Episode 11: In Pursuit of the Holy Grail: Portable, Performant Programming
Host: Nicole Huesman, Intel
Guests: James Brodman, Intel; Tom Deakin, University of Bristol

---

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.

On our path to exascale, the next generation of supercomputers all have *at least* one thing in common: they will all contain a variety of hardware architectures. This is not only true in the world of supercomputers. It is increasingly true across the computing landscape. The ability to efficiently program for these environments is key and the need for performance portability is a hot, and as we'll see, *tricky* topic.

Today's guests *live* at this forefront.

Dr. Tom Deakin is a senior research associate and lecturer in the High-Performance Computing Research Group at the University of Bristol, and a contributor to the Khronos SYCL Working Group. He's well-recognized in the topic of performance portability, and we're thrilled to have him join us.

Tom Deakin: Yeah, it's great to be here. Thanks for having me.

Nicole Huesman: And Dr. James Brodman is a software engineer at Intel, and also a contributor to the Khronos SYCL Working Group. He focuses on languages and compilers for parallel programming. James, so great to have you on today's program.

James Brodman: Thank you for having me.

Nicole Huesman: The term 'performance portability' seems to mean different things to different people. And there's often this tension between performance and portability. Let's start there. I'll let the two of you dive in.

James Brodman: So, what are your thoughts, Tom?

Tom Deakin: Well, I think performance portability is often a very contentious subject. And part of this comes from, I think, what it really means for something to be performance portable, and the fact that what developers are willing to accept as performance portable. Part of this, of course, is because portability is a requirement for performance portability. You have to write code that is portable. It has to at least run on all the platforms that you might care about. And then you can start thinking about what the performance is like on each of those platforms and the degree of performance you're willing to accept for some degree of portability, I think, can mean this definition is different for different people.

James Brodman: Yeah, I absolutely agree. It seems like the Holy Grail, of course, would be to be able to write one code that would run everywhere and get the absolute best performance, but we're not quite there yet, I think. So, there's this trade-off everyone has to make between how much common code they want to have, and how much code they're willing to have that's specific to whatever device or system they're targeting.

Tom Deakin: Yeah, exactly. And hopefully the specialism you have to do is only to get the last few percent that you care about. That's the goal. But you know, there is some pragmatism there. You have to allow for some specialization to go that last few yards, if you need to.

James Brodman: For sure. And that's what keeps everyone involved in HPC gainfully employed!

Nicole Huesman: So, the research group that you're a part of at University of Bristol has done some fantastic work around how you define performance portability. Can you talk a little bit about that?

Tom Deakin: So, for us in Bristol in the HPC Research Group, we've been publishing on performance portability for a number of years now, and one definition that we use—although there are lots of sort of hand-wavy concepts in it—is to say a code is performance portable where it achieves a similar level of performance efficiency across all the platforms that you care about. Now we're in HPC, so that's high performance, so we hope that that efficiency is high. We're hoping to get, say 80% of the peak performance that is possible on those architectures, but it's this idea of high performance and high, consistent performance that we'd look for when trying to measure performance portability.

Nicole Huesman: So, James, you mentioned there's this Holy Grail. What role do open standards play in that?

James Brodman: So I think open standards are very important here because when you're talking about defining portable code, you need everyone to agree on what they'll support on various systems. And if everyone has their preferred dialect of the way to program their own system, then your portability is going to drop because one dialect may not be supported in another platform. But when you have an open standard and everyone agrees to implement that standard on their system, then that gives you a common language and set of tools to use when writing your programs.

Tom Deakin: Yeah, I think that's really important. You need to remember that this is performance portability, and portability has to come first. So if you write your program in a language that is only supported by one vendor, then you're going to have a really hard job convincing another vendor to make that run on their platform too. So with something like an open standard, it provides this, you know, fair playground for all the vendors to come together and help make something good for everybody. And if you write your code in that, then you can ask for it for when you buy systems, and you can be sure that there's going to be a healthy ecosystem around that with lots of choices of compilers and tools and people to help you out. And in that way, you can have a code that will be portable, and hopefully, as a flexible model that allows you to write something that's performant everywhere as well.

James Brodman: Exactly. And open standards have come in many flavors over the years, the C language or the C++ languages are open standards and you can get a C or C++ compiler for pretty much any system these days. But those languages alone aren't necessarily going to get you all the performance that you need on your systems. So you also need standards that have these higher-level abstractions to enable use of all the different parallel components in today's systems. Things like SIMD units, multiple cores, accelerators, and things like that. So that's where something like the SYCL standard can come in and provide these higher-level abstractions that still give you the portability, but also start giving you hooks into all the performance that modern computing systems have today.

Tom Deakin: Yeah, that's right. These open standards like SYCL and OpenCL, which SYCL was inspired by, give you this abstraction over what an HPC device, what a computing thing, looks like. It gives you these building blocks that are common across different architectures. So even down at the low level of abstraction, you have this portability, but then something like SYCL allows you to target those because they appear in the same language in the same way between vendors. You can write your code, targeting those particular features, and know that there'll be there and map to the sensible thing on the underlying hardware.

James Brodman: Yes. And it's this expression of these higher-level abstractions and common parallel programming patterns that have kind of guided the evolution of these standards over the years.

In particular, the latest version of SYCL, the 2020 provisional specification, added even more of these higher-level parallel building blocks, things like support for reductions or other kind of higher-level collective operations, which enables implementations of these to provide efficient versions of them across different targets.

Tom Deakin: Yeah, these parallel patents are an important concept. For those of you who haven't read Tim Mattson's book on that subject, it's kind of a standard text that you should check out. But this idea that you want to paralyze a loop and all the iterations of that loop are independent. You just want a programming model that can represent that level of abstraction. That's a portable concept between the different architectures. So something like SYCL with those high-level abstractions allows you to express that. But if you need to kind of break it down and start worrying about the things under the hood, then there is this hierarchy of abstraction that SYCL will give you that enables you to go deeper if you need to.

James Brodman: Yes, definitely. Providing support for all these higher-level patterns is very, very useful. But since we're realists living in the real world, and if you have a large machine, and you do want to be able to get the best out of it, you have to find this trade-off between using these higher-level abstractions and having the ability to dive really deep down when you need to at the cost of potentially some portability.

So I think the biggest thing that the SYCL Working Group is focused on these days is finishing the next version of the SYCL spec. We released the 2020 provisional specification. In some sense, it's a beta spec that provides a public guideline of how we're thinking and where we're looking to go. And so we've been addressing a lot of feedback from those issues and trying to fine-tune everything and make sure that the next version of the final specification is going to be a really great release with a lot of important new features.

A lot of us come from the kind of implementation point of view, where we're working on building implementations of SYCL, but it's very important in doing all of this to get the perspective of the people who are actually going to be using it. And Tom represents the users here, and has provided a lot of valuable insight into what matters for the people using the specification, this language, at the end of the day.

Tom Deakin: And I think you're right there in that the main focus of the SYCL Working Group right now is to make big strides from SYCL 1.2.1 into SYCL 2020. There's a lot of new features. All of them are really helpful in helping us write code both quickly for us as developers, but also in a way that allows us to write performance portable things. So like the reductions, for example. There's many ways we might want to implement a reduction, and that will be different on different architectures that you're targeting. But as developers, we just want to say, do a reduction for us. So this is the kind of interaction that happens within a working group. There's a feature that we'd like and implementers and users can all start kind of discussing around this and figuring out the best way to write this down. Another positive of the SYCL Working Group is this mixture of users and implementers and vendors as well. So there's people that make hardware, people that make compilers and run times, there's people that write libraries that sit on top of programming models, such as SYCL, and then there's users like me, that will go and write SYCL directly.

James Brodman: Yeah, like you mentioned, in the working group, there's this great mix of people. And in particular, SYCL has really been developing over the last year in that there are several different implementations now that target pretty much all the major hardware vendors out there. You can write a SYCL program today and run that on Intel hardware, on NVIDIA hardware, AMD hardware, Xilinx hardware, ARM hardware. And it's really kind of exciting because you actually can now write a single program that can run across a great many different types of systems. So we're actually starting to see the portability promise delivered in practice.

Tom Deakin: Yeah, a lot of my work has been involved in testing out that ecosystem, in taking a code that I've written in SYCL and trying all the different implementations that are there, and seeing what performance is possible on all the different architectures that are supported. And with this mixture of implementations, there's also more than one path from a SYCL application to a particular architecture. So if you're going to be running on a particular GPU, say from one vendor, there may be two or three of those implementations that support that particular device. So that means that we can then start trying these things and seeing which one's going to work best for us. This is something we're very much used to even just regular programming that we have a choice of compiler. There's always more than one compiler that we can use to build our C++ program. So having this choice of implementation as well, really kind of shows this strong and healthy and growing ecosystem around SYCL.

James Brodman: So how have you been evaluating this? Do you have a set of applications that you've been using to kind of test on all the different platforms?

Tom Deakin: Yeah, so there's a number of mini apps that we use. And these mini apps are sort of a distillation of much bigger codes, and they really capture the computational and memory access patterns of these larger codes, but they're in a much more agile state that allow you to try them out on different systems. So we'll take these applications and we'll maybe port them to SYCL, and then we will get them running on as many different systems as we can. So I have a paper coming out, for a workshop at SC [Supercomputing 2020], the P3HPC Workshop. This paper takes a couple of applications. We look at 15 different architectures and we take a number of programming models, SYCL included, and we try to get those codes running in each programming model on all of those 15 different architectures and measure the performance efficiency that we can achieve. And then we can take those efficiencies and start trying to understand the performance portability of that data set that we've just collected. How consistent are we? How close to peak performance do we get across our set of architectures?

James Brodman: Oh, that sounds very interesting. Can you give us a teaser as to what you've seen or should we wait for the workshop?

Tom Deakin: Well, obviously you should sign up for the workshop, but as a teaser, what we're finding is that the open standard programming models are doing really well. They help with the portability side of things. This helps us get the coverage across the 15 platforms. SYCL is definitely growing and there's some room for improvement on some of the architectures, but some of the models that have had this ecosystem growing for a little bit longer, like OpenMP is showing really nice results as well.

James Brodman: Are these codes that anyone can just go check out, try on their system, if they want?

Tom Deakin: Yeah. So the codes are all open source, as are the scripts that we use to download, build and run those codes on all the different platforms. So it's no mean feat taking a code and getting it to run on 15 different architectures. So we capture our workflow of how we've done this, and we make those available as well.

James Brodman: Very cool. It's definitely useful to have tools and infrastructure like that available for people to try it out themselves.

Nicole Huesman: To that point, if I haven't programmed using SYCL yet, how easy or difficult is it for me to get started?

James Brodman: I think it's pretty easy, although I might be a little biased. In particular, if you get started with the latest version of SYCL—SYCL 2020—the working group made a lot of improvements to really make SYCL approachable to people that haven't done it. SYCL is based on modern C++, so if you're not familiar with C++, there might be a slight learning curve there, but hopefully not one that's insurmountable. Over the last year, there's been a lot of effort on the parts of multiple people in the community to start putting together really high-quality training materials and introductory materials for SYCL. In particular, Codeplay, another implementer of SYCL, has been very active in this field. They have this website, SYCL.tech that has lots of tutorials, presentations, links to talks available for people. Intel, as well, has put together a lot of training material. A lot of it is available as part of the Intel® DevCloud. If you sign up for that, which I believe is free and has lots of different hardware you can try out, there are a set of training modules based on Jupyter notebooks. So it's an interactive session available in your browser that will kind of guide you through learning all the different components of a SYCL application, and the things you need to know to really get started.

Tom, what's been your experience getting up to speed and writing applications with SYCL?

Tom Deakin: Well, the first thing to remember is that writing parallel programs is hard. Finding parallelism in programs is a tricky thing to do. And actually this is true, no matter what language you might write in. So, taking a vanilla serial code that doesn't have any parallelism in, and getting it to run in parallel, and updating it so that it's possible to be correct when running in parallel, those are challenges that we all face, no matter what we write it in.

The next stage is then, how do we express that in parallelism? And we've found something like SYCL is pretty straightforward to actually express the parallelism in once you've found it. You have these very high levels of abstraction. You can just say parallel four, and you give it the iteration space that you'd like to run your loop over in parallel, and that's it. You're pretty much away at that point.

There's much more flexibility built into SYCL, if you need it. You can express that parallelism on your hardware in a kind of hierarchical manner, if you like. But also, you can then start using the tasking model that's built into SYCL as well to express more complicated dependencies between those parallel loops. So, there's a lot of flexibility to delve in once you need it. But actually writing a parallel program in SYCL to start with, we've found pretty straightforward.

So for those mini apps in our studies, we find it typically takes someone a couple of weeks to port from one program to another, SYCL included.

Nicole Huesman: Let's shift the conversation a bit and talk about what's next. What are you both looking forward to?

Tom Deakin: Well, for me, the thing I'm looking forward to is watching the ecosystem around SYCL continue to grow. Our work, where we've been exploring it as much as we can, is showing that the support across different vendors is certainly growing. And the compilers and tools are becoming much more mature and able to give us that portability across platforms that is important as a first step. The results do show that there is some more work to do. And on some platforms, we do see there is some improvements in terms of the performance that we can see. So that's what I'm looking forward to next.

The other big important thing for me is the release of the next specification, the next SYCL 2020. This is a big step in what SYCL is going to look like going forward. And the working group is working hard to make sure that it looks right and it looks good and is going to do the right job for what we need.

James Brodman: Yeah, I absolutely agree. I'm really looking forward to SYCL 2020 being finished and released into the world because I think it's really going to improve a lot of things for a lot of users.

I'm also looking forward to the growing ecosystem. We have multiple implementations of SYCL now, and each of them have been getting more and more mature. Probably the most recent up-and-comer is hipSYCL out of Heidelberg. And recently Intel and Heidelberg launched this joint Center of Excellence, which I find rather exciting because hipSYCL has been rapidly iterating and is now adding support for a lot of the new features defined in SYCL 2020. So it's going to be nice to see those features available on many different platforms.

Nicole Huesman: It has been so wonderful to have both of you here today, and so exciting to see how the SYCL community is coming together to advance portability in parallel programming. I know Tom, you mentioned the P3HPC Workshop that certainly I'm looking forward to. I know James you're intimately involved in a workshop at Supercomputing [Supercomputing 2020] as well. There are so many exciting things happening at Supercomputing this year. So with that, where can listeners go to learn more about all of the exciting developments?

James Brodman: So you've already mentioned the P3HPC Workshop. I think that's going to be very, very interesting this year. There are several SYCL focused activities that SC [Supercomputing 2020] this year as well. There are two tutorials that will cover various aspects of SYCL. There is also going to be a Birds of a Feather (BoF) session about SYCL and heterogeneous C++. Intel's implementation of SYCL, the Data Parallel C++ compiler, has been developed as an open source project, and that's available on GitHub for anyone who wants to check it out, or even send a pull request if you want to add support for a new system or implement a new feature. oneapi.com is always a great resource. SYCL.tech is also a very good resource for learning more about SYCL.

Nicole Huesman: And Tom, where can listeners go to learn more about what you're doing at University of Bristol?

Tom Deakin: Yeah, so all of my work you can find on my website, hpc.tomdeakin.com. And you can find links to the research group on GitHub there—that's uob-hpc.github.io—that has links to all the source code and our benchmark repository, where you can see how to run all of our applications.

Nicole Huesman: Tom, thanks so much for being here today to share your insights with us.

Tom Deakin: Thanks for having me.

Nicole Huesman: And James, so great to have you on today's program.

James Brodman: Thanks Nicole, and thanks Tom.

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