Episode 18: Porting GROMACS Across Heterogeneous Architectures
Host: Nicole Huesman, Intel
Guests: Erik Lindahl, Stockholm University & KTH Royal Institute of Technology; Andrey Alekseenko, KTH Royal Institute of Technology & SciLifeLab; Roland Schulz, Intel

---

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.

Earlier, we talked to Erik Lindahl and Roland Schulz about how GROMACS, one of the world's most widely used open source molecular dynamics applications, is helping provide insight into the world around us, and the computational challenges and immense benefits it brings.

Today, we're picking this discussion back up and talking about porting GROMACS across heterogeneous architectures.

Erik Lindahl is a biophysics professor at Stockholm University and KTH Royal Institute of Technology. He's been instrumental in the development of GROMACS. Welcome back to the program, Erik.

Erik Lindahl: Great to be here again, Nicole.

Nicole Huesman: And Roland Schulz is a parallel software engineer at Intel. He's worked on GROMACS for over a decade, contributing primarily to parallel efficiency improvements and code modernization. Great to have you with us again, Roland.

Roland Schulz: Thanks for having me.

Nicole Huesman: Andrey Alekseenko also joins the discussion. Andrey is a postdoctoral fellow in the Department of Applied Physics at the KTH Royal Institute of Technology and SciLifeLab. He's done much of the work we'll talk about today. Andrey, welcome!

Andrey Alekseenko: Thank you for inviting me.

Nicole Huesman: Erik, when we last talked, you made an intriguing statement: "Exascale is not a goal, it's the next milestone that we're gonna fly right by." Why don't super computers solve everything?

Erik Lindahl: Oh, that's a good question. The supercomputers are great, but they just provide real power, right? And as expensive as they are, the one thing that's more precious is human brains. And I guess Alexa is a great example of that. We need smart people to solve the problems.

Roland Schulz: We started to talk last time about the different programming languages. What's your view on the status of the programming languages we have available today and how potentially oneAPI is the next step in this development?

Erik Lindahl: I would say that they're adequate. They're not perfect, but I'm not sure whether perfect is the goal here, right? That's as a researcher, it's easy to think that, 'Oh, all I would really like is a single threaded program, like old-fashioned Fortran code, and then it should be up to the vendor and the compiler makers to make my code magically go fast. But as we all in this discussion know, that's not magically going to happen. There are some fundamental limits of physics here. Full disclosure: I hate programming most of these languages, but the problem is that that's not really the language I'm hating, right? What I'm hating is that the fundamental limits that physics is setting here, that there are latencies to worry about and everything, and these languages at least expose those limits, so smart people like Andrey here can try to work around them. For that reason, I would say that radically, I don't like them, but it's the limitations we have to live with.

Andrey Alekseenko: So, languages can hide the hardware details for some users, but eventually all the abstractions start leaking. If you really want to push the hardware to its limits, you have to go low level. You have to really dig into the operations of the hardware. And I guess here, it's not that much about the language, but about the hardware and whether the language allows you to work with it, not about the syntax sugar or anything like that, although that is certainly nice to have.

Erik Lindahl: I think a great example is, Roland, you asked specifically about oneAPI, and even things like accelerators, GPUs, you name it, they too evolve. The latest trend we're starting to see now is that we're actually increasingly going to have devices with multiple chips, just as we got multiple chips in CPUs two decades ago, and that's of course a pain because none of us have the algorithms to make this work. But suddenly we’re starting to have APIs where it’s at least possible to express some of these things. It will actually take a year or two before most of our programs can really fully exploit it, but short term, it's a pain. Long-term, the only thing that's worse than that pain would be having APIs where I can't express this because then I can't make my code foster.

Roland Schulz: Andrey has said, to get good performance, one has to program towards the hardware, and I completely agree. But on the other hand, GROMACS is over a million lines of code and we can't write every line of code for every new hardware coming out. And we're doing a lot of work in GROMACS to avoid having too much code specific to each individual hardware. What do you think is a good way of doing both portability as much as possible while getting performance?

Andrey Alekseenko: That's a really good question. And for the current release cycle, a big goal for us is to unify where possible the divergent code branches for different architectures. So, I don't think there is an easy and straightforward answer here. Of course, on one hand, we want to have as unified a code base as possible, as few device-specific workarounds and branches as possible. On the other hand, just writing a single code and hoping it will work magically on all devices and achieve the highest possible performance probably is a bit too much. So, it is very important to strike a balance here, how much we can maintain, how much effort we can put into optimizing for a specific hardware, because of course, there is also the limitation of the human resources available to us. You cannot manually tune for every single device in the market.

Erik Lindahl: No, and it's, I know a few years ago when we got Mark Abraham, and you were around then too, Roland, to join the team that I think we've over the years that, in academia, we always see codes as assets, right? When we're professional code developers, they've learned to see code as a liability. And I think we're gradually getting there, but it's hard. Any vendor or skilled student or something that comes up with something that speeds things up by 2%, by itself, 2% is awesome performance, but then we have to realize that, well, somebody will have to maintain this and we're going to have to debug this long-term for three or four or five different architectures, and the really hard part is saying no to something that, in principle, is good, but it's going to cost too much long-term.

Roland Schulz: Because we've talked about the language features, how much do the language features matter for this? So currently, we have OpenCL as one of the code branches for GPU support and it's C-based. And then we have support for CPU where we're not using the SPMD programming model, but we're doing explicit SIMD programming. How much do you think we can in the future share code by having one programming model which works across different CPU architectures?

Erik Lindahl: That's a great question. I'm actually looking forward to hear what Andrey thinks about this because he doesn't have all the burden of 20 years of looking at this code like I do! Part of our interest in SYCL is … this sounds horrible to get rid of OpenCL, right? … but C++ comes with many advantages. It allows us to organize code at a higher level. It allows us to abstract things away. So, our plan right now is by introducing SYCL as the portable layer, we can gradually retire OpenCL and have native C++ everywhere. But your challenge is even greater. Will we have a future where we can have a common code base for CPU and accelerators? I don't know. I think short-term, it's going to be difficult, but I would love to be proven wrong there.

Andrey Alekseenko: Yes. I don't believe we're going to see a single code base for CPU and GPU in the near future. Even right now, one can run OpenCL on CPUs, but usually the programming model is not a good fit for CPUs. So while it is possible to have a common OpenCL or better SYCL code base because yes, in our case, supporting OpenCL is quite a big pain, to be honest, not only because it's plain C, but because it's a small island of C surrounded by a sea of C++. So, we have to keep in mind the interoperability of the, say, headers that are included in both C++ and C code base.

Erik Lindahl: Yeah. And I would expand on that by saying that maybe we're asking the wrong question, right? Of course, we can make a SYCL code run well on a CPU, but that's inherently a SYCL type of code. What many of us—and in particular, Roland has been active in this—juicing CPU to the maximum extent possible. And […] the CPU is that I can execute a single instruction and then have the results just like five clock cycles later. And this has allowed us to write exceptionally fine-tuned code for things that are extremely latency sensitive. It is a limitation of SYCL in the sense that, the whole accelerator model is that we're paying in latency for getting throughput, and that using that type of programming API for code that we've been able to explore at very short latencies, I think that's difficult. And it's probably even the wrong optimization to do.

I expect that we will see a convergence where, what we today call CPU, just as we have a floating-point unit in them now, we might very well have a throughput-focused set of execution units in them in 10 years. And at that point, who knows, we might have some sort of SYCL programming environment where we use some slightly different programming constructs for latency- versus throughput-focused things. But my conscious of programs in general should start to think about is this particular part of my algorithm latency or throughput sensitive because neither the compiler nor the API will solve that magic for you.

Roland Schulz: Yes, very good point. So, I think historically the GROMACS project always tries to base things as much as possible on open standards, like whether it's languages like C++ or OpenMP and so on. Why is it currently not possible to just use plain C++? Why are extra languages needed for GPU programming? And what do you think is the future path towards having GPU support in just C++?

Erik Lindahl: I think those are two very key differences. The first thing, it would be an awesome world if we could just write our algorithms, forget about C++. What if I could write my algorithms as equations and then just magically have a computer solve everything. I think we've tried that, including Intel, a couple of times! It usually didn't work out particularly well. The reason for expressing the limitations of hardware with that also comes the possibility of using the special properties of the hardware, right? That it matters occasionally whether I have the SIMD units, that's four, eight or 16 units wide. And if I cannot just compile an existing algorithm, but if I can tune my algorithm to benefit from, say, operating on 4x4 units, that of course means that I can exploit that. But if I don't change my algorithm, it would have to be an exceptionally intelligent compiler to not just change the implementation, but completely change the algorithm to fit the hardware better, and I just don't see that ever happening.

Roland Schulz: I agree. My view on it is that, while there are interesting new developments happening or proposed for the C++ standard, it's still quite a few years out until we have all of those features you just mentioned or the capability to express those things you said we need to be able to express in just ISO C++.

Erik Lindahl: Yeah, but there will of course be many simple things like sorting or obvious processes we do in STL containers. They will obviously be GPU-accelerated there as they are in other places in the standard, there is little specific code as possible, but things that are very algorithm-specific ... I would so love to be proven wrong here, but I just don't see it happening. This would kind of be like me writing a bubble sort, and then the compiler changing that to a quicksort, not just by linking to a different library function, but by recognizing the algorithm. And I can't imagine compilers being that smart in my lifetime.

Roland Schulz: Yes. And as you mentioned, the trend is going towards more hardware diversity and all those factors you just mentioned mean that we will need to even more have different algorithms to run well on those diverse set of hardware, correct?

Erik Lindahl: Yes. But in many cases I would argue we don't need completely different algorithms, but we might need algorithms with a few tuneables, right? So that's, for instance, if I run on a completely scalar architecture, it's likely best to calculate a 1x1 interactions at the time. If there is some sort of fundamental 4-way SIMD, it's likely better to do that 4x4, but in principle, it's the same algorithm. It's just that we have some sort of tuneable parameter here and that's at least for parts of GROMACS I think we've been able to achieve that, that these are not four different directors with completely orthogonal code. It's one directory, but depending on the hardware, we end up using slightly different versions of the code.

Andrey Alekseenko: I totally agree. For example, right now in GROMACS, GPU offload supports most devices, it supports NVIDIA, Intel and AMD GPUs, and it is mostly the same code with a few vendor-specific blocks here and there. However, to get good performance on NVIDIA, we have to use CUDA primarily for newer devices. So, we have to maintain an additional copy of mostly the same code in CUDA. So, while there is a variant of hardware, the algorithms are mostly identical, with a few knobs or a few blocks here and there that are vendor-specific. But the zoo of different platforms, which are vendor-specific, make maintaining the support for these different hardware quite difficult.

Erik Lindahl: Roland, you spoke about the importance that we should have standards, right? Since a couple of hours ago, SYCL is now a standard from the Khronos Group.

Roland Schulz: Yeah, exactly. Today, as of this recording, the new version of the SYCL 2020 standard was released. And given that it was released today, you probably haven't had time to look in detail at this, but the provisional has been out before. Are there any specific things you're excited for the new release of the standard?

Erik Lindahl: Well, as you said, we've been able to monitor the process from the inside. But if you look over the last year, I like the fact how this entire interaction and bringing in lots of developers has helped. We've been able to shave off a few warts, some things like avoiding having too many things like naming templates and kernels and everything. It's certainly an easier language to use than it was a year ago!

Roland Schulz: When it comes to alternative programming models, the ones already mentioned, and also the Pragma-based ones, why did you choose to use SYCL to target Intel GPUs?

Erik Lindahl: You mean compared to OpenCL?

Roland Schulz: Compared to both Pragma-based, so using OpenMP offload, as well as the other alternatives available, yes.

Erik Lindahl: So, if we compare to OpenMP offloads are, in theory, interesting. For many users, I would encourage them to start with OpenMP because it's simple, it's portable and it will work everywhere, right? Our curse with GROMACS is that we spent so many years optimizing that, so we know a whole lot about how to explore details of hardware to get good performance. And with a language like SYCL, we can dig deeper here. Once you've hit your head against brick walls too many times and you just can't push OpenMP further, that's probably the stage where you should start using SYCL. This might sound strange, but don't immediately jump to these languages unless you absolutely need them, right? Unfortunately, over the years we have accepted that we do need them. So again, writing things with explicit instructions targeting hardware is a necessary evil, not something you should do just because it's fun.

Roland Schulz: How was the experience like so far? The DPC++ release is still the first release, so as with all new software, there are still some initial hurdles, but Andrey, how was the experience so far porting to SYCL? And how did you go about doing that? I know you did a lot of refactoring at the beginning. Why did you choose to do that?

Andrey Alekseenko: Most of the refactoring was related to things that our code base is large and has a lot of history. Right now, we have some code duplicated between CUDA and OpenCL backends, and a new backend will require a third copy of essentially the same code. The task of unifying all these code branches hasn't been finished. It was planned to be finished in 2021 release. Unfortunately, it was not due to time constraints of the year ending, but it was partially done, and it was a great help because the less code duplications there is, the easier the overall product is to maintain.

Regarding SYCL itself, the experience was pretty positive. As Roland mentioned, there were some pains related to the novelty of the whole toolset, the early versions of standards, and so on. But that was to be expected.

Erik Lindahl: Yeah, I was, I've been very positively surprised with this. All these languages have bugs. I don't worry so much about bugs, to tell the truth, but that we're gradually seeing a community with people exchanging information, right? That's usually how we find our ways around not necessarily bugs, but even limitations that something is just slow, has anybody found a faster way to do that? Then of course, SYCL is nowhere near as mature as CUDA. That would be a horribly unfair comparison to make given that there's almost 20 years of difference and developments that we are seeing a budding community here. So, you're not going to be all alone if you're developing SYCL code today.

Andrey Alekseenko: And what's particularly nice is that, not only the community of SYCL developers is growing, but also the number of backends we see in active development, not only C++, but also implementations like ComputeCpp or hipSYCL. So, this standard itself is getting widely adopted. And your support, it's no longer a vendor-specific thing because it's actually a community-wide standard.

Erik Lindahl: Yeah, this might sound strange in a podcast organized by Intel, but I would argue this is potentially the strongest side of SYCL or DPC++ or whatever you call it that, I love the example that happened about a week ago, that NERSC are now contracting with ComputeCpp to make sure that SYCL will actually work on Perlmutter […], but as developers, of course, that means that you can start writing your code. And I bet not even NERSC knows what architecture their next system after Perlmutter is going to have. But I bet that the SYCL code is going to run on it. And that has never, ever been the case as a developer before, that you can trust that your code will run on the next-generation accelerator too. GROMACS is in some ways an outlier, right? Because it's very difficult to take performance away from people when you've given it to them. That's also why we can't just switch to SYCL overnight, but we have several of these other projects on the team and new acceleration efforts we're starting, [and for these] then I will increasingly just use SYCL. It might not give me a hundred percent everywhere, but I would get 90% everywhere as a gigantic savings in development efforts.

Roland Schulz: So Andrey, I know the preview release of the SYCL version, which came out just a few days after the new GROMACS 2021 release is out, where can people find it and what can people expect to be in that release and where can they get more information?

Andrey Alekseenko: The version 2021 SYCL was released on February 5, a few days from the main 2021 release, and can be downloaded on our website, GROMACS.org right below the main section for 2021 series. Regarding features, it has support for offloading the non-bonded computations to the SYCL devices, particular Intel GPUs. It's made a separate release because while it is stable, we do want to be able to update it more often than the main release because SYCL is in very active development right now, and you might expect more features and more hardware support during the upcoming year.

Nicole Huesman: As we wrap up today, Andrey, Eric, where can listeners go to learn more?

Episode 18: Porting GROMACS Across Heterogeneous Architectures
Host: Nicole Huesman, Intel
Guests: Erik Lindahl, Stockholm University & KTH Royal Institute of Technology; Andrey Alekseenko, KTH Royal Institute of Technology & SciLifeLab; Roland Schulz, Intel

---

Erik Lindahl: So, you can find a lot of information at GROMACS.org. That website itself can look pretty outdated and that's because all the real action is happening on GitLab where you see the code. One of the things I love personally about open source—not just our own project—is that you can literally look under the hood, dig out all the plumbing and check the patches as they're coming in hour by hour, minute by minute. So, all this development happens completely in the open. So, go and have a look at Andrey's code and see what he and others in the team are doing and learn from that. Please steal our code, that's the whole point of open source.

Nicole Huesman: Excellent. So well said, Eric, thank you. And Roland, is there anything more you'd like to share with folks, any other resources?

Roland Schulz: The same is true for our DPC++ compiler. It's on GitHub and all of the source is available. People can experiment with the internals of the compiler, file issues both regarding bugs but also ideas both when it comes to implementation as well as to the specification. So, the SYCL center is open and our extensions are open and it's part of DPC++. If there's anything which we missed or where you have ideas of improvements, we're very much looking forward for feedback.

Nicole Huesman: Great. Thank you. Erik, it's always so great to have you on the program. Thanks so much for sharing your knowledge.

Erik Lindahl: It's always a pleasure talking to you, Nicole.

Nicole Huesman: And Andrey, thanks so much for diving down into the gory details with us and providing us with your experience.

Andrey Alekseenko: Thank you for the pleasure to be here.

Nicole Huesman: And Roland, always wonderful to have you with us. Your insights are truly invaluable.

Roland Schulz: Thanks, this was a fun discussion.

Nicole Huesman: And thanks to all of you out there for joining us. Until next time! Let's continue the conversation at oneapi.com.