**Episode 15: A Proving Ground for Open Standards**
**Host: Nicole Huesman, Intel**
**Guests: Mike Voss, Intel; Andrew Lumsdaine, Northwest Institute for Advanced Computing**

---

**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.

In earlier episodes, we've talked about the need to make it easier for developers to build code for heterogeneous environments in the face of increasingly diverse and data-intensive workloads. And the industry shift to modern C++ with the C++11 release. Today, we'll continue that conversation, exploring parallelism and heterogeneous computing from a user's perspective with:

Andrew Lumsdaine. As Chief Scientist at the Northwest Institute for Advanced Computing, Andrew wears at least two hats: Laboratory Fellow at Pacific Northwest National Lab, and Affiliate Professor in the Paul G. Allen School of Computer Science and Engineering at the University of Washington. By spanning a university and a national lab, Andrew has the opportunity to work on research questions, and then translate those results into practice. His primary research interest is High Performance Computing, with a particular attention to scalable graph algorithms. I should also note that Andrew is a member of the oneAPI Technical Advisory Board. Andrew, so great to have you with us!

**Andrew Lumsdaine:** Thank you. It's great to be here.

**Nicole Huesman:** And Mike Voss. Mike is a Principal Engineer at Intel, and was the original architect of the Threading Building Blocks flow graph API, a C++ API for expressing dependency, streaming and data flow applications. He participates in the ISO C++ standards committee and is currently evaluating the impact of the C++ executors proposals on Intel's runtimes and libraries. He is also co-author of over 40 publications on topics related to parallel programming, including the recently released book *Pro TBB: C++ Parallel Programming with Threading Building Blocks*. Welcome back to the program, Mike!

**Mike Voss:** Thanks Nicole. Happy to be here.

**Nicole Huesman:** To get us started, Andrew, can you give us your perspectives on parallelism and C++, given your deep expertise?

**Andrew Lumsdaine:** So, I started programming and really doing parallel programming in C before C++ was really in widespread use. And so, in the early days of parallel programming, that really meant high performance computing scientific codes and so forth, and the reason that people were using parallelism in that case was for performance—performance was really the most important thing. And unfortunately, in the very early days of the first attempts of trying to use C++ in that context for parallelism in high-performance, C++ got this reputation of not being the right language to use, let's say, for high performance computing. So, there was some skepticism of C++ that continued for some time. I think it's gone by now, but it was there initially.

For my part, though, as I mentioned, I started working in HPC, started programming in C, and I was also concerned about performance, but I also saw that HPC programs were not just, quote unquote, codes anymore, but were becoming large software systems. And so, they really did need to move towards a programming language like C++ that could provide the kinds of abstractions and interfaces for constructing large software systems. So, I kind of started on this long path of developing a series of libraries initially aimed at the HPC community and kind of aimed at showing that abstraction was not necessarily the enemy of performance. And along the way, I have to give credit where credit is truly due, I've had the good fortune of working with some incredible graduate students and postdocs: Jeff Squyres, Todd Veldhuizen, Jeremy Siek, Doug Gregor, Jaako Järvi, Jeremiah Willcock, people know from the C++ community, they all made contributions to these libraries and to C++ along the way. And so, starting with this first linear algebra library, that matrix template library, we evolved into sparse linear algebra, and then of course, from sparse linear algebra to graphs and then parallel graphs.

And excuse the pun, along the way, uh, you know, kind of in parallel to those efforts, we also spun off some of the tools we had developed for that into parts of the C++11 standards. So, variadic templates, lambda, and my own favorite: enable if. What was interesting and maybe challenging during that was that there was still this kind of distance between the HPC world and the parallel programming world and C++. For instance, we contributed C++ bindings to the MPI standard, but those actually have been removed since they were initially contributed. So, I kind of felt that, you know, I was trying to bridge this world between the HPC and parallel computing worlds and C++. But things evolve and change and, you know, mainstream applications started to become more ambitious and needed more performance, and as clock speeds slowed down or stalled, let's say parallelism and heterogeneity now have become essential for getting performance. And I think on the side of the HPC and parallel computing, people are realizing that they need to use a language like C++ that can provide the kinds of, again, abstractions, for building large systems. And I think by this point, there's this realization that there isn't any longer a contradiction between abstraction and performance in C++.

**Mike Voss:** So as Andrew points out, in HPC, the primary goal is performance. It's called high performance computing, right? But I think it's both this recognition that abstractions are necessary in large applications and they don't preclude good performance, but also right now, currently there's this need for heterogeneity. There's a recognition that not only do you need to take advantage of your CPU, but also these accelerators and there's this tension always. People want to write portable code because it just saves time and money. But again, the goal is performance. So, we're at this point now where we're talking about heterogeneity and how exactly do you do this in a performance portable way? And so, this is why we invite people like Andrew to participate in the [oneAPI] Technical Advisory Board because we're trying to develop DPC++ and oneAPI as a way to do heterogeneous programming in a portable way, but being really pragmatic about it.

So, we know that you can't do single source and expect the best performance on each particular target architecture. We know that if you want the absolute best performance, you're going to have to do tuning on each architecture, but people like Andrew, they have experience working through these issues of dealing with abstractions, dealing with C++ libraries, dealing with parallelism. And first of all, communicating to people that it's not always a bad thing and also are very pragmatic about the need to still tune. So, bringing people like Andrew into the discussion about oneAPI, this is very important because it helps us make sure that we are designing interfaces that achieve both goals, portability and the potential for best performance, if tuning is done.

So, I know that oneAPI has been talked a lot about in these podcasts in the past, but just to reorient people, if maybe this is their first one that they're listening to. So, when we talk about oneAPI and DPC++, both of these are standards that we're working with industry partners and academics to define.

We say DPC++, and what that means is actually a combination of things which is ISO C++ plus SYCL, which is also a standard, plus some extensions. And that is the way that we do direct programming of accelerators in oneAPI. And then oneAPI also includes some libraries for providing tuned kernels for applications. So, we have things like oneMKL for math kernels, oneDNN for deep neural networks, and so on.

So, this is really important that we're doing this all in the open. So, there's spec.oneapi.com where you can go and actually look at all the specifications. They link to the other ones like C++ and SYCL, all of these things. We're trying to do it in the open, trying to build a community around it. And that's why people like Andrew are so important to be a part of this community. I don't know, Andrew, if you want to talk at all about what the TAB is like, the Technical Advisory Board, and what you think about oneAPI and DPC++.

**Andrew Lumsdaine:** Sure. Well, so I'm going to emphasize one thing that you mentioned Mike, about portability. In some sense that word doesn't really convey the importance of portability because these days software lives much, much longer than any generation of hardware. I think of hardware, myself, almost as a disposable or consumable like paper or toner, right? It comes and goes a supercomputer is not really a capital expense because it's going to be installed and then taken away in three years or four years and a completely new architecture and the new machine will be installed, you know, whereas an application, a code, or some programs are going to live, you know, 10, 20 years and through multiple generations of hardware and even multiple paradigms, so being able to insulate programmers and codes from these changes in the underlying architectures is just essential for being able to make advances on the application side. And this in some sense was amplified greatly with the introduction of heterogeneity into the mix, so now, it wasn't just architectures and so forth might be changing, but even within a single code at one time, you might have different architectures, different instruction sets and so forth that your program needs to interact with.

**Mike Voss:** There are some common patterns though that exist across architectures, like data parallelism is important, right? Those are the things that you try to leverage, I guess, when you design some sort of long-lived portable API, right?

**Andrew Lumsdaine:** Exactly, yeah. And some of the paradigms persist and last across those. There's actually an interesting paper that Mary Shaw wrote—I think they even turned it into a small book—*Prospects for an Engineering Discipline of Software*, I think was the name of it. And she studied what happens in different disciplines in terms of how communities evolve from having different ways of trying to do the same thing, to kind of coalescing around a community, standards, folklore, to actually having standards. And once you have standards that suddenly puts in a new floor on which you can start to build the next set of things. And so I think in what we've seen in heterogeneous programming, parallel programming, is a distillation through, you know, a number of years in the community of what are common cross-cutting patterns and paradigms, as you mentioned, that can be used in programming parallel systems and programming heterogeneous systems. And so, you know, having then a standard like oneAPI, again, that can set a floor on which, you know, the next set of advances can be made is absolutely essential in our community. And so, I'm happy to participate in that.

**Mike Voss:** Another thing I think is helpful is, you know, I'm obviously coming from Intel, but what I think is useful is that oneAPI as a software ecosystem is being made available now. There was a beta and now there's gold, the gold release of oneAPI, even though some of the hardware that prompted some of the players like Intel to be involved is not all there yet. So, I think this is also helpful because like you've talked about this floor and the standards, this gives something concrete for people to start using even before maybe the particular piece of hardware that they are interested in is available. So, there's some things available now, like, I know in early November, there's DevCloud (Intel® DevCloud for oneAPI), which is the infrastructure that anybody can use to try out oneAPI. So, you can go to DevCloud and try out oneAPI on different pieces of hardware. And just earlier this month, they did actually add some nodes that can be publicly accessed that have Intel discrete graphics. So, there's nodes that actually, you can have dual Intel® Iris® X$^e$ MAX discrete graphics on. So, I think that's helpful too, right?

**Andrew Lumsdaine:** Absolutely. The term you hear sometimes in this is co-design, right, or I think maybe co-evolution or so forth that, you know, the hardware and software, but you don't necessarily want one lagging the other, so having tools available so that applications can be tested, and again, with the portability, they can be developed now and tested and used on existing platforms and then hopefully migrated in a straightforward way to the new ones. And I'm going to have to try the DevCloud with the new Iris GPUs that you mentioned. Now I'm curious to try those out.

**Mike Voss:** Yeah, so before, they were not available to the general public, but now they are. So, anyone can go to DevCloud and make a free account and just start using the discrete GPUs that are available.

So, we've talked about oneAPI as a standard and SYCL as a standard. I kind of think the end game though, is to move all this heterogeneity into C++ proper itself. Currently what's going on there is there's a lot of discussion about things like executors, schedulers, just abstractions for execution. So how do you describe the 'when' and the 'where' and 'how' of execution for, you know, the standard algorithms and just code, in general, in the standard library as well as, you know, your own code. So, I think that's probably the end game. One of the challenges, I think in the ISO C++ right now is that maybe the execution model is not quite there yet. Do you have any thoughts about that, Andrew? Well, first of all, do you agree? That's where we want to go is have it in C++ and then, you know, what do you think the challenges are?

**Andrew Lumsdaine:** No, I definitely agree. I mean, this is in some sense, it's been, you know, my goal from day one was to have some of these things baked into the standard language and I agree completely that the big issue, and this might be slightly flippant, but I think, you know, much of the C++ machine model is still a carry-over from C, which was a carry-over from BCPL. And it's basically a PDP-11, which, you know, has been fine, but obviously things have changed quite a bit in the interim. And so I think, you know, rethinking the memory model, the execution model of the environments where programs will be executing will be essential because now we have situations where there are multiple memory spaces, you have to maybe have different consistency models. And, you know, hopefully this will also lead to one of my favorite things, real thinking about how true distributed memory will be incorporated into C++. But yeah, I definitely agree. I think the end goal should be standardization. And I think the approach oneAPI is taking is, I think, a good one. This reminds me of what we did in standardizing MPI in that there was an effort to standardize message passing. And the way it came about was there were proposals and discussions about different features. The community was able to try things out as they were being proposed because there was a reference implementation available. And, you know, so it wasn't just a design and then a standardization of that design, but people could actually try it out and really decide whether certain decisions were good ones or not. And so, I think with oneAPI and having it part of the open community allows the community to try things out. I think also having, you know, the DevCloud as a platform where people can try out the aspects of heterogeneity and so forth that they might not be able to on their own machines also helps that effort quite a bit. All the pieces are coming together in the right way, I think, in this approach.

**Mike Voss:** I know you've been working on proposals related to graph libraries and C++ recently. Does this play it all together? So, with heterogeneity and the execution of the graph algorithms and executors and all that, what do you see there?

**Andrew Lumsdaine:** Yes. Before standardization, there was, of course, the Boost library effort and that, in some sense, was an attempt to try to do what I just mentioned, provide a testing ground of different library ideas in particular, prior to standardization. But since our initial attempt at a library, the Boost graph library, we've evolved that into something much more modern, particularly after C++11—I think it was really in many ways, a new language compared to C++ prior to that. And so, with our new proposal, the proposed standard graph library, we're also including parallel execution. So, the algorithms would take an execution policy similar to how standard library algorithms do right now. In fact, the libraries build as much as we can on top of standard library algorithms. But at the same time, we're realizing that the environment in the future where one would want to execute graph algorithms is going to be this much richer ecosystem of heterogeneous processing units and maybe distributed memory and so forth. So, we're watching executors and other parts of the standard closely as we're preparing the proposal for the graph library standardization.

**Mike Voss:** So as a member of the Technical Advisory Board for oneAPI, any things you want to see in terms of the direction of oneAPI?

**Andrew Lumsdaine:** No, I'm actually pretty happy with what I see there. Of course, you know, it might actually be interesting to maybe incubate some of the graph library things there, you know, as it's moving through standardization. But one thing I hope to see happen, Mike, is that when we were doing some of these parallel versions of our graph algorithms, we did find that the parallel mechanisms currently in C++ were really not rich enough for what we needed to be able to do. So, for instance, we needed a much finer control over partitioning of parallel, just a four, for instance. And so, we ended up using TBB [Threading Building Blocks] in several cases directly, and TBB might have even been ahead of the game in some ways from where the execution policies are now in C++ because they really were based on ranges rather than iterator pairs. And so, I would hope to see also kind of come out of oneAPI and its evolution into C++ would be the range-based notions of parallelization that were in TBB as well as concurrent containers.

**Mike Voss:** Yeah, so, oneAPI is DPC++, which is C++ plus SYCL and extensions, and then also libraries. And one of the libraries that we include in oneAPI is oneTBB because we do recognize that on the host, when you're expressing threading, there are still things missing from standard C++ and SYCL for controlling and expressing, more generally, parallelism. And you point out some of those, like, how you partition things. That's actually why oneTBB is part of oneAPI currently. And I agree with you, the hope is that working with partners in these communities, we can actually evolve both C++ and SYCL to add the necessary features for addressing those what currently might be limitations.

**Andrew Lumsdaine:** And I think one thing that, again, this might not be part of the API per se, but one thing that is always helpful in some of these exploratory co-development and so forth are diagnostics and profiling tools and so forth. So, I know part of oneAPI is the DPC++ compiler. Do you have plans to have a DPVTune?

**Mike Voss:** So, oneAPI has two faces. There's the specification, you know, we welcome people to contribute to the specification, to implement parts, if they want. And then there's also our implementation of the standard. So, we have products, for example, there's the oneAPI Base Toolkit from Intel that is an implementation of oneAPI that follows the standard. And also, it includes some tools like VTune and Advisor. So VTune, which used to be just about CPUs, now there are extensions where you can analyze the performance on your GPU. Advisor, which is a tool for helping you identify opportunities within your code for parallelism or tuning. There's now Offload Advisor that helps you decide if a piece of code would be profitable to target at an accelerator. So, we do have these tools. So far, those tools are not part of the oneAPI specification, but they are part of our Base Toolkit, which we provide. And the Base Toolkit is also free for those who are interested in looking at it. So yeah, as an implementation, we're definitely doing those things.

**Andrew Lumsdaine:** No, that's good. You reminded me of the difference between a specification and an implementation. That was certainly, in MPI, that was always an important distinction to make. And I'm glad there is a high-quality implementation, as we used to say about some of them in MPI, for oneAPI. So, I'll definitely take a deeper look at some of the tools that are available. I've just been focusing on oneTBB and on DPC++.

**Mike Voss:** So, one advantage is SYCL, for example, there are other implementations. So CodePlay has an implementation. There's triSYCL. There's hipSYCL. So, there are other implementations which are tuned for different platforms and tuned for different purposes. Like Codeplay's version. They really are focused on embedded systems. Intel's implementation, although it's useful in many places, we put a lot of focus on HPC and high performance and GPU and CPU kinds of things. So, it is really good to have these standards because you can get different flavors of implementations with different quality of implementations for different targets. So that's an important part of the ecosystem too.

**Nicole Huesman:** So, we're having a fantastic discussion here. I think you guys could continue to chat. I'd love to have you back on the program. As we wrap up, all of these—C++, SYCL, DPC++, oneAPI—they're all evolving based on such vital community and ecosystem input. For those listeners who want to dive in and participate, contribute, and help shape these, how can they get started?

**Mike Voss:** I would recommend going to spec.oneapi.com. I was thinking about this before the podcast and I was starting to come up with a list of all sorts of places to go, but actually spec.oneapi.com, because oneAPI is a collection of standard things plus extensions, that page gets you links to everything. It links you to the SYCL standards, the provisional SYCL 2020 standard. It gives you the right links to the ISO C++ standard as well as all the extensions and the libraries that are included in oneAPI. So, I would suggest that as a starting point.

And then actually involvement in the communities, probably Andrew can comment ‘cause he's involved in C++, in the TAB, and so he probably is better at explaining how to get involved in those things.

**Andrew Lumsdaine:** Well, in the ‘before’ times, I would have said, just come to the next C++ in-person meeting and just show up and go to the sessions that seem interesting and start contributing because there's no membership fee to just contribute and so forth. That's a little more complicated now, of course, because the meetings tend to now all take place online. But I think if you start with the pointer to the ISO standard, there are pointers to the different working groups and ways to join the appropriate mailing lists. And C++ goes in cycles aimed at three years. And there are different working groups trying to move different libraries and different language features through. So, joining that would really mean joining one of the working groups and participating in their mailing list, wikis, and the online meetings, and those vary depending on the particular group. So, the best way to start is where Mike mentioned, and then following the C++ links, and then to the particular working groups that you might be interested in.

**Mike Voss:** And one additional comment I'd make about the ISO C++ groups are that, if you attend either physically or virtually, the meetings sometimes are straw polls where they try to get the feeling of the room of what people think on certain topics, but there's always the option to not vote. So, if you're new to it and you're worried that you'll feel intimidated or that you won't know what's going on, many people don't vote. So, they'll come up to a topic and say, here's what we're voting on, are you for or against, and so on. And I think it's very, very rare that everyone in the room actually votes. It's not something intimidating where you feel like you have to know what's going on for your first meetings or for any particular topic. You can just attend and be in the room and listen in to what's going on.

**Nicole Huesman:** So, we've talked about the SYCL ecosystem and how much it's growing and thriving with its different implementations—ComputeCpp, hipSYCL, triSYCL, DPC++. Can you give our listeners a pointer as to how to get involved in that community?

**Mike Voss:** So that is in Khronos. So, you can go to the Khronos organization. They have a number of standards that they work on. Just like ISO C++, Khronos and the SYCL standard are very open. Anyone can go to those as well and participate.

**Nicole Huesman:** And then I also wanted to mention, the two of you touched on DevCloud quite a bit, the link for that, and we'll also have all of these links available for listeners, the link for that is devcloud.intel.com/oneAPI.

So, with that, so wonderful to have you both on the program. Andrew, thanks so much for sharing your insights with us.

**Andrew Lumsdaine:** Oh, thank you. It was a blast.

**Nicole Huesman:** And Mike. So great to have you back on the program.

**Mike Voss:** Thank you. It was a pleasure.

---

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