MITOCW | MIT6_172_F10_lec01_300k-mp4

PROFESSOR: Welcome to 6.172, Performance Engineering of Software Systems. This is a fun class. It's all about going fast. We all like to go fast-- well, not all of us. But everybody in this class likes to go fast. Otherwise you won't like the class very much.

It's all about making stuff go fast. So a couple of brief announcements. We're going to be videotaping for OpenCourseWare. So if you don't want to be videotaped, you should sit it in a place where the camera won't catch you.

Because we're in such a large classroom, I'd like everybody to sort of every day, without me having to draw-- I'm not really an artist. My sister's an artist, actually. She's a painter. But maybe that gene escaped me. It went that way and not this way. So I don't want to have to draw it up again. So everybody, try to confine yourself to this sort of quarter of the classroom. And that way, it'll feel more like a size room that we ought to be in.

Let me go to our course information sheet. Did everybody get a copy of the course information handout? So we're going to the class-- we're going to start at the beginning with the Administrivia stuff, try to run through that as quickly as we can. And then my co-lecturer, professor Saman Amarasinghe-- if you can't pronounce or spell his last name, you can just say "Saman"-- will give today's lecture.

And we have four teaching assistants. So teaching assistants, please stand up. They are Josh, John, Reed, and Eric. And let me tell you, what they have done in the last few days to get us ready for this course is nothing short of superhuman. So you really need to accord them your respect, and they're going to really work hard to help you folks do well in this class.

We have, I think, a pretty decent ratio. I think even with concerns about cutbacks, et cetera, to have four TAs for this is really good. And we need it, because it's a software class. And software is, as you know, bugs can take an arbitrarily long time

to find. You guys can sit.

Besides our teaching assistants, we have two support staff people helping us, Mary McDavitt, who works for Saman, and Marcia Davidson, who works for me. And they're generally pretty good about being able to track us down if we're in some far corner of the world or something. Our lectures are going to be held on Tuesdays and Thursdays in this room, 2:30 to 4:00 PM. You should plan to attend regularly. We dispense all kinds of wonderful gems that you will miss if you think you're going to get it just from reading and even looking at videotapes or whatever.

I'm not sure what our arrangement is going to be about having videos available. OK, they're not going to be available, except unless there's something really extenuating circumstance. You have to go into a hospital to have your forebrain removed or something-- something serious, OK?

All material in the lecture, even if it's delivered oral, is fair game for projects and quizzes. We will post the lecture slides, but they don't generally convey all of the information. And we're going to be video-recorded, as I mentioned. If you don't want to appear, sit behind the camera, and you won't appear. We have no recitations in this class.

However, the teaching assistants will periodically hold primers where we will bring you up to speed on various technologies that you may not be up to speed in, such as C. How many people have programmed in C before? And how many have not programmed in C before? A few. OK. Tools such as VTune. Who's used VTune before? Who has not used VTune? Good.

Pin-- who's used Pin? OK. Who has used Cilk++? OK, a couple people. So you see, we're going to have some primers to bring you up to speed in those kinds of things. We don't expect that you came in knowing any of those things, particularly.

We're going to be using Stellar for our course management. And here's the URL. And you should make sure that you're registered as a member of 6.172, because otherwise, you're not going to get any announcements. We're going to do all of our

administration through Stellar. So make sure that you're registered there. At the end of today, I think it is, we'll send out an announcement to the student mailing list. If you don't get something by the end today-- so you wake up tomorrow morning, you have no email from the course staff. What that means is you should email us.

AUDIENCE: Well, that mailing list would be the preregistered ones.

PROFESSOR: That'd be the preregistered ones. And then that way, we'll get all the people. So if you know you're not preregistered, you can send us email before then. There are two one-hour quizzes given during class time.

AUDIENCE: One and a half.

PROFESSOR: What's that?

AUDIENCE: One and a half hour.

PROFESSOR: Oh, you're right. It's one and a half hour quizzes. Yes, two one and a half hour quizzes. We'll correct that in the online version. Given during class time. Can somebody, by the way, correct this as we go along? Great.

The dates are on the course calendar, which is available on Stellar. We've had mixed results with the ICS feed from Stellar, but you're supposed to be able to download the calendar to your iPhone or other ICS-capable devices. But we were successful in downloading, but all the times were screwed up.

But anyway, hopefully, maybe they'll allow that. But anyway, the calendar is there. The quizzes will generally be closed book and closed notes, but you will be permitted crib sheets, one sheet that you summarize anything that you feel that you need to know.

Generally, everything that we cover in here is fair game for the quizzes, including things that are in prerequisite courses. There is no final exam because we have a final project.

This is a fast-moving class, so if you get caught behind, it's not a good sign,

because it's really hard to work double-hard to keep up in this class. You really have to keep on top of things. And so for that reason and others-- because it's really difficult for us-- it's fast moving for us as well-- we generally cannot accept late projects. If you haven't been able to complete a project, you should just package up your partial solution and hand that in, and you'll get partial credit on it. Do not just say, oh, I'm not going to hand something in, and expect it to get it done later.

If you do find yourself in an unusual situation which you think there are some extenuating circumstances, like the aforementioned brain surgery or something, let us know in advance. So if you're going to get in a car accident, it's really helpful if you tell us the day before so that we can plan. Obviously can't always do that, but if it is something for which there is any reasonable expectation it should be done beforehand, you should let us know. And we may require, for those kinds of exceptions, some kind of confirmation from a dean or medical professional. So if that's the route that you're in, you can plan a little bit and make sure that those confirmations are on their way.

The grading, this is essentially approximately how we're going to grade. As you can see, there's a bunch of content in the projects and a bunch in the quizzes. And then there's also participation grade. And the participation is both in-- we will have design reviews and such, so that's part of the participation, as well as asking questions and such in class. And because we're video-recording everything, we know exactly who's asking questions, unless you choose to sit outside the video zone.

In addition, if there is a significant missing element to your work, it doesn't matter how well you did on the other things, you will receive a failing grade. You must, essentially, do all the work. So if you do not get substantial credit on the final project or any other two assignments, for example, you can expect to receive a failing grade.

It goes downhill very quickly. If you're missing one assignment, generally don't expect to get higher than a C, for example. It goes downhill very, very quickly, because that's what this is all about is hands on.

What's permissible in this class? So it's your responsibility to satisfy both the letter and spirit of the rules. So if you have any questions, let us know. Here's what the rules are. Also, let me just say how Saman and I treat this. We have been involved in cheating incidents off and on over the years. I've been at MIT almost 30 years. Saman, you've been here--

PROFESSOR: 13.

PROFESSOR: 13 years. So we've been here long time, longer than you folks. We've been through this a bunch of times. It's really a pain when somebody cheats. It robs me of a huge amount of my time that I could otherwise be spending on teaching and students and my own professional development.

However, we both take cheating incidents extremely seriously. So we have this thing that is sometimes really inconvenient called integrity. And I'd rather, if I find somebody's cheating, just give you an F and not deal with it. Drat. Instead, when somebody's cheating, I tend to want to take it to the Committee on Discipline, where you face things like expulsion, et cetera.

So I don't like to do that. It's very time consuming for me. But I find it hard to just look the other way, because I view the issue of integrity as protecting the vast majority of students who are obeying the rules. So when a cheating incident occurs, I sort of feel honor bound to push it to its proper conclusion.

So please don't cheat. We will use a variety of means at our disposal, including technological ones, to detect cheating. Also, if you think that sharing your results with somebody else means that they're cheating and you're not-- in some cultures that's the way it's treated-- no. In this culture, in particular the culture in this classroom, they're both equally guilty of academic dishonesty, whether you are the giver or the taker. Equally, we treat that as equal offense.

So here's the particular things. You may not share, generally, your solutions with people who are not in your group. So we will have a bunch of group projects, generally in pairs. And that includes both people in the class and people outside the

class. So generally, you keep things to yourself. When you're in a group, of course, you can share ideas within the group. But we expect that everybody's making a fair contribution.

So you should be concerned if you are not holding up your end of your group participation. And we're going to ask you to describe the contributions of the people in your group. Generally, the group, as I say, is two. So it's basically saying, what did you do? How did you divide up your project? What did you do? What did the other person do?

You generally can't share it with anybody else. You can't copy or transcribe solutions from other places. The work you submit must be your own. Pretty straightforward. You may go out and use general conceptual material, such that you might get from a good textbook or from Wikipedia or someplace like that. That's perfectly fine, resources you would ordinarily have available. But if you do use any material from an external source, you should properly cite it in normal academic that fashion. Cite where it is so that somebody else can go and find and look to see what it is that you're citing.

If you feel that you have transgressed in some way, let us know. It always goes way easier. I generally tend to be kind of a nice guy when somebody comes to me saying, oops, verses I hear from somewhere else or one of our tools detects something else. Then I'm kind of cranky. So I'd rather, if you come to me, we could work something out, generally.

Most of your out-of-class time will spent completing six projects. And there's an outline of what the projects are. They're all fun projects. You can ask the TAs. The TAs will tell you how much fun they are, because they've generally helped to design and built their own solutions to them. Some will be done in pairs, some individually. The final project can either be done in pairs or in threes, I think we said.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yes, we also have-- we didn't put that in here. We also have a Project 0, which will

be available today after class. So you should go to Stellar and get Project 0. Project 0 is not to be handed in. It's getting warmed up with all of the machines and tools that we'll be using in the class. We have been very fortunate to have very nice contributions from Dell and Intel of a new Cluster facility that has 16 12-core machines. And they're Intel Core i7 machines.

Each chip has six cores, and there's two cores per machine, each with 48 gigabytes of memory. So these are kind of honking modern facility. So as I say, it's fun. It's fun when you do something on your laptop, and then you go and you do it on one of these machines and it goes like this. It's really fun when these things go fast.

So Project 0, you should start on today. And it's basically the handout called Getting Started. It will be available on Stellar. And it will take you through getting access to these machines and so forth. In particular, in order to use these machines, you must have a CSAIL account. Who does not have a CSAIL account? So that is going to be the first thing you need to do, because you've got to get that.

So what you want to do is get registered for a CSAIL account-- and we give instructions in there. It's basically going to a web page-- pretty much as soon as you get out of the classroom. Because you won't be able to start anything until you have this CSAIL account. And hopefully, we'll be able to give you access within a few hours after you register. Yeah, question?

AUDIENCE: What operating systems are the tools--

PROFESSOR: Linux. We'll be using Linux. So there are a variety of ways that we describe for how you use these machines remotely. We have the Andrew File System-- AFS-- which you can use so that you can edit things locally but have an SSH terminal window on the machine, which makes it very convenient. On the other hand, for some people, often some dorm rooms with poor connectivity, doing things by AFS may be too slow for you.

But there are a variety of other ways that you can-- you can actually edit right on the machine. And there are a variety of ways of having a repository on your machine,

because we'll be using Git as our versioning mechanism. So you'd be able to have it there and be able to make copies back on the other machine.

So there are a variety of ways to work. And you're free to make the software run on your own laptops or workstations. The one thing I'll say is we will not support that. We don't have the resources to support beyond using the machines that we actually provide.

I think I've got to go faster here. So let's see. So software engineering. So research in programmer productivity has demonstrated that pair programming, where two programmers sit together, one at the keyboard and one looking on, is actually faster than two programmers coding separately. Why do you suppose that is? Yeah?

AUDIENCE: You don't get stuck. The other person can tell you-- it's like when you're [INAUDIBLE] writing on the blackboard, you get nervous. You don't know what you're doing. [INAUDIBLE]

PROFESSOR: Yeah, you could be stuck. What else?

AUDIENCE: You don't procrastinate.

PROFESSOR: Yeah, you don't procrastinate. That's a good one. Don't procrastinate.

AUDIENCE: Finding bugs.

PROFESSOR: Yeah, finding bugs. So the main one is finding bugs, that the person who's looking on finds bugs. In fact, I was just pair programming with Reed the other day, and we were trying to figure out, why is this thing not working? And I say, gee, it looks fine. It says, next y equals x. And he says, no, next y is supposed to equal y.

And it's like, we had both read over that code, but as soon as I spelled it out, I didn't even notice. But of course, him in hearing it says, oh, it's supposed to be x and y and y, not x and y. So anyway, that kind of thing, you pick up very quickly. It makes it very easy to debug.

Likewise, regression testing demonstrably promotes fast co-development. So this is

where you build a battery of tests that includes tests for every mistake that you've ever made. And it can also be helpful to write unit tests, where you test individual ones. And you should also use tools like the assert package. And some of these things will be taken through in Project 0.

The MIT POSSE. We have recruited senior software engineers in the region to share with you their invaluable knowledge and experience. We call them Masters In The Practice Of Software Systems Engineering, which has the great acronym MIT POSSE. These are people who are not being paid. And they're senior. They're doing this because they're volunteering, because they want to help.

You have a lot to learn from these people, and we will have code and design reviews with these people with your assigned master. You want to treat them nicely. They're not doing this so that some snotty-nosed kid can say, I don't care about you. That's not what they're doing. They're there to help you get a good grade.

And these are some really talented people, people who are very famous in their own areas of expertise. So be gracious, is the main thing I would say. Be gracious.

Let's see. The assignments. You can read about the assignments. Basically, generally, what we're going to do is have a beta submission in which we're going to test everything. And we're also going to take all your tests. We're going to throw all the tests into a big pool and run everybody's tests against everybody software. You will lose points if you fail some of the tests, assuming the tests are correct tests. You will lose points if you have an incorrect test.

You will lose points if somebody passes your test and they have a bug. You will get points if your test picks up something that very few other groups tested for. So you have to both submit code and submit tests, which is OK, because you've got to test your own stuff before you hand it in. But we're also going to be using it to test each other.

After beta, we will have a design and code review with your master. And let me just say, the master's there not affecting your grade in any way. They do give us reports

about what happened in their meeting, but they have no bearing on the grade. They're just there to help.

So you don't have to view them as-- they're not part of the course staff. If you want to get angry at somebody about how things are done, get angry at Saman and me. Don't get angry at them.

The final submission is generally due a couple weeks after the beta. And it's after you've gotten feedback on your design, feedback on things. We'll give you the whole battery of everybody's regression tests, so you can test out your code against everybody's regression test and use that. And then generally, the scoring is such that people who get fast code get high marks. People who get slow codes get lower marks.

If you need help, [UNINTELLIGIBLE]. Speed is fun. Enjoy the class. Yeah, question?

AUDIENCE: In terms of the bug testing, I'm thinking of bugs like-- I don't know-- [INAUDIBLE] or something like that. If you happen to not have made a bug in your own program [INAUDIBLE] would that count against you in the final testing.

PROFESSOR: You have to view your beta test as testing other peoples' code also.

PROFESSOR: But if you [UNINTELLIGIBLE] a test that finds bugs in somebody's program, and if very few people wrote that test, then you will get points. So you don't have to test for the world, everything. If you fail somebody's test, but not yours, you will be deducted points. If your test finds other people's bugs, you will gain points for that.

But on the other hand, if you don't have a test for-- what you said, I think you don't have to go and test for every possible condition in there. The negative side is, if you miss something, we will deduct points. Positive side is, if your test finds somebody else's bugs, you'll get points for it. We'll go into detail exactly how [UNINTELLIGIBLE].

Another thing we did last year, which was kind of interesting-- I think there were mixed-- some people really liked it, some people were not that happy-- is a little bit

of competitive grading. So here, what we are doing is not the full grade-- small part of the grade. What you do is when you first submit your beta, rerun performance tests of everybody.

And of course, one of you are going to get the fastest code for that. At that point, we are going to say, OK, this is the fastest code. And how [UNINTELLIGIBLE] everybody else with that code? And the person who gets fastest code for that part of the grade gets the full points. And everybody else will get a partial part of the points.

However, you get a chance in your final submission to basically improve your code. If you do better than the last time's best code, you won't get any more additional points, because then we don't want to create [? an arms race. ?] So you at least had to match the previous times to get full points. So you get a chance to see how you stand against your peers and get a chance to actually [UNINTELLIGIBLE] math with your peers.

So first time we did it, we said, OK, it's the fraction of your performance against the performance of the best code. The problem was, there were a couple of codes that 1000x performance improvement, and everybody got 10x. So you've got, like, 0.25 grade. That didn't help.

So now we actually are doing a logarithmic log scale. So we'll get the fraction, and we'll take the log off that. So there will be a little bit more equality. So you will know how worse off you are, and you get a chance to [UNINTELLIGIBLE]. Because it is very important, because a lot of times, there might be a crucial thing that you missed. And instead of trying to be, "aha, got you" in this class, we are trying to give you a chance to actually learn.

So if you missed it, OK, you missed it the first time. But you get the second chance to come and fix it and see whether you can do well. And of course, if you do better than the best grade, there are bragging rights, but you don't get extra points.

With that, let me-- where are we? So again, as Charles pointed out, I'm pretty

excited about this class. This is a really fun class. Unfortunately, I'm a little bit under the weather. So I'm going to sit down. And if I don't sound that enthusiastic, hopefully I'll get enthusiastic later.

And that's one reason we ask you to come forward, because we don't have to come here and shout. So hopefully people in the back can here. If you don't, still, there are some seats in the front. You can move forward.

So for the rest of the class, what we want to do is go through an example and look at why performance matters. For that, I selected matrix multiply. And I think your Project 0, you are going to also play with matrix multiply. And I think that you'll get a feel for what kind of things will happened and how much room you have in doing well in here.

And at the same time, as I go through this, you'll get a feel for what's the process that you might be following through the class and what kind of things-- because linear programming, after taking a bunch of programming classes, probably, you have a methodology set up in your mind, how to go about doing that. And the same thing after this class, we want you to have a methodology of going through and improving performance. And this is basically a little bit of a contrite example, but you'll get a feel.

So what's matrix multiply? It's a very fundamental operation for many computations. So you are going to produce Matrix A by multiplying Matrix B and C. So you want to calculate each value in here to take a row of B and a column of C, and you get each element of A. And you do that for every element of A. And this is the possible simplest code you can write in a language like C, a [? three-loop nest, ?] and this matrix multiply.

So, so far, things should be pretty simple. So what I did was I looked at matrix multiply and said, look, you have taken software engineering class. You have thought about object-oriented programming, lot of nice techniques to build very useful, portable software. And let's try to write matrix multiply with some of those concepts.

So the first thing I want to do is matrix multiply write object oriented using immutable classes, and that, in fact, I want matrix class to represent both integers and doubles. So here's my class structuring here. And actually, I'm going to show you a bunch of code now, next couple of slides.

I'm not expecting you to follow everything instantaneously. I'm going to go through somewhat faster. But you have the slides. So I always expect you to, perhaps, in some free time, to go and look at some of the slides, get a little bit of better feel for what happened.

So here's what matrix multiply does. I have a value class that figures out the matrix type. And if it's integer, it keep integers. If it's a double, I keep a double. And here, basically, do this value in here. And then I have a matrix class. Basically, given a matrix type and a matrix, it can create a matrix here. This basically creates your matrix in here by first creating a bunch of rows here, and then instantiating each row in here, whether it's integer or double, depending on the type you want.

And then, of course, there are other things in matrix, basically, in here. And you can update a matrix. This is going to be a little bit interesting. I will give a little bit here, because what I'm going is I'm doing immutable types. That means I can't change anything in the matrix. If you are updating, you have to create a new matrix.

So I'll show a little bit about how to go about doing that. And then there's this row class. I'm going through this fast, because it's not that important. If you get a chance, go through this and get a feel for what I'm doing. But the idea is doing this very software engineering way of implementing a matrix multiply.

So then here's my matrix multiply in here, that what I'm doing is I'm updating each of these matrices by giving a value of B and C and adding it to A and updating A again in here. So when I run matrix multiply, 1,024 by 1,024 matrix, this is the time I got. Actually, it took a long time, in fact. And the key thing, is the performance good?

That's the first question to ask. Because you run, you get a result. When the result is correct, OK, great. You get the correct result. But how do you do? And this is

sometimes a very hard thing, to know that you have a problem is itself a huge win-- that you figure out you have a problem.

And what I'm going to do is go through a little bit of a calculation to see how we did it. It took about five hours to do this matrix multiply, and that doesn't look right already. So if you think about matrix multiply, it's an N-cube operation, basically, what matrix multiply is doing. That's the algorithmic [? class. ?] If you look at Charles' book, that's what it would say. It's N-cube operation. So this is the number of operations you want to do.

And for each matrix operation, you only do three index updaters, add, a multiply, and a check-- probably six ops to do, basically. And that means I need to do this many operations. Basically, you can think about machine operations to do, basically, because you have to read three values, update one, do a multiply and add, and do a branch.

And if I look at that with my hours, I am doing this many operations per second, which looks high. Big number. But my machine, actually, on my somewhat old PC, transfers even much faster. So if you think about it, I am taking about 8,300 cycles to do any kind of a machine operation. This doesn't look right.

So at this point, this is what I would call a "backup [UNINTELLIGIBLE] calculation." I haven't proven anything. This one worked well in the theory class, but if you can just jot down a couple of these things, you can get a feel for where you stand very fast. And then you realize, OK, I am a couple of orders of magnitude off here on what best I could do. And that should give you a feel for what's going on.

OK, so what's going on here? How can we improve performance? Because we know we have a problem. So what we can do is we can look deeply into the program execution.

And at some points, OK, this program runs slow. [UNINTELLIGIBLE] time spent. So we can figure out time spent, basically, things like by method-- which method you spend most of the time running. Sometimes by line. And we will learn a bunch of

profiling tools that will give you this feel. So instead of reading a million lines of code, looking at every line in there, you can very fast converge onto the culprits where most of the time being spent. And that's a very easy way of triaging your problem.

And you look at things like time spent, cumulative time spent within that and everything called underneath, number of invocations of the functions. And a lot times, you have a feel for what it should be. And when you look at this, if it doesn't look right, you say, wait a minute, this goes against my intuition. Perhaps there's a problem.

So what you can do is, by doing this, you can identify hot spots. Basically, if 90% of the time it's it one routine, that's what you want to fix. And hopefully, that routine is doing something too slow. Then you can actually get good performance in here.

So here's the profile I ran for a matrix multiply. And interestingly, what I found was, most of the time spent in basically creating doubles-- number of calls, time spent here, this much time was spending, basically creating new values in here, this initializing a double in here. So I'm doing matrix multiply. Why should I be initializing values? So this is an issue. And I have the most number of calls to that too. So we can go look at it and say, hey, that would seem to be a problem here.

So the problem here was kind of obvious. What happened was I didn't use immutable [? class. ?] So this is my matrix representation. I have a bunch of pointers pointing to each row. And now assume I won't update this element. And of course, I can't modify the entire thing. It's immutable.

So what I had to do is create a copy that updates. And then, of course, I had to create a new matrix. I don't have to copy everybody else, because these guys were not changed. So I can place a pointer to these things and create this new matrix that points to these things.

And then, if you think about what's going on, in fact, and this is my new matrix after I do that, I [? updated ?] two end copies for each update, because what I had to do is I had to copy this entire row up to get that. And I had to create this entire index

again for each point in each row. So just to update one element, I am creating two end copies.

So what that means is now our N cube matrix multiply ends up being N to the power of 4 algorithm. So suddenly, you realize, wait a minute, I'm doing something bad in here. I'm actually [UNINTELLIGIBLE] N-cube algorithm, and the way I saw, it becomes n to the power of 4 because I'm making all these copies in there.

So this is a bad thing to do, of course. And copying is very costly, because you're making duplicates in here. And of course, you're creating a huge amount of garbage, because you are just copying and leaving the previous value for garbage collection. So garbage collector's coming up all the time, and huge memory footprint. So this is no good. And can we do better? Hopefully, we can.

So the next thing we did was, OK, just get rid of these immutable types. That seemed to be a [UNINTELLIGIBLE] here. And then, I actually have-- I will go faster over the code, so if you are interested, you can go look at the code in here. So here, what we did was, still we have the issues like object oriented and ability to represent both integers and doubles.

So what that means is each matrix can be the integer matrix or a double matrix. So this is where you're instantiating with integers or doubles in here. So we do that. And how much you think this can improve performance by doing that? In fact, this was pretty nice. I actually got a 219x performance improvement because I went from n to the power 4 algorithm to n to the power 3 algorithm. So here is interesting, algorithm has changed with it, and we have a pretty nice performance improvement.

So if I look at this now, what I find is I'm spending a lot of method time, I'm doing matrix multiply. That's good. But I'm spending all this time doing get double and get methods. I'm trying to do matrix multiply. Why am I spending all this time trying to do get double and get?

What I found was issue is the method call overhead. Now, what happened is matrix row has two different matrix types, the integer row and double row. So every time I

want to get a row, I don't know if I have integer row or double row. I have to do a look-up to figure out my type. Am I working with the integer matrix or matrix in doubles?

And this is what we call "dynamic dispatch." So instead of knowing exactly the methods we are to go, I had to actually look up, OK, which method I'm supposed to go. I have to do that since, if the matrix is integer, I might have to call a different method than matrix is double. I have to check that.

And then that call, instead of direct call, becomes indirect branch. I have to check that and basically branch on that. And these indirect branches are costly, because what that means is I don't where I am going until I do that test. So no machine-- we'll go into details and pipelines and stuff as we go on. What that means is normally, a machine can have a lot of instruction in flight.

So that means that I have to know a lot more information to get the instructions going. But here, until you resolve the instructions, you can't do the next one. So it's called a, basically, pipeline stall. So instead of having hundreds of instructions going, basically, suddenly I can't do anything until the previous instruction is done. So the machine slows down a lot because of this.

Normally, what you can do is keep fetching next instructions here. We have to fetch, test, then the test result is available, then that's when you can fetch the next instruction. Let me go through that file. So direct batch means target address is known. You can fetch ahead of the targets/ If I know where I am going, even if I don't get there, I can fetch ahead of the target. I can go start fetching what happens next ahead, basically, into [? the branch. ?]

Sometimes, you can even fetch both directions, or the direction you might always go might get fetched by the architect. We'll talk about a lot of architecture details in the future lectures. Indirect branch means until I calculate the address, I have no idea where I'm going. That address is only calculated. And because of that, hardware slows down like crazy.

So this seems to be not a good thing to have these integers and doubles instead of having one nice class of just [UNINTELLIGIBLE] might be nice. I just basically create a double class. It's all matrices are doubles. If I want integers, I will just rewrite my matrix class or copy and replace.

Now my matrix class becomes much more simpler. This is what I got, and this is my row in here. And voila, minute I do that, I basically went up about a 2.4x performance improvement, because now I don't have to keep doing that.

And altogether now, I have improved my performance by about 500x here. And for ops per cycle, we have gone down from 8,000 to 38 to 16. So we do still better. And now we look at, again, my profile data. And what I see is, OK, I am spending a lot of time in matrix multiply. That's OK. But also, I am spending time in this matrix get method. I am trying to get each, basically, row and data elements of the matrix.

So here is the summary of these things. I'm going to just keep that one, because I gave you [INAUDIBLE]. So the problem in this object-orientedness is I [? handle ?] each row is represented separately-- nice objects. These objects overload the memory. And in fact, if I want to get to my row, I have to appoint a [? chase ?] to go get that. It's not contiguous in memory. And to get to every row, I have to actually go through this [? point ?] indexing in here.

[UNINTELLIGIBLE] matrices are one, nice, simple objects. So we can say, OK, wait a minute. This method call overhead actually keeps dominating. And because I have done this nice object-oriented thing in here, so I can get to the object-oriented file here and say, OK, get to the objects.

And I write this nice loop of [UNINTELLIGIBLE]. And I have two-dimensional matrices in here. And voila, I got another 2.2x performance improvement in here. OK, now I am down to about seven cycles for each operation in here. That's good.

So what can we do here? So we did all of these things in Java, because it's supposed to be a nice language. But Java has a lot of overhead. It's like you do memory-bound checks. You have to do this bytecode interpreter and stuff like that.

On the other hand, C can run much faster. You don't have to do any kind of memory-bound checks. The compilation happens before you run, so it is not part of your cost of doing anything. So this can work very much like C. You can just almost cut and paste.

And if you know, C is [? index chain, ?] [? and this ?] to C is just changing a few lines. And I did that, and this is, basically, the C code in here. This is the matrix multiply. This is the kind of allocation in here. C can be a little bit painful to allocate. This is my matrix multiply. It will look similar.

And I got 2.1x just by going from Java to C. This, I was surprised, because that piece of Java code looked very much close to C. But by just doing that, I actually got another 2.1x performance gain.

Now this is kind of the high-level stuff. Now we have to go to nitty-gritty details in the hardware. So in modern hardware, there are these things called "performance counters" that hardware [UNINTELLIGIBLE] that we can look at what's happening inside hardware. We can get a feel for what happens inside hardware.

And there are things called CPI, which clock cycles per instruction. So that means if the instructions are running very slow, you will have multiple clock cycles per instructions. That means instructions are stalling. Something is wrong.

Cache misses. So data [UNINTELLIGIBLE] [? pay attention to ?] caches and stuff like that. If data are supposed to be in cache and have cache hits all the time, but if you are getting cache misses, something is wrong. Accesses are not good.

And then also the instructions retired. That means, how many instructions are being executed and retired? If you are doing too much work, the number of instructions retired will go high in here. This will give you a few how many [? work been done. ?]

So if you look at this one, I have a CPI of 4. That means for every instruction, need about almost close to five clock cycles, which is not good. I have pretty high cache miss rate, too-- L1 cache miss rate and some L2 cache miss rate. This shows how many instruction are in what they call [? vector in the ?] instruction form called SSE

instructions. And here's the number of instructions got retired. So we just use that baseline case.

So one thing I can look at was my cache miss rates seem to be pretty darn high. What might be happening here? So if you look at matrix multiply-- so I have Matrix A in here and Matrix B-- what happens is, to calculate this value in [? Matrix A ?] here, basically, I'm going to go through this row in here, and I'm going through column in here.

Problem with the column is it's all [? low in ?] the cache. This is your linear memory. So because this is here two dimensional, but inside your hardware, there's no two-dimensional memory. You have a linear memory, so this is your linear memory. This is how the memory look like. And this is how the memory look like.

So each row and column is in these areas. But here, to calculate this value, this is nice. I go through memory that is contiguous. But C, I am jumping all over my memory. And jumping on memory is not that great cache behavior.

And contiguous can do much better, because what you do is, when you get a cache, you get a cache line that has more than one word. So in here, what you're doing is you are getting large cache lines only using one word before you go to the next one. So that's not great in cache behavior. So what you can do is you can make this memory contiguous.

So what you can do is, in matrix multiply, normally, n to the cubed computation to the squared data. So to help on n to the cubed, it's OK to do some n to the squared processing. That means you can do a data transport before you start matrix multiply. Do some pre-processing. So this will also come in handy.

A lot of these problems you'll do, you'll say, wait a minute, can I do a pre-process that's cheaper that will really improve my main processing? This is a theme that comes again and again. And then we can get this n to the cubed running faster since this n to the cubed is much larger than n to the squared. You can amortize the cost of doing that.

And basically, to get that, we can transpose some matrix going into the squared operations. And then now n to the cubed might run much faster. And in fact, what I have done here is do the transpose of this matrix. And now we run this one, and then everything will be in contiguous memory. And voila, I got another 3.4x performance improvement.

Now I am running almost at one cycle per op, which is really nice. But modern superscalars actually can do better, so we can actually end up being even better. So let's see. So we are doing that.

So here is the interesting thing. Now I look at my transpose [? spot. ?] So if you look at that number of retired instructions, it's more or less the same. Number of SSE instructions, more or less the same. I have a 2x improvement in my L1 cache miss rate. But more than that, I have a 5x improvement in the CPI.

That means my instructions are stalling a lot less now than before. So this is where I got all my performance [UNINTELLIGIBLE], because my instructions are running much faster. So by looking at data-- this is the kind of things you're learning during the class-- then you can get a feel for what actually is happening and why you are getting that.

You can go do more in the cache, in memory system. So remember, the memory system, if you're [UNINTELLIGIBLE], if you look at cache, what they're doing is we have a small amount of memory called cache that are very fast access. And you have a large amount of memory that is sitting out that has slow access. So what you want to give user feel as if they have a lot of memory, everything is very fast access. So hardware does a lot of crazy things to give you the illusion that you have very fast memory and you have a huge amount of that.

Of course, that's not true, because you have these caches. And what happens is, if you do things wrong, these get to slow down. So in the cache system, you have cache [UNINTELLIGIBLE] You have each multicore going to its own L1 cache that goes L2 cache, which goes to L3 cache, which goes to the main memory system.

And you want all of the data to be here. If that's the case, things run very fast. If not, you have to go here. It will be a lot more slow, here, even slower, here, very slow. So the key thing is to get the data as much as possible here.

So here's the kind of cycles, basically. One cycle to get here-- three cycles. If you go to L2 to L3, you have to do 14 cycles [UNINTELLIGIBLE]. If you go to L3 to main memory, you might sometimes get hundreds of cycles delaying here, so it's very costly.

So the caches are very temperamental beasts. They work beautifully, but when they don't work, these go really haywire. So a lot of times, if there's a performance issue, look at the cache miss rate and say, aha, the caches are not behaving. And can I help in here?

So the interesting way to look at that is how much data I touch. These graphics didn't show up. Let me explain. So if you do matrix multiply, I am trying to calculate one value in here. So this is one value. I have to calculate row and the column.

This slide got screwed up a little bit. So just look at this calculating [? row bar. ?] So if I want to calculate a row of values here, what I have to do is I have to get this row in B, and I have to use entire C to calculate one row of A. That means I have touched this much data items just to get one row of A. So that means I am calculating 1,024 values of A. I am touching this much data items to get that.

On the other hand, if I do block matrix multiply, that means I calculate, basically, a block in here. This block is also 32 by 32. It has 1,024 elements in here. But I only need a block here and block here. Voila, I only have looked at 25,000 elements. I got the same number of output calculated. But I have only touched much less elements altogether to calculate that. Do you see that?

And nice thing is, if this thing fits in the cache, I am much better than this might not fit in the cache. So I can calculate blocks in matrix multiply by looking at a lot less number of data, that has a good chance at fitting in cache-- and of course, when you calculate. you make sure it fits in cache-- than trying to just calculate row by

row.

So one thing you can do is-- this is a lot of common things we do. There's many ways to get same results. You can change the execution order. You can change the algorithm. You contain data structures. And some of them actually might do better than the other.

And here, we can actually say we can do the execution of the change and make sure we are only looking at small amount of data, you get the same number of data calculated. And of course, sometimes you have to be careful, because if you do things in different order, you might not get the exact same results.

So if you are a numerical [UNINTELLIGIBLE], you say, OK, of course, if you're doing A plus B plus C, what order you do will make a difference [UNINTELLIGIBLE]. So if you're really careful, you can't do too many things.

But most times, you can do some of these things. And in fact, in our final thing, we might say-- the final project-- we will give you something that you can change the algorithm as long as the outcome of the [? image ?] is the same. If you look at the [? image, ?] and the [? image ?] outcome is the same, you can go change things.

So a lot of times, you might have a lot of freedom to change the amount of errors you get, the precision, and, in fact, doing that, that you can actually get a lot of good performance. So in here, we are not trying to change the matrix multiply, of course. But we are going to change the way order of operations is being done. So some people say you can't do that. It would make matrix multiply different. But in most cases, it's OK.

So what we have done is done what we call block matrix multiply. So instead of calculating rows, you calculate blocks at a time. So calculating each block, you only need a few amount of data. So this is block matrix multiply.

And voila, by doing block matrix multiply, I got even 1.7x performance gain from that. Now, it's very interesting. Now with each block cycle, I am doing two operations. So my [? superscale ?] is finally coming to being. So every clock cycle, I

am doing multiple operations here. So this is very interesting, the number here.

So now when you look at what happens here, I am actually executing a little bit more instructions here. Why? Because I did block matrix multiply. That means I am doing a little bit more overhead in here. I have more loops in here. My inner loops are small. So I'm doing more instructions to do execute there. So I'm doing more instructions, but I have a huge cache performance gain. My cache miss rate now going down to very, very little-- 0.02%.

And I have an 8x improvement in cache, and that means I got a 3x improvement in my CPI in here. That's very nice. That means even though I'm doing a little bit more work, my cache behavior really, really helped me in here. So kind of says why I am getting this performance gain.

So what else can we do? We can go look at trying to do a lot more crazy optimizations because if you look at the modern Intel, they have this thing called [? retro ?] instructions. We'll get a little bit more detail into that later. And what we can do sometimes is you can nudge the compiler to take advantage of these instructions.

Of course, if nothing helps, you can actually go write to assembly [UNINTELLIGIBLE]. But hey, I don't want you guys to write assembly code, at least in this class, even though you can get really good performance sometimes. But you can kind of give enough hints to compiler and pray and kind of [UNINTELLIGIBLE] there's no nice way on these things, and hope for the compiler to do the right thing.

So in here, we can play with, a lot of times, compiler flags. And this is not exact size. We keep trying those things, and sometimes we can even get information back and say, OK, what did the compiler do? It might give you some good hints and say, I couldn't do something because of this part of the code. And worst comes to worst, you can actually generate assembly and stare at it. And sometimes in this class, you might actually want to stare at assembly and see what goes on in here.

So here is the assembly for here. And this is my loop body in here. And then what I

can see, I'm not going to go through that. Everything is converted into SSE instructions. At least I know what SSE instructions mean, so this seems to be doing well, so I'm happy. And in fact, I'm really happy because I got a 2.8x improvement by doing this.

Now what I am actually doing each clock cycle about [? five of my ?] operations. So I'm going really fast in here. So now, incidentally, I'm running my program about 30,000x faster than when I started. So if you haven't noticed, these things slowly add up in here.

And now, if we look at what's going on here, it's very interesting. What I have done is actually, my each instruction is now CPI has gone down. That means each instruction is running slower because I'm doing these SSE instructions. And in fact, my cache miss rate has also gone up. But I am doing a lot less instructions, because SSE means in one instruction, I am doing a vector.

So I'm actually getting four operations done in one instruction. So the number of instructions I execute has drastically gone down. So even though my instructions are running a little bit slower, I can basically amortize that, and I get good performance in here.

And of course, this one says, OK, now 88% of the instructions are SSEs. So I have [UNINTELLIGIBLE] most of the things to run on vector mode in here. And of course, you can keep doing more and more and more. And there are these four things [UNINTELLIGIBLE] matrix multiply, there are people who spend their entire lifetime trying to run it faster. And they have libraries like that, and this thing called a BLAS library and this thing called Intel Math Kernel. So they have done a lot more other things, like pre-fetching, getting everything right.

So here what I do is basically call their library, don't do anything myself, and this library will do it for me. And in fact, that library itself gave me another 2.7x performance gain in there. Now I'm actually doing 11 instructions every clock cycle by doing that.

And if I look at it carefully, what they have done is they are also very into SSE heavy, so they are running almost the same amount of instructions. But they managed to actually get, again, a better miss rate. They probably figured out some stuff in here, things like pre-fetching instructions. And that means they actually brought down the CPI back to where it was before. And that's why they gave that performance [INAUDIBLE].

And finally, we are doing a little bit of parallel execution. So as you know, multicores are here. And you're actually going to these 12-core machines, which is really nice. Fastest possible machines you can get your hands on these days. And it's kind of fun to work with them. And we can basically get concurrency for parallel performance. And a large part of this class at the end is trying to get good parallel performance.

But there are a lot of issues with parallelism. I will go a couple of issues now. Of course, when we get to that part, we will go in detail. One thing is called Amdahl's Law. Because if you're trying to run something parallel, you can't run everything parallel. When you start something, there's part of the code that basically starts the computation. And then there are the parts you can run parallel, and then other parts you have to also wind down, measure, things like that.

So what Amdahl's Law says is, even if you have an infinite number of processors, the maximum speedup you can get is this-- this the part of the code that are sequential and [? parts of ?] the code are parallel. That means if you have only 90% of the code can be parallelized, the maximum speedup you can get is 10, even if you have an infinite number of processors. So 99% of the code is parallel, the maximum speedup you can get is 100. So this is very interesting, because even if you have huge [UNINTELLIGIBLE], if your code has a certain amount of sequential part, there is a limit of what you can do.

[UNINTELLIGIBLE] also think of load balancing. That means when you get parallel, what you are assuming is all processes are busy all the time. That's very hard to achieve. Sometimes some processes are busy. Sometimes others are not.

Sometimes something's finished early. So if that happens, even though you have parallels, you might not get really good performance in here.

And then, of course, there's a thing called granularity of parallelism. Normally, what happens in programming, you run sequential part, you run parallel part, you run sequential parallel part. To start parallelism is a big task. You have to tell all the other processors, OK, look, you guys have some work. Go do that, and finish that, and come back. And that itself is very expensive.

And the work you give is very small. You give the work. The start-up and tear-down cost of parallelism itself might be too much than the speedup you get. In fact, you can make programs parallel and slow them down really drastically.

So you have to be very careful in that, because you can say, oh, I'm running in parallel, but it's running slow. Why? Because you might have a granularity issue, because we are giving so little work that even though you're getting parallelism, you might not get that much performance.

So if you look at matrix multiply, what happens here is you can divide the matrix into two different parts in here and calculate as two different matrix multiplies in a different core or a different process. Or in here, I can make it two or whatever in matrix multiplies to do that. So this is nicely parallelized. No big issue in here. And when I parallelize, I got rather 3.5x performance improvement.

Of course, I could have gotten more here, but because I had to start here, since I didn't want to wait for a week for this to finish, I had to come up with a smaller matrix size. But now, when I get here, this is so fast, my granularity is too small. So that's why I didn't get-- because this was an eight-core machine, I didn't get 8x because the minute I get parallelism, it just is over because it's running so fast in here.

Now, at this point, using eight cores, I am actually getting 36. So this is not that interesting, because this is not running on one core, every clock cycle, I am trying to get 36 operations done. So the interesting thing to follow in here is that this example might be somewhat contrived, because I started with this immutable-type matrix

multiply. Hopefully, you and your colleagues who haven't taken this class probably won't go that far.

But even if you start somewhere here, still, there's a huge gain to be made. So to put this in perspective, OK, so if you look at here, even if I start here somewhere out in C, I already got 131x performance improvement. That's huge. So that means in this class, we are not talking about small things.

And in fact, this is why I had to go log scale in doing absolute grades, because we had people who figured out exactly how, in some problems, to change data structures, change execution ordering, do some really cool thing, get 1,000x performance improvement. And there are other people who did little bit things and got 2x improvement. So the fact that what you can do the best here can be really, really high. So that's why we actually started giving you the second chance to also go basically see, if you missed a critical point, to go back and learn that.

So to put this in perspective-- so in summary, what we have done is-- matrix multiply might be an exception. Every time you go, you might not get that much, but in here, going from this very software-engineered, nice [UNINTELLIGIBLE] to a BLAS [UNINTELLIGIBLE], I got a 296,000 times performance improvement. This is huge.

If you put this in perspective, if you look at miles per gallon between these two-- a supertanker and this scooter-- it's only 14,000x, basically. So what that means is you have to have 20 supertankers to basically match this much improvement. So if somebody can tell you, look, I can't get a supertanker to run at the same miles per gallon as your scooter, that would be revolutionary.

But you can do something similar in your computer. And in fact, if you look at these large data centers, the computers are actually burning a lot of energy, a lot of cost. And in fact, if you can reduce that, that can have impact. So what you get out of this class is to think through these programs this way. Learn about correctness before [UNINTELLIGIBLE] 6.170, you hopefully are good at writing correct programs.

But that's not good enough in a lot of times. And how to think about programs, how

to make it run faster, there are multiple [UNINTELLIGIBLE] necessary. So for example, in here, it's basically direct energy. So can you get the data center to burn 10% less energy, that's huge. For example, if you look at today's supercomputing centers, they spend about $10 million a year on energy, just for the power grid. So if you can say, I'm [? playing with ?] 10%, that's huge.

Other thing, one thing Charles says a lot of times is, what performance gives you is currency, because performance itself might not be that directly useful, but it gives you currency to buy something else. What it is? So assume you have a nice application. Your GUI is doing OK. It's up to the point that people don't feel stalled.

But if you want to do something addition, if you add a feature, you don't want to slow down the program too much, because your users are going to complain. So what performance gives you is, if you improve the performance, it gives you currency, the ability to buy some additional features into your system. Or you have a system that you created a start up, and you created this nice server somewhere.

Suddenly, it became very popular, and a million people want to use your software. And the software might not be ready to do that. It might just crash and burn, or you might have to pay a lot of money for Amazon for the [? cycles, ?] so by actually making programs faster, you can make it scalable, you can make it efficient. So this gives you the ability, this currency, to go do a lot more other interesting things to your software because now you can run it faster. It gives you this margin to play with.

So in this class, we are going to learn a lot about these things. We are going to learn about architecture, how looking at the architectural issues, how to identify when something is wrong. We are going to look at algorithmic issues in here, how to look at things. We are looking at architectural things like memory systems, parallelism. And also we will talk about all the tools, so you will actually know something has gone wrong. Because today, you run the program, it runs, it runs correctly, is it good enough or not? How do you identify that? That's, I think, the very interesting place to start.

So that's all I have today. Make sure you get a Project 0. And especially people who haven't done C, we will-- Charles, are we going to do a C remediation? We did last time, didn't we? OK, guys, one more thing. Last year, we did an evening C remediation class.

So if you know Java, if you want to know C, we will send out a time with TAs that will give you kind of a one-hour crash course into C in there. And we will send information. So this class moves very fast. Get on with Project 0, and see you next week.