Tony (00:04):

Welcome to Code Together, a podcast for developers by developers, where we discuss technology and trends in industry. I'm your host, Tony Mongkolsmai.

(00:17):

In several of our past episodes, we have talked to various experts who work on the Aurora project at Argonne National Laboratory. We've talked about how scientists will utilize the various capabilities of the hardware and software that Aurora provides. And today we're going to talk a little more about one of the core programming models at the heart of Aurora, SYCL. We've talked about SYCL in the past in this podcast, so we won't go too much into detail about what SYCL is. But today we'll talk a little more specifically about the benefits to Aurora's architecture using the SYCL programming language, and where SYCL fits into the landscape of programming models.

(00:50):

Today I'm lucky enough to be joined by two experts, Nevin Liber and James Reinders. Nevin is a Computer Scientist in the Argonne Leadership Computing Facility division of Argonne National Laboratory, where he works on Kokkos and Aurora. He also represents Argonne on the SYCL and C++ committees, the latter as the Chair of the Admin Group. Vice Chair of the Library Evolution Work Group, Study Group 18, which focuses on technologies that aren't quite ready for the Library Evolution Work Group. And he's the Vice Chair of the US delegation to the C++ Committee. Welcome to the podcast, Nevin.

Nevin (01:23):

Thank you. Glad to be here.

Tony (01:25):

We also have James Reinders, who's an engineer who has helped develop some of the world's fastest supercomputers and the software tools to program them. He has authored 10 technical books about high performance computing. And James works to promote use of open multi-vendor solutions, including Fortran, C++, SYCL, oneAPI, OpenMP, and MPI. Nice to have you James.

James (01:45):

It's a pleasure to be here.

Tony (01:49):

So today we're going to focus a little bit on SYCL and how Argonne is using SYCL, so for that lets go to Nevin, who's our expert here. Why is Argonne and Aurora focused on using SYCL to enable the Aurora project?

Nevin (02:04):

First off, it is our compiler vendor tool from Intel, oneAPI is based on SYCL. And what we are looking for at Argonne is, we want something that is device independent. Because as new supercomputers show up, we need something that could be very easily ported to those new supercomputers. If we tie into one vendor's architecture, that's a much harder proposition. And so SYCL meets that bill fine and also because we can get involved in standardization, we can influence the direction it's going to take. And the other one would be like OpenMP, but OpenMP is more C focused where SYCL is really C++ focused.

Tony (02:40):

And James you've had a lot of experience over the years in your career at Intel. When I started Intel, you were already talking about compilers and languages and things like that. Can you talk a little bit about why you think SYCL is a nice evolution on top of the things that Nevin was talking about, like the OpenMP, et cetera?

James (02:58):

It fits very well into the history of these things. As Nevin said, it's critical to evolve what we're programming in the manners that are more portable and performance portable. If you look back, Fortran was introduced in 1957 I think, by IBM, it was an IBM language. But it evolved into an ISO standard that we all use. OpenMP was born out of many different directives, commonly called the Cray Directives or there were, VAX had a set of directives. They did the same thing.

(03:32):

And now we have a situation where there's an explosion happening of varieties of different hardware accelerators. Intel's introducing its Ponte Vecchio which will help power the Aurora system at Argonne. NVIDIA has its CUDA, AMD has its ROCm. It's a logical time to look at how can we have industry standards consolidate those so that we can all be programming the same thing. That's where SYCL and where oneAPI come in.

(03:59):

It follows a tradition in the industry of looking for vendor neutral solutions that are generalized a little more. It takes time and we should give credit to the things that come before that that help inform us, that blaze the trails. But it is logical to want to generalize those and it solves real problems.

(04:17):

So that's why I'm excited to be working on SYCL. I think it's creating an industry dialogue on what is the right way to really get this right so that we're not advantaging one hardware over another, but we're really trying to serve the needs of all of them, which is what the users need.

Nevin (04:33):

And these are needs that, I don't want to say there are holes in both C and C++, these are things that those languages are not currently addressing, because there's no general idea of what hardware looks like in terms of heterogeneous hardware. C++ doesn't even know two processes on the same CPU work together.

James (04:51):

Absolutely. I'm actually conservative in this manner. I don't think it's time to rush into adding those two languages. I think that's a fools errand. OpenMP has been around a long time solving problems like this and it's not been absorbed into C++ or C or Fortran. Maybe SYCL never gets absorbed, maybe the concepts of remote memory or the concepts of disjoint compute units stay outside the language. I don't know, that's a good topic to discuss, but it's one that should not be rushed too casually. So SYCL plays an important role at adding that functionality into these environments like C++.

Nevin (05:27):

It gives us a playground for practical experimentation with real world problems, not just some theoretical kind of thing. And yeah, if we can find very common abstractions, we can propose them for

the C++ language. And make no mistake even proposing something, that can take 10 or 20 years, these are hard things. It takes a long time to get put into the standard. Because once it's there, it's there forever and in some countries it has the force of law.

Tony (05:53):

What's interesting there is like GPUs are relatively new. I mean if we look back to what the original Voodoo processors in the late nineties, but compared to what you were talking about with Fortran in 1957, they're still kind of in their infancy. And it's really interesting because now we have other different types of accelerators. Now, we look at the matrix multipliers for AI workloads versus just GPUs which are a straight vector, shader originally for graphics processing.

(06:21):

It is interesting to consider, will SYCL be able to handle the growth of different types of accelerators and then why is SYCL a better method to do that than C++ or some other core language? You guys touched on that a little bit. But it is an interesting thought experiment to see, where should we be thinking it's going to go and is SYCL a good way to enable this?

Nevin (06:43):

I think it is because we can really counter things like FPGAs, right? And we're still relatively young so we can shift things in whatever direction as hardware shifts. And C++, it is not relatively young, it's been around what, early eighties? Can't remember. I started using it as you know since the eighties, but yeah.

James (07:04):

Yeah, finally standardized C++ in '99, I think.

Nevin (07:09):

'99 was the first standard. But yeah, coming out of Bell Labs in the eighties.

James (07:12):

Oh yeah.

Nevin (07:13):

Apple just started using it the late eighties for its OS. I'm an ex Apple employee who was at Apple in the nineties. So yeah, I remember C++ there as well.

James (07:22):

I like to think of SYCL as very foundational, and so I agree with Nevin, I think it's sufficient or it's designed well to stretch across hardware. But it's also very much in the vogue of how C++ does that. It gives you access to programming very efficiently. So when you write very efficient programs, your programs may look a little different on different hardware. I certainly don't think there'll be very many kernels you write exactly the same way to run on FPGA versus a GPU. You can code both in SYCL and SYCL will allow you to pick the different algorithms, but you may find a different algorithm is going to work better on an FPGA than a GPU or a CPU. And so SYCL gives you the foundation to do that.

(08:06):

What I think is really interesting is you can build abstractions on top of that, that are a little more domain specific and may do a better job of stretching across different hardware. But somewhere in your software stack there has to be software that allows you to get the best out of each piece of hardware no matter what it takes. And SYCL gives you that capability. You can build libraries on top of that; you can build other systems on top of that. TensorFlow and PyTorch might be examples of where they can be a little more abstract and can sit on top of something like SYCL to solve their needs at that foundational level.

Nevin (08:44):

As well as SYCL will give you portability. The first thing when going to new hardware is, you want your code to run correctly to start with even before you start trying to optimize it for new hardware. And if you can't accomplish that then it's a lot more work.

Tony (08:58):

So a lot of people when they talk about SYCL or they hear about SYCL, they talk about SYCL being a different language than C++, which as somebody at Intel who works on evangelizing oneAPI, we very often talk about how SYCL is NOT a different language than C++. But since you guys are the experts and you guys actually work on this stuff day to day. Can you guys talk about how you think about SYCL and how it relates to C++?

Nevin (09:24):

Well so far it really is one of our design constraints for SYCL, is that it's for the most part straight C++ code. Sometimes that's about things like hacks. Naming kernels have that weird parallel_for angle bracket, class name, which was weird, and when I first saw that I had to look up something, I guess it was valid C++, forward declaring a local class, which was weird. But that is one of our internal design constraints.

(09:52):

That being said, if we need to make changes to the language, I think we will. But they would have to be very carefully considered because we don't really want a brand new language. We want something that eventually we can take pieces of and propose for C++ proper.

James (10:06):

Sometimes I'm amused by the question because I have friends and colleagues that will debate whether OpenMP is a language or not and I don't think so, I think SYCL is a programing model. But I understand people who suddenly want to label everything a language, I just don't think it's that useful. I think it's very important to see as Nevin was saying, SYCL is extremely aligned with C++. It doesn't try to invent anything new that C++ does. It tries to build on C++. It gives us a capability to find devices, to share data with remote memories, to offload. All concepts that C++ doesn't currently have any concept of.

(10:45):

So SYCL's not trying to modify a language, it's adding in the finest tradition of template libraries, except with a little bit of magic mixed in. So I don't think of SYCL as a different language because I think of C++ as the language. But if you will, it's a programming model that you need to learn in addition to C++. But it's really oriented towards solving distributed parallel programming problems, not basic language problems.

Nevin (11:15):

And a lot of it can be provided by just a SYCL library, although you do better when it is provided as an actual part of a compiler. That may be where people get mixed up because it typically gets provided as part of a tool chain, so they don't see it as a library even though our design behind SYCL is, yeah, we should be able to do it as a library.

Tony (11:36):

I think most of our listeners have probably heard James talk, I think James has been on this podcast before I started hosting it. And Nevin, I don't know if our guests would've known what you're working on. Can you give us a little history of how you got involved in C++ and SYCL? And what are you doing day to day with Aurora? A lot of our listeners listening to this series around Aurora are interested in what are people doing in Aurora at Argonne. Can you talk a little bit about that?

Nevin (12:00):

So I got introduced to C++ in the eighties. I was working at Bell Labs. A friend of mine from Apple gives me a call, it's like, "What do you know about C++? You guys invented it." And that was enough to get me interested. So I was a very shy engineer in the eighties. I found the person who had a tape of C++ and I went to his office, turned out that person was Jim Coplien, started doing C++ and I eventually came up with a question he couldn't answer. At the time, for loops they declared variable inside of for loop, that went past the end of the closing brace of the for loop at scope was the end, and he didn't know why. So I spent two days sweating over an email to send to Bjarne Stroustrup. He was very nice, a giving very nice answer of why it was basically at the time backwards compatibility, although that eventually did get fixed.

(12:44):

And so that's how I got involved with C++ and I've been doing it mostly ever since. I took a little divergence doing other stuff, which I wasn't very happy with. And the way I got involved with standardization is, I went to the first Boost Con and met late Beman Dawes, one of the founders of Boost. And he goes, "Hey, you're from Illinois, we're having a meeting at Fermilab in three weeks." And I could have said, "Wait, wait, I have a job, I can go to a meeting before that." But it's imposter syndrome right, I'm scared to go because all these book authors and luminaries.

(13:14):

And I go to the meeting at Fermilab and come back to my company and say, "Hey we should go in together, we ready, we should." We did. And then I said, "I should go to the next meeting, which is in Madrid." It was my first overseas trip since I was 12, and went to that. And I came back and said, "We should sponsor a meeting." And that took about a week and a half and they said, "That sounds like a good idea." So we sponsored the C++ meeting in Chicago in 2013, and the C meeting for that matter.

(13:40):

And then I knew someone from Argonne and they hired me in 2019. And so I joined Argonne then. And that's when Aurora and SYCL were starting up. So it's like, "We should get involved with SYCL standardization as well as C++ standardization." And that's how I joined the Kronos group in 2019.

(13:59):

So my main project is working on Kokkos, which is a performance portability library. And the big question that comes up a lot is, what's the difference between Kokkos and SYCL? And SYCL for us is a tool that vendors give us, and Kokkos must be a pure library.

(14:14):

We don't want to get into the tool business. Because across a whole bunch of supercomputers, across the Department of Energy labs, we don't have a budget for that and the manpower for that. So that's the main difference between the two. And I work on Kokkos, both SYCL and Aurora specific stuff as well as just general stuff. Right now I've been working on things like parallelism deduction guides, because the next release of Kokkos is going to be based on C++ 17 and we can make things a lot easier with deduction guides, as Intel did with SYCL for SYCL 2020.

(14:44):

It really makes the user experience much better because there's a bunch of template parameters that users may not care about and they're just sometimes hard to specify and they don't need to. In C++ we give trivial examples of using a vector, passing ints. That doesn't really help that much. But in some of the template types we have in things like SYCL and in Kokkos, it helps immensely.

James (15:08):

Kokkos, Nevin you could tell us a little better than I can, has helped inform some things that have affected C++ and SYCL.

Nevin (15:15):

Yes. The two big ones we've had are atomic_ref came out of the Kokkos group, and for C++ 23, mdspan.

James (15:21):

And these are fantastic because they've gotten real world usage, Kokkos implemented them, you've got real applications using them. When things like that make it to standards, they're my favorite things to get put in standards because they're proven, they're vetted, we've worked on different hardware with them. They truly represent the sort of things that when they make it into the standard are likely to hold up and be used for a long time.

Nevin (15:45):

When I say it came from the Kokkos group, that's how it started, but we try to be very encompassing. People are interested, get them on board, get them as paper authors involved with this. And I also think things like mdspan are also affecting the design of SYCL. For instance, being able to change the layout. We're looking at that right now for SYCL Next. How do we integrate mdspan and how do we integrate some of the concepts behind mdspan into making SYCL more general. SYCL's roots are in graphics, right? OpenCL. There's three dimensional limits, we're trying to get rid of and just things like layout which matter to high performance computing.

Tony (16:20):

So out of that, I think there's actually two interesting questions that come out of what you guys were just talking about. The first one is something that probably is more commonly what many people would think of, which is, okay you're saying Kokkos is a library, SYCL is a tool chain. When would I use one over the other?

Nevin (16:40):

It may depend on what your portability needs are. In the Department of Energy, we know what our portability needs are, we know there's going to be a new supercomputer. Every lab when it's their time will get a new supercomputer, very likely from a different vendor. So for our use case, we don't know that we would get SYCL from other vendors. We're hoping to, we're certainly going to be pushing them to do so. And so that the balance of what you write your fundamental code in, may change over time.

James (17:02):

And if you fast forward a bit, just ask a question. Let's just say for sake of argument, everyone adopts SYCL, 10 years from now SYCL is what every vendor supports. Kokkos still has a meaningful place. In fact Kokkos gets simplified by having one backend, but Kokkos still serves a purpose. Maybe you could describe how you think about that because what you're saying right now is it's absolutely critical because every vendor is giving you a different backend to use. But even if we consolidate those backends, Kokkos is still very important.

Nevin (17:32):

It's still a common library, it still has the very common abstractions in it. It's why we've been able to add things to the C++ standard and there's more stuff we want to add to the C++ standard that comes out of Kokkos. And it's been a very good place where experimenting... Basically anything that needs to be pure library things, to the standard.

(17:51):

And where there's deficiency. We're all seeing where there's deficiencies, like C++ doesn't support restrict. And we know that's a deficiency, we're given a way in Kokkos and there's ways it's SYCL to add that kind of thing. But we also need to figure out, how do we change C++ to do it? And there's certainly issues with it. People want generalized restrict, which your object has pointers underneath it. That's really hard generally to say in something like C++ versus C where we know all the fundamental types, we know all the types basically.

Tony (18:23):

And then the other question that came out of what you guys were talking about is, you mentioned SYCL Next, it's a very abstract term. Can you talk a little bit about what's going into SYCL Next and why you're excited about it?

Nevin (18:34):

It's the next evolution. We don't have a commit to ship date yet, so we don't have a date for it. We want things like mdspan. So SYCL 2020 is based on C++17. What should SYCL Next be based on? Could be C++20, it could be 23. So the one language feature we got in C++ for mdspan, is we have a variadic operative square bracket, which would be very nice to have right in the next version of SYCL as well, for mdspan and accessors and a bunch of things.

(19:03):

But if we require that then we have to say 23 is our base. And if we don't require it, we are going to have to very likely support operator parenthesis, the function call operator, as the only other variadic way of doing this, roughly the same looking kind of syntax. So it's what the next version is. How do we get rid of some of the three dimensional limits we have? Complex number support is going to be added to it.

James (19:26):

Yeah, what you hear is you hear a lot of concern about C++ and that's very genuine with SYCL aligning with C++ so forth is absolutely critical and it's a very valuable feature of SYCL. So yes, with C++ evolving, a key question before the SYCL committee is, how does SYCL evolve with that? And I think the mdspan thing's very fascinating because while SYCL and OpenCL and others have this tradition of being limited to three dimensions, it seems obvious. Wow, why don't you fix that? Why don't you give us more? Well it's a little more complicated than that because we want to do it with performance, and that creates interesting discussions.

Nevin (20:06):

Yeah, because we can make it variadic, that's not that hard, but how do we do that and give you performance, is a much harder question.

James (20:13):

That's what gives the committee things to discuss. And extremely valuable to get real user feedback and so forth because it's critical that this stays aligned with C++, stays efficient. And those two have some challenges when you try to map it onto diverse hardware

Nevin (20:30):

In some sense it's early days, we just had our face to face meeting last month and we've really just started discussing what we want in the next version of SYCL.

Tony (20:39):

So what do you guys see as the long term goal of what SYCL is enabling? Where should SYCL be focusing its efforts, versus you talked about C++ obviously and C++ being a long time standard, where it's going to focus its effort. Sometimes it seems like they go together, right? You're talking about mdspan existing in both places and being really needed in both places. But where do you think SYCL needs to go, which is different from where C++ needs to go?

Nevin (21:07):

I think we still need to experiment with different kinds of accelerators. I don't know what the hardware landscape in five or 10 years is going to look like. And C++ will not standardize things unless it's pretty sure that's the direction things are going to go. And it's kind of a blurry line. We want more concepts from SYCL put into C++. Not only makes our job easier, but it makes it ubiquitous.

(21:30):

Why we want mdspan is because that is what we call a vocabulary type, it's used across domains that everybody can say, "This is the interface." Because if we have our own, then we have to have ways to convert between our interface and say the C++ interface, and we'd rather not do that. That might cost you performance. It certainly has cognitive load on developers trying to figure out what interface do I have to support?

(21:51):

And they're hard questions. There's times we've put things in name space SYCL that might exist in the standard, or they exist in a future standard. The question is are they the same type or different types? And those are hard trade-offs to make.

James (22:04):

Now my biggest hope for SYCL is that we navigate this difficult thing Nevin was talking about, which is, always giving high performance access to whatever hardware. And we've entered this so called new golden age of computer architecture and we're going to see unbelievable explosion. We're going to continue to see explosion in different hardware.

(22:27):

There's a lot of factors involved. Not least of which is the ability now to hook different chiplets together, means that the barrier to entry for new innovative hardware in a sense is lower. Someone can come up with an innovative accelerator and have it attached to someone else's processor as a matter of course. Very similar to what we used to do by having accelerator cards plugged into PCIE slots, but at a different level. I hope SYCL continues to give us opportunity to navigate that and support all hardware. SYCL's very foundational that way.

(22:59):

It's difficult to figure out exactly where hardware is going. A lot of accelerators now support Unified Shared Memory in one way or another, or they're rushing to support that. C++ doesn't understand disjoint memories and maybe it never should. SYCL does. You could debate whether we should only focus on hardware that supports USM in 10 years or not. I don't know the answer to that. I don't know what innovations will happen.

(23:26):

But I hope SYCL continues to navigate full performant access to any hardware out there as a foundation that we can build on. Because if it fails at that, then we've got a problem. Then do we need to invent more techniques and so on so I think SYCLs up to that task and that's what I hope to continue to see with it.

Nevin (23:46):

Yeah, it's very hard to predict the future where hardware's going to go. I remember when GPUs first came out before Argonne and a few jobs before I was working at a trading firm where we were using GPUs for Black-Sholes models with some very clunky tools because when they invented GPUs, they really weren't seeing a generalized processing, it was graphic processing. And then some very clever people go, "Oh, we have problems that kind of fit this and we can use this as an accelerator." And that also sways where hardware goes as we come up with new problems to solve.

James (24:18):

I believe 10 years ago that we were reaching a point where there was a good argument to be made, GPUs weren't going to continue to be that critical. And then AI, deep learning happened, and oh my gosh, I'd spent my whole career worrying about math precision and larger precision. And suddenly you have algorithms that can thrive on low precision, the opposite direction where we were driving. And yes, that's had a profound impact on the hardware that's needed for these systems. Definitely affected Aurora's design, right?

Nevin (24:50):

Oh yeah.

James (24:51):

People love to report on, "Oh when's Aurora going to happen?" Or this or that. Every supercomputer has this challenge, you're going to build it at some point, does it need some applications. And things have gotten a lot more dynamic, especially with the introduction of AI, as to what an excellent machine needs to deliver.

Nevin (25:09):

Yeah, that's AI. Machine learning is another big related, but different field in AI itself.

James (25:16):

I use AI a little too extensively. There's a very specific set of algorithms that are driving this explosion right now. And who knows what the next set of algorithms will be and it will affect hardware profoundly so. If SYCL can keep maintaining its status of giving us access to that, it gives us a quick way to on ramp new hardware and explore it.

Tony (25:39):

I think that's probably close to the end of our time today. Are there links or references you guys think would be useful to our listeners?

James (25:46):

Well, if you want to learn more about SYCL, lots of resources available at SYCL.tech. T-E-C-H. Link to a book about it, link to tutorials, online materials and so forth. If you want to dig in a little bit more, that's a great place to go to learn more about SYCL.

Nevin (26:04):

And we're very open to ideas for new versions of SYCL. So if there's things we're not addressing, let us know. You may not agree with the solutions we come up with.

Tony (26:14):

That's the state of technology in the world today.

James (26:18):

Yeah, I can't encourage people too much to engage. Nevin was talking about experiences early in his career, which are similar to mine, of being timid about, oh my gosh these luminary people have written these books or are working on these supercomputers. They're kind of scary, aren't they? And it turns out that they're, maybe I shouldn't call them sweethearts, but I'll tell you that the best people I've met in the entire world are these luminaries.

(26:42):

People who've won Nobel Prizes, people who have changed the world like Hennessy and Patterson or Jack Dongarra and Nevin. People who are really on the forefront. Engage. You'll find this is a great community of brilliant people who are more than happy to engage in honest dialogue about how do we solve these tough problems? Because we do it together.

Nevin (27:02):

Yes, they're very hard problems. And what I've learned on both the C++ Committee and the Kronos SYCL Committee is that I certainly don't have all the answers, people come up with questions that I hadn't thought about at all. It makes me a better engineer and hopefully I'm contributing back as well. But we need that to make things better, that kind of dialogue.

Tony (27:22):

Well there you go, So there's our call to action everybody, get excited about it and figure out a way to contribute and hopefully you can either find us through a SYCL tech website. There's also the oneAPI.io website, and you can hear about these types of things from Nevin and James at places like Supercomputing, which we just had where they were talking.

(27:41):

So with that we'll end our podcast today, thanks to our listeners for tuning in and we'll be back next time with more Tech Talk.