**MIT CSAIL Alliances | Charles_Leiserson_Project_5**

Welcome to MIT's Computer Science and Artificial Intelligence Lab's*AlliancesPodcast.* I'm Kara Miller.

[MUSIC PLAYING]

On today's show, the crucial skill that isn't being talked about nearly enough in software engineering.

I had one company who told me a story of a student with a master's degree from a very prestigious place. They were asked to add some functionality to a system that they already had. They added the functionality, but in the process, slowed down the system by a factor of 10.

Charles Leiserson, a legendary computer scientist, teacher, and advisor to businesses, talks about why companies and universities need to think a lot more about optimizing performance because, in a post-Moore's law world, there's real power in bringing in people who know how to do just that.

I think the real thing that's concerning is that for years, people said performance didn't matter in software, and that's not true anymore, and yet, people tend to live with their old myths.

But here's the paradox-- when you make compute power more efficient and cheaper, people want more of it.

I think this is particularly concerning when we're in an AI arms race with some of our national adversaries, where is it worth throwing more into the atmosphere in order to make sure that we're better than the other? And they're thinking the same thing. It's kind of mutually assured destruction if you will.

Join us today for a discussion about the under-the-radar force that's going to drive software advances and that forces unintended consequences.

[MUSIC PLAYING]

A few years ago, Charles Leiserson co-authored a paper in the magazine Science. In it, he tried to figure out how you continue to improve computer performance now that Moore's law is over. The law is named after Intel co-founder, Gordon Moore, who predicted in the 1970s that transistors on an integrated circuit would double every two years. But now that that's history, companies cannot exactly sit back and watch performance skyrocket, which for five-plus decades, they could.

Like every two years, you have as much computing as you have in the whole history of computing.

In many ways, the pressure is on because we're at the beginning of a revolution in machine learning and artificial intelligence. That revolution came from algorithms and from large pools of labeled data on the web, but it also, more recently, came from enormous and growing compute power, and those three forces had to come together. So is computing productivity in for a rougher patch than most people believe?

The type of innovation is going to have to change because we cannot simply rely on having additional computing power for the new things we would like to do. There will be some improvements from where we are now in capacity, but not at the relentless pace that Moore's law gave us.

Think about transitioning from working life to retirement, says Leiserson, who's a Professor of Computer Science and Engineering at MIT. When you're working, you can try to keep getting your salary to increase, but when you're retired, large increases in income are unlikely. So instead, you try to squeeze the most out of what you've got.

So doubling the capacity every two years, I don't know how much your salary is going up, but I'd be happy with that.

That's amazing. Just think where you'd be in 10 years.

Just think-- 10 years, exactly. So this is really dramatic, going from this relentless pace to something that's just a crawl.

Because compute power used to come so easily, he argues, we have a lot of legacy bloat in the way we do things. In that science article, he and his colleagues tried to figure out how much. So he took a common operation, a naive triple-nested parallel loop for matrix multiplication, and tried to figure out how much he could performance-engineer it.

So the first implementation in Python took about six hours to multiply 4,000 by 4,000 matrices. We optimized it, and in the end, it was less than half a second, which turns out to be like a factor of 60,000 or something.

It's a substantial improvement. Six hours, you said, to under a second.

From six hours to under a second, under half a second. Not that the extra two matters at that point. You're not going to see that on most applications, but there's substantial opportunity.

What Leiserson worries about is that many people don't yet realize how critical software performance engineering is going to turn out to be, which is what that science article was about. Not enough students are learning about it. Not enough businesses are fully attuned to it, although, he finds, some can already see its importance.

So for example, the trading companies, they care about it. The oil exploration, a lot of places that have things. And some people say, well, why don't you go use a supercomputer? And in some sense, the answer is yeah, but most computation these folks do is on laptops or workstations or in the cloud. They're not using supercomputers.

A very tiny percentage of people actually use supercomputers for very important things, but they need performance out of what's economical and cheap. Supercomputers are not cheap.

Leiserson and his colleagues have found that those with a software performance engineering background have almost identical salaries to those in machine learning even though machine learning gets a ton of attention and software performance engineering really doesn't.

In particular, on the recruiting front, companies that I have talked with talk about how they need to hire people and they can't hire people who've graduated with any understanding of performance. And as a consequence, most of the people that they are hiring to do performance-related things end up either being in their 50s and 60s because they grew up in an era where performance mattered more--

Interesting, OK.

--or they have to train them on their own. And so they're looking, and they do not have a ready supply of people from universities who have these kinds of skills.

Do you feel like when you talk to CEOs or CTOs, is this one of the big problems that they come to you with? And then I also wonder are there a couple more, like what are the sort of top things people say this feels like a real issue to us?

Yeah, I would say I'm hearing it less at those levels because since software performance engineering is not, today, a legitimate academic discipline, it's not recorded in the database as a category for all these jobs, et cetera, it's hard for them to articulate that that might be what they're after.

But when I talk with the people who are more on the front lines of computing, this is very, very common to hear how come people are coming out and have never used any kind of software performance engineering tools like a simple profiler. They just don't know.

I had one company who told me a story of a student with a master's degree from a very prestigious place. They were asked to add some functionality to a system that they already had. They added the functionality, but in the process, slowed down the system by a factor of 10.

And when they tried to talk with-- this was a student intern who'd come in. When they tried to talk to the student about it, the student said, but it works so what are you worried about? It works. It does what it's supposed to do. It's like, well, no, we have to do it efficiently and we have to do it in response, sometimes, to a real-time demand or whatever.

But the hard part was they couldn't do that. When they trained people in-house, one of the problems is that when hiring, they don't know if people don't have any background in it, whatsoever, they don't know which people have aptitude for it. And so it ends up being very, very concerning.

Now I'll tell you, however, that's mostly the application side. On the server side, many of these companies have lots of people doing software performance engineering, like people who are providing cloud services like Google or Facebook or what have you. They understand the value because if they can save a small percentage of time and computing something, that's that many fewer servers that they have to deploy.

I wonder if those companies-- I don't know if you would put Amazon Web Services in there. I wonder if they are gobbling up the people who know how to deal with this and leaving for the rest of the industry much slimmer pickings.

Much slimmer pickings, yes. In fact, this is-- one of the companies that I talked to, who was in oil exploration and such, commented that they can't get the MIT students who know about this stuff because there just aren't enough of them.

Does it frustrate you at all that AI, machine learning just gets tremendous-- the media attention and the headlines are tremendous?

Oh, not at all.

It doesn't? No, this is good.

In comparison. That's good.

This is good. This is good. And in fact, it's actually the AI and machine learning and so forth which is a huge area of opportunity for software performance engineering. There's, for example, now at NeurIPS, which is one of the big AI conferences, there are things like MLSys, where people are starting to focus on, well, how do you actually build the systems, and in particular, how do you get performance out of them and such. So I don't bemoan other people's successes. And in particular, usually, if you look, it turns out, well, they need you as much as anybody else.

And do you think academia and business are catching up to how much you feel like software performance--

No.

--is needed?

No.

No, they're not there yet.

No, we're so far behind. And it's hard. It's hard for a lot of things. There's no textbook in the area. There are no agreed-upon things that you should teach. There's no academic community devoted solely to that. So there's really a lot of, right now-- and as I say, it's because it's balkanized.

Everybody's doing their corner of it so they're a minority in the space that they're working in, but there isn't something has pulled everybody together. And that's something I'm currently working on, but that's a big mountain to climb.

This is kind of related, but as I've heard folks at CSAIL talk, one of the things that struck me is the percentage of people with degrees in CS but also like PhD-- undergraduate degrees but also PhDs in CS who now go into industry-- huge numbers, and that's gone up over time.

In some ways, that's really great. People can make a lot of money, that's wonderful. Do you worry at all-- and this is a little connected to what we're saying-- that that leaves fewer people to teach the next generation of people who need to be really good computer scientists?

Yeah, that's certainly been-- the way it's been in academia, you do sacrifice-- you have to have a good reason to be in academia. I like the freedom. I like the freedom I've had to go consult elsewhere, do entrepreneurship. I like teaching. I'm from a family that believes in service, and service makes me happy.

When somebody comes up and asks me to sign my textbook, that makes me feel good. So for me, this is very good, but I can see the challenges. And certainly, we've seen a large decrease in the number of students who will stay over the summer to work on academic kinds of things. It's just too tempting when the job offers are as big as they are these days.

What has happened in the past is we've had a cyclic economy, where when business isn't doing so well, suddenly, we get a big influx of people who want to spend time in the university. But as the Fed and other people have better managed the economy over recent years, we don't have recessions the way we used to. It's not as much boom and bust.

And so there's been a more tendency to just have industry be stable and growth. Whether the end of Moore's law helps that, which has driven a reasonable fraction of our economic health in recent years, continues-- since Moore's law is over, whether that continues, I don't know. I'm not an economist.

Do you worry that that could slow things down more than the average person realizes?

First of all, I don't know that it will. People are very creative, so even with less-- you might say, well, there's less opportunity. Sometimes necessity is the mother of invention, and sometimes, that's exactly what prompts somebody to do something that is creative and original, as opposed to yeah, it's easy so there's no point in being clever.

We talked a little bit about AI and machine learning. One thing I know that you've talked about and a lot of people are thinking about is as we see the rise of systems that need more compute power and more power-hungry applications, there's a lot of carbon output associated with running these really, really complicated models. Do you worry about that and do you think businesses realize that yet?

I think some businesses realize this, but they're kind of in a tragedy of the commons when it comes to how much money should they put into computing that causes large amounts of carbon emissions. If a large company says, oh, I can improve my penetration, my sales, or whatever, by 1% by running some big AI training model-- so they're only getting 1% more-- they look at that, the economics are really good, whereas the cost of what they put into the atmosphere in terms of carbon is borne by everybody, so there's no motivation for them not to prefer more money over everybody else because everybody else is paying. It's a free ride.

So I think that's a great concern, that without any steps to stem that, and it being a tragedy of a commons type problem, it's not something that has a technical solution. This is about economics and politics and so forth. But I think there are things that we as technologists can do. Namely, we can do things like instrument our software and our hardware so that you can figure out how much energy, or whatever, power, et cetera, you're using.

And when that happens, if you were to do that, then at least people would be able to see what they're using, possibly.

Versus it being behind a curtain and they don't realize.

Behind a curtain or-- and policymakers could then do something because they know what the data is, rather than just saying, well, let's all try to do a little bit better. I think this is particularly concerning when we're in an AI arms race with some of our national adversaries, where is it worth throwing more into the atmosphere in order to make sure that we're better than the other? And they're thinking the same thing.

It's kind of mutually assured destruction if you will, except that unlike with nuclear weapons, this is where it's not even deploying the technology, it's just training the technology. That's where you have some real concerns.

I don't know if this has come up as you've talked to folks in business, but I would think it would be a little bit of a dilemma for people of-- as you said, you can optimize sales a little bit more, but people are also thinking about ESG and carbon footprints. And I mean, they also have a business interest in not polluting the atmosphere because they told their shareholders or their whoever--

Well, that's good--

--we won't.

--if you could tell how much, but here's the thing-- all things being equal, if you can compute something twice as fast, you just saved half the energy if you're using the same resources. Unfortunately, the world doesn't work like that because if I can-- for example, if I told your gas price-- price at the pump for gas was halved, what happens?

People go about running gas or they buy gas-guzzling cars. They always do, right?

Right, they basically drive more and are not so concerned about buying an electric car or a Prius or what have you. They--

Exactly. And we always see that with the data. Gas prices go way up and people are buying the Prius. And it goes down--

Same thing with computation. OK, you make it more efficient, you lower the cost, then people are just going to try to use more, and that's the risk. On the other hand, if you're actually measuring how much people are using, you can do things like the energy companies do with your home bill, where they send you a report on how you're doing compared to your neighbors. And that turned out to be extraordinarily effective in having people reduce their own use because why?

They can quantify it and they can tell you where you are compared to other people. And then people who are civic and community-minded, say, oh, we can do better than this, without anybody leaning over them or what have you. But also, policymakers can make decisions if we have things instrumented. But right now, nothing's instrumented. You can't tell where in your software you might change the coding in order to use less energy.

When you think about your field is this the thing that's most concerning to you?

This is a thing that's very motivating-- one of many motivating things. I think the real thing that's concerning is that for years, people said performance didn't matter in software. And that's not true anymore, and yet people tend to live with their old myths. And so therefore, we're not on a path to really addressing any of these issues unless universities start to recognize they've got to be teaching performance to their students.

The funding agencies, you've got to be supporting how we get more energy-efficient software and faster software. And that takes a lot of big institutions-- institutions, they move like aircraft carriers. You want to turn them, it takes a few miles before they can turn.

So that leads me to my last question or two about education. I know it's a really big question to ask because computer science education goes over many-- there are people who take it in high school and there's people who take in college and graduate school, and so on. But how do you feel like we're doing?

I mean you've been somebody who's not just been developing new things and publishing papers, but you've been really-- you've cared a lot about education over your career. How do you feel like we're doing with computer science education? Not at MIT but like the big, big broad picture.

Let me compare that with where I started. So when I started, there were famous computer scientists who said, I wouldn't hire somebody with a computer science education. I'd much prefer to get somebody who had a math background or an application area, psychology, or whatever because they disparaged computer science.

They were famous computer scientists but they didn't really believe in--

But they didn't believe in the-- right, they didn't really believe in computer science. And it was like, if you think about it, that's a compelling story for somebody like, here's a computer scientist who says don't learn about computer science. Learn about other things.

And when I was a graduate student and, early on, the faculty here at MIT, I said, well, that may have been true 15 years ago because we hadn't developed a lot of the things that we developed soon, but at this point-- and this was, I would say, by, say, the 1980s-- there was a lot of really valuable stuff to know in computer science, and it was intellectually deep and it was important to know if you're going to try to do any computationally interesting research or task or what have you.

Today, it's even more so. You just look at how this area has burgeoned, and yet, when you look overall, most universities, computer science has very, very little political power despite the fact that they have most of the students. And everybody looks at them and say, well, computer science is such a big beast in various universities, including here at MIT, and yet, they don't have that same kind of representation.

It reminds me of taxation without representation. Yeah, we have learned an enormous amount in computer science. We have amazing results and contributions people have made. Everything from how do you safely transmit financial information over the internet, to inventing the World Wide Web.

Some people say, oh, the World Wide Web, that's something that just emerged. No, that was invented by Tim Berners-lee in our lab. It wasn't something that just happened. It's something that a computer scientist-- although he's a physicist at the time, but essentially doing computer science-- invented.

And you can see that all over computer science, machine learning, all of these things are things people should learn about. It's really very exciting. And I think it would be more healthy to have more people understand computation in other fields.

That's really interesting, yeah.

Because I think every other field of academic research, you look at it, computing has had a tremendous impact. We analyze Shakespeare with computers.

And then thinking about biology and medicine, so transformative.

Yeah, it's like math, but unlike math, where you have to do the work, here you can have somebody build a tool that does it for you. And so it just builds on itself and it's so economical, and so forth. It's really quite remarkable. I didn't come into computing because I thought there was going to be a big future or something. I came in because I liked the computer. It was fun to play with and to code, and so forth, and I liked the math that went along with it, but I got really lucky.

A final math-related question for you, which is, you wrote this big textbook that's been incredibly popular.

Million copies.

A million copies about algorithms. Who knew? But very popular. Why do you think it's been popular for like 30 years, right?

For over 30 years.

Over 30 years, OK.

Well, there are a couple of things. One is that we hit the window just right when an algorithms book of that nature was needed. And we decided to be comprehensive but still make it introductory. So that met a niche at the time.

The real growth in it turns out to have come along with-- frankly, I can't take any credit for it. It comes along with machine learning and AI. That as more people have decided they're going to come into computer science, they all need to do algorithms. And so the large enrollments in computer science, and while they-- and they all need to take an algorithms course, and this is the best book on the market.

So we get the benefit even though they may not, per se, be interested in algorithms. I always have to temper any of my pride in what I think is one of my favorite contributions I've been able to make with some humility, that, "You got lucky there, Charles. You got lucky."

I think I heard you've dreamed-- you'd always dreamed of writing a textbook. Is this right?

I had, yes.

A common dream for a young man-- writing a textbook.

Yeah, when I was in high school, I thought I would write a geometry textbook.

OK, well--

And I used that as the subject of an essay for one-- for what was essentially humanities school, and they said, nah, we're not interested. [LAUGHS] They said, what book would you write? And I said, OK, it'd be a textbook. And I think they weren't impressed. [LAUGHS]

Well, it's done. It's done incredibly well. Charles Leiserson, Professor of Computer Science and Engineering at MIT, thanks so much for joining me. This was great.

Thank you. It's been a pleasure.

[MUSIC PLAYING]

And if you're interested in learning more about the CSAIL Alliance's program and the latest research at CSAIL, please visit our website at cap.csail.mit.edu. And grab this podcast on Spotify, Apple Music, or wherever you listen to your podcasts. I'm Kara Miller. Our show is produced by Matt Purdy and Nate Caldwell, with help from Audrey Woods. Tune in next month for a brand new edition of the *CSAIL Alliances* podcast and stay ahead of the curve.

[MUSIC PLAYING]