MIT CSAIL FastCode Seminar: Rezaul Chowdhury, October 19, 2020

1  00:00:05.100 --> 00:00:10.769
Julian Shun: Hi everyone. Good afternoon, and welcome to the fast code seminar. So, today our speaker is Rezaul Choudhury.

2  00:00:11.370 --> 00:00:21.270
Julian Shun: Rezaul is an associate professor of computer science at Stony Brook University and he's a core faculty member of the Institute for Advanced computational science.

3  00:00:21.840 --> 00:00:28.620
Julian Shun: Rezaul’s research interests are in the fields of algorithm design and engineering as well as their intersections with other sciences.

4  00:00:29.040 --> 00:00:37.050
Julian Shun: And he's particularly interested in scheduling and theoretical and practical performance analysis of parallel cache efficient algorithms.

5  00:00:37.560 --> 00:00:51.660
Julian Shun: And his research interests also include computational biology and bioinformatics, in particular protein, protein docking and fast energetics computations. Rezaul is a recipient of the NSF Career Award in

6  00:00:53.490 --> 00:01:05.640
Julian Shun: And today he's kind enough to tell us about his work on automatic derivation of efficient parallel recursive dividing conquer algorithms for dynamic programs. So I'll turn it over to you Rezaul.

7  00:01:06.240 --> 00:01:09.330
Rezaul Chowdhury: Thanks for the introduction duty and thanks for inviting me.

8  00:01:10.350 --> 00:01:22.650
Rezaul Chowdhury: Good afternoon, everyone. So today I'll be talking about one way of driving dividing conquer algorithms for dynamic programming problems and also why such algorithms are desirable.

9  00:01:23.400 --> 00:01:36.870
Rezaul Chowdhury: So this is a joint work with two of my former students promote and just me and the super tech and computer aided programming research groups at MIT and our collaborator at Fordham University and

10  00:01:38.310 --> 00:01:41.280
Rezaul Chowdhury: So this, this was actually
Rezaul Chowdhury: Part of the thesis research of a postdoc who is now a research assistant professor at Stony Brook University.

Rezaul Chowdhury: Let's start with answering this question, why do we want to generate recursive divide and conquer algorithms. So the answer basically lies in the memory hierarchy of modern machines. So we know that the textbook model.

Rezaul Chowdhury: Of the computer is as shown at the top you have a processor which is directly connected to the RAM. But in practice, real machines really have a sequence of caches between the RAM and the processor.

Rezaul Chowdhury: As you keep going to it. The processor, the size the caches become faster and faster, but at the same time, they become smaller.

Rezaul Chowdhury: And on multiple machines. These caches can be even more complicated. So in order to perform well on such machine algorithms need to have high

Rezaul Chowdhury: High locality in their data, data access pattern. So they want to they must keep frequent nexus get items and data items that will be accessed in near future as close to the CPU as possible. Otherwise, they may not perform well because of penalty of caching pieces.

Rezaul Chowdhury: So, one easy way of seeing this phenomena is by looking at this, I mean trying out this is a different expression. Multiply codes.

Rezaul Chowdhury: So this is really just the European history for new origin entrepreneurship for loop that multiplies to square matrices, X and Y and put the result in an emergency.

Rezaul Chowdhury: And all these six this six different versions are obtained simply by polluting the loop de loops. So, there is also the main. Correct. And just that is genius. Is that how you access the matrices we assume that both x y AMP z are our enrollment order, say suppose. So we have six versions.
Rezaul Chowdhury: Now if you

So if you now around them on a say on a modern machine. So we ran them on air on an interview on machine and used on one call it had

Private 32 K BIA one cache and to 656 Gabriel to and share 2020 megabytes all three and 32 GB RAM and we turned off all optimizations.

And it turned out that to have the variance to have the variance ran much faster than two other and the remaining two were in between. So you can see that in the middle plot where we show the running times and turns out that the

IKEA J and K. I. T. Those two versions, the loop with those are loop orders that means the loop for k

In the outer most part. And then we have say I then we have Jay like that guy, Jason. So the I can share and Kitty codes, they are much faster than, say, Jackie i n ke VI.

So, and it turns out that this these faster running times really follow their l one, I to cache misses you can see them on the left hand side and on the right hand side.

And all these boils down to how the code access the matrices.

The more sequential access means more special locality and better performance and non sequential accesses result in cache misses that means

When the processor is looking for data items senior in, say, one cache it and it doesn't find it there, it will have to go to the alteration beyond so that causes and then one caching means there is a penalty for that and that slows it down.

And what happens is this is only a special locality, whatever block you bringing
Rezaul Chowdhury: Into the cache. You want to make sure that it has as much as too late as possible.

Rezaul Chowdhury: But there is another type of locality call Kemper and locality, that is related to deter us. That means whatever you bring meaning to the cashier

Rezaul Chowdhury: You want to make sure that you work on it as much as possible before throwing it out what you can do in order to improve temporary locality. What you can do is block the input matrices, X and Y, and also the output metrics see

Rezaul Chowdhury: I mean, brought them into smaller sub matrices. So that freezer, the smallest of matrices can feed into the cache to cache size is m and up to them and then what you do, you

Rezaul Chowdhury: Multiply the blogger metrics block by block. So since three matrices freedom to those cache is unification is fully associative and all those

Rezaul Chowdhury: Then you only in the cache misses for reading domain and writing them out and for the computation. You do not incur cache misses that gives you data us without cache misses and that that improves performance.

Rezaul Chowdhury: But we can also achieve data reuse.

Rezaul Chowdhury: Data Use using recursive divide and conquer say

Rezaul Chowdhury: If we're aggressively divide matrices, X y&z into smaller and smaller sub matrices. And because in the process to submit races.

Rezaul Chowdhury: What will happen is, at some point, at some level of recursion, the matrices will feature to the cashier and again is, even though the caches are fully associative and all those good things, then you, you will not incur cache misses. Okay.
Rezaul Chowdhury: Except for editing them in and writing the data out. That's it. So it gives you also then data us, but we need some constant factor of the tiling method that tiling approach that uses optimal tile sizes, but the tiny approach needs to know the size of the cashier, but

46 00:08:08.160 --> 00:08:13.200
Rezaul Chowdhury: In contrast, these digital divide and conquer algorithm doesn't need to know the size of the question.

47 00:08:14.790 --> 00:08:19.950
Rezaul Chowdhury: Okay, so, so the speakers to divide and conquer has to important

48 00:08:21.270 --> 00:08:22.020
Rezaul Chowdhury: To 14

49 00:08:23.550 --> 00:08:33.840
Rezaul Chowdhury: advantages over time. So he has cache efficiency, energy efficiency bandwidth efficiency, all those compared to ties looping

50 00:08:34.980 --> 00:08:40.860
Rezaul Chowdhury: But the most important our cache obliviousness that means that you

51 00:08:43.830 --> 00:08:54.540
Rezaul Chowdhury: Know, yeah. That means that the code is portable in the sense that he doesn't need to know the size of the caches, whatever the size of the cache it will fit into the cache at some level of recursion.

52 00:08:54.930 --> 00:09:03.030
Rezaul Chowdhury: And it will indeed feed into every level of every Kashi level at some level of aggression. So simultaneously fits. I mean,

53 00:09:04.680 --> 00:09:12.870
Rezaul Chowdhury: I mean, at some level of experimental feet in every level of every casual level, which doesn't happen in case of dialing you explicitly need to tell them

54 00:09:13.230 --> 00:09:23.760
Rezaul Chowdhury: To make sure that they fit into specific levels. So that makes it portable they want me to change the code when you move from one machine to another or even for different levels of of the cache it

55 00:09:24.390 --> 00:09:35.040
Rezaul Chowdhury: And the room for another important property is cache adaptability. So that means that these algorithms, because of divide and conquer can be

56 00:09:35.490 --> 00:09:47.130
Rezaul Chowdhury: Basically self adapt basically sells adapt to available caches space. That means when the caches space changes on the fly because other processes are running and running and all those

57 00:09:47.670 --> 00:09:59.850
Rezaul Chowdhury: They can basically self adapt, so they are cache adaptive but a time code is not cache adaptive they suffer when cache size changes without their knowledge.

58 00:10:00.780 --> 00:10:11.250
Rezaul Chowdhury: So these are the two main reasons why we want you guys to do that and conquer and later you will see that also recursive divide and conquer algorithm for dynamic programs will give you better parallelism.

59 00:10:12.090 --> 00:10:22.140
Rezaul Chowdhury: So these are the two main reasons why we want you guys to do that and conquer and later you will see that also recursive divide and conquer algorithm for dynamic programs will give you better parallelism.

60 00:10:23.400 --> 00:10:23.850
Rezaul Chowdhury: Ah,

61 00:10:25.110 --> 00:10:40.950
Rezaul Chowdhury: So the main reason is this dynamic programming is a widely used widely used algorithm design technique, so it appears in many application areas, including computational biology, economics, finance, even a sports operation research and everywhere.

62 00:10:42.150 --> 00:10:42.630
Rezaul Chowdhury: And

63 00:10:45.600 --> 00:10:59.970
Rezaul Chowdhury: So, so that means if we can make these algorithms efficient this if we can solve this dynamic programming problems efficiently, then that will benefit many areas application areas.

64 00:11:00.660 --> 00:11:13.560
Rezaul Chowdhury: Okay, so what is a dynamic programming approach for solving problems. So BP solves the problem. What it does is that it triggers in the sub divides the original problem into smaller sub problems.

65 00:11:14.100 --> 00:11:24.240
Rezaul Chowdhury: And it solves a real problem exactly once. And it distorts the solutions of the sub problem in a table so that it doesn’t need to be computed when it

66 00:11:24.960 --> 00:11:28.200
Rezaul Chowdhury: Encounters the same problem again in the future.
And a dp is usually is specified using a recurrence relation, like here, the reconciliation, you'll see here he says that the DB table X digest entry of the DPT X is computed from entry ik KJ.

Of the same DPT boom and I kick a j k goes from it. J, like that. So these three guys also they'll see in which order the supplements needs to be solved, because some of the sub problems dependency on solutions of other sub problems.

So if such a recurrence is given to you as you can see it is a straightforward to write code iterative code to solve this recurrence.

Here for example for this one, you need to write three nested loops i, j and k and just update the x i j based on X idea. It's like an extension and that's it. So we call this iterative dynamic iterative loop. He is looking codes as IDP tentative DP. Let's call them IDP.

But you can often solve these requests relations every district translation relations using recursive divide and conquer.

You call first a using the original up table and then it calls itself twice and then it calls function be and function be calls function be and function see eight times and function see is calling itself. So that's the algorithm. So that's the algorithm. So

LM said that, though.
Rezaul Chowdhury: And it's not.

L is and this is A, this kind of algorithms are a little bit more complicated than straightforward grouping code as you can see and implementation of these codes correct and

efficient implementation can is not necessarily straightforward. There are many, many things you need to worry about.

First is the structure is much more complicated. Second reason this is recursion, not a straightforward whooping code.

And also you need to make sure the implementation is correct. And there are also, though this is recursive at some point of recursion. You need to move you need to move to generative codes so that you do not.

So that you can control the overhead of recursion. And also how to penalize the code and all those things you need to worry about not very straightforward as looking good.

And what happens is this is so.

You can see we've got in conquer is not a straightforward looping code is very straightforward. And the thing is that.

These dynamic programming problems are often created and evaluated by non CS researchers, so they do not necessarily have any formal CS background. So one.

So these are already difficult for CS.

Researchers to implement this. Why do we expect not. I mean someone without
Rezaul Chowdhury: CS background to do. I mean to spend time trying to implement it efficiently.

Rezaul Chowdhury: So then the question is okay. Can we order it auto generate these RDP implemented RSVP RSVP means because we divide and conquer DP the recursive divide and conquer implementation. Can we

Rezaul Chowdhury: Are to generate them. Why don't we simply auto generate them say you are given a DVD currents and then the Beatles, and then you design a system that will take that returns and automatically generate efficient because we do and and content creation.

Rezaul Chowdhury: And also, I mean, maybe there's possible because we already did something like that for, for, for, for instance in competition. This is bush one, which again we did with the super tech

Rezaul Chowdhury: Privileges license. So, and

Rezaul Chowdhury: In case of Bush, we're, what happens is he were given a stencil, which is also like a recurrence. It simply says that

Rezaul Chowdhury: You are given a given some input values at timestamps zero and then you are given this stencil which says how you compute values a timestamp one using values from timestamp zero and then four times to want to timestamp two times two times two times two, three, then

Rezaul Chowdhury: You keep lying that for many, many times steps and this has many applications in many areas so

Rezaul Chowdhury: So partial could take such a stencil and could generate a very efficient cache efficient implementation of that of that stands in computation.

Rezaul Chowdhury: But the difference is that in case of DP. We don't even know what else for what algorithm will have will have begin at the implementation for
Rezaul Chowdhury: For sure though the stencil can change the base, the base algorithm, the trapeze order and the composition algorithm remains the same.

Rezaul Chowdhury: You simply need to generate an implementation on this episode of the composition algorithm. I mean, that uses the stencil given to you.

Rezaul Chowdhury: But for DP problems. We may not even know the recursive divide and conquer. We don't, for which we need that we need to implement. Okay, so what happens is there are, first of all, there are numerous many, many existing DB occurrences and deeply occurrences are

Rezaul Chowdhury: Discovered I mean all the time, and also the main problem is is slight change in the DVD governance matching the recursive algorithm completely. So there's the main problem there is no single element. And for many recurrences. We do not even dirty had no Delgado

Rezaul Chowdhury: And it means

Rezaul Chowdhury: Something

Rezaul Chowdhury: Okay, so here. See, I mean the

Rezaul Chowdhury: Show to the consumer and conquer algorithms for two different

Rezaul Chowdhury: Two different DP problems. First one is the advantages problems. Second one is flawed washers. A PSP.

Rezaul Chowdhury: Understand that these are not entirely visible but that's not the point here. The point here is to show that

Rezaul Chowdhury: The two algorithms are really completely different and independence problem we have three recursive calls in the prod marshals all shortest paths we have four different recursive calls, and they do make the recursive calls differently.
Rezaul Chowdhury: The inputs and outputs are treated differently. So, but the recurrences themselves are not that different. They have some similarities. Still, the algorithms are completely different. So

Rezaul Chowdhury: For efficient implementation. So you need to deal with such complicated algorithms. Okay. So that's the main problem. And so that means that if I give you a deaconess KP recurrence and I want you to

Rezaul Chowdhury: Output high performing implementation of that recurrence for us. We'll have to figure out what is the recursive divide and conquer algorithm that evaluates that returns. So that's the part that I will be talking about today. That's the part that auto Gentiles.

Rezaul Chowdhury: So if it

Rezaul Chowdhury: Did is a recursive divide and conquer algorithm and generic pseudo code for that.

Rezaul Chowdhury: So indeed, in fact, so there that there are two two systems that we worked on oxygen, and Bill mania.

Rezaul Chowdhury: Oxygen. This is an inductive approach for generating these reconsidered in contact with them and then one is Bill mania, which is also with with the computer aided programming group and also super tech

Rezaul Chowdhury: We worked on that. So, and Bill mania is a directive genetic code using deductive reasoning. So they are different. So at least Doc, I will only

Rezaul Chowdhury: Talk about oxygen. So what happens is, in case of opportunity to give it any black box implementation of a difference.

Rezaul Chowdhury: And generous and efficient auto enroll speakers to divide and conquer pseudo code that portable means cache oblivious robust means cache adaptive efficient means highly parallel cache efficient and all this stuff.
Rezaul Chowdhury: Okay, so

Rezaul Chowdhury: Although Jian works for problems DP problems that belong to a class, which we call fractal DP. Okay, so it doesn't work for everything.

Rezaul Chowdhury: It was for one special specific class. But fortunately, this is a large plus

Rezaul Chowdhury: And this class has several properties, two of the major ones is that one will sweep good party. That means if so it depends on, and understand why of the DB table, then why is fully updated before X rays from why

Rezaul Chowdhury: That means once X restaurant. Why you can no longer update why anymore. So that means it cannot use why and this why is finally so there’s one with super party.

Rezaul Chowdhury: And second one is factor party. So that means basically that means from high level is that for any given the problem.

Rezaul Chowdhury: The dependency patterns you observe in sa to the poor and the way to the bottom DB table is independent of M except possibly for very small, very often, that means that the dependency patterns in a DP table doesn't depend on the size of the typical so that's another requirement.

Rezaul Chowdhury: So we also assume that the number of recursive functions is upper bounded by a constant for efficiency reasons and also

Rezaul Chowdhury: Also, the way update updated repeatable cell, though, you can you can you can apply any update function. But when you when you update the DP table you use the associative and commutative operator of operation to update it is. I mean, Max. Plus, multiply on those

Rezaul Chowdhury: And so the way you update the DP table is that you apply the operation to update the cell and then use the associative and commutative property to update the rest of the table.

Rezaul Chowdhury: So the way you update the DP table is that you apply the operation to update the cell and then use the associative and commutative property to update the rest of the table.
Rezaul Chowdhury: So as I said before, fractal DP includes meaning well on dynamic programming problems like Benton's problem for us. A PSP sequence alignment on a second restrictive production. I mean, like many of them.

00:22:36.120 --> 00:22:50.190
Rezaul Chowdhury: So what is auto just core idea, the core idea basic idea is this that it it exam means the DP table accesses when you around if you run a DP.

00:22:51.240 --> 00:23:05.280
Rezaul Chowdhury: DP what DP table cells are these called excesses and from that it tries to find a recursive pattern among those x

00:23:06.540 --> 00:23:10.860
Rezaul Chowdhury: So that's the idea. And then it builds an algorithm around that you get better.

00:23:12.540 --> 00:23:16.200
Rezaul Chowdhury: So let's see. So this is a

00:23:17.460 --> 00:23:18.270
Rezaul Chowdhury: So it's a

00:23:19.560 --> 00:23:30.360
Rezaul Chowdhury: Really high level overview of how the algorithm really works. So it starts like this. See this is at the top you can see Luke parentheses, that is

00:23:30.960 --> 00:23:47.460
Rezaul Chowdhury: A complete nested loop evaluating the parenthesis recurrence. . As you can see in line for x equals mean of exposure and exciting plus x KJ wi kitchen. So that means it's it depends on it psyche and escape.

00:23:48.540 --> 00:24:16.710
Rezaul Chowdhury: Okay, so instead of doing this update or in addition to doing this update. What we do is that we modify the code, .

00:23:57.360 --> 00:24:07.050
Rezaul Chowdhury: Or yes, when you find the code to output. This DP table coordinates. That means when it is updating excited using x i can escape. We update

00:24:07.440 --> 00:24:16.710
Rezaul Chowdhury: Hi, Jay IK and KJ coordinates. Those are the coordinates that we update. So initially what we do for a very small DB tables outside 6464
Rezaul Chowdhury: We learn these generic DB table updates code to generate all these updates that applying this is not really efficient. There is another more efficient way of doing it, but

Rezaul Chowdhury: ELYSEE, we do it in this way we collect all updates the DB table makes so you can see on the hand side all the updates.

Rezaul Chowdhury: So the great part is the table is the sale that we are updating and blue birds are the sale that we are reading from, so I did this and we're writing to I can't gauge the sales, we are reading from. So these are the, this is a set of updates for a small DB table of size 64 by 64

Rezaul Chowdhury: Then what we do is this, we think, though, that set now. So the original DB table supposes x that means

Rezaul Chowdhury: These in these updates every, every sin coordinate belongs to X say at the top, you can see a cell, an update 131113 and 131113 all those belong to x, which is a 64 by 64 tables.

Rezaul Chowdhury: Now what we do, we now categorize each of these updates based on which, in which one trend here. This is a two dimensional guess that same thing what

Rezaul Chowdhury: Which one trend each of those entries belong to, say, for example, at the top, we can see that 1311 and one three all those belong to quadrant.

Rezaul Chowdhury: X one, one. Or maybe if I consider 130 311 and 133 then 133 belongs to quadrant X 1211 belongs to one one x one, one and one type. It belongs to quadrant.

Rezaul Chowdhury: So we can look to all the updates and we categorize them based on which quadrants what teams, they belong to.

Rezaul Chowdhury: here. As you can see, we end up with four different categories. There are really
Rezaul Chowdhury: Four times four times four, four cute. That means 64 different possible categories, but we ended up with four categories that are not empty.

150 00:26:49.230 --> 00:27:00.000
Rezaul Chowdhury: , so the leftmost branch. You can see it's one one, it's one one and x one, one. That means all updates that went to that category have their entries in x one, one.

151 00:27:00.660 --> 00:27:13.290
Rezaul Chowdhury: And the next one. The second branch has x two x one x one, two, that means that the first cell belongs to x one to the second one is one, one, the third one. The next one to like that we have for them.

152 00:27:14.940 --> 00:27:16.590
Rezaul Chowdhury: But as you can see

153 00:27:18.270 --> 00:27:29.070
Rezaul Chowdhury: The first and the last branch, the first and last branch they align the Oriental algorithm. As you can see that originally the original problem all

154 00:27:29.820 --> 00:27:38.250
Rezaul Chowdhury: Sell Koreans belong to sell a matrix. Next in the first and last branch look in the first branch all sales really belong to one one.

155 00:27:38.940 --> 00:27:51.300
Rezaul Chowdhury: So you were reading from x one one and updating. It's one one and the last one. Also, we don't exclude that means you are reading from X to to underwriting, too. It's too too so they look like this is a little bit more complicated than that. But they look like

156 00:27:52.740 --> 00:27:58.020
Rezaul Chowdhury: They look like dots in a function, a and but

157
00:27:59.370 --> 00:28:08.910
Rezaul Chowdhury: The second entire branch you different we're seeing the second branch, you were reading from x one one and you are writing to x one to you. Also, of course.

158 00:28:09.300 --> 00:28:20.130
Rezaul Chowdhury: I reading from one to over there. So you are reading from, next one, next one to anyone. I didn't want to in the third one. And so in the second one, the third one. Similarly, . So,

159 00:28:21.390 --> 00:28:30.750
Rezaul Chowdhury: Then what happens is that if we find that two categories are writing to and reading from the same quadrant. We marched them.
we will resolve these conflicts later. And what happens is also we now mean the function scenes. The first branch and the last branch look like a

We named them functioning, So they are released more instances of origin problem and the next on the middle one, which we didn’t encounter before. Just give it a new name so function be

So in fact, these depend on input and output.

Fingerprints so input fingerprints means that

The inputs look structure is similar, if if between two functions inputs local structure is similar. We are. We say that they they have seen in proofing. They have same input fingerprint. The same fingerprints and also the output fingerprints means that our fingerprints are

computed based on how the input matrices are distributed recursive calls later so if both of the math. Then he said that those two functions that the thing. So now we have any function will be

A we have already encountered so we do not need to do anything about it. We have any information being and now we go ahead and do exactly the same thing that we did with functionality functionality.

So here we do take function, be it has these upset of updates that we had. And then we again.

categorize them, divide see

X you and be here. You say, and based on which quadrants of each of these matrices. They belong to, we categorize them. And this time we'll end up with a non empty category is among the 64
Rezaul Chowdhury: Among the state. It turns out that four of them are like the original function \( P \), they are reading from to these giant matrices even be in function beaded

Rezaul Chowdhury: What we are doing function we we are reading from to disjointed. He says he when we and we are writing to \( x \) as well as reading from \( x \)

Rezaul Chowdhury: The same thing happens in the bottom for function calls they are structure is similar. So we've called them function be again because they are really function.

Rezaul Chowdhury: And the top four, they are different in the top four. As you can see they are reading from to disjointed submit races, the blooper and they are writing to

Rezaul Chowdhury: Another these joints of metrics and they are not reading from that distance of metrics. So he had

Rezaul Chowdhury: This one, so this is this is the report the writing to and the blue bars are the ones they are reading from the joint.

Rezaul Chowdhury: So this is we call a different function. Let's name it. See, and then let's go ahead and expand. See the exactly the same way we expanded in be

Rezaul Chowdhury: And when we expand see turns out that will end up with it smaller instances of see

Rezaul Chowdhury: And there are no other functions. And we stop there. We have since we have not encountered any new functions at this level. That means even if we keep

Rezaul Chowdhury: Expanding the scenes here, we will not encounter anything new. And we are done.

Rezaul Chowdhury: so that means we end up with this is we call an algorithm tree we under the tree like structure that shows
Rezaul Chowdhury: We had initially started with functioning and function calls itself twice in a it calls function v and function we makes for because of course to function equal to itself and for recursive calls to another function called see another function, that function see makes educative calls

Rezaul Chowdhury: So that's what we end up with, but the here as you can see what these are little tree doesn't tell you is in which order these functions must be executed.

Rezaul Chowdhury: See there are a dry dependencies among the function some functions are writing two locations that some other functions will be reading from and also the, this one doesn't tell you

Rezaul Chowdhury: Which function calls can be executed in parallel, we resolve all those using these retry dependencies there are four rooms. I'm not going to the rules, but there are four rules that we use that

Rezaul Chowdhury: And that reason based on based on this rich try dependencies to tell you

Rezaul Chowdhury: If given two functions which function should be executed before if one function should basically be for the other are both directions of the same or they can be executed in in parallel. Okay, so there are false that we use to resolve these issues.

Rezaul Chowdhury: And then what happens is based, once we resolve those, we figured out in which order they must be executed will end up with DEX like this. The first. First one is a deck for function,

Rezaul Chowdhury: So that says that in functioning as you can see that it is calling at the top that is functioning. We are calling on metrics X during part is the part that will be updated, and

Rezaul Chowdhury: Then it calls itself twice on smaller triangles and you can exhibit both of them in parallel, that what is is showing you is equal to is in parallel, followed by a call to function be
Rezaul Chowdhury: And then in the next one. You see the function being executed and it first calls itself, then it costs two instances, a function. See, followed by two instances, a function being parallel.

Rezaul Chowdhury: Then followed by again function. See, followed by another function. See, and finally function be. And finally, we have expression for function see in function, see what it does is that it calls itself.

Rezaul Chowdhury: Four times in parallel for instances of exam in parallel, followed by under 40 instances in parallel, so

Rezaul Chowdhury: This is different. So these are the complete decks of, of the, of the recursive divide and conquer with them.

Rezaul Chowdhury: Here. One interesting to think to observe is this that we started with function, a

Rezaul Chowdhury: In function A what happened is you were reading from which exists and also you are writing to metrics.

Rezaul Chowdhury: So if it seems you are a reading from and writing to the same metrics. There are EJ dependencies that will prevent you from reordering the loops. If you are using looping code.

Rezaul Chowdhury: That will prevent you from reordering the loops without making the correct that will also prevent you efficiently parallelize the coders vector eyes the code and mine.

Rezaul Chowdhury: But as you can see function is better than that in function be what happens is that you are reading from to

Rezaul Chowdhury: Other places the bloopers is even see that are completely disjointed from the place you are writing. So this is more flexible than function A you can make some
Rezaul Chowdhury: Optimizations on the some validation some vectorization some more compiler optimizations on this, but his team, you are writing to a reading from some I mean from

Rezaul Chowdhury: From one part of it. So,

Rezaul Chowdhury: Though is still it is so that means mysterious not completely flexible. Then finally, what you have is function see

Rezaul Chowdhury: In function. See what happens is you are updating a matrix X using other two matrices you and fee that are completely discharged from x and

Rezaul Chowdhury: This means that this is almost like matrix multiplication, where you are. You multiply two matrices human V and put the result in a complete it is john metrics x. So you are reading from and writing to these giant matrices.

Rezaul Chowdhury: That opens up a lot of opportunities for optimization for parallelization for vectorization. And since we have experienced in optimizing

Rezaul Chowdhury: Optimizing matrix multiplication code. A lot of people worked on that. So that means you can optimize this function seem very easy not very easily. I mean,

Rezaul Chowdhury: You can optimize it very well. That's. And what also turns out that and that's that in this province apprentices problem for which we are generating this code.

Rezaul Chowdhury: These calls to function see really seem totally dominates the work done by function. See, it seemed totally dominates work done by functions. A and B. So that means once you optimize function seen anywhere. Almost done.

Rezaul Chowdhury: , so, so these are then we have these decks from the Dax. We have these
Rezaul Chowdhury: These recursive divide and conquer algorithm. A, B, and C, when the matrix is small. You do looting code. Otherwise you recur simply call

212 00:37:47.850 --> 00:37:58.080
Rezaul Chowdhury: Other functions and there's idea. And what happens is Allison, the routing code for matrix A is for function A is not very flexible, so it cannot be optimized very well.

213 00:37:58.620 --> 00:38:11.100
Rezaul Chowdhury: But the looping code for function see is very flexible. It can be optimized almost as an optimization can be almost as good as matrix multiplication unfortunately function see really dominates the computation. Okay.

214 00:38:11.400 --> 00:38:20.160
Rezaul Chowdhury: So that's done. So that means because we divide and conquer units poses you opportunities for validation that realization and other compiler optimizations.

215 00:38:21.180 --> 00:38:21.630
Rezaul Chowdhury: So,

216 00:38:23.430 --> 00:38:29.370
Rezaul Chowdhury: So that, that's how we generate the code, . So more details are in the paper so

217 00:38:31.110 --> 00:38:37.860
Rezaul Chowdhury: So it does have this that these Ardipithecus into an account algorithms are efficient in theory. So he released

218 00:38:38.670 --> 00:38:45.690
Rezaul Chowdhury: A Lister many problems fantasies problem. The gap problem Broadway shows. A PSP gap is a sequence alignment with that find gap costs.

219 00:38:46.140 --> 00:38:53.190
Rezaul Chowdhury: protein folding function approximation and all of those and here in the first two columns.

220 00:38:54.120 --> 00:39:08.880
Rezaul Chowdhury: Released the theater shop lexicon parallelism of eternity version of the dynamic programming code that means IDP. So, and as you if you look at the car Kesha complexity, none of them really have has none of them really has

221 00:39:11.850 --> 00:39:22.830
Rezaul Chowdhury: An m, m is the size of the cashier in the denominator, , the complexity is either n cubed intuitive, or B or N squared over be like that. That means they have at most.
Rezaul Chowdhury: Special locality no temporal locality. If you have temporal locality, you will have your cache complexity will reduce, which will be a reduction will be a function of him.

Rezaul Chowdhury: And in case because we divide and conquer. It turns out that the serial Kesha complexity.

Rezaul Chowdhury: Really is much better than the car cache implicit debate a diversion and you can see $M^2$ of $M$ appearing the denominator.

Rezaul Chowdhury: Next, so that means, the larger the value of $M$ the cache size, the smaller the number of cache misses and also the parallelism.

Rezaul Chowdhury: . And many times the parallelism, you get better parallelism than the

Rezaul Chowdhury: Than the iterative code. For example, in case of fantasies problem you're validating is entity but $t$ minus

Rezaul Chowdhury: Law three and that is a symbiotic any larger which is entity, but I think 1.41 or something which is increasingly larger than the colorism you get from native code in neither so they're efficient in theory, and it turns out that they are also efficient in practice. So

Rezaul Chowdhury: So this is

Rezaul Chowdhury: This is a word that was that was done four or five years back. So we have experimented results that were done on machines available at that time. So we, we ran our code we implemented the code and ran our code.

Rezaul Chowdhury: On a 16 core into a Sandy Bridge processor two times eight, eight cores per socket and gratitude to be one cache private 256 K be pivotal to 20 megabyte shared

Rezaul Chowdhury: L three and 32 megabytes of RAM we implemented during into c++ compiler 13.0 intensive class conferencing platform.
Rezaul Chowdhury: And the competition parameter search here given. So with vectorization and we generated times the derivative DP codes using a state of the art Oriental compiler called Pluto.

Rezaul Chowdhury: And so our and

Rezaul Chowdhury: So these are the plots 14 different DP problems running templates and cadences

Rezaul Chowdhury: We did fantasies and we did gap that is that is seekers alignment with their find gap costs and fraud was a PSP. So the first rule.

Rezaul Chowdhury: The runtime the runtime performance. Here we are plotting for three different algorithms three different implementations reconsider and conquer our DP, which is green eternity code just without dialing which ad for me to tiling that is blue IDP and we have titled IDP that is red.

Rezaul Chowdhury: And the plots in the top row are indeed updates per second. That means that the larger the better for parentheses problem. As you can see, so we didn't all of course all those all those

Rezaul Chowdhury: Codes on 16 processors all 16% sorts. So, and along the x axis we are increasing the dimension to the PT one dimension.

Rezaul Chowdhury: And on along the way it is, we are plotting the gigabits per second in log scale.

Rezaul Chowdhury: As you can see, for parentheses problem are the RDP request into an and conquer and tie the IDP they performed almost the same. They are basically the same at that one but IDP was much, much slower. And it turns out that it is perhaps 18 times slower or something like that for gap problem.
Rezaul Chowdhury: Again recursive divide and conquer was much faster and much more efficient compared to both dial the IDP an IDP and for fraud was an exam again the same thing, because we do a new control as much

243 00:43:05.520 --> 00:43:06.480
Rezaul Chowdhury: Much more efficient.

244 00:43:07.740 --> 00:43:22.170
Rezaul Chowdhury: It turned out that in fact the result generated called the tile dial code tile the IDP. What happened is that for practices problem. It was able to dial in all three dimensions. And it was as efficient as

245 00:43:23.310 --> 00:43:34.830
Rezaul Chowdhury: As our typical and it was optimized for the best type size for that machine. So, but for gap and fraud washers. A PSP. It was not able to optimize. So you're

246 00:43:35.250 --> 00:43:44.400
Rezaul Chowdhury: Dialing all three directions you don't retire and on two dimensions. And that's why they're cache performance for gap and productions. A PSP.

247 00:43:45.240 --> 00:44:02.070
Rezaul Chowdhury: Were not didn't have really Kesha performance was bad for fantasies problem.

248 00:44:02.820 --> 00:44:16.980
Rezaul Chowdhury: Time the IDP had both temporal and spatial locality for all of our shows. A PSP have a special locality for Gabby didn't have anything so

249 00:44:18.270 --> 00:44:27.660
Rezaul Chowdhury: And for gap and for the actual CPU speed dial the DPS Kesha complexity matched the cache complexity of an IDP without diving. So, and that's what the second row is showing that

250 00:44:28.140 --> 00:44:42.450
Rezaul Chowdhury: Tiled IDP and RDP in got almost the same amount of cache me says, which is much smaller than just IDP. But in case of gap and produce shells. A PSP rica RDP was much better than both IDP and tied it up.

251 00:44:44.130 --> 00:45:01.860
Rezaul Chowdhury: And then next are good scalability and lower energy consumption. So in the first row we show for fantasies gap and ferocious. A PSP like we finished in 228192 so to reporting tends to be about 13 or two to the power

00:45:18.090
Rezaul Chowdhury: It was 13 so and we didn't both all those three implementations as we increase the number of processors still 16 and it turns out that our dp is in fact

00:45:29.220
Rezaul Chowdhury: Status very well. It was always almost always 18 times faster for parentheses 17 times for gap and around six times faster. It's 16 processors. When we reach

00:45:35.370
Rezaul Chowdhury: Far, far, far versus the PSP and the bottom, bottom floor a little bit more

00:45:46.680
Rezaul Chowdhury: Complicated looking but they are showing the energy consumption, with respect to our DP for eternity dp and Tyler IDP

00:45:53.070
Rezaul Chowdhury: These plots are showing the ratio of energy consumption.

00:45:57.150
Rezaul Chowdhury: DRM and package energy consumption two different clubs for

00:46:03.810
Rezaul Chowdhury: For IDP and tidy up with respect to, ah, our DPS energy consumption.

00:46:06.630
Rezaul Chowdhury: And it turns out that

00:46:09.810
Rezaul Chowdhury: For granted this problem.

00:46:20.910
Rezaul Chowdhury: Like before time the IDP and RDP with the same but IDP was much more efficient and in case of gap and productions. A PSP.

00:46:23.850
Rezaul Chowdhury: Both

00:46:30.060
Rezaul Chowdhury: IDP entitled ADP were more efficient, compared to
Rezaul Chowdhury: Compared to article

Rezaul Chowdhury: So the next block is more interesting because in case of fantasies, as we have seen in fact bind IDP tiling code and recursive divide and conquer are basically the same, then why should we use

Rezaul Chowdhury: Because you'd have to divide and conquer. So this one gives you some motivation for for using still using some recursive divide and conquer this plot is trying to show you that.

Rezaul Chowdhury: That you just do it and conquer albums adapt.

Rezaul Chowdhury: To

Rezaul Chowdhury: Any available caches space when multiple say contractors and running. I mean, concurrent processes or other programs running and you have a shared Keisha, and he assured in the shared Keisha space becomes

Rezaul Chowdhury: increases or decreases as other programs and start running or terminate and then your program will have to adapt to that available caches space, .

Rezaul Chowdhury: Ensures that in fact the recursive divide and conquer algorithm can adapt will

Rezaul Chowdhury: Better much better than dial it up an IDP time iterative and just simply to do the first block it shows on time. , it should sign. So what we need here is this that we fixed one socket.

Rezaul Chowdhury: With a processing costs on the machine and we are really restricted every process that will run. Write to us at most two of the course to course.

Rezaul Chowdhury: And what happens is for independently independently separately for our DP IDP untied IDP. We ran. First one instance of that code.
Rezaul Chowdhury: on the machine and then the using two goals. And then we went to instances using to goes into both for both then we ran three instances of the same code using

Rezaul Chowdhury: Six goals than 40 instances of the same code using so they are using each course so they are not fighting for the course. What they are fighting for is the caching spacing LT LT shared cashier

Rezaul Chowdhury: So in the first part, what we are showing is that travel time we say 124 processes, compared to run time. We only want those as we did

Rezaul Chowdhury: When we read only one process we recorded the runtime and then we ran. We ran two, three or four processes we take we took the maximum running time they were running simultaneously.

Rezaul Chowdhury: The one that to the maximum time we recorded that time. And here we are showing the ratio

Rezaul Chowdhury: Anti as it turns out, as you can see as the number of processes increases along the horizontal axis you see number of processes and on the vertical axis, you see the ratio.

Rezaul Chowdhury: As a number of processes take place, the time IDP code becomes slower and slower ready to start taking

Rezaul Chowdhury: More and more time. So this is just time so that the lower the better. So it starts to take more and more time and interesting. We also just regenerative code is also taking more time. And on the right hand side.

Rezaul Chowdhury: We show number filter cache misses. Again, the ratio with on a single processor. It turns out that for for IDP just iterative code, the number of cache and he says, Didn't change as the cache space.

Rezaul Chowdhury: Changed when you start in the process of starting to fight for Kesha space.
Rezaul Chowdhury: Because you're dividing. He doesn't have any temper locality. It can work with the constant number of cache of blocks. That's it. It is not storing anything and reusing anything that's why cache misses doesn't didn't change. But in case of

Rezaul Chowdhury: Time the IDP. The number of caching mixes increase to affect or 4.5 when we have purposes, but as you can see for recursive divide and conquer the cache misses increased to 1.5 when we have forecloses as compared to, and we had on one classes. , so

Rezaul Chowdhury: In case of them are timed IDP. The running time increase because of this bear its inability to adapt to changing cache size when it is fighting with other processes. But in case of just IDP it slowed down, slow down, but it's not because of

Rezaul Chowdhury: Because it, it fought for caches based it slowed down because of

Rezaul Chowdhury: Pressure on memory bandwidth, because it was already incurring a lot of cache basis as more important also started to run simultaneously. The number of casualties has increased that increased pressure and memory, man. Wait, and that costs, some slow down.

Rezaul Chowdhury: So these shows why we we may prefer

Rezaul Chowdhury: We may desire to have

Rezaul Chowdhury: Just one contract when compared to the IDP. So the next next blog is also showing some other experiment for Kesha sharing. I will not go into that.

Rezaul Chowdhury: So skip over that. So, this is the second last slide so what

Rezaul Chowdhury: We have seen is that or to the empty lives efficient as it were in contact with us for the faculty, because it is it does it for one specific plus of dynamic programs and
Rezaul Chowdhury: And it is fully automatic, and the United algorithms are marketed cache oblivious and cache adaptive and as you have seen expected results show that they are really efficient, but there is one.

Rezaul Chowdhury: Maybe multiple problems, but one who one thing is that auto Jen really works on regular DPS with data oblivious.

Rezaul Chowdhury: Regular data of the BS done any programming problems. It doesn't work on a eating there and the data dependent dynamic programming problems like the say return, return. So the knapsack problem or the CEO of the data centers. So our

Rezaul Chowdhury: It has a

Rezaul Chowdhury: So, and also in some cases the cache or performance or the parallelism.

Rezaul Chowdhury: Of the general audience can be improved, sometimes using more space or something like that. And finally, there are some extension efforts. Some, some extensions were made it to the

Rezaul Chowdhury: To the automation system. Again, as I said, we only have research prototype that generic pseudo code. So these are based on that so

Rezaul Chowdhury: In 2017

Rezaul Chowdhury: We showed that

Rezaul Chowdhury: Some that are generated by Otter Jen can have better optimization. If we can reduce artificial dependencies. So the recursive divide and conquer algorithm.

Rezaul Chowdhury: Forces some artificial dependencies that that do not exist in our human recurrence, because you are course learning and the blocks so that we do.
Rezaul Chowdhury: And also there is biometric multi recursive divide and conquer algorithms. So that means that in can see in the recursive Dorian contributions that agenda that are generated by auto can those are.

Rezaul Chowdhury: Now we are dividing into halves all the blocks are divided into hubs, but

Rezaul Chowdhury: Sometimes it is a it is more useful. If you can, if you can change how many blocks, you can generate and finally

Rezaul Chowdhury: So there is also an MPa is distributed memory implementation.

Rezaul Chowdhury: For oxygen, so that is based on oxygen. So, this

Rezaul Chowdhury: So this concludes my talk.

Rezaul Chowdhury: Thanks for listening.

Julian Shun: Thanks a lot. Rezaul, for the very interesting talk. So does anyone have any questions?

Julian Shun: So actually I see a question in the chat from Richard Barnes. How does one obtain the source for auto Gen I assume this is source code, it is not available.

Rezaul Chowdhury: So we have any such prototype I can share this with you. I mean, he did. He made me so I can share the source code so that elicited research prototyping generic pseudo code not actually implementation part bill mania. The other system.

Rezaul Chowdhury: That we did with the

Rezaul Chowdhury: Computer Aided programming group also so protective of that also generates
Rezaul Chowdhury: Interesting presentations. So next time!

Julian Shun: Okay.

Great.

Julian Shun: I was also curious if auto Gen can generate code for other architecture like GPUs, or is that currently just for multi-course

Rezaul Chowdhury: So in fact, I mean, we didn't exactly extend oxygen for doing that. But we did implement all these

Rezaul Chowdhury: recursive version contract with them for also for GPUs. We didn't implement them separately, but

Rezaul Chowdhury: Yeah, yes, it is truly difficult to generate for course for GPUs.

Julian Shun: Okay.

Thanks.

Yeah.

Julian Shun: Another question I had was you presented the theoretical balance on parallelism then cache complexity for these algorithms. I'm wondering if those are the best balance that are out there for those algorithms are there better ones that use different

Rezaul Chowdhury: Not necessarily, not necessarily the best ones. Okay, let me go back.

Rezaul Chowdhury: So
Rezaul Chowdhury: So what happens is a

Rezaul Chowdhury: Parallelism in case of parallelism. So we later short that

Rezaul Chowdhury: So in that spot 2017 paper that

Rezaul Chowdhury: If we remove artificial dependencies, then you can include the parallelism say

Rezaul Chowdhury: We can improve valleys and for say fantasy problem for say

Rezaul Chowdhury: Also not for Broadway show maybe for maybe protein folding, things like that. And also, cache, your performance wise, a gap problem for the gap problem.

Rezaul Chowdhury: So it turns out that the cache or performance can be improved from MTV or b squared of em that released over there, too. And you do a BM

Rezaul Chowdhury: This was done by a young student who had a guide book, so they had a paper on that. And of course, that requires you to use additional space now our region is completely in place.

Rezaul Chowdhury: So, So these are not necessarily all of the. Some of them are optimal. Of course, you can prove them. They are optimal nor can improve say Kesha complexity was insane, So, but some of them can still be improved.

Julian Shun: Okay, great.

Julian Shun: Thanks a lot.

Julian Shun: So there’s a question from Lucca. Are there any PROGRAMS THAT GENERATES SUCH code for those other classes of DP problems.
Rezaul Chowdhury: I'm not aware of.

Rezaul Chowdhury: Anything like that. I mean, at least not recursive. Divide and conquer algorithms. They are there are systems that generate optimized looping code and so

Rezaul Chowdhury: For maybe many. I mean, say,

Rezaul Chowdhury: DP problems. Other than that are outside this class, but for eternity divide and conquer. I'm not aware of any other system that does it. So far, I haven't seen

Julian Shun: Okay great, thanks.

Luka Govedič: Are there is like a

Luka Govedič: Like a quick follow up.

Luka Govedič: Did you like I have no idea, like how complex, would it be like, do you, did you consider trying to extend it to those problems like is it even possible, would it require like a completely different approach.

Rezaul Chowdhury: So say for example, I mean, they may be quite complicated might be different approaches. so that a good question. That's a good question for, for example, the data. Data oblivious things that we are using for origin that helps us a lot to come up with

Rezaul Chowdhury: Come up with the requested and contract with them easily if the dependencies, or do I mean if the updates depend on data like inventor be our

Rezaul Chowdhury: Next topic, then it's not so the kind of approach on adversely how it would work. So you may need to come up with completely different approaches for that.

Julian Shun: Okay, great.
Julian Shun: Are there any additional questions unresolved?

Julian Shun: Okay, so it looks like there are no more questions now.

Julian Shun: Well thanks again Rezaul for this interesting topic.

Julian Shun: And if you want to see the source code, you can feel free to contact Rezaul.

Rezaul Chowdhury: Yes, I will share the code.

Julian Shun: Great. Well, thanks again. And thanks, everyone, for coming.