Music] it means that I can't see your face but you can you can see mine okay so let's let's a little early so I'm just gonna fill the space a little bit just for some introduction I want to think I'm finding interesting is that in these kinds of talks I've been delivering talks about GPU programming for a long time probably this would open MP and now with sickle and the audience keeps getting bigger and bigger and bigger it's sort of indicative the trend that GPU programming is really beginning the hip hit mainstream so I'm really glad that's to see a really big reception but at the beginning I was able to talk to a fairly empty hall so if I make mistakes nobody noticed now unfortunately you guys are gonna know what's gonna happen so today we're gonna talk a little bit about modern GPUs and how to program them but not just in any unique proprietary way but I want to talk about how to program a using very standard C++ because after all I'm really a C++ standard guy I've heard several hats now I worked in the C++ standard I work in the chrono specification standard I chair cycle which is their heterogeneous C++ programming language I remember I've been a member C++ for a long time but be neat and I'm chairing a number of groups that's involved with some form of of GPU and heterogeneous programming but underneath of all this I do have a deep history back of about 20 25 years working at IBM as a research scientist working at OpenMP so I actually came from a high-performing community high-performance computing community recently transitioning into autonomous vehicles consumers of consumer devices so now I really am seeing both sides of the fence in a way so this work is actually written by myself and my colleague Olin Browne who's not here with us today and it's similar to talks that we have extended and delivered there's a bunch of standard boilerplate about what we do and why I'm here I do work for this now work for a UK company now a Scottish company in Edinburgh my entire team is in Edinburgh I'm actually still in Canada but ACC is the perfect car friends to talk about the some of the technology that the UK has developed because we do do deliver a lot of solutions originally for graphics like the physics processors and games but now probably but these processors now these graphics processors now are transitioning into machine learning processors from for things like autonomous vehicles for advanced drive driver assistance and in these are the places with GPU programming efficient GPU programming is necessary so that's why one of the reason why I'm I've been involved so I always like to break up my talks into three parts because I think that when I look at talks I really get a cognitive overload when I see just so many topics being involved so I really only talk about three things now each one of these could be somewhat involved we're gonna talk about what's still missing in C++ we're gonna talk about what makes GPUs work so fast now the picture I have here is a little bit incongruous it's actually for my LLVM keynote from last year where I actually talked about the problems of heterogeneous programming having done it for almost 15 years now I feel like we really know what the problems are and some of the best solutions that we can achieve up to this date so that's why you actually see the Four Horsemen of the Apocalypse I hope I'm not offending anybody here that's particularly religious and we're gonna talk about what's modern C++ they can work on GPUs CPUs and pretty much everything so what's still missing in from C++ just a bit of a recap I know you didn't sign up for this but I want to recap what CP what we've achieved so far in C++ 20 which is many of you guys and myself included see Roger I see a few people other people who's from the standard Committee what we did achieve in C++ 20 is that we've got concepts both the convenient syntax and the original syntax we have contracts we have ranges we have

Co routines and we have modules what we didn't get was reflection and what we didn't get and I was really hoping was executors we had a good chance but towards the end we just kind of erred on the side of caution because executives because networking depends on executives we really would have gone we might have gone networking if we gotten executives but that's not the case either and up coming after executives there'll be a new vote there should be a new version of a sink and a new version of future then so why do I bring up all of this stuff it's because it's now becoming increasingly clearly clear what the right abstraction is when you're programming C++ for parallelism and heterogeneity if you're using cores or hardware thread you really should be using C++ 11 14 17 threads multiple people at this conference and at other conferences have talked about what choices you should make you can use things like async if you really want to know how many hardwood threads you have there's this thing called work concurrency if you're using vectors before C++ 20 there wasn't any there was just parallelism ts2 okay and then if using Atomics and fences and things like that there's a bunch of TS there's c++ 11 14 17 Atomics if you're using parallel loops there are things like a sink okay but until now and until now there was also C++ 417 parallel parallel algorithms if you're doing heterogeneous offload to some GPU devices there really isn't anything in the standard that gives you that up disability you pretty much have to go outside of the standard and use well-known practitioners frameworks like OpenCL cycle HSA OpenMP open ACC which I when I came from to some not ACC but OpenMP Coco's Raja if you're in the US Department of Energy okay they have forms their own frameworks on these things if you want to do distribute a computer that is just distributed computing like know to know computing you would want to do something you've probably have heard of MPI okay you would probably use something called but if it's very C++ specific it would have to be something like HP X from Louisiana State University there's also something called UPC plus plus if you want to use caches C+ 417 actually has false sharing support how many people know that okay somebody should give a talk on those hardware those hardware our features that api's that allows you to do cash with control caches if you want to do Numa there is nothing okay there's not anything that you can do to control the non-uniform memory access of your system if you're doing if you want to do threat local storage other than on a CPU there is nothing okay there's I didn't put it down there but I should actually when I adjust my slide there's threat local which gives you access to the CPU in a threat local manner but not on GPUs on anything else if you want to do exception handling it's really only available in a sequential environment and not in a concurrent environment these empty areas space is the stuff that I work on the stuff that I have constant telecom cause would put experts around the world on how to how to work on these things okay so the first thing I want to talk to you guys about is just a kind of a crash course on tasks versus data parallelism I know some of you guys may be familiar with this but essentially the idea is that you want to maximize the utilization of the hardware and reduce dependencies how you design your algorithms is going to vary depending on whether you're describing tasks or data parallelism so task parallelism you usually have few large tasks with different operations like and it's it's about control flow and it's usually optimizing for latency and in the case of data parallelism you're gonna have things where you have many small tasks with the same operations or multiple data in these cases you're gonna optimize for throughputs now right off the bat I want to talk about some of these terminologies I know they get thrown around a lot they're

kind of like the blind man with feeling the elephant they do they really mean somewhat similar things well let me just put it this way latency is going to be the amount of time in this particular case that it takes to travel through the tube and bandwidth is basically how wide that cube is and if you were moving water for instance to the tube the amount of water you would move through would be your throughput the definition isn't particularly interesting what I want to do is go through an example that many of us are familiar with especially in the in North America and I suspect similarly here we have in North America where this institution called the DMV the Department of Motor Vehicle I'm sure there's a similar one in the UK and have you ever noticed that when you go to this particular institution the lines are extra long and you pretty much want to reserve your lunch hour plus another hour to go do whatever operation you want to do at this place well that's how it is in North America the question is what's wrong with this what's wrong with these people you always go ahead why is it that I have to wait so long well the core the thing is that the thing is that it depends on what you're optimizing for and it turns out that what you want to optimize for is different one than what the DMV is optimizing for your goal is basically optimizing for latency that is you want to have to spend the least amount of time there right I mean nobody wants to spend any more time than they have to and CPUs are optimized for latency so they minimize the amount of time per task now the DMV the Department of Motor Vehicles goal is not your goal okay they're optimizing for throughput they wanted to have they're looking at the number of serves serves number of people they serve per day and they want they actually want long lines to keep the employees busy okay they do not want actually the lines to be short so this gives you an idea why that CPU and GPU is actually designed differently to the point where they're nearly on the opposite end of a spectrum the GPUs are optimized for throughput they are like the DMV okay the CP and so and and when you have latency it lags the bandwidth so in computer graphics Winston's we care about the pixels per second more than the latency of any one particular pixel we're willing to make the processing on one pic so made them and take two times longer as long as we get lots of pixels going okay now you finally understand why it is that you will always wait a long time at the DMV so you will here in this this area as the state of the art about what's generally known as Flynn's taxonomy and influence taxonomy they're these things called single instruction single data single instruction multiple data and then further on there you'll have things like multiple instruction single data which is weird and multiple instructions multiple data and we're gonna talk a little bit about that because it's important to understand that essentially all this means is that in single instruction single data it's it's a sequential computer that exploits no parallelism I do in the instruction all the data stream there's a single control unit that fetches a single instruction from memory and then the control unit then generates the appropriate control signals to direct single processing elements to operate on single data stream on the other hand something like a sim D is the beginning of parallelism where it represents the organization of a single computer controlling a control unit processing unit and a memories unit or something like that and the instructions are executed sequentially but it can be achieved by pipelining on multiple multiple functioning units if you have multiple instructions Cindy that's a little bit weird this is a pretty uncommon architecture and it might be used for things like before you think it doesn't exist it does it's used for things like fault tolerance for instance heterogeneous systems could operate on the same data stream and has to agree on the result and if

they all agree then you have fault tolerance if they don't agree something's wrong the space shuttle is actually something a good example is something like that then finally you have multiple instructions multiple data which is multiple autonomous processors and simultaneously executing different instructions on different data these kinds of multiple mi M the architectures include things like multi-core superscalar processors and distributed systems they are using either either one shared memory space or distributed memory space okay so when you look at what kind of processor we should build you see that Bay's this the designers the CPUs and GPUs are aiming for different things for cpus they tend to be small number of processors of large processors that's the world I came from the last 20 years of my life there's more control there's less processing and it requires more power they optimize for latency that's what you guys used to want these days you now are asking for more GPU kinds of processors and that's something that I've been involved with also for about 15 years these are large numbers of small processors with less control with a lot more processing they require less power and they optimize purely for throughputs okay and they optimize especially for the type of computation that's a sim D single instructions and multiple data they almost all have some sort of vector processing unit that allows you to do that when you combine a lot of these either multiple cheap CPU cores and GPU close together you get heterogeneity in a multi-core CPU each core might be optimized for a single thread this just fast serial processing you might be it has to be good at everything and you pretty much have to minimize the latency of one thread with lots of big chips or big chip caches with sophisticated controls okay so simply architectures essentially just basically means using data parallelism okay it basically has an improved trade-off with the area and power and the parallelism is exposed to programmers and the Kerr and the compiler it's hard for compiler actually so when I was a compiler writer for many many years and the dream was always to get something called Auto factorization to have your code automatically be vectorized it turns out that that's actually quite difficult because there's so many different kinds of vector instructions that's out there in fact Intel themselves has probably about eight when I used to work at IBM we have three ourselves and almost every other company including AMD has several forms of these and what's worse is these instructions are getting wider and wider and bigger and bigger they're not necessarily compatible with the previous ones making it very difficult for compilers to figure o