Code Together Podcast Episode 30: A *Big Step Forward: Moving Ginkgo to oneAPI*
July 2021
Host: Brenda Christoffer, Intel
Guests: Hartwig Antz & Terry Cojean of KIT, and George Silva of Intel

Brenda Christoffer (00:04): Welcome to Code Together, an interview series exploring the possibilities of cross-architecture development with those at the forefront. I'm your host, Brenda Christoffer.

Brenda (00:16): Today's topic focuses on research being done with Ginkgo, a production-ready, sparse linear algebra library used for high-performance computing on GPU architectures. Ginkgo was designed with a high level of performance portability, and focuses on software sustainability. It is also using oneAPI, which is an open cross-architecture programming model. Joining us today is Hartwig Anzt. Hartwig is a group leader at Karlsruhe Institute of Technology, which we'll refer to as KIT and a research scientist at the University of Tennessee. Welcome, Hartwig.

Hartwig Anzt (00:53) Thanks for having me.

Brenda (00:55): Our next guest is Terry Cojean. Terry is a lead developer of Ginkgo software at KIT, and supports modern parallel programming paradigms and HPC technologies. Terry, thanks for joining us.

Terry Cojean (01:-8): Thank you for having me. It's a pleasure to be here.

Brenda (01:13): Our third guest is George Silva. George is a program manager for Intel academic programs for oneAPI. George is also a developer and he's been at Intel for 9 years. Great to have you with us, George.

George Silva (01:24): Good morning, thank you.

Brenda (01:25): So, let's get started. Hartwig, could you share with us some details about the Ginkgo project?

Hartwig (01:31): Yeah, sure. I'm happy to do so. So, as you already mentioned, Ginkgo is designed as a math library and, even though it is focusing on GPUs, from the beginning on, portability was one of the key concepts. And if you look at the code, it is actually very well reflected that we have a core that contains all the algorithms and all the library infrastructure, but all the core is useless without the actual backends. So the backends that we then deploy for different hardware architectures. And historically, we started with a CUDA backend because when we started Ginkgo and video GPUs was basically what most of the users had access to, but over the time, that has changed, and now we're also looking into other GPU architectures and, in particular, also the Intel GPU architectures that will be coming up.

Hartwig (02:21): I think having that performance portability in mind was one of the success concepts in Ginkgo. And we're very positive that this concept of decoupling the algorithms from the kernels that form the backend, is actually the right decision to do if we also want to run on future architectures that maybe look totally different from what we have today. And the oneAPI ecosystem has proven to be a very powerful and useful option for us to actually target different architectures that are all supported by oneAPI and the DPC++ ecosystem. In particular, when we extended our effort, going from CUDA to other hardware architectures, we learned a lot of lessons, and we actually really understood, it's also important to actually acknowledge that other programming models may be needed for other GPU architectures. And in that step, it was also very helpful to actually get support from those who are behind the oneAPI ecosystem. We have to acknowledge that we got a lot of help from Intel, and we also got a lot of help of the developers of the compilers and so on. And that allowed us to have a very smooth transition.

Hartwig (03:36): Nevertheless, of course, there are some obstacles when moving from one back-end architecture to another back-end architecture. Fortunately, the process of moving from one to the other was simplified by actually tools that were available from Intel. And one of these very useful tools was the DPC++ Compatibility Tool that is offered by Intel, and that is free for everyone, and that can be used to actually convert CUDA code into code that can be executed in the oneAPI ecosystem. But even with that tool, we had some obstacles. And I think Terry can give a lot of details about what these obstacles actually were.

Terry (04:17): Yeah. So before going into that, precisely, I think it's good to know that DPC++ compatibility tool helps developers a lot in porting the code to the oneAPI framework in a sense that, when you convert your code, it tries its best to convert the architecture from CUDA to oneAPI or DPC++, which are two very different languages.

Terry (04:39): So it translates your source file from CUDA code to a SYCL code. And when it tries to do so, you get a nice warning saying "oh, I was not about convert that feature here, maybe you want to take a look at the precise aspect and the lines of code." So you need manual checks for that. And the rest is about to port maybe 80, 90%, maybe more of the features, but of course there is some parts that can be a bit lacking. So we will dive into those specialized aspects. I think it's important to know that the main difficult part in workflow, which was also developed for the first time when we brought it from CUDA to HIP, we had a similar experience, and we were able to define the workflow which we could re-use for oneAPI porting. And what we basically do is that it's in three small steps. We first add a new non-working backend, as Hartwig said, we have multiple back-ends like CUDA, OpenMP for CPUs and so on.

Terry (05:37): And what we do is we add the new non-working back-end with stub kernels, but main features are the right way so that we can execute on the device. And it was the features which all parts of the code rely on. Afterwards we manually port, little by little, every single kernel and functionality from CUDA to the target language, in that case, DPC++ and SYCL. So, that part is somewhat automatic, and, of course, the final part is coming back to everything that we ported and tune the performance, manually finding the potential parts where performance is lacking and need some more work. Yeah, that's the main overview of the workflow.

Hartwig (06:16): Challenging that, maybe we have to mention is that oneAPI supports a lot of different hardware architectures, and the different hardware architectures differ, for example, in the workgroup size and the subgroup size and so on. And the challenge is now that we want to have a back-end that not only runs on all these hardware architectures, but that also gives a very good performance on all these hardware architectures.

Hartwig (06:42): And that is somewhat different from the CUDA approach where you have the GPU's from Nvidia that all have very similar characteristics, so they all have a warp size of 32. But now all the different hardware architectures that are supported by oneAPI, you cannot assume that they all have the same warp size or group size or subgroup size. And that was actually a challenging obstacle for us. So we had to find a possibility that we implement these kernels and instantiate these kernels, and they still get a very good performance on different hardware architectures. So Terry and his colleagues have found a very smooth workaround that actually allowed us to encode the hardware characteristics in the signature. And that signature is now used to, during runtime, select the correct kernel while compiling kernels for different hardware architectures at that time. Also important is to notice that the oneAPI ecosystem and DPC++ is built on SYCL, and SYCL is fundamentally different in the execution model from CUDA.

Hartwig (07:46): And maybe Terry can give more details on that.

Terry (07:49): Yes, that's a very good point that DPC++ to CUDA have many differences, which were some of the changes, also during porting. A developer needs to be aware of those differences. So one aspect for example DPC++ is task-based. That means that the basic execution model is that everything is asynchronous in some sense, and you have a run time, which automatically will manage memory movements, for example, between CPU and GPU and all of that. So that is very nice for some memories, which can benefit from that and, and so on.

Terry (08:25): So that's a good usage model, but on the other hand, Aurora is also used to manual and precise tuning, so we know when to communicate this data, where it should arrive, and all of that. And we like to be able to move all that data also. One good extension that Intel added for that, and which allows to follow that model, which is very close to the CUDA model, is unified shared memory a feature of DPC++, which adds basically functionality for just doing that: copying specific pointer from one place to another. And you can also remove some of the asynchronous variables that you choose, in particular with other tools, so that you follow the priorities of your kernels. Using those two features, you're able to get them back to a CUDA model somewhat. And being able to go back to a CUDA model ... which is a very useful aspect.

Hartwig (09:20): With all that being said and all of these challenges, we still have to admit that it took about two months to three months for one experienced programmer to actually generate a working back-end for Ginkgo to run on oneAPI hardware. And I think that is a big success story. And that also shows how comprehensive the porting tools are and how well they are actually also working. One fundamental difference between the oneAPI programming model and other portability layers is that oneAPI basically goes in the direction of open specifications. So the idea is to have one public community-based interface, and every vendor and every developer is actually free to implement kernels and functionality for specific hardware following that interface.

Hartwig (10:15): And I think that is a very big step forward, moving away from the vendor-owned interfaces and in a direction where, for the programmers and the application scientists, it's much easier to actually write a code and then use that code on different hardware platforms. So this will certainly make the work much easier for us, much easier for the application specialists. And it is definitely something that is well appreciated by the whole community, not only by us, but also, if I'm talking to application specialists, they really like that idea. Of course, there will still be the possibility for vendors to implement their own library following that specifications. And I guess oneMKL provided by Intel is one of these examples. But again, it is possible that someone also implements functionality for Intel GPUs based on DPC++ that provides the same functionality, and it could even be that that functionality outperforms Intel's is oneMKL. So we think that this open interface and open standard and open specifications is the right move forward. In particular, if we're looking into an area of increasingly different hardware architectures on the market.

Terry (11:32): Yes, I definitely agree on that topic. I think - related to that it's very nice that Intel has all that support behind the scenes for applications, which are willing to adopt those standards of porting that with oneMKL API design. With everything being accessible in a public repositories and so on. That is one aspect. But on the other hand, there is also the question of how much features we put into that and exceptions only carry us so far as they exist. So for example, if we want to consider architecture-specific features like TensorFlow or any other features such as mentioned, hardware details like what sizes and key numbers coming directly from each CPU and GPU, there's a question of how we can add that information in a way in that can standout in DPC++.

Hartwig (12:18): That obviously is a big question. So how much should general portability layers actually buy into those special function units and so on. Of course, we don't want to lose a lot of performance by not supporting them. Like for example, we don't want to run on GPUs that have Tensor Cores, and just ignore these Tensor Cores.

But at the same time, it gives a lot of burden to the developers of that interface. And it also gives a lot of burden to the application scientists that then use the portability layer by actually understanding all of the technical details, because I cannot use the Tensor Cores, unless I really understand what is happening and for which kind of operations they give me good performance. So that is really a challenge. And I think there's no clear answer to that right now.

Hartwig (13:03): I think it will always be a trade-off. And we also think that the oneAPI standard committee has that in mind, and they also think about that, and therefore I'm positive they will find a middle ground where the features that really are necessary to get performance will be supported while other features that are probably of minor relevance will be left out. And that is also true for the future. I mean, we don't know what the future will bring, but maybe we will, at some point, have devices that have like a quantum accelerator or something like that. And that of course will be a totally new game to us. And the question is then, how will the standard evolve to also support these aspects? And especially in that context, I think, it is very important that the whole community is included in these discussions.

George (13:51): So thanks so much for introducing the project and everything the team is doing. So how did you find Intel oneAPI resources, how did you connect to Intel first time? So what kind of research you're looking for that you had the chance to share more about the project that the research team is doing?

Hartwig (14:06): So I think the initial contact to that was SuperComputing some years ago when it was still in person and Intel actually announced this oneAPI ecosystem. But, later on, what really drove the whole process forward was Intel® DevMesh where I created a project for Gingko, and we started a discussion on that. Then we started setting up a call and discussing what are our interests? What are our needs? Intel was very welcoming in terms of supporting us with expert knowledge. So in terms of how the compilers translate things, how are the characteristics?

Hartwig (14:43): And then we soon got access to the Intel® DevCloud, which actually contained some early hardware devices that are looking similar to what we can expect for the Ponte Vecchio, and getting access to these devices was very helpful in getting actually hands-on experience. How is it running on these products? And I can really recommend researchers to go to DevMesh.Intel.com and find out about this concept and also start a project there. It's really useful, and it worked out well for us.

George (15:18): What makes oneAPI different from other portability layers?

Terry (15:22):Yes. That's a big question. I think it did not occur to us to be aware that it was a significant effort by AMD to get portability from CUDA to their GPUs using the HIP ecosystem. So on that topic, I think they actually did a good job in that respect. So it's also open on GitHub. It was and is available for the researchers. It's very close to CUDA, so it's very easy to port to such an ecosystem.

Terry (15:48): On the other hand, I think that HIP ecosystem, because it's so close to CUDA, there's also some dangers with that ecosystem because what's happened after CUDA 11 is that CUDA changed the interfaces of a lot of their API. And suddenly the portability layer was broken and they need to do significant effort to support the new APIs. But on the other hand, what Intel is doing with DPC++ and SYCL, because there is two-part vendors like Codeplay, for example, which is very invested in the SYCL project and allows to execute code on CUDA hardware directly, and there is also other vendors, including AMD, as well as OpenCL portability layer Adonis. I think SYCL is a nice portability layer in the sense that you can really get the code, which you write once and apply it once. So it's

in terms of compatibility for all architectures. In terms of portability, it's very powerful, and that's a big feature of that portability layer.

Brenda (16:50): The work here at KIT has been quite incredible. George, you work with a number of universities. What do you find is most insightful from the collaborations that you have with them?

George (17:02): So working with oneAPI universities, we got to learn a lot of different usages they will make of the technology. The idea of oneAPI supporting multiple architectures, so we will see amazing projects like Gingko using GPU, but also you have multiple projects sharing the execution between CPU, FPGAs, and GPUs, some projects use them all together. So it was very exciting to see one language supporting multiple architectures and let them sharing the execution and get amazing performance out of them. Amazing ideas. So, that's the most exciting part. And, of course, all the feedback that the universities provide to us help us to get to the next level to oneAPI. So, improving all the compilers and libraries and getting to the max level that we need for the hardware to be successful. It's amazing to learn more on DevMesh.Intel.com. We have a number of projects, a lot of professors, they are posting their projects there.

Hartwig (18:01): So looking forward, Ginkgo will be the numerical core of the EuroHPC project MICROCARD. And that project actually aims at the simulation of the cardiac effect and the human heart to better understand cardiac arrhythmia. And Ginkgo would provide the numerical code, and with this your EuroHPC project having the goal to actually run on the future systems that are procured in Europe and in the U.S., we are hopeful that the oneAPI portability layer that we currently support in Ginkgo will help in actually getting the performance out of the systems there. Right now we're already looking into getting performance on pre-Exascale hardware that is available and the Intel DevCloud, and that is also available and the pre-Exascale systems or pre-production systems in Argonne. And we have very good experience with using these systems.

Hartwig (18:56): And therefore we are positive that the steps we are taking now will help us in getting good performance on Aurora and other Intel-based Exascale systems.

Terry (19:06): Yeah. So if I can add to that, so far the porting has been quite straightforward and very good. Of course there were a few issues because we were using maybe not too much of an ecosystem at the beginning, but as soon as we were able to get past that, we were already at satisfying performances on the hardware we have access to. So I think we can definitely get good performance in the future on those systems.

Hartwig (19:29): And with good performance, we mean, if we're talking about the performance relative to the hardware characteristics, we are actually getting a better performance ratio than on other hardware systems and other GPU hardware. In the end, we probably want to also give you a pointer to the Ginkgo and repository, and you can find Ginkgo under github.com, Ginko-project/Ginko. And there you see the current state of the art of the implementation, and also the support for DPC++. And you're more than welcome to check out what is there right now, what is coming in the nearby future by looking at the pull requests. And please also give us feedback in terms of what we can improve and what your requirements are in terms of solving sparse linear systems on oneAPI-supported hardware.

Brenda (20:24): Well, thank you, Hartwig, and Terry, your insights here were really valuable for the audience. And it's been really exciting to hear about your work. Thank you again, Hartwig.

Hartwig (20:36):Thank you, it was a pleasure.

Brenda (20:38): And Terry, thank you so much for some of your technical insights. It's really valuable to hear about the difference that you've had in working with SYCL and oneAPI.

Terry (20:50): Thank you very much.

Brenda (20:52): George. I love hearing about the great university research that you and your associates and all of the universities are working on. Thank you so much for your time.

George (21:02):Thank you.

Brenda (21:03): And also I'd like to thank our listeners. Thank you so much for joining us. Let's continue the conversation at oneAPI.com.