MIT CSAIL Alliances | Saman_Amarasinghe

Welcome to MIT'S Computer Science and Artificial Intelligence Labs alliance's podcast series. I'm Cara Miller.

[MUSIC PLAYING]

There was a time when gains in tech weren't too tough to come by. You could say the living was easy. Basically, What happened was, if you want to double your performance, you don't do anything to your program. You just wait a year and the performance doubled.

But, says Saman Amarasinghe, those days are gone. And in a world shaped by machine learning, boosting performance is going to require a new set of tools, which might shake up everything.

If you write the description of what you want to get done, with that description, is it possible to write a program? So we are coming closer and closer to English. We are not there yet. But the next thing is, if we get to English, what's that? Is that a programmer?

On today's show, Saman Amarasinghe is a professor in the Department of Electrical Engineering and Computer Science at MIT, and a principal investigator at CSAIL. And he joins us for a wide ranging conversation about why computing has changed and what lies ahead?

[MUSIC PLAYING]

On July 6th, artificial intelligence made front page news in the Wall Street Journal. Excitement over AI, they said, was helping boost the tech sector out of its 2022 doldrums when the NASDAQ plunged more than 30%. The rise of AI and machine learning, it feels like a major inflection point for tech, but there's still a lot to play out. If we want to understand what sort of inflection we're looking at.

There's something happening with Large Language Models. But it's hard to know because two things-- these Large Language Models we have access to two things like GPT 3 GPT 4, they are still infants.

And the thing about infants is, it's really hard to know what they're going to be when they grow up.

The first big Large Language Model only came about a year ago that we started tracking this. So we have this a year old infant that is just going very fast and improving its capabilities. The interesting thing is, where will it land in a year or two?

Amarasinghe says that the split between those who are euphoric about where Large Language Models are going and those who are pessimistic, it's a little bit too stark.

When I talk to people, half the people say, oh, GPT 4 can't do this, this, this. It's not good. And so they are trying to look at the other part. Other group of people are so scared saying, oh my God, this is going to do everything. It's going to take over humans. It's hard to find that middle ground saying, OK, it's going to stabilize in something because I don't think we know.

Still, he notes the potential is huge, but many people don't realize the problems that have to be overcome.

Problems Amarasinghe thinks can be surmounted but are not always acknowledged.

Right now, language models running training and inference is extremely expensive. So it costs about hundreds of-- 100 million plus more than that to actually train a model like a GPT 4. And still, it's costing a fair amount to basically do an inference each token at a time, and you need a lot of tokens in there.

So in fact, we are looking at ways to make that faster and cheaper. And my hope is in two years, this will be cheaper. Because right now at this cost, you can't just change something like a Google search to Large Language Model and it will bankrupt Google because you can't yet because it's too expensive.

And there is another problem for machine learning. A lot of it, Amarasinghe says, isn't 100% accurate, a problem he thinks is going to continue.

So the interesting thing here is there are a lot of problems. If you get a 99.9% accuracy answer, you are OK. But the area I work on compilers. People expect the compiled program to be 100% accurate. It's unfortunate. I'd rather them not have that expectation. But that is problematic in that sense because if machine learning say, I will do it. But once in a while, I will do something buggy, people say, no, I don't want that.

So how do we get from accurate to very, very accurate. Well, you know how people use tools? Maybe Large Language Models could too.

My feeling is Large Language Models, they are starting to do that. They will start using tools to basically overcome some of these deficiencies. So my hope is in two years, this Large Language Models will have bunch of tools they can use. And with the tools, they might be able to do a lot more than right now today things like GPT 4 is capable of doing. So it will be very interesting to see what that path is. Hopefully, basically run that-- right now, we think in inflection point in this kind of thing. Run through with that would be very fun.

In some ways, it's no surprise that Amarasinghe loves inflection points. As his career, you could say, was built around one. An inflection point that has, much like AI, rocked and reshaped the tech industry. This particular inflection point started, unassumingly enough, when a guy in his mid 30s, a director of R&D at a semiconductor company, realized something.

The number of transistors in an integrated circuit kept climbing. It was 1965 when he made the observation, and the engineer was named, of course, Gordon Moore. Moore's law shaped computing for half a century. It changed the fortunes of companies. And it's disappearance, coupled with the development of AI and machine learning, well, it's reshaping the tech landscape all over again.

If you look at before Moore's law-- I will call it the time of the pioneers, the machines were very small. So our ambition was always larger than the machine. So to get anything out of the machines, you had to do incredible performance engineering hacks because the machines had little memory little compute power and we want to do more. So those days, it was all about lower level performance engineering to get anything done. Then we reached this golden age of Moore's law.

Basically, what happened was if you want to double your performance, you don't do anything to your program. You just wait a year and the performance doubled. And in fact, in that time, the hard part was not your lacking resources, what to do with excess resources you get. So we came up with lot of new ideas, new ways to do that. But also when there's excess, there's a little bit of gluttony. So people became inefficient because there's no need for being efficient. No need to be concise. So there's a lot of that happened.

OK. And that help is a lot on the hardware side. The hardware will come and help you.

Yeah. Because what happened was, I'm getting double the performance. If I don't need it, I can be more relaxed. I can be lot less. I can just use things-- so for example, to simplify my life then looking at performance. So a lot of people did things that you could have done more lot more efficiently but you don't need to because you are getting all this performance.

So Moore's law ended about, I think, two decades ago basically, but now we are hitting to a point now there's not much fat left at this point. We have trimmed them out. And then machine learning suddenly said, we need a lot more performance things like machine learning, training, and inference required a lot of performance that right now we can't provide. The hardware and software can't provide that they can use.

So there's a huge renaissance of these performance engineering part that was in the pioneering year. Because right now there is no free lunch. So if you want to get performance, you have to work for it. And so there was this entire middle part that nobody cared about it because the performance was free. But now we had to go back to lot of these old techniques are coming back, old thinking is coming back. So this is very interesting in that sense.

Let's talk about some of the specifics of your work. Maybe first, it might be good for listeners who are a little bit less familiar with how a code that a programmer writes gets run on a piece of hardware. So do you want to talk a little bit about what that process looks like? And then maybe insert yourself.

OK. So normally, people writing very high level language. That's what we understand. And then a machine understand a much lower level machine instructions. So you need a translation from high level to low level. So you give a high level set of instructions, and you need to get a machine instruction.

So if you have been following all these things about machine learning, you would say, hey, we know how to do translation. You give the high level program to neural network, and it will translate. That's what Google Translate do everything. But there's a small problem. Google Translate and all these translate software, if you get a 95% or 99% accuracy, you are very happy.

But when you write a piece of program and when you run it-- translate, you are expecting 100% accuracy. So you can't directly use machine learning. So there are multiple ways of doing that. If you want really high performance, you write that program in a language like C or perhaps Java or Julia or something like that. I'll talk about language like C. C, the program runs through a set of processes called compilation.

A compiler takes the high level program, not only it just translates it, but analyze it trying to figure out how to get the best out of the hardware. And the compilation take a long time because it's looking at all the choices all the options to say, OK, how should I get the best out of the hardware and hopefully it will produce a assembly program that can run on the machine that get the best performance?

Or our golden era of Moore's law, people realize, I don't need performance, why do I compile? So bunch of very high efficiency languages emerge, easy programmable languages called dynamic languages. A language like Python. There, writing a program is very easy. And it doesn't get compiled. It's what we call interpreted.

So what that means is you don't care about performance. The language is very easy to write. And there, at runtime, you figure out, what do I need to do to run this language and run that in here. So a lot of my work has focused more on the compilation side. The places where you really, really need high performance and how to do that, and that's what I have been working on.

OK. Give me a sense of then, how that connects to why focus on programming languages compilers for those post Moore's law gains?

So if you want to have really, really high performance for a small piece of code, you can give it to engineer, ask him to spend-- him or her to spend a year or six months, and really optimize that for certain very small things, and they will probably directly write in assembly language or whatever and get really good performance. The problem is that's very inefficient.

So what we want to do is we want to have the cake and eat it too. We want to have programmers be effective, productive. So what that means is, we like them to write the programs very close to this these highly productive languages like Python. But then, give them really, really high performance.

So a lot of work I have been doing is to look at ways to get there so instead of asking programmers to do all this low level details, spend hours or months optimizing something, write it in a very highly productive way and then still give performance. So one thing we have been working on is area called domain specific languages.

So if you look at something like C, It's a general purpose language. It's a one thing that fits everybody. But if you look at how people program, there are domains like graphics or computation biology, or Earth simulation, they are doing some very specific tasks in here. It's not one program. They are doing a set of tasks in that.

And in those set of tasks, they also certain things you can do they have learned over the years that get good performance. So what domain specific languages try to do is capture those tasks natively and do these things we know how to get those tasks done faster. So there's an added benefit that you got from doing domain specific languages because now, we are trying to capture the domain at a very high level.

So sometimes if you get a good domain specific language, it becomes easier to program than writing in C or languages like that, even without optimization. So you write this program at a very high level, that is, into the domain, but the compiler and that domain specific language and its compiler knows how to take that and get really, really good performance.

So for example, about 10 to 12 years ago, we developed this language called Halide. And at that point, it was me and Professor Fred Durant and Jonathan Ragan-Kelley, who is now a faculty at MIT but at that time, he was a graduate student. It was his PhD. That language was targeting graphics programmers.

Because we understood the graphics programmers, they spend months really optimizing programs because if you are trying to write a filter, for example, Photoshop, that's not that easy. It takes months to get that filter to perform. So if you run it through a large picture, it doesn't take minutes to do. It can do in seconds. And so we understood those things. And all the filters have certain patterns, certain characteristics.

So this language, Halide, basically said, OK, tell me those filters in a much more higher level in this language. And then we will deal with the kind of things you do for filters and basically get you good performance. So it became easier for these people writing these graphics programs to write those programs, and they had to spend weeks to get tuned for performance, they could do it in hours. So that is the kind of things we try to do to get good performance.

Are you thinking a lot, when you're doing this, about the efficiency and maybe even the ease and experience of the programmers?

Yes. A lot of times what happens is, we need to capture what's important for programmers because a domain specific language can't do everything. But if it doesn't do important things for the programmer, they can't use it either. So we need to capture all the necessary components in a way that it's useful for them. They can do what they want it to so and solve their problems, but it gives you the ability to actually give good performance.

If you put too many things, it becomes harder for compilers to work because it becomes too complicated. So you need to find the minimal instruction set of rules that we incorporate in there that programmers can use, and then that will become a useful domain specific language.

Do you expect that we're getting efficient enough-- and I think programmers think about this, companies think about this, that we will need fewer programmers within companies?

That's a very interesting thing. So what a lot of my work has been to give this expert programmer knowledge to non-expert programmers. So beforehand, for example, if you think about Halide to write that filter that goes through Photoshop, there's only very few people at Adobe who was able to do things like that. It is a very expert something that you have been trained for a long time in there.

What Halide said was, OK, you can have a lot more 1,000 people who can at least try to understand the filter, can write it, and get that expert level performance. So we have always tried to make it possible for more people to use that, but that didn't change the number of people, pardon me, perhaps it was faster for them so you didn't need that much time, so it reduced the workload. So that was going forward.

But now, with things like large language models in there, there's a very interesting thing in that because can some of these things we have to tell the write programs, can it be done a lot more automatically? So for example, if you write the description of what you want to get done, with that description, is it possible to write a program? It's not there yet, but there's this early indications that direction might be possible.

So for example, in 5 or 10 years, the programming might be more writing English and telling them what didn't work, and having that conversation. Then actually go in there and sitting and coding at a low level language. We have always-- when we started programming-- we programmed in assembly language. We programmed it very, very low level.

And then when we went to languages like Fortran and C. We raised the bar. And then some Python, you raised the bar again. So we are coming closer and closer to English. We are not there yet, but the next thing is, if we get to English, what's that? Is that a programmer? Is that a user? Who is that person?

They still will have to understand how to instruct the computer. They can't be just anybody. So there should be a certain level of knowledge, understanding, and education to do that. But they might be educated at a much different level. So this is what's very fascinating, what it's going to be in next 10 years?

Obviously, I think it's probably clear the number of programmers might go down, the amount of need. Because there's a lot of menial things normal programmers do, a lot of those things can be automated like developing websites, maintaining websites. Doing this a lot of menial parts that a lot of right now you need programmers to do might get automated.

But where it ends up is I think anybody's guess.

Right. Because I think it's an interesting question, who then is a programmer? What do you call those people if they're employed in a company, maybe they aren't programmers but they're something and what is their skill set look like?

I mean, even within programmers, there's this argument that real programmers only write C or real programmers only write assembly. Everybody else who's doing high level, they are not real programmers. So there's even that argument. So the next time-- so for example, a product manager normally will write a description in English and the data in probably Excel spreadsheet saying, here's what I want. And they normally hand it into the program who actually programs it. Now, it might be they hand at ChatGPT or whatever it is that will actually write the program. So there's an interesting part in there.

Finally, two big picture questions. One is, just what is the biggest challenge that you see staring us in the face right now that maybe it doesn't get covered enough in the media, maybe people are not talking about enough? But is there a challenge that you think is out there?

So the interesting thing right now is computing is dominant. Computing is everywhere. So if you look at your cell phones to your laptops to data centers, it's taking a fair amount of energy in the world. So as we go and think about global warming, even your iPhone doesn't look that much and doesn't seem that much. When you have billions of these kind of devices, it adds up.

So part of that is, even though you don't feel like, OK, I don't need any more efficiency, it's good enough, I can charge it, I think to reduce the energy use, reduce the amount of this consumption even at the small level that doesn't seem that much, when everybody has now not 1, 2, or 3 devices, it adds up. So there's a lot that has to happen even though you don't see the need yourself to make things efficient, make things less energy consuming.

As a global-- as a societal thing, I think that's very important. So there has to be a lot more, I think, done in that direction. When things are slow for us, we will say, OK, we need to make it run faster, more efficient. Or if we are running something very large and something like AWS start charging you hundreds of thousands of dollars, you say, OK, look, it's expensive. I want to make it cheaper.

Yeah.

But that part doesn't have to hit that kind of path because the things like small devices, everything, the sum total is taking a lot of world's energy. So how to reduce that, I think, is very important that. I don't think is that much attention at this point.

We talked to Vivian see last time about that, and she really-- something she really focuses on. And I think as you say, people do not think that much about that energy and global warming dimension of the phone in their hand, right?

Yeah. And that's what we are trying to do, For example, every additional instruction you need to run on this thing is using additional energy. So what compilers try to do and a lot of my research is to make things very efficient. Right now, basically to get traction, we do it where it's really needed like things like machine learning or these very large systems that performance is very critical.

Then it's obvious. They're like, OK, I need this. I can use it. But it should be needed and it should be taken advantage of in things where it's not obvious that I need it. But if billion people reduce their energy consumption in their devices by 50%, that's going to have a huge impact in the m .

Finally, I know you've done a lot with MIT'S global startup labs which helps create startups around the world. I just wonder-- you've got a big view here. Do you feel like the startup ecosystem is healthy? Do you notice big geographical differences? I just wonder what you see because most people don't have that high level view.

So one thing very interesting is, these days how easy it to build something. So when I did my-- we'll say first or second startup, even to get something going, you need to buy computers, you need to buy office space, you need to buy this, you need to do that. You need to have a pretty large budget to just do anything, just to get out of the door if you're doing anything in there.

Right now, if you are sitting in a dormitory as a student or something, you can basically build a product to a point it can get users with almost no money. Because assume you already have a computer. You can go to Amazon AWS, you can get these tools. Stuff like that open source libraries, you can do amazing things in very little money. So especially in the IT side, the dramatic jump is huge. So let me tell you a little bit of story about global startup labs.

Sure.

So what happened was, in us, if you have a good idea, it's easy to get access to capital. There's a VC ecosystem, a lot of ecosystem that is looking at the idea. It doesn't look at who you are. If you go to a lot of other countries, especially developing countries, and capital is restricted mainly, it's mostly hereditary.

If you are born to capital, you have access to capital. If not, it's very hard to get access to capital. Only way a bank loan is if you have to-- if you give some collateral. So if you are a student who came from a poor family or even a middle class family, it's very hard to get access to capital. So in a lot of these systems, people don't even think about doing startup because that was out of the question.

So one thing we realized about 20 years ago was, with the advent of smartphones even before the phones, there are a lot of opportunities that doesn't require that much capital. So global startup lab was born to send a couple of MIT students to mostly developing country. Go to a University, take 20 students and say, look around you. What problem can we solve using your phone? Mostly it's phone because people didn't have access to computer. Your phone in there.

OK. Let's see whether you can build a prototype to solve this problem. Let's see whether you can make a company out of it. OK. Talk about both the entrepreneurship side of building a company and also the technology side of-- because a lot of these universities in fact didn't teach them that much practical knowledge like how to build app? How to program your phone? How to build a backend for an app on something like the Android-- mostly Android operating system and stuff like that.

OK.

So we train them, and a lot of interesting ideas came about in these things. For example, starting from, a lot of these places don't have good health care, so you have to be proactive. So how do you know when a kid needs the next injection or next vaccination. So you can build a very simple app that basically record those things and send reminders say, OK, now it's five years, you need to get-- so things like that builds very simple apps in there.

The other interesting thing is, beforehand, to build a multinational, you need billions of dollars-- millions of dollars in there. Because you need to understand countries. You have to go there, open offices, hire people in there. Right now, with all these tools available. You can have a multinational product without going anywhere.

So in Sri Lanka, one of the startups came out of this thing called 4XE solutions. They have a drawing app called drawing desk on iPhone. They said, OK, we'll just create this app ourselves. And they, within Sri Lanka-- the two people who started the company didn't-- haven't even left Sri Lanka. They came from a middle class poor middle class background, so they haven't even left the country.

And within the country, they sit there, wrote this app, and they start basically using all the tools available, market it in China, market in the US, market it in like what 100 countries. And right now if you go to the iPhone marketplace and you look at these productivity apps, it is ranked higher than Amazon-- higher than Adobe app. It's sometimes ranked at number two or number three productivity app.

All run from-- now they have a nice office, but beforehand, one person office, one room office in Sri Lanka, never leaving it, and they were able to create this entire multinational organization that is basically in 100 countries selling this app. And that is incredible that able to do that in there with very limited resources, very limited knowledge even. Everything is basically gathered from the web, gathered from internet, and they have hired people outside in China, other places to help them out, but they're sitting in there.

So that's very interesting. So that's in a developing environment. Even in the US, I see so many students that are taking-- at MIT are just trying out these things. If you have idea, you don't need a bucket million to basically try it out. You can do it in probably $1,000, sometimes even less. So there's a lot of democratization of creating ideas in there. Of course, at some point, you need money, some point you need professional support, stuff like that. But to get something going testing something out is right now, there's no time that it's cheaper. It's easier.

Right. That's so interesting. The landscape has really changed. Saman Amarasinghe is a professor in the Department of Electrical Engineering and Computer Science at MIT. He's a principal investigator at CSAIL. Saman, thank you so much. This is great.

Thank you so much for talking to me. This was a lot of fun.

[MUSIC PLAYING]

If you're interested in learning more about the CSAIL Alliance Program and the latest research at CSAIL, please visit our website at cap.csail.mit.edu. And listen to our podcast series on Spotify, Apple Music, or wherever you listen to your podcasts. I'm Kara Miller. Our show is produced by Matt Purdy. Tune in next month for a brand new edition of the CSAIL alliances podcast and stay ahead of the curve.

[MUSIC PLAYING]