prepare codes for exascale, a lot of people started to build on C++. And there's certainly a lot of interest when we can have something like SYCL that brings in modern C++ that enables the newer, latest features. And so, scientists and developers are free to, you know, develop with these new, like, template meta programming and all these advanced concepts that we didn't have. HPC had a tendency to be quite legacy-focused. And we're kind of like getting to move into this new modern age. And that's exciting.

Ruymán: Yeah, it is. And it's good that you mentioned C++ and modern C++, right. C++ is a stable language. It's an ISO standard. And that gives you the stability. This is going to be there for a very long time based on long industry and community support around it. So you have a base language that, you know, it is going to be that forever, in a way, but it's also a language that allows people to build both the low level features that people need, but also the scaffolding that others are going to need to create higher level instructions that will be useful for everyone, right?

These are effective for many different minds. So HPC is one of them and you can see how people are working in things like Kokkos and Raja that builds this middleware that bridges the gap between low-level interfaces or programming models and the codes the scientists are building. Also in many other domains, like finance and industry, or, you know, oil and gas, they all have started to switch, at least partially, to C++ because they see that it's a modern code base that is still something that they can maintain and rely on being there.

It's very good that we are doing this effort with SYCL and C++ because then we are taking what is a solid base for everyone to rely on, which is C++, but then we are going to work on top on having the things that are specifically for the accelerators that we need and we're going to sprinkle those on top of modern C++, so that we can have something that will be useful for everyone in the C++ community. But the important thing is, we will use these under something that the SYCL Working Group has been working on for a long time is, to make sure that the experience we have in SYCL are fed into the C++ standard, and the ISO standard, so that when in the future, there is more support for more innovative hardware in the C++ standard, the experience of SYCL will be included into that, and they will be able to support larger code bases in more ISO-friendly way.

Kevin: Yeah, that's a great benefit because, I mean, obviously there's a lot of people who are interested in pushing features into ISO C++, but at the same time, it's quite a long time before those show up on systems and being able to work on SYCL and have an open source, we can kind of prototype those ideas, look at, see how they work on current systems and evolve things at a little bit more rapid pace than always waiting for the next draft three years from now. And so that's like an exciting thing that all the

labs will be interested in participating in—kind of feeding these features forward toward the ISO C++ standard.

Ruymán: Yeah. From a vendor perspective, so Codeplay, we build implementations for SYCL, but we also continue with ISO C++, it is much easier for us to have a fast turnaround when we talk with people about SYCL on what we can do on implementations on a spec than when we talk about C++ features because when we're talking about what I'm going to do for C++ 23, or even later on, then our customers come to us and they change their minds saying, 'Well, that's going to be something in the future, right? But I have the new Intel GPUs coming, I have the new genesis of CUDA GPUs coming, what are we going to do this year and the next year and how are we going to be sure we are prepared for whatever comes in the future?' And that balancing that we have with SYCL, that we can do something today that will be supporting the future is something that is really positive for the industry at this point.

Kevin: Yeah. And also, you guys have contributed a little bit to some of the like SYCL-based libraries as well. And that's one of the great things that, you know, the labs are interested in building out, not just the language, but the overall ecosystem, getting more libraries, more ideas, more innovation around SYCL and DPC++. And that's definitely something we're interested in.

Ruymán: Yeah. And I think the oneAPI initiative is really good in the sense that we can have a common talking point between vendors and users, right? Because usually one of the problems we have is, well, we go to … not just, not only to us … but companies say, what do you need? And then they say, 'Oh, we need to implement this algorithm,' right? And to implement the algorithm, then we need to have some interfaces, some libraries, and we have to come up with all of these, and we have to explain to the customers on different interfaces, but having the oneAPI spec, which includes the library and the ecosystem there, and things like oneMKL and oneDNN means that we can go to customers and say, right, 'These are the building blocks you are going to need to build your applications,' right? And they're well-defined in their interfaces. You can implement things on top of those today. And then we will work with hardware vendors, or we will work with you to give you the interfaces and the support on the backends that you need. And that's what we did with CUDA, right? So we implemented a CUDA backend for oneMKL. So you can actually use cuBLAS directly. And we implemented a cuDNN backend for oneDNN so you can use cuDNN in oneAPI. So when you write an application using the oneAPI spec with DPC++, and you're targeting the GPUs, if you use the oneMKL and oneDNN interfaces, you know that you're going to be using the most efficient performance building blocks that you have in the platform, right? You don't have to think too much about it because otherwise, if you are doing this by yourself, then you have say, right, 'I am going to be using an Nvidia GPU, therefore I have to go and use the cuBLAS and cuDNN libraries and interfaces here on there, and the pointers and accessors, and all of these kinds of plumbing problems, are for us as implementers and vendors to solve, and it's not for users to care about, if that makes sense.

Kevin: Yeah. And obviously the best languages and the best system are great, but if they give up too much performance, then you know, no one uses them, and so, one of the things that NERSC, ALCF and Codeplay are focusing on is, you know, looking at how performance is on the DPC++ CUDA backend, and, you know, in our initial, some work done by one of my colleagues, Brian Homerding, he saw that, you know, we looked at the RAJA parallel performance kernels, and a lot of those show equivalent performance to CUDA, some slightly better, and then of course, some slightly worse. Usually there are

some differences in the memory access patterns between the generated code, but we're seeing, like, really good performance, and so that's something we want to keep working toward.

Ruymán: Yeah. That's very important, right? And performance comes from having the right programming model, or at least, a programming model that is portable … it's never perfect … and having the libraries there to support you because in some specific cases, some things like matrix multiplication, yeah, you can engineer for matrix multiplication application, but I am sure between Intel and NVIDIA there's been many, many years of engineering effort tuning to the absolute minimal nanosecond or microsecond, it has to get performance out of those, right? So, you might as well use those building blocks instead of trying to come up with your own. And there's also that sense of you writing your own blog. So for example, we have SYCL BLAS and SYCL DNN, which are equivalent to cuBLAS and cuDNN, but for our own hardware platforms, and you can write those in SYCL or you can write the code manually, or you can use auto generated code, so we have auto tuners. And those things can be built on SYCL. But sometimes it's just a matter of finding the right building block and just using it when you know this.

Kevin: So one of the other big areas of interest in discussion that we've been talking about and SYCL is a new SYCL 2020 standard and bringing some of those features into DPC++ because, you know, although there's a lot of interest in buffers and accessors, unified shared memory (USM) is considered, like, a big win, and one of the most critical features to really give the science users a lot of flexibility to build their codes.

Ruymán: Yeah. So, I think there's a space for both, obviously. USM is an important thing when you have an existing code base or you have large C++ code that you need to port to SYCL and you just want to get the first person running and you know, it's going to work, right? You know, you can do your pointers, your offsets and everything, and you're going to get some performance of the hardware supported. So, this is very good that we have in SYCL 2020, and that we have the Intel extensions there.

Our background in Codeplay comes from situations where having USM is not necessarily possible. Although you can emulate it, the performance is not going to be great. So if you are working with non-HPC applications, like, you know, embedded systems or DSP—so, kind of small things in hardware—you need to have something that is much more manually controlled when you use buffers. And that's why we have buffers and accessors from the beginning of SYCL. And sometimes when you have a code that you have to modify to use buffers and accessors, then that forces you to see it in a different way. And you need to change your data structures to make sure that they will fit with buffers and accessors. And when you do that work, we have seen cases that people, often just by doing that conversion, give you more performance, even when you use the same platforms that are supported, right, because they are forcing you to think in a different way about your code. I think it's very good that we have USM because one of their higher entry barriers for SYCL in SYCL 1.1 was the fact that people force it to go through the buffer and accessor path, and if you had just wanted to port the last code base, you don't necessarily want to do that therefore, right, because you don't know where it's going to take you, or it might just be impossible, right? Sometimes just, you know, having to modify a million lines of a code base to use buffers, and changing the pointers to buffers everywhere … basically, we have CUDA before … it's just too much work, right?

We did originally, Eigen, which is a math library that was done that had a CUDA backend, and we wrote a SYCL backend. Obviously, this was even before SYCL 1.2.1. We did it in one of the big releases of SYCL that was using buffers and accessors. We still maintain that code today. But there was a tremendous amount of effort for us to do it, it took us like a year and a half just to get everything working. And that experience actually means that now we have a very efficient Eigen implementation that we can use for a lot of different hardware, right? So, it has value on its own. But if we were to do that today, instead of doing it from scratch on buffers and accessors, we will start with USM, have something done in, and then we can start, you know, optimizing the bits and pieces that are important, and maybe port those to accessors and buffers.

Kevin: Yeah. I think that's one of the most important parts for codes is the USM speeds up the initial porting because oftentimes these very large scientific codes, possibly 90% of the code base is functional aspects but are critical to performance. So USM makes it easy to get all of that working. And then later that 10% of the code you can potentially switch over, optimize, but it's a much more rapid startup.

Ruymán: Indeed, and that rapid start-up is what gives you the edge. When you have someone that is evaluating the programming model or the platform, and has a window of, let's say, a month to get some results and see if this works or not, if you have to spend that month writing code with buffers and accessors, you might not get to see to the end, but if you have USM, then you can have any support in very quick and very early on, and that gives you the value of the programming model, and then you can move on and say, well, now I'm going to invest more time and effort to implement and other ways of doing it, right?

The other thing that I really liked from SYCL, and maybe it's not the SYCL 2020, but just SYCL in general, is that when you use buffers and accessors, you have this data flow model that allows you to automatically scale interdependencies. When you use USM, you have to kind of monitor and use events, which is kind of what CUDA does, but one of the features from SYCL is, if you go down the buffer and accessor path, then you can let the SYCL runtime do optimizations under the hood and figure it out when to do copies and when to update processor memory, and that's also an interesting feature of SYCL.

Kevin: Yeah, for me personally, one of my other favorite features in SYCL 2020 isn't a particular feature in itself, but many of these things like buffers, accessors, queue management. There's been a lot of simplifications in the syntax that make writing code a lot easier, a lot quicker. And that's just easier to teach people when, you know, it's not quite so dense a syntax and a lot of those elements, it's just streamlined. The overall language is great.

Ruymán: Yeah. I think there has been a lot of work, especially from people like in Intel to simplify the syntax, right? So you have the same features—you have even more features in SYCL 2020—but the way you can write them and you can benefit from things like CTAD in C++17 it is much simpler. So, your code, it looks cleaner in a way, and it's much easier to read. So instead of not seeing a lot of templates that some people don't like, then you get to see much more code at this level. It’s really good that all that effort in simplifying the syntax has happened on SYCL 2020 because at least I now can see my code in the slides.

Kevin: Yeah. That's always a good thing.

Ruymán: Yeah. It's easier for you to explain to customers.

So Kevin, one of the things we have been doing on this project is to enable features like Tensor Cores and async_copy on Nvidia A100 GPUs. There are similar features in other hardware architectures. But what are your thoughts on enabling these from the programming model perspective? Do you like to see these things as something that magically happened? Or do you want to have control on how to write codes that will take benefit of these things, even if that makes the code a little bit less portable?

Kevin: Well, I think, you know, the science user would obviously like the most optimal things to happen with the least amount of work, right? That's always the case. So one of the big things that I think we like about SYCL is there's the opportunity to have some non-standard things or extensions where if you want to support a Tensor Core, okay, what do we need to try to add in to support that specific hardware and see what kind of performance gains that gives? And that's a great experiment and good for people who want to run today. But the worst thing I think that can happen is where people try to jump ahead and be like, okay, let's have this somewhat language-hardware-independent feature that will somehow be implementable by everyone. And that's easy to get wrong and just creates problems. So I like the Argonne and NERSC, we'd like to work with lots of different hardware vendors, and so we don't want to lock anyone out or lock anyone in. So just having the flexibility to try different things, work around, and then eventually when we see there's some commonality, we can start driving toward a standard and then standardize for the future. Because I think we're at a point where specialized hardware like matrix engines, Tensor Cores, those types of things are going to become more and more popular to get past the Moore's Law type evolution. And so, you know, I think we're going to need to figure out how to support those things. And I think it's important not to jump in too soon, and kind of feel our way around, and see what works and then try to bring it to the language spec.

Ruymán: Yeah, I think that makes a lot of sense. One of the good things of having an open standard within the SYCL Working Group is everyone can contribute, right? And the way we will do this in the SYCL Working Group is every different vendors can come up with vendor extensions. So we could come up with a vendor extension from Codeplay to support Tensor Core in Nvidia GPUs, right? Other vendors like Intel could come up with an extension to support their own matrix engine kind of thing. And after we have a few vendor extensions and we have some implementation experience, and we can all come together in one of the SYCL Working Group calls that are IP-safe and we can discuss what are the commonalities between the two, what are the things that we want people to be able to target, the things that will get performance out of it and will make it usable, and the national labs and others can be there having these conversations, and then we can come up with something that works for everyone, right?

And maybe what works for everyone has a lot of different things. But in the end, it's going to be a solution that the community has agreed upon. And that's going to be better than anything that we can come up with in a desk somewhere just drawing in a board, right? We don't want to have pragmas inside of a camera doing magical things just because someone's thought it will be a good idea, but we want something that will be an effort from everyone that has a stake on it.

Kevin: Yeah, exactly. I think even though vendors might be competitive at the end of the day, they all want to kind of bring together some like, quote unquote, good solution, right? That requires talking and working together. And I think, you know, SYCL provides a good forum for that.

Ruymán: Yeah. And there is no point in the vendors involved in the SYCL Working Group to create some accidental vendor lock-in because the point of working together is so that developers see the value of choosing the right platform for them, right? And if your platform is best for all of the cases, that's even better, right? But by working together, we ensure that vendor lock-in doesn't happen.

Nicole: This has been such a great discussion. Unfortunately, that's all of the time that we have today. We'd love to have you back on the program to talk about the progress of this collaboration as you get further down the field, so to speak.

So with that, Ruymán, as we wrap up today, where can listeners go to learn more?

Ruymán: We are working in the Intel LLVM repo. So you will see in the repo some of the partners from our team there. We have a landing page for the DPC++ CUDA. And obviously, you can reach out to us in Twitter or emails or anything to get more information.

Nicole: Great. Thanks. And Kevin, where can we point listeners to learn more?

Kevin: If you want to know more about Perlmutter, you can go to nersc.gov and find lots of details about Perlmutter there. And for Argonne, if you'd like to learn about our next upcoming machine, Aurora, go to alcf.anl.gov and find all about the resources we have.

Nicole: Excellent. Thank you. It's been really exciting to hear about this new collaboration. And as I said, we really look forward to hearing more. Kevin, thanks so much for joining us.

Kevin: Thanks for having me on.

Nicole: And Ruymán, thanks so much for sharing your unique perspectives with us. We look forward to catching up with you as we get down the road here.

Ruymán: Thank you for the time here.

Nicole: And a big thank you to all of our listeners for joining us. Let's continue the conversation at oneapi.com.