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

11 00:01:42.480 --> 00:01:51.570
Rezaul Chowdhury: Part of the thesis research of a postdoc who is now a research assistant professor at Stony Brook University.

16 00:02:02.670 --> 00:02:16.500
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.

17 00:02:17.610 --> 00:02:29.880
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

18 00:02:30.690 --> 00:02:39.330
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.

19 00:02:40.410 --> 00:02:53.160
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

20 00:02:54.300 --> 00:03:11.550
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.

21 00:03:12.840 --> 00:03:24.030
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.

22 00:03:24.450 --> 00:03:34.680
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.

23 00:03:35.430 --> 00:03:59.700
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.

24 00:04:00.750 --> 00:04:02.940
Rezaul Chowdhury: Now if you

25 00:04:04.290 --> 00:04:15.120
Rezaul Chowdhury: 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

26 00:04:15.630 --> 00:04:27.870
Rezaul Chowdhury: 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.

27 00:04:28.350 --> 00:04:46.800
Rezaul Chowdhury: 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

28 00:04:48.510 --> 00:04:56.430
Rezaul Chowdhury: IKEA J and K. I. T. Those two versions, the loop with those are loop orders that means the loop for k

29 00:04:57.480 --> 00:05:12.330
Rezaul Chowdhury: 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.

30 00:05:13.410 --> 00:05:24.150
Rezaul Chowdhury: So, and it turns out that this these faster running times really follow their l one, l to cache misses you can see them on the left hand side and on the right hand side.

31 00:05:24.870 --> 00:05:31.800
Rezaul Chowdhury: And all these boils down to how the code access the matrices.

32 00:05:32.340 --> 00:05:41.910
Rezaul Chowdhury: The more sequential access means more special locality and better performance and non sequential accesses result in cache misses that means

33 00:05:42.150 --> 00:05:54.180
Rezaul Chowdhury: 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.

34 00:05:55.770 --> 00:06:02.160
Rezaul Chowdhury: So, and what happens is this is only a special locality, whatever block you bringing

35 00:06:02.760 --> 00:06:07.380
Rezaul Chowdhury: Into the cache. You want to make sure that it has as much as too late as possible.

36 00:06:07.770 --> 00:06:15.300
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

37 00:06:15.660 --> 00:06:30.150
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

38 00:06:31.380 --> 00:06:42.060
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

39 00:06:43.170 --> 00:06:51.960
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

40 00:06:52.470 --> 00:07:06.600
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.

41 00:07:08.010 --> 00:07:12.720
Rezaul Chowdhury: But we can also achieve data reuse.

42 00:07:13.980 --> 00:07:17.640
Rezaul Chowdhury: Data Use using recursive divide and conquer say

43 00:07:19.500 --> 00:07:28.860
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.

44 00:07:29.400 --> 00:07:45.090
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.

45 00:07:46.470 --> 00:08:06.840

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: OK, so now the next question is, okay, why are we targeting dynamic programming problems. Why do we want to make them efficient, so

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.

67 00:11:29.160 --> 00:11:45.450
Rezaul Chowdhury: And a dp is usually is 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

68 00:11:46.230 --> 00:12:02.790
Rezaul Chowdhury: 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.

69 00:12:05.100 --> 00:12:15.480
Rezaul Chowdhury: 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

70 00:12:16.080 --> 00:12:37.530
Rezaul Chowdhury: 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

71 00:12:40.740 --> 00:12:50.040
Rezaul Chowdhury: But you can often solve these requests relations every district translation relations using recursive divide and conquer.

72 00:12:50.730 --> 00:13:00.510
Rezaul Chowdhury: . But these recursive divide and conquer algorithms are often are quite complicated like here. This is solving basically the same recurrence

73 00:13:00.990 --> 00:13:08.370
Rezaul Chowdhury: Though I have simplified it a lot, but it is this. This is a high level view of the algorithm that solves the same recurrence

74 00:13:08.820 --> 00:13:14.700
Rezaul Chowdhury: Using recursive divide and conquer. As you can see it has three different functions. A, B and C.

75 00:13:15.210 --> 00:13:33.630
Rezaul Chowdhury: 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

76 00:13:34.920 --> 00:13:38.370
Rezaul Chowdhury: LM said that, though.

77 00:13:39.930 --> 00:13:40.530
Rezaul Chowdhury: And it's not.

78 00:13:44.280 --> 00:13:54.960
Rezaul Chowdhury: 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

79 00:13:55.620 --> 00:14:02.130
Rezaul Chowdhury: efficient implementation can is not necessarily straightforward. There are many, many things you need to worry about.

80 00:14:02.520 --> 00:14:07.950
Rezaul Chowdhury: First is the structure is much more complicated. Second reason this is recursion, not a straightforward whooping code.

81 00:14:08.280 --> 00:14:21.570
Rezaul Chowdhury: 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

82 00:14:23.190 --> 00:14:31.890
Rezaul Chowdhury: 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.

83 00:14:33.270 --> 00:14:37.230
Rezaul Chowdhury: And what happens is this is so

84 00:14:38.640 --> 00:14:44.220
Rezaul Chowdhury: You can see we've got in conquer is not a straightforward looping code is very straightforward. And the thing is that

85 00:14:45.360 --> 00:15:00.150
Rezaul Chowdhury: 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

86 00:15:01.590 --> 00:15:04.950
Rezaul Chowdhury: So these are already difficult for CS

87 00:15:06.330 --> 00:15:13.050
Rezaul Chowdhury: Researchers to implement this. Why do we expect not. I mean someone without

88 00:15:14.580 --> 00:15:19.500
Rezaul Chowdhury: CS background to do. I mean to spend time trying to implement it efficiently.

89 00:15:21.120 --> 00:15:31.890
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

90 00:15:32.580 --> 00:15:46.260
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.

91 00:15:47.490 --> 00:15:59.700
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

92 00:16:00.990 --> 00:16:03.660
Rezaul Chowdhury: Privileges license. So, and

93 00:16:04.710 --> 00:16:10.710
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

94 00:16:11.160 --> 00:16:25.950
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, three, then

95 00:16:27.240 --> 00:16:34.590
Rezaul Chowdhury: You keep lying that for many, many times steps and this has many applications in many areas so

96 00:16:35.940 --> 00:16:44.250
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.

97 00:16:45.030 --> 00:16:55.230
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

98 00:16:55.980 --> 00:17:03.120
Rezaul Chowdhury: For sure though the stencil can change the base, the base algorithm, the trapeze order and the composition algorithm remains the same.

99 00:17:03.750 --> 00:17:13.080
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.

100 00:17:13.530 --> 00:17:28.350
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

101 00:17:29.760 --> 00:17:47.520
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

102 00:17:51.120 --> 00:17:51.870
Rezaul Chowdhury: And it means

103 00:17:55.170 --> 00:17:55.590
Rezaul Chowdhury: Something

104 00:17:58.770 --> 00:18:01.800
Rezaul Chowdhury: Okay, so here. See, I mean the

105 00:18:03.360 --> 00:18:06.990
Rezaul Chowdhury: Show to the consumer and conquer algorithms for two different

106 00:18:08.130 --> 00:18:15.060
Rezaul Chowdhury: Two different DP problems. First one is the advantages problems. Second one is flawed washers. A PSP.

107 00:18:15.480 --> 00:18:21.720
Rezaul Chowdhury: Understand that these are not entirely visible but that's not the point here. The point here is to show that

108 00:18:22.080 --> 00:18:33.300
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.

109 00:18:33.900 --> 00:18:46.050
Rezaul Chowdhury: The inputs and outputs are are treated differently. So, but the recurrences themselves are not that different. . They have some similarities. Still, the algorithms are completely different. , so

110 00:18:47.160 --> 00:19:02.100
Rezaul Chowdhury: For efficient implementation. So you need to deal with such complicated 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

111 00:19:03.600 --> 00:19:18.930
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.

112 00:19:20.790 --> 00:19:22.530
Rezaul Chowdhury: So if it

113 00:19:24.000 --> 00:19:28.110
Rezaul Chowdhury: Did is a recursive divide and conquer algorithm and generic pseudo code for that.

114 00:19:30.120 --> 00:19:38.280
Rezaul Chowdhury: So indeed, in fact, so there that there are two two systems that we worked on oxygen, and Bill mania.

115 00:19:38.820 --> 00:19:55.080
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

116 00:19:56.130 --> 00:20:09.090
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

117 00:20:09.600 --> 00:20:16.380
Rezaul Chowdhury: Talk about oxygen. So what happens is, in case of opportunity to give it any black box implementation of a difference.

118 00:20:16.920 --> 00:20:28.890
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.

119 00:20:30.210 --> 00:20:32.040
Rezaul Chowdhury: Okay, so

120 00:20:34.260 --> 00:20:45.150
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 .

121 00:20:46.020 --> 00:20:51.150
Rezaul Chowdhury: It was for one special specific class. But fortunately, this is a large plus

122 00:20:51.750 --> 00:21:06.030
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

123 00:21:06.720 --> 00:21:19.470
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.

124 00:21:20.370 --> 00:21:29.280
Rezaul Chowdhury: And second one is factor party. So that means basically that means from high level is that for any given the problem.

125 00:21:29.760 --> 00:21:48.210
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.

126 00:21:52.110 --> 00:22:00.060
Rezaul Chowdhury: So we also assume that the number of recursive functions is upper bounded by a constant for efficiency reasons and also

127 00:22:01.050 --> 00:22:20.010
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

128 00:22:21.780 --> 00:22:34.980

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.

129 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.

130 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

131 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.

132 00:23:12.540 --> 00:23:16.200
Rezaul Chowdhury: So let's see. So this is a

133 00:23:17.460 --> 00:23:18.270
Rezaul Chowdhury: So it's a

134 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

135 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.

136 00:23:48.540 --> 00:23:56.190
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, .

137 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

138 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

139 00:24:17.340 --> 00:24:27.330

Rezaul Chowdhury: We learn these generic DB table updates code to generate all these updates that is applying this is not really efficient. There is another more efficient way of doing it, but

140 00:24:28.410 --> 00:24:37.410
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.

141 00:24:38.670 --> 00:25:03.090
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 6464 seven

142 00:25:04.800 --> 00:25:14.670
Rezaul Chowdhury: Then what we do is this, we think, though, that set now. So the original DB table supposes x that means

143 00:25:15.780 --> 00:25:34.980
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 6464 tables.

144 00:25:36.390 --> 00:25:48.660
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

145 00:25:49.170 --> 00:25:59.850
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.

146 00:26:00.780 --> 00:26:16.890
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.

147 00:26:17.340 --> 00:26:28.230
Rezaul Chowdhury: So we can look to all the updates and we categorize them based on which quadrants what teams, they belong to.

148 00:26:29.250 --> 00:26:37.230
Rezaul Chowdhury: here. As you can see, we end up with four different categories. There are really

149 00:26:38.340 --> 00:26:48.750

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.

160 00:28:31.140 --> 00:28:41.910
Rezaul Chowdhury: , we will 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

161 00:28:42.390 --> 00:28:53.040
Rezaul Chowdhury: 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

162 00:28:54.060 --> 00:28:57.750
Rezaul Chowdhury: . So in fact, these depend on input and output.

163 00:28:58.770 --> 00:29:01.140
Rezaul Chowdhury: Fingerprints so input fingerprints means that

164 00:29:02.910 --> 00:29:19.830
Rezaul Chowdhury: 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

165 00:29:21.210 --> 00:29:33.960
Rezaul Chowdhury: 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

166 00:29:34.770 --> 00:29:45.480
Rezaul Chowdhury: 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.

167 00:29:46.290 --> 00:29:53.700
Rezaul Chowdhury: So here we do take function, be it has these upset of updates that we had. And then we again.

168 00:29:54.720 --> 00:29:59.190
Rezaul Chowdhury: categorize them, divide see

169 00:30:00.810 --> 00:30:13.380
Rezaul Chowdhury: 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

170 00:30:13.920 --> 00:30:28.650
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

171 00:30:29.880 --> 00:30:38.040
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

172 00:30:38.550 --> 00:30:47.970
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.

173 00:30:48.480 --> 00:31:02.880
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

174 00:31:04.380 --> 00:31:11.580
Rezaul Chowdhury: Another these joints of metrics and they are not reading from that distance of metrics. . So he had

175 00:31:12.690 --> 00:31:19.110
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.

176 00:31:20.010 --> 00:31:28.650
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

177 00:31:29.160 --> 00:31:35.850
Rezaul Chowdhury: And when we expand see turns out that will end up with it smaller instances of see

178 00:31:36.840 --> 00:31:45.780
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

179 00:31:46.290 --> 00:31:50.820
Rezaul Chowdhury: Expanding the scenes here, we will not encounter anything new. And we are done.

180 00:31:51.780 --> 00:31:59.610
Rezaul Chowdhury: , so that means we end up with this is we call an algorithm tree we under the tree like structure that shows

181 00:32:00.090 --> 00:32:17.010
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

182 00:32:18.090 --> 00:32:18.840
Rezaul Chowdhury: , so

183 00:32:19.950 --> 00:32:31.260
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.

184 00:32:31.500 --> 00:32:43.050
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

185 00:32:45.060 --> 00:33:01.110
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

186 00:33:02.640 --> 00:33:08.700
Rezaul Chowdhury: And that reason based on based on this rich try dependencies to tell you

187 00:33:09.150 --> 00:33:25.680
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.

188 00:33:27.540 --> 00:33:39.900
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,

189 00:33:40.530 --> 00:33:52.650
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

190 00:33:54.120 --> 00:34:07.710
Rezaul Chowdhury: Then it calls itself twice on smaller triangles and you can exhibit both of them in parallel, that what is it is showing you is equal to is in parallel, followed by a call to function be

191 00:34:08.190 --> 00:34:19.650
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

192 00:34:20.160 --> 00:34:34.620
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.

193 00:34:35.820 --> 00:34:43.980
Rezaul Chowdhury: Four times in parallel for instances of exam in parallel, followed by under 40 instances in parallel. , so

194 00:34:45.270 --> 00:34:52.710
Rezaul Chowdhury: This is different. So these are the complete decks of, of the, of the recursive divide and conquer with them. .

195 00:34:53.100 --> 00:34:57.840
Rezaul Chowdhury: Here. One interesting to think to observe is this that we started with function, a

196 00:34:58.320 --> 00:35:04.680
Rezaul Chowdhury: In function A what happened is you were reading from which exists and also you are writing to metrics.

197 00:35:05.040 --> 00:35:16.920
Rezaul Chowdhury: So if he 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.

198 00:35:17.490 --> 00:35:28.560
Rezaul Chowdhury: That will prevent you from reordering the loops without making the 14 correct that will also prevent you efficiently parallelize the coders vector eyes the code and mine.

199 00:35:29.310 --> 00:35:38.070
Rezaul Chowdhury: But as you can see function is better than that in function be what happens is that you are reading from to

200 00:35:38.730 --> 00:35:49.740
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

201 00:35:50.670 --> 00:36:03.480
Rezaul Chowdhury: Optimizations on the some validation some vector ization some more compiler optimizations on this, but his team, you are writing to a reading from some I mean from

202 00:36:04.980 --> 00:36:07.500
Rezaul Chowdhury: From one part of it. So,

203 00:36:08.970 --> 00:36:16.170
Rezaul Chowdhury: Though is still it is so that means mysterious not completely flexible. Then finally, what you have is function see

204 00:36:16.710 --> 00:36:27.570
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

205 00:36:28.260 --> 00:36:41.370
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.

206 00:36:41.820 --> 00:36:52.500
Rezaul Chowdhury: That opens up a lot of opportunities for optimization for parallelization for vector ization . And since we have experienced in optimizing

207 00:36:54.180 --> 00:37:03.750
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,

208 00:37:04.380 --> 00:37:16.020
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.

209 00:37:16.530 --> 00:37:31.020
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.

210 00:37:32.340 --> 00:37:39.780
Rezaul Chowdhury: , so, so these are then we have these decks from the Dax. We have these

211 00:37:40.650 --> 00:37:47.550

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.

222 00:39:25.020 --> 00:39:34.650
Rezaul Chowdhury: Special locality no temporal locality. If you have temper locality, you will have your cache complexity will reduce, which will be a reduction will be a function of him.

223 00:39:35.250 --> 00:39:40.920
Rezaul Chowdhury: . And in case because we divide and conquer. It turns out that the serial Kesha complexity.

224 00:39:41.370 --> 00:39:49.680
Rezaul Chowdhury: Really is much better than the car cache implicit debate a diversion and you can see M squared of M r M appearing the denominator.

225 00:39:50.100 --> 00:39:57.240
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.

226 00:39:58.020 --> 00:40:02.490
Rezaul Chowdhury: . And many times the parallelism, you get better parallelism than the

227 00:40:03.270 --> 00:40:08.340
Rezaul Chowdhury: Than the iterative code. For example, in case of fantasies problem you're validating is entity but t minus

228 00:40:08.760 --> 00:40:28.650
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

229 00:40:29.670 --> 00:40:30.450
Rezaul Chowdhury: So this is

230 00:40:31.590 --> 00:40:45.390
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.

231 00:40:45.990 --> 00:40:57.630
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

232 00:40:58.140 --> 00:41:06.750
Rezaul Chowdhury: L three and 32 megabytes of RAM we implemented during into c++ compiler 13.0 intensive class conferencing platform.

233 00:41:07.410 --> 00:41:19.260
Rezaul Chowdhury: And the competition parameter search here given. So with vector ization and we generated times the derivative DP codes using a state of the art Oriental compiler called Pluto.

234 00:41:20.340 --> 00:41:23.310
Rezaul Chowdhury: And so our and

235 00:41:24.900 --> 00:41:32.220
Rezaul Chowdhury: So these are the plots 14 different DP problems running templates and cadences

236 00:41:33.870 --> 00:41:44.070
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.

237 00:41:44.820 --> 00:42:03.480
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.

238 00:42:04.590 --> 00:42:16.920
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

239 00:42:18.300 --> 00:42:26.850
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.

240 00:42:27.630 --> 00:42:31.470
Rezaul Chowdhury: And on along the way it is, we are plotting the gigabits per second in log scale.

241 00:42:31.980 --> 00:42:50.100
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.

242 00:42:51.660 --> 00:43:03.720

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

24500:43:23.310 --> 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 --> 00:43:52.320
Rezaul Chowdhury: Were not didn't have really Kesha performance was bad for fantasies problem.

248 00:43:53.070 --> 00:44:02.070
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 00:44:02.820 --> 00:44:16.980
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 00:44:18.270 --> 00:44:27.660
Rezaul Chowdhury: In the second row we show total cache to cache to cache misses the lower the better in the first for the first block for panties problem. As you can see both

251 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.

252 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

253 00:45:04.140 --> 00:45:18.090
Rezaul Chowdhury: It was 13 so and 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

254 00:45:19.230 --> 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

255 00:45:30.300 --> 00:45:35.370
Rezaul Chowdhury: Far, far, far versus the PSP and the bottom, bottom floor a little bit more

256 00:45:36.810 --> 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

257 00:45:47.880 --> 00:45:53.070
Rezaul Chowdhury: These plots are showing the ratio of energy consumption.

258 00:45:54.180 --> 00:45:57.150
Rezaul Chowdhury: DRM and package energy consumption two different clubs for

259 00:45:58.230 --> 00:46:03.810
Rezaul Chowdhury: For IDP and tidy up with respect to, ah, our DPS energy consumption.

260 00:46:05.370 --> 00:46:06.630
Rezaul Chowdhury: And it turns out that

261 00:46:08.790 --> 00:46:09.810
Rezaul Chowdhury: For granted this problem.

262 00:46:11.220 --> 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.

263 00:46:23.370 --> 00:46:23.850
Rezaul Chowdhury: Both

264 00:46:26.460 --> 00:46:30.060
Rezaul Chowdhury: IDP entitled ADP were more efficient, compared to

265 00:46:31.440 --> 00:46:32.550
Rezaul Chowdhury: Compared to article

266 00:46:33.570 --> 00:46:47.640
Rezaul Chowdhury: So the next the next block is more interesting because he 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

267 00:46:48.030 --> 00:47:00.390
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.

268 00:47:01.740 --> 00:47:05.010
Rezaul Chowdhury: That you just do it and conquer albums adapt.

269 00:47:06.120 --> 00:47:06.570
Rezaul Chowdhury: To

270 00:47:07.710 --> 00:47:21.060
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

271 00:47:21.720 --> 00:47:32.940
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, .

272 00:47:33.570 --> 00:47:40.050
Rezaul Chowdhury: Ensures that in fact the recursive divide and conquer algorithm can adapt will

273 00:47:40.980 --> 00:47:55.560
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.

274 00:47:56.220 --> 00:48:08.280
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.

275 00:48:09.300 --> 00:48:20.550
Rezaul Chowdhury: And what happens is for independently independently separately for our DP IDP untied IDP. We ran. First one instance of that code.

276 00:48:21.150 --> 00:48:30.600
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

277 00:48:30.930 --> 00:48:42.060
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

278 00:48:42.420 --> 00:48:52.290
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

279 00:48:52.830 --> 00:49:01.590
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.

280 00:49:01.950 --> 00:49:05.820
Rezaul Chowdhury: The one that to the maximum time we recorded that time. And here we are showing the ratio

281 00:49:06.300 --> 00:49:15.510
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. .

282 00:49:15.900 --> 00:49:25.500
Rezaul Chowdhury: As a number of processes take place, the time IDP code becomes slower and slower ready to start taking

283 00:49:26.430 --> 00:49:43.770
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.

284 00:49:45.570 --> 00:49:58.710
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.

285 00:49:59.790 --> 00:50:03.570
Rezaul Chowdhury: Changed when you start in the process of starting to fight for Kesha space.

286 00:50:04.050 --> 00:50:18.330
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

287 00:50:20.190 --> 00:50:35.490
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

288 00:50:36.570 --> 00:50:57.270
Rezaul Chowdhury: In case of them are timed IDP. The running time increase because of this fear 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

289 00:50:58.530 --> 00:51:02.700
Rezaul Chowdhury: Because it, it fought for caches based it slowed down because of

290 00:51:03.930 --> 00:51:17.400
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.

291 00:51:19.050 --> 00:51:24.090
Rezaul Chowdhury: So these shows why we we may prefer

292 00:51:25.440 --> 00:51:26.970
Rezaul Chowdhury: We may desire to have

293 00:51:28.140 --> 00:51:36.660
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.

294 00:51:37.770 --> 00:51:39.660
Rezaul Chowdhury: So skip over that. So,

295 00:51:42.870 --> 00:51:53.820
Rezaul Chowdhury: Great, because we are almost out of time. So, so the final eaten. Are these that. So this is the second last slide so what

296 00:51:55.530 --> 00:52:05.610
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

297 00:52:06.630 --> 00:52:19.740
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

298 00:52:20.850 --> 00:52:29.700
Rezaul Chowdhury: Maybe multiple problems, but one who one thing is that auto Jen really works on regular DPS with data oblivious.

299 00:52:30.840 --> 00:52:43.020
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

300 00:52:44.490 --> 00:52:45.390
Rezaul Chowdhury: It has a

301 00:52:46.740 --> 00:52:51.030
Rezaul Chowdhury: So, and also in some cases the cache or performance or the parallelism.

302 00:52:52.290 --> 00:53:04.140
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

303 00:53:05.280 --> 00:53:12.510
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

304 00:53:13.800 --> 00:53:15.210
Rezaul Chowdhury: In 2017

305 00:53:17.670 --> 00:53:19.050
Rezaul Chowdhury: We showed that

306 00:53:20.100 --> 00:53:28.260
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.

307 00:53:28.620 --> 00:53:39.540
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.

308 00:53:40.620 --> 00:53:51.510
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

309 00:53:52.050 --> 00:53:57.330
Rezaul Chowdhury: Now we are dividing into halves all the blocks are divided into hubs, but

310 00:53:59.100 --> 00:54:08.910
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

311 00:54:10.500 --> 00:54:13.890
Rezaul Chowdhury: So there is also an MPa is distributed memory implementation.

312 00:54:14.910 --> 00:54:21.450
Rezaul Chowdhury: For oxygen, so that is based on oxygen. So, this

313 00:54:22.530 --> 00:54:24.270
Rezaul Chowdhury: So this concludes my talk.

314 00:54:25.740 --> 00:54:26.310
Rezaul Chowdhury: Thanks for listening.

315 00:54:28.440 --> 00:54:34.350
Julian Shun: Thanks a lot. Rezaul, for the very interesting talk. So does anyone have any questions?

316 00:54:35.700 --> 00:54:45.480
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.

317 00:54:45.660 --> 00:55:04.230
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.

318 00:55:05.400 --> 00:55:06.510
Rezaul Chowdhury: That we did with the

319 00:55:07.650 --> 00:55:12.840
Rezaul Chowdhury: Computer Aided programming group also so protective of that also generates

320 00:55:13.980 --> 00:55:17.910
Rezaul Chowdhury: Interesting presentations. So next time!

321 00:55:19.290 --> 00:55:19.620
Julian Shun: Okay.

322 00:55:19.740 --> 00:55:20.190
Great.

323 00:55:21.390 --> 00:55:31.980
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

324 00:55:32.670 --> 00:55:41.100
Rezaul Chowdhury: So in fact, I mean, we didn't exactly extend oxygen for doing that. But we did implement all these

325 00:55:42.480 --> 00:55:47.640
Rezaul Chowdhury: recursive version contract with them for also for GPUs. We didn't implement them separately, but

326 00:55:49.860 --> 00:55:53.670
Rezaul Chowdhury: Yeah, yes, it is truly difficult to generate for course for GPUs.

327 00:55:56.130 --> 00:55:56.580
Julian Shun: Okay.

328 00:55:56.670 --> 00:55:57.180
Thanks.

329 00:55:58.440 --> 00:55:58.740
Yeah.

330 00:56:00.360 --> 00:56:14.100
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

331 00:56:14.940 --> 00:56:18.750
Rezaul Chowdhury: Not necessarily, not necessarily the best ones. Okay, let me go back.

332 00:56:21.210 --> 00:56:22.350
Rezaul Chowdhury: So

333 00:56:23.880 --> 00:56:25.380
Rezaul Chowdhury: So what happens is a

334 00:56:26.790 --> 00:56:30.750
Rezaul Chowdhury: Parallelism in case of parallelism. So we later short that

335 00:56:32.160 --> 00:56:34.740
Rezaul Chowdhury: So in that spot 2017 paper that

336 00:56:36.090 --> 00:56:41.370
Rezaul Chowdhury: If we remove artificial dependencies, then you can include the parallelism say

337 00:56:42.390 --> 00:56:47.490
Rezaul Chowdhury: We can improve valleys and for say fantasy problem for say

338 00:56:50.070 --> 00:56:59.670
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.

339 00:57:00.240 --> 00:57:08.310
Rezaul Chowdhury: So it turns out that the cache or performance can be can be improved from MTV or b squared of em that released over there, too. And you do a BM

340 00:57:09.060 --> 00:57:23.280
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.

341 00:57:24.240 --> 00:57:37.020
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.

342 00:57:38.640 --> 00:57:39.450
Julian Shun: Okay, great.

343 00:57:39.480 --> 00:57:40.140
Julian Shun: Thanks a lot.

344 00:57:41.160 --> 00:57:52.110
Julian Shun: So there's a question from Lucca. Are there any PROGRAMS THAT GENERATES SUCH code for those other classes of DP problems.

345 00:57:53.760 --> 00:57:54.990
Rezaul Chowdhury: I'm not aware of.

346 00:57:56.340 --> 00:58:06.630
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

347 00:58:07.830 --> 00:58:11.820
Rezaul Chowdhury: For maybe many. I mean, say,

348 00:58:13.200 --> 00:58:24.900
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

349 00:58:26.670 --> 00:58:28.110
Julian Shun: Okay great, thanks.

350 00:58:29.700 --> 00:58:30.750
Luka Govedič: Are there is like a

351 00:58:31.260 --> 00:58:32.460
Luka Govedič: Like a quick follow up.

352 00:58:33.660 --> 00:58:45.870
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.

353 00:58:48.090 --> 00:59:04.170
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

354 00:59:05.400 --> 00:59:14.760
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

355 00:59:15.390 --> 00:59:25.080
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.

356 00:59:30.120 --> 00:59:31.050
Julian Shun: Okay, great.

357 00:59:33.360 --> 00:59:36.030
Julian Shun: Are there any additional questions unresolved?

358 00:59:40.800 --> 00:59:43.320
Julian Shun: Okay, so it looks like there are no more questions now.

359 00:59:43.800 --> 00:59:47.310
Julian Shun: Well thanks again Rezaul for this interesting topic.

360 00:59:51.060 --> 00:59:56.220
Julian Shun: And if you want to see the source code, you can feel free to contact Rezaul.

361 00:59:58.470 --> 00:59:58.860
Rezaul Chowdhury: Yes, I will share the code.

362 01:00:00.180 --> 01:00:03.480
Julian Shun: Great. Well, thanks again. And thanks, everyone, for coming.